

Work Factor as a Security Metric:
A Comparison of Traditional Middleware,
REST, and GraphQL API Active Reconnaissance

By

Laura M. Saxton

Supervisors

Douglas Millward
Dr. Cathryn Peoples

University of Essex
Department of Computer Science
Master of Science, Cyber Security
M.Sc. Dissertation

*To Gavin, for whom
all this was done*

Declaration

I hereby declare that this thesis presented for the degree of Master of Science in Cyber Security has

- i. been written solely by myself.
- ii. resulted solely from my own work except where explicitly stated otherwise.
- iii. not been included, in part or in full, in any other publication, nor has this thesis been submitted for any other degree or professional qualification.

Additionally, I declare that

- i. use of AI technology in this thesis was limited to troubleshooting coding errors and/or deprecated coding libraries during the artefact construction phase.
- ii. all inclusion and utilisation of third-party software is compliant with the licensure of said software from the date last accessed.

Abstract

While APIs are a common target of cyber attack, there is little understanding of how architectural idiosyncrasies between them impact security against malicious actors. This study endeavored to perform such a comparison by applying Saltzer & Schroeder's (1975) *work factor* principle to an active reconnaissance sequence performed by hackers in the wild. The aim of this study was to demonstrate the viability of the *work factor* metric as a security variable by comparing differences in automated active reconnaissance output between a traditional views-rendering middleware, REST API, and GraphQL API. The composite *work factor* metric was calculated using *TOPSIS-AHP* against security variables identified during automated active reconnaissance. Chi-square calculations were conducted to demonstrate the validity of the *TOPSIS-AHP* calculation and the viability of the metric. While it was found that the GraphQL target had the most advantageous *work factor*, it also disclosed the least amount of attack surface information. The inverse was true of the REST API, which had the least advantageous *work factor* yet resulted in complete disclosure of the target's attack surface. These findings suggest that while *work factor* was demonstrated to be viable as a security metric, further research is needed to determine its role in the decision-making practices of malicious actors in the wild.

Keywords: middleware, API, REST, GraphQL, Saltzer & Schroeder, work factor, hacking, active reconnaissance

Acknowledgments

Thank you, firstly, to Doug and Cathryn, whose many hours of encouragement, anecdotes, and advice provided the solid foundation from which I jumped when I embarked on this project. Thank you to the *Csec Club* for their rallying spirit and morale throughout trying episodes of dissertation distress – and for being the best colleagues I have ever encountered while sitting a course. Thank you, finally, to my loving wife, Namwan, who, through pregnancy, birth, and infant, provided nothing less than complete love and support during this educational endeavour. My *beskert*, I would not have been able to do this without you.

Table of Contents

Declaration.....	3
Abstract.....	4
Acknowledgments.....	5
1. Introduction.....	15
2. API Design and Security.....	17
2.1 REST API Design.....	17
2.2 GraphQL Architecture.....	21
2.3 API Security.....	25
2.4. API Security Metrics.....	28
2.4.1 Cryptographic APIs & Saltzer & Schroeder.....	28
2.4.2 Hacking REST & GraphQL.....	29
3. Ethical Considerations.....	35
4. Dissertation Artefacts.....	36
4.1 Web Application Artefacts.....	36
4.2 Active Reconnaissance Artefacts.....	40
5. Methodology.....	43
6. Data & Results.....	46
6.1 Raw Data Results.....	46
6.2 <i>TOPSIS with AHP</i> Results.....	50
6.3 Variable Correlation Results.....	51
7. Analysis & Discussion.....	56
7.1 Results Analysis.....	56
7.2 Study Limitations.....	60

7.3 Knowledge Gained.....	62
8. Conclusion.....	64
9. Cited References.....	67
10. Bibliography.....	72
11. Appendices.....	75
11.1 Appendix I: Views-Rendering Middleware Source Code.....	75
11.1.1 Main Program Code.....	75
11.1.1.1 package.json.....	75
11.1.1.2 server.js:.....	75
11.1.2 Models.....	79
11.1.2.1 order.js:.....	79
11.1.2.2 product.js.....	79
11.1.2.3 user.js.....	80
11.1.3 Controlllers.....	83
11.1.3.1 admin.js:.....	83
11.1.3.2 auth.js:.....	85
11.1.3.3 errors.js:.....	89
11.1.3.4 shop.js:.....	89
11.1.4 Views.....	93
11.1.4.1 404.ejs:.....	93
11.1.4.2 500.ejs:.....	93
11.1.4.3 Admin.....	94
11.1.4.3.1 add-product.ejs:.....	94
11.1.4.3.2 edit-product.ejs:.....	95

11.1.4.3.3 products.ejs:.....	96
11.1.4.4 Auth.....	98
11.1.4.4.1 create-user.ejs:.....	98
11.1.4.4.2 login.ejs:.....	99
11.1.4.5 Includes.....	101
11.1.4.5.1 add-to-cart.ejs:.....	101
11.1.4.5.2 end.ejs:.....	101
11.1.4.5.3 head.ejs:.....	101
11.1.4.5.4 navigation.ejs:.....	101
11.1.4.6 Shop.....	103
11.1.4.6.1 cart.ejs.....	103
11.1.4.6.2 index.ejs:.....	104
11.1.4.6.3 orders.ejs:.....	105
11.1.4.6.4 product-detail.ejs:.....	106
11.1.4.6.5 product-list.ejs:.....	107
11.1.5 Public.....	109
11.1.5.1 CSS.....	109
11.1.5.1.1 form.css:.....	109
11.1.5.1.2 main.css:.....	110
11.1.5.1.3 product.css:.....	111
11.1.5.2 JavaScript.....	112
11.1.5.2.1 admin.js:.....	112
11.1.6 Routes.....	113
11.1.6.1 admin.js.....	113

11.1.6.2	auth.js.....	114
11.1.6.3	shop.js.....	115
11.1.7	Middleware.....	117
11.1.7.1	is-auth.js:.....	117
11.1.8	Util.....	118
11.1.8.1	path.js:.....	118
11.2	Appendix II: REST Backend Source Code.....	119
11.2.1	Main Program Code.....	119
11.2.1.1	package.json.....	119
11.2.1.2	app.js.....	119
11.2.2	Models.....	122
11.2.2.1	post.js.....	122
11.2.2.2	user.js.....	122
11.2.3	Controllers.....	124
11.2.3.1	auth.js.....	124
11.2.3.2	feed.js.....	126
11.2.4	Routes.....	131
11.2.4.1	auth.js.....	131
11.2.4.2	feed.js.....	132
11.2.5	Middleware.....	134
11.2.5.1	is-auth.js.....	134
11.3	Appendix III: GraphQL Backend Source Code.....	135
11.3.1	Main Program Code.....	135
11.3.1.1	package.json:.....	135

11.3.1.2 app.js:.....	136
11.3.2 GraphQL.....	139
11.3.2.1 resolvers.js:.....	139
11.3.2.2 schema.js:.....	145
11.3.3 Models.....	147
11.3.3.1 post.js:.....	147
11.3.3.2 user.js:.....	147
11.3.4 Controllers.....	149
11.3.4.1 auth.js:.....	149
11.3.4.2 feed.js:.....	151
11.3.5 Middleware.....	156
11.3.5.1 auth.js:.....	156
11.4 Appendix IV: Views-Rendering Application Demonstration.....	157
11.5 Appendix V: REST Backend Demonstration.....	160
11.6 Appendix VI: GraphQL Backend Demonstration.....	163
11.7 Appendix VII: GraphQL Postman JSON Queries.....	166
11.8 Appendix VIII: Active Reconnaissance Source Code.....	168
11.8.1 Views-Rending Middleware.....	168
11.8.1.1 Step 1: Baseline Scanning.....	168
11.8.1.2 Step 2: URI Brute-Forcing.....	169
11.8.1.3 Step 3: Content Discovery.....	171
11.8.2 REST API.....	174
11.8.2.1 Step 1: Baseline Scanning.....	174
11.8.2.2 Step 2: URI Brute-Forcing.....	175

11.8.2.3 Step 3: Content Discovery.....	177
11.8.3 GraphQL API.....	181
11.8.3.1 Step 1: Baseline Scanning.....	181
11.8.3.2 Step 2: URI Brute-Forcing.....	182
11.8.3.3 Step 3: Introspection Verification.....	184
11.8.3.4 Step 4: Schema Extraction.....	186
11.9 Appendix IX: Brute-Forcing Word List.....	189
11.10 Appendix X: Active Reconnaissance Full Data Results.....	197
11.11 Appendix XI: Weekly Update Example.....	199
11.12 Appendix XII: Bloom’s Taxonomy Worksheet.....	200

Index of Tables

Table 1: REST API Constraints.....	18
Table 2: OWASP API Security Cheat Sheets.....	26
Table 3: Saltzer & Schroeder Secure Design.....	28
Table 4: Hacker Types and Motivation.....	30
Table 5: OWASP Top 10 API Security Risks.....	33
Table 6: Work Factor Variables for Active Reconnaissance.....	44
Table 7: Averaged Active Reconnaissance Output.....	47
Table 8: TOPSIS with AHP Results.....	51
Table 9: Chi Square – System Information.....	52
Table 10: Chi-Square – Timestamp.....	52
Table 11: Chi-Square – HTTP Actual %.....	53
Table 12: Chi-Square – HTTP Scan %.....	53
Table 13: Chi-Square – Attack Surface.....	53
Table 14: Chi-Square – Scans Performed.....	53

Table of Figures

Figure 1: RESTful Orthogonality.....	17
Figure 2: Sub-Constraints of the REST Uniform Interface.....	18
Figure 3: Richardson Maturity Model.....	19
Figure 4: Palma et al. Research Trend.....	20
Figure 5: GraphQL Type System.....	22
Figure 6: GraphQL Component Interaction.....	23
Figure 7: GraphQL Type System Interaction.....	24
Figure 8: REST and GraphQL Attack Sequences.....	32
Figure 9: REST API Automated Reconnaissance Sequence.....	34
Figure 10: GraphQL API Automated Reconnaissance Sequence.....	34
Figure 11: VRMA – Unauthenticated.....	38
Figure 12: VRMA – Authenticated.....	38
Figure 13: REST API Backend – ‘User A’ Login.....	39
Figure 14: GraphQL API Backend – 'User A' Login.....	39
Figure 15: Automated Reconnaissance Tools.....	41
Figure 16: REST API – Dirsearch CLI Argument Example.....	42
Figure 17: AHP Spreadsheet Calculations.....	45
Figure 18: TOPSIS Spreadsheet Calculations.....	45
Figure 19: VRMA – Nmap Output.....	47
Figure 20: VRMA – Initial Dirsearch Output (HTTP GET).....	48
Figure 21: VRMA – Authenticated Dirsearch Output (HTTP GET).....	48
Figure 22: REST API – Nmap Output.....	48
Figure 23: REST API – Initial Dirsearch Output (HTTP PUT).....	48

Figure 24: REST API – Authenticated Dirsearch Output (HTTP PUT).....	48
Figure 25: GraphQL – Nmap Output (Initial).....	49
Figure 26: GraphQL – Dirsearch (HTTP POST).....	49
Figure 27: GraphQL – Nmap Output (Introspection).....	49
Figure 28: GraphQL – Graphw00f Output.....	49
Figure 29: GraphQL API – Clairvoyance Output.....	49
Figure 30: AHP of Independent Variables.....	50
Figure 31: TOPSIS Calculation Results.....	50
Figure 32: Trend – System Information.....	54
Figure 33: Trend – Timestamp.....	54
Figure 34: Trend – HTTP Actual %.....	54
Figure 35: Trend – HTTP Scan %.....	54
Figure 36: Trend – Attack Surface.....	54
Figure 37: Trend – Scans Performed.....	54
Figure 38: GraphQL IDE – Field Suggestion.....	59

1. Introduction

In 2023, 29% of all cyber attacks on the web targeted API technology (Akamai, 2024). While this percentage has not yet eclipsed the rate of attack found against traditional web applications, APIs are “at the heart of most digital transformations today” (Akamai, 2024: 3) and are therefore poised to become a larger attack vector as their profile on the web continues to grow. Among the most popular architectures used are REST and GraphQL APIs (Bernardino et al., 2021; Buna, 2019), and while several comparative studies concerning response time (Dimitrovski et al., 2020), memory utilisation (Brito et al., 2019), CPU load (Arifin et al., 2023), and throughput (Lawi et al., 2021) have demonstrated differences in quality of service between these API architectures, there do not appear to be similar studies comparing REST and GraphQL against a quantifiable security metric.

Application of a security principle first cited in the 1970s may be able to bridge this gap: Saltzer and Schroeder’s (1975) *work factor* for computer security. This principle has the potential to measure a resource much in demand to the malicious actor: the minimal effort required for exploitation relative to vulnerability disclosure and/or exploit success. While no API architecture is without vulnerability, it may be true that one could be more difficult to attack than another. If an API architecture has a less advantageous *work factor* than another by virtue of its design, this could impact a malicious actor’s desire for attack. And yet, Saltzer & Schroeder (1975) found *work factor* to be somewhat incongruous to the measurement of computer security. As a result, there is yet to be a standardised, demonstrable quantification of this principle for computer security, which presents the opportunity to explore two related research questions:

RQ1: Can *work factor* be quantified as a valid security metric which represents differences in security between API architectures?

RQ2: Between traditional views-rendering middleware, REST, and GraphQL API architectures, which has the most advantageous active reconnaissance *work factor* for malicious actors?

With corresponding hypotheses:

H0: Views-rendering middleware, REST, and GraphQL API architectures produce no discernible difference in *work factor* score.

H1: Views-rendering middleware, REST, and GraphQL API architectures produce a discernible difference in *work factor* score.

To investigate the above, an exploratory study will be conducted against a traditional views-rendering middleware application, a REST API, and a GraphQL API to observe possible differences between the architectures' *work factor* rankings. To do so, the study will demonstrate a quantification of the *work factor* metric as a composite score based upon isolated security variable output from the above artefacts. Measures of the *work factor* results will then be compared to reach possible conclusions concerning, firstly, the validity of the *work factor* quantification as well as its viability as a security metric, and, secondly, any observed differences in *work factor* between the artefacts which may provide insight into their security profiles.

Thus, the remainder of this study is comprised of the following: *Section 2* provides a review of current literature concerning REST and GraphQL APIs at the design rule level, as well as trends in API security research and malicious actor motivation and methodology. *Section 3* presents ethical considerations in accordance with the requirements of computer science research (Bailey et al., 2013). *Section 4* discusses the building of the three middleware artefacts, the attack scripts developed, and third-party tools utilised. *Section 5* presents the methodology of the study, including *work factor* quantification methods and validity/viability measurements. *Section 6* presents the re-

sults of the study followed by *Section 7*, which provides in-depth analysis, study limitations, and knowledge-gained discussions. Finally, *Section 8* concludes the study along with recommendations for future research.

2. API Design and Security

2.1 REST API Design

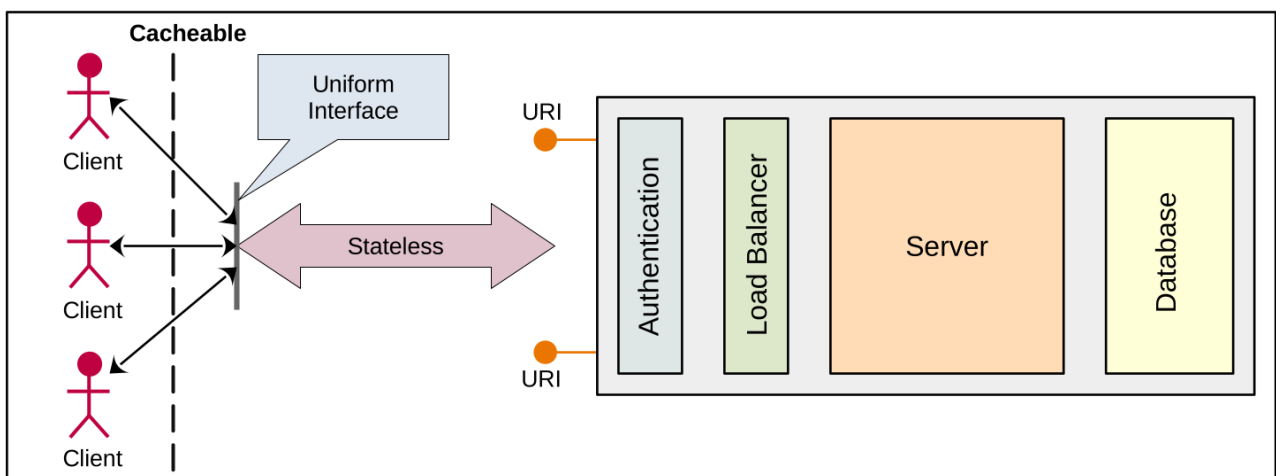


Figure 1: RESTful Orthogonality; adapted from Raj & Subramanian (2019)

The REST API design is essentially “a model of how the web should work” (Bernardino et al., 2021: 958), and is comprised of six high-level design constraints (Fielding, 2000; Table 1) along with “four guiding principles” (Friedrich, 2013: 7) for the uniform interface (Figure 2). A representation of the orthogonal nature of the REST API constraints (Raj & Subramanian, 2019) can be seen in *Figure 1*. These constraints are “aligned with the features of the Hypertext Transfer Protocol (HTTP)” (Bogner & Kotstein, 2021: 155), though themselves do not constitute a standard protocol. Arcuri et al. (2023) have noted this lack of standardisation has led to a wide variety of interpretations for low-level implementation. Various textbooks and guidelines of *best practice* principles exist to bridge the gap between theory and practice, yet these guidelines often differ in length, clarity, specificity, and can provide contradictory advice (Alliyu et al., 2017).

Table 1: REST API Constraints

Constraint	Principle
Client-Server model	Provides a separation of concerns
Stateless architecture	Constrains the client-server interaction
Caching system	Improves network efficiency
Uniform interface	Decouples implementations from services
Layered system	Constrains component behaviour through hierarchical layers that cannot “see beyond the immediate layer with which they are interacting” (Fielding, 2000: 82-83)
Code-on-demand	Simplifies clients “by reducing the number of features required to be preimplemented” (Fielding, 2000: 84)

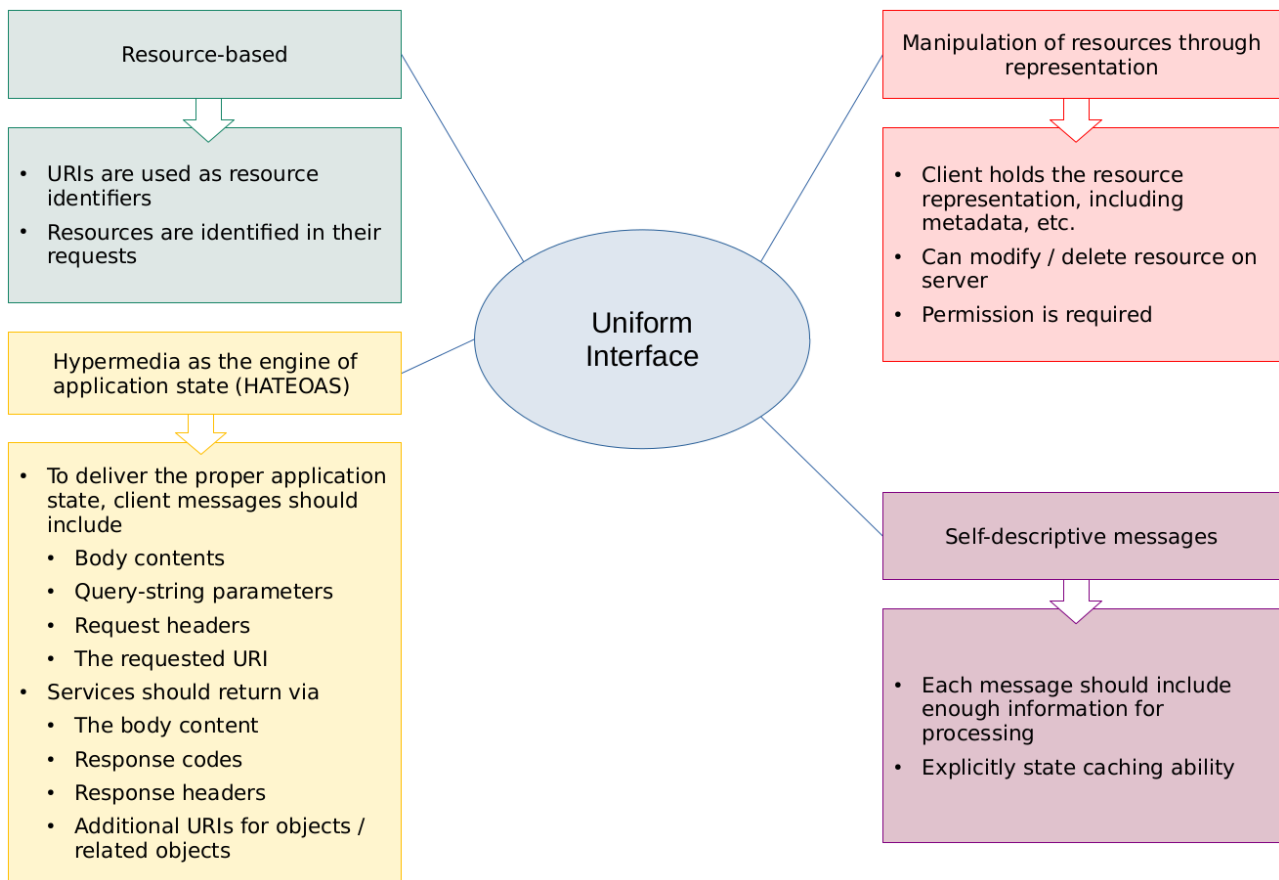


Figure 2: Sub-Constraints of the REST Uniform Interface

As a result, most of the real-world APIs surveyed throughout the above studies tend to hold a maturity level of two according to the Richardson Maturity Model (Fowler, 2010; Meshram, 2021: 3; Figure 3). This level appears to be the industry standard – a conclusion also reached by Bogner & Kotstein (2021). At the same time, Bernardino et al. have noted that “violating any of [REST’s] set of principles puts its objective at risk” (2021: 958), which is to say its ability to mirror the lightweight protocol architecture of the web. What is not clear is how the same violations may affect API security.

Empirical study of REST API design seems to cluster in two areas: (i) quantifying the level of REST design rule adoption within the industry, and (ii) exploring developers’ attitudes and understanding of REST design rules. Interesting trends can be found in both areas. In the former, studies measuring design rule adherence by Guéhéneuc et al. (2016), Baez et al. (2016), and Bernardino et al. (2021) produce varying results. Of the 73 principles tested by Guéhéneuc et al. (2016), 61% were properly implemented; Baez et al. (2016) found an adherence rate of approximately

Richardson Maturity Model

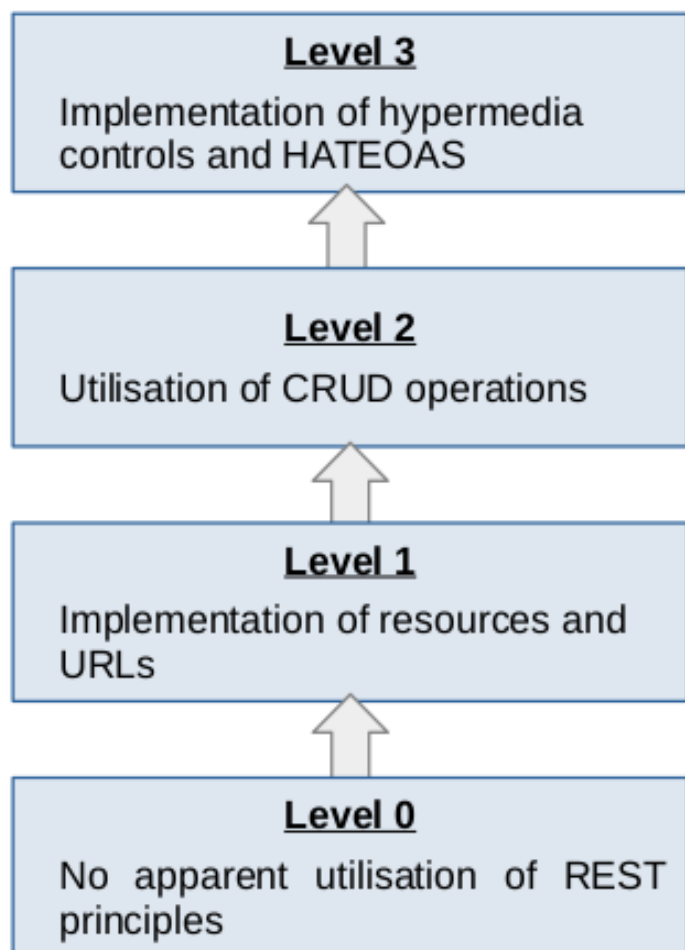


Figure 3: Richardson Maturity Model

88% among 17 heuristics; and finally, of the 26 API features surveyed in Bernardino et al. (2021), the proper implementation rate was 50.56% with very little overlap in design rule choice.

At a more granular level, Palma et al.'s research (2014; 2017; 2021; 2022) concerns the prevalence of linguistic *anti-patterns* in REST API design rules. *Patterns* and *anti-patterns* are conceptualised around low-level implementation, measuring the use of, for example, 'CRUDy URIs' versus 'Verbless URIs', or 'Contextualised Resources' versus 'Contextless Resources'. The trend of the empirical results shows an almost dichotomous decrease in linguistic *anti-pattern* utilisation between 2014 (53.01%) and 2022 (79.16%) (Figure 4), demonstrating an overall trend of API maturation over time. A notable omission throughout the above studies, though, is the discussion of whether an increased adoption of proper RESTful linguistic patterns improves overall API security.

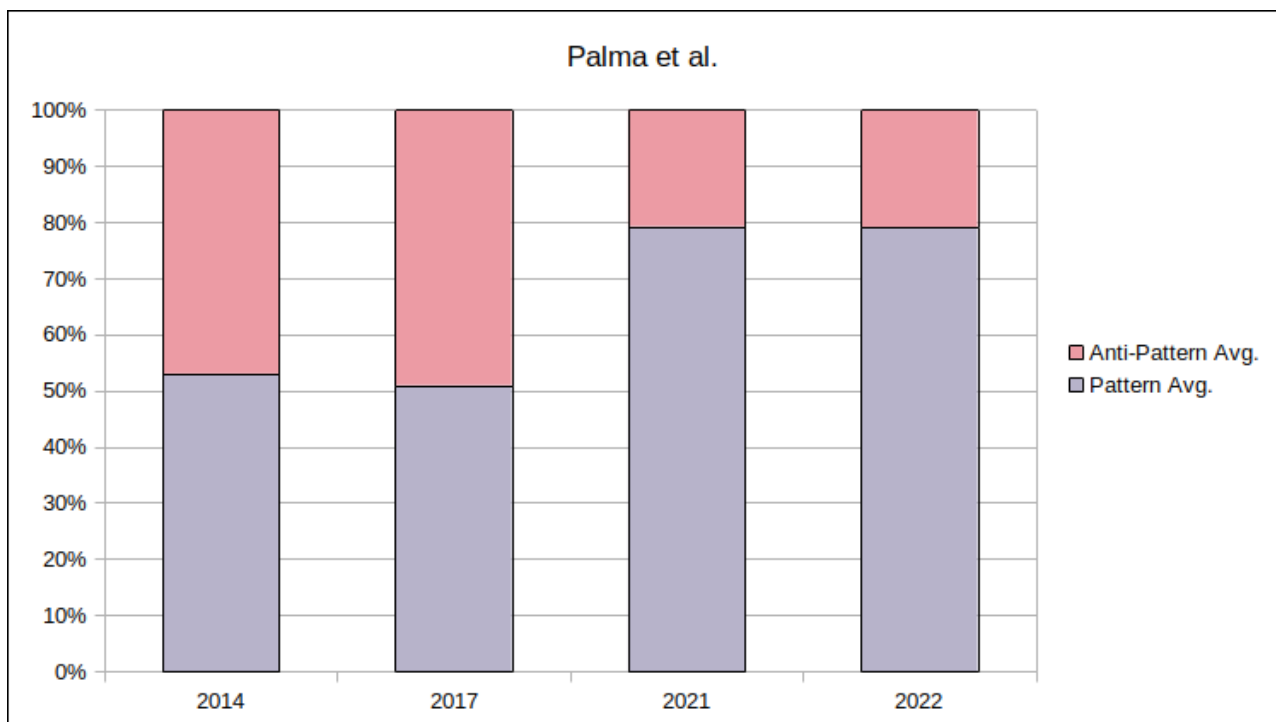


Figure 4: Palma et al. Research Trend

At the same time, empirical analysis of security at the design-rule level may be prohibitively complex. As design rule variation between APIs is not consistent, idiosyncrasies across API architectures have been shown to hinder the generalisability and construct validity of results by a

single study (Baez et al., 2016; Bernardino et al., 2021; Bogner et al., 2022; Guéhéneuc et al., 2016; Palma et al., 2014). Harder still is the extrapolation of individual design rules onto generic security parameters. It is perhaps not unexpected, then, that security metrics are missing from these analyses.

In contrast, when looking at studies detailing API developers' attitudes and understanding of REST API design, security as a topic is not so much absent but rather appears as a non-consideration. An evaluation study conducted against 32 REST API design guidelines (Alliyu et al., 2017) found that, while the topic of security was mentioned in 15 guidelines, security was not considered an "interesting or important [topic] by practicing API designers" (Alliyu et al., 2017: 3). These results are similar to a finding in Bogner and Kotstein's Delphi study (2021) regarding industry professionals' opinions of the importance of REST API design rules. When asked about "perceived rule impact on software quality" (Bogner & Kotstein, 2021: 165), none of the participants considered any rules assigned high-to-medium importance to have an effect on security. Bogner & Kotstein are careful to avoid a generalisation of results due to the small sample size and homogeneous geography of the participants; yet, when considered in tandem with Alliyu et al. (2017), it appears that developers are not deeply engaged with API security conceptualisation at the design level.

2.2 GraphQL Architecture

GraphQL is an API "query language and execution engine" (Fernandez et al. 2023: 202:3) distinct from RESTful architecture. Developed by Facebook in 2012 and later released as an open source standardisation, GraphQL is meant to be "an alternative to solve the [performance] problems found in traditional REST technology" (Fernandez et al., 2023: 202:3). To accomplish this, rather than executing static request protocols over HTTP, GraphQL "encodes in a uniform language the model capabilities of a type system based on five design principles" (Fernandez et al.

2023: 202:3; GraphQL, 2021; Figure 5) dynamically delivered over the HTTP GET and/or POST methods. This type system defines “schematic types for objects [at] the application level that are further used to formulate queries” (Bansal, 2023: 9), resulting in advantageous usability and performance alongside complex query construction and response execution.

GraphQL Type System

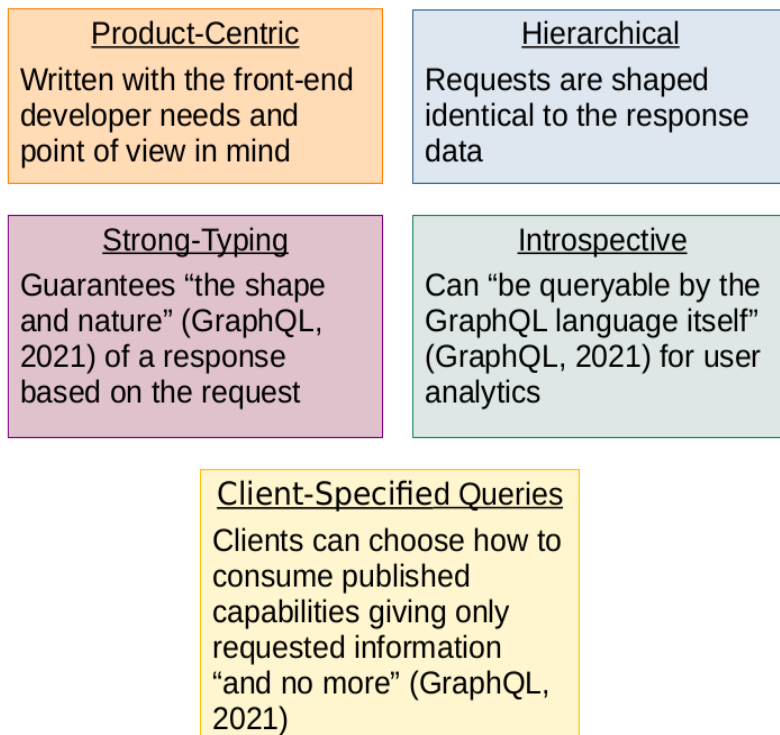


Figure 5: GraphQL Type System

of the type system can produce nested queries and responses that may require multiple cycles to return the desired response. These nested queries are “a traversal of [one node] into neighboring nodes” (Diaz et al., 2020: 207) wherein a list type includes a field of another list type, both of which must be cycled through to identify a desired resource. These queries also include a “worst case response size [that is] exponential in the size of the query” (Baudart et al., 2019: 14), which results in either a denial of service or program crash (Brito et al., 2019).

The GraphQL execution engine involves many components to craft specific schematic queries or changes to data (‘mutations’) on the client-side and return as-needed resources from the server-side (Fernandez et al. 2023; Figures 6 & 7). This complex operational dependency relies on the schema structure and type system for client-side queries and mutations, as well as server-side responses. The hierarchical nature

Much attention has been given to identifying and mitigating these problem queries to avoid such outcomes in the wild. To this end, Hartig and Perez’s (2018) semantic formalisation of GraphQL queries, Diaz et al.’s attempt to expand upon Hartig and Perez by proposing a “mechanized formalisation of GraphQL” (2020: 202), and Baudart et al.’s dynamic Machine Learning “solution that predicts query cost based on experience generated over multiple user-server communication sessions” (2021: 1146) have been successful in quantifying and identifying worst-

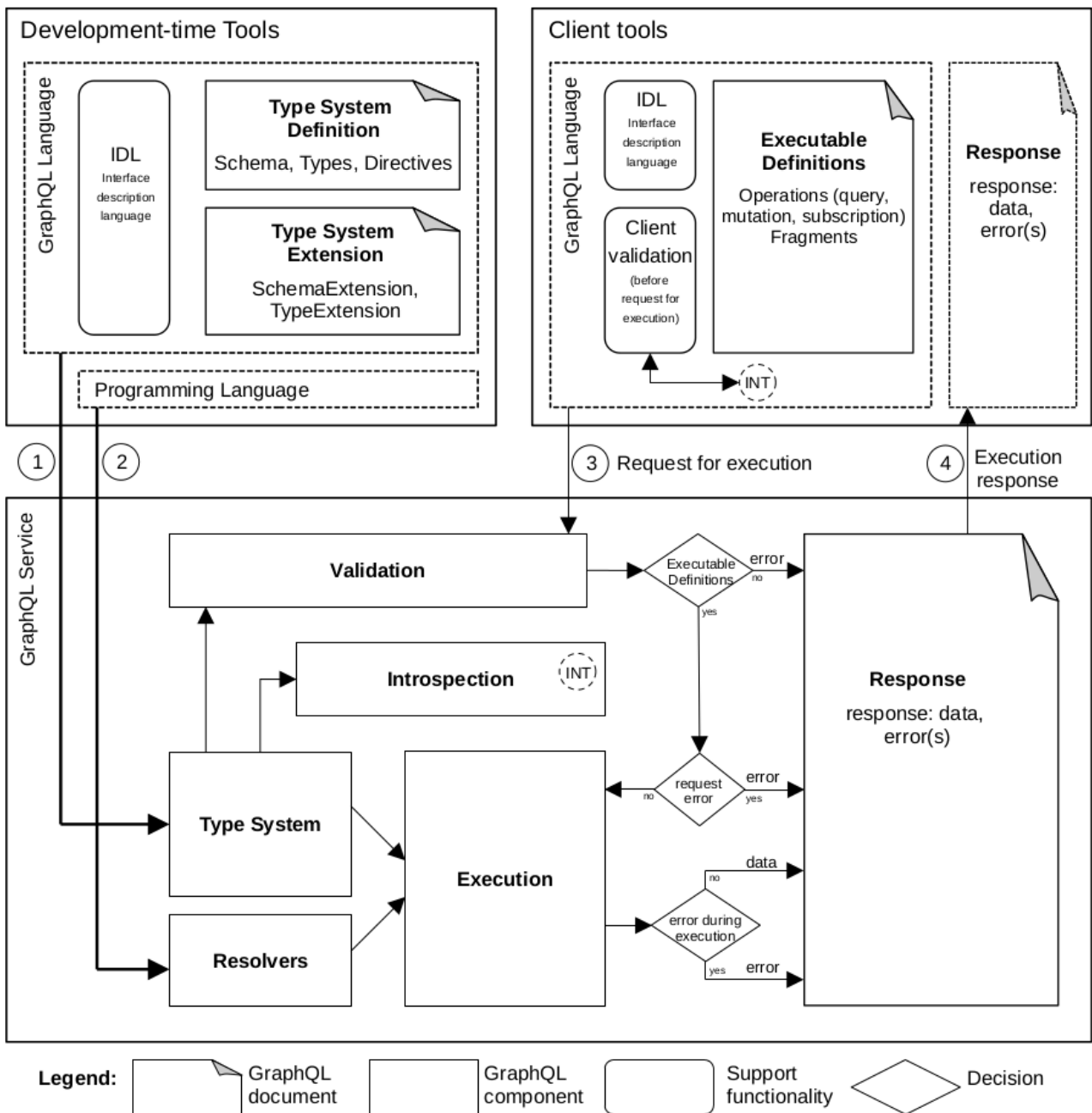


Figure 6: GraphQL Component Interaction; adapted from Fernandez et al. (2023)

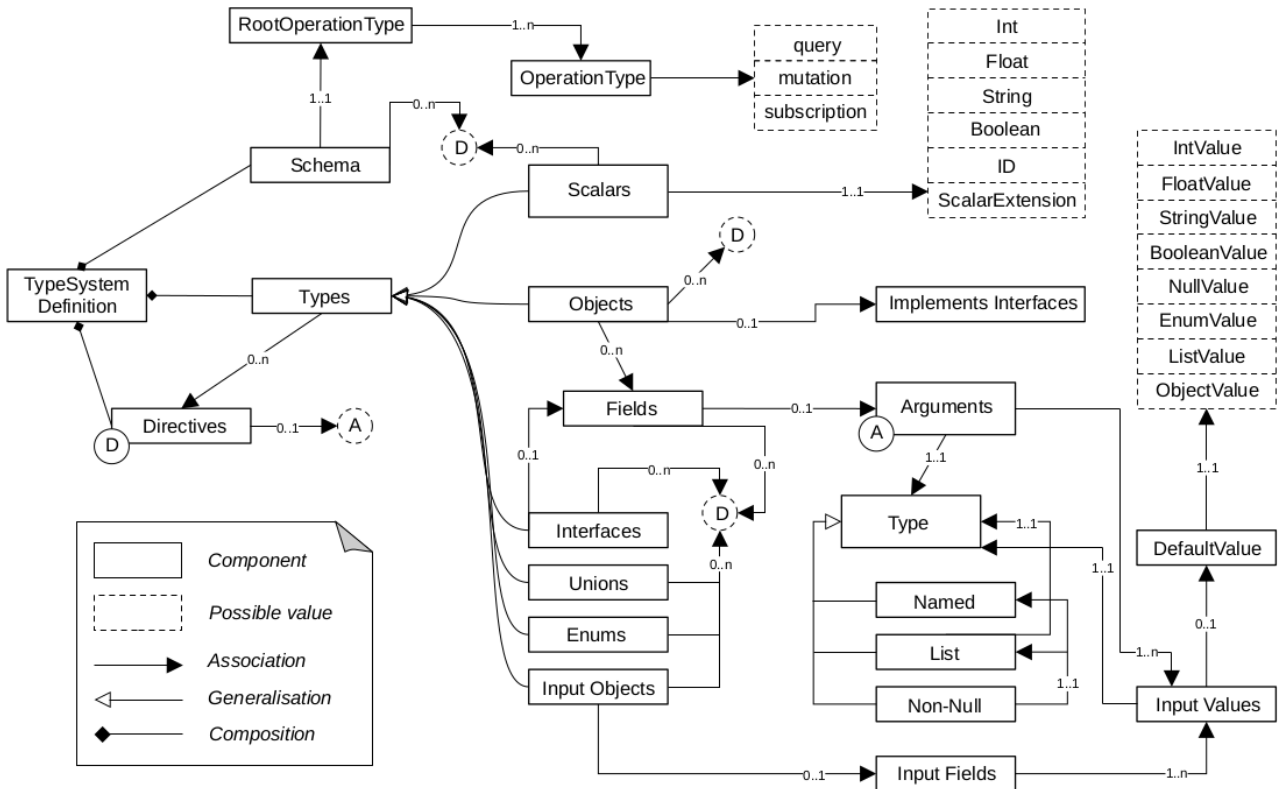


Figure 7: GraphQL Type System Interaction; adapted from Fernandez et al. (2023)

case response sizes through static and automated methods. An interesting inclination within these studies is the language employed by researchers to describe these worst-case response sizes. Researchers favour economic metaphor (e.g., high *cost*, prohibitively *expensive* (Baudart et al., 2019: 14)) to equivalent security vocabulary (e.g., *excessive resource consumption*, *denial of service* (OWASP, 2023)), which could suggest a disconnect between observable trends in GraphQL architectures and the security concerns they imply.

Part of this disconnect could stem from the difficulty of API security quantification, so that researchers avoid conjecture by focussing only on economic qualities that are more reliably measurable. Yet, an absence of security within the GraphQL design itself may also be presumed by researchers, as the security philosophy of the architecture has been described colloquially as “You do security. We give you an API specification [...] then you do security the way you need to.” (Doolittle, 2019: 119). Empirical studies evaluating GraphQL design seem to reflect this sentiment. Basti-

das et al.’s quality in use evaluation found their GraphQL “application’s implementation complied with 84.11% [of the ISO 25040 quality model] which is considered an ‘opportunity’ to improve economic, health, or environmental outcomes” (2022: 25). But the security implications of lower-than-expected *Errors in a Task* and *Trust* measurements are not discussed. Consens et al.’s simple cycle evaluation found that 39.7% of evaluated schema contained “at least one simple cycle” (2019: 4), with the majority of these (68.9%) containing multiple cycles. While the results identify an interesting “distribution [resembling] a power law” (Consens et al., 2019: 4), no security implications of such a large portion of vulnerable cycles are explicitly discussed.

Key advantages and disadvantages of the GraphQL architecture in Brito et al.’s (2019) migration study do isolate important components of GraphQL, such as client-specified queries, introspection, information hiding, and complex caching, but do not discuss possible security holes when migrating to these structures from REST. Interestingly, Baudart et al.’s (2019) empirical study of GraphQL schemas does mention the vulnerability implications of exponential responses and lack of pagination patterns in schemas, but again favours economic metaphor over security-specific language (“avoid the negative consequences of expensive queries” (Baudart et al., 2019: 14)). At the same time, possible implications of the varied use of naming conventions and the security implications regarding “reliance on different GraphQL features (e.g., unions, custom directives, subscription, mutation)” (Baudart et al., 2019: 11) are not discussed. Baudart et al.’s (2021) subsequent exploration of the effect of Machine Learning on GraphQL accuracy, feature selection, and practicality is similarly absent of any security discussion.

2.3 API Security

Though security at the design level of REST and GraphQL APIs is not overwhelmingly present in formal literature, industry whitepaper guidelines on the topic are quite robust. The

OWASP *cheat sheet* series (2024a; 2024b; Table 2) provides a comprehensive checklist of security measures tailored to the idiosyncrasies of REST and GraphQL and their architectural needs. It is important to note that, while input validation, access control, rate limiting/API keys, and error handling are present for both architectures, many of the security recommendations are architecture-

Table 2: OWASP API Security Cheat Sheets; adapted from OWASP (2024a & 2024b)

REST Security Cheat Sheet	GraphQL Security Cheat Sheet
HTTPS	Input Validation
Access Control	General Practices
JWT	Injection Prevention
API Keys	Process Validation
Restrict HTTP Methods	DOS Prevention
Input validation	Query Limiting (Depth & Amount)
Validate Content Types	Timeouts
Validate request content types	Query Cost Analysis
Send safe response content types	Rate Limiting
Management Endpoints	Server-side Batching and Caching
Error handling	System Resource Management
Audit Logs	Access Control
Security Headers	General Data Access
CORS	Query Access (Data Fetching)
Sensitive Information in HTTP Requests	Mutation Access (Data Manipulation)
HTTP Return Code	Batching Attacks
	Mitigating Batching Attacks
	Secure Configurations
	Introspection + GraphQL
	Don't Return Excessive Errors

specific. HTTP/headers security, endpoint management, and content validation are unique to REST, while denial of service (DOS) prevention, batching attacks, and introspection are unique to GraphQL. This is not to suggest that HTTP is not a component of GraphQL security, nor should REST be considered immune to DOS attacks; yet, what is apparent is a different emphasis on the critical aspects of their respective attack surfaces.

Wider API security literature tends to approach vulnerability mitigation at a higher, architecturally agnostic level of analysis. Munsch & Munsch (2021) have noted that API security and traditional web application security share very little overlap, while Qazi found that “APIs [used in cloud computing] are insecure [and] users depend on the overall security of the network instead of standalone API security” (2023: 8). Al-Rumain & Pawar (2023) noted that different API types (e.g., private APIs, partner APIs, and public APIs) have different security risk sensitivities. These findings support similar, non-empirical assertions of API security (Aleks & Farhi, 2023; Ball, 2022; Siriwardena, 2020), and, importantly, highlight the possibility that different APIs may be vulnerable in different ways to different attacks. But overall, as with the architecture-specific literature above, the possibility of low-level, divergent security needs is not explored. APIs are instead grouped together as a security monolith with identical attack profiles and mitigation suggestions, which are left to the API developers to parse and implement.

The question may then be: how can a more individualised set of standards for middleware security metrics be, firstly, isolated, and, secondly, quantified? As the structure of every API is as varied as the intentions of malicious attackers who may seek to compromise them, relying on a single security variable would not be recommended. Yet it may be possible to quantify and correlate different security variables to better understand how different API architectures react to certain vulnerabilities. This may be feasible by extrapolating trends found within cryptographic API security research onto the practices of API hackers, both ethical and malicious.

2.4. API Security Metrics

2.4.1 Cryptographic APIs & Saltzer & Schroeder

To this end, a longitudinal literature review by Dwyer et al. (2023) provides a starting point by proving a link between 47 years of cryptographic API design recommendations and Saltzer and Schroeder’s *Secure Design* paradigm (1975; Table 3). Subsequent API design recommendations by Gutman (2002), Bloch (2006), Green and Smith (2016), and Myers and Stylos (2016) all have a common ancestor in Saltzer and Schroeder, while Acar et al. (2017), Assal & Chiasson (2019), Hallett et al. (2019), Anthonysamy et al. (2020), and Fulton et al. (2020) provide empirical analysis that can be traced back to the principles of *economy of mechanism*, *open design*, *fail-safe defaults*, *separation of privilege*, and *least privilege*.

Of particular relevance among these studies is the methodology employed in the Fulton et

Table 3: Saltzer & Schroeder Secure Design; adapted from Saltzer & Schroeder (1975)

Secure Design Principles
(1) Economy of mechanism
(2) Fail-safe defaults
(3) Complete mediation
(4) Open design
(5) Separation of Privilege
(6) Least privilege
(7) Least common mechanism
(8) Psychological acceptability
(9) Work factor
(10) Compromise recording

al. study to measure the severity “metrics across different vulnerability types and subtypes” (2020: 119). This method (i) reduced attacks to their individual sequence components, (ii) calculated how many steps were required for vulnerability execution, and (iii) quantified the difficulty of completing such an execution, resulting in three separate security variables. While the goal of Fulton et al. was to determine “how the different vulnerability types differ from each other” (2020: 118), a similar method could be extended to quantify possible

security differences among API architectures if applied in a way that takes the security variable results of a target and combines them into a composite security metric.

The need for a composite score that reliably measures differences in API architectural security may find a candidate in Saltzer & Schroeder’s *work factor* principle. *Work factor* measures “[a comparison of] the cost of circumventing the mechanism with the resources of the potential attacker” (Saltzer & Schroeder, 1975: 1283). Considered by Saltzer and Schroeder to “apply only imperfectly to computer systems” (1975: 1283), with the proliferation of automated hacking tools (Farhi, 2024; Nmap, n.d.; Soria, 2022; Stupin, 2023) one could surmise an even larger chasm than when the paper was first published. But if ‘cost’ is understood to be the information uncovered by the hacker for their time alongside the tools they must use to extract it, *work factor* becomes highly applicable to the modern hacker.

Recent studies surrounding hacker motivations have found ease of attack to be a notable influence when selecting a possible target (Cito & Happa, 2023; Hu et al., 2018), and as networks, web technologies, and hacking methodologies have advanced, so too has the difficulty of breaking systems. *Work factor* could serve as a type of composite metric composed of certain security variables related to a demonstrable exploitation that measures the efficacy of an attack on particular architectures or technologies. The composite score could be used to compare these architectures or technologies, or to correlate relationships between *work factor* and other security variables. Adapting security variables into a single *work factor* score which accurately reflects the efficacy of an API exploitation is dependent on the hacker mindset and toolbox, which will now be explored.

2.4.2 Hacking REST & GraphQL

Though “historically, hackers were known as one generic group” (Chng et al., 2022: 2), the hacker mindset is not uni-dimensional. Chng et al. (2022), in their longitudinal literature and topol-

ogy review, found thirteen different hacker types, all with varying methods and motivations (Table 4). In addition to the listed motivations below, Hu et al. also found “enjoyment [and] maximiz[ing] expected value” (2018: 384) to be important factors for target selection among surveyed bug bounty hunters. The typical hacker is usually looking to make the biggest impact in the shortest amount of time, with only state actors engaging in drawn-out, multi-stage attack sequences (Chowdhury et al., 2022).

Table 4: Hacker Types and Motivation; adapted from Chng et al. (2022)

Types	Definition	Strategies	Motivation
Novices	Low-skilled, mostly tool-dependent	Reuse the code/tools/scripts of others. No sophistication or anonymity	Curiosity, notoriety, recreation
Cyberpunks	Low-to-medium skill; cause chaos for fun	Can modify code; basic attack vectors/DOS	Financial, notoriety, revenge, recreation
Insiders	Disgruntled current or ex-employees	Sell company info / launch an insider attack	Financial, revenge, ideology
Old Guards	Non-malicious, but have “no respect for personal privacy” (Chng et al., 2022: 4)	Use scripts and tools to expose vulnerabilities in existing systems. White hats / grey hats	Curiosity, notoriety, recreation, ideology
Professionals	Extremely high-skilled, gun-for-hire / criminal activity	Full attack vectors; highly sophisticated	Financial, revenge
Hactivists	Further political agenda / fuel political change	Use injections, web server misconfiguration, etc.	Notoriety, revenge, recreation, ideology
Nation States	Highly trained, cyber warfare	Conduct multi-level attacks	Financial, revenge, ideology

Students	Non-malicious, hack “to gain knowledge” (Chng et al, 2022: 4)	May use code like novices with modifications to test systems. Likely to report vulnerabilities	Curiosity
Petty Thieves	Low-to-medium skill, scammers	Use Trojans, ransomware, etc.	Financial, revenge
Digital Pirates	Distribution of copyrighted material	Steal copyrighted content	Financial
Online Sex Offenders	Cyber predator	Target social media and the dark web	Sexual impulses
Crowdsourcers	Group hacking, usually to solve a problem through dubious means	Pool skill sets for malware, botnets, etc.	Notoriety, revenge, recreation, ideology
Crime Facilitators	Tool / technical know-how sharing	Offer Cyber-criminal services; phishing, malware, etc.	Financial

The concerns of the malicious actor could be framed as *work factor* considerations, as those engaging in illegal activity would have a desire to leave as small an attack footprint with the largest payout in the shortest amount of time to avoid detection and possible legal action. Additionally, a minority of those who engage in hacking activities are considered highly skilled, with the rest sy- phoning attack vectors from open-source/released code and tools. As a result, most attack se- quences involve *known* attack vectors rather than *zero-day* (Chowdhury et al., 2022) or *business logic* (Ball, 2022) attacks, which are novel to the application and take considerable skill to exploit. In other words, very few malicious actors are skilled enough to complete high-stakes, long term cyber attacks. Most are looking for the quickest, easiest form of attack to accomplish their goals.

API hacking itself has *known* API attack sequences (Figure 8), developed for REST by Corey J. Ball (2022) and for GraphQL by Nick Aleks and Dolev Farhi (2023); both sequences can uncover and exploit vulnerabilities found in the OWASP API Top Ten API Security Risks (Table 5)

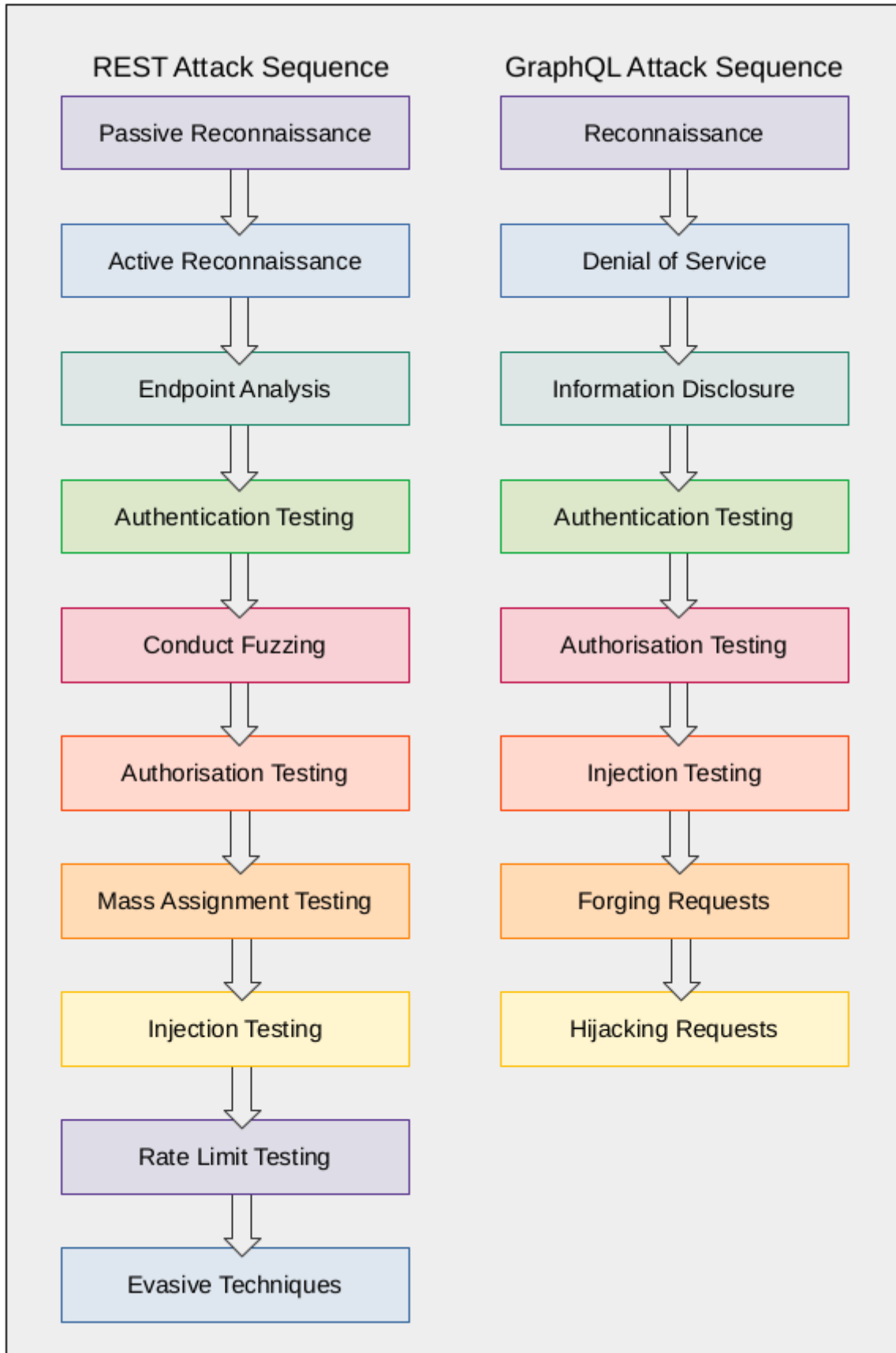


Figure 8: REST and GraphQL Attack Sequences; adapted from Ball (2022) & Aleks & Farhi (2023)

Though any of the OWASP security risks and corresponding API attack sequences could qualify for *work factor* security quantification, active reconnaissance (AR) may be the most pertinent sequence for an initial demonstration of the *work factor* security metric due to the following: first, active reconnaissance is a required preliminary activity before any malicious action, no matter the middleware architecture (Ball, 2022). At the same time, it can be a time-consuming process (Paxton-Fear, 2021), which may influence the overall effectiveness of AR on a particular API architecture. Critical system and target information (Ball, 2022) is often disclosed, which can offer convenient security variables to calculate a composite score. And finally, key aspects can be automated with brute-force scanning tools (Aleks & Farhi, 2023; Ball, 2022) which simplifies timestamping the AR sequence.

Automated testing aims to discover the baseline and content information of a target API (Figures 9 & 10), which can then be analysed more thoroughly with manual testing. Automation aims to minimise the effort re-

quired to locate or brute-force certain targets, which is the exact type of data a *work factor* metric could be used to measure. While differences in REST and GraphQL AR do exist, these variances can be calculated into the composite metric, which would normalise discrepancies between architectural measurements and prevent potential false equivalencies that could be present between

Table 5: OWASP Top 10 API Security Risks; adapted from OWASP (2023)

Risk
Broken Object Level Authorisation
Broken Authentication
Broken Object Property Level Authorisation
Unrestricted Resource Consumption
Broken Function Level Authorisation
Unrestricted Access to Sensitive Business Flows
Server Side Request Forgery
Security Misconfiguration
Improper Inventory Management
Unsafe Consumption of APIs

other measures of the same data.

Quantifying the raw data into a comprehensive *work factor* metric would involve the calculation of a composite score into a single measurement. *TOPSIS* with *AHP* (*TOPSIS-AHP*) has been demonstrated in research as a highly reliable composite metric calculation (Balioti et al., 2018; Borroel et al., 2022; Cokrowibowo et al., 2020). *AHP* stands for ‘Analytic Hierarchy Process’ and performs a normalisation of category weights to be used in the *TOPSIS* calculation for representative weighting of a positive or negative factor. *TOPSIS* is essentially a composite calculation comprised of several steps to assess multiple independent variables of a single dependent variable, which results in a normalised composite score ranked against other dependent variables. The final *TOPSIS* metric can be interpreted as a ranking of each dependent variable’s *distance from the ideal*, with the ideal being an averaged score of 100%.



Figure 9: REST API Automated Reconnaissance Sequence

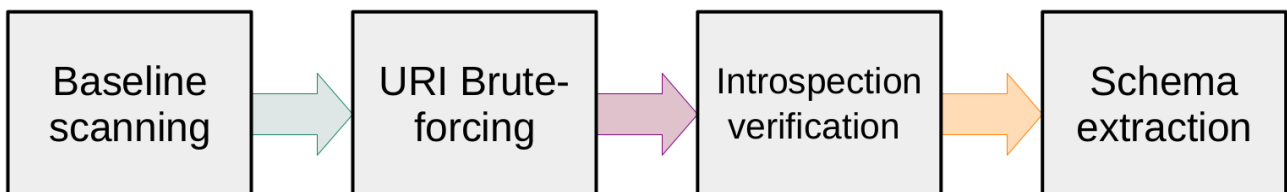


Figure 10: GraphQL API Automated Reconnaissance Sequence

While *TOPSIS-AHP* is most often used within commercial industry to rank certain risk factors essential for operation, the equation could be extended to quantify *work factor*. The metric herein has been conceptualised as essentially a ranked risk score (the risk being wasted time, i.e.,

work factor) that requires the quantification of multiple, sometimes divergent variables with disparately assigned weights. It is important to note, though, that the *TOPSIS-AHP* calculation is subjective insomuch that the weighted values, though normalised through *AHP*, are determined by the author's opinion, however informed by research. Variable comparisons should be carried out alongside *TOPSIS-AHP* to determine the degree of correlation for metric validity, for example in the form of chi-squares (Downey, 2014).

It should additionally be noted that the composite variable of the AR *work factor* would not result in a definitive security metric but rather provide a *work factor*-based metric against which other metrics could be measured, either comparatively or correlatively (Downey, 2014) depending on the focus of the research. That said, within this study it may nevertheless be possible to determine which design aspects of a given API architecture impact the work flow of AR and, therefore, to present conclusions regarding the security of an API architecture from a *work factor* perspective.

3. Ethical Considerations

While this dissertation does not involve human participants, from an ethical standpoint “publication and wide dissemination of vulnerability research should [include the consideration of] its benefit to malicious actors” (Bailey et al., 2013: 9). In this regard, the dissertation certainly has the ethical risk of disclosure, whereby the publishing of a legitimate active reconnaissance hacking sequence, and the results of deployment success or failure against targeted API architectures, could be used as an effective blueprint for hacking with minimal effort by a malicious actor.

To prevent such a scenario, this study confines the active reconnaissance sequence to automated aspects only, as the automated process is limited to exploratory information revelations that do not detail how to spot vulnerabilities as such but rather showcases where certain system information types can be found. Subsequent investigation and possible exploitation of any vulnerable

aspects of a target are not extrapolated upon. This strategy intends to keep the study replicable, albeit with obfuscation as a deterrent to low-skilled actors without the means or knowledge to further exploit vulnerabilities. This is done with the intention to satisfy the ethical requirement of avoiding excessive, unnecessary disclosure in vulnerability research.

Additionally, this project involves the use of third-party programming tools within the active reconnaissance sequence. All included tools have been checked for applicable licensure to verify permission granted for use in research studies. Any tools that did not provide proper permissions have been excluded. Likewise, all instances of tool use mentioned in the dissertation proper shall include citation and acknowledgement of authorship/ownership as required.

4. Dissertation Artefacts

To successfully test the project hypothesis two artefact types were required: web application code for testing and AR scripts for scanning. While original web application source code could be written in full, third-party active reconnaissance tools were utilised for AR as these are widely used by bug bounty hunters and malicious actors in the wild (Aleks & Farhi, 2023; Ball, 2022; Cito & Happe, 2023). This was done, firstly, to streamline timestamp and variable output, and secondly, to align as much as possible with real-world application of the automated active reconnaissance process. The characteristics and design methodology for both artefacts are expanded upon below.

4.1 Web Application Artefacts

The web application artefacts comprised of a views-rendering middleware application (VRMA), a REST API backend, and a GraphQL backend (Appendices I - III). The code and design of the artefacts were written in JavaScript on a Node.js backend with a MongoDB database system

based on the content of Maximillian Schwartzmüller's course *Node.js - The Complete Guide* (2019). ChatGPT (OpenAI, 2023) was used for troubleshooting deprecated or broken code examples when present; otherwise, ChatGPT was not consulted as an independent coding source. Authentication, authorisation, and validation measures were implemented alongside relevant HTTP headers and HTTP method calls. The Schwartzmüller (2019) course content was utilised due to the scope of the modules, which ultimately produced three separate, industry-relevant web application artefacts with different middleware strategies: VRMA, REST, and GraphQL, while at the same time being very similar in design. Application similarity across the artefacts was sought to minimise the chance of spurious variables impacting the validity of the *work factor* measurement.

The VRMA (Figures 11 & 12) was designed to emulate a simple book shop in the classic MVC style, equipped with 17 endpoints and middleware tied directly to HTML views files. Routes were executed along HTTP GET and POST methods, with application logic for *delete* executing both along the DELETE HTTP method call and the POST HTTP method call. The presence of differing *delete* method calls was meant to showcase the multiple routing options possible in MVC applications in the course instruction and was kept for authenticity. The authentication mechanism of this application employed *session cookies* to verify authorisation, with user validation implemented within the application models. Testing comprised of console-logged success and error messages, which were then investigated and mitigated as needed. User functionality was enforced through authentication and authorisation, with user-specific access controls in place (Appendix IV).

The REST and GraphQL API backends did not include JavaScript frontends, as such additions were determined to be out of scope for this study. Both REST and GraphQL backends, equipped with seven and nine endpoints, respectively, emulate a message feed system wherein ind-

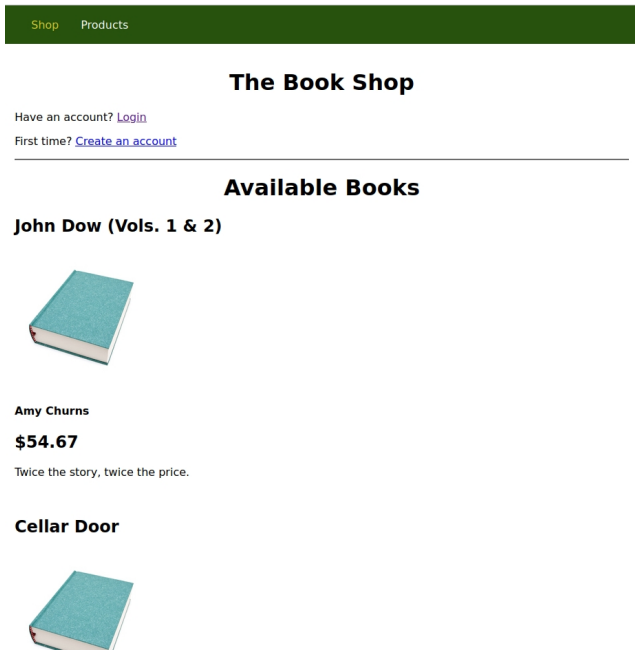


Figure 11: VRMA – Unauthenticated

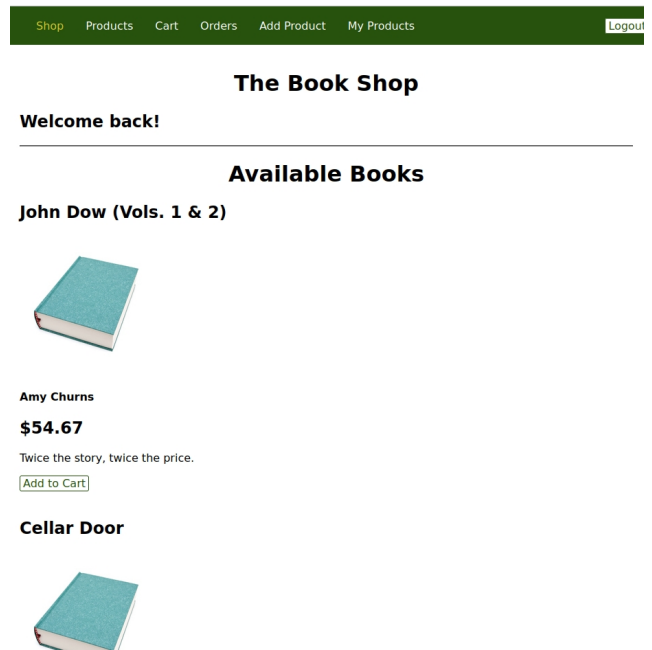


Figure 12: VRMA – Authenticated

individual users can create, edit, and delete a message post and see the posts of other users. For the REST backend (Figure 13), HTTP GET, POST, PUT, and DELETE method calls were utilised among the endpoints, avoiding CRUDy-URL language more common in traditional application middleware (Palma et al., 2022). The GraphQL backend (Figure 14) utilised both HTTP GET and POST methods for a single API endpoint: `http://localhost:5000/api/graphql`. Both APIs were written following MVC conventions, albeit with some architectural modifications, including the absence of views. Postman (Postman, 2024) was used to visualise data without a frontend, and to test endpoint and application logic errors.

The authentication mechanism of both APIs utilised a JWT token generated upon login. The token was included in the response to the HTTP request by course instruction, and so was left intact during subsequent API testing and scanning. Testing the APIs included HTTP requests for the REST API, which returned failed authentication messages or failed authorisation messages when applicable (Appendix V). GraphQL endpoints lay in the schema and resolver functions, which were each tested with a JSON query or mutation generated by ChatGPT (OpenAI, 2024; see Ap-

pendix VII), similarly resulting in failed authentication messages or failed authorisation messages when applicable (Appendix VI).

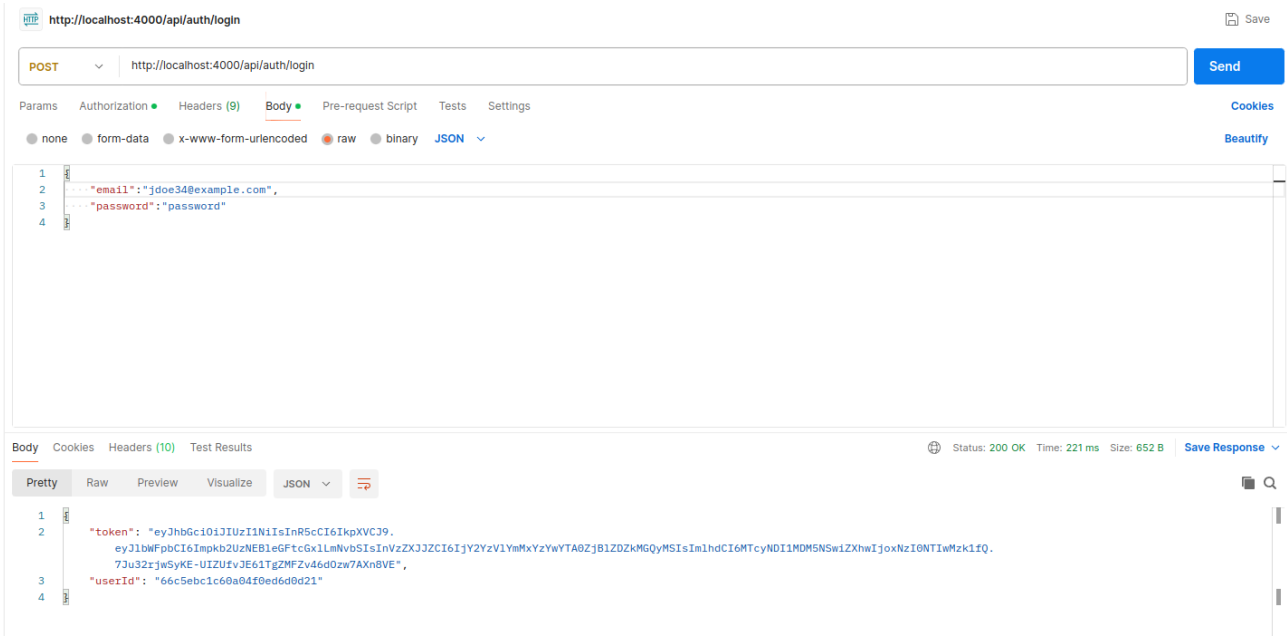


Figure 13: REST API Backend – ‘User A’ Login

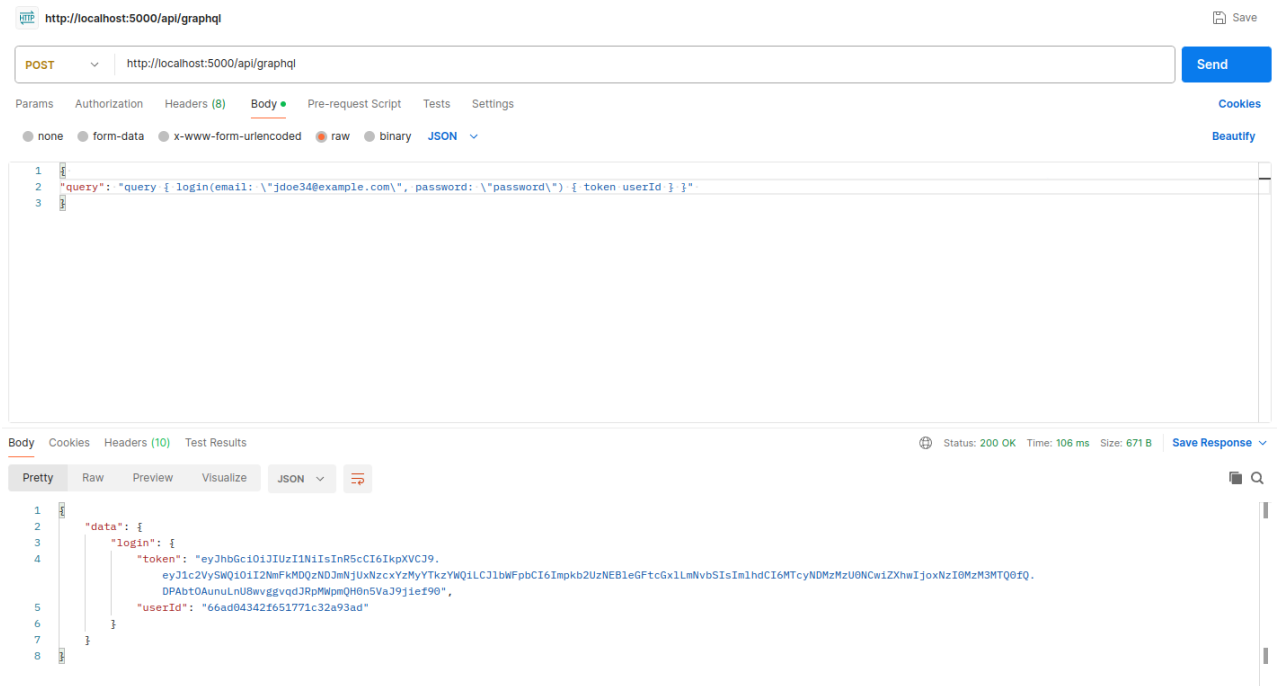


Figure 14: GraphQL API Backend – ‘User A’ Login

4.2 Active Reconnaissance Artefacts

As noted above, the active reconnaissance process involves gathering baseline intelligence leading to API content discovery, traditionally in the form of endpoint discovery (Ball, 2022). Because this study sought to align with the ethical considerations discussed above, exploration was limited to automated scanning reconnaissance activities. Automated reconnaissance involves divergent sequences for VRMAs/REST APIs and GraphQL APIs (Figure 15; see Appendix VIII). The tools chosen for scanning were based on the following criteria:

1. popularity and relevance within the hacking community
2. ability to be executed on a CLI terminal and integrated into BASH script
3. range of customisable commands for scanning

Nmap (Nmap, n.d.), a network scanning tool, was chosen for baseline scanning and GraphQL introspection verification, as it can disclose helpful preliminary information. *Dirsearch* (Soria, 2022), an HTTP/URI brute-forcing tool, was chosen for the initial scan of URI endpoints for all three artefacts, as this tool has the ability to scan different HTTP method calls when clarified in the command line argument. *Graphw00f* (Farhi, 2024), a GraphQL reconnaissance tool, was used in addition to *Nmap* introspection scanning as a server fingerprinting tool during introspection verification. As GraphQL introspection was not implemented in the GraphQL artefact code, *Clairvoyance* (Stupin, 2023) was chosen to reverse engineer the GraphQL by brute-forcing the schema, a common tactic when introspection is disabled or missing (Aleks & Farhi, 2023).

The word list utilised for the brute-forcing tasks (Appendix IX) was adapted from the *2m-subdomains.txt* word list distributed by the popular word list provider *assetnote.io* (Assetnote, 2024). The first 1000 words of the list were included, along with VRMA-, REST-, and GraphQL-

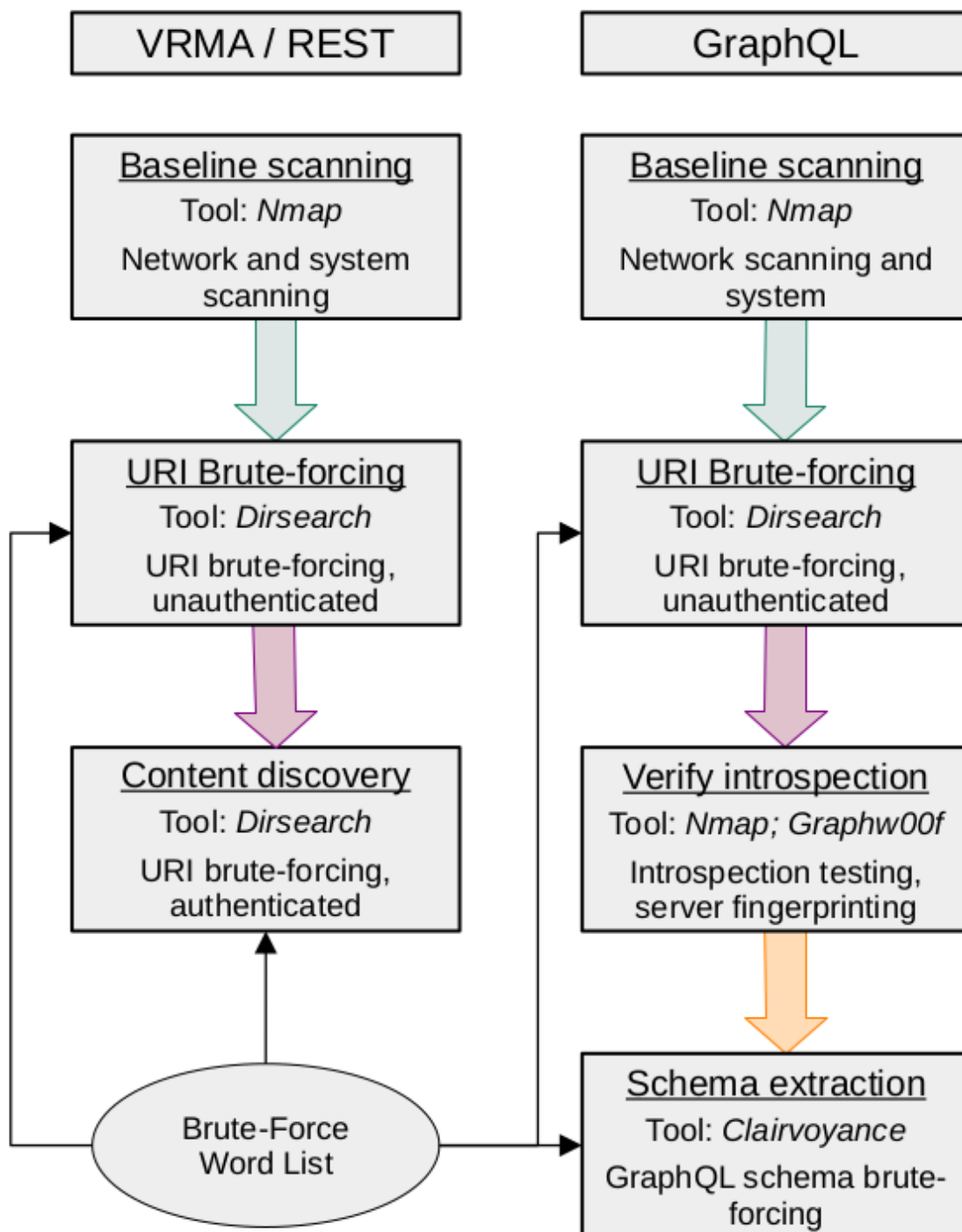


Figure 15: Automated Reconnaissance Tools

specific vocabulary, to allow for complete attack-surface disclosure during reconnaissance testing.

Though it is unlikely for such an outcome to occur in the wild, because a goal of this study is to

demonstrate *work factor* as a viable security metric, the possibility for full attack-surface disclosure would result in the most accurate metric demonstration in relation to the artefacts tested.

Command line arguments for each tool with each artefact were first cultivated for optimal results, including subdirectory or suffix commands when necessary (Figure 16). The addition of subdirectories or suffixes to CLI arguments were chosen based on common VRMA and API endpoint syntax (Ball, 2022; Aleks & Farhi, 2023) along with artefact idiosyncrasies. To provide statistically significant averages across the security variables, multiple instances of the active reconnaissance sequence were implemented. BASH code was used to automate each scan sequence to perform 50 iterations with generated output and timestamps. ChatGPT (OpenAI, 2024) was utilised for troubleshooting deprecated/broken code attempts. Finally, required files and directories were created to store the scan sequence output for subsequent data analysis and manipulation.

Because the different middleware artefacts required different AR foci, BASH code differences exist between the artefacts. The REST and GraphQL API BASH code for *URI Brute-forcing*

```
{
  echo "POST: $(date)"
} >> "$localhost4000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:4000
--auth-type=bearer --auth=jwt_token --delay=1 -m POST
--suffixes /,/{id} --subdirs /,api/,admin/,auth/,api/admin/,api/
auth/ -w $HOME/api/wordlists/dummylist.txt -o
$localhost4000_output/POSTscan-$scan_number.txt

gql_output=$(grep -oP "graphql" "$localhost4000_output/
POSTscan-$scan_number.txt")
if [ -n "$gql_output" ]; then
  echo "GraphQL endpoint detected -- SCAN STOPPED!"
  {
    echo "Finish: $(date)"
    echo "-"
  } >> "$localhost4000_log"
  ((var++))
  ((scan_number++))
  continue
fi
```

Figure 16: REST API – Dirsearch CLI Argument Example

and *Content Discovery* (REST API scans only) was equipped with a ‘GraphQL tripwire’ (Figure 16), whereby if the */graphql* endpoint were appended to the output document a parser would add this result to an array and the program would terminate upon completion of HTTP GET and POST scans.

This was done to best mirror real-world scanning, wherein were a hacker to find a */graphql*

endpoint, the scan would normally be terminated.

Additionally, as VRMAs use limited HTTP call methods, *URI Brute-forcing* and *Content Discovery* sequences utilised only the main four HTTP calls: GET, POST, PUT, and DELETE while REST scans used five, adding the PATCH call. This was done to reflect the information disclosed during *Baseline Scanning*, including the software version an application runs (e.g., *Express Middleware*) which impacts the proceeding *URI Brute-forcing* and *Content Discovery* scan arguments. The VRMA artefact required 3 subdirectory variables (`/auth`, `/admin`, `/user`) for the initial *Dirsearch* (Soria, 2022) scan with four HTTP method calls, totalling an iteration of 12. The same scan with two additional suffix variables (`/` and `{id}`) was performed for the authenticated *Dirsearch* (Soria, 2022) scan, totalling the same iteration with a doubled word list to account for both suffixes.

The API-specific code used against both REST and GraphQL APIs required 5 subdirectory variables (`/api`, `/auth`, `/admin`, `/api/auth`, `/api/admin`) and five HTTP method calls for their initial *Dirsearch* (Soria, 2022) scan, totalling an iteration of 25. The same scan with two additional suffix variables (`/` and `{id}`) was similarly performed against the REST API for the authenticated *Dirsearch* (Soria, 2022) scan, totalling the same iteration with a doubled word list to account for both suffixes. The GraphQL sequence diverged into *Introspection Verification* with *Nmap* (Nmap, n.d.) and *GraphWoof* (Farhi, 2024) scans, as well as *Schema Extraction* with the *Clairvoyance* (Stupin, 2023) tool. None of the resulting scans contained additional suffixes or sub-directories to effect iteration.

5. Methodology

As discussed above, *work factor* as a security metric requires the analysis of multiple inde-

pendent security variables interacting with a dependent artefact variable. As automated AR outputs data which can be used as various independent variables, scan code was designed to return 50 iterations of raw data to be averaged and utilised in the *work factor* calculation. Each scan produced different elements of information disclosure which were classified into the following categories: *System Information*, *Timestamp*, *HTTP Calls*, and *Scans Performed* (Table 6). Two measurements of *HTTP Calls* were identified in the test data: *HTTP Actual %*, which measured the positive ratio of discovered HTTP calls to all HTTP calls present in the application or API, and *HTTP Scan %*, which measured the positive ratio of discovered HTTP calls to all HTTP calls scanned during data collection. Both were included as HTTP variables to best represent disclosed information during AR.

Table 6: Work Factor Variables for Active Reconnaissance

	System Information	Time Factor	HTTP Calls	Attack Surface	Scans
Scope	Server, port, URI, plugin, and application-specific information	Timestamp generation during scanning activities – total scan time	Required calls for scanning, and discovered calls in applications	Uncovered endpoints and schema extraction, depending on the application	Automated scanning tools for network and application reconnaissance
Positive Work Factor	Maximum amount of information divulged	Minimum amount of time elapsed from start to finish of scanning activities	Maximum ratio of discovered calls to required calls	Maximum ratio of existing endpoints/ schema and discovered endpoints/ schema	Minimum amount of scans required to uncover desired information

TOPSIS-AHP was used to compute two *work factor* ideals: the positive, or advantageous, *work factor* (+Work Factor) which represents the ideal best use of time for a malicious actor, and the negative, or non-advantageous, *work factor* (-Work Factor) which represents the ideal worst use of time for a malicious actor. *TOPSIS* weights were decided based on perceived importance of

the data disclosed using the *AHP Approximate Eigenvector Method* (Leal, 2020; SpiceLogic, 2022), which was performed in LibreOffice by spreadsheet calculation (Figure 17). This calculation ranked independent variables by factors of 1, 3, 5, 7, and 9 in terms of absolute importance,

RATING	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.
SYS INFO	1.0000	=1/3	2.0000	1.0000	=1/3	1.0000
TIMESTAMP	3.0000	1.0000	=5	3.0000	1.0000	3.0000
HTTP ACT. %	=1/2	=1/5	1.0000	=1/2	=1/5	=1/2
HTTP SC. %	1.0000	=1/3	2.0000	1.0000	=1/3	1.0000
ATT. SURF.	3.0000	1.0000	5.0000	3.0000	1.0000	3.0000
SCANS PER.	1.0000	=1/3	=2	1.0000	=1/3	1.0000
AVERAGE	=SUM(AM13:AM18)	=SUM(AN13:AN18)	=SUM(AO13:AO18)	=SUM(AP13:AP18)	=SUM(AQ13:AQ18)	=SUM(AR13:AR18)

NORMALISE	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.	AVERAGE
SYS INFO	=AM13/AM\$19	=AN13/AN\$19	=AO13/AO\$19	=AP13/AP\$19	=AQ13/AQ\$19	=AR13/AR\$19	=AVERAGE(AM22:AR22)
TIMESTAMP	=AM14/AM\$19	=AN14/AN\$19	=AO14/AO\$19	=AP14/AP\$19	=AQ14/AQ\$19	=AR14/AR\$19	=AVERAGE(AM23:AR23)
HTTP ACT. %	=AM15/AM\$19	=AN15/AN\$19	=AO15/AO\$19	=AP15/AP\$19	=AQ15/AQ\$19	=AR15/AR\$19	=AVERAGE(AM24:AR24)
HTTP SC. %	=AM16/AM\$19	=AN16/AN\$19	=AO16/AO\$19	=AP16/AP\$19	=AQ16/AQ\$19	=AR16/AR\$19	=AVERAGE(AM25:AR25)
ATT. SURF.	=AM17/AM\$19	=AN17/AN\$19	=AO17/AO\$19	=AP17/AP\$19	=AQ17/AQ\$19	=AR17/AR\$19	=AVERAGE(AM26:AR26)
SCANS PER.	=AM18/AM\$19	=AN18/AN\$19	=AO18/AO\$19	=AP18/AP\$19	=AQ18/AQ\$19	=AR18/AR\$19	=AVERAGE(AM27:AR27)
TOTAL	=SUM(AM22:AM27)	=SUM(AN22:AN27)	=SUM(AO22:AO27)	=SUM(AP22:AP27)	=SUM(AQ22:AQ27)	=SUM(AR22:AR27)	=SUM(AS22:AS27)

Figure 17: AHP Spreadsheet Calculations

I. Convert Timestamps

stamp	hour	minute	second	ms	Total Min.
00:36:16.87	0	36	16	87	36.1687
01:14:18.54	1	14	18	54	74.1854
00:11:12.99	0	11	12	9	11.1299

II. Construct Table

(Weight)	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.
TARGET	=AS22	=AS23	=AS24	=AS25	=AS26	=AS27
Views Middleware	6	=AQ34	0.50	0.25	0.47	3
REST API	7	=AQ35	1.00	0.80	1.00	3
GraphQL API	10	=AQ36	1.00	1.00	0.00	5

1. Calculate the Normalised Matrix

TARGET	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.
Views Middleware	=AM41/((AM41*2)+(AM42*2)+(AM43*2))	=AN41/((AN41*2)+(AN42*2)+(AN43*2))	=AO41/((AO41*2)+(AO42*2)+(AO43*2))	=AP41/((AP41*2)+(AP42*2)+(AP43*2))	=AQ41/((AQ41*2)+(AQ42*2)+(AQ43*2))	=AR41/((AR41*2)+(AR42*2)+(AR43*2))
REST API	=AM42/((AM41*2)+(AM42*2)+(AM43*2))	=AN42/((AN41*2)+(AN42*2)+(AN43*2))	=AO42/((AO41*2)+(AO42*2)+(AO43*2))	=AP42/((AP41*2)+(AP42*2)+(AP43*2))	=AQ42/((AQ41*2)+(AQ42*2)+(AQ43*2))	=AR42/((AR41*2)+(AR42*2)+(AR43*2))
GraphQL API	=AM43/((AM41*2)+(AM42*2)+(AM43*2))	=AN43/((AN41*2)+(AN42*2)+(AN43*2))	=AO43/((AO41*2)+(AO42*2)+(AO43*2))	=AP43/((AP41*2)+(AP42*2)+(AP43*2))	=AQ43/((AQ41*2)+(AQ42*2)+(AQ43*2))	=AR43/((AR41*2)+(AR42*2)+(AR43*2))

2. Calculate the Weighted Normalised Matrix

TARGET	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.
Views Middleware	=AM47*AM\$39	=AN47*AN\$39	=AO47*AO\$39	=AP47*AP\$39	=AQ47*AQ\$39	=AR47*AR\$39
REST API	=AM48*AM\$39	=AN48*AN\$39	=AO48*AO\$39	=AP48*AP\$39	=AQ48*AQ\$39	=AR48*AR\$39
GraphQL API	=AM49*AM\$39	=AN49*AN\$39	=AO49*AO\$39	=AP49*AP\$39	=AQ49*AQ\$39	=AR49*AR\$39

3. Calculate the ideal best and ideal worst value

V+	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.
V+	=MAX(AM53:AM55)	=MIN(AN53:AN55)	=MAX(AO53:AO55)	=MAX(AP53:AP55)	=MAX(AQ53:AQ55)	=MIN(AR53:AR55)
V-	=MIN(AM53:AM55)	=MAX(AN53:AN55)	=MIN(AO53:AO55)	=MIN(AP53:AP55)	=MIN(AQ53:AQ55)	=MAX(AR53:AR55)

4. Calculate the Euclidean distance from ideal best
5. Calculate the Euclidean distance from ideal worst
6. Calculate performance score

S+	S-	Pi
=((AM53-SAM558)^2)+((AN53-SAN558)^2)+((AO53-SAO558)^2)+((AP53-SAP558)^2)+((AQ53-SAQ558)^2)+((AR53-SARS558)^2)	=((AM53-SAM559)^2)+((AN53-SAN559)^2)+((AO53-SAO559)^2)+((AP53-SAP559)^2)+((AQ53-SAQ559)^2)+((AR53-SARS559)^2)	=AT37/(AU37+AT37)
=((AM54-SAM558)^2)+((AN54-SAN558)^2)+((AO54-SAO558)^2)+((AP54-SAP558)^2)+((AQ54-SAQ558)^2)+((AR54-SARS558)^2)	=((AM54-SAM559)^2)+((AN54-SAN559)^2)+((AO54-SAO559)^2)+((AP54-SAP559)^2)+((AQ54-SAQ559)^2)+((AR54-SARS559)^2)	=AT38/(AU38+AT38)
=((AM55-SAM558)^2)+((AN55-SAN558)^2)+((AO55-SAO558)^2)+((AP55-SAP558)^2)+((AQ55-SAQ558)^2)+((AR55-SARS558)^2)	=((AM55-SAM559)^2)+((AN55-SAN559)^2)+((AO55-SAO559)^2)+((AP55-SAP559)^2)+((AQ55-SAQ559)^2)+((AR55-SARS559)^2)	=AT39/(AU39+AT39)

Figure 18: TOPSIS Spreadsheet Calculations

with 1 being most important and 9 being least important. Rankings of 2, 4, 6, and 8 were permitted for in-between values and were utilised for some variables. To calculate the *TOPSIS* score, the timestamp variables were converted to total minutes. A table containing the average of each metric was constructed, with the exception of *HTTP Actual %*, *HTTP Scan %*, and *Attack Surface* which reflected the highest percentile result among the output data. The remaining steps of the equation were carried out in a series of LibreOffice spreadsheet equations (Figure 18) adapted from a lecture by Manoj Mathew (2018).

Firstly, the normalised matrix of all data was calculated, and then the weighted normalised matrix was likewise calculated using the weighted averages resulting from the *AHP* calculations. Once the weighted normalised matrix was found, calculations for the ideal best and ideal worst scores for each variable category were performed. These results were then taken along with the weighted normalised matrix results and used to find, firstly, the Euclidean distance of a weighted datum from the corresponding ideal best score, which was then squared and summed along with the other data instances. The same was repeated for the Euclidean distance between a weighted datum and its corresponding ideal worst score. The results of the Euclidean ideal best and Euclidean ideal worst were then used to calculate the final performance score of the dependent variable, thus producing a final *work factor* (both a +Work Factor and -Work Factor) metric.

Additionally, chi-square calculations were performed between the +/-Work Factor scores and independent security variables to measure possible correlation. To assess the possible validity of the *work factor* metric, the test statistics for both +Work Factor and -Work Factor were subtracted from their respective *p-values* to compare the distance between them. The strongest correlation metric from each pair was subsequently used as an *AHP* weight comparator. Secondly, to demonstrate the viability of *work factor* as a security metric, the same test statistic results were compared against visualised trends present among the security variables. This was done to determine possible statistical significance in the absence of a traditional *p-value* measure, such as a *t-Test*, which was out of scope for the study.

6. Data & Results

6.1 Raw Data Results

System Information was revealed in the initial *Nmap* (n.d.) scan for all three targets (Figures 19, 22, & 25), while the *Dirsearch* (Soria, 2022) scans resulted in HTTP method disclosure

for all three targets, and endpoint disclosure for VRMA and REST API targets but not for the GraphQL API target (Figures 20 23, & 26). GraphQL *Dirsearch* scans terminated after successful completion of HTTP GET and POST method scans, which shortened the runtime compared to the VRMA and REST API targets (see Table 7). The *Dirsearch* (Soria, 2022) scan CLI commands

Table 7: Averaged Active Reconnaissance Output

Target	System Information	Timestamp	HTTP Actual %	HTTP Scan %	Attack Surface	Scans Performed
VRMA	6	00:36:16.87	33.00%	25.00%	47.37%	3
REST	7	01:14:18.54	100.00%	80.00%	100.00%	3
GraphQL	10	00:11:12.09	100.00%	100.00%	0.00%	5

were the most technically complex, differing between the VRMA and APIs based on the data disclosed in the initial *Nmap* (Nmap, n.d.) scan. For content discovery of the VRMA and REST API targets, *Dirsearch* with JWT authentication was run, disclosing additional endpoints for the REST API target but not for the VRMA target (Figures 21 & 24). This was the final scan performed for either of these targets. Introspection verification and server fingerprinting were carried out against the GraphQL API target using *Nmap* (Nmap, n.d.; Figure 27) and *Graphw00f* (Farhi, 2024; Figure 28), respectively, which resulted in additional *System Information* disclosures. Schema extraction was attempted with the *Clairvoyance* (Stupin, 2023) tool, but was unsuccessful (Figure 29).

```

2 Nmap scan report for localhost (127.0.0.1)
3 Host is up (0.000094s latency).
4 Not shown: 998 closed ports
5 PORT      STATE SERVICE VERSION
6 631/tcp   open ipp      CUPS 2.4
7 | http-robots.txt: 1 disallowed entry
8 |_/
9 |_http-server-header: CUPS/2.4 IPP/2.1
10|_http-title: Home - CUPS 2.4.1
11 3000/tcp  open http    Node.js (Express middleware)
12|_http-title: The Book Shop

```

Figure 19: VRMA – Nmap Output

```

3 422    1KB  http://127.0.0.1:3000/login
4 200    2KB  http://127.0.0.1:3000/signup
5 200    7KB  http://127.0.0.1:3000/products
6 302    28B  http://127.0.0.1:3000/orders    -> REDIRECTS TO: /login
7 302    28B  http://127.0.0.1:3000/cart      -> REDIRECTS TO: /login
8 301    171B http://127.0.0.1:3000/js        -> REDIRECTS TO: /js/
9 301    173B http://127.0.0.1:3000/css       -> REDIRECTS TO: /css/
10 302    28B  http://127.0.0.1:3000/admin/add-product  -> REDIRECTS TO: /login
11 302    28B  http://127.0.0.1:3000/admin/products    -> REDIRECTS TO: /login

```

Figure 20: VRMA – Initial Dirsearch Output (HTTP GET)

```

~
3 422    2KB  http://127.0.0.1:3000/login/
4 200    2KB  http://127.0.0.1:3000/signup/
5 200    9KB  http://127.0.0.1:3000/products/
6 200    2KB  http://127.0.0.1:3000/cart/
7 200    2KB  http://127.0.0.1:3000/orders/
8 200    3KB  http://127.0.0.1:3000/admin/add-product/
9 200    4KB  http://127.0.0.1:3000/admin/products/

```

Figure 21: VRMA – Authenticated Dirsearch Output (HTTP GET)

```

2 Nmap scan report for localhost (127.0.0.1)
3 Host is up (0.000082s latency).
4 Not shown: 998 closed ports
5 PORT      STATE SERVICE VERSION
6 631/tcp   open  ipp      CUPS 2.4
7 | http-robots.txt: 1 disallowed entry
8 |_/
9 |_http-server-header: CUPS/2.4 IPP/2.1
10 |_http-title: Home - CUPS 2.4.1
11 4000/tcp  open  http     Node.js Express framework
12 |_http-cors: GET POST PUT DELETE PATCH
13 |_http-title: Error

```

Figure 22: REST API – Nmap Output

```

2
3 422    295B  http://127.0.0.1:4000/api/auth/signup

```

Figure 23: REST API – Initial Dirsearch Output (HTTP PUT)

```

^
3 500    27B   http://127.0.0.1:4000/api/post/{id}
4 422    295B  http://127.0.0.1:4000/api/auth/signup/

```

Figure 24: REST API – Authenticated Dirsearch Output (HTTP PUT)


```

2 Nmap scan report for localhost (127.0.0.1)
3 Host is up (0.000083s latency).
4 Not shown: 998 closed ports
5 PORT      STATE SERVICE VERSION
6 631/tcp   open  ipp      CUPS 2.4
7 | http-robots.txt: 1 disallowed entry
8 |_/
9 |_http-server-header: CUPS/2.4 IPP/2.1
10|_http-title: Home - CUPS 2.4.1
11 5000/tcp  open  http     Node.js Express framework
12 |_http-cors: GET POST PUT DELETE PATCH
13 |_http-title: Error

```

Figure 25: GraphQL – Nmap Output (Initial)

```

4
3 400      66B      http://127.0.0.1:5000/api/graphql

```

Figure 26: GraphQL – Dirsearch (HTTP POST)

```

2 Nmap scan report for localhost (127.0.0.1)
3 Host is up (0.000062s latency).
4
5 PORT      STATE SERVICE
6 5000/tcp  open  upnp

```

Figure 27: GraphQL – Nmap Output (Introspection)

```

2 Exception: Unable to get TypeRef for ['query { MathJax-master }, 'query { MathJax-master
  { lol } }'] in context Field.
3
4 It is very likely that Field Suggestion is not fully enabled on this endpoint.

```

Figure 28: GraphQL API – Clairvoyance Output

```

1 url,detected_engine,timestamp
2 localhost:5000,Apollo,2024-08-11

```

Figure 29: GraphQL – Graphw00f Output

The BASH automation code was designed to produce 50 output iterations (Appendix X), the average of which can be seen in *Table 7*. These averages represent the raw data which were collected during active reconnaissance, and consist of the independent variables used to calculate the *work factor* metric with *TOPSIS-AHP*. The GraphQL API target had the highest rate of *System Information* disclosure, as well as 100% disclosure among scanned and actual HTTP method calls.

GraphQL had the shortest *Timestamp*, the lowest rate of *Attack Surface* disclosure, and the highest count of *Scans Performed*. The REST API had the longest *Timestamp* despite having a lower *Scans Performed* count. REST *Attack Surface* disclosure was 100% along with 100% disclosure of *HTTP Actual %*. The VRMA overall showed middling results, but had the lowest count of *System Information* disclosure along with the lowest *HTTP Actual %* and *HTTP Scan %* disclosure rates.

6.2 TOPSIS with AHP Results

RATING	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.
SYS INFO	1.0000	0.3333	2.0000	1.0000	0.3333	1.0000
TIMESTAMP	3.0000	1.0000	5.0000	3.0000	1.0000	3.0000
HTTP ACT. %	0.5000	0.2000	1.0000	0.5000	0.2000	0.5000
HTTP SC. %	1.0000	0.3333	2.0000	1.0000	0.3333	1.0000
ATT. SURF.	3.0000	1.0000	5.0000	3.0000	1.0000	3.0000
SCANS PER.	1.0000	0.3333	2.0000	1.0000	0.3333	1.0000
AVERAGE	9.5000	3.2000	17.0000	9.5000	3.2000	9.5000

NORMALISE	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.	AVERAGE
SYS INFO	0.1053	0.1042	0.1176	0.1053	0.1042	0.1053	10.70%
TIMESTAMP	0.3158	0.3125	0.2941	0.3158	0.3125	0.3158	31.11%
HTTP ACT. %	0.0526	0.0625	0.0588	0.0526	0.0625	0.0526	5.70%
HTTP SC. %	0.1053	0.1042	0.1176	0.1053	0.1042	0.1053	10.70%
ATT. SURF.	0.3158	0.3125	0.2941	0.3158	0.3125	0.3158	31.11%
SCANS PER.	0.1053	0.1042	0.1176	0.1053	0.1042	0.1053	10.70%
TOTAL	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	100.00%

Figure 30: AHP of Independent Variables

i. Convert Timestamps

stamp	hour	minute	second	ms	Total Min.
00:36:16.87	0	36	16	87	36.1687
01:14:18.54	1	14	18	54	74.1854
00:11:12.09	0	11	12	9	11.1209

ii. Construct Table

(Weight)	10.70%	31.11%	5.70%	10.70%	31.11%	10.70%
TARGET	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.
Views Middleware	6	36.1687	0.33	0.25	0.47370	3
REST API	7	74.1854	1.00	0.80	1.00	3
GraphQL API	10	11.1209	1.00	1.00	0.00	5

1. Calculate the Normalised Matrix

TARGET	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.
Views Middleware	0.0324324324324324	0.00521514294057315	0.156479681350467	0.146842878120411	0.386885997241618	0.0697674418604651
REST API	0.0378378378378378	0.0106967478815549	0.474180852577173	0.469897209985316	0.816732103106646	0.0697674418604651
GraphQL API	0.0540540540540541	0.00160351583351959	0.474180852577173	0.587371512481645	0	0.116279069767442

2. Calculate the Weighted Normalised Matrix

TARGET	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.
Views Middleware	0.00348902630184364	0.0016223194029826	0.00891199423583994	0.0157065556985089	0.12035287299814	0.0074624403004776
REST API	0.00404719735215091	0.00332755514917049	0.0270060431389089	0.0502609782352284	0.254069848637987	0.0074624403004776
GraphQL API	0.00578171050307273	0.00049882332721007	0.0270060431389089	0.0628262227940355	0	0.012437400500796

3. Calculate the ideal best and ideal worst value

V+	V-
0.00578171050307273	0.00049882332721007
0.00348902630184364	0.00332755514917049
0.00891199423583994	0.00891199423583994
0.0157065556985089	0.0157065556985089
0.254069848637987	0
0.0074624403004776	0.012437400500796

4. Calculate the Euclidean distance from ideal best
5. Calculate the Euclidean distance from ideal worst
6. Calculate performance score

+Work Factor

S+	S-	PI
0.0204344941622327	0.0145124754965886	0.58472864347695
0.0001688956304139	0.0660979752202406	0.0025487189638781
0.0645762382159244	0.00256100786483509	0.96185414186107

-Work Factor

S+	S-	PI
0.0145124754965886	0.0204344941622327	0.41527135652305
0.0660979752202406	0.000168895630413924	0.997451281036122
0.00256100786483509	0.0645762382159244	0.0381458581389301

+Work Factor Results

Target	PI	Rank
Views Middleware	0.58472864347695	2
REST API	0.0025487189638781	3
GraphQL API	0.96185414186107	1

-Work Factor Results

Target	PI	Rank
Views Middleware	0.41527135652305	2
REST API	0.997451281036122	1
GraphQL API	0.0381458581389301	3

Figure 31: TOPSIS Calculation Results

The AHP calculation (Figure 30) in this study ranked *Timestamp* and *Attack Surface* variables as 1, or most important, as the correlation between these two variables should have the

strongest impact on +Work Factor and -Work Factor scores. *System Info*, *HTTP Scan %*, and *Scans Performed* were rated as 3 – moderately important for the *work factor* metric, but not as much so as the former variables. *HTTP Actual %* was rated the lowest score of 5, or neutral, as, though it is a valuable measurement for *work factor* performance, it is predicated on the *HTTP Scan %* results: the former cannot be found without the latter. These rankings were then normalised to produce individual weighted averages to be applied during the *TOPSIS* calculation (Figure 31).

The results for +Work Factor and -Work Factor measurements present complementary inversions of each other: the GraphQL API target had the highest +Work Factor score at 0.9619 and lowest -Work Factor score at 0.0381. VRMA was second in terms of *work factor*, with a +Work Factor score of 0.5847 and a -Work Factor score of 0.4153. The REST API target had the lowest +Work Factor score at 0.0025, with a -Work Factor score of 0.9975. These findings clearly demonstrate a difference in *work factor* measurement, thus allowing a rejection of the null hypothesis.

Table 8: TOPSIS with AHP Results

<u>+Work Factor</u>			<u>-Work Factor</u>		
Target	Pi	Rank	Target	Pi	Rank
GraphQL API	0.96185414	1	REST API	0.99745128	1
VRMA	0.58472864	2	VRMA	0.41527136	2
REST API	0.00254872	3	GraphQL API	0.03814586	3

6.3 Variable Correlation Results

In addition to measuring the *work factor metric* of the above artefacts, a chi-square analysis was conducted between each active reconnaissance variable and -/+Work Factor to measure possible correlation. +Work Factor correlations were found among *System Information* and *Scans Performed* variables (Tables 9 & 14), while -Work Factor correlations were found among *Timestamp*

and *Attack Surface* variables (Tables 10 & 13). No correlations were found between either *work factor* variable and *HTTP Scan %* or *HTTP Actual %* (Tables 11 & 12). This was done to assess the validity of the *AHP* category weights and demonstrate the viability of *work factor* as a security metric.

As a representation of validity, the strength of the correlation between *-/+Work Factor* and the security variables can be observed to coincide with the ranking strength employed between the *AHP* category weights, with the exception of *HTTP Scan %*. Alongside the highest *AHP* rank, *Attack Surface* and *Timestamp* have the most significant correlative *p-value* distances: *Attack Surface* with a *+Work Factor p-value* distance of 1.591 and *Timestamp* with a *+Work Factor p-value* distance of 5.268, respectively. *System Information*, and *Scans Performed* showed middling *p-value* tances, with a *-Work Factor* distance of 0.612 and 0.738, respectively, which mirrors their likewise

Table 9: Chi Square – System Information

System Information / -Work Factor		System Information / +Work Factor	
Alpha	0.001	Alpha	0.001
df	2	df	2
P-value	0.5569480708434080	P-value	0.7216752573662290
Test Statistic	1.1705665469693900	Test Statistic	0.6523600466256560
Critical Value	13.8155105579643000	Critical Value	13.8155105579643000
p-Value dist.	0.6136184761259820	p-Value dist.	-0.0693152107405728

Table 10: Chi-Square – Timestamp

Timestamp / -Work Factor		Timestamp / +Work Factor	
Alpha	0.001	Alpha	0.001
df	2	df	2
P-value	0.9599647519379040	P-value	0.0693362049264179
Test Statistic	0.0817174238513610	Test Statistic	5.3375761431972900
Critical Value	13.8155105579643000	Critical Value	13.8155105579643000
p-Value dist.	-0.8782473280865430	p-Value dist.	5.2682399382708800

Table 11: Chi-Square – HTTP Actual %

HTTP Actual % / -Work Factor		HTTP Actual % / +Work Factor	
Alpha	0.001	Alpha	0.001
df	2	df	2
p-Value	0.6920057147431310	p-Value	0.6233893634150210
Test Statistic	0.7363221301984910	Test Statistic	0.9451679480194000
Critical Value	13.8155105579643000	Critical Value	13.8155105579643000
p-Value dist.	0.0443164154553601	p-Value dist.	0.3217785846043790

Table 12: Chi-Square – HTTP Scan %

HTTP Scan % / -Work Factor		HTTP Scan % / +Work Factor	
Alpha	0.001	Alpha	0.001
df	2	df	2
p-Value	0.6449804036785660	p-Value	0.6457808474820470
Test Statistic	0.8770706890832000	Test Statistic	0.8745901562801920
Critical Value	13.8155105579643000	Critical Value	13.8155105579643000
p-Value dist.	0.2320902854046340	p-Value dist.	0.2288093087981440

Table 13: Chi-Square – Attack Surface

Attack Surface / -Work Factor		Attack Surface / +Work Factor	
Alpha	0.001	Alpha	0.001
df	2	df	2
p-Value	0.9793101800377830	p-Value	0.3743324058314090
Test Statistic	0.0418137061812121	Test Statistic	1.9652221813164300
Critical Value	13.8155105579643000	Critical Value	13.8155105579643000
p-Value dist.	-0.9374964738565710	p-Value dist.	1.5908897754850200

Table 14: Chi-Square – Scans Performed

Scans Performed / -Work Factor		Scans Performed / +Work Factor	
Alpha	0.001	Alpha	0.001
df	2	df	2
p-Value	0.5303024945146140	p-Value	0.7602219518757960
Test Statistic	1.2686153817627900	Test Statistic	0.5482896927917420
Critical Value	13.8155105579643000	Critical Value	13.8155105579643000
p-Value dist.	0.7383128872481730	p-Value dist.	-0.2119322590840530

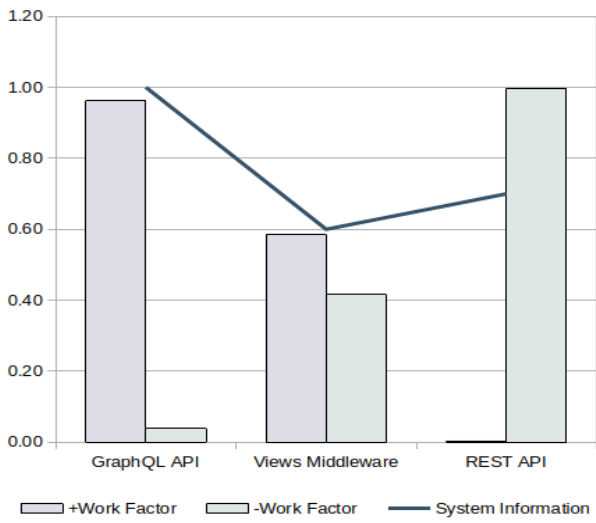


Figure 32: Trend – System Information

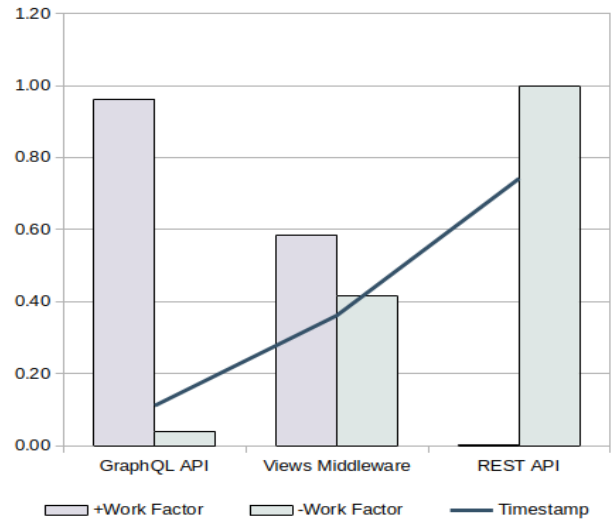


Figure 33: Trend – Timestamp

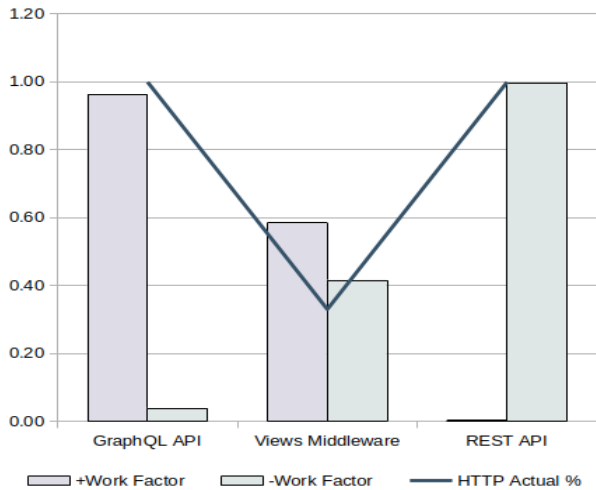


Figure 34: Trend – HTTP Actual %

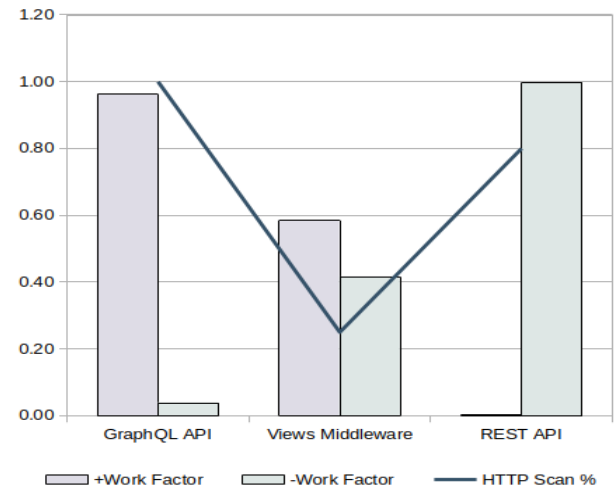


Figure 35: Trend – HTTP Scan %

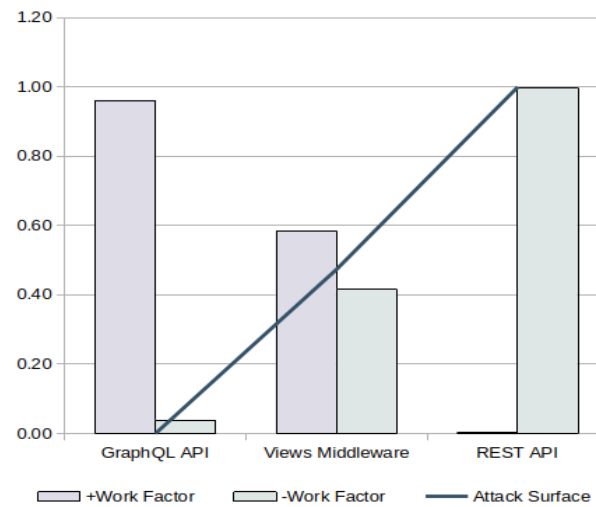


Figure 36: Trend – Attack Surface

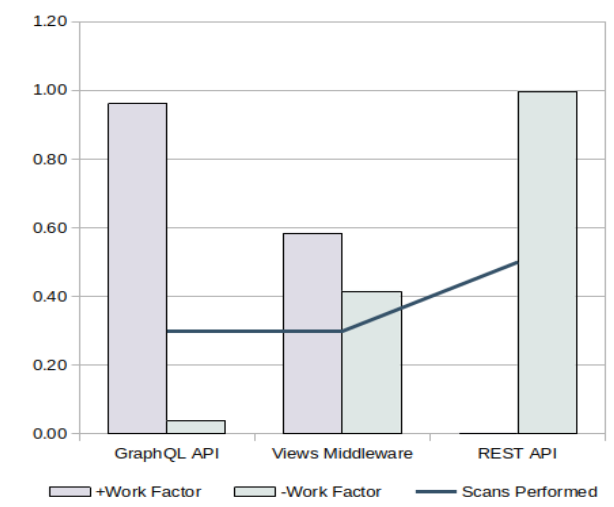


Figure 37: Trend – Scans Performed

middling *AHP* rank. Finally, *Actual Scan %* shows the weakest correlation with a -Work Factor *p-value* distance of 0.322 (test statistic 0.945), which reflects its position as the lowest ranked *AHP* category weight. An exception is found in the *HTTP Scan %* correlation results, which, like *HTTP Actual %*, revealed a weak +Work Factor *p-value* distance of 0.229 (test statistic: 0.875). This suggests that the *AHP* category rank of three may have been too generous, with five seeming more appropriate.

Concerning demonstrable viability of the *work factor* metric, the chi-square results likewise appear to reflect trends found in the visualisation of the relationship between the -/+Work Factor and independent security variable averages (Figures 32 – 37). Comparing the strength of the chi-square *p-value* distances with the slope of the plotted line of the independent security variables demonstrates this most accurately. *Attack Surface* and *Timestamp* variables have near-linear slopes on their respective graphs, with *p-value* distances of -0.937/1.591 and -.878/5.268, respectively.

System Information shows a lesser correlation between -/+Work Factor *p-value* distances (0.613/-0.693), which corresponds with the dichotomous slope incline for both +Work Factor and -Work Factor. Similar trends can be observed between the chi-square results for -/+Work Factor *p-value* distances and *Scans Performed* (0.738/-0.212). *HTTP Scan %* (0.232/0.229) and *HTTP Actual %* (0.044/0.322) again show no relationship to either +/-Work Factor, which suggests that the current *work factor* metric may not be viable for these variables.

7. Analysis & Discussion

7.1 Results Analysis

RQ1. Results of the *work factor* demonstration have produced a few interesting findings: firstly, that *work factor* as a metric can be quantified and applied to other security variables for

comparison. While the studies surveyed above, notably Fulton et al. (2020), made use of calculating the steps and degree of difficulty involved in executing specific exploits, those studies stopped short of calculating a composite *work factor* ranking as a novel metric. This study appears to be the first to modify the calculation of *work factor* as conceptualised by Saltzer and Schroeder into a singular metric for computer security. Extending this type of raw data into a composite metric allows additional comparative measures to be conducted between *work factor* and other security variables, as has been demonstrated with the chi-square calculations.

At the same time, while *Attack Surface*, *System Information*, *Timestamp*, and *Scans Performed* variables proved correlative to the assigned *AHP* weights and overall *work factor* metric, *HTTP Scan %* and *HTTP Actual %* produced disparate results. This lack of demonstrated viability for these variables suggests two findings: firstly, they were not properly quantified. It appears that use of a percentile as the comparative metric did not account for the differing HTTP method call requirements among the artefacts during scanning. This suggests a more robust quantification is required for these variable calculations, or, if not possible, that the metric should most likely be excluded from measurement. Secondly, as these variables were static in nature, without a true zero, it may be that *work factor* is best quantified with dynamic variables that contain a true zero, to which all of the correlative variables in this study can be included.

Additionally, the *HTTP Scan %* variable was found to most likely be overweighted, though the assigned weight and the valid weight were not significantly disparate. It can thus be assumed any negative impact on the validity of the overall *work factor* metric calculation to be negligible in this regard, though still present. The above findings demonstrate that the correlative relationship between *work factor* and corresponding security variables can indeed be spurious, thereby providing some support to the notion that those which are not spurious can thus be considered valid. That said, correlative discrepancies would impact the overall validity of the *work factor* metric depend-

ing on its severity. Yet, given the relative insignificance of the *HTTP Scan %* weight when compared to *Timestamp* or *Attack Surface* variables, and the overall positive demonstration of correlation between *work factor* and the viable security variables, the quantification and weighting errors found in the *HTTP Scan %* variable do not disprove the viability of the *work factor* metric as a whole.

RQ2. Among the architectures themselves, the *work factor* variation between VRMA, REST, and GraphQL reflects interesting divergences during active reconnaissance. The GraphQL +Work Factor is significantly higher than both the VRMA and REST API, with a distance of .377 and .959, respectively. Indeed, among the artefacts tested, GraphQL had the most advantageous *work factor* for malicious actors, while REST had the least and VRMA was middling. While the REST API artefact divulged 100% of its attack surface and the GraphQL API divulged 0%, the amount of time in which it took to do so (74.185 minutes versus 11.121 minutes, respectively) appears to have had the predominant influence on the *work factor* calculations, even with identically weighted *Timestamp* and *Attack Surface* variables.

Time as the dominant variable of the *work factor* metric is further reflected in the chi-square calculations, which demonstrated strong correlations between, firstly, +Work Factor and *Timestamp* and, secondly, -Work Factor and *Attack Surface*. This appears to be the novel contribution of the *work factor* metric as a security analytic, as it is not comparing how much time an attack sequence takes to execute but rather the quality of the relationship a time measurement has to important security variables within an attack sequence. As a result, an attacker or developer can be informed about how the idiosyncrasies of a target architecture can affect the quality of an attack sequence. A primary example from the data is the strongly inverted relationship between the *Timestamp* and *Attack Surface* variables. While this finding could be assumed to apply to any *Timestamp/Attack Surface* comparison, the extreme disparity has most likely resulted from an ar-

chitectural quirk in the GraphQL artefact rather than a naturally inverse relationship.

While REST APIs and VRMAs run endpoints along HTTP method calls, GraphQL does so through schema queries and various resolver functions, as discussed above. Additionally, GraphQL APIs provide an automatic IDE sandbox to test schema endpoints on a browser, offering field suggestions when query errors are encountered (Aleks & Farhi, 2023). The artefact in this study was no exception: manually entering warped queries into the IDE resulted in automated field suggestions (Figure 38). What this artefact lacked was code instructing the use of introspection; it was simply not included in the course material. The absence of introspection functionality is reflected in the *Nmap (Introspection)* scan (Figure 27), as it returns network information with nothing else. No mention of introspection application is presented.

But the *Clairvoyance* output (Figure 29) attained during schema extraction reveals something of more novel interest. This data suggests that, though field suggestions are functional on the GraphQL IDE, they are not fully implemented on the endpoint and thus the brute-forcing scan cannot be completed. No literature studies on the subject of GraphQL field suggestions were found to support or extrapolate this finding, so the prevalence and definitive cause of this result cannot yet be determined. That said, it appears the absence of introspection code in the artefact resulted in the field suggestion function not being fully implemented on the endpoint. The result, a highly advantageous *Timestamp average* alongside a highly disadvantageous *Attack Surface* average, may impact the likelihood of further exploitation, depending on the motivations of the attacker.

It should be noted, though, that such a result for most GraphQL applications in the wild would be unlikely, given the appearance of introspection use among commercial and open source schemas (Baudart et al., 2019) and production-grade server configurations (Schwartzmuller, 2019). The ability to reverse engineer GraphQL endpoints through schema extraction in those instances

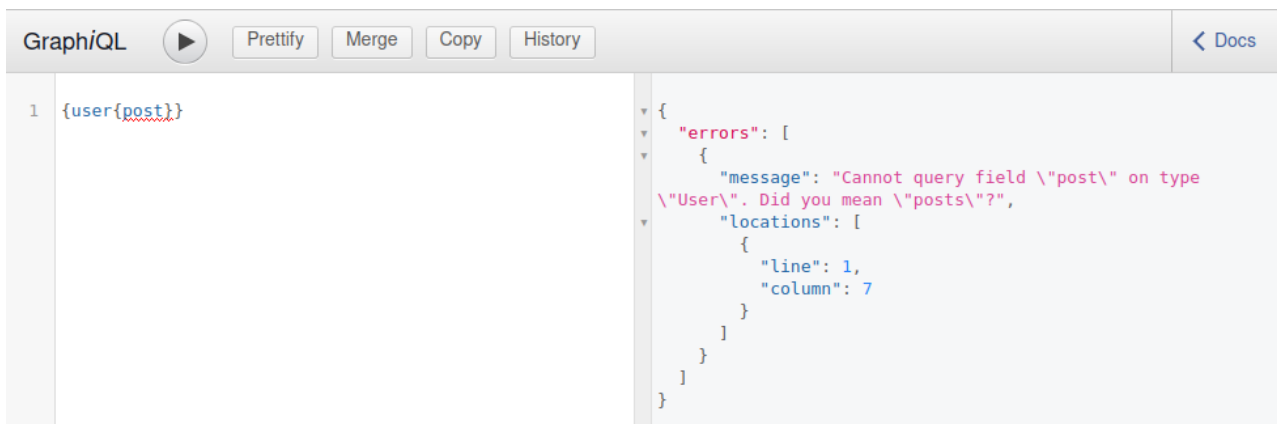


Figure 38: GraphQL IDE – Field Suggestion

may produce a *work factor* that more closely resembles the REST API or VRMA *work factor* results. More testing of GraphQL applications with various implementations of introspection would need to be conducted to better understand the effect of introspection absence on the *work factor* metric. That said, it may be advantageous for GraphQL developers to use a field suggestion obfuscation tool to possibly generate the same result, as such would greatly increase the subsequent *work factor* of required manual testing.

A second example can be found in the *work factor* metric for the REST API artefact. Its significantly low +Work Factor score appears to be the result of more comprehensive scanning requirements compared to the VRMA and GraphQL targets. Because the initial *Nmap* (*Nmap*, n.d.) scan identified the VRMA as a *Node.js Expressware* application, a five-method HTTP call scan cycle with additional subdirectories including `/api` as a prefix were not necessary, which shortened the *Timestamp* output. The REST API was identified alongside five HTTP method calls, which made their inclusion mandatory. While the GraphQL API was also identified alongside five HTTP method calls, the ‘GraphQL tripwire’ terminated before running through all five HTTP methods which lowered its overall *Timestamp* output.

Additionally, because of the middleware configuration, all of the VRMA *Attack Surface*

output occurred during the initial GET scans, while the REST *Attack Surface* output was scattered across four of the five HTTP method calls, a practice known as *HTTP Interaction Design* (Massé, 2011). Although not reflected in this study, an impatient malicious actor could choose to terminate the VRMA scan cycle prematurely to begin further exploitation on the immediate results. This is not an option with the REST *Attack Surface* output. As malicious actors must wait longer for REST API endpoint exposure, this may discourage thorough scanning. At the same time, the prospect of total *Attack Surface* disclosure may prove the outsized *work factor* requirement acceptable for some attackers and attack vectors.

The influence of the *work factor* metric seems to ultimately depend on the motivations of the malicious actor, which would be contingent on the actor's skill level, desired attack vector, desired outcome, and interest in the target. But from a *work factor* perspective, these two examples seem to demonstrate that API design idiosyncrasies can impact the security of an application by making it prohibitively obtuse or time-consuming to scan comprehensively. It appears advisable for developers and security professionals to be aware of the effect *work factor* may have on the attack desirability of a target at the design level, as this may provide an additional layer of security against malicious action.

7.2 Study Limitations

While this study does successfully demonstrate the arrangement, calculation, and further manipulation of a *work factor* security metric, the generalisability of the metric results across web application architectures is limited due, firstly, to the validity of the *TOPSIS-AHP* calculations of *Scan HTTP %* and, secondly, to the small sample size of artefacts the *work factor* metric was assembled against: a single VRMA, REST API, and GraphQL API. The small sample size was due to the requirements, scope, and time limitations of the dissertation, and while the results do show a

demonstrable difference in the *work factor* metrics of each artefact, these cannot be extended to other applications as a rule.

Three areas for improved generalisability exist. Firstly, security variable calculations should be tested to prove validity in relation to the target measurement of the *work factor* metric before *TOPSIS-AHP* is performed. This would reduce the chance of a spurious relationship between variables, as was found between the percentile *HTTP Scan %* and *HTTP Actual %* variables and the *work factor* metric. Secondly, given the subjective nature of *AHP* weighting, it may be judicious to calculate these weights from an averaged sample based on informed industry opinion. Though this study's results found mostly valid weight measurements via the chi-square results, as the weights were subject to the author's sole opinion, an opposite finding could have just as easily occurred. Thirdly, target testing should be significantly larger and more diverse, especially when considering the possible variation across REST API architectures and GraphQL field suggestion implementations, in order to better generalise differences in their respective results.

Though the results of the *work factor* metric are not currently generalisable, several measures were taken to design the metric to be reproducible by limiting other threats to validity: firstly, the risk of construct validity was reduced by declaring each dependent and independent variable in the study, as well as how each would be measured according to their use. This was done to minimise inadequate pre-operational explication of constructs and mono-method bias. Expanding the variables which comprise the *work factor* metric to include the entirety of the active reconnaissance output was done to reduce the risk of confounding constructs and level of constructs. All artefact code and active reconnaissance scripts were tested independently before deployment to reduce any possible internal validity threat.

Additionally, the artefact code was selected from an online course to reduce the risk of in-

valid instrumentation, selection, and ambiguity about the direction of causal influence. This code was likewise chosen to limit possible risks to external validity, as adherence to course content demonstrating a wide range of techniques meant for professional application production was more likely to adhere to industry standards than original, unverified artefact code. All of this was done to ensure researchers would have the ability to expand on the *work factor* metric in possible future research.

7.3 Knowledge Gained

Several theoretical and technical skills were gained over the course of study:

Time management. A six-month timeframe for a dissertation is deceptively short. From the outset it was clear that though 28 weeks were available to develop and execute the written project, a very strict schedule for reading, artefact production, hypothesis testing, and writing was required to meet the deadline while still producing quality output. Procrastination is an easy bedfellow, so weekly written updates (Appendix XI) were dispatched to create a sense of accomplishment with each completed task alongside accountability for any missed or late updates as to not fall too far behind. This proved to be an instrumental motivating factor during the arduous portions of the project.

Criticality. Employing criticality within the literature review was a daunting task, as this skill had been a source of difficulty in past modules. While the logic of comparing and contrasting different references became clear after engagement with Bloom's Taxonomy (1956; Appendix XII), the creative aspect of criticality, whereby one is able to see beyond the criticism and form a complementary extension of thought, remained elusive. But inspiration struck in an unlikely finding by Bogner & Kotstein (2021), in which developers found no security benefits present among REST API architectural best practice. This opened a floodgate of investigation, which then ex-

panded into GraphQL literature, Saltzer and Schroeder (1975), and finally the *work factor* metric itself.

Artefact Assembly. As the author's technical knowledge was limited at the beginning of the project, the need to rely on a comprehensive coding course to produce valid artefacts became apparent once the literature review was finished and the project outline was written. Time management was instrumental while finishing the VRMA and API source code artefacts. Frequent code examples would prove broken or deprecated, requiring sometimes prohibitively advanced mitigation attempts relative to the coding skills of the author. If a record of completion had not been kept from the outset, the building of the three web application artefacts would very likely have been abandoned for a smaller project. But because the author felt beholden to providing quality weekly updates, these provided the perseverance required to finish the artefacts as originally conceived.

Metric Conceptualisation. Previous knowledge of the hacker mindset played a pivotal role in choosing *work factor* as a viable security metric and its quantification as a composite metric. While very few formal studies will include patience as an elite hacking skill, those in the know know that very few hackers will wait endlessly for an exploit to pay off. Attacks are often quick and dirty, not only because of the motivations of the actor but because there is substantial risk increase when one sustains long-term exposure to a target. Similarly, useful data and disclosures must come along the way for an exploit to be beneficial – thus requiring *work factor* to be a composite score rather than a simple timestamp metric. Having this prerequisite understanding resulted in the logical conclusion that components making up the composite *work factor* should measure the benefits of patience for a particular attack on a particular target as well as the timestamp associated. The difficulty lay in validating and verifying the quality of such a metric, and for this the statistics instruction encountered on the research methods course (Outram, 2023) was truly invaluable, as the author was ultimately able to find a reasonable *p-value* statistic off which to base

metric validity through chi-square analysis. Use of this statistic as a validation measure is arguably the hinge on which the validity of the entire study rests.

8. Conclusion

Though API architectures are among the most vulnerable technologies to cyber attack, quantifying independent security variables to measure the possible security effects of architectural idiosyncrasies has proven difficult. This study has attempted to bridge this gap by demonstrating the viability of *work factor* as a security metric through its application to the results of an automated active reconnaissance cycle of a traditional views-rendering middleware application, REST API, and GraphQL API. To accomplish the above, information disclosures gathered during active reconnaissance were categorised as independent security variables, averaged, and weighted using *TOPSIS-AHP* to calculate a composite *work factor* metric.

A chi-square analysis was conducted to assess the validity of the metric and the viability of the ranked results as a comparative measure against other security variables. Firstly, the chi-square tests were able to validate all variables with dynamic output and a true zero quantified by *TOPSIS-AHP*, which lends credibility to the metric results but does not prove the definitive validity of the quantification as currently conceived. Furthermore, while a correlative relationship was demonstrated among the validated security variables, of particular interest was the strong correlation found between disadvantageous (negative) *work factor* and the rate of *Attack Surface* disclosure.

GraphQL was shown to have the highest ranked positive *work factor* (0.9619) alongside the least *Attack Surface* disclosure (0%), REST having the lowest rank (0.0025) alongside the highest *Attack Surface* disclosure (100%), and the views-rendering middleware placing in the middle for both measures (0.4153 and 47.37%). The extreme difference between these *work factor* metric findings is most likely due to coding decisions which led to the GraphQL field suggestion feature

not being fully enabled on the endpoint of the artefact itself, along with inherent REST API architectural idiosyncrasies requiring a wider scope of reconnaissance. That said, given the limited scope of data in the study, these findings are not currently generalisable. Further extrapolation of the apparent inverse relationship between *work factor* and *Attack Surface* is needed to better inform researchers and developers about the likelihood of certain attack vectors on certain technologies and actor motivations.

API security is often ignored at the architectural level, leaving much of the security handling to validation, authentication, and authorisation layers. The *work factor* metric demonstration in this study suggests this practice may be shortsighted when certain aspects of low-level, architectural security could increase the overall *work factor* of a target. Increasing the *work factor* in this way may deter impatient malicious actors from performing comprehensive testing of a target, or from testing a target altogether. Failure to fully implement automated field suggestions on the GraphQL endpoint serves as an example of this. While this missing feature did produce an advantageous *work factor* score for automated active reconnaissance, it also resulted in 0% *Attack Surface* disclosure which, conjecture would permit, would most likely lead to a higher *work factor* requirement in subsequent manual testing.

Likewise, when comparing the *work factor* scores of the REST API and traditional views-rendering middleware application, though the REST API had 100% *Attack Surface* disclosure, the time involved to do so was more than twice the time it took to disclose nearly half of the traditional middleware application attack surface. REST's use of diverse HTTP methods for API calls appears to be the primary reason for the disparity in the automated active reconnaissance *work factor*, as this increased the scope of the API attack surface greatly. Malicious actors who do not wish to expend the time required for full attack surface disclosure may only partially scan HTTP method calls, which would result in far less attack surface disclosure. These examples would seem to dis-

pel the notion that adherence to API architectural rules has no impact on security, at least from a *work factor* perspective.

While the results of this study have successfully demonstrated the potential of *work factor* as a security metric for computer security, the testing of the *work factor* metric against three separate artefacts is not so statistically significant as to prove a generalisation which should be applied to other architectures and technologies. Further studies could build upon the conceptualisation of the *work factor* calculation as well as expand uses for the metric within security research. Perhaps the most pertinent areas of further research are in generalisability: being able to widely apply the *work factor* metric with reproducible *AHP* calculations, and comprehensive testing of the *work factor* metric on a statistically significant data set. It is also important to note that application of the *work factor* metric in security research is not limited to APIs and active reconnaissance. Further studies incorporating other attack vectors or target technologies, which could expand upon and mature the understanding of the *work factor* metric in terms of viable application and limitations, are also possible.

9. Cited References

- Acar, Y., et al. (2017) Comparing the Usability of Cryptographic APIs. In: IEEE Symposium on Security and Privacy. IEEE: 154 – 171
- Akamai (2024) Lurking in the Shadows: Attack Trends Shine Light on API Threats. SOTI, 10 (1): 1 – 33
- Aleks, N. & Farhi, D. (2023) Black Hat GraphQL. No Starch Press. San Francisco, CA, USA.
- Alliyu et al. (2017) Preliminary Analysis on REST API Style Guidelines. In: PLATEAU'17 Workshop on Evaluation and Usability of Programming Languages and Tools, 23 October, 2017, Vancouver, CA: 1 – 9
- Al-Rumain, A. & Pawar, J. D. (2023) API Security Challenges – Security Professionals Overview. China Petroleum Processing and Petrochemical Technology, 23 (2): 3115 - 3135
- Anthonyssamy et al. (2020) Schrödinger's Security: Opening the Box on App Developers' Security Rationale. In: ICSE '20, 23 – 29 May, 2020, Seoul, Republic of Korea. ACM: 149 - 160
- Arcuri, A., Marculescu, B., & Zhang, M. (2022) On the Faults Found in REST APIs by Automated Test Generation. ACM Transactions on Software Engineering and Methodology, 31 (3): 41:1 – 41:43
- Assal, H. & Chiasson, S. (2018) Security in the Software Development Lifecycle. In: 14th Symposium on Usable Privacy and Security. USENIX: 281 – 296
- Assetnote (2020) 2m-subdomains.txt | Manually Generated Wordlists. wordlists.assetnote.io [available online] <https://wordlists.assetnote.io/>
- Baez et al. (2016) REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. ICWE 2016, LNCS 9671: 21 - 39
- Bailey, M., Dittrich, D., & Kenneally, E. (2013) Applying Ethical Principles to Information and Communication Technology Research: A Companion to the Menlo Report. US Department of Homeland Security: 1 – 32
- Balioti, V., Evangelides, C., & Tzimopoulos, C. (2018) Multi-Criteria Decision making Using TOPSIS Method Under Fuzzy Environment. Application in Spillway Selection. MDPI, 2: 1 - 8
- Ball, C. J. (2021) Hacking APIs: Breaking Web Application Programming Interfaces. No Starch Press. San Francisco, CA, USA
- Bansal, A. (2023) HGQL: Supporting Schematic Hypergraphs in GraphQL. In: IDEAS 2023, 05 – 07 May, 2023, Heraklion, Crete, Greece. ACM: 9 – 16
- Bastidas, E. et al. (2022) Quality in Use Evaluation of a GraphQL Implementation. In: CIT 2021, LNNS 405. Springer Nature: 15 – 27
- Baudart, G et al. (2019) An Empirical Study of GraphQL Schemas. In: ICSOC 2019, LNCS 11895: 3 – 19
- Baudart, G. et al. (2021) Learning GraphQL Query Cost. In: 36th IEEE/ACM International Conference on Automated Software Engineering. IEEE: 1146 – 1150

- Bernardino, J., Laranjeiro, N., & Neumann, A. (2021) An Analysis of Public REST Web Service APIs. *IEEE Transactions on Services Computing*, 14 (4): 957 - 970
- Bloch, J. (2006) How to Design a Good API and Why it Matters. In: 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications. ACM: 506 – 507
- Bloom, B. S. et al. (1956) Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook 1: Cognitive Domain. Longmans. London, WI, USA.
- Bogner, J. & Kotstein, S. (2021) Which RESTful API Design Rules Are Important and How Do They Improve Software Quality? A Delphi Study with Industry Experts. Springer Nature. SummerSOC 2021, CCIS 1429: 154 – 173
- Bogner, J., Kotstein, S., & Pfaff, T. (2023) Do RESTful Design Rules have an Impact on the Understandability of Web APIs? *Empirical Software Engineering*, 28:132: 1 - 35
- Borroel, E. Z. et al. (2023) CODAS, TOPSIS, and AHP methods Application for Machine Selection. *Journal of Computational and Cognitive Engineering*, 2 (4):322 - 330
- Brito, G., Momach, T., & Valente, M. T. (2019) Migrating to GraphQL: A Practical Assessment. In: SANER 2019, Hangzhou, China. IEEE: 140 – 150
- Buna, S. (2021) GraphQL in Action. Manning. Shelter Island, NY, USA.
- Chng et al (2022) (2018) Hacker Types, Motivations, and Strategies: A Comprehensive Framework. *Computers in Human Behavior Reports*, 5: 1 – 8
- Chowdhury, P. D., Tahaei, M., & Rashid, A. (2022) Better Call Saltzer & Schroeder: A Retrospective Security Analysis of SolarWinds & Log4j. *ArXiv*: 1 -8
- Cito & Happa (2023) Understanding Hackers’ Work: An Empirical Study of Offensive Security Practitioners. ESEC/FSE ‘23, 3 – 9 December, 2023, San Francisco, CA, USA. ACM: 1669 - 1680
- Cokrowibowo, S., Firgiawan, W., & Zulkarnaim, N. (2020) A Comparative Study using SAW, TOPSIS, SAW-AHP, and TOPSIS-AHP for Tuition Fee (UKT). In: The 3rd EPI International Conference on Science and Engineering 2019. IOP: 1 – 5
- Consens, M. P., Hartig, O., & Kin, Y. W. (2019) An Empirical Anslsysis of GraphQL API Schemas in Open Code Respositories and Package Registeries. In: AMW, 3 June 2019: 1 – 5
- Diaz, T., Olmedo, F., & Tanter, E. (2020) A Mechanized Formalization of GraphQL. In: CPP ‘20, 20 – 21 January, New Orleans, LA, USA. ACM: 201 - 214
- Dimitrovski, I., Kitanovski, I., & Spasev, V. (2020) An Overview of GraphQL: Core Features and Architecture. *Ukim.mk*: 1 – 14 [Available Online]
https://repository.ukim.mk/bitstream/20.500.12188/19669/1/ICT_2020_submission_40.pdf
- Doolittle, J. (2023) APIs with GraphQL. *IEEE Software*: 118 – 120
- Downey, A. B. (2014) Think Stats: Exploratory Data Analysis. O’Reilly. Sebastapol, CA, USA.
- Dwyer, A. C., et al. (2023) SLR: From Saltzer and Schroeder to 2021...47 Years of Research on the Development and Validity of Security API Recommendations. *ACM Transactions on Software Engineering and Methodology*, 32 (3): 60:1 – 60:31

- Farhi, D. (2024) graphw00f | dolevf. github.com [Available online]
<https://github.com/dolevf/graphw00f>
- Fernandez, P. et al. (2023) GraphQL: A Systematic Mapping Study. *ACM Computing Surveys*, 55 (10): 202:1 – 202:35
- Fielding, R. T. (2000) *Architectural Styles and the Design of Network-based Software Architecture*. Doctoral thesis.
- Fowler, M. (2010) Richardson Maturity Model: Steps Toward the Glory of REST | Articles. martinfowler.com. [Available Online]
<https://martinfowler.com/articles/richardsonMaturityModel.html>
- Friedrich, T. (2013) RESTful Service Best Practices: Recommendations for Creating Web Services. RestApiTutorial.com [Available Online]
https://raw.githubusercontent.com/tfredrich/RestApiTutorial.com/master/media/RESTful%20Best%20Practices-v1_2.pdf
- Fulton, K. R., et al. (2020) Understanding Security Mistakes Developers Make: Qualitative Analysis from Build It, Break It, Fix It. In: 29th USENIX Security Symposium. *USENIX*: 1 – 18
- GraphQL (2021) GraphQL: October 2021 Edition. [Available Online]
<https://spec.graphql.org/October2021/>
- Green, M. & Smith, M. (2016) Developers are Not the Enemy!: The Need for Useable Security APIs. *IEEE Secure Privacy*, 14 (5): 40 – 46
- Guéhéneuc, Y-G., et al. (2016) Are REST APIs for Cloud Computing Well Designed? An Exploratory Study. In: *ICSOC 2016, LNCS 9936*. Springer: 157 – 170
- Gutman, P. (2002) Lessons Learned in Implementing and Deploying Crypto Software. In: *Usenix Security Symposium*. *USENIX*: 315 – 325
- Hallett, J., Rashid, A, & Patnik, N. (2019) Usability Smells: An Analysis of Developers' Struggle with Crypto Libraries. In: 15th Symposium on Usable Privacy and Security. *USENIX*: 245 – 257
- Hartig, O. & Pérez, J. (2018) Semantics and Complexity of GraphQL. In: *WWW 2018*, 23 – 27 April, 2018, Lyon, FR. *ACM*: 1155 – 1164
- Hu et al. (2018) Hackers vs Testers: A Comparison of Software Vulnerability Discovery Process. In: 2018 IEEE Symposium on Security and Privacy. *IEEE*: 374 - 391
- Lawi, A., Panggabean, B. L. E., & Yoshida, T. (2021) Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System. *Computers*, 10 (138): 1 – 16
- Leal, J. E. (2020) AHP-Express: A Simplified Version of the Analytical Hierarchy Process Method. *MethodsX* 7: 1 - 11
- Massé, M. (2011) *REST API Design Rulebook*. O'Reilly. Sebastapol, CA, USA
- Mathew, M. (2018) TOPSIS Using Excel – MCDM Problem | Manoj Mathew. youtube.com [Available online] <https://www.youtube.com/watch?v=Br1NQK0Iumg>
- Meshram, S. U. (2021) Evolution of Modern Web Services -- REST API with its Architecture and Design. *International Journal of Research in Engineering, Science, and Management*, 4 (7): 83 – 86

- Munsch & Munsch (2021) The Future of API (Application Programming Interface) Security: The Adoption of APIs for Digital Communications and the Implications for Cyber Security Vulnerabilities. *Journal of International Technology and Information Management*, 29 (3): 25 - 45
- Myers, B. A. & Stylos, J. (2016) Improving API Usability. *Communications ACM*, 59 (6): 62 – 69
- Nmap (n.d) Chapter 15. Nmap Reference Guide | Nmap Network Scanning. nmap.org [Available online] <https://nmap.org/book/man.html>
- OpenAI ChatGPT (2024). *Various ChatGPT responses to Laura M. Saxton*. 6 July – 1 August 2024.
- Outram, K. (2023) ‘Unit 8: Inferential Statistics’, *Research Methods and Professional Practice June 2023*, University of Essex [Available Online] <https://www.my-course.co.uk/course/view.php?id=10163§ion=14>
- OWASP (2023) OWASP Top 10 API Security Risks – 2023. OWASP API Security Project. [Available Online] <https://owasp.org/API-Security/editions/2023/en/0x11-t10/>
- OWASP (2024a) REST Security Cheat Sheet | OWASP Cheat Sheet Series. cheatsheetseries.owasp.org [Available Online] https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html
- OWASP (2024b) GraphQL Cheat Sheet | OWASP Cheat Sheet Series. cheatsheetseries.owasp.org [Available Online] https://cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html
- Palma, F., et al. (2014) Detection of REST Patterns and Antipatterns: A Heuristics-based Approach. *Service-Oriented Computing*. Berlin, Heidelberg. Springer: 230 – 244
- Palma, F, et al. (2017) Semantic Analysis of RESTful APIs for the Detection of Linguistic Patterns and Antipatterns. *International Journal of Cooperative Information Systems*, 26 (02): 1742001
- Palma, F, , Sadia, A., & Zarraa, O. (2021) Are Developers Equally Concerned About Making Their APIs RESTful and the Linguistic Quality? A Study on Google APIs. In: Hacid H, et al. (eds.) *Service-Oriented Computing*, Springer International Publishing, Cham, CH: 171 – 187
- Palma, F. et al. (2022) Investigating the Linguistic Design Quality of Public, Partner, and Private REST APIs. In: 2022 IEEE International Conference on Services Computing. IEEE: 20 – 30
- Paxton-Fear, K. (2021) API Hacking 101, w/ Dr. Katie Paxton-Fear | Traceable AI. [Available Online] <https://www.youtube.com/watch?v=qC8NQFwVORO>
- Postman (2024) Postman API Platform. [postman.com](https://www.postman.com/) [Available Online] <https://www.postman.com/>
- Qazi, F. A. (2023) Application Programming Interface (API) Security in Cloud Applications. *EAI Endorsed Transactions on Cloud Systems*: 1 – 14
- Raj, P. & Subramanian, H (2019) *Hands on RESTful API Design Patterns and Best Practices*. Packt Publishing. Birmingham, UK.
- Saltzer, J. H. & Schroeder, M. D. (1975) The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63 (9): 1278 – 1308
- Schwartzmüller, M. (2019) *Node.js – The Complete Guide* | Academind. Packt Publishing. [Available Online] <https://learning.oreilly.com/course/node-js-the/9781838826864/>

Siriwardena, P. (2020) *Advanced API Security: OAuth 2.0 and Beyond*. 2nd Ed. New York, NY, USA. Apress.

Soria, M. (2022) *dirsearch* | maurosoria. github.com [Available online]
<https://github.com/maurosoria/dirsearch?tab=readme-ov-file>

SpiceLogic Inc. (2022) *AHP Calculation Methods* | Tutorials. spicelogic.com [Available Online]
<https://www.spicelogic.com/docs/ahpsoftware/intro/ahp-calculation-methods-396>

Stupin, N. (2023) *clairvoyance* | nikitastupin. github.com [Available online]
<https://github.com/nikitastupin/clairvoyance>

10. Bibliography

- Abeck, S. et al. (2016) Checklist for the API Design of Web Services Based on REST. *International Journal of Advances in Internet Technology*, 9(3 & 4): 41 – 51
- Afanasieva, A. M., & Tkachov, V. M. (2023) The Differences Between GraphQL and REST APIs: Understanding their Development Lifecycles. *Kharkov.ua*: 21 [Available Online] <https://repository.kpi.kharkov.ua/server/api/core/bitstreams/88e08c47-08fe-4379-a290-c9089f1696d4/content>
- Amundsen, M. (2022) *RESTful Web API Patterns and Practices Cookbook*. O'Reilly. Sebastapol, CA, USA.
- Arts, A. et al. (2021) Can GraphQL Replace REST? A Study of Their Efficiency and Viability. In: *IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice*. IEEE: 10 – 17
- Atzori, M. et al. (2020) Special Issue on "Data Exploration in the Web 3.0 Age". *Future Generation Computer Systems.*, 112: 1177 -1179
- Bahari, M. et al. (2020) Understanding Service-Oriented Architecture (SOA): A Systematic Literature Review and Direction for Further Investigation. *Information Systems*, 91: 1 – 27
- Balcer et al. (2008) *Applied SOA: Service Oriented Architecture and Design Strategies*. Wiley. Indianapolis, IN, USA: 27 – 76
- Basishtha et al. (2014) Web 1.0 to Web 3.0 - Evolution of the Web and its Various Challenges. In: *2014 International Conference on Reliability, Optimisation, and Information Technology, India, 6 - 8 Feb, 2014*: 86 – 89
- Bodden, E. et al. (2022) FUM – A Framework for API Usage Constraint and Misuse Classification. In: *IEEE International Conference on Software Analysis, Evolution, and Reengineering*. IEEE: 673 – 684
- Bosse et al. (2019) Internet of Things Middleware: How Suitable are Service-Oriented Architecture and Resource-Oriented Architecture. In: *Proceedings of the 3rd International Conference on Internet of Things, Big Data, and Security*. SCITEPRESS: 229 – 236
- Brito, G. & Valente, M. T. (2020) REST vs. GraphQL: A Controlled Experiment. *International Conference on Software Architecture*. IEEE: 81 – 91
- Cabot, J., Izquierdo, J. K. C., & Rodriguez-Echeverria, R. (2018) Towards a UML and IFML Mapping to GraphQL. In: *ICWE 2017, LNCS 10544*. Springer Nature: 149 – 155
- Calibrant (2024) *The Step-by-Step Guide for Roadmap to Web 3.0 Application in 2024* | Calibrant. [Available Online] <https://www.calibrant.com.au/blog/roadmap-to-web-3-applications>
- Chris, K. (2023) *SOLID Design Principles in Software Development* | freeCodeCamp. [freecodecamp.org](https://www.freecodecamp.org/news/solid-design-principles-in-software-development/). [Available Online] <https://www.freecodecamp.org/news/solid-design-principles-in-software-development/>
- Chou, W. et al. (2016) Design Patterns and Extensibility of REST API for Networking Applications. *IEEE Transactions on Network and Service Management*, 13 (1): 154 – 167

- Chowdhury et al., (2021) Developers are Neither Enemies Nor Users: They are Collaborators. In: *2021 IEEE Secure Development Conference (SecDev)*. IEEE: 47 - 55
- Decker, S. et al. (2021) LISSIU: Integrating Semantic Web Concepts into SOA Framework. In: *Proceedings of the 23rd International Conference on Enterprise Information Systems*, 1. SCITEPRESS: 855 – 865
- Duan, S. et al (2023) Analysis and Discussion of Resource-Oriented Service Architecture for Online Programming Ecology. In: *26th ACIS International Winter Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. IEEE: 64 – 68
- Dyyak, I., Marchuk, Y., Makar, I. (2023) Performance Analysis of Database Access: Comparison of Direct Connection, ORM, REST API, and GraphQL Approaches. In: *13th International Conference on Electronics and Information Technologies*. IEEE: 174 – 176
- Eskandarzadeh, A. (2023) The Future of Web 3: A Roadmap for the Decentralized Web. In: Kaur, G. & Lekhi, P. (eds) *Concepts, Technologies, Challenges, and the Future of the Web*. IGI Global. Hershey, PA, USA.
- Flores-García, E., Heredia, J. S., & Solano, A. R. (2019) Comparative Analysis Between Standards Oriented to Web Services: SOAP, REST, and GraphQL. In: Botto-Tobar, M., et al. (eds) *Applied Technologies. ICAT 2019. Communications in Computer and Information Science*, 1193. Springer, Cham: 286 – 300
- Ford, N. & Richards, M. (2020) *Fundamentals of Software Architecture*. 4th Revision. O'Reilly Media. Sebastapol, CA, USA
- Escoffier, C. & Lalanda, P. (2017) Resource-Oriented Framework for Representing Pervasive Context. In: *2017 IEEE International Congress on Internet of Things*. IEEE: 155 – 158
- Gan, W. et al. (2023) Web 3.0: The Future of the Internet. In: *WWW '23 Companion, 30 April - 4 May, 2023, Austin, TX, USA*. ACM: 1 – 10
- Gorski, P. L. et al. (2022) “I Just Looked for the Solution!” On Integrating Security-Relevant Information in Non-Security API Documentation to Support Secure Coding Practices. *IEEE Transactions on Software Engineering*, 48 (9): 3467 – 3484
- Guha, S. & Majumder, S. (2020) A Comparative Study Between GraphQL & RESTful Services in API Management of Stateless Architecture. *International Journal on Web Computing*, 11 (2): 1 – 16
- Hohpe, G. & Woolf, B. (2004) *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Pearson. Boston, MA, USA.
- Hohpe, G. & Woolf, B. (2023) *Remote Procedure Invocation | Messaging Patterns*. Enterprise Integration Patterns. [Available Online]
<https://www.enterpriseintegrationpatterns.com/patterns/messaging/EncapsulatedSynchronousIntegration.html>
- Höst, M., et al. (2012) Planning. In: *Experimentation in Software Engineering*, Berlin, Heidelberg. Springer Berlin Heidelberg: 89 – 116
- Hustad, E. & Olsen, D. H. (2021) Creating a Sustainable Digital Infrastructure: The Role of Service-Oriented Architecture. *Procedia Computer Science*, 181: 597 - 604

- Ireland, S. M. & Martin, A. C.R. (2021) GraphQL for the Devliery of Bioinformatics Web APIs and Application to ZincBind. *Bioinformatics Advances*: 1 – 7
- Jambusaria, A. (2021) GraphQL Service Layer to Enable Client-driven, Optimised, and Secure Front-end Architecture. *International Research Journal of Engineering and Technology*, 8 (4): 303 – 308
- Konieczny, M., Rokseła, P., & Zielinski, S. (2020) Evaluating Execution Strategies of GraphQL Queries. *TSP 2020*. IEEE: 640 – 644
- Kornienko, D. V. et al. (2021) Principles of Securing RESTful API Web Services Developed with Python Frameworks. *Journal of Physics: Conference Series*, 2094: 1 – 11
- Lamo, Y. et al. (2019) A GraphQL Approach to Healthcare Information Exchange with HL7 FHIR. *Procedia Computer Science*, 160: 338 – 345
- Liu Y. et al. (2008) Resource-Oriented Architecture for Business Purposes. In: *15th Asia-Pacific Software Engineering Conference*. IEEE: 395 – 402
- Manuba, I. B. K. & Vohra, N. (2022) Implementation of REST API vs GraphQL in Microservice Architecture. *International Conference on Information Management and Technology*, 11 – 12 August, 2022. IEEE: 45 – 50
- Martin, R. C. (2023) *Functional Design: Principles, Patterns, and Practices*. Addison-Wesley. Hoboken, New Jersey, USA.
- Rasheedh, J. A. & Saradha, S. (2020) Review of Micro-services Architectures and Runtime Dynamic Binding. In: *Fourth International Conference on I-SMAC*. IEEE: 1130 – 1137
- Richardson, L. & Ruby, S. (2007) *Restful Web Services*. O'Reilly. Sebastapol, CA, USA.
- StudySmarter (2024) *Distributed Programming | Explanations*. StudySmarter [Available Online] <https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/distributed-programming/>
- Wilkins, P. (2022) *The Differences Between APIs and Messaging for Application Communication | What is an API?*. Oracle Canada. [Available Online] <https://www.oracle.com/ca-en/cloud/cloud-native/api-management/what-is-api/apis-and-messaging-differences/>
- Zanevych, O. (2024) Advancing Web Development: A Comparative Analysis of Modern Frameworks for REST and GraphQL Back-end Services. *International Scientific Journal «Grail of Science»*, (37): 216 – 228

11. Appendices

11.1 Appendix I: Views-Rendering Middleware Source Code

11.1.1 Main Program Code

11.1.1.1 package.json

```
{
  "name": "node-project",
  "version": "1.0.0",
  "description": "Complete Node.js Guide",
  "main": "sever.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon server.js",
    "start-server": "node server.js"
  },
  "author": "L. M. Saxton",
  "license": "ISC",
  "devDependencies": {
    "nodemon": "^3.1.0"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.20.2",
    "connect-flash": "^0.1.1",
    "connect-mongo": "^5.1.0",
    "csurf": "^1.11.0",
    "ejs": "^3.1.10",
    "express": "^4.19.2",
    "express-session": "^1.18.0",
    "express-validator": "^7.1.0",
    "mongodb": "^6.8.0",
    "mongoose": "^8.5.1",
    "sqlite3": "^5.1.7",
    "uuid": "^10.0.0"
  }
}
```

11.1.1.2 server.js:

```
// Node-specific dependencies
const http = require('http');
```

```

// Third-party packages
const bodyParser = require('body-parser');
const path = require('path');
const express = require('express');
const session = require('express-session');
const mongoose = require('mongoose');
const MongoDBStore = require('connect-mongo');
const csrf = require('csrf');
const flash = require('connect-flash');

const errorController = require('./controllers/errors');

const MONGODB_URI = 'mongodb://127.0.0.1:27017/shop'

const server = express();
const sessionStore = new MongoDBStore({
  mongoUrl: MONGODB_URI,
  collection: 'sessions',
  ttl: 24 * 60 * 60,
});

const csrfProtection = csrf();

server.set('view engine', 'ejs');
server.set('views', 'views');

// Own dependencies
const adminRoutes = require('./routes/admin');
const shopRoutes = require('./routes/shop');
const authRoutes = require('./routes/auth');

const User = require('./models/user');
const rootDir = require('./util/path')

// middleware
server.use(bodyParser.urlencoded({ extended: false })); // x-www-
form-urlencoded
server.use(express.static(path.join(rootDir, 'public')));
server.use(
  session({
    secret:
'98273yrvb9837wy4cv987wy4c987y984w7y4987yt49n87tya,kcxdpurvjf987es
yr78364n938h38gtb394ng3984thn3847gt',
    resave: false,
    saveUninitialized: false,
    store: sessionStore,
  })
);

```

```

    })
  );
  // initialise CSRF token
  server.use(csrfProtection);
  // initialise flash middleware
  server.use(flash());

  // user middleware
  server.use((request, response, next) => {
    if (!request.session.user) {
      return next();
    }
    // mongoose built-in function
    User.findById(request.session.user._id)
      .then(user => {
        // mongoose user model
        request.user = user;
        next();
      })
      .catch(err => {
        console.log('Error finding user', err);
        next(err);
      });
  });

  // set local variables passed into views
  server.use((request, response, next) => {
    // Authentication
    response.locals.isAuthenticated = request.session.isLoggedIn;
    // CSRF token
    response.locals.csrfToken = request.csrfToken();
    next()
  })

  server.use('/admin', adminRoutes);
  server.use(shopRoutes);
  server.use(authRoutes);

  server.get('/500', errorController.get500);

  server.use(errorController.get404);

  server.use((error, request, response, next) => {
    response.status(500).render('500', {
      pageTitle: 'Error - 500',
      path: '/500',
    });
  });

```

```
        isAuthenticated: request.session.isLoggedIn,
    });
});
```

mongoose

```
.connect('mongodb://127.0.0.1:27017/shop')
.then(result => {
  console.log('Database connected!')
  server.listen(3000);
  console.log('Server listening...');
})
.catch(err => {
  console.log(err);
})
```

11.1.2 Models

11.1.2.1 order.js:

```
const mongoose = require('mongoose')

const Schema = mongoose.Schema;

const orderSchema = new Schema({
  products: [{
    product: {
      type: Object,
      required: true,
    },
    quantity: {
      type: Number,
      required: true,
    }
  }],
  user: [{
    email: {
      type: String,
      required: true,
    },
    userId: {
      type: Schema.Types.ObjectId,
      required: true,
      ref: 'User'
    },
  }],
});

module.exports = mongoose.model('Order', orderSchema);
```

11.1.2.2 product.js

```
const mongoose = require('mongoose');

const Schema = mongoose.Schema;

// define a data schema
const productSchema = new Schema({
  // use key: value pairs w/ object type
  title: {
    type: String,
    required: true,
  }
});
```

```

    },
    author: {
      type: String,
      required: true,
    },
    price: {
      type: Number,
      required: true,
    },
    imageUrl: {
      type: String,
      required: true,
    },
    description: {
      type: String,
      required: true,
    },
    userId: {
      type: Schema.Types.ObjectId,
      ref: 'User',
      required: true,
    }
  });

module.exports = mongoose.model('Product', productSchema);

```

11.1.2.3 user.js

```

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const userSchema = new Schema({
  email: {
    type: String,
    required: true,
  },
  password: {
    type: String,
    required: true,
  },
  cart: {
    items: [
      {
        productId: {

```



```

        type: Schema.Types.ObjectId,
        ref: 'Product',
        required: true,
      },
      quantity: {
        type: Number,
        required: true,
      },
    }
  ]
},
});

// creating a function for mongoose
userSchema.methods.addToCart = function (product) {
  if (!this.cart.items) {
    this.cart.items = [];
    console.log('Cart Initialised!');
  } else {
    console.log('Cart Found!');
  }
  const cartProductIndex = this.cart.items.findIndex(cartProd =>
{
  return cartProd.productId.toString() ===
product._id.toString();
});
  let newQuantity = 1;
  const updatedCartItems = [...this.cart.items];
  if (cartProductIndex >= 0) {
    newQuantity = this.cart.items[cartProductIndex].quantity +
1;
    updatedCartItems[cartProductIndex].quantity = newQuantity
  } else {
    updatedCartItems.push({
      productId: product._id,
      quantity: newQuantity,
    });
  }
  const updatedCart = {
    items: updatedCartItems
  };
  this.cart = updatedCart;
  // built-in mongoose function
  return this.save();
};

```

```
userSchema.methods.removeFromCart = function (productId) {
  const updatedCartItems = this.cart.items.filter(item => {
    return item.productId.toString() !== productId.toString();
  });
  this.cart.items = updatedCartItems;
  return this.save();
};

userSchema.methods.clearCart = function () {
  this.cart = { items: [] };
  return this.save();
};

module.exports = mongoose.model('User', userSchema);
```

11.1.3 Controllers

11.1.3.1 admin.js:

```
const { validationResult } = require('express-validator')

// use capital for class name
const Product = require('../models/product');

// GET Add-Product Page
exports.getAddProduct = (request, response, next) => {
  response.render('admin/add-product', {
    pageTitle: 'Add Product',
    path: '/admin/add-product',
    addedProducts: '',
    hasError: false,
    productErrorMessage: null,
  });
};

// POST a new product through POST route
exports.postAddProduct = (request, response, next) => {
  const title = request.body.title;
  const author = request.body.author;
  const imageUrl = request.body.imageUrl;
  const price = request.body.price;
  const description = request.body.description;
  const errors = validationResult(request);

  if (!errors.isEmpty()) {
    console.log(errors.array());
    // if don't return, sends double headers.
    return response.status(422).render('admin/add-product', {
      pageTitle: 'Add Product',
      path: '/admin/add-product',
      editing: false,
      hasError: true,
      addedProducts: {
        title: title,
        author: author,
        imageUrl: imageUrl,
        price: price,
        description: description,
      },
    },
```

```

        productErrorMessage: 'Cannot submit an incomplete
product.'
    })
};

const addedProducts = new Product({
    title: title,
    author: author,
    imageUrl: imageUrl,
    price: price,
    description: description,
    userId: request.user._id,
});
addedProducts
    .save()
    .then(result => {
        console.log('PRODUCT ADDED');
        response.redirect('/admin/products');
    })
    .catch(err => {
        const error = new Error(err);
        error.httpStatusCode = 500;
        return next(error);
    })

};

exports.getProductList = (request, response, next) => {
    Product.find({userId: request.user._id})
        .then(products => {
            response.render('admin/products', {
                addedProducts: products,
                pageTitle: 'Admin Products',
                path: '/admin/products',
                editing: false,
                hasError: false,
            })
        })
        .catch(err => {
            const error = new Error(err);
            error.httpStatusCode = 500;
            return next(error);
        });
}

exports.deleteProduct = (request, response, next) => {

```

```

const productId = request.params.productId;
console.log(productId)
Product.findById(productId)
  .then(product => {
    if (!product) {
      return next(new Error('Product not found'));
    }
    // function built-in to mongoose
    return Product.deleteOne({
      _id: productId,
      userId: request.user._id
    });
  })
  .then(() => {
    console.log('PRODUCT DELETED')
    // ASYNC js views request => won't render a new page,
just return new data
    response.status(200).json({
      message: 'Success!'
    });
  })
  .catch(err => {
    response.status(500).json('Deleting product failed.');
```

11.1.3.2 auth.js:

```

const bcrypt = require('bcryptjs');
const { validationResult } = require('express-validator');

const User = require('../models/user');

exports.getLogin = (request, response, next) => {
  let loginMessage = request.flash('loginError')
  if (loginMessage.length > 0) {
    loginMsg = loginMessage[0];
  } else {
    loginMsg = null;
  }
  return response.status(422).render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: 'Invalid email or password',
```

```

        loginErrorMessage: loginMsg,
        oldInput: {
            email: '',
            password: '',
        },
    });
};

exports.postLogin = (request, response, next) => {
    const email = request.body.email;
    const password = request.body.password;

    User.findOne({email: email})
        .then(user => {
            if (!user) {
                request.flash('loginError', 'Invalid email or
password.');
```

```

                return response.status(422).render('auth/login', {
                    path: '/login',
                    pageTitle: 'Login',
                    loginErrorMessage: 'Invalid email or
password',

                    oldInput: {
                        email: email,
                        password: password,
                    },
                });
            }
            bcrypt
                .compare(password, user.password)
                .then(doMatch => {
                    if (doMatch) {
                        console.log('Login successful!')
                        request.session.isLoggedIn = true;
                        request.session.user = user;
                        return request.session.save((err) => {
                            console.log(err);
                            response.redirect('/');
                        });
                    } else {
                        request.flash('loginError', 'Invalid email
or password.');
```

```

                        return
response.status(422).render('auth/login', {
                    path: '/login',
                    pageTitle: 'Login',

```

```

        loginErrorMessage: 'Invalid email or
password',
        oldInput: {
            email: email,
            password: password,
        },
    });
    });
    .catch(err => {
        console.log(err);
        response.redirect('/login')
    });
})
};

exports.postLogout = (request, response, next) => {
    request.session.destroy(() => {
        response.redirect('/');
    });
};

exports.getCreateUser = (request, response, next) => {
    let emailMessage = request.flash('emailError');
    if (emailMessage.length > 0) {
        emailMsg = emailMessage[0];
    } else {
        emailMsg = null;
    }

    let passwordMessage = request.flash('passwordError')
    if (passwordMessage.length > 0) {
        passwordMsg = passwordMessage[0];
    } else {
        passwordMsg = null;
    }

    response.render('auth/create-user', {
        pageTitle: 'Create an Account',
        path: '/signup',
        emailErrorMessage: emailMessage,
        passwordErrorMessage: passwordMessage,
        validationErrorMessage: null,
        oldInput: {
            email: "",
            password: "",

```

```

        confirmPassword: "",
    }

    });
};

exports.postCreateUser = (request, response, next) => {
    const email = request.body.email;
    const password = request.body.password;
    const confirmPassword = request.body.confirmPassword;
    const errors = validationResult(request);

    if (!errors.isEmpty()) {
        console.log(errors.array());
        // common code for failed validation
        return response.status(422).render('auth/create-user', {
            path: '/signup',
            pageTitle: 'Create an Account',
            validationErrorMessage: errors.array()[0].msg,
            oldInput: {
                email: email,
                password: password,
                confirmPassword: request.body.confirmPassword,
            },
            // return the first error message
            emailErrorMessage: null,
            passwordErrorMessage: null,
        });
    };

    bcrypt
        .hash(password, 12)
        .then(hashedPassword => { // chain the .then block
            if (password === confirmPassword) {
                const user = new User({
                    email: email,
                    password: hashedPassword,
                    cart: { items: [] },
                })
                return user.save();
            } else {
                request.flash('passwordError', 'Passwords do
not match; please try again.');
```



```

        .then(result => {
            response.redirect('/login');
        })
        .catch(err => {
            console.log(err);
        });
};

```

11.1.3.3 errors.js:

```

// throws a 404 error
exports.get404 = (request, response, next) => {
    response.status(404).render('404', {
        pageTitle: 'Page Not Found',
        path: request.path,
    });
}

exports.get500 = (request, response, next) => {
    response.status(500).render('500', {
        pageTitle: 'Error - 500; Internal Error',
        path: '/500',
    })
}

```

11.1.3.4 shop.js:

```

// use capital for class name
const Product = require('../models/product');
const Order = require('../models/order');

// GET products on home page
exports.getProducts = (request, response, next) => {
    Product.find()
        .then(products => {
            response.render('shop/product-list', {
                addedProducts: products,
                pageTitle: 'Available Products',
                path: '/products',
            });
        });
};

exports.getProduct = (request, response, next) => {

```

```

const productId = request.params.productId;
// findById == mongoose built in method
Product.findById(productId)
  .then(product => {
    response.render('shop/product-detail', {
      pageTitle: product.title,
      product: product,
      path: '/products',
    });
  })
  .catch(err => console.log(err));
};

exports.getIndex = (request, response, next) => {
  Product.find()
    .then((products) => {
      response.render('shop/index', {
        pageTitle: 'The Book Shop',
        addedProducts: products,
        path: '/',
      });
    })
    .catch(err => {
      console.log(err);
    })
};

exports.getCart = (request, response, next) => {
  request.user
    .populate('cart.items.productId')
    .then(user => {
      const products = user.cart.items;
      response.render('shop/cart', {
        path: '/cart',
        pageTitle: 'Your Cart',
        addedProducts: products,
      });
    })
    .catch(err => console.log(err));
};

exports.postCart = (request, response, next) => {
  // .productId is the name used in the view
  const productId = request.body.productId
  // mongoose built-in function
  Product.findById(productId)

```

```

        .then(product => {
            return request.user.addToCart(product);
        })
        .then(result => {
            response.redirect('./cart');
        })
        .catch(err => {
            console.log(err);
        });
};

exports.postCartDeleteProduct = (request, response, next) => {
    const productId = request.body.productId;
    request.user
        .removeFromCart(productId)
        .then(result => {
            response.redirect('./cart');
        })
        .catch(err => {
            console.log(err);
        });
};

exports.postOrder = (request, response, next) => {
    // get the products in the user's cart
    request.user
        .populate('cart.items.productId')
        .then(user => {
            console.log(user.cart.items);
            // map out the products for later use
            const products = user.cart.items.map(item => {
                // wrap productId in {...x._doc } to access the
data we want
                return { quantity: item.quantity, product:
{ ...item.productId._doc } };
            });
            // initialise a new order
            const order = new Order({
                user: {
                    email: request.user.email,
                    userId: request.user,
                },
                products: products,
            });
            return order.save();
        })
};

```

```

    .then(result => {
      return request.user.clearCart();
    })
    .then(() => {
      response.redirect('./orders');
    })
    .catch(err => {
      console.log(err);
      next(err);
    });
  });
}

```

```

exports.getOrders = (request, response, next) => {
  Order
    .find({
      "user.userId": request.user._id,
    })
    .then(orders => {
      response.render('shop/orders', {
        path: '/orders',
        pageTitle: 'Your Orders',
        orders: orders,
      });
    })
    .catch(err => {
      console.log(err);
    });
};

```

11.1.4 Views

11.1.4.1 404.ejs:

```
<%- include('includes/head.ejs') %>
<link rel="stylesheet" href="/css/product.css">
<link rel="stylesheet" href="/css/form.css">
</head>

<body>

  <%- include('includes/navigation.ejs') %>

  <main>
    <h1 class="centered">Error - 404</h1>
    <hr>
    <br>
    <h1>Page not found!</h1>

  </main>

<%- include('includes/end.ejs') %>
```

11.1.4.2 500.ejs:

```
<%- include('includes/head.ejs') %>
<link rel="stylesheet" href="/css/product.css">
<link rel="stylesheet" href="/css/form.css">
</head>

<body>

  <%- include('includes/navigation.ejs') %>

  <main>
    <h1 class="centered">Error - 500</h1>
    <hr>
    <br>
    <h1>SERVER ERROR</h1>

  </main>

<%- include('includes/end.ejs') %>
```

11.1.4.3 Admin

11.1.4.3.1 add-product.ejs:

```
<%- include('../includes/head.ejs') %>

<link rel="stylesheet" href="/css/product.css">
<link rel="stylesheet" href="/css/form.css">
</head>

<body>

  <%- include('../includes/navigation.ejs') %>

  <main>
    <h1 class="centered">Add A Book</h1>
    <% if (productErrorMessage) { %>
      <div class="centered"><%= productErrorMessage %></div>
      <br>
    <% } %>
    <br>
    <form class="product-form" action="/admin/add-product"
method="POST">
      <div class="form-control">
        <label for="title">Title</label>
        <input type="text" name="title" id="title"
value="<%= addedProducts.title %>">
      </div>
      <div class="form-control">
        <label for="author">Author</label>
        <input type="text" name="author" id="author"
value="<%= addedProducts.author %>">
      </div>
      <div class="form-control">
        <label for="imageUrl">Image URL</label>
        <input type="text" name="imageUrl" id="imageUrl"
value="<%= addedProducts.imageUrl %>">
      </div>
      <div class="form-control">
        <label for="title">Price</label>
        <input type="number" name="price" id="price"
step="0.01" value="<%= addedProducts.price %>">
      </div>
      <div class="form-control">
        <label for="description">Description</label>
```

```

        <textarea name="description" id="description"
rows="5" value="<%= addedProducts.description %>"></textarea>
    </div>
    <div>
        <input type="hidden" name="_csrf" value="<%=
csrfToken%>">
        <button type="submit">Add Product</button>
    </div>
</form>

```

```
</main>
```

```
<%- include('../includes/end.ejs') %>
```

11.1.4.3.2 edit-product.ejs:

```
<%- include('../includes/head.ejs') %>
```

```
<link rel="stylesheet" href="/css/product.css">
```

```
<link rel="stylesheet" href="/css/form.css">
```

```
</head>
```

```
<body>
```

```
<%- include('../includes/navigation.ejs') %>
```

```
<main>
```

```
<h1 class="centered">Add A Book</h1>
```

```
<% if (productErrorMessage) { %>
```

```
<div class="centered"><%= productErrorMessage %></div>
```

```
<br>
```

```
<% } %>
```

```
<br>
```

```
<form class="product-form" action="/admin/edit-product"
method="POST">
```

```
<div class="form-control">
```

```
<label for="title">Title</label>
```

```
<input type="text" name="title" id="title"
```

```
value="<%= product.title %>">
```

```
</div>
```

```
<div class="form-control">
```

```
<label for="author">Author</label>
```

```
<input type="text" name="author" id="author"
```

```
value="<%= product.author %>">
```

```
</div>
```

```
<div class="form-control">
```

```

        <label for="imageUrl">Image URL</label>
        <input type="text" name="imageUrl" id="imageUrl"
value="<%= product.imageUrl %>">
    </div>
    <div class="form-control">
        <label for="title">Price</label>
        <input type="number" name="price" id="price"
step="0.01" value="<%= product.price %>">
    </div>
    <div class="form-control">
        <label for="description">Description</label>
        <textarea name="description" id="description"
rows="5" value="<%= product.description %>"></textarea>
    </div>
    <div>
        <input type="hidden" name="_csrf" value="<%=
csrfToken%>">
        <button type="submit">Add Product</button>
    </div>
</form>

</main>

<%- include('../includes/end.ejs') %>

```

11.1.4.3.3 products.ejs:

```

<%- include('../includes/head.ejs') %>
<link rel="stylesheet" href="/css/product.css">
<link rel="stylesheet" href="/css/form.css">
</head>

<body>

    <%- include('../includes/navigation.ejs') %>

    <main>
        <h1 class="centered">Book List</h1>
        <hr>
        <% if (addedProducts.length > 0) { %>
        <div class="grid">
            <% for (let product of addedProducts) { %>
                <article class="card product-item">
                    <header>
                        <h1 class="product__title"><%=
product.title %></h1>

```



```

        </header>
        <div class="card__image">
            ">
        </div>
        <div class="card__content">
            <h4 class="product__author"><%=
product.author %></h4>
            <h2 class="product__price">$<%=
product.price %></h2>
            <p class="product__description"><%=
product.description %></p>
        </div>
        <div class="card__actions">
            <input type="hidden" name="_csrf" value="<
%= csrfToken %>">
            <input type="hidden" value="<%=
product._id %>" name="productId">
            <button class = 'btn' type="button"
onclick="deleteProduct(this)">Delete</button>
        </div>
        <br>
    </article>
    <% } %>
</div>
<% } else { %>
    <h1>No Products Found!</h1>
<% } %>
</main>

<%- include('../includes/end.ejs') %>
<!-- Put the js on the bottom -->
<script src="/js/admin.js"></script>

```

11.1.4.4 Auth

11.1.4.4.1 create-user.ejs:

```
<%- include('../includes/head.ejs') %>

<link rel="stylesheet" href="/css/product.css">
<link rel="stylesheet" href="/css/form.css">
</head>

<body>

  <%- include('../includes/navigation.ejs') %>

  <main>
    <h1 class="centered">Create An Account</h1>
    <br>
    <% if (validationErrorMessage) { %>
      <div class="centered"><%= validationErrorMessage
%></div>
    <% } %>
    <% if (emailErrorMessage) { %>
      <div class="centered"><%= emailErrorMessage %></div>
    <% } %>
    <% if (passwordErrorMessage) { %>
      <div class="centered"><%= passwordErrorMessage
%></div>
    <% } %>
    <form class="product-form" action="/signup" method="POST">
      <div class="form-control">
        <label for="email">Email</label>
        <input type="text" name="email" id="email"
value="<%= oldInput.email %>">
      </div>
      <div class="form-control">
        <label for="password">Password</label>
        <input type="password" name="password"
id="password" value="<%= oldInput.password %>">
      </div>
      <div class="form-control">
        <label for="confirmPassword">Confirm
Password</label>
        <input type="password" name="confirmPassword"
id="confirmPassword" value="<%= oldInput.confirmPassword %>">
      </div>
    </div>
  </main>

```

```

        <input type="hidden" name="_csrf" value="<%=
csrfToken %>">
        <button type="submit">Signup</button>
    </div>
</form>

</main>

<%- include('../includes/end.ejs') %></body>

```

11.1.4.4.2 login.ejs:

```

<%- include('../includes/head.ejs') %>

<link rel="stylesheet" href="/css/product.css">
<link rel="stylesheet" href="/css/form.css">
</head>

<body>

    <%- include('../includes/navigation.ejs') %>

    <main>
        <h1 class="centered">User Login</h1>
        <br>
        <% if (loginErrorMessage) { %>
            <div class="centered"><%= loginErrorMessage %></div>
            <br>
        <% } %>
        <form class="product-form" action="/login" method="POST">
            <div class="form-control">
                <label for="email">Email</label>
                <input type="text" name="email" id="email"
value="<%= oldInput.email %>">
            </div>
            <div class="form-control">
                <label for="password">Password</label>
                <input type="password" name="password"
id="password" value="<%= oldInput.password %>">
            </div>
            <input type="hidden" name="_csrf" value="<%= csrfToken
%>">
            <button type="submit">Login</button>
        </form>

    </main>

```

```
<%- include('../includes/end.ejs') %>
```

11.1.4.5 Includes

11.1.4.5.1 add-to-cart.ejs:

```
<form action="/cart" method="POST">
  <input type="hidden" name="_csrf" value="<%= csrfToken %>">
  <button type="submit">Add to Cart</button>
  <input type="hidden" name="productId" value="<%= product._id
%>">
</form>
```

11.1.4.5.2 end.ejs:

```
</body>
```

```
</html>
```

11.1.4.5.3 head.ejs:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <meta http-equiv="X-US-Compatible" content="ie=edge">
  <title><%= pageTitle %></title>
  <link rel="stylesheet" href="/css/main.css">
```

11.1.4.5.4 navigation.ejs:

```
<header class="main-header">
  <nav class="main-header__nav">
    <ul class="main-header__item-list">

      <li class="main-header__item"><a class="<%= path
=== '/' ? 'active' : '' %>" href="/">Shop</a></li>
      <li class="main-header__item"><a class="<%= path
=== '/products' ? 'active' : '' %>"
href="/products">Products</a></li>

    <% if (isAuthenticated) { %>
```

```

        <li class="main-header__item"><a class="<%= path
=== '/cart' ? 'active' : '' %>" href="/cart">Cart</a></li>
        <li class="main-header__item"><a class="<%= path
=== '/orders' ? 'active' : '' %>" href="/orders">Orders</a></li>

        <li class="main-header__item"><a class="<%= path
=== '/admin/add-product' ? 'active' : '' %>" href="/admin/add-
product">Add Product</a></li>
        <li class="main-header__item"><a class="<%= path
=== '/admin/products' ? 'active' : '' %>"
href="/admin/products">My Products</a></li>

    </ul>
    <form class="main-header__item" action="/logout"
method="POST">
        <input type="hidden" name="_csrf" value="<%=
csrfToken %>">
        <button type="submit">Logout</button>
    </form>
    <% } %>
</nav>
</header>

```

11.1.4.6 Shop

11.1.4.6.1 cart.ejs

```
<%- include('../includes/head.ejs') %>
<link rel="stylesheet" href="/css/product.css">
<link rel="stylesheet" href="/css/form.css">
</head>

<body>

  <%- include('../includes/navigation.ejs') %>
  <main>
    <h1 class="centered">Your Cart</h1>
    <hr>
    <% if (addedProducts.length > 0) { %>
      <ul>
        <% addedProducts.forEach(productDisplay => { %>
          <li>
            <h1>
              <%= productDisplay.productId.title %>
            </h1>
            <h2>
              Quantity: <%= productDisplay.quantity
%>
            </h2>
            <form action="/cart-delete-item"
method="POST">
              <input type="hidden" name="_csrf"
value="<%= csrfToken %>">
              <input type="hidden" value="<%=
productDisplay.productId._id %>" name="productId">
              <button type="submit">Delete</button>
            </form>
            <br>
          </li>
        <% }) %>
      </ul>
      <hr>
      <br>
      <div class="centered">
        <form action="/create-order" method="POST">
          <input type="hidden" name="_csrf" value="<%=
csrfToken %>">
          <button type="submit">Order Now!</button>
        </form>
```

```

        </div>

    <%> else { %>
        <br>
        <h1>No Products in Cart!</h1>
    <% } %>
</main>
<%- include('../includes/end.ejs') %>

```

11.1.4.6.2 index.ejs:

```

<%- include('../includes/head.ejs') %>
<link rel="stylesheet" href="/css/product.css">
<link rel="stylesheet" href="/css/form.css">
</head>

<body>

    <%- include('../includes/navigation.ejs') %>

    <main>
        <h1 class="centered">The Book Shop</h1>
        <% if (isAuthenticated) { %>
            <h2>Welcome back!</h2>
        <% } else { %>
            <div class = "grid">
                <p>Have an account? <a href="/login"
method="GET">Login</a></p>
                <p>First time? <a href="/signup" method="GET"
>Create an account</a></p>
            </div>
        <% } %>
        <hr>
        <h1 class="centered">Available Books</h1>
        <% if (addedProducts.length > 0) { %>
            <div class="grid">
                <% for (let product of addedProducts) { %>
                    <article class="card product-item">
                        <header>
                            <h1 class="product__title"><%=
product.title %></h1>
                        </header>
                        <div class="card__image">
                            ">
                        </div>
                <% } %>
            </div>
        <% } %>
    </main>

```



```

        <div class="card__content">
            <h4 class="product__author"><%=
product.author %></h4>
            <h2 class="product__price">$<%=
product.price %></h2>
            <p class="product__description"><%=
product.description %>.</p>
        </div>
        <div class="card__actions">
            <% if (isAuthenticated) { %>
                <%- include('../includes/add-to-
cart.ejs', {product: product}) %>
            <% } %>
        </div>
        <br>
    </article>
    <% } %>
</div>
<% } else { %>
    <h1>No Products Found!</h1>
<% } %>
</main>

<%- include('../includes/end.ejs') %>

```

11.1.4.6.3 orders.ejs:

```

<%- include('../includes/head.ejs') %>
<link rel="stylesheet" href="/css/product.css">
<link rel="stylesheet" href="/css/form.css">
</head>

<body>

    <%- include('../includes/navigation.ejs') %>
    <main>
        <h1 class="centered">Your Orders</h1>
        <hr>
        <br>
        <% if (orders.length <= 0) { %>
            <h1>No Orders!</h1>
        <% } else { %>
            <ul class="orders">
                <% orders.forEach(order => { %>
                    <li class="orders__item">
                        <h1>Order # <%= order._id %></h1>
                <% } %>
            </ul>
        <% } %>
    </main>

```

```

        <ul class="orders__products">
          <% order.products.forEach(product => { %>
            <li class="orders__products-item">
              <h1>
                <%= product.product.title %>
              </h1>
              <h2>
                Price: $<%=
product.product.price %>
              <br>
                Quantity: <%= product.quantity
%>
              </h2>
            </li>
          <% }) %>
          <hr>
          <br>
        </ul>
      </li>
    <% }) %>
  </ul>
</main>
<%- include('../includes/end.ejs') %>

```

11.1.4.6.4 product-detail.ejs:

```

<%- include('../includes/head.ejs') %>
<link rel="stylesheet" href="/css/product.css">
<link rel="stylesheet" href="/css/form.css">
</head>

<body>

  <%- include('../includes/navigation.ejs') %>
  <main class="centered">
    <% if (isAuthenticated) { %>
      <h1><%= product.title %></h1>
    <% } else { %>
      <h1><%= product.title %></h1>
      <div class = "left">
        <p>Have an account? <a href="/login"
method="GET">Login</a></p>
        <p>First time? <a href="/signup" method="GET"
>Create an account</a></p>

```

```

        </div>
    <% } %>
    <hr>
    <div>
        ">
    </div>
    <h2><%= product.price %></h2>
    <p><%= product.description %></p>
    <% if (isAuthenticated) { %>
        <%- include('../includes/add-to-cart.ejs', {product:
product}) %>
    <% } %>
</main>
<%- include('../includes/end.ejs', {product: product}) %>

```

11.1.4.6.5 product-list.ejs:

```

<%- include('../includes/head.ejs') %>
<link rel="stylesheet" href="/css/product.css">
<link rel="stylesheet" href="/css/form.css">
</head>

<body>

    <%- include('../includes/navigation.ejs') %>

    <main>
        <% if (isAuthenticated) { %>
            <h1 class="centered">Available Books</h1>
        <% } else { %>
            <h1 class="centered">Available Books</h1>
            <div class = "grid">
                <p>Have an account? <a href="/login"
method="GET">Login</a></p>
                <p>First time? <a href="/signup" method="GET"
>Create an account</a></p>
            </div>
        <% } %>
        <hr>
        <% if (addedProducts.length > 0) { %>
        <div class="centered">
            <br>
            <% for (let product of addedProducts) { %>
                <article class="card product-item">
                    <header>

```

```

        <h1 class="product__title"><%=
product.title %></h1>
        </header>
        <div class="card__image">
            ">
        </div>
        <div class="card__content">
            <h4 class="product__author"><%=
product.author %></h4>
            <h2 class="product__price">$<%=
product.price %></h2>
            <p class="product__description"><%=
product.description %>.</p>
            </div>
            <div class="card__actions">
                <input type="hidden" name="_csrf" value="<
%= csrfToken %>">
                <form action="/products/<%= product._id
%>" method="GET">
                    <button type="submit">Details</button>
                </form>
                <% if (isAuthenticated) { %>
                    <%- include('../includes/add-to-
cart.ejs', {product: product}) %>
                    <% } %>
                </div>
                <br>
                <hr>
            </article>
            <% } %>
        </div>
        <% } else { %>
            <h1>No Products Found!</h1>
        <% } %>
    </main>

<%- include('../includes/end.ejs') %>

```

11.1.5 Public

11.1.5.1 CSS

11.1.5.1.1 form.css:

```
.form-control {
    margin: 1rem 0;
}

.form-control label,
.form-control input,
.formcontrol textarea {
    display: block;
    width: 100%;
}

.form-control input,
.formcontrol textarea {
    border: 1px solid #26520d;
    font: inherit;
    border-radius: 2px;
}

.form-control input:focus,
.form-control textarea:focus {
    outline-color: #00695c;
}

button {
    font: inherit;
    border: 1px solid #26520d;
    color: #26520d;
    background: white;
    border-radius: 3px;
    cursor: pointer;
}

button:hover,
button:active {
    background-color: #26520d;
    color: white;
}
```

11.1.5.1.2 main.css:

```
body {
  margin: 0;
  padding: 0;
  font-family: sans-serif;
}

main {
  padding: 1rem;
}

.main-header {
  width: 100%;
  height: 3.5rem;
  background-color: #26520d;
  padding: 0 1.5rem;
}

.main-header__nav {
  height: 100%;
  display: flex;
  align-items: center;
  justify-content: space-between;
}

.main-header__item-list {
  list-style: none;
  margin: 0;
  padding: 0;
  display: flex;
}

.main-header__item {
  margin: 0 1rem;
  padding: 0;
}

.main-header__item a {
  text-decoration: none;
  color: white;
}

.main-header__item a:hover,
.main-header__item a:active,
```

```
.main-header__item a.active {  
    color: #d6cb2c;  
}
```

```
.centered {  
    text-align: center;  
}
```

```
.left {  
    text-align: left;  
}
```

11.1.5.1.3 product.css:

```
.product-form {  
    width: 20rem;  
    max-width: 90%;  
    margin: auto;  
}
```

11.1.5.2 JavaScript

11.1.5.2.1 admin.js:

```
// executes on the client side views with <script>

const deleteProduct = (btn) => {
  // extract the product ID value
  const productId =
btn.parentNode.querySelector('[name=productId]').value;
  // extract the CSRF value
  const csrf =
btn.parentNode.querySelector('[name=_csrf]').value;

  const productElement = btn.closest('article');
  // method supported by the browser for fetching/sending HTTP
requests
  fetch('/admin/product/' + productId, {
    method: 'DELETE',
    headers: {
      'csrf-token': csrf,
    }
  })
  .then(result => {
    return result.json();
  })
  .then(data => {
    console.log(data);
    productElement.parentNode.removeChild(productElement);
  })
  .catch(err => {
    console.log(err)
  });
};
```


11.1.6 Routes

11.1.6.1 admin.js

```
const path = require('path');
const express = require('express');
const { check, body } = require('express-validator');

// Own dependencies
const rootDir = require('../util/path');
const adminController = require('../controllers/admin');
const isAuth = require('../middleware/is-auth');

const router = express.Router();

// /admin/add-product => GET
router.get('/add-product', isAuth, adminController.getAddProduct);

// // /admin/products => GET
router.get('/products', isAuth, adminController.getProductList)

// /admin/add-product => POST
router.post('/add-product', [
  body('title')
    .isString()
    .isLength({ min: 3 })
    .trim(),
  body('author')
    .isString()
    .isLength({ min: 3 })
    .trim(),
  body('imageUrl')
    .isURL(),
  body('price')
    .isFloat(),
  body('description')
    .isLength({ max: 40 })
], isAuth, adminController.postAddProduct);

// /admin/product/productId => DELETE
router.delete('/product/:productId', isAuth,
adminController.deleteProduct);
```

```
module.exports = router;
```

11.1.6.2 auth.js

```
const express = require('express');
// import check function from package
const { check, body } = require('express-validator');

const authController = require('../controllers/auth');
const isAuth = require('../middleware/is-auth');
const User = require('../models/user');

const router = express.Router();

// /login => GET
router.get('/login', authController.getLogin);

// /log => POST
router.post(
  '/login',
  [
    check('email')
      .isEmail()
      .withMessage('Please enter a valid email.')
      .normalizeEmail(),

    body('password', 'Invalid password.')
      .isLength({ min: 8 })
      .isAlphanumeric()
      .trim(),
  ],
  authController.postLogin);

// /logout => POST
router.post('/logout', check(), isAuth, authController.postLogout);

// /signup => GET
router.get('/signup', authController.createUser);

// /signup => POST
router.post(
  '/signup',
  [
```

```

        check('email')
            .isEmail()
            .withMessage('Please enter a valid email.')
            .custom((value, { request }) => {
                // Async validation
                return User.findOne({ email: value })
                    .then(userDoc => {
                        // validation rejection
                        if (userDoc) {
                            return Promise.reject('Email
already exists.')
                        }
                    })
            })
            .normalizeEmail(),
        body('password')
            .isLength({ min: 8 })
            .withMessage('Password needs to be at least 8
characters.')
            .isAlphanumeric()
            .withMessage('Password can only contain letters
and numbers.')
            .trim(),
        body('confirmPassword')
            .isLength({ min: 8 })
            .withMessage('Password needs to be at least 8
characters.')
            .isAlphanumeric()
            .withMessage('Password can only contain letters
and numbers.')
            .trim(),
    ],
    authController.postCreateUser
);

module.exports = router;

```

11.1.6.3 shop.js

```

const path = require('path');
const express = require('express');

const shopController = require('../controllers/shop');
const isAuth = require('../middleware/is-auth');

```

```
const router = express.Router();

// /index => GET
router.get('/', shopController.getIndex);

// /products => GET
router.get('/products', shopController.getProducts);

// /products/productId => GET
router.get('/products/:productId', shopController.getProduct);

// /cart => GET
router.get('/cart', isAuthenticated, shopController.getCart);

// /cart => POST
router.post('/cart', isAuthenticated, shopController.postCart);

// /cart-delete-item => POST
router.post('/cart-delete-item', isAuthenticated,
shopController.postCartDeleteProduct);

// /orders => GET
router.get('/orders', isAuthenticated, shopController.getOrders);

// /create-order => POST
router.post('/create-order', isAuthenticated, shopController.postOrder);

// /checkout => GET
// router.get('/checkout', shopController.getCheckout);

// /checkout => POST
// router.post('/checkout', shopController.postCheckout);

module.exports = router;
```

11.1.7 Middleware

11.1.7.1 is-auth.js:

```
module.exports = (request, response, next) => {  
  if (!request.session.isLoggedIn) {  
    return response.redirect('/login');  
  }  
  next();  
};
```

11.1.8 Util

11.1.8.1 path.js:

```
const path = require('path');  
module.exports = path.dirname(require.main.filename);
```

11.2 Appendix II: REST Backend Source Code

11.2.1 Main Program Code

11.2.1.1 package.json

```
{
  "name": "rest-api",
  "version": "1.0.0",
  "description": "Complete Node.js Guide",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon app.js",
    "start-server": "app server.js"
  },
  "author": "L. M. Saxton",
  "license": "ISC",
  "devDependencies": {
    "nodemon": "^3.1.0"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.20.2",
    "connect-flash": "^0.1.1",
    "connect-mongo": "^5.1.0",
    "csurf": "^1.11.0",
    "ejs": "^3.1.10",
    "express": "^4.19.2",
    "express-session": "^1.18.0",
    "express-validator": "^7.1.0",
    "jsonwebtoken": "^9.0.2",
    "mongodb": "^6.8.0",
    "mongoose": "^8.5.1",
    "multer": "^1.4.5-lts.1",
    "sqlite3": "^5.1.7",
    "uuid": "^10.0.0"
  }
}
```

11.2.1.2 app.js

```
// Node Dependencies
const path = require('path');
```

```

// Third Party Dependencies
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

// Own dependencies
const feedRoutes = require('./routes/feed');
const authRoutes = require('./routes/auth');

const app = express();

// Server Middleware
app.use(express.json()); // POSTMAN doesn't work with this
included
// image upload service

app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*'); // allows
access to every client
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT,
PATCH, DELETE'); // allows the origins to use spec HTTP methods
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type,
Authorization'); // Which headers are always allowed || content
types allowed in request / allows extra authentication data from
clients
  // moves response ahead
  next();
});

// Routes middleware
app.use('/api', feedRoutes);
app.use('/api/auth', authRoutes);

// Error middleware
app.use((error, req, res, next) => {
  console.log(error);
  const status = error.statusCode || 500;
  const message = error.message;
  const data = error.data;
  res.status(status).json({
    message: message,
    data: data,
  });
})

mongoose.connect('mongodb://127.0.0.1:27017/rest')

```



```
.then(result => {
  console.log('Database connected...');
  app.listen(4000);
  console.log('Server listening...')
})
.catch(err => {
  console.log('DATABASE CONNECTION FAILED!');
  console.log(err);
});
```

11.2.2 Models

11.2.2.1 post.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const postSchema = new Schema({
  title: {
    type: String,
    required: true,
  },
  content: {
    type: String,
    required: true,
  },
  creator: {
    type: Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
},
{ timestamps: true }
);

module.exports = mongoose.model('Post', postSchema);
```

11.2.2.2 user.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const userSchema = new Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
  },
  password: {
    type: String,
    required: true
  }
});
```

```
    },
    status: {
      type: String,
      default: 'I am new!'
    },
    posts: [{
      // store schema / reference for posts
      type: Schema.Types.ObjectId,
      ref: 'Post',
    }]
  });

module.exports = mongoose.model('User', userSchema);
```

11.2.3 Controllers

11.2.3.1 auth.js

```
const { validationResult } = require('express-validator');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

const User = require('../models/user');

exports.putSignup = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    const error = new Error('Validation failed!');
    error.statusCode = 422;
    error.data = errors.array();
    throw error;
  };

  const name = req.body.name;
  const email = req.body.email;
  const password = req.body.password;

  // hash the password
  bcrypt.hash(password, 12)
    .then(hashPw => {
      const user = new User({
        name: name,
        email: email,
        password: hashPw,
      });
      // save user object to db
      return user.save();
    })
    .then(result => {
      res.status(201).json({
        message: 'User created!',
        userId: result._id,
      });
    })
    .catch(err => {
      if (!err.statusCode) {
        err.statusCode = 500;
      }
      next(err);
    });
};
```

```

    });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  let loadedUser;
  User.findOne({ email: email })
    .then(user => {
      if (!user) {
        const error = new Error('Invalid credentials!');
        // not authenticated = 404
        error.statusCode = 404;
        throw error;
      };
      loadedUser = user;
      return bcrypt.compare(password, user.password);
    })
    .then(isEqual => {
      if (!isEqual) {
        const error = new Error('Invalid credentials!');
        error.statusCode = 404;
        throw error;
      };
      const token = jwt.sign({
        email: loadedUser.email,
        userId: loadedUser._id.toString(),
      },
        // secret string, ASCII 64 bit
        'f42952d5d0f2e342c4b6ebe80ab6cc968d8bcc7020ff1c7a5a8b057c790039e59
8df323029cbac2993e7d6742b6362c3ca5cad0279925c43b288c9419e6122bb',
        // token expiration
        { expiresIn: '75h' }
      );
      res.status(200).json({
        token: token,
        userId: loadedUser._id.toString(),
      })
    })
    .catch(err => {
      if (!err.statusCode) {
        err.statusCode = 500;
      }
      next(err);
    })
};

```

```
};
```

11.2.3.2 feed.js

```
// Third Party Dependencies
const { validationResult } = require('express-validator');
const fs = require('fs');
const path = require('path');

// Own dependencies
const Post = require('../models/post');
const User = require('../models/user');

exports.getPosts = (req, res, next) => {
  Post.find()
    .then(posts => {
      // send back JSON as a response
      // status codes are very important for APIs
      res.status(200).json({
        posts: posts,
      })
    })
    .catch(err => {
      if (!err.statusCode) {
        err.statusCode = 500;
      }
      next(err);
    })
};

// naming convention: HTTP method + object
exports.postPost = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    const error = new Error('Validation failed!');
    console.log(error);
    // 422 => validation failed
    error.statusCode = 422;
    throw error;
  }

  const title = req.body.title;
  const content = req.body.content;
  let creator;
```

```

// create post in db
const post = new Post({
  title: title,
  content: content,
  creator: req.userId,
});
// save model to db
post
  .save()
  .then(result => {
    return User.findById(req.userId);
  })
  .then(user => {
    creator = user;
    // push the post object to the user
    user.posts.push(post);
    user.save();
  })
  .then(result => {
    // 201 = successful created resource
    res.status(201).json({
      post: post,
      creator: {
        _id: creator._id,
        name: creator.name,
      }
    });
  })
  .catch(err => {
    if (!err.statusCode) {
      err.statusCode = 500;
    }
    next(err);
  });
};

exports.getPost = (req, res, next) => {
  const postId = req.params.postId;
  Post.findById(postId)
    .then(post => {
      if (!post) {
        const error = new Error('POST NOT FOUND!');
        error.statusCode = 404;
        // passes error to catch block
        throw error;
      }
    });
};

```

```

        }
        res.status(200).json({
            post: post,
        })
    })
    .catch(err => {
        if (!err.statusCode) {
            err.statusCode = 500;
        }
        next(err);
    });
}

exports.putPost = (req, res, next) => {
    const postId = req.params.postId;
    const errors = validationResult(req);

    if (!errors.isEmpty()) {
        const error = new Error('VALIDATION FAILED');
        console.log(error)
        // 422 => validation failed
        error.statusCode = 422
        throw error;
    };

    const title = req.body.title;
    const content = req.body.content;

    Post.findById(postId)
        .then(post => {
            if (!post) {
                const error = new Error('NO POST FOUND');
                error.statusCode = 404;
                throw error;
            }
            // make sure the user ID matches the JWT
            if (post.creator.toString() !== req.userId) {
                const error = new Error('NOT AUTHORISED!');
                error.statusCode = 403;
                throw error;
            }
            // overwrite / save updated post
            post.title = title;
            post.content = content;
            return post.save();
        })
}

```



```

    .then(result => {
      // Not a new resource, so not 201
      res.status(200).json({
        post: result,
      })
    })
  .catch(err => {
    if (!err.statusCode) {
      err.statusCode = 500;
    }
    next(err);
  });
};

```

```

exports.deletePost = (req, res, next) => {
  const postId = req.params.postId;
  Post.findById(postId)
    .then(post => {
      console.log(post.creator);
      console.log(req.userId);
      if (!post) {
        const error = new Error('NO POST FOUND');
        error.statusCode = 404;
        throw error;
      }
      if (post.creator.toString() !== req.userId) {
        const error = new Error('NOT AUTHORISED!');
        error.statusCode = 403;
        throw error;
      }
      return Post.findByIdAndDelete(postId);
    })
    .then(result => {
      return User.findById(req.userId);
    })
    .then(user => {
      user.posts.pull(postId);
      user.save();
    })
    .then(result => {
      res.status(200).json({
        message: 'POST DELETED',
      });
    })
    .catch(err => {
      if (!err.statusCode) {

```

```
        err.statusCode = 500;
    };
    next(err);
});

};
```

11.2.4 Routes

11.2.4.1 auth.js

```
const express = require('express');
const { check, body } = require('express-validator');

const isAuth = require('../middleware/is-auth');

const User = require('../models/user');
const authController = require('../controllers/auth');

const router = express.Router();

router.put('/signup',
  [
    body('name')
      .trim()
      .notEmpty(),
    body('email')
      .isEmail()
      .withMessage('Please enter a valid email.')
      .custom((value, { req }) => {
        return User.findOne({
          email: value
        }).then(userDoc => {
          if (userDoc) {
            return Promise.reject('Email address
already exists!');
          }
        });
      })
      .normalizeEmail(),
    body('password')
      .trim()
      .isLength({ min: 8, max: 20 })
  ],
  authController.putSignup);

router.post('/login', authController.postLogin);

module.exports = router;
```

11.2.4.2 feed.js

```
// Third Party Dependencies
const express = require('express');
const { check, body } = require('express-validator');
const isAuth = require('../middleware/is-auth');

// Own dependencies
const feedController = require('../controllers/feed');

const router = express.Router();

// GET /feed/posts
router.get('/posts', isAuth, feedController.getPosts);

// POST /feed/post
router.post(
  '/post',
  isAuth,
  [
    body('title')
      .trim()
      .isLength({
        min: 5,
        max: 40,
      }),
    body('content')
      .trim()
      .isLength({
        min: 5,
        max: 500,
      }),
  ],
  feedController.postPost
);

// GET /feed/post/:postId
router.get('/post/:postId', isAuth, feedController.getPost);

router.put('/post/:postId',
  isAuth,
  [
    body('title')
      .trim()
      .isLength({
        min: 5,
```

```
        max: 40,
      }),
      body('content')
        .trim()
        .isLength({
          min: 5,
          max: 500,
        }),
    ],
    feedController.putPost
  );

router.delete('/post/:postId', isAuth, feedController.deletePost);

module.exports = router;
```

11.2.5 Middleware

11.2.5.1 is-auth.js

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  const authHeader = req.get('Authorization');
  if (!authHeader) {
    const error = new Error('AUTHENTICATION FAILED');
    error.statusCode = 401;
    throw error;
  }
  const token = authHeader.split(' ')[1];
  console.log(token);
  let decodedToken;
  try {
    // verify the JWT token + secret key
    decodedToken = jwt.verify(token,
      'f42952d5d0f2e342c4b6ebe80ab6cc968d8bcc7020ff1c7a5a8b057c790039e59
      8df323029cbac2993e7d6742b6362c3ca5cad0279925c43b288c9419e6122bb');
  } catch (err) {
    err.statusCode = 500;
    throw err;
  };
  if (!decodedToken) {
    const error = new Error('AUTHENTICATION FAILED!');
    error.statusCode = 401;
    throw error;
  };
  // stores decoded user ID for future requests
  req.userId = decodedToken.userId;
  next();
}
```

11.3 Appendix III: GraphQL Backend Source Code

11.3.1 Main Program Code

11.3.1.1 package.json:

```
{
  "name": "graphql-api",
  "version": "1.0.0",
  "description": "Complete Node.js Guide",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon app.js",
    "start-server": "app server.js",
    "codegen": "graphql-codegen --config code-gen.js"
  },
  "author": "L. M. Saxton",
  "license": "ISC",
  "devDependencies": {
    "@graphql-codegen/jsdoc": "^3.0.0",
    "@graphql-codegen/schema-ast": "^4.1.0",
    "nodemon": "^3.1.0",
    "ts-graphql-plugin": "^4.0.3",
    "typescript": "^5.5.4",
    "@graphql-codegen/typescript": "4.0.9",
    "@graphql-codegen/typescript-mongodb": "3.0.0",
    "@graphql-codegen/typescript-document-nodes": "4.0.9",
    "@graphql-codegen/typescript-resolvers": "4.2.1",
    "@graphql-codegen/introspection": "4.0.3",
    "@graphql-codegen/cli": "5.0.2"
  },
  "dependencies": {
    "@babel/plugin-transform-class-properties": "^7.24.7",
    "@graphql-codegen/cli": "^5.0.2",
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.20.2",
    "connect-flash": "^0.1.1",
    "connect-mongo": "^5.1.0",
    "csurf": "^1.11.0",
    "ejs": "^3.1.10",
    "express": "^4.19.2",
    "express-graphql": "^0.12.0",
    "express-session": "^1.18.0",
    "express-validator": "^7.1.0",
    "graphql": "^15.9.0",
```

```

    "jsonwebtoken": "^9.0.2",
    "mongodb": "^6.8.0",
    "mongoose": "^8.5.1",
    "multer": "^1.4.5-lts.1",
    "sqlite3": "^5.1.7",
    "uuid": "^10.0.0",
    "validator": "^13.12.0"
  }
}

```

11.3.1.2 app.js:

```

// Node Dependencies
const path = require('path');

// Third Party Dependencies
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const graphqlHttp = require('express-graphql');

// Own dependencies
const graphqlSchema = require('./graphql/schema');
const graphqlResolvers = require('./graphql/resolvers');
const auth = require('./middleware/auth');

const app = express();

// Server Middleware
app.use(express.json()); // POSTMAN doesn't work with this
included
// image upload service

app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*'); // allows
access to every client
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT,
PATCH, DELETE'); // allows the origins to use spec HTTP methods
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type,
Authorization'); // Which headers are always allowed || content
types allowed in request / allows extra authentication data from
clients
  if (req.method === 'OPTIONS') {
    return res.sendStatus(200);
  }
}

```



```

    // moves response ahead
    next();
  });

  // returns isAuth = false / true depending
  // kicks it back to resolver
  app.use(auth);

  app.use(
    '/api/graphql',
    gqlHttp.graphqlHTTP({
      schema: gqlSchema,
      rootValue: gqlResolvers,
      graphql: true,
      customFormatErrorFn(err) {
        if (!err.originalError) {
          return err;
        }
        const data = err.originalError.data;
        const message = err.message || 'An error
occurred.'; // || => default value if variable = null
        const code = err.originalError.code || 500;
        return { message: message, status: code, data: data };
      }
    }
  ));

  // Error middleware
  app.use((error, req, res, next) => {
    console.log(error);
    const status = error.statusCode || 500;
    const message = error.message;
    const data = error.data;
    res.status(status).json({
      message: message,
      data: data,
    });
  })

  mongoose.connect('mongodb://127.0.0.1:27017/graphql')
    .then(result => {
      console.log('Database connected...');
      app.listen(5000);
      console.log('Server listening...')
    })
    .catch(err => {
      console.log('DATABASE CONNECTION FAILED!');
    });

```

```
    console.log(err);  
  });
```

11.3.2 GraphQL

11.3.2.1 resolvers.js:

```
// Third Party Dependencies
const bcrypt = require('bcryptjs');
//requires GQL@15.9.0 --> not latest; no found vulns
const validator = require('validator');
const jwt = require('jsonwebtoken');

// Own Dependencies
const User = require('../models/user');
const Post = require('../models/post');

module.exports = {
  createUser: async function ({ userInput }, req) {
    // resolve userInputData with async userInput
    const errors = []
    // validation statements
    if (!validator.isEmail(userInput.email)) {
      errors.push({
        message: 'Email is invalid!'
      });
    }
    if (validator.isEmpty(userInput.password) ||
    20 ))) {
      errors.push({ message: 'Password too short!' });
    }
    if (errors.length > 0) {
      const error = new Error('Invalid input!');
      // linked to the formatError setting for GQL
      error.data = errors;
      error.code = 422;
      throw error;
    }
    const existingUser = await User.findOne({ email:
    userInput.email })
    if (existingUser) {
      const error = new Error('User already exists!');
      throw error;
    }
    const hashedPw = await bcrypt.hash(userInput.password,
    12);
    const user = new User({
```

```

        email: userInput.email,
        name: userInput.name,
        password: hashedPw,
    });
    const createdUser = await user.save();
    // user object returned to schema by resolver
    return {
        ...createdUser._doc,
        _id: createdUser._id.toString()
    };
},
login: async function({ email, password }) {
    const user = await User.findOne({ email: email });
    if (!user) {
        const error = new Error('User not found!');
        error.code = 401;
        throw error;
    }
    const isEqual = bcrypt.compare(password, user.password);
    if (!isEqual) {
        const error = new Error('Password is incorrect!');
        error.code = 401;
        throw error;
    }
    const token = jwt.sign({
        userId: user._id.toString(),
        email: user.email,
    },
        // Secret key, ASCII 64 bit
        'f42952d5d0f2e342c4b6ebe80ab6cc968d8bcc7020ff1c7a5a8b057c790039e59
8df323029cbac2993e7d6742b6362c3ca5cad0279925c43b288c9419e6122bb',
        { expiresIn: '1hr' }
    );
    return { token: token, userId: user._id.toString() };
},
createPost: async function ({ postInput }, req) {
    // check is user is authenticated
    if (!req.isAuthenticated()) {
        const error = new Error('ACTION FORBIDDEN');
        error.code = 404;
        throw error;
    }
    const errors = [];
    if (validator.isEmpty(postInput.title) ||

```

```

        !validator.isLength(postInput.title, { min: 5, max: 40
    ))) {
        errors.push({
            message: 'Invalid title: [min: 5, max: 40]'
        })
    }
    if (validator.isEmpty(postInput.content) ||
500 ))) {
        !validator.isLength(postInput.content, { min: 5, max:
        errors.push({
            message: 'Invalid content [min: 5, max: 500]'
        })
    }
    if (errors.length > 0) {
        const error = new Error('Invalid input!');
        // linked to the formatError setting for GQL
        error.data = errors;
        error.code = 422;
        throw error;
    }
    // get user from db
    const user = await User.findById(req.userId);
    if (!user) {
        const error = new Error('USER NOT FOUND!');
        error.code = 401;
        throw error;
    }
    const post = new Post({
        title: postInput.title,
        content: postInput.content,
        creator: user,
    });
    const createdPost = await post.save();
    // sends post to user's db file
    user.posts.push(createdPost);
    await user.save();
    return {
        ...createdPost._doc,
        _id: createdPost._id.toString(),
        createdAt: createdPost.createdAt.toISOString(),
        updatedAt: createdPost.updatedAt.toISOString()
    };
},
posts: async function (args, req) {
    if (!req.isAuthenticated) {
        const error = new Error('ACTION FORBIDDEN');

```

```

        error.code = 404;
        throw error;
    }
    const posts = await Post
        .find()
        .sort({ createdAt: -1 })
        .populate('creator');
    // format post data to that which GraphQL can read / return
data
    return {
        posts: posts.map(post => {
            return {
                ...post._doc,
                _id: post._id.toString(),
                createdAt: post.createdAt.toISOString(),
                updatedAt: post.updatedAt.toISOString(),
            }
        }
    ),
};

},
post: async function ({ postId }, req) {
    if (!req.isAuthenticated()) {
        const error = new Error('ACTION FORBIDDEN');
        error.code = 404;
        throw error;
    }
    const post = await
Post.findById(postId).populate('creator');
    if (!post) {
        const error = new Error('POST NOT FOUND!');
        error.code = 404;
        throw error;
    }
    return {
        ...post._doc,
        _id: post._id.toString(),
        createdAt: post.createdAt.toISOString(),
        updatedAt: post.updatedAt.toISOString(),
    };
},
updatePost: async function ({ postId, postInput }, req) {
    if (!req.isAuthenticated()) {
        const error = new Error('ACTION FORBIDDEN');
        error.code = 404;
        throw error;
    }
}

```

```

    const post = await
Post.findById(postId).populate('creator');
    if (!post) {
        const error = new Error('POST NOT FOUND!');
        error.code = 404;
        throw error;
    }
    if (post.creator._id.toString() !== req.userId.toString())
{
        const error = new Error('ACTION FORBIDDEN');
        error.code = 403;
        throw error;
    }
    const errors = [];
    if (validator.isEmpty(postInput.title) ||
        !validator.isLength(postInput.title, { min: 5, max: 40
})) {
        errors.push({
            message: 'Invalid title: [min: 5, max: 40]'
        })
    }
    if (validator.isEmpty(postInput.content) ||
500 !validator.isLength(postInput.content, { min: 5, max:
})) {
        errors.push({
            message: 'Invalid content [min: 5, max: 500]'
        })
    }
    if (errors.length > 0) {
        const error = new Error('Invalid input!');
        // linked to the formatError setting for GQL
        error.data = errors;
        error.code = 422;
        throw error;
    }
    post.title = postInput.title;
    post.content = postInput.content;
    const updatedPost = await post.save();
    return {
        ...updatedPost._doc,
        _id: updatedPost._id.toString(),
        createdAt: updatedPost.createdAt.toISOString(),
        updatedAt: updatedPost.updatedAt.toISOString(),
    };
},
deletePost: async function({ postId }, req) {

```

```

    if (!req.isAuthenticated) {
      const error = new Error('ACTION FORBIDDEN');
      error.code = 404;
      throw error;
    }
    const post = await
Post.findById(postId).populate('creator');
    if (!post) {
      const error = new Error('POST NOT FOUND!');
      error.code = 404;
      throw error;
    }
    if (post.creator._id.toString() !== req.userId.toString())
{
      const error = new Error('ACTION FORBIDDEN');
      error.code = 403;
      throw error;
    }
    // delete post from db
    await Post.findByIdAndDelete(postId);

    //delete post from user
    const user = await User.findById(req.userId);
    user.posts.pull(postId);
    await user.save();
    return true;
  },
  user: async function(args, req) {
    if (!req.isAuthenticated) {
      const error = new Error('ACTION FORBIDDEN');
      error.code = 404;
      throw error;
    }
    const user = await User.findById(req.userId);
    if (!user) {
      const error = new Error('USER NOT FOUND!');
      error.code = 404;
      throw error;
    }
    return {
      ...user._doc,
      _id: user._id.toString(),
    }
  },
  updateStatus: async function ({ status }, req) {
    if (!req.isAuthenticated) {

```



```

        const error = new Error('ACTION FORBIDDEN');
        error.code = 404;
        throw error;
    }
    const user = await User.findById(req.userId);
    if (!user) {
        const error = new Error('USER NOT FOUND!');
        error.code = 404;
        throw error;
    }
    user.status = status;
    await user.save();
    return {
        ...user._doc,
        _id: user._id.toString(),
    },
};

```

11.3.2.2 schema.js:

```

const { buildSchema } = require('graphql');

// build / export the GraphQL schema
module.exports = buildSchema(`
    type Post {
        _id: ID!
        title: String!
        content: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }

    type User {
        _id: ID!
        name: String!
        email: String!
        password: String!
        status: String!
        posts: [Post!]!
    }

    type AuthData {
        token: String!
    }
`);

```

```

        userId: String!
    }

    type PostData {
        posts: [Post!]!
    }

    input UserInputData {
        email: String!
        name: String!
        password: String!
    }

    input PostInputData {
        title: String!
        content: String!
    }

    type RootQuery {
        login(email: String!, password: String!): AuthData!
        posts: PostData!
        post(postId: ID!): Post!
        user: User!
    }

    type RootMutation {
        createUser(userInput: UserInputData): User!
        createPost(postInput: PostInputData): Post!
        updatePost(postId: ID!, postInput: PostInputData): Post!
        deletePost(postId: ID!): Boolean
        updateStatus(status: String!): User!
    }

    schema {
        query: RootQuery
        mutation: RootMutation
    }
};

```

11.3.3 Models

11.3.3.1 post.js:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const postSchema = new Schema({
  title: {
    type: String,
    required: true,
  },
  content: {
    type: String,
    required: true,
  },
  creator: {
    type: Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
},
{ timestamps: true }
);

module.exports = mongoose.model('Post', postSchema);
```

11.3.3.2 user.js:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const userSchema = new Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
  },
  password: {
    type: String,
    required: true
  }
});
```

```
    },
    status: {
      type: String,
      default: 'I am new!'
    },
    posts: [{
      // store schema / reference for posts
      type: Schema.Types.ObjectId,
      ref: 'Post',
    }]
  });

module.exports = mongoose.model('User', userSchema);
```

11.3.4 Controllers

11.3.4.1 auth.js:

```
const { validationResult } = require('express-validator');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

const User = require('../models/user');

exports.putSignup = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    const error = new Error('Validation failed!');
    error.statusCode = 422;
    error.data = errors.array();
    throw error;
  };

  const name = req.body.name;
  const email = req.body.email;
  const password = req.body.password;

  // hash the password
  bcrypt.hash(password, 12)
    .then(hashPw => {
      const user = new User({
        name: name,
        email: email,
        password: hashPw,
      });
      // save user object to db
      return user.save();
    })
    .then(result => {
      res.status(201).json({
        message: 'User created!',
        userId: result._id,
      });
    })
    .catch(err => {
      if (!err.statusCode) {
        err.statusCode = 500;
      }
      next(err);
    });
};
```

```

    });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  let loadedUser;
  User.findOne({ email: email })
    .then(user => {
      if (!user) {
        const error = new Error('Invalid credentials!');
        // not authenticated = 404
        error.statusCode = 404;
        throw error;
      };
      loadedUser = user;
      return bcrypt.compare(password, user.password);
    })
    .then(isEqual => {
      if (!isEqual) {
        const error = new Error('Invalid credentials!');
        error.statusCode = 404;
        throw error;
      };
      const token = jwt.sign({
        email: loadedUser.email,
        userId: loadedUser._id.toString(),
      },
        // secret string, ASCII 64 bit
        'f42952d5d0f2e342c4b6ebe80ab6cc968d8bcc7020ff1c7a5a8b057c790039e59
8df323029cbac2993e7d6742b6362c3ca5cad0279925c43b288c9419e6122bb',
        // token expiration
        { expiresIn: '1h' }
      );
      res.status(200).json({
        token: token,
        userId: loadedUser._id.toString(),
      })
    })
    .catch(err => {
      if (!err.statusCode) {
        err.statusCode = 500;
      }
      next(err);
    })
};

```

```
};
```

11.3.4.2 feed.js:

```
// Third Party Dependencies
const { validationResult } = require('express-validator');
const fs = require('fs');
const path = require('path');

// Own dependencies
const Post = require('../models/post');
const User = require('../models/user');

exports.getPosts = (req, res, next) => {
  Post.find()
    .then(posts => {
      // send back JSON as a response
      // status codes are very important for APIs
      res.status(200).json({
        posts: posts,
      })
    })
    .catch(err => {
      if (!err.statusCode) {
        err.statusCode = 500;
      }
      next(err);
    })
};

// naming convention: HTTP method + object
exports.postPost = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    const error = new Error('Validation failed!');
    console.log(error);
    // 422 => validation failed
    error.statusCode = 422;
    throw error;
  }

  const title = req.body.title;
  const content = req.body.content;
  let creator;
```

```

// create post in db
const post = new Post({
  title: title,
  content: content,
  creator: req.userId,
});
// save model to db
post
  .save()
  .then(result => {
    return User.findById(req.userId);
  })
  .then(user => {
    creator = user;
    // push the post object to the user
    user.posts.push(post);
    user.save();
  })
  .then(result => {
    // 201 = successful created resource
    res.status(201).json({
      post: post,
      creator: {
        _id: creator._id,
        name: creator.name,
      }
    });
  })
  .catch(err => {
    if (!err.statusCode) {
      err.statusCode = 500;
    }
    next(err);
  });
};

exports.getPost = (req, res, next) => {
  const postId = req.params.postId;
  Post.findById(postId)
    .then(post => {
      if (!post) {
        const error = new Error('POST NOT FOUND!');
        error.statusCode = 404;
        // passes error to catch block
        throw error;
      }
    });
};

```



```

        }
        res.status(200).json({
            post: post,
        })
    })
    .catch(err => {
        if (!err.statusCode) {
            err.statusCode = 500;
        }
        next(err);
    });
}

exports.putPost = (req, res, next) => {
    const postId = req.params.postId;
    const errors = validationResult(req);

    if (!errors.isEmpty()) {
        const error = new Error('VALIDATION FAILED');
        console.log(error)
        // 422 => validation failed
        error.statusCode = 422
        throw error;
    };

    const title = req.body.title;
    const content = req.body.content;

    Post.findById(postId)
        .then(post => {
            if (!post) {
                const error = new Error('NO POST FOUND');
                error.statusCode = 404;
                throw error;
            }
            // make sure the user ID matches the JWT
            if (post.creator.toString() !== req.userId) {
                const error = new Error('NOT AUTHORISED!');
                error.statusCode = 403;
                throw error;
            }
            // overwrite / save updated post
            post.title = title;
            post.content = content;
            return post.save();
        })
}

```

```

    .then(result => {
      // Not a new resource, so not 201
      res.status(200).json({
        post: result,
      })
    })
  .catch(err => {
    if (!err.statusCode) {
      err.statusCode = 500;
    }
    next(err);
  });
};

exports.deletePost = (req, res, next) => {
  const postId = req.params.postId;
  Post.findById(postId)
    .then(post => {
      console.log(post.creator);
      console.log(req.userId);
      if (!post) {
        const error = new Error('NO POST FOUND');
        error.statusCode = 404;
        throw error;
      }
      if (post.creator.toString() !== req.userId) {
        const error = new Error('NOT AUTHORISED!');
        error.statusCode = 403;
        throw error;
      }
      return Post.findByIdAndDelete(postId);
    })
    .then(result => {
      return User.findById(req.userId);
    })
    .then(user => {
      user.posts.pull(postId);
      user.save();
    })
    .then(result => {
      res.status(200).json({
        message: 'POST DELETED',
      });
    })
    .catch(err => {
      if (!err.statusCode) {

```

```
        err.statusCode = 500;
    };
    next(err);
});

};
```

11.3.5 Middleware

11.3.5.1 auth.js:

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  const authHeader = req.get('Authorization');
  if (!authHeader) {
    req.isAuth = false;
    return next();
  }
  const token = authHeader.split(' ')[1];
  let decodedToken;
  try {
    // verify the JWT token + secret key
    decodedToken = jwt.verify(token,
'f42952d5d0f2e342c4b6ebe80ab6cc968d8bcc7020ff1c7a5a8b057c790039e59
8df323029cbac2993e7d6742b6362c3ca5cad0279925c43b288c9419e6122bb');
  } catch (err) {
    req.isAuth = false;
    return next();
  };
  if (!decodedToken) {
    req.isAuth = false;
    return next();
  };
  // stores decoded user ID for future requests
  req.userId = decodedToken.userId;
  req.isAuth = true;
  next();
};
```

11.4 Appendix IV: Views-Rendering Application Demonstration

Figures 1 & 2: Unauthenticated *Home* and authenticated *Home*

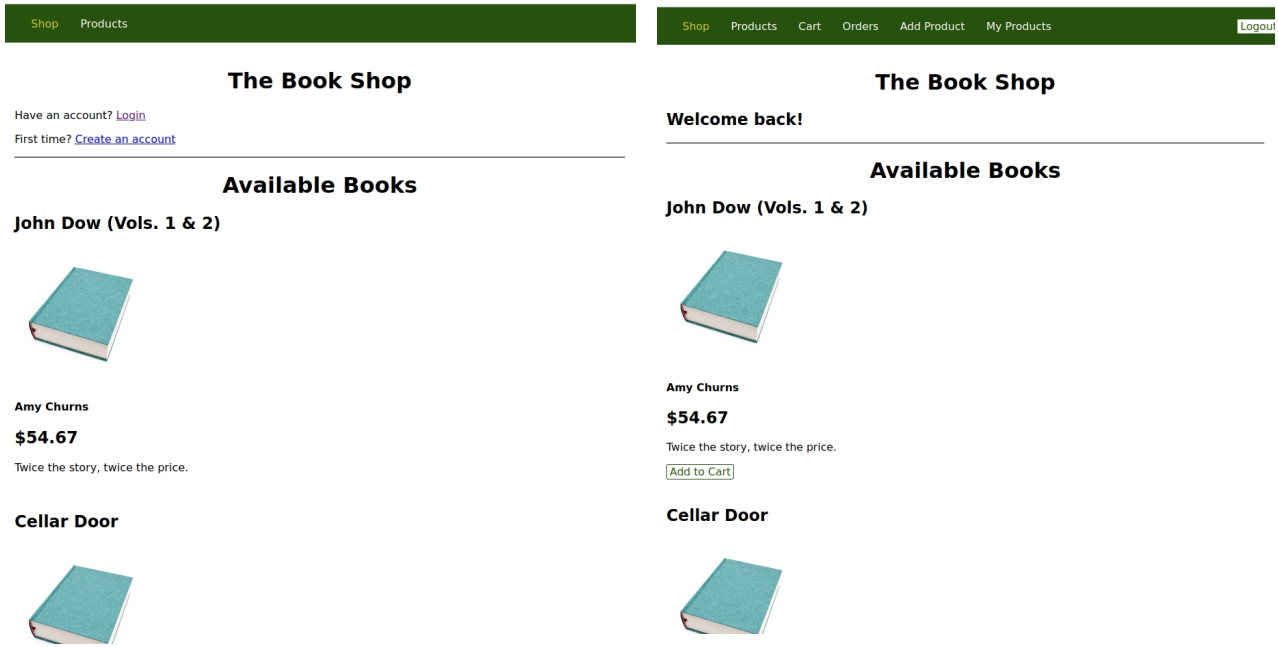


Figure 3: User Validation Example – Add Product

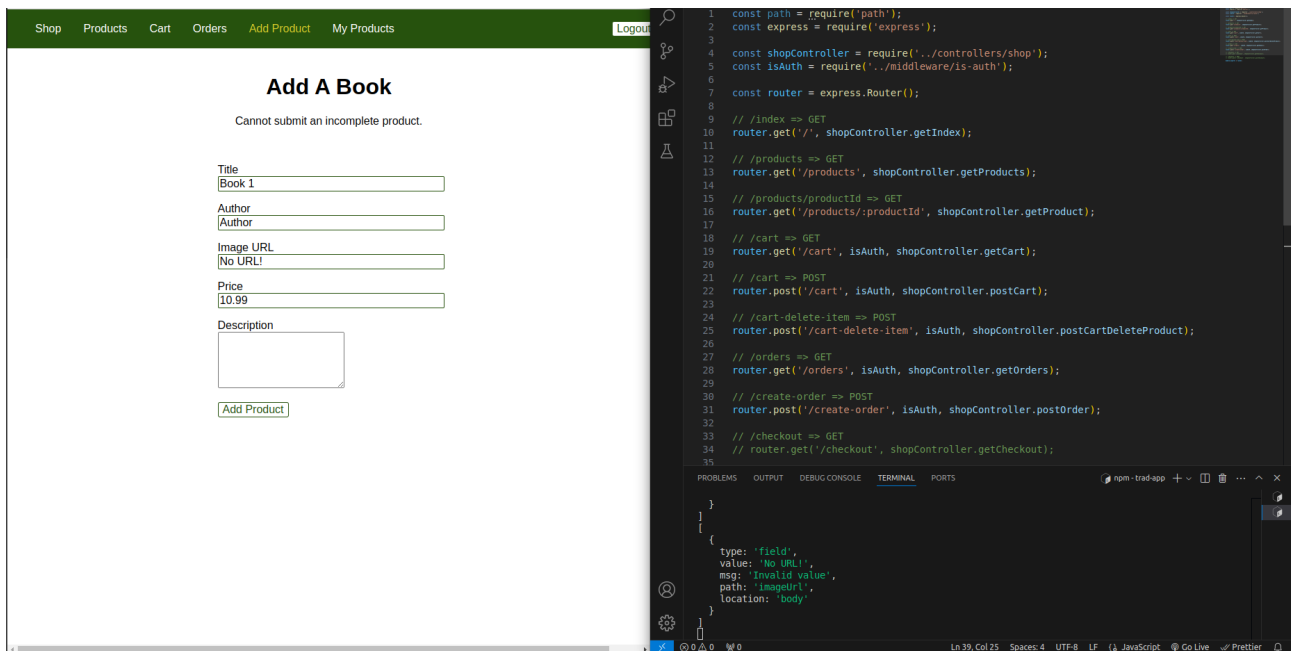


Figure 4: Application Logic Error – Order Placement

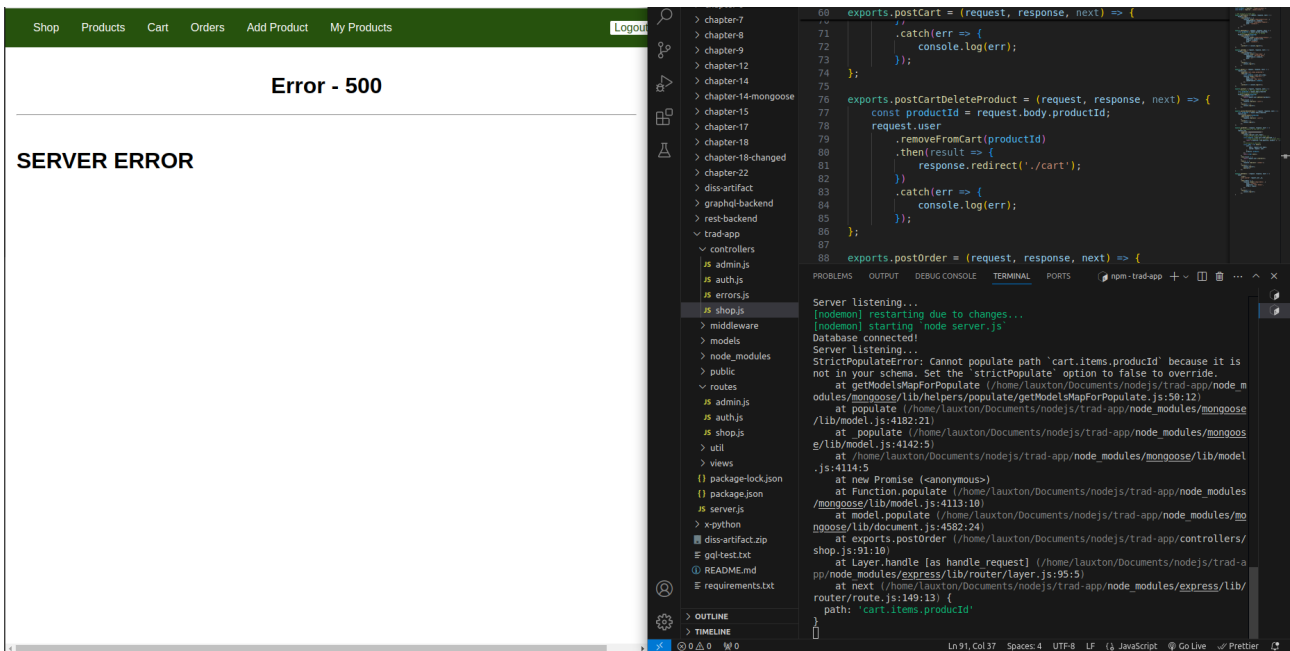


Figure 5: Fixed Application Logic Error – Order Placement

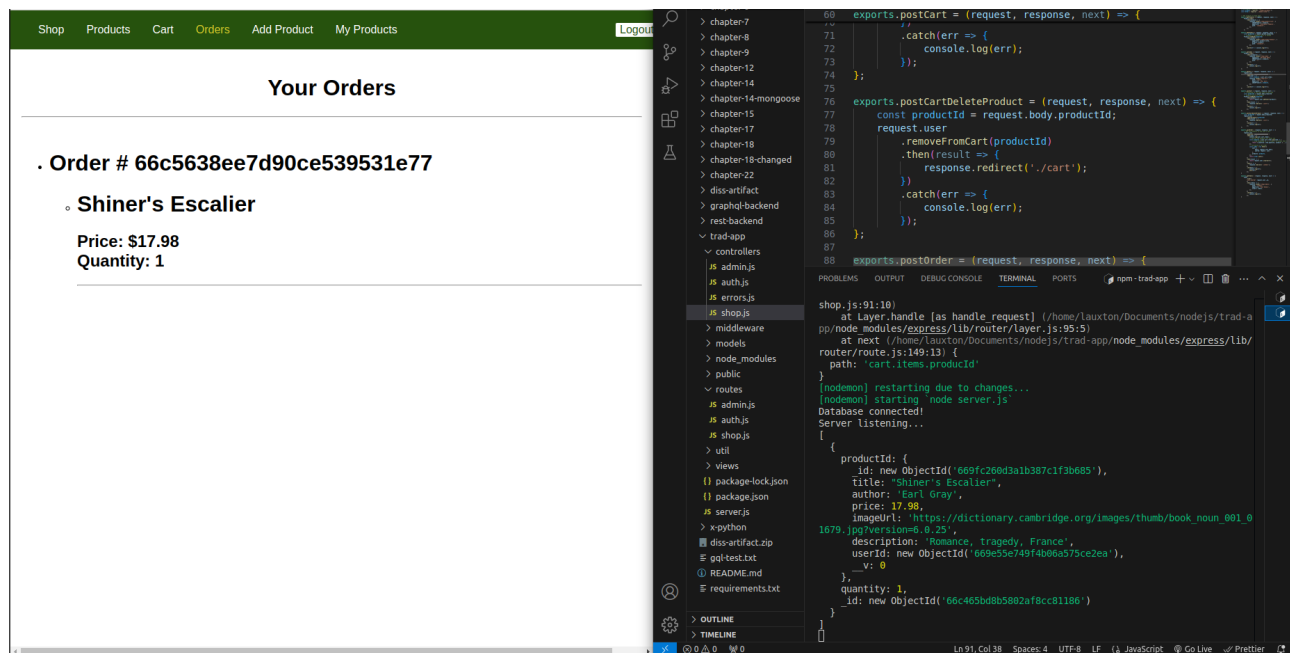
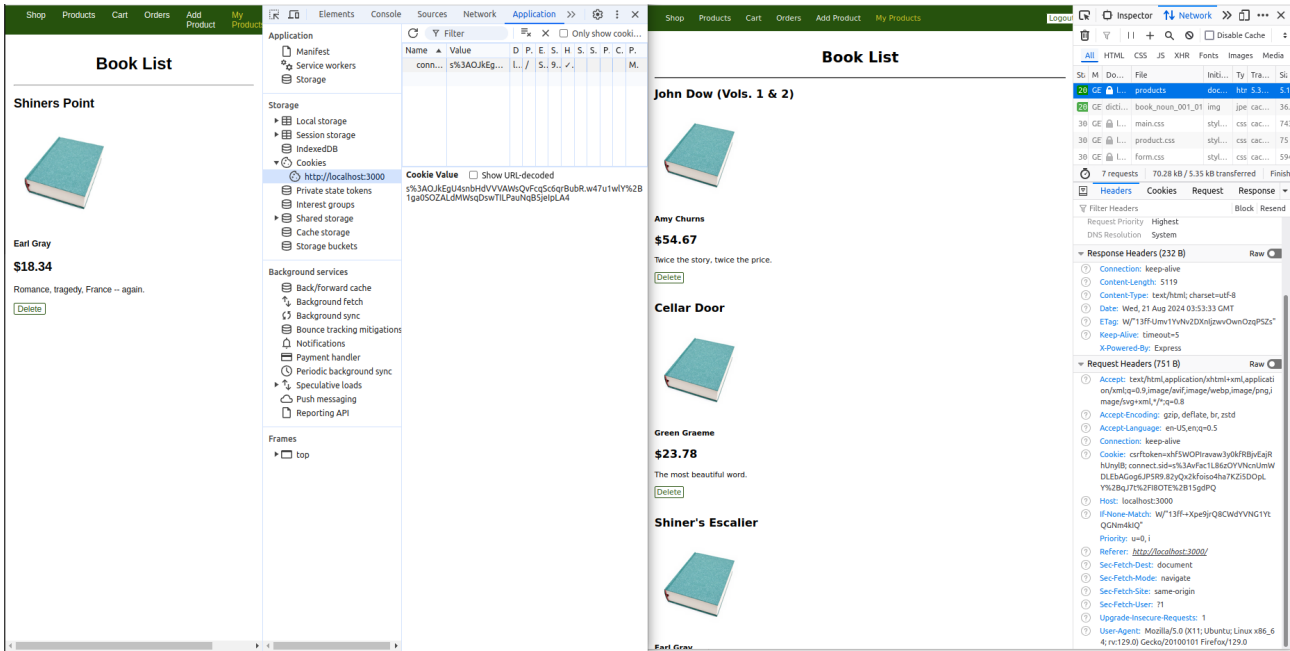


Figure 6: Access Control Example – User Sessions



11.5 Appendix V: REST Backend Demonstration

Figure 1: REST API Backend – ‘User A’ Login

The screenshot displays a REST client interface for the endpoint `http://localhost:4000/api/auth/login`. The request is a **POST** method with a body containing the following JSON:

```
1 {
2   "email": "jdoe34@example.com",
3   "password": "password"
4 }
```

The response is a **200 OK** status with a time of 221 ms and a size of 652 B. The response body is displayed in Pretty JSON format:

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmFpbCI6Impkb2UzNEIeGfGtGx1LmNvbSIsInVzZCJ2ZC6IjYyYzVlYmMxYzYwYT8zYj81ZDZkMGYMSIsImh0dCI6MTcyNDM0MSNswiZlhwIjoxNzI0NTIwMzk1fQ.7Ju32zjwSyKE-UIZUfvJE61TgZMFZv460zw7AXn8VE",
3   "userId": "66c5ebc1c6a04f9ed6d9d21"
4 }
```

Figure 2: REST API Backend – Unauthenticated GET

The screenshot displays a REST client interface for the endpoint `http://localhost:4000/api/posts`. The request is a **GET** method. The authorization type is set to **No Auth**. The response is a **401 Unauthorized** status with a time of 2 ms and a size of 432 B. The response body is displayed in Pretty JSON format:

```
1 {
2   "message": "AUTHENTICATION FAILED"
3 }
```

The interface also shows a message: "This request does not use any authorization. [Learn more about authorization](#)"

Figure 3: REST API Backend – ‘User B’ Login

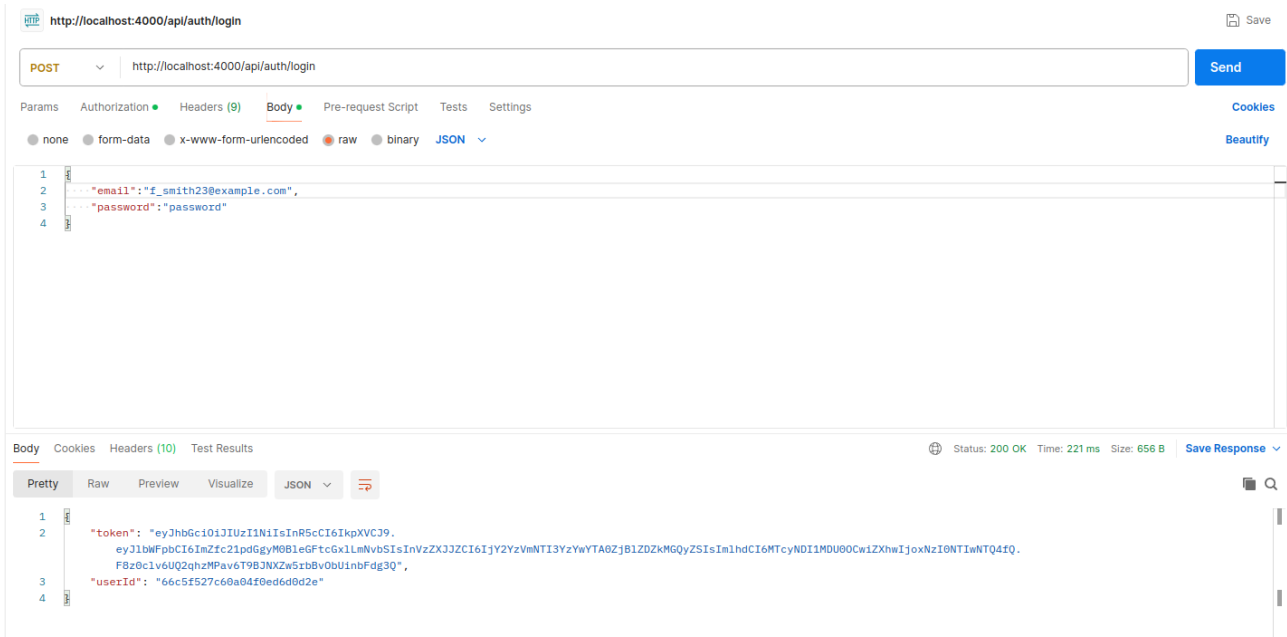


Figure 4: REST API Backend – Authenticated GET

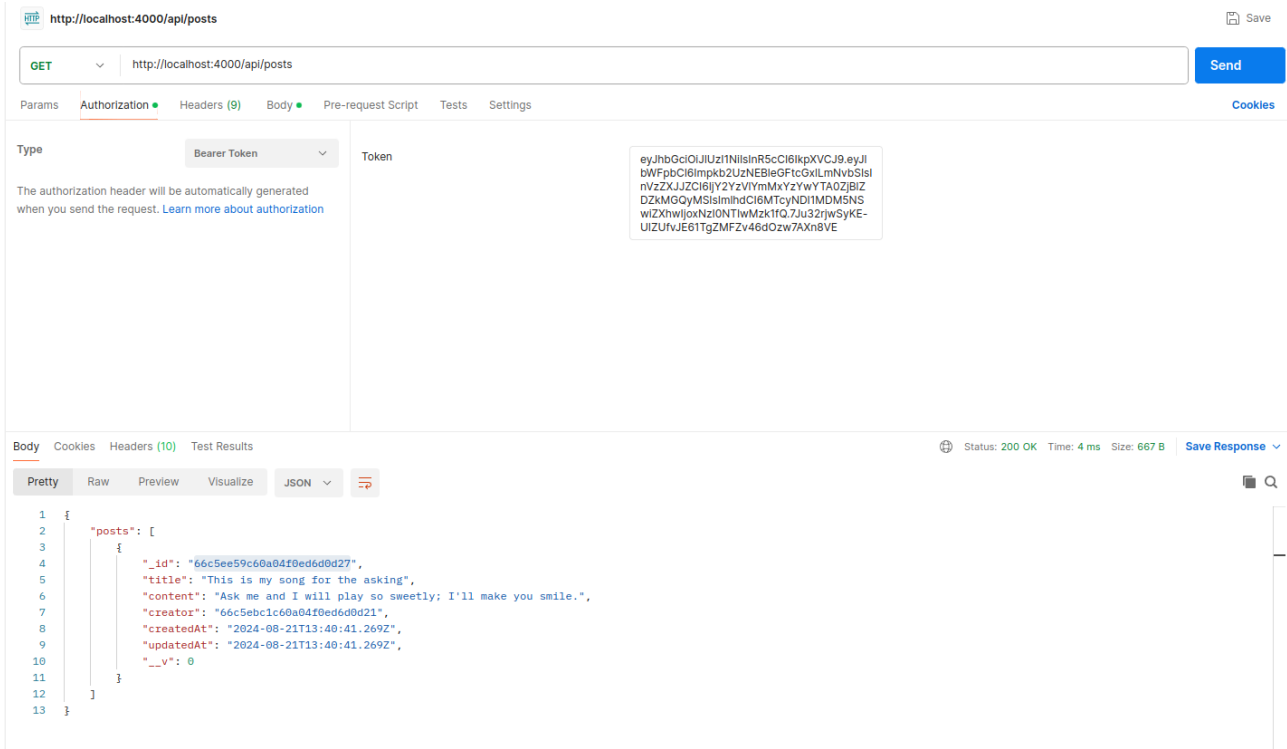


Figure 5: REST API Backend – Unauthenticated PUT Request

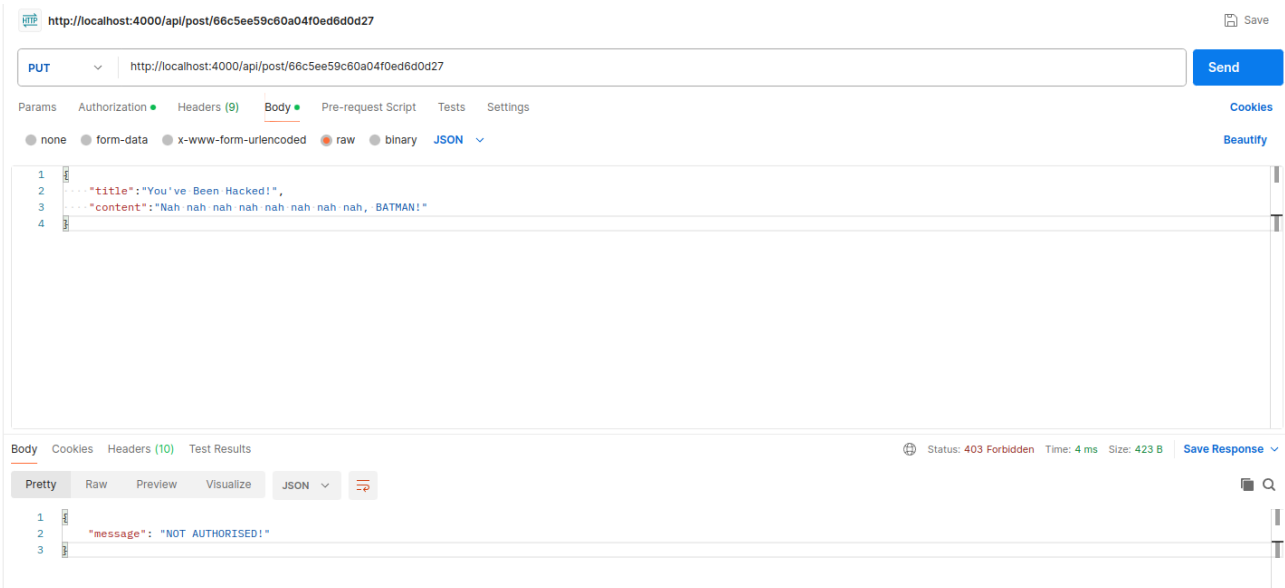
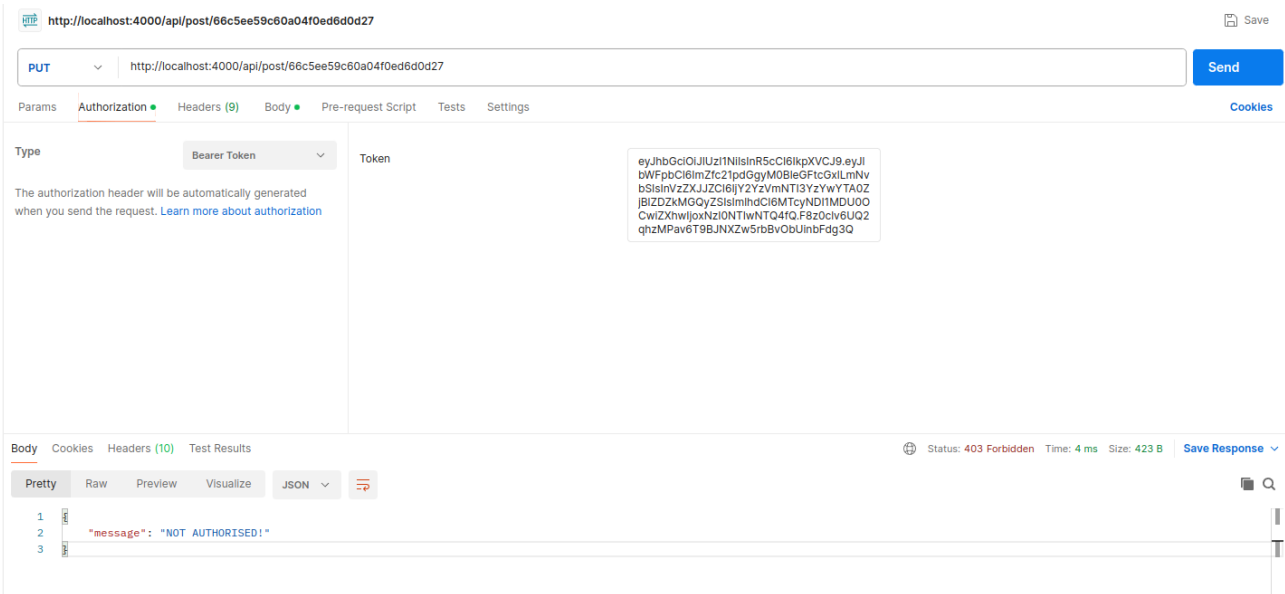
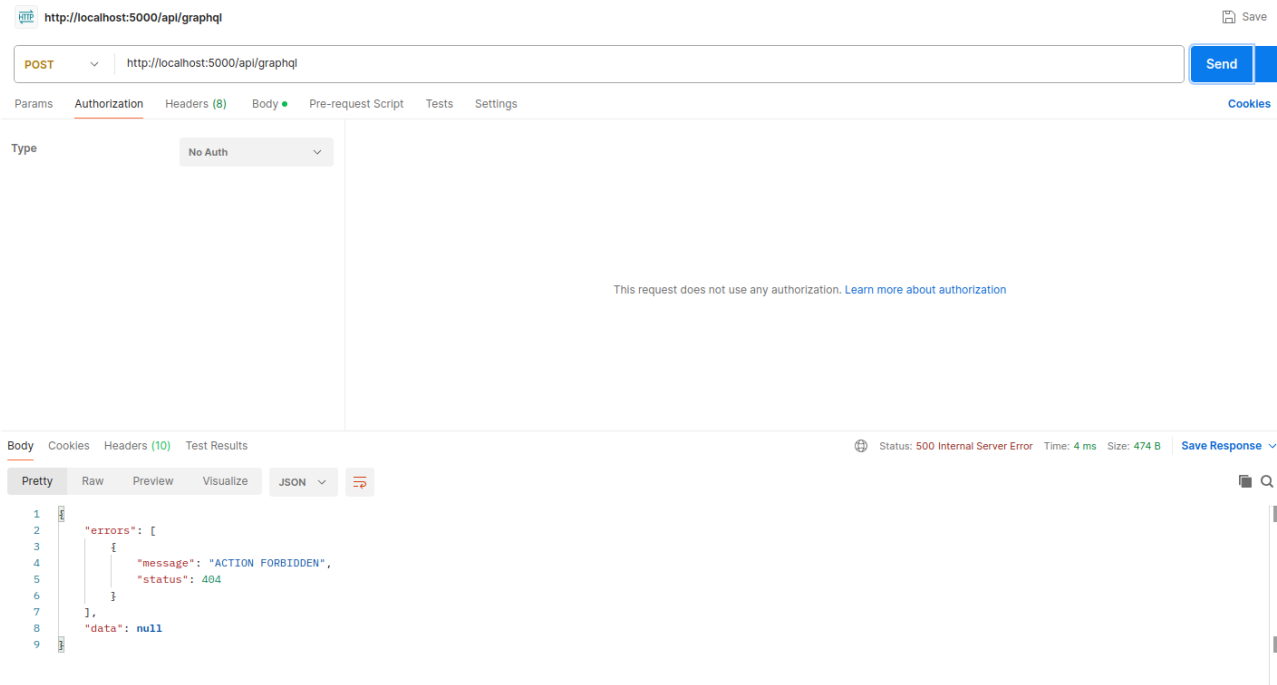


Figure 6: REST API Backend – Unauthenticated PUT Request Bearer Token



11.6 Appendix VI: GraphQL Backend Demonstration

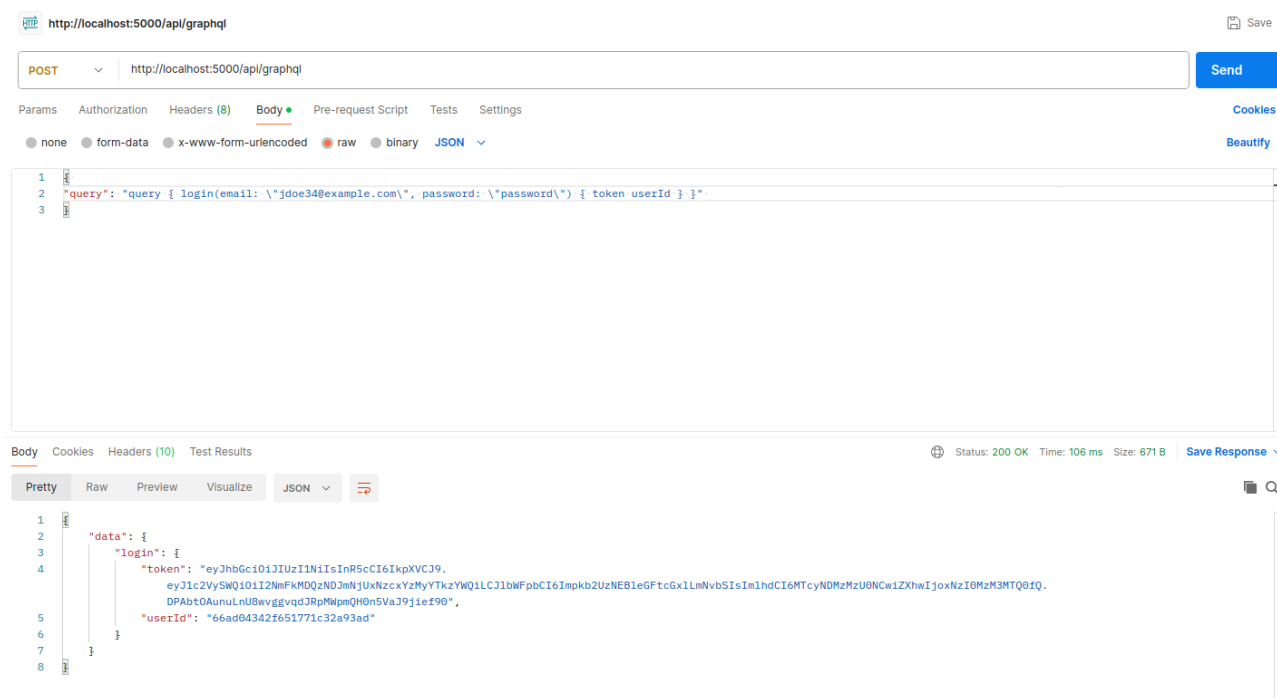
Figure 1: GraphQL API Backend – Unauthenticated Query



The screenshot shows a REST client interface for a GraphQL API. The URL is `http://localhost:5000/api/graphql` and the method is `POST`. The authorization is set to `No Auth`. The status is `500 Internal Server Error` with a time of `4 ms` and a size of `474 B`. The response body is a JSON object:

```
1 {
2   "errors": [
3     {
4       "message": "ACTION FORBIDDEN",
5       "status": 404
6     }
7   ],
8   "data": null
9 }
```

Figure 2: GraphQL API Backend – ‘User A’ Login



The screenshot shows a REST client interface for a GraphQL API. The URL is `http://localhost:5000/api/graphql` and the method is `POST`. The body contains a query:

```
1 {
2   "query": "query { login(email: \"jdoe34@example.com\", password: \"password\") { token userId } }"
3 }
```

The status is `200 OK` with a time of `106 ms` and a size of `671 B`. The response body is a JSON object:

```
1 {
2   "data": {
3     "login": {
4       "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ3aXZlIjoiInR5cCI6IkpXVCJ9.eyJ3aXZlIjoiInR5cCI6IkpXVCJ9",
5       "userId": "66ad84342f651771c32a93ad"
6     }
7   }
8 }
```


Figure 5: GraphQL API Backend – Unauthenticated Mutation

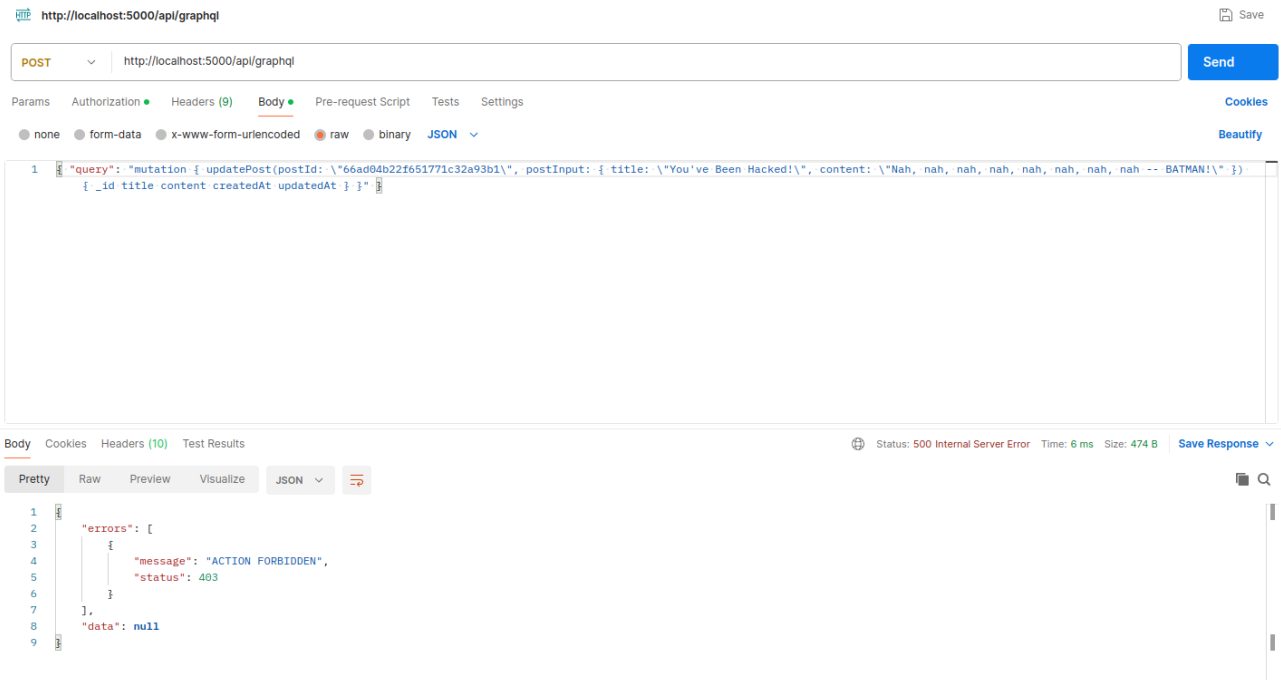
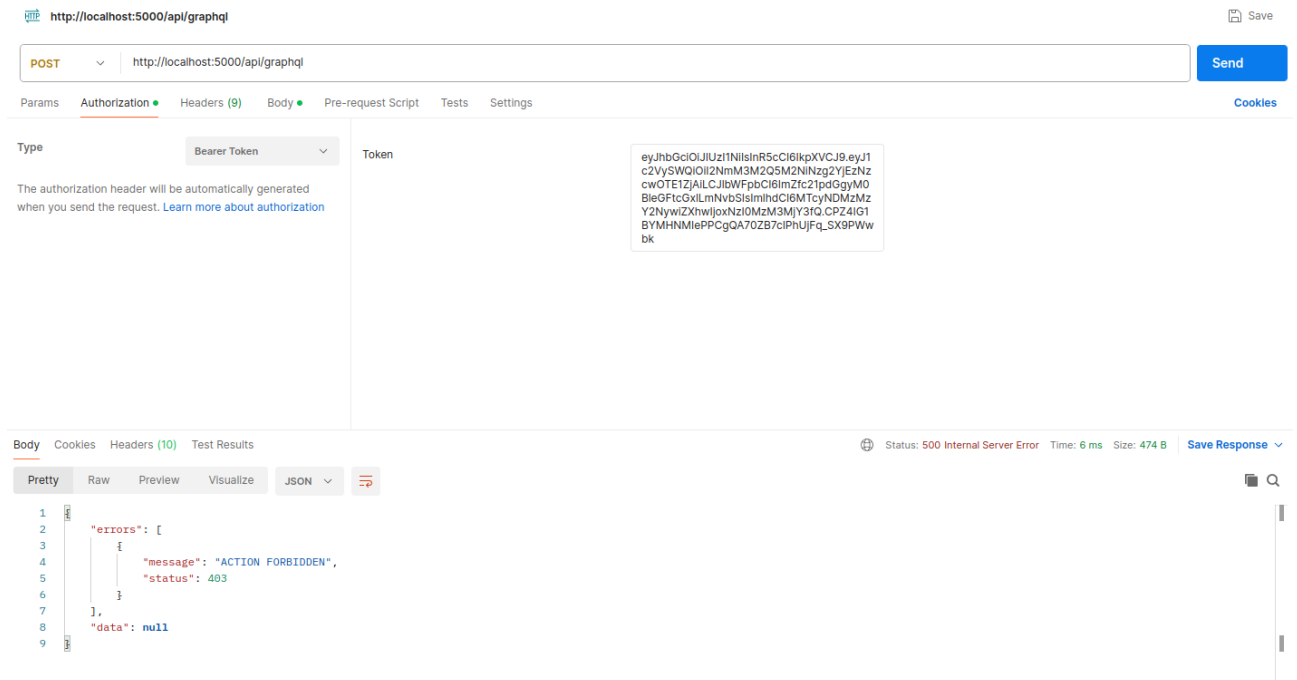


Figure 6: GraphQL API Backend – Unauthenticated Mutation Bearer Token



11.7 Appendix VII: GraphQL Postman JSON Queries

CREATE USER

```
{  
  "query": "mutation { createUser(userInput:  
    { email: \"newuser@example.com\", name: \"New User\",  
      password: \"securepassword\" }) { _id name email } }"}
```

LOGIN

```
{  
  "query": "query { login(email: \"user@example.com\", password:  
    \"yourpassword\") { token userId } }"}
```

GET USER

```
{  
  "query": "query { user { _id name email status posts { _id } } }"}
```

UPDATE USER

```
{  
  "query": "mutation { updateStatus(status: \"Just Got a Fender  
    Strat!\") { _id name email status } }"}
```

CREATE POST

```
{  
  "query": "mutation($postInput: PostInputData!)  
    { createPost(postInput: $postInput) { _id title content  
      creator { _id name } createdAt updatedAt } }", "variables":  
    { "postInput": { "title": "My New Post", "content": "This is  
      the content of the new post." } }  
}
```

GET POSTS

```
{  
  "query": "query { posts {posts { _id title content  
    creator{name}} } }"}
```

GET SINGLE POST

```
{  
  "query": "query { post(postId: \"66a67ef4ef394c4103be9f67\") {  
    _id title content creator { name } } }"}
```

UPDATE POST

```
{  
  "query": "mutation  
  { updatePost(postId: \"66a7611058def1d6f635af9d\", postInput:  
  { title: \"The Long and Winding Road\", content: \"Brought me  
  here to your door. Don't make me, nah, nah, nah -- I forget  
  the words.\" }) { _id title content createdAt updatedAt } }"}
```

DELETE POST

```
{  
  "query": "mutation  
  { deletePost(postId: \"66a68012b6439bf6f4dc083c\") }"}
```

11.8 Appendix VIII: Active Reconnaissance Source Code

11.8.1 Views-Rending Middleware

11.8.1.1 Step 1: Baseline Scanning

nmap.sh:

```
#!/bin/bash

# Log file
localhost3000_log="$HOME/diss/localhost3000/nmap/timestamps/nmap-
timestamps.txt"
localhost3000_output="$HOME/diss/localhost3000/nmap/output/"

# create file if not exist
touch "$localhost3000_log" || {
    echo "Failed to create file $localhost3000_log";
    exit 1;
}

touch "$localhost3000_output" || {
    echo "Failed to create file $localhost3000_output";
    exit 1;
}

# Scan variable
scan_number=1

# increment variable
var=0

{
    echo "NMAP Scan Timestamps for localhost 3000"
    echo "Make sure to cross reference with scan output"
    echo "-"
    echo "NMAP"
    echo "-"
} >> "$localhost3000_log" || {
    echo "Failed to write to $localhost3000_log";
    exit 1;
}

while [ $var -lt 50 ];
do
    {
```



```

        echo "Scan $scan_number"
        echo "Start: $(date +%T.%N)"
    } >> "$localhost3000_log"

    nmap -sC -sV localhost -oN $localhost3000_output/output-scan-
$scan_number

    {
        echo "Finish: $(date +%T.%N)"
        echo "-"
    } >> "$localhost3000_log"

    ((var++))
    ((scan_number++))
done

```

11.8.1.2 Step 2: URI Brute-Forcing

dirsearch.sh

```

#!/bin/bash

# Log life
localhost3000_log="$HOME/diss/localhost3000/dirsearch/timestamps/
timestamps.txt"
localhost3000_output="$HOME/diss/localhost3000/dirsearch/reports"

# create file if not exist
touch "$localhost3000_log" || {
    echo "Failed to create file $localhost3000_log";
    exit 1;
}

# Scan variable
scan_number=1

# increment variable
var=0

{
    echo "Dirsearch Scan Timestamps for localhost 3000"
    echo "Make sure to cross reference with scan output"
    echo "Methods: GET, POST, PUT, DELETE"
    echo "-"
} >> "$localhost3000_log" || {

```

```

    echo "Failed to write to $localhost3000_log";
    exit 1;
}

while [ $var -lt 50 ];
do
    {
        echo "DIRSEARCH"
        echo "Scan $scan_number"
        echo "GET: $(date) "

        } >> "$localhost3000_log"
    python3 ~/dirsearch/dirsearch.py -t 30 -u
    http://127.0.0.1:3000 --delay=1 -m GET --
    subdirs /,admin/,auth/,user/ -w $HOME/api/wordlists/dummylist.txt
    -o $localhost3000_output/GETscan-$scan_number.txt
    {
        echo "POST: $(date) "
        } >> "$localhost3000_log"
    python3 ~/dirsearch/dirsearch.py -t 30 -u
    http://127.0.0.1:3000 --delay=1 -m POST --
    subdirs /,admin/,auth/,user/ -w $HOME/api/wordlists/dummylist.txt
    -o $localhost3000_output/POSTscan-$scan_number.txt
    {
        echo "PUT: $(date) "
        } >> "$localhost3000_log"
    python3 ~/dirsearch/dirsearch.py -t 30 -u
    http://127.0.0.1:3000 --delay=1 -m PUT --
    subdirs /,admin/,auth/,user/ -w $HOME/api/wordlists/dummylist.txt
    -o $localhost3000_output/PUTscan-$scan_number.txt
    {
        echo "DELETE: $(date) "
        } >> "$localhost3000_log"
    python3 ~/dirsearch/dirsearch.py -t 30 -u
    http://127.0.0.1:3000 --delay=1 -m DELETE --
    subdirs /,admin/,auth/,user/ -w $HOME/api/wordlists/dummylist.txt
    -o $localhost3000_output/DELETEScan-$scan_number.txt
    {
        echo "Finish: $(date) "
        echo "-"
        } >> "$localhost3000_log"

    ((var++))
    ((scan_number++))
done

```

```
shutdown now
```

11.8.1.3 Step 3: Content Discovery

dirsearch-auth.sh

```
#!/bin/bash

# Log life
localhost3000_log="$HOME/diss/localhost3000/dirsearch/timestamps/
timestamps-auth.txt"
localhost3000_output="$HOME/diss/localhost3000/dirsearch/reports/
auth"

# create file if not exist
touch "$localhost3000_log" || {
    echo "Failed to create file $localhost3000_log";
    exit 1;
}

# Scan variable
scan_number=1

# increment variable
var=0

{
    echo "Dirsearch Scan Timestamps for localhost 3000"
    echo "Make sure to cross reference with scan output"
    echo "Methods: GET, POST, PUT, DELETE"
    echo "-"
} >> "$localhost3000_log" || {
    echo "Failed to write to $localhost3000_log";
    exit 1;
}

while [ $var -lt 50 ];
do
    {
        echo "DIRSEARCH"
        echo "Scan $scan_number"
        echo "GET: $(date)"

    } >> "$localhost3000_log"
```

```

python3 ~/dirsearch/dirsearch.py -t 30 -u
http://127.0.0.1:3000 --delay=1 -m GET --
cookie="csrftoken=xhf5WOPiravaw3y0kfrBjvEajRhUnylB; connect.sid=s
%3ARzTpER2jBUm4i6E43sb5WdJZQUyC8Q3A.m8XqSPEJkcmPm%2FBaRt8P997jJ
%2BTu0ZiNhPKh0%2B2m470" --suffixes /,{id} --
subdirs /,admin/,auth/,user/ -w $HOME/api/wordlists/dummylist.txt
-o $localhost3000_output/GETscan-$scan_number.txt
{
    echo "POST: $(date)"
} >> "$localhost3000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u
http://127.0.0.1:3000 --
cookie="csrftoken=xhf5WOPiravaw3y0kfrBjvEajRhUnylB; connect.sid=s
%3ARzTpER2jBUm4i6E43sb5WdJZQUyC8Q3A.m8XqSPEJkcmPm%2FBaRt8P997jJ
%2BTu0ZiNhPKh0%2B2m470" --suffixes /,{id} --delay=1 -m POST --
subdirs /,admin/,auth/,user/ -w $HOME/api/wordlists/dummylist.txt
-o $localhost3000_output/POSTscan-$scan_number.txt
{
    echo "PUT: $(date)"
} >> "$localhost3000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u
http://127.0.0.1:3000 --
cookie="csrftoken=xhf5WOPiravaw3y0kfrBjvEajRhUnylB; connect.sid=s
%3ARzTpER2jBUm4i6E43sb5WdJZQUyC8Q3A.m8XqSPEJkcmPm%2FBaRt8P997jJ
%2BTu0ZiNhPKh0%2B2m470" --suffixes /,{id} --delay=1 -m PUT --
subdirs /,admin/,auth/,user/ -w $HOME/api/wordlists/dummylist.txt
-o $localhost3000_output/PUTscan-$scan_number.txt
{
    echo "DELETE: $(date)"
} >> "$localhost3000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u
http://127.0.0.1:3000 --
cookie="csrftoken=xhf5WOPiravaw3y0kfrBjvEajRhUnylB; connect.sid=s
%3ARzTpER2jBUm4i6E43sb5WdJZQUyC8Q3A.m8XqSPEJkcmPm%2FBaRt8P997jJ
%2BTu0ZiNhPKh0%2B2m470" --suffixes /,{id} --delay=1 -m DELETE --
subdirs /,admin/,auth/,user/ -w $HOME/api/wordlists/dummylist.txt
-o $localhost3000_output/DELETEDscan-$scan_number.txt
{
    echo "Finish: $(date)"
    echo "-"
} >> "$localhost3000_log"

((var++))
((scan_number++))
done

```

shutdown now

11.8.2 REST API

11.8.2.1 Step 1: Baseline Scanning

nmap.sh

```
#!/bin/bash

# Log file
localhost4000_log="$HOME/diss/localhost4000/nmap/timestamps/nmap-
timestamps.txt"
localhost4000_output="$HOME/diss/localhost4000/nmap/output"

# create file if not exist
touch "$localhost4000_log" || {
    echo "Failed to create file $localhost4000_log";
    exit 1;
}

touch "$localhost4000_output" || {
    echo "Failed to create file $localhost4000_output";
    exit 1;
}

# Scan variable
scan_number=1

# increment variable
var=0

{
    echo "NMAP Scan Timestamps for localhost 3000"
    echo "Make sure to cross reference with scan output"
    echo "-"
    echo "NMAP"
    echo "-"
} >> "$localhost4000_log" || {
    echo "Failed to write to $localhost4000_log";
    exit 1;
}

while [ $var -lt 50 ];
do
    {
        echo "Scan $scan_number"
        echo "Start: $(date +%T.%N)"
    }
```

```

} >> "$localhost4000_log"

nmap -sC -sV localhost -oN $localhost4000_output/output-scan-
$scan_number

{
    echo "Finish: $(date +%T.%N)"
    echo "-"
} >> "$localhost4000_log"

((var++))
((scan_number++))
done

```

11.8.2.2 Step 2: URI Brute-Forcing

dirsearch.sh

```

#!/bin/bash

# Log life
localhost4000_log="$HOME/diss/localhost4000/dirsearch/timestamps/
timestamps.txt"
localhost4000_output="$HOME/diss/localhost4000/dirsearch/reports"

# create file if not exist
touch "$localhost4000_log" || {
    echo "Failed to create file $localhost4000_log";
    exit 1;
}

# Scan variable
scan_number=1

# increment variable
var=0

# JWT

jwt_token="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6Imprb2
UzNEBLEGFtcGxlLmNvbSIsInVzZXJJZCI6IjY2YjM4NjRiZTZjZWQ1MDZlZGY5ZmYx
MCI6ImhhdCI6MTcyMzA0MTQwNCwiZXhwIjojNzIzZjZjZWQ1MDZlZGY5ZmYx
FH5_3fEJDzlh5gSKrqwRGP8evu0"

```

```

{
    echo "Dirsearch Scan Timestamps for localhost 4000"
    echo "Make sure to cross reference with scan output"
    echo "Methods: GET, POST, PUT, DELETE, PATCH"
    echo "-"
} >> "$localhost4000_log" || {
    echo "Failed to write to $localhost4000_log";
    exit 1;
}

while [ $var -lt 50 ]; do
{
    echo "DIRSEARCH"
    echo "Scan $scan_number"
    echo "GET: $(date)"
} >> "$localhost4000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:4000 --
auth-type bearer --credentials jwt_token --delay=1 -m GET --
subdirs /,api/,admin/,auth/,api/admin/,api/auth/ -w
$HOME/api/wordlists/dummylist.txt -o
$localhost4000_output/GETscan-$scan_number.txt

{
    echo "POST: $(date)"
} >> "$localhost4000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:4000 --
delay=1 -m POST --subdirs /,api/,admin/,auth/,api/admin/,api/auth/
-w $HOME/api/wordlists/dummylist.txt -o
$localhost4000_output/POSTscan-$scan_number.txt

gql_output=$(grep -oP "graphql" "$localhost4000_output/POSTscan-
$scan_number.txt")
if [ -n "$gql_output" ]; then
    echo "GraphQL endpoint detected -- SCAN STOPPED!"
    {
        echo "Finish: $(date)"
        echo "-"
    } >> "$localhost4000_log"
    ((var++))
    ((scan_number++))
    continue
fi

{
    echo "PUT: $(date)"
} >> "$localhost4000_log"

```



```
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:4000 --
delay=1 -m PUT --subdirs /,api/,admin/,auth/,api/admin/,api/auth/
-w $HOME/api/wordlists/dummylist.txt -o
$localhost4000_output/PUTscan-$scan_number.txt
```

```
{
    echo "DELETE: $(date) "
} >> "$localhost4000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:4000 --
delay=1 -m DELETE --subdirs
/,api/,admin/,auth/,api/admin/,api/auth/ -w
$HOME/api/wordlists/dummylist.txt -o
$localhost4000_output/DELETEScan-$scan_number.txt
```

```
{
    echo "PATCH: $(date) "
} >> "$localhost4000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:4000 --
delay=1 -m PATCH --subdirs
/,api/,admin/,auth/,api/admin/,api/auth/ -w
$HOME/api/wordlists/dummylist.txt -o
$localhost4000_output/PATCHscan-$scan_number.txt
```

```
{
    echo "Finish: $(date) "
    echo "-"
} >> "$localhost4000_log"
((var++))
((scan_number++))
done
```

shutdown now

11.8.2.3 Step 3: Content Discovery

dirsearch-auth.sh

```
#!/bin/bash
```

```
# Log life
```

```
localhost4000_log="$HOME/diss/localhost4000/dirsearch/timestamps/
timestamps-auth.txt "
```

```
localhost4000_output="$HOME/diss/localhost4000/dirsearch/reports/
auth"
```



```

suffixes /,/{id} --subdirs
/,api/,admin/,auth/,api/admin/,api/auth/ -w
$HOME/api/wordlists/dummylist.txt -o
$localhost4000_output/POSTscan-$scan_number.txt

gql_output=$(grep -oP "graphql" "$localhost4000_output/POSTscan-
$scan_number.txt")
if [ -n "$gql_output" ]; then
    echo "GraphQL endpoint detected -- SCAN STOPPED!"
    {
        echo "Finish: $(date)"
        echo "-"
    } >> "$localhost4000_log"
    ((var++))
    ((scan_number++))
    continue
fi

{
    echo "PUT: $(date)"
} >> "$localhost4000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:4000 --
auth-type=bearer --auth=jwt_token --delay=1 -m PUT --suffixes
/,/{id} --subdirs /,api/,admin/,auth/,api/admin/,api/auth/ -w
$HOME/api/wordlists/dummylist.txt -o
$localhost4000_output/PUTscan-$scan_number.txt

{
    echo "DELETE: $(date)"
} >> "$localhost4000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:4000 --
auth-type=bearer --auth=jwt_token --delay=1 -m DELETE --
suffixes /,/{id} --subdirs
/,api/,admin/,auth/,api/admin/,api/auth/ -w
$HOME/api/wordlists/dummylist.txt -o
$localhost4000_output/DELETEScan-$scan_number.txt

{
    echo "PATCH: $(date)"
} >> "$localhost4000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:4000 --
auth-type=bearer --auth=jwt_token --delay=1 -m PATCH --
suffixes /,/{id} --subdirs
/,api/,admin/,auth/,api/admin/,api/auth/ -w
$HOME/api/wordlists/dummylist.txt -o
$localhost4000_output/PATCHscan-$scan_number.txt

```

```
{
    echo "Finish: $(date) "
    echo "-"
} >> "$localhost4000_log"
((var++))
((scan_number++))
done

shutdown now
```

11.8.3 GraphQL API

11.8.3.1 Step 1: Baseline Scanning

nmap.sh

```
#!/bin/bash

# Log file
localhost5000_log="$HOME/diss/localhost5000/nmap/timestamps/nmap-
timestamps.txt"
localhost5000_output="$HOME/diss/localhost5000/nmap/output/"

# create file if not exist
touch "$localhost5000_log" || {
    echo "Failed to create file $localhost5000_log";
    exit 1;
}

touch "$localhost5000_output" || {
    echo "Failed to create file $localhost5000_output";
    exit 1;
}

# Scan variable
scan_number=1

# increment variable
var=0

{
    echo "NMAP Scan Timestamps for localhost 3000"
    echo "Make sure to cross reference with scan output"
    echo "-"
    echo "NMAP"
    echo "-"
} >> "$localhost5000_log" || {
    echo "Failed to write to $localhost5000_log";
    exit 1;
}

while [ $var -lt 50 ];
do
    {
        echo "Scan $scan_number"
        echo "Start: $(date +%T.%N)"
    }
```

```

    } >> "$localhost5000_log"

    nmap -sC -sV localhost -oN $localhost5000_output/output-scan-
$scan_number

    {
        echo "Finish: $(date +%T.%N)"
        echo "-"
    } >> "$localhost5000_log"

    ((var++))
    ((scan_number++))
done

```

11.8.3.2 Step 2: URI Brute-Forcing

dirsearch.sh

```

#!/bin/bash

# Log life
localhost5000_log="$HOME/diss/localhost5000/dirsearch/timestamps/
timestamps.txt"
localhost5000_output="$HOME/diss/localhost5000/dirsearch/reports"

# create file if not exist
touch "$localhost5000_log" || {
    echo "Failed to create file $localhost5000_log";
    exit 1;
}

# Scan variable
scan_number=1

# increment variable
var=0

{
    echo "Dirsearch Scan Timestamps for localhost 5000"
    echo "Make sure to cross reference with scan output"
    echo "Methods: GET, POST, PUT, DELETE, PATCH"
    echo "-"
} >> "$localhost5000_log" || {
    echo "Failed to write to $localhost5000_log";

```

```

        exit 1;
    }

while [ $var -lt 50 ]; do
{
    echo "DIRSEARCH"
    echo "Scan $scan_number"
    echo "GET: $(date) "
} >> "$localhost5000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:5000 --
delay=1 -m GET --subdirs /,api/,admin/,auth/,api/admin/,api/auth/
-w $HOME/api/wordlists/dummylist.txt -o
$localhost5000_output/GETscan-$scan_number.txt

{
    echo "POST: $(date) "
} >> "$localhost5000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:5000 --
delay=1 -m POST --subdirs /,api/,admin/,auth/,api/admin/,api/auth/
-w $HOME/api/wordlists/dummylist.txt -o
$localhost5000_output/POSTscan-$scan_number.txt

gql_output=$(grep -oP "graphql" "$localhost5000_output/POSTscan-
$scan_number.txt")
if [ -n "$gql_output" ]; then
    echo "GraphQL endpoint detected -- SCAN STOPPED!"
    {
        echo "Finish: $(date) "
        echo "-"
    } >> "$localhost5000_log"
    ((var++))
    ((scan_number++))
    continue
fi

{
    echo "PUT: $(date) "
} >> "$localhost5000_log"
python3 ~/di search/dirsearch.py -t 30 -u http://127.0.0.1:5000 --
delay=1 -m PUT --subdirs /,api/,admin/,auth/,api/admin/,api/auth/
-w $HOME/api/wordlists/dummylist.txt -o
$localhost5000_output/PUTscan-$scan_number.txt

{
    echo "DELETE: $(date) "
} >> "$localhost5000_log"

```

```

python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:5000 --
delay=1 -m DELETE --subdirs
/,api/,admin/,auth/,api/admin/,api/auth/ -w
$HOME/api/wordlists/dummylist.txt -o
$localhost5000_output/DELETEScan-$scan_number.txt

{
    echo "PATCH: $(date) "
} >> "$localhost5000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:5000 --
delay=1 -m PATCH --subdirs
/,api/,admin/,auth/,api/admin/,api/auth/ -w
$HOME/api/wordlists/dummylist.txt -o
$localhost5000_output/PATCHscan-$scan_number.txt

{
    echo "Finish: $(date) "
    echo "-"
} >> "$localhost5000_log"
((var++))
((scan_number++))
done

shutdown now

```

11.8.3.3 Step 3: Introspection Verification

nmap-introspection.sh

```

#!/bin/bash

# Log life
localhost5000_log="$HOME/diss/localhost5000/dirsearch/timestamps/
timestamps.txt"
localhost5000_output="$HOME/diss/localhost5000/dirsearch/reports"

# create file if not exist
touch "$localhost5000_log" || {
    echo "Failed to create file $localhost5000_log";
    exit 1;
}

# Scan variable
scan_number=1

```



```

# increment variable
var=0

{
    echo "Dirsearch Scan Timestamps for localhost 5000"
    echo "Make sure to cross reference with scan output"
    echo "Methods: GET, POST, PUT, DELETE, PATCH"
    echo "-"
} >> "$localhost5000_log" || {
    echo "Failed to write to $localhost5000_log";
    exit 1;
}

while [ $var -lt 50 ]; do
{
    echo "DIRSEARCH"
    echo "Scan $scan_number"
    echo "GET: $(date)"
} >> "$localhost5000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:5000 --
delay=1 -m GET --subdirs /,api/,admin/,auth/,api/admin/,api/auth/
-w $HOME/api/wordlists/dummylist.txt -o
$localhost5000_output/GETscan-$scan_number.txt

{
    echo "POST: $(date)"
} >> "$localhost5000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:5000 --
delay=1 -m POST --subdirs /,api/,admin/,auth/,api/admin/,api/auth/
-w $HOME/api/wordlists/dummylist.txt -o
$localhost5000_output/POSTscan-$scan_number.txt

gql_output=$(grep -oP "graphql" "$localhost5000_output/POSTscan-
$scan_number.txt")
if [ -n "$gql_output" ]; then
    echo "GraphQL endpoint detected -- SCAN STOPPED!"
    {
        echo "Finish: $(date)"
        echo "-"
    } >> "$localhost5000_log"
    ((var++))
    ((scan_number++))
    continue
fi

{

```

```

    echo "PUT: $(date) "
} >> "$localhost5000_log"
python3 ~/di search/dirsearch.py -t 30 -u http://127.0.0.1:5000 --
delay=1 -m PUT --subdirs /,api/,admin/,auth/,api/admin/,api/auth/
-w $HOME/api/wordlists/dummylist.txt -o
$localhost5000_output/PUTscan-$scan_number.txt

{
    echo "DELETE: $(date) "
} >> "$localhost5000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:5000 --
delay=1 -m DELETE --subdirs
/,api/,admin/,auth/,api/admin/,api/auth/ -w
$HOME/api/wordlists/dummylist.txt -o
$localhost5000_output/DELETEScan-$scan_number.txt

{
    echo "PATCH: $(date) "
} >> "$localhost5000_log"
python3 ~/dirsearch/dirsearch.py -t 30 -u http://127.0.0.1:5000 --
delay=1 -m PATCH --subdirs
/,api/,admin/,auth/,api/admin/,api/auth/ -w
$HOME/api/wordlists/dummylist.txt -o
$localhost5000_output/PATCHscan-$scan_number.txt

{
    echo "Finish: $(date) "
    echo "-"
} >> "$localhost5000_log"
((var++))
((scan_number++))
done

shutdown now

```

11.8.3.4 Step 4: Schema Extraction

schema-bf.sh (Clairvoyance)

```

#!/bin/bash

# Log file
localhost5000_log="$HOME/diss/localhost5000/clairvoyance/
timestamps/schema-bruteforcing.txt"

```

```

localhost5000_output="$HOME/diss/localhost5000/clairvoyance/
output/"

# create file if not exist
touch "$localhost5000_log" || {
    echo "Failed to create file $localhost5000_log";
    exit 1;
}

touch "$localhost5000_output" || {
    echo "Failed to create file $localhost5000_output";
    exit 1;
}

# Scan variable
scan_number=1

# increment variable
var=0

{
    echo "Clairvoyance Schema Brute Forcing Timestamps for
localhost 5000"
    echo "Make sure to cross reference with scan output"
    echo "-"
    echo "Clairvoyance"
    echo "-"
} >> "$localhost5000_log" || {
    echo "Failed to write to $localhost5000_log";
    exit 1;
}

while [ $var -lt 50 ];
do
    {
        echo "Scan $scan_number"
        echo "Start: $(date +"%T.%N") "
    } >> "$localhost5000_log"

    python3 -m clairvoyance http://localhost:5000/api/graphql -p
slow -w ~/api/wordlists/dummylist.txt -o
$localhost5000_output/output-scan-$scan_number

    {
        echo "Finish: $(date +"%T.%N") "
        echo "-"
    }
}

```

```
} >> "$localhost5000_log"  
  
((var++))  
((scan_number++))  
done
```

11.9 Appendix IX: Brute-Forcing Word List

auth	examples	library
login	assets	source
posts	wp-admin	common
post	web	server
post	packages	security
signup	scripts	Source
logout	third	mysql-test
admin	components	templates
graphql	doc	bin
add-product	administrator	gitignore
products	Library	life
product	bower	img
cart	package	platform
cart-delete-item	build	config
orders	themes	includes
create-order	static	recipes
ajax	deps	res
drivers	chrome	Samples
arch	profiles	user
node	target	api
src	sql	python
vendor	images	samples
include	Godeps	dep
tests	libjava	frontend
Documentation	spec	framework
fs	libraries	content
lib	resources	contrib
wp-content	media	Casks
sites	java	Pods
sound	Assets	crypto
net	js	admin
app	README	aec
public	application	dist
core	release	site
files	libs	misc
modules	LICENSE	extensions
test	www	android
tools	external	mod
docs	contributors	gas
data	pkg	code
gcc	django	platforms
wp-includes	client	fonts
addons	firmware	backend
plugins	system	bundles
kernel	LayoutTests	ui

libstdc
storage
mm
openerp
htdocs
ld
db
linux
3rdparty
css
html
apps
Example
linux-3
project
board
ext
Examples
testsuite
t
xbmc
icons
website
engine
Resources
jni
sampleData
block
thirdparty
share
3rdParty
jdk
staging
gdb
boost
demo
documentation
plugin
Modules
concrete
sim
hardware
2007
metadata
puphpet
tensorflow
usr
theme

srcpkgs
projects
sass
example
idea
tmp
2011
Benchmark
frameworks
output
module
venv
atom
Data
webapp
3party
upload
desktop
base
TMessagesProj
features
protected
docroot
pix
dobb
cocos2d
utils
cmd
demos
svn
camel-core
Tests
sysdeps
sources
inc
views
po
cpp
org
travis
pages
old
index
main
lms
kbe
Externals
lang

dataset
ios
etc
llvm
newlib
input
vendors
install
Libraries
dependencies
01
com
repository
interpreter
virt
blog
WebContent
applications
sample
artworks
backup
env
dev
Project
validation-test
cms
linux-2
testing
raw
bundled
python3-alpha
debian
reference
models
Scripts
language
drupal
cluster
sdk
native
builtin
Tools
web-app
extras
typo3
php
hg
go

interface
cookbooks
eclipse
webapi
erpNext
aclImdb
chromium
game
v1
ThirdParty
gui
tutorials
mods
conf
nova
question
font-awesome
Android
xml
Code
actor-apps
layouts
Sources
engines
stacks
man
third-party
App
Web
database
var
Core
less
trunk
3rd
scipy
ClassPrj
hadoop-yarn-project
i18n
Packages
toolchain
sw
flask
script
Unity
hw
hadoop-common-project
skontarredhat

Server
Gemfile
Lib
appinventor
setup
wordpress
template
icon-themes
hadoop-hdfs-project
bundle
Experiment
libgo
languages
webroot
image
extern
browser
war
gradle
crashes-duplicates
datasets
Client
git
webkit
mustella
2005
Makefile
bsp
Src
testcases
mediatek
vim
software
sandbox
externals
generated
init
util
puppet
ppapi
hadoop-mapreduce-project
traces
MathJax
out
Classes
feeds
benchmarks
blocks

closure
c
Public
distribution
iOS
cross
classes
ipc
testdata
mc
1
lib-src
2004
skin
intermediates
styles
Demo
sklearn
phpmyadmin
results
cmake
mac
win32
langtools
ReactAndroid
Thirdparty
wiki
hack
de
mobile
2002
Frameworks
support
fuel
extra
sys
cache
services
objects
configs
nodejs
tags
mibs
composer
Jint
design
challenge6
locale

shell	installation	2016
npc	win-client	art
WebRoot	mathjax	crosstool
easybuild	explorer	ARMTToolChain
country	extension	Application
binutils	CHANGELOG	federation
System	6	cf
MathJax-master	angular	3
cpu	dashboard	resource
settings	train	wsgi
graphics	emacs	0
neo	Rakefile	spring-boot-autoconfigure
pimcore	home	workspace
Numix	elpa	tweet
editor	skins	exec
ash	branches	libgloss
help	tpl	gulp
google	compiler	view
C	alps	work
submodules	requirements	Website
daemon	Dependencies	grunt
SampleVideos	roms	gradlew
sass-cache	tasks	qa
shared	Common	experiments
Python	remoting	appcompat
openstack	local	Test
Plugins	spring-boot-samples	snippets
builder	iphone	docker
gitattributes	service	ndk
B2G	yii	Images
tiles	catalog	angular-1
lisp	published	Models
hotspot	Carthage	slides
rice-middleware	grade	items
2	CONTRIBUTING	ckeditor
taipower	vagrant	opendaylight
Solutions	koha-tmpl	report
presto-main	archive	libr
React	grails-app	root
solr	perl	Microsoft
ezpublish	clients	scss
libgcc	roles	model
m4	deploy	uefi
libsodium	jquery	BuildDeps
VocabularyTest	terraform	bfd
jspm	COPYING	laravel
javascript	custom	JavaScript
bootstrap	compare	libgomp

maintenance
hal
mojo
python-build
math
none
riscv
uploads
backends
javadoc
webpack
releases
class
style
boards
basic
runtime
Console
jssource
rawdata
buildroot-avr32-v3
packaging
JabbR
moodle
neutron
touch
frappe
ClickablesVsDrugbank
maps
Kernel
rest-api-spec
generators
gst
editorconfig
cc
json
front-end
sympy
ThinkPHP
youtube
Testing
wagtail
Content
targets
apdos
dotCMS
functions
XcodeClasses

front
proj
fixtures
depend
libc
gems
Chapter
wa-data
enrol
dalvik
Pod
Libs
gpu
utilities
Homeworks
asset
Projects
auth
Framework
agent
pkgs
experimental
spring-boot
openmaxims
matlab
Output
UI
guides
blinky
eZ
4
lapack-netlib
MediaBrowser
filter
libgfortran
Zend
Engine
translations
course
epan
pics
jaxws
i386-squashfs-root
R
openshift
MACOSX
os
integration

presentation
03
private
training
External
CU
windows
portal
shop
types
source4
benchmark
drv
reports
corpus
hadoop-tools
basis
depends
dotnet
pom
patches
02
mk
installer
csv
android-2
teenie
forum
mt6732
panda
chromeos
original
guava
obj
stoqs
Game
head
qtopiacore
cloud
svg
repo
sync
png
wp
uohCorpus
Docs
asterixdb
ports

builds	symfony	gdx
Build	guava-tests	erts
ArduinoAddons	flags	keepalived-vip
libcloud	WordPress	perun-web-gui
name	ffmpeg	chef
account	Backend	archdroid-icon-theme
audio	guava-gwt	min-gcc
Gruntfile	license	Sample
v2	compilers	ZXingObjCTests
book	win	imgs
playbooks	AUTHORS	gccsrc
ide	environments	tinymce
gold	aio	mezzanine
sapphire	visualizations	boot
cordova	CompareToDrugbank	toolchains
distrib	mt6592	Components
jshintrc	zlib	smileys
shopizer	default	Demos
extjs	python	ParseStarterProject
pogs	semantic	jhipster-parisjug
AllJoyn	opencascade-6	rules
labs	sky	prebuilts
intermine	ansible	rice-framework
INSTALL	ExtLibs	memcheck
subprojects	MathJax-2	servers
make	zerver	courses
Vendor	phpMyAdmin	mlbgameday
Coding	en	logs
readme	m2048	games
numpy	spring-boot-tools	ProjectSettings
pegasus	39	bindings
l10n	esb	device
schema	browser-ext	tooling
actionbarsherlock	hotel-dashboard	command
linux-4	liquibase-core	bench
5	UnrealPyEmbed	opcodes
labcodes	qt	cli
New	uClinux-dist	cygwin
invenio	DNN	meteor
gulpfile	d8	gitmodules
uportal-war	exe	Archive
Software	SVG	gm
OpenRA	tcl	dll
magento	ingress	MarsBase
integration-cli	spring-boot-actuator	MANIFEST
romil	cluster-autoscaler	dts
temp	Programming	xen
new	Top1000	oceanbase

master
known
reveal
font
Arduino
ap
AtomViewer
mingw-w64-i686-gcc
jre
buildtools
run
QtEsrc
Java
manual
AnkiDroid
controllers
election
SMITE
py
v0
migrations
controls
github
20
Doc
Formula
Magento-CE-2
Main
processing
nbproject
grobid-trainer
graf2d
dat
KERNEL
Faenza
ryu
Templates
legacy
samigo
rootfs
ext-4
manifests
2017
widgets
lucene
flamingo2-web
screenshots
howl-proto

gemfire-core
actions
Native
WebCore
libiberty
wire
Primes
Utilities
dependencies64
dubbo-admin
react
white
jaxp
documents
prototype
de4dot
black
TT
qca
host
samurai-examples
nsc
linux-lts-quantal-3
Graphics
jps
vainilla
store
stylesheets
answerkey
manpages
search
product
repos
webapps
articles
my
closure-library
Leanote
source3
configure
JuceLibraryCode
blazetest
gtk
vsprojects
cake
zh
testsrc
me

coeus-help
ruby
ShareX
addon
uitest
subjects
distancemat
log
service-loadbalancer
doxygen
pcf-elastic-runtime-1
intl
packager
activities
US
csharp
Kooboo
MTA10
codec
console
framework-res
facebook
rest
news
grub-core
hot
compat
ow
workflow
Dockerfile
leavesfish
routes
decaf
exercises
Numix-Circle
GUI
provisioning
psi4
AppKit
dev-python
manager
sdks
transport
npmignore
spring-boot-cli
sounds
dojo
UltimateAndroidNormal

programs
Symfony
mnist
regression
subdomains

UltimateAndroidGradle
API
Presentation
lua
hbase-server

groups
FreeBSD
test-suite
onvif
elements

11.10 Appendix X: Active Reconnaissance Full Data Results

Figure 1: Views-rendering Middleware Application

LOCALHOST3000 (Views Middleware)					DIRSEARCH - Init							DIRSEARCH - Auth											
SCAN	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SCAN %	HTTP Calls	ATTACK SURFACE	Endpoints	SCAN	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SCAN %	HTTP Calls	ATTACK SURFACE	Endpoints	SCAN	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SCAN %	HTTP Calls	ATTACK SURFACE	Endpoints
1	6	00:00:11.35	0	0	0	0.47368410526316	9	2	0	00:12:45	0.5	0.25	1	0.47368410526316	9	2	0	00:23:19	0.5	0.25	1	0.368421052631579	7
2	6	00:00:11.35	0	0	0	0.47368410526316	9	3	0	00:12:48	0.5	0.25	1	0.47368410526316	9	3	0	00:23:20	0.5	0.25	1	0.368421052631579	7
3	6	00:00:11.35	0	0	0	0.47368410526316	9	4	0	00:12:49	0.5	0.25	1	0.47368410526316	9	4	0	00:23:19	0.5	0.25	1	0.368421052631579	7
4	6	00:00:11.35	0	0	0	0.47368410526316	9	5	0	00:12:49	0.5	0.25	1	0.47368410526316	9	5	0	00:23:19	0.5	0.25	1	0.368421052631579	7
5	6	00:00:11.35	0	0	0	0.47368410526316	9	6	0	00:12:49	0.5	0.25	1	0.47368410526316	9	6	0	00:23:20	0.5	0.25	1	0.368421052631579	7
6	6	00:00:11.35	0	0	0	0.47368410526316	9	7	0	00:12:46	0.5	0.25	1	0.47368410526316	9	7	0	00:23:19	0.5	0.25	1	0.368421052631579	7
7	6	00:00:11.41	0	0	0	0.47368410526316	9	8	0	00:12:46	0.5	0.25	1	0.47368410526316	9	8	0	00:23:19	0.5	0.25	1	0.368421052631579	7
8	6	00:00:11.33	0	0	0	0.47368410526316	9	9	0	00:12:49	0.5	0.25	1	0.47368410526316	9	9	0	00:23:20	0.5	0.25	1	0.368421052631579	7
9	6	00:00:11.40	0	0	0	0.47368410526316	9	10	0	00:12:46	0.5	0.25	1	0.47368410526316	9	10	0	00:23:19	0.5	0.25	1	0.368421052631579	7
10	6	00:00:11.36	0	0	0	0.47368410526316	9	11	0	00:12:46	0.5	0.25	1	0.47368410526316	9	11	0	00:23:20	0.5	0.25	1	0.368421052631579	7
11	6	00:00:11.41	0	0	0	0.47368410526316	9	12	0	00:12:49	0.5	0.25	1	0.47368410526316	9	12	0	00:23:20	0.5	0.25	1	0.368421052631579	7
12	6	00:00:11.39	0	0	0	0.47368410526316	9	13	0	00:12:46	0.5	0.25	1	0.47368410526316	9	13	0	00:23:19	0.5	0.25	1	0.368421052631579	7
13	6	00:00:11.41	0	0	0	0.47368410526316	9	14	0	00:12:48	0.5	0.25	1	0.47368410526316	9	14	0	00:23:20	0.5	0.25	1	0.368421052631579	7
14	6	00:00:11.39	0	0	0	0.47368410526316	9	15	0	00:12:45	0.5	0.25	1	0.47368410526316	9	15	0	00:23:20	0.5	0.25	1	0.368421052631579	7
15	6	00:00:11.39	0	0	0	0.47368410526316	9	16	0	00:12:47	0.5	0.25	1	0.47368410526316	9	16	0	00:23:20	0.5	0.25	1	0.368421052631579	7
16	6	00:00:11.39	0	0	0	0.47368410526316	9	17	0	00:12:45	0.5	0.25	1	0.47368410526316	9	17	0	00:23:19	0.5	0.25	1	0.368421052631579	7
17	6	00:00:11.35	0	0	0	0.47368410526316	9	18	0	00:12:46	0.5	0.25	1	0.47368410526316	9	18	0	00:23:20	0.5	0.25	1	0.368421052631579	7
18	6	00:00:11.35	0	0	0	0.47368410526316	9	19	0	00:12:46	0.5	0.25	1	0.47368410526316	9	19	0	00:23:19	0.5	0.25	1	0.368421052631579	7
19	6	00:00:11.39	0	0	0	0.47368410526316	9	20	0	00:12:46	0.5	0.25	1	0.47368410526316	9	20	0	00:23:20	0.5	0.25	1	0.368421052631579	7
20	6	00:00:11.35	0	0	0	0.47368410526316	9	21	0	00:12:46	0.5	0.25	1	0.47368410526316	9	21	0	00:23:19	0.5	0.25	1	0.368421052631579	7
21	6	00:00:11.35	0	0	0	0.47368410526316	9	22	0	00:12:46	0.5	0.25	1	0.47368410526316	9	22	0	00:23:20	0.5	0.25	1	0.368421052631579	7
22	6	00:00:11.39	0	0	0	0.47368410526316	9	23	0	00:12:46	0.5	0.25	1	0.47368410526316	9	23	0	00:23:20	0.5	0.25	1	0.368421052631579	7
23	6	00:00:11.39	0	0	0	0.47368410526316	9	24	0	00:12:46	0.5	0.25	1	0.47368410526316	9	24	0	00:23:19	0.5	0.25	1	0.368421052631579	7
24	6	00:00:11.39	0	0	0	0.47368410526316	9	25	0	00:12:46	0.5	0.25	1	0.47368410526316	9	25	0	00:23:19	0.5	0.25	1	0.368421052631579	7
25	6	00:00:11.35	0	0	0	0.47368410526316	9	26	0	00:12:46	0.5	0.25	1	0.47368410526316	9	26	0	00:23:19	0.5	0.25	1	0.368421052631579	7
26	6	00:00:11.40	0	0	0	0.47368410526316	9	27	0	00:12:47	0.5	0.25	1	0.47368410526316	9	27	0	00:23:20	0.5	0.25	1	0.368421052631579	7
27	6	00:00:11.39	0	0	0	0.47368410526316	9	28	0	00:12:45	0.5	0.25	1	0.47368410526316	9	28	0	00:23:19	0.5	0.25	1	0.368421052631579	7
28	6	00:00:11.39	0	0	0	0.47368410526316	9	29	0	00:12:46	0.5	0.25	1	0.47368410526316	9	29	0	00:23:20	0.5	0.25	1	0.368421052631579	7
29	6	00:00:11.35	0	0	0	0.47368410526316	9	30	0	00:12:46	0.5	0.25	1	0.47368410526316	9	30	0	00:23:20	0.5	0.25	1	0.368421052631579	7
30	6	00:00:11.39	0	0	0	0.47368410526316	9	31	0	00:12:46	0.5	0.25	1	0.47368410526316	9	31	0	00:23:19	0.5	0.25	1	0.368421052631579	7
31	6	00:00:11.34	0	0	0	0.47368410526316	9	32	0	00:12:47	0.5	0.25	1	0.47368410526316	9	32	0	00:23:19	0.5	0.25	1	0.368421052631579	7
32	6	00:00:11.39	0	0	0	0.47368410526316	9	33	0	00:12:46	0.5	0.25	1	0.47368410526316	9	33	0	00:23:19	0.5	0.25	1	0.368421052631579	7
33	6	00:00:11.35	0	0	0	0.47368410526316	9	34	0	00:12:46	0.5	0.25	1	0.47368410526316	9	34	0	00:23:20	0.5	0.25	1	0.368421052631579	7
34	6	00:00:11.35	0	0	0	0.47368410526316	9	35	0	00:12:45	0.5	0.25	1	0.47368410526316	9	35	0	00:23:19	0.5	0.25	1	0.368421052631579	7
35	6	00:00:11.34	0	0	0	0.47368410526316	9	36	0	00:12:46	0.5	0.25	1	0.47368410526316	9	36	0	00:23:19	0.5	0.25	1	0.368421052631579	7
36	6	00:00:11.34	0	0	0	0.47368410526316	9	37	0	00:12:46	0.5	0.25	1	0.47368410526316	9	37	0	00:23:20	0.5	0.25	1	0.368421052631579	7
37	6	00:00:11.39	0	0	0	0.47368410526316	9	38	0	00:12:46	0.5	0.25	1	0.47368410526316	9	38	0	00:23:19	0.5	0.25	1	0.368421052631579	7
38	6	00:00:11.34	0	0	0	0.47368410526316	9	39	0	00:12:46	0.5	0.25	1	0.47368410526316	9	39	0	00:23:19	0.5	0.25	1	0.368421052631579	7
39	6	00:00:11.39	0	0	0	0.47368410526316	9	40	0	00:12:46	0.5	0.25	1	0.47368410526316	9	40	0	00:23:19	0.5	0.25	1	0.368421052631579	7
40	6	00:00:11.34	0	0	0	0.47368410526316	9	41	0	00:12:46	0.5	0.25	1	0.47368410526316	9	41	0	00:23:19	0.5	0.25	1	0.368421052631579	7
41	6	00:00:11.39	0	0	0	0.47368410526316	9	42	0	00:12:45	0.5	0.25	1	0.47368410526316	9	42	0	00:23:20	0.5	0.25	1	0.368421052631579	7
42	6	00:00:11.35	0	0	0	0.47368410526316	9	43	0	00:12:46	0.5	0.25	1	0.47368410526316	9	43	0	00:23:19	0.5	0.25	1	0.368421052631579	7
43	6	00:00:11.39	0	0	0	0.47368410526316	9	44	0	00:12:46	0.5	0.25	1	0.47368410526316	9	44	0	00:23:19	0.5	0.25	1	0.368421052631579	7
44	6	00:00:11.34	0	0	0	0.47368410526316	9	45	0	00:12:46	0.5	0.25	1	0.47368410526316	9	45	0	00:23:19	0.5	0.25	1	0.368421052631579	7
45	6	00:00:11.31	0	0	0	0.47368410526316	9	46	0	00:12:46	0.5	0.25	1	0.47368410526316	9	46	0	00:23:19	0.5	0.25	1	0.368421052631579	7
46	6	00:00:11.39	0	0	0	0.47368410526316	9	47	0	00:12:46	0.5	0.25	1	0.47368410526316	9	47	0	00:23:19	0.5	0.25	1	0.368421052631579	7
47	6	00:00:11.34	0	0	0	0.47368410526316	9	48	0	00:12:46	0.5	0.25	1	0.47368410526316	9	48	0	00:23:19	0.5	0.25	1	0.368421052631579	7
48	6	00:00:11.34	0	0	0	0.47368410526316	9	49	0	00:12:46	0.5	0.25	1	0.47368410526316	9	49	0	00:23:20	0.5	0.25	1	0.368421052631579	7
49	6	00:00:11.39	0	0	0	0.47368410526316	9	50	0	00:12:45	0.5	0.25	1	0.47368410526316	9	50	0	00:23:19	0.5	0.25	1	0.368421052631579	7
50	6	00:00:11.39	0	0	0	0.47368410526316	9	AVERAGE	6	00:12:46.06	0.5	0.25	1	0.47368410526316	9	AVERAGE	6	00:23:19.44	0.5	0.25	1	0.368421052631579	7

AVERAGE	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.
LOCALHOST3000	6	00:12:46.06	80%	25%	47.37%	9

AVERAGE	SYS INFO	TIMESTAMP	HTTP ACT. %	HTTP SC. %	ATT. SURF.	SCANS PER.
LOCALHOST4000	7	01:14:18.54	100.00%	100.00%	100.00%	3

Figure 3: GraphQL API

LOCALHOST5000 (GraphQL API)

SCAN	SYS INFO	TIMESTAMP
1	7	00:00:11.37
2	7	00:00:11.37
3	7	00:00:11.37
4	7	00:00:11.37
5	7	00:00:11.37
6	7	00:00:11.37
7	7	00:00:11.37
8	7	00:00:11.37
9	7	00:00:11.37
10	7	00:00:11.37
11	7	00:00:11.37
12	7	00:00:11.37
13	7	00:00:11.37
14	7	00:00:11.37
15	7	00:00:11.37
16	7	00:00:11.37
17	7	00:00:11.37
18	7	00:00:11.37
19	7	00:00:11.37
20	7	00:00:11.37
21	7	00:00:11.37
22	7	00:00:11.37
23	7	00:00:11.37
24	7	00:00:11.37
25	7	00:00:11.37
26	7	00:00:11.37
27	7	00:00:11.37
28	7	00:00:11.37
29	7	00:00:11.37
30	7	00:00:11.37
31	7	00:00:11.37
32	7	00:00:11.37
33	7	00:00:11.37
34	7	00:00:11.37
35	7	00:00:11.37
36	7	00:00:11.37
37	7	00:00:11.37
38	7	00:00:11.37
39	7	00:00:11.37
40	7	00:00:11.37
41	7	00:00:11.37
42	7	00:00:11.37
43	7	00:00:11.37
44	7	00:00:11.37
45	7	00:00:11.37
46	7	00:00:11.37
47	7	00:00:11.37
48	7	00:00:11.37
49	7	00:00:11.37
50	7	00:00:11.37
AVERAGE	7	00:00:11.37

SCAN	SYS INFO	TIMESTAMP	HTTP Req %	HTTP %	HTTP Calls	ATTACK SURFACE	Endpoints
1	0	00:11:00	1	1	2	0	0
2	0	00:11:00	1	1	2	0	0
3	0	00:11:00	1	1	2	0	0
4	0	00:11:00	1	1	2	0	0
5	0	00:11:00	1	1	2	0	0
6	0	00:11:00	1	1	2	0	0
7	0	00:11:00	1	1	2	0	0
8	0	00:11:00	1	1	2	0	0
9	0	00:11:00	1	1	2	0	0
10	0	00:11:00	1	1	2	0	0
11	0	00:11:00	1	1	2	0	0
12	0	00:11:00	1	1	2	0	0
13	0	00:11:00	1	1	2	0	0
14	0	00:11:00	1	1	2	0	0
15	0	00:11:00	1	1	2	0	0
16	0	00:11:00	1	1	2	0	0
17	0	00:11:00	1	1	2	0	0
18	0	00:11:00	1	1	2	0	0
19	0	00:11:00	1	1	2	0	0
20	0	00:11:00	1	1	2	0	0
21	0	00:11:00	1	1	2	0	0
22	0	00:11:00	1	1	2	0	0
23	0	00:11:00	1	1	2	0	0
24	0	00:11:00	1	1	2	0	0
25	0	00:11:00	1	1	2	0	0
26	0	00:11:00	1	1	2	0	0
27	0	00:11:00	1	1	2	0	0
28	0	00:11:00	1	1	2	0	0
29	0	00:11:00	1	1	2	0	0
30	0	00:11:00	1	1	2	0	0
31	0	00:11:00	1	1	2	0	0
32	0	00:11:00	1	1	2	0	0
33	0	00:11:00	1	1	2	0	0
34	0	00:11:00	1	1	2	0	0
35	0	00:11:00	1	1	2	0	0
36	0	00:11:00	1	1	2	0	0
37	0	00:11:00	1	1	2	0	0
38	0	00:11:00	1	1	2	0	0
39	0	00:11:00	1	1	2	0	0
40	0	00:11:00	1	1	2	0	0
41	0	00:11:00	1	1	2	0	0
42	0	00:11:00	1	1	2	0	0
43	0	00:11:00	1	1	2	0	0
44	0	00:11:00	1	1	2	0	0
45	0	00:11:00	1	1	2	0	0
46	0	00:11:00	1	1	2	0	0
47	0	00:11:00	1	1	2	0	0
48	0	00:11:00	1	1	2	0	0
49	0	00:11:00	1	1	2	0	0
50	0	00:11:00	1	1	2	0	0
AVERAGE	0	00:11:00	100.00%	100.00%	2	0.00%	0

AVERAGE	SYS INFO	TIMESTAMP	HTTP ACCT %	HTTP SC %	ATT SURF	SCANS PER
LOCALHOST5000	9	00:11:37.00000	100.00%	100.00%	0.00%	5

SCAN	SYS INFO	TIMESTAMP
1	2	00:00:00.0050
2	2	00:00:00.0050
3	2	00:00:00.0050
4	2	00:00:00.0050
5	2	00:00:00.0050
6	2	00:00:00.0050
7	2	00:00:00.0050
8	2	00:00:00.0050
9	2	00:00:00.0050
10	2	00:00:00.0050
11	2	00:00:00.0050
12	2	00:00:00.0050
13	2	00:00:00.0050
14	2	00:00:00.0050
15	2	00:00:00.0050
16	2	00:00:00.0050
17	2	00:00:00.0050
18	2	00:00:00.0050
19	2	00:00:00.0050
20	2	00:00:00.0050
21	2	00:00:00.0050
22	2	00:00:00.0050
23	2	00:00:00.0050
24	2	00:00:00.0050
25	2	00:00:00.0050
26	2	00:00:00.0050
27	2	00:00:00.0050
28	2	00:00:00.0050
29	2	00:00:00.0050
30	2	00:00:00.0050
31	2	00:00:00.0050
32	2	00:00:00.0050
33	2	00:00:00.0050
34	2	00:00:00.0050
35	2	00:00:00.0050
36	2	00:00:00.0050
37	2	00:00:00.0050
38	2	00:00:00.0050
39	2	00:00:00.0050
40	2	00:00:00.0050
41	2	00:00:00.0050
42	2	00:00:00.0050
43	2	00:00:00.0050
44	2	00:00:00.0050
45	2	00:00:00.0050
46	2	00:00:00.0050
47	2	00:00:00.0050
48	2	00:00:00.0050
49	2	00:00:00.0050
50	2	00:00:00.0050
AVERAGE	2	00:00:00.0050

SCAN	SYS INFO	TIMESTAMP	ENDPOINTS
1	1	00:00:00.0750	0
2	1	00:00:00.0750	0
3	1	00:00:00.0750	0
4	1	00:00:00.0750	0
5	1	00:00:00.0750	0
6	1	00:00:00.0750	0
7	1	00:00:00.0750	0
8	1	00:00:00.0750	0
9	1	00:00:00.0750	0
10	1	00:00:00.0750	0
11	1	00:00:00.0750	0
12	1	00:00:00.0750	0
13	1	00:00:00.0750	0
14	1	00:00:00.0750	0
15	1	00:00:00.0750	0
16	1	00:00:00.0750	0
17	1	00:00:00.0750	0
18	1	00:00:00.0750	0
19	1	00:00:00.0750	0
20	1	00:00:00.0750	0
21	1	00:00:00.0750	0
22	1	00:00:00.0750	0
23	1	00:00:00.0750	0
24	1	00:00:00.0750	0
25	1	00:00:00.0750	0
26	1	00:00:00.0750	0
27	1	00:00:00.0750	0
28	1	00:00:00.0750	0
29	1	00:00:00.0750	0
30	1	00:00:00.0750	0
31	1	00:00:00.0750	0
32	1	00:00:00.0750	0
33	1	00:00:00.0750	0
34	1	00:00:00.0750	0
35	1	00:00:00.0750	0
36	1	00:00:00.0750	0
37	1	00:00:00.0750	0
38	1	00:00:00.0750	0
39	1	00:00:00.0750	0
40	1	00:00:00.0750	0
41	1	00:00:00.0750	0
42	1	00:00:00.0750	0
43	1	00:00:00.0750	0
44	1	00:00:00.0750	0
45	1	00:00:00.0750	0
46	1	00:00:00.0750	0
47	1	00:00:00.0750	0
48	1	00:00:00.0750	0
49	1	00:00:00.0750	0
50	1	00:00:00.0750	0
AVERAGE	1	00:00:00.0750	0

SCAN	Endpoints	TIME STAMP
1	0	00:00:00.2011
2	0	00:00:00.1986
3	0	00:00:00.1961
4	0	00:00:00.1936
5	0	00:00:00.1911
6	0	00:00:00.1886
7	0	00:00:00.1861
8	0	00:00:00.1836
9	0	00:00:00.1811
10	0	00:00:00.1786
11	0	00:00:00.1761
12	0	00:00:00.1736
13	0	00:00:00.1711
14	0	00:00:00.1686
15	0	00:00:00.1661
16	0	00:00:00.1636
17	0	00:00:00.1611
18	0	00:00:00.1586
19	0	00:00:00.1561
20	0	00:00:00.1536
21	0	00:00:00.1511
22	0	00:00:00.1486
23	0	00:00:00.1461
24	0	00:00:00.1436
25	0	00:00:00.1411
26	0	00:00:00.1386
27	0	00:00:00.1361
28	0	00:00:00.1336
29	0	00:00:00.1311
30	0	00:00:00.1286
31	0	00:00:00.1261
32	0	00:00:00.1236
33	0	00:00:00.1211
34	0	00:00:00.1186
35	0	00:00:00.1161
36	0	00:00:00.1136
37	0	00:00:00.1111
38	0	00:00:00.1086
39	0	00:00:00.1061
40	0	00:00:00.1036
41	0	00:00:00.1011
42	0	00:00:00.0986
43	0	00:00:00.0961
44	0	00:00:00.0936
45	0	00:00:00.0911
46	0	00:00:00.0886
47	0	00:00:00.0861
48	0	00:00:00.0836
49	0	00:00:00.0811
50	0	00:00:00.0786
AVERAGE	0.00%	00:00:00.1000

Week 14 (10/06/2024)

Introduction

The following was completed during week 13:

- Literature review
 - Section: 2.1
 - Topic: REST API design principles
 - Word count: 1,210 words (in-line references included)

Progress Findings

- Literature review
 - this section outlines all of the major studies of REST API design principles I could find
 - major trends and data findings are compared with security in mind
 - two conjectures were made which could aid the research question

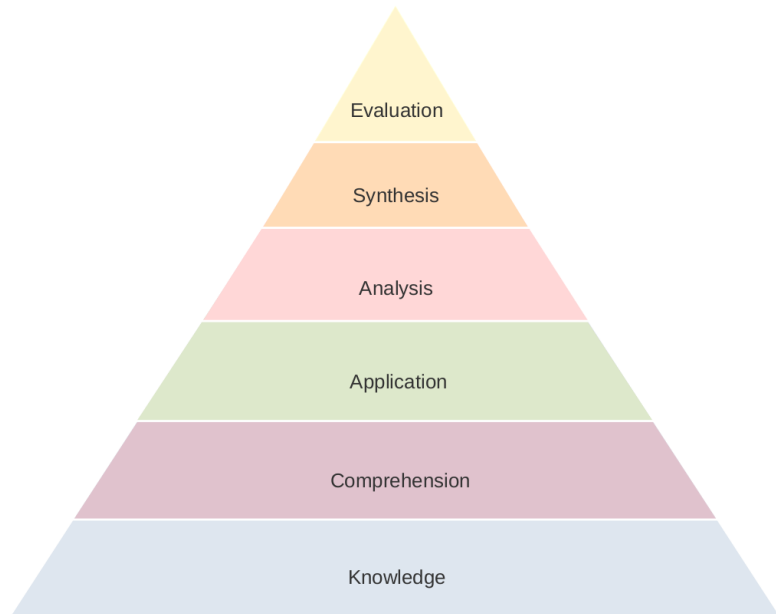
Plan for the Week

- Literature review
 - Section 2.2
 - Topic: GraphQL design principles
 - Word count: ~800 (younger technology, not so much research)
 - Section 2.3
 - Topic: REST vs GraphQL
 - Word count: ~1000 - 1200

11.12 Appendix XII: Bloom's Taxonomy Worksheet

How to use this reference:

- Read through each criteria
- Compare your references / notes / understanding to the requirements
- Make a spreadsheet like the below substituting your understanding for the criteria text underneath the headings
- Add more information as the literature review / project progresses
- Make note of knowledge gaps / inconsistencies as they appear



Knowledge	Comprehension	Application	Analysis	Synthesis	Evaluation
<p><u>1.10 Knowledge of specifics</u> The recall of specific information and isolatable bits of information.</p> <p><u>1.11 Knowledge of terminology</u> Knowledge of the referents for specific verbal and non-verbal symbols.</p> <p><u>1.12 Knowledge of specific facts</u> Knowledge of dates, events, persons, places, sources of info, etc.</p> <p><u>1.20 Knowledge of ways and means of dealing with specifics</u> Knowledge of the ways of organising, studying, judging, and criticising ideas and phenomena.</p> <p><u>1.21 Knowledge of conventions</u> Knowledge of characteristic ways of treating and presenting ideas and phenomena.</p> <p><u>1.22 Knowledge of trends and sequences</u> Knowledge of the processes, directions, and movements of phenomenon with respect to time.</p> <p><u>1.23 Knowledge of classifications and categories</u> Knowledge of the classes, sets, divisions, and arrangements which are regarded as fundamental or useful for a given subject field, purpose, argument, or problem.</p> <p><u>1.24 Knowledge of criteria</u> Knowledge of the criteria by which facts, principles, opinions, and conduct are tested or judged.</p> <p><u>1.25 Knowledge of methodology</u> Knowledge of the methods of inquiry, techniques, and procedures employed in a particular subject field as well as those employed in investigating particular problems and</p>	<p><u>2.10 Translation</u> Translation behaviour: Occupies a transitional position between the behaviours classified under the category of knowledge and types of behaviour under the headings interpretation, extrapolation, analysis, synthesis, application, evaluation.</p> <p>Is dependent on the prerequisites of knowledge.</p> <p><u>2.20 Interpretation</u> To interpret communication the reader must: Be able to translate each of the major parts of it: words, phrases, various representational devices. Comprehend the relationships between its various parts, reorder and rearrange such in their minds. Secure a total view of what the communication contains Relate it to their own fund of experiences and ideas.</p> <p><u>2.30 Extrapolation</u> The writer should state what they believe to be the truth of the matter and some consequences of it. Should be exhaustive for criticality and should go beyond the limits set by topic of communication and/or the writer themselves. Requires ability to: Translate, as well as interpret the document, and to extend the trends or tendencies beyond the given data and findings of the document. Determine implications, consequences, corollaries, effects, etc. in accordance with the conditions as literally described in the original communication.</p>	<p>The whole cognitive domain of the taxonomy is arranged in a hierarchy: Each classification within it demands the skills / abilities which are lower in the classification order. To apply something: Requires comprehension of the method, theory, principle, or abstraction applied. Distinctions between the two: <u>First way</u> Comprehension → know an abstraction well enough to demonstrate its use. Application → know which abstraction to use without prompting when faced with a question. <u>Second way</u> A problem is presented. Step 1: is perceived as unfamiliar or familiar. Step 2: if unfamiliar, use familiar elements to restructure problem into familiar context. If familiar, there is still some restructuring to make resemblance to familiar model more complete. Step 3: classification of problem as familiar in type. Step 4: selection of abstraction (theory, principle, idea, method) suitable to problem type. Step 5: use of abstraction to solve problem. Step 6: solution to problem.</p>	<p>Emphasises the breakdown of the material into its constituent parts. Detects the relationships of the parts and of the way they are organised. <u>4.10 Analysis of elements</u> A communication may be conceived as composed of a large number of elements. Can be explicit: hypotheses, conclusions. Can be implicit: unstated assumptions by the writer, the nature or function of a statement in the communication → fact, value, or intent. Must be able to tease them out. <u>4.20 Analysis of relationships</u> Determine the major relationship among the elements and the various parts of the communication. Explicit level: relationship of hypotheses to the evidence, and conclusions and the hypotheses as well as evidence. Also the relationship among the different types of evidence presented Implicit level: analysis of a communication into parts which are essential, or which form the main thesis as contrasted with those parts or elements which may expand, develop, support the thesis. <u>4.30 Analysis of organisational principles</u> An even more complex and difficult level: the task of analysing the structure and organisation of a communication. This can discern the purpose, point of view, attitude, or general conception the author has of the field. The analysis of the form, pattern, structure, organisation of arguments, evidence, or other elements around these can help in the comprehension / evaluation of the entire communication. Is often impossible to make an</p>	<p>The putting together of elements and parts as to form a whole. Works with elements, parts, etc., combining them in such a way as to constitute a pattern or structure not clearly there before. Generally involves a recombination of parts of previous experience with new material, restructured into a new, well-integrated whole. Requires creative behaviour on the part of the learner within the limits set by particular problems, materials, or some theoretical / methodological framework. <u>5.10 Production of a unique communication</u> Objectives in which primary emphasis is upon communication – getting ideas, feelings, experiences across to others. Controlling/limiting factors: The kinds of effects to be achieved The nature of the audience in whom the effects are to be achieved. The medium through which the student expresses themselves. The particular ideas / experiences chosen to draw upon or communicate. “Effects” → the response or change in response desired in some audience, with such outcomes as the following: The acquisition of information the understanding of an idea, point of view, etc. The acceptance of an idea, point of view, etc. Motivation to carry out a purpose that the author has in mind Change an attitude or belief The creation of a mood, or feeling enjoyment or emotional satisfaction. Therefore, the nature of the audience the student addresses is often crucial in determining what the student does</p>	<p>The making of judgments about the value of ideas, works, solutions, methods, materials, etc. Involves use of criteria, standards for appraising the extent to which particulars are accurate, effective, economical, or satisfying. Can be quantitative or qualitative. Involves some combination of all the behaviours that come before it, and therefore doesn't necessarily come last. <u>6.10 Judgments in terms of internal evidence</u> Internal judgments → concerned with tests of accuracy of work as judged by consistency, logical accuracy, and the absence of internal flaws. <u>6.11 Judgments in terms of external criteria</u> External standards → required if a work is to be appraised. Is derived from a consideration of the ends to be served and the appropriateness of specific names for achieving these ends. Primarily based on considerations of efficiency, economy, utility of specific means for particular ends. Must use a criteria which is regarded as appropriate for members of the class of phenomena being judged (i.e. in terms of standards of excellence / effectiveness commonly used in the field or in comparison of particular phenomena within the field).</p>

<p>phenomena.</p> <p><u>1.30 Knowledge of the universals and abstractions in a field</u></p> <p>Knowledge of the major ideas, schemas, and patterns by which phenomena and ideas are organised.</p> <p><u>1.31 Knowledge of principles and generalisations</u></p> <p>Knowledge of particular abstractions which summarise observations and phenomena.</p> <p><u>1.32 Knowledge of theories and structures</u></p> <p>Knowledge of the body of principles and generalisations together with their interrelationships which present a clear, rounded, and systematic view of a complex phenomenon, problem, or field.</p>			<p>evaluation until this has been done.</p>	<p>→ must have a specific audience.</p> <p><u>5.20 Production of a plan, or proposed set of operations</u></p> <p>Objectives that fall into this subcategory generally aim at the production of a plan of operations.</p> <p>The production of the plan often constitutes the act of synthesis.</p> <p>The product, plan, task must satisfy the requirements of the task, usually as specifications or data to be taken into account.</p> <p>Student is encouraged to put their own ideas into the product, apart from other considerations.</p> <p><u>5.30 Derivation of a set of abstract relationships</u></p> <p>Objectives that require the student to produce, derive a set of abstract relations.</p> <p>Two task types:</p> <p><u>First type</u></p> <p>Those in which the student begins with concrete data / phenomena and which they must deduce other propositions or relations (induction).</p> <p>May take the form of classifying certain phenomena by accounting for the relations existing among a range of phenomena (e.g. the periodic table).</p> <p>May also take the form of explaining certain phenomena.</p> <p>Must formulate a hypothesis that will adequately account for a wide range of seemingly interrelated phenomena which fit the facts be internally consistent (i.e. free from logical contradictions).</p> <p><u>Second type</u></p> <p>Those in which the student begins with some basic propositions or other symbolic representations and from which he must deduce other propositions or relations (deduction).</p> <p>Little emphasis on developing a classification scheme.</p> <p>Begins with abstract symbols, propositions, etc., rather than concrete data.</p> <p>Must move these symbolic representations to deductions that can be reasonably made.</p> <p>Student would operate within some theoretical framework, and must reason in terms of it → rigorous object criteria which the product synthesis must meet; no subjective standards.</p>	
--	--	--	---	---	--

References

Bloom, B. S. et al. (1956) *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook 1: Cognitive Domain*. Longmans. London, WI, USA.