A Gamified Cybersecurity Learning Platform for Enhancing Beginner Engagement and Awareness



Muhammad Sanwal Noor

09-242242-005

BAHRIA UNIVERSITY ISLAMABAD

A Gamified Cybersecurity Learning Platform for Enhancing Beginner Engagement and Awareness

Muhammad Sanwal Noor
09-242242-005

A thesis submitted in fulfilment
of the requirements for the award
of the degree of Master of
Science (Computer Engineering)

Department of Computer Engineering

BAHRIA UNIVERSITY ISLAMABAD

AUGUST 2026

© Muhammad Sanwal Noor, 2026

MS-13
Thesis Completion Certificate

Student Name: **Muhammad Sanwal Noor** Registration Number: **71579**

Program of Study: **Master of Science in Computer Engineering**

Thesis Title: **A Gamified Cybersecurity Learning Platform for Enhancing Beginner Engagement and Awareness**

It is to certify that the above student's thesis has been completed to my satisfaction and, to my belief, its standard is appropriate for submission for evaluation. I have also conducted a plagiarism test of this thesis using HEC-prescribed software and found the similarity index at 10% that is within the permissible limit set by the HEC for MS/MPhil/PhD. I have also found the thesis in a format recognized by Bahria University for MS/MPhil/PhD theses.

Principal Supervisor's Signature**:** _____

Principal Supervisor's Name: Dr. Syed Khawar Hussain

August 20, 2026

MS-14A
Author's Declaration

I, **Muhammad Sanwal Noor** hereby state that my MS thesis titled **"A Gamified Cybersecurity Learning Platform for Enhancing Beginner Engagement and Awareness"** is my own work and has not been submitted previously by me for taking any degree from **"Bahria University, Islamabad"** or anywhere else in the country or world.

At any time if my statement is found to be incorrect, even after my graduation, the university has the right to withdraw or cancel my MS degree.

_____

Muhammad Sanwal Noor
09-242242-005
August 20, 2026

MS-14B
Plagiarism Undertaking

I, **Muhammad Sanwal Noor**, solemnly declare that the research work presented in the thesis titled **"A Gamified Cybersecurity Learning Platform for Enhancing Beginner Engagement and Awareness"** is solely my own research work with no significant contribution from any other person. Small contributions or help, wherever taken, have been duly acknowledged, and the complete thesis has been written by me.

I understand the zero-tolerance policy of Bahria University and the Higher Education Commission of Pakistan towards plagiarism. Therefore, I, as the author of the above-titled thesis, declare that no portion of my thesis has been plagiarized and that any material used has been properly referenced and cited.

I undertake that if I am found guilty of any formal plagiarism in the above-titled thesis, even after the award of the MS degree, the university reserves the right to withdraw or revoke my MS degree, and that HEC and the university have the right to publish my name on the HEC/University website on which the names of students who submitted plagiarized theses are placed.

_____

Muhammad Sanwal Noor
09-242242-005
August 20,2026

DEDICATION

This thesis is dedicated to my parents for their unwavering love and prayers, my siblings for their constant support and encouragement, and my friends for their companionship, understanding, and motivation throughout this journey.

Acknowledgments

I thank Almighty Allah for all His blessings and providing me strength and potential-to-complete-my-thesis.

I would like to thank my supervisor, Dr. Syed Khawar Hussain Shah, for the direction, constructive criticism, and encouragement provided throughout the course of this research. Their feedback shaped both the scope of the project and the rigor with which it was carried out.

I also wish to thank the faculty and staff of the Department of Computer Engineering at Bahria University, Islamabad, for the academic environment and resources that supported this work, and my fellow students who tested early versions of the platform and offered candid feedback. Finally, I thank my family for their unwavering support during my studies.

Abstract

Cybersecurity education faces a persistent gap between conceptual knowledge and the applied, decision-making skills that practitioners actually need. Conventional lectures and multiple-choice assessments rarely place learners in the position of investigating an incident, reasoning about evidence, and choosing a defensible response under realistic constraints. This thesis presents CyberQuest, a gamified and story-driven web platform designed to develop practical cybersecurity skills through a sequence of narrative missions that place the learner inside plausible security scenarios such as an intrusion into a smart-city power grid, a compromised smart-home network, a phishing campaign, weak-credential exploitation, an attack on an autonomous vehicle, and a multi-vector distributed denial-of-service (DDoS) incident.

The platform is implemented as an ASP.NET Core 8 Model-View-Controller (MVC) application backed by a Microsoft SQL Server relational database, in which all mission content, progression rules, achievements, and learner telemetry are stored rather than hard-coded. Each mission is decomposed into ordered stages that the learner clears by investigating evidence through a simulated command-line terminal and submitting a captured “flag.” A progression system awards experience points, advances the learner through named levels, and unlocks achievement badges, while a progressive hint mechanism and a lightweight adaptive-difficulty component aim to keep learners within a productive challenge range. A global leaderboard and personal dashboard provide feedback and social motivation.

The system was developed iteratively and validated through functional testing of its authentication, mission-play, scoring, and persistence workflows, together with a structured walkthrough of each mission to confirm that its intended learning outcome is reachable. The resulting platform demonstrates that a database-driven, gamified design can deliver a coherent, extensible, and engaging environment for hands-on cybersecurity practice, and it provides a foundation on which richer adaptive and analytics-driven features can be built in future work.

Keywords: cybersecurity education, gamification, serious games, capture-the-flag, adaptive learning, ASP.NET Core, web application, DDoS defens

Contents

DEDICATION vii

Acknowledgments viii

Abstract ix

List of Tables xvii

Acronyms and Abbreviations xviii

1. Introduction 1

1.1 Background and Context 1

1.2 Motivation and Problem Description 2

1.3 Problem Statement 4

1.4 Research Objectives 4

1.5 Research Questions 5

1.6 Scope of the Work 6

1.7 Contributions 6

1.8 Significance of the Study 7

1.9 Thesis Organization 7

2. Literature Review 8

2.1 Pedagogical Foundations 8

2.2 Gamification and Serious Games 9

2.3 Capture-the-Flag and Hands-on Security Platforms 9

2.4 Adaptive and Personalized Learning 11

2.5 Standards and Frameworks Informing Content 11

2.6 Comparison with Existing Approaches 12

2.7 Motivation and Engagement Theory 12

2.8 Scaffolding and the Zone of Proximal Development 14

2.9 Assessment in Skill-Based Learning 14

2.10 Systematic Review of Recent Literature (2024–2025) 14

2.11 Research Gap 18

2.12 Chapter Summary 19

Background Technologies and Concepts 21
3.1 The Model-View-Controller Pattern 21
3.2 Object-Relational Mapping 22
3.3 Password Security and Adaptive Hashing 22
3.4 Sessions and Authentication State 23
3.5 Cybersecurity Concepts in the Mission Content 23
3.6 Common Web Application Vulnerabilities 25
3.7 Software Testing Concepts 25
3.8 Chapter Summary 26
Requirements Analysis 27
4.1 Stakeholders and Elicitation 27
4.2 Functional Requirements 28
4.2.1 Account and Authentication 28
4.2.2 Mission Play 29
4.2.3 Progression and Rewards 30
4.2.4 Hints and Adaptivity 30
4.2.5 Dashboard and Reporting 31
4.3 Non-Functional Requirements 31
4.3.1 Performance 31
4.3.2 Security 31
4.3.3 Usability 32
4.3.4 Reliability, Maintainability, Portability 32
4.4 System Constraints and Assumptions 33
4.4.1 Design Constraints 33
4.4.2 Assumptions 33
4.5 Use-Case Model 34
4.6 Detailed Use-Case Specifications 35
4.7 Data Requirements 36
4.8 Requirements Traceability 37
4.9 Chapter Summary 38
System Design 39

5.1 Development Approach 39
5.2 System Architecture 40
5.2.1 Rationale for the Layered Architecture 42
5.3 Component Design 43
5.4 Actors and Use Cases 44
5.5 Mission and Progression Model 45
5.5.1 Progression Economy 45
5.5.2 Mission Lifecycle 47
5.5.3 Progressive Hints and Adaptive Difficulty 47
5.6 Data Flow 49
5.7 Database Design 50
5.7.1 Schema Design Rationale 51
5.7.2 Indexing and Query Efficiency 52
5.8 Class Design 52
5.9 Security Architecture 53
5.10 User Interface and Interaction Design 54
5.11 Deployment Model 55
5.12 Model Consistency and Correspondence with the Implementation 55
5.12.1 Mutual Consistency of the Models 55
5.12.2 Correspondence with the Implementation 56
5.13 Chapter Summary 57
Detailed Mission Design 58
6.1 Design Principles Common to All Missions 58
6.2 Mission 1: Smart City Power Grid Hack 59
6.3 Mission 2: Smart Home IoT Hack 60
6.4 Mission 3: Phishing Awareness Challenge 61
6.5 Mission 4: Social Media Password Guessing 61
6.6 Mission 5: Self-Driving Car Hijack 62
6.7 Mission 6: DDoS Defense Simulation 63
6.8 Coverage and Progression Rationale 65
6.8.1 A Rubric for Difficulty Levels 65

6.9 Chapter Summary 67
Implementation 68
7.1 Development Environment 68
7.2 Technology Stack 68
7.3 Application Structure 69
7.4 Persistence and the Data Context 71
7.5 Key Workflows 72
7.5.1 Authentication 72
7.5.2 Playing a Mission and Submitting a Flag 73
7.5.3 Access Control: Preventing Access to Locked Missions 74
7.5.4 Hints and Adaptive Difficulty 75
7.6 User Interface 76
7.7 The View Layer 79
7.8 Client-Side Terminal 80
7.9 Configuration and Connection Management 82
7.10 Database Schema and Seeding 82
7.10.1 The Seeding Procedure and Its Reproducibility 83
7.11 Progression and Reward Implementation 84
7.12 Mission Loading and Rendering 84
7.13 Dashboard and Leaderboard Implementation 85
7.14 Error Handling and Robustness 85
7.15 Security Considerations 86
7.16 Chapter Summary 87
Results and Evaluation 88
8.1 Evaluation Methodology 88
8.2 Mission Catalog 89
8.3 Functional Testing 89
8.4 Learning-Outcome Reachability 91
8.5 Case Study: The DDoS Defense Mission 91
8.6 Detailed Test Cases 93
8.7 Evaluation of Non-Functional Qualities 94

8.8 Demonstration of Extensibility 94
8.9 Discussion of Results 95
Discussion 96
9.1 Interpretation of Findings 96
9.2 Strengths of the Approach 96
9.3 Challenges Encountered 97
9.4 Threats to Validity 98
9.5 Lessons Learned 98
9.6 Detailed Comparison with Related Systems 99
9.7 Generalizability of the Approach 99
9.8 Ethical Considerations 100
9.9 Limitations 101
Conclusion and Future Work 102
10.1 Conclusion 102
10.2 Revisiting the Objectives 103
10.3 Summary of Answers to Research Questions 103
10.4 Future Work 103
10.5 Concluding Remarks 105
Supporting Code and Schema Listings 107
A.1 Database Context 107
A.2 Registration Handler 108
A.3 Flag Submission Handler 109
A.4 Schema Creation Script 111
A.5 Leaderboard Query 112
Mission Content Summary 113
B.1 Smart City Power Grid Hack 113
B.2 Smart Home IoT Hack 113
B.3 Phishing Awareness Challenge 113
B.4 Social Media Password Guessing 113
B.5 Self-Driving Car Hijack 114
B.6 DDoS Defense Simulation 114

Glossary of Key Terms 115
Glossary of Terms 117
References 123

Table of Figures

Figure 1 Gamification elements mapped to the motivational functions they serve 13
Figure 2 Decomposition of system requirements into functional and non-functional categories 27
Figure 3 Use cases of the account and authentication subsystem 34
Figure 4 Use cases of the mission-play subsystem 35
Figure 5 The iterative and incremental development cycle followed in the project 39
Figure 6 Project development timeline across the major work packages 40
Figure 7 Layered system architecture of the CyberQuest platform 41
Figure 8 ASP.NET Core MVC request-processing pipeline 43
Figure 9 Component diagram showing the principal software components and their dependencies 44
Figure 10 Overall use-case diagram showing both actors and the main system interactions 45
Figure 11 Mission unlock progression and the experience-point economy 46
Figure 12 Level thresholds relative to the cumulative experience available from the mission catalog 46
Figure 13 State diagram of the mission lifecycle for a learner 47
Figure 14 Progressive hint scaffolding and the experience trade-off within a stage 48
Figure 15 Rule-based adaptive-difficulty recommendation from observed performance signals 48
Figure 16 Level-1 data-flow diagram for the CyberQuest platform 49
Figure 17 Entity-relationship diagram of the core CyberQuest database 50
Figure 18 Complete physical database schema showing all twenty-two tables 51
Figure 19 Class diagram of the principal domain, persistence, and controller classes 53
Figure 20 Deployment diagram showing the client, application, and database nodes 55
Figure 21 Directory structure of the CyberQuest project 70
Figure 22 Sequence diagram for user registration and authentication 72
Figure 23 Sequence diagram for playing a mission stage and submitting a flag 73
Figure 24 Activity diagram for a stage attempt, hints, and adaptive difficulty 75
Figure 25 Screenshot of the login screen: the combined login and registration panel 76
Figure 26 Screenshot of the mission-catalog screen: mission cards with domain, difficulty, reward, completion state, and locked missions 77
Figure 27 Screenshot of the mission-play screen: the stage brief and hints, the simulated terminal, and the flag-submission control 77
Figure 28 Screenshot of the dashboard screen: experience, rank, completed missions, badges, per-mission progress, and streak 78
Figure 29 Screenshot of the leaderboard screen: the podium and the ranked table with the current learner highlighted 78
Figure 30 Screenshot of the profile screen: level progression, badge showcase, and recent activity 79
Figure 31 The four-phase incident-response flow of the DDoS Defense mission 92
Figure 32 A roadmap organizing the future work by horizon 105

List of Tables

Table 1 Research objectives mapped to the problems they address 5
Table 2 Comparison of CyberQuest with representative categories of hands-on 12
Table 3 Systematic review of literature on gamified and hands-on cybersecurity education (2024–2025) 15
Table 4 Mapping of theoretical foundations to CyberQuest design responses 19
Table 5 Functional requirements: account and authentication subsystem 29
Table 6 Functional requirements: mission-play subsystem 29
Table 7 Functional requirements: progression and rewards subsystem 30
Table 8 Functional requirements: hints and adaptivity subsystem 30
Table 9 Functional requirements: dashboard and reporting subsystem 31
Table 10 Non-functional requirements: performance 31
Table 11 Non-functional requirements: security 32
Table 12 Non-functional requirements: usability 32
Table 13 Non-functional requirements: reliability, maintainability, portability, and scalability 32
Table 14 Use-case specification: Log In 36
Table 15 Principal data requirements of the platform 37
Table 16 Requirements traceability summary 37
Table 17 Responsibilities and boundaries of the four architectural layers 41
Table 18 Cross-model consistency: how the design views correspond 56
Table 19 Correspondence between architectural layers and implementation artifacts 57
Table 20 Rubric defining the three difficulty levels 66
Table 21 Difficulty progression and skill dependencies across the mission catalog 66
Table 22 Technology stack and the role of each component 69
Table 23 Principal failure cases and the application's handling of each 86
Table 24 Summary of the mission catalog (the platform's content dataset) 89
Table 25 Functional test results for the principal workflows 90
Table 26 Learning-outcome reachability walkthrough for each mission 91
Table 27 Test case TC-01: successful registration 93
Table 28 Research questions and the answers established by this thesis 103
Table 29 Glossary of Terms 117
Table 30 Complete requirements validation matrix 120

Acronyms and Abbreviations

The following abbreviations and acronyms are used throughout this thesis.

Abbreviation	Definition
API	Application Programming Interface
ASP.NET	Active Server Pages .NET (Microsoft web framework)
BCrypt	Blowfish-based adaptive password hashing function
CAN	Controller Area Network
CTF	Capture The Flag
CRUD	Create, Read, Update, Delete
DDoS	Distributed Denial of Service
DFD	Data Flow Diagram
EF Core	Entity Framework Core
ER	Entity Relationship
FR	Functional Requirement
HEC	Higher Education Commission (Pakistan)
HTTP(S)	Hypertext Transfer Protocol (Secure)
IoT	Internet of Things
JSON	JavaScript Object Notation
MFA	Multi-Factor Authentication
MITM	Man-in-the-Middle
MITRE ATT&CK	MITRE Adversarial Tactics, Techniques, and Common Knowledge
MVC	Model-View-Controller
NFR	Non-Functional Requirement
NICE	National Initiative for Cybersecurity Education
NIST	National Institute of Standards and Technology
ORM	Object-Relational Mapping
OT	Operational Technology
SCADA	Supervisory Control and Data Acquisition
SDLC	Software Development Life Cycle
SLA	Service-Level Agreement
SQL	Structured Query Language
SSMS	SQL Server Management Studio

Abbreviation	Definition
UI/UX	User Interface / User Experience
UML	Unified Modeling Language
VLAN	Virtual Local Area Network
WAF	Web Application Firewall
XP	Experience Points

Chapter 1

1. Introduction

The security of digital systems has become inseparable from the everyday functioning of society. Power grids, homes, vehicles, financial services, and communication networks now depend on interconnected software, and each new connection widens the surface that an attacker can target. As this dependence grows, so does the demand for people who can defend these systems competently. Yet the way cybersecurity is commonly taught has not kept pace with the way it is actually practiced. This chapter introduces the problem that motivates this thesis, states the objectives of the work, defines its scope, describes the contributions, and outlines the structure of the remainder of the document.

1.1 Background and Context

Cybersecurity is fundamentally a practical discipline. A defender must be able to read unfamiliar evidence, form a hypothesis about what is happening, and decide on an appropriate response, often under time pressure and with incomplete information. These are procedural and judgment-based skills, and they are difficult to develop through passive study alone. Traditional instruction tends to emphasize the recall of definitions, protocols, and taxonomies, which is necessary but not sufficient; a learner may be able to define a phishing attack yet still fail to spot a well-crafted one, or may know the definition of a denial-of-service attack without ever having reasoned through how to triage and mitigate one.

The consequences of this gap are not merely academic. The global shortage of qualified cybersecurity professionals is well documented, and employers repeatedly report that graduates arrive with theoretical knowledge but limited hands-on ability. Closing this gap requires learning environments in which students do security rather than only read about it, and in which the doing is structured, safe, motivating, and progressively challenging.

The stakes of this shortfall are rising rather than falling. As more of the physical world is brought under software control, from vehicles and medical devices to industrial plant and civic infrastructure, the consequences of a security failure grow more severe, and the breadth of knowledge a defender must command widens accordingly. A modern

practitioner must be conversant not only with traditional network and application security but also with the security of embedded systems, operational technology, and the human factors that social-engineering attacks exploit. An education that develops only isolated fragments of this knowledge, and only at the level of definition, leaves graduates poorly prepared for the integrated, judgment-laden work that real defense demands. This widening gap between what is taught and what is needed is the broad context within which the present work is situated.

Two broad ideas from education and human-computer interaction inform the approach taken in this thesis. The first is gamification, the use of game design elements such as points, levels, challenges, and achievements in non-game contexts to increase engagement and motivation [1]. The second is the closely related tradition of serious games and, within cybersecurity specifically, capture-the-flag (CTF) exercises, in which learners solve security challenges to recover a hidden token that proves the task was completed [2]. Both approaches share a central premise: that learning is more durable when the learner is an active participant who makes decisions and experiences their consequences, rather than a passive recipient of information.

CyberQuest, the platform presented in this thesis, applies these ideas in a structured and narrative-driven form. Rather than presenting isolated puzzles, it embeds each learning objective inside a short story in which the learner takes on the role of a security operative responding to a plausible incident. The intention is to preserve the active, evidence-driven quality of a CTF while adding the context and motivation that a coherent scenario provides.

1.2 Motivation and Problem Description

The motivation for this work arises from a specific and recurring gap: the distance between knowing about security and being able to do security. Several concrete shortcomings of common learning environments make this gap wider than it needs to be.

- Limited transfer from theory to practice. Learners who study only conceptual material often struggle to apply it when confronted with realistic, messy evidence. Knowledge that is never exercised in a decision-making context tends not to transfer to real situations.

- Weak engagement and high drop-off. Purely expository material provides little intrinsic motivation. Without a sense of progress, challenge, or reward, learners disengage before they reach the more demanding and more valuable parts of a topic.
- Fragmented and context-free exercises. Many hands-on resources present challenges as disconnected tasks with no narrative or progression. This makes it hard for a learner to see how individual skills fit together into the workflow of an actual investigation or defense.
- One-size-fits-all difficulty. A fixed level of challenge frustrates newcomers and bores more advanced learners. Environments that cannot adjust to the individual leave many learners outside the productive zone in which learning is most effective.
- Content that is hard to maintain and extend. When learning content is embedded directly in program code, adding or revising a scenario requires a developer and a redeployment, which discourages the continual expansion that a healthy learning platform needs.

These problems are not independent; they reinforce one another. A platform that addresses them together, by combining realistic and contextualized practice, motivating game mechanics, a coherent progression, adaptive support, and a maintainable content architecture, stands to be substantially more effective than one that addresses any single problem in isolation. This observation is the central motivation for CyberQuest.

It is worth noting how the problems compound. A learner who is not engaged will not persist long enough to benefit from realistic practice, so weak engagement undermines transfer. A platform that cannot adapt its difficulty will lose both the newcomer, who is overwhelmed, and the advanced learner, who is bored, so the engagement problem and the difficulty problem are intertwined. And a platform whose content is hard to extend will stagnate, so that even a well-designed system gradually ceases to hold learners' interest as its fixed catalog becomes familiar. Because the problems compound, a piecemeal solution that fixes one while leaving the others is likely to disappoint. This interdependence is precisely why the thesis pursues an integrated design in which the several remedies reinforce rather than merely coexist with one another.

1.3 Problem Statement

The problem addressed by this thesis can be stated as follows: existing cybersecurity learning environments frequently fail to combine realistic, hands-on practice with sustained engagement, coherent progression, and maintainable content, and as a result they develop conceptual familiarity more effectively than the applied, decision-making competence that practitioners require. There is a need for an integrated platform that situates security learning inside realistic narrative missions, rewards and sustains engagement through well-designed game mechanics, adapts its support to the individual learner, and stores its content and rules in a manner that allows the catalog of scenarios to grow without re-engineering the system.

1.4 Research Objectives

To address the problem described above, this thesis pursues the following objectives, which are shown in relation to the problems they address in Table 1.1.

1. To design and implement a gamified, story-driven web platform that presents cybersecurity learning as a sequence of narrative missions, each targeting a defined learning outcome.
2. To develop a mission model in which each mission is decomposed into ordered stages that a learner clears by investigating evidence through a simulated terminal and submitting a correct flag.
3. To implement a progression system based on experience points, named levels, and achievement badges that provides feedback and sustains motivation.
4. To provide a progressive hint mechanism and a lightweight adaptive-difficulty component intended to keep learners within a productive range of challenge.
5. To design a fully database-driven architecture in which all mission content, progression rules, and learner telemetry are stored in a relational database rather than hard-coded, so that the platform is maintainable and extensible.
6. To validate the platform through functional testing and a structured walkthrough confirming that each mission's intended learning outcome is reachable.

Table 1 Research objectives mapped to the problems they address

Objective	Problem Addressed
Narrative, mission-based learning	Weak engagement; fragmented, context-free exercises
Staged missions with flag capture	Limited transfer from theory to practice
XP, levels, and badges	Weak engagement and high drop-off
Progressive hints and adaptive difficulty	One-size-fits-all difficulty
Database-driven content architecture	Content that is hard to maintain and extend

1.5 Research Questions

The objectives above can be framed as a set of research questions. Following the review of 2024–2025 literature in Chapter 2, each question is stated so that it targets one facet of the research gap identified there (Section 2.11), keeping the questions, the gap, and the literature in strict alignment around this thesis's single core objective, the development of an integrated, database-driven, gamified platform for hands-on cybersecurity skill development.

- RQ1. How can realistic cybersecurity scenarios be represented as narrative, staged missions that require genuine investigative reasoning to complete? *(Addresses the gap that recent work seldom unifies narrative with genuine hands-on reasoning.)*
- RQ2. What progression and reward mechanics sustain learner motivation without undermining the learning objective? *(Addresses the gap that engagement mechanics are rarely integrated with a coherent progression.)*
- RQ3. How can support (hints and difficulty) be adapted to the individual learner in a lightweight, maintainable way? *(Addresses the gap that recent adaptivity is generic and not maintainable.)*
- RQ4. What software architecture allows all learning content and rules to reside in data, so that the platform is extensible without re-engineering? *(Addresses the gap that content is bespoke rather than extensible data, and that realism depends on heavy infrastructure.)*
- RQ5. Does the resulting platform function correctly, and are the intended learning outcomes of each mission genuinely reachable through play? (Addresses the gap that game-based designs are not always validated for outcome reachability.)

1.6 Scope of the Work

The scope of this thesis is the design, implementation, and functional validation of the CyberQuest platform as a self-contained web application. The delivered system includes a catalog of six narrative missions spanning distinct cybersecurity domains, a complete progression and achievement subsystem, user authentication with securely hashed credentials, a personal dashboard, a global leaderboard, and a relational database that holds all content and learner data. The security scenarios are simulated for a learning context: the terminal, network scans, and tool outputs are faithful in form and terminology to real practice but are reproduced within the platform rather than executed against live systems, which keeps the environment safe and self-contained.

The work does not include a large-scale empirical study of learning gains with human participants, nor a production deployment with a live user base; these are identified as future work. The adaptive-difficulty component is deliberately lightweight and rule-based rather than a fully trained machine-learning model, reflecting the priority placed on delivering a coherent, complete, and maintainable platform within the scope of an MS thesis.

1.7 Contributions

The principal contributions of this thesis are as follows:

1. A design for representing cybersecurity learning as narrative, staged, flag-capture missions, together with a concrete catalog of six such missions across distinct domains.
2. An integrated gamification model, combining experience points, named levels, achievement badges, and streaks, tied directly to meaningful learning actions rather than trivial engagement.
3. A fully database-driven content architecture in which every mission, stage, hint, item of terminal evidence, level, and badge is stored as data, enabling extension without code changes.
4. A working implementation of the complete platform on a modern web stack, validated through functional testing and outcome-reachability walkthroughs.

1.8 Significance of the Study

The significance of this work lies in three areas. Educationally, it offers a concrete, working environment in which cybersecurity learners can develop applied skills in a safe, engaging, and structured way, addressing a gap that has practical consequences for the supply of capable practitioners. Technically, it demonstrates a data-centric architecture for learning platforms in which content is treated as first-class data rather than embedded in code, a design that materially lowers the cost of extending and maintaining such a platform. Methodologically, it shows how a substantial educational software artifact can be specified, designed, implemented, and validated within the scope of an MS thesis using standard engineering practice, providing a template that other student projects might follow.

The work is also significant for the institution and region in which it is undertaken. A self-contained, low-infrastructure platform is well suited to educational settings where the resources for a full cyber range are not available, and its database-driven design means that instructors can extend the catalog with locally relevant scenarios without specialized development skills. In this sense the platform is not only a research artifact but a potentially useful teaching tool.

1.9 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 reviews related work in cybersecurity education, gamification, serious games, and capture-the-flag platforms, and positions CyberQuest with respect to that literature. Chapter 3 provides the technical background on the web-application architecture, persistence, security primitives, and cybersecurity concepts on which the work draws. Chapter 4 presents a detailed requirements analysis, including functional and non-functional requirements and the system's use cases. Chapter 5 presents the system design, including the overall architecture, the mission and progression models, the database design, and the supporting UML models. Chapter 6 details the design of each individual mission. Chapter 7 describes the implementation, covering the technology stack, the structure of the application, representative workflows, and the database schema. Chapter 8 reports the results and evaluation, including functional testing and the mission walkthroughs. Chapter 9 discusses the findings, strengths, and limitations of the work. Chapter 10 concludes the thesis and outlines directions for future work. Supporting material, including representative code and schema listings and a per-mission content summary, is provided in the appendices.

Chapter 2

2. Literature Review

This chapter surveys the research and practice that inform CyberQuest. It begins with the pedagogical foundations of active and experiential learning, proceeds to gamification and serious games in education, then focuses on cybersecurity-specific approaches, including capture-the-flag platforms and cyber ranges. It closes by identifying the gap that this thesis addresses and by positioning CyberQuest against representative existing systems in a structured comparison.

2.1 Pedagogical Foundations

The design of CyberQuest rests on a well-established view in education that people learn procedural, judgment-based skills most effectively by doing rather than by listening. Experiential learning theory frames learning as a cycle in which concrete experience leads to reflection, the formation of abstract concepts, and active experimentation that feeds back into new experience [3]. Applied to security education, this suggests that a learner who investigates a simulated incident, reflects on the evidence, forms a hypothesis, and tests it by acting will develop more durable competence than one who only reads about the same incident.

A second foundational idea is the notion of an optimal challenge level. Work on flow describes a state of focused engagement that arises when the difficulty of a task is matched to the ability of the person performing it; too little challenge produces boredom and too much produces anxiety [4]. This principle directly motivates the progressive-hint and adaptive-difficulty features of CyberQuest, whose purpose is to keep the learner near the productive boundary between the two extremes.

A third relevant idea is constructivism, the position that learners build knowledge actively by integrating new experiences with existing mental models. Constructivist design favors authentic tasks, scaffolding that is gradually withdrawn, and immediate feedback [5]. Each of these principles has a direct analogue in CyberQuest: authentic tasks in the form of realistic mission scenarios, scaffolding in the form of progressive hints, and immediate feedback in the form of flag validation and experience awards.

2.2 Gamification and Serious Games

Gamification is commonly defined as the use of game design elements in non-game contexts [1]. In education, these elements typically include points, levels, badges, leaderboards, challenges, and narrative. A substantial body of work reports that, when applied thoughtfully, such elements can increase learner motivation, participation, and persistence [6]. The evidence is not unconditional: poorly designed gamification that rewards trivial actions or emphasizes competition over mastery can undermine intrinsic motivation. The consensus that emerges is that game elements should be tied to meaningful learning actions and should support, rather than replace, sound instructional design.

Serious games extend this idea by embedding learning objectives within a complete game experience rather than layering isolated mechanics onto conventional material. A serious game situates the learner in a goal-directed world where decisions have consequences, which supports the experiential cycle described above. CyberQuest sits between pure gamification and a full serious game: it uses a coherent narrative and role for each mission,

while retaining the lightweight, browser-based delivery and the explicit points-and-badges economy characteristic of gamified systems.

The specific mechanics adopted in CyberQuest, experience points, named levels, achievement badges, streaks, and a leaderboard, are chosen because each maps to a documented motivational function. Points and levels provide a sense of steady progress; badges recognize distinct accomplishments and encourage completeness; streaks reward consistency; and a leaderboard introduces a social, comparative dimension. The design deliberately ties each mechanic to a genuine learning action so that engagement and learning are aligned rather than in tension.

2.3 Capture-the-Flag and Hands-on Security Platforms

Within cybersecurity education specifically, the capture-the-flag format is the dominant hands-on paradigm. In a CTF, a learner completes a security task, such as analyzing a file, exploiting a weakness, or inspecting network traffic, in order to recover a hidden token, or flag, that proves completion [2]. Two broad styles exist: jeopardystyle CTFs present independent challenges grouped by category, while attack-defense CTFs pit teams against one another over shared infrastructure [7]. The jeopardy style is the more

suitable basis for structured learning because each challenge can target a specific skill and can be sequenced by difficulty.

A number of widely used platforms deliver hands-on security practice. Educational CTF competitions have been shown to attract and develop talent effectively, particularly when challenges are scaffolded for newcomers [8]. Interactive learning sites provide guided exercises and browser-based labs; open CTF frameworks allow instructors to host their own challenge sets; and cyber ranges provide full virtualized environments in which learners operate real tools against real, isolated systems.

These platforms are powerful, but each embodies trade-offs. Cyber ranges offer high fidelity at the cost of significant infrastructure and maintenance. Open CTF frameworks offer flexibility but leave narrative, progression, and pedagogy to the instructor. Guided learning sites offer structure but often present challenges as isolated tasks with limited overarching story or adaptive support.

CyberQuest is positioned deliberately within these trade-offs. It adopts the flag-capture mechanic and the guided, sequenced structure of jeopardy-style learning, but it wraps each challenge in a narrative role and connects the challenges through an explicit progression economy. Because its scenarios are simulated within the application rather than executed against live virtual machines, it avoids the infrastructure burden of a cyber range while remaining safe and fully self-contained, which suits an educational setting with limited resources.

It is worth dwelling on why the jeopardy-style, single-player structure was chosen over the attack-defense alternative. Attack-defense competitions, in which teams simultaneously attack one another's systems and defend their own, are exciting and realistic but are poorly suited to structured introductory learning: they require substantial infrastructure, presuppose considerable prior skill, and provide little scaffolding for a newcomer who does not yet know where to begin. The jeopardy style, by contrast, allows each challenge to be self-contained, to target a specific skill, and to be ordered by difficulty, which makes it a natural fit for a curriculum that must take a learner from novice to competent. CyberQuest builds on this foundation by adding the narrative and progression that plain jeopardy challenges typically lack, thereby retaining the pedagogical tractability of the jeopardy style while addressing its tendency toward fragmentation.

The choice to simulate rather than virtualize the scenarios also deserves justification. A platform that spins up real, isolated virtual machines for each challenge offers the highest fidelity, but at a cost in infrastructure, maintenance, and startup latency that is difficult to bear in a resource-constrained educational setting and that raises the barrier to adding new content. By simulating the evidence and tool output within the application, CyberQuest trades some fidelity for a dramatic reduction in operational burden and for the content-authoring simplicity that its database-driven design provides. For the introductory and intermediate conceptual learning the platform targets, this is a favorable trade; the transfer to real tooling that virtualization would provide is identified as future work.

2.4 Adaptive and Personalized Learning

Adaptive learning systems adjust the content, sequence, or difficulty of instruction to the individual learner. Approaches range from simple rule-based rules that branch on performance thresholds to sophisticated models that estimate a learner's knowledge state and select the next item accordingly. In the context of games, dynamic difficulty adjustment continuously tunes challenge to keep the player engaged. CyberQuest incorporates a deliberately lightweight, rule-based form of this idea: it observes signals such as accuracy and hint usage within a mission and recommends whether to maintain, ease, or raise the challenge. This choice reflects the thesis's priority on delivering a complete, maintainable platform rather than a fully trained model, while leaving a clear path toward richer personalization in future work.

2.5 Standards and Frameworks Informing Content

The content of CyberQuest is informed by recognized cybersecurity frameworks so that the skills it develops map to those valued in practice. The NICE Cybersecurity Workforce Framework organizes the field into categories, specialty areas, and work roles, and provides a vocabulary for describing what practitioners actually do [9]. Each CyberQuest mission is aligned to a NICE-style category, such as "Protect and Defend" or "Analyze," so that its learning outcome corresponds to a recognized competency.

For the incident-response content, and in particular the DDoS Defense mission, the design draws on the NIST computer-security incident-handling guide, which structures response into preparation, detection and analysis, containment, eradication and recovery, and post-incident activity [10]. The four phases of the DDoS mission, triage,

classification, mitigation, and documentation, mirror this structure. The attack classification content draws on the MITRE ATT&CK knowledge base, which catalogs adversary tactics and techniques and assigns them identifiers [11]. Using these identifiers in the mission gives learners exposure to the same vocabulary used in professional threat reporting.

2.6 Comparison with Existing Approaches

Table 2.1 summarizes how CyberQuest compares with representative categories of existing hands-on security learning environments across the dimensions most relevant to this thesis: the presence of a guiding narrative, an explicit progression economy, adaptive support, the infrastructure burden imposed, and the maintainability of the content. The comparison is qualitative and is intended to clarify the niche that CyberQuest occupies rather than to rank the systems, each of which is well suited to its own purpose.

Table 2 Comparison of CyberQuest with representative categories of hands-on

Dimension	Open CTF frameworks	Guided learning sites	Cyber ranges	CyberQuest
Guiding narrative	Rare	Partial	Scenario-based	Mission story per topic
Progression economy	Scoreboard only	Points/streaks	Varies	XP, levels, badges
Adaptive support	None	Limited hints	Instructor-led	Progressive hints + rules
Infrastructure burden	Moderate	Low	High	Low (self-contained)
Content maintainability	Config files	Proprietary	Complex	Database-driven

2.7 Motivation and Engagement Theory

Because a central aim of CyberQuest is to sustain engagement, it is worth examining the theories of motivation that explain when game elements help and when they hinder. Self-determination theory distinguishes intrinsic motivation, which arises from the inherent satisfaction of an activity, from extrinsic motivation, which arises from external rewards. It identifies three psychological needs whose satisfaction supports intrinsic motivation: competence, the sense of being effective; autonomy, the sense of acting by one's own

volition; and relatedness, the sense of connection to others. Well-designed gamification supports these needs rather than substituting external rewards for them: points and levels can signal growing competence, freely chosen missions and optional hints support autonomy, and a leaderboard and shared achievements support relatedness.

The risk that gamification can undermine intrinsic motivation, sometimes called the over justification effect, occurs when extrinsic rewards come to feel controlling and displace the inherent satisfaction of the task. The design response adopted in CyberQuest is to attach rewards to genuine accomplishments, completing a mission, mastering a stage without hints, rather than to trivial or repetitive actions, so that the rewards signal competence rather than merely incentivize activity. This alignment of reward with mastery is the reason the level thresholds are set above the base experience available from simple completion, as described in Chapter 5.

Figure 2.1 shows how the specific game elements used in CyberQuest map onto the psychological needs identified by self-determination theory. Each element is included because it supports one or more of these needs, rather than for its own sake.

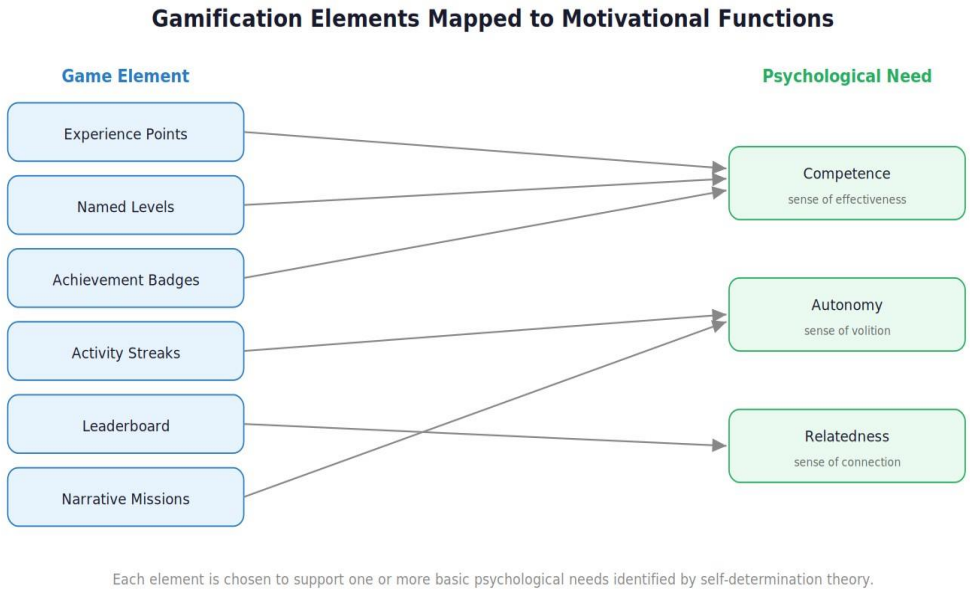


Figure 1 Gamification elements mapped to the motivational functions they serve

As Figure 2.1 indicates, experience points, levels, and badges primarily support the sense of competence; freely chosen missions and optional hints support autonomy; and the leaderboard and shared achievements support relatedness. This deliberate mapping is what distinguishes principled gamification from the superficial addition of points that the literature warns against.

2.8 Scaffolding and the Zone of Proximal Development

The progressive-hint mechanism of CyberQuest is grounded in the educational concept of scaffolding, itself derived from the zone of proximal development, the range of tasks a learner can accomplish with guidance but not yet independently. Effective scaffolding provides just enough support to keep a learner progressing and is gradually withdrawn as competence grows. In CyberQuest, the ordered hints for each stage form a scaffold: the first hint offers a gentle orientation, later hints become more specific, and the final hint effectively reveals the answer. Because each hint carries a small experience cost, learners are encouraged to attempt the task with as little support as they need, which both preserves the challenge and provides the system with a signal of where difficulty lies. This mirrors the fading of scaffolding recommended in the literature, adapted to a self-directed, gamified context.

2.9 Assessment in Skill-Based Learning

A further theme in the literature concerns how skill-based learning is assessed. Traditional assessment through multiple-choice testing measures recalls but correlates weakly with the ability to perform. Performance-based assessment, in which the learner is judged by their success on an authentic task, aligns far better with the goals of skill development. The flag-capture mechanism is, in effect, a form of performance-based assessment: a learner cannot capture a flag without performing the investigative action the stage is designed to teach, so success on the task is itself evidence of the skill. This property is what the reachability walkthroughs in Chapter 8 verify, and it is a key reason the flag mechanic was chosen over quiz-style questions.

2.10 Systematic Review of Recent Literature (2024–2025)

To ground this thesis in the current state of the art, this section reviews twenty recent primary and secondary studies published in 2024 and 2025 on gamification, serious games, and hands-on approaches to cybersecurity skill development, the domain of this thesis's single core objective. Ten studies are drawn from 2024 and ten from 2025. For each, Table 2.2 records the authors and year, the paper title, the methodology employed, the key findings, and the limitation or gap the study leaves open. The studies are cited in the numbered style used throughout this thesis, and full details appear in the References. The recurring gaps identified in the final column are synthesized in Section 2.11 into the research gap that this thesis addresses.

Table 3 Systematic review of literature on gamified and hands-on cybersecurity education (2024–2025)

Author(s) & Year	Paper Title	Methodology	Key Findings	Limitation / Gap Addressed
Namukasa et al. (2024) [12]	Evaluation of an IoT device-based educational approach to engage a more diverse cybersecurity workforce	Quasiexperimental, pre/post-test with high-school students	Hands-on IoT activities raised engagement, especially among underrepresented groups; challenge–skill balance was key	Narrow to IoT hardware and one activity; no persistent progression or software platform
Alothman (2024) [13]	Cyber gamification: implementing gamified adaptive learning environments for cybersecurity teams	System design and pilot within a training platform	An adaptive, gamified platform customized content to team performance and improved engagement	Adaptivity is team-oriented and not tied to a maintainable, data-driven content model
Karagiannis & Magkos (2024) [14]	Leveraging gamification and game-based learning in cybersecurity education	Design and deployment of two CTF variants (open-ended and story-based)	A story-based CTF engaged non-cyber students more than an open-ended one; narrative aids motivation	Content is hand-built per event; no reusable, extensible content architecture
Yasin et al. (2024) [15]	Can serious gaming tactics bolster spearphishing and phishing resilience?	Controlled serious-game study on phishing resilience	Serious-game tactics measurably improved phishing and spear-phishing resistance	Single-domain (phishing); mechanics not generalized across a multi-topic curriculum
Bellemans et al. (2024) [16]	Cybersecurity serious games development: a systematic review	PRISMA sys-tematic review of serious-game development	Catalogued design elements; found most games are bespoke and hard to extend or maintain	Confirms lack of maintainable, content-driven platforms; no unifying architecture proposed
Zhang (2024) [17]	Enhancing educational gamification through AI in higher education	Conceptual analysis of AI-enabled gamification	Identified adaptive content delivery and automated feedback as high-value directions	Remains conceptual; no concrete cybersecurity platform or evaluation

James et al. (2024) [18]	Improving retention while enhancing student engagement and learning outcomes using gamified mobile technology	Empirical study of a gamified mobile application	Gamified delivery improved engagement, retention, and outcomes over non-gamified controls	General education context; not cybersecurity, and content is fixed in the app
Ofte (2024) [19]	The awareness of operators: a goal-directed task analysis in SOCs for critical infrastructure	Goal-directed task analysis in security operations centers	Characterized the real investigative tasks defenders perform under uncertainty	Descriptive; does not propose a training platform that rehearses those tasks
Pramod (2024) [20]	Gamification in cybersecurity education: a state of-the-art review and research agenda	PRISMA systematic literature review	Mapped game-design elements to training effectiveness; set a research agenda	Review only; calls for integrated platforms but builds none
Namukasa et (2024b) [21] al.	Diversifying cybersecurity: an IoT-based training course to bridge the diversity gap	Quasiexperimental, within-subjects repeated measures	An eight-module hands-on course improved self-efficacy, interest, and knowledge	Course-based and hardware-dependent; not a self-contained, extensible software platform
Schafeitel-Tähtinen et al. (2025) [22]	Teaching and learning cybersecurity using capture the flag: effectiveness in Finland and Czechia	Pre/post-survey quasi-experiment across two universites	Gamified CTF scenarios improved knowledge and self-efficacy; effects varied by group	Requires a cyber-arena infrastructure; content is not portable or data-driven
Ali (2025) [23]	Unlocking cybersecurity: a game-changing frame-work for training and awareness—a systematic review	PRISMA systematic review with snowballing	Serious-game frameworks outperform traditional training for awareness and hands-on skill	Synthesizes frameworks but none unify narrative, progression, adaptivity, and extensibility
González-López et al. (2025) [24]	Adaptive gamification in higher education: tailoring game elements to player profiles	Cross-sectional survey by player type	One-size-fits-all gamification is sub-optimal; tailoring elements to player type helps	Adaptivity is profile-based and generic; not applied to cybersecurity skill tasks

Rahartomo et al. (2025) [25]	Phishing awareness via game-based learning	Serious-game design with pre/post evaluation	Players showed a 24% awareness gain and 30% confidence boost against phishing	Single-topic game; no progression economy or reusable content model
Yasin et al. (2025) [26]	What goes wrong during phishing education? A probe into a game-based assessment with unfavorable results	Empirical game-based assessment study	Poorly designed game assessment can fail to teach; design fidelity is decisive	Cautionary; underscores need for principled, verifiable outcome design
Lazarov et al. (2025) [27]	Lessons learned from using a cyber range to teach cybersecurity at different levels of education	Multi-cohort action-research across education levels	Cyber ranges aid hands-on learning but demand heavy setup and tailoring per level	High infrastructure burden; motivates low-infrastructure, self-contained alternatives
Kavvadias & K silieris (2025) [28] ot-	Understanding the role of demographic and psychological factors in users' susceptibility to phishing emails: a review	Structured literature review	Susceptibility is driven by measurable cognitive and behavioral factors	Identifies factors but offers no interactive platform to build resistance
Zulqarnain et (2025) [29] al.	Adaptive gamification in collaborative virtual classroom: a systematic review	PRISMA review (2019–2024 corpus)	Grounds adaptive gamification in Self-Determination Theory and constructivism	Review only; no concrete, content-extensible cybersecurity implementation
Bahrini et (2025) [30] al.	Game elements in cybersecurity education: hype or help?	Empirical evaluation of individual game elements	Some elements aid learning while others add little; effects must be measured, not assumed	Argues for evidence-based element selection; no integrated platform delivered

As Table 2.2 shows, the 2024–2025 literature strongly affirms the value of gamified, hands-on, and adaptive cybersecurity education, yet a consistent pattern emerges in the final column: individual studies advance one dimension, engagement,

adaptivity, narrative, or hands-on realism, while treating learning content as bespoke, hard to extend, or dependent on heavy infrastructure. This pattern is the empirical basis for the research gap articulated next.

2.11 Research Gap

The systematic review of 2024–2025 literature in Table 2.2, together with the theoretical foundations reviewed earlier in this chapter, makes it possible to state precisely what current work lacks. Four gaps recur across the recent studies.

First, recent work optimizes single dimensions in isolation. Studies advance engagement [18], [25], adaptivity [13], [24], [31], narrative [14], or hands-on realism [12], [22], but no single 2024–2025 study integrates narrative-driven practice, an explicit progression economy, adaptive support, and maintainable content in one platform.

Second, learning content is treated as bespoke rather than as extensible data. Systematic reviews of the period find that serious games and CTF exercises are typically hand-built per event or per course and are therefore hard to extend or maintain [20], [16], [23]. No recent study proposes a content architecture in which every mission, stage, hint, and item of evidence is stored as first-class data so that the catalog grows without re-engineering.

Third, effective hands-on training still depends on heavy infrastructure. The most realistic 2024–2025 approaches rely on cyber ranges or arenas that demand substantial setup and per-level tailoring [27], [22], which limits their reach in resource-constrained settings and motivates a low-infrastructure, self-contained alternative.

Fourth, adaptivity, where present, is generic and not maintainable. Recent adaptive gamification work tailors to player profiles or uses rule-based difficulty in general education [24], [29], [31] but is neither specialized to cybersecurity investigative tasks nor expressed as data-driven rules that a maintainer can change without touching code.

Statement of the gap. Taken together, the 2024–2025 literature lacks an *integrated, low-infrastructure, database-driven platform that unifies narrative staged missions, an explicit progression economy, and lightweight adaptive support, in which all learning content and rules reside in data so the platform is extensible without re-engineering.* This is precisely the gap that CyberQuest is designed to fill, and it directly motivates the research questions of Section 1.5, each of which targets one facet of this gap.

The theoretical foundations reviewed in this chapter, and the design responses they motivate, are summarized in Table 2.3, which makes explicit the correspondence between each principle and the CyberQuest feature that applies it.

Table 4 Mapping of theoretical foundations to CyberQuest design responses

Theoretical foundation	CyberQuest design response
Experiential learning	Staged, investigative missions in which the learner acts on evidence
Flow / optimal challenge	Progressive hints and adaptive-difficulty recommendation
Constructivism	Authentic scenarios, faded scaffolding, immediate feedback
Self-determination theory	Rewards tied to mastery; optional hints; individual progression
Scaffolding / zone of proximal development	Ordered, costed hints per stage
Performance-based assessment	Flag capture that requires the target skill to succeed
Professional frameworks (NICE, NIST, MITRE)	Mission alignment to recognized competencies and vocabulary

It also provides a clear theoretical basis for the specific design choices made in this thesis: experiential learning and performance-based assessment justify the flag-capture mission model; self-determination theory and the careful alignment of rewards with mastery justify the progression economy; and scaffolding theory justifies the progressive-hint mechanism. The gap that CyberQuest addresses is therefore not the invention of a new mechanic but the integration of narrative-driven, flag-based practice, an explicit progression economy, lightweight adaptive support, and a fully database-driven content architecture into a single coherent, low-infrastructure platform, with each element grounded in an established principle. The chapters that follow describe how this integration is specified, designed, implemented, and validated.

2.12 Chapter Summary

This chapter has surveyed the foundations on which CyberQuest is built. It reviewed the pedagogical theories, experiential learning, flow, and constructivism, that justify an active, appropriately challenging, feedback-rich learning environment. It examined gamification and serious games, drawing the lesson that game elements must be tied to meaningful learning actions, and reviewed motivation theory to explain when such

elements help. It surveyed capture-the-flag and other hands-on platforms, situating CyberQuest as a self-contained, narrative-driven, guided environment, and reviewed adaptive learning and the professional frameworks that inform the mission content. Finally, it identified the research gap, the absence of a single platform integrating narrative, progression, adaptivity, and maintainable content, that the thesis addresses, and mapped each theoretical foundation to the corresponding design response. The next chapter provides the technical background needed to understand how that design is realized.

Chapter 3

Background Technologies and Concepts

This chapter provides the technical background needed to understand the design and implementation choices made in the remainder of the thesis. It reviews the web-application architecture on which CyberQuest is built, the object-relational mapping approach used for persistence, the security primitives employed, and the cybersecurity concepts that the mission content draws upon. Readers already familiar with these topics may treat this chapter as reference material.

3.1 The Model-View-Controller Pattern

The Model-View-Controller (MVC) pattern is an architectural pattern that separates an application into three kinds of component with distinct responsibilities. The model represents the application's data and the rules that govern it. The view is responsible for presenting the model to the user. The controller mediates between the two, interpreting user actions, updating the model, and selecting the view to render. This separation of concerns is valuable because it allows each aspect of the application to change independently: the presentation can be redesigned without altering the data rules, and the data rules can evolve without rewriting the presentation.

In a web context, the MVC pattern maps naturally onto the request-response cycle. A request from the browser is routed to a controller action, which obtains or updates the relevant model data and then returns a view that the framework renders to HTML and sends back to the browser. ASP.NET Core, the framework used by CyberQuest, implements this pattern directly: controllers are classes whose public methods are actions, models are ordinary objects, and views are Razor templates that combine HTML with embedded server-side expressions. The request pipeline through which this occurs was illustrated in Chapter 5.

The benefits of MVC for a platform like CyberQuest are concrete. Because the mission-play logic lives in a controller, the badge logic in a service, and the presentation in views, a change to the appearance of the dashboard does not risk altering how experience is awarded, and a change to the progression rules does not require touching

the presentation. This separation is one of the reasons the codebase remained maintainable through the significant mid-project migration described in Chapter 9.

3.2 Object-Relational Mapping

Relational databases store data in tables of rows and columns, whereas object-oriented programs manipulate objects with fields and references. Bridging these two representations by hand is tedious and error-prone: a developer must write queries to load rows, construct objects from them, track which objects have changed, and write queries to save them back. Object-relational mapping (ORM) automates this bridge. An ORM maps classes to tables and objects to rows, generates the necessary queries, and tracks changes so that they can be persisted with a single operation.

Entity Framework Core, the ORM used by CyberQuest, allows the developer to declare entity classes and a context class that exposes collections of those entities. Queries are written in a high-level, type-safe query language that the ORM translates into parameterized SQL. This translation to parameterized queries is significant for security, because it prevents the injection vulnerabilities that arise when queries are built by string concatenation. The ORM also manages the mapping of relationships, such as the one-to-many relationship between a mission and its stages, so that navigating from an object to its related objects is natural in code.

For CyberQuest, the ORM is what makes the fully database-driven design practical. Because the ORM handles the translation between the rich object model used by the application and the normalized tables used by the database, the developer can work with missions, stages, and hints as objects while the content itself lives entirely in the database. The one place where explicit configuration was required, the non-default primary key of the level entity and the unique indexes that enforce integrity, was handled in the model-building method shown in Chapter 7.

3.3 Password Security and Adaptive Hashing

Storing user passwords securely is a foundational requirement of any application with accounts. Passwords must never be stored in plaintext, because a breach of the database would then immediately expose every user's credential. The accepted practice is to store only a one-way hash of the password, from which the original cannot feasibly be recovered, and to verify a login by hashing the submitted password and comparing it with the stored hash.

However, not all hash functions are suitable. Fast, general-purpose hash functions allow an attacker who obtains the hashes to try enormous numbers of candidate passwords per second. The remedy is an adaptive hash function such as BCrypt, which is deliberately slow and whose cost can be tuned upward as hardware improves, and which incorporates a per-password random salt so that identical passwords produce different hashes and precomputed attack tables are ineffective [32]. CyberQuest uses BCrypt for exactly these reasons: the salt defeats precomputation and the tunable work factor keep brute-force attack expensive. This choice realizes the security requirement NFR-5 and is reflected in the registration and login handlers shown in the appendix.

3.4 Sessions and Authentication State

The Hypertext Transfer Protocol is stateless: each request is independent, and the server does not inherently remember that a previous request came from the same user. To maintain the notion of a logged-in user across requests, web applications use sessions. On successful authentication, the server creates a session, stores the user's identity in server-side session state, and gives the browser a session identifier in a cookie. Subsequent requests carry the cookie, allowing the server to associate them with the session and thus

with the authenticated user.

Keeping the authoritative session state on the server, rather than trusting data held by the client, is important for security, because a client could otherwise tamper with its own identity. CyberQuest stores the learner's identity and summary state in server-side sessions and treats the session cookie as opaque, realizing the security requirement NFR-7. The session is configured with a reasonable idle timeout and marked so that it is not accessible to client scripts, reducing the risk of theft.

3.5 Cybersecurity Concepts in the Mission Content

The mission content draws on a range of cybersecurity concepts, briefly reviewed here so that the mission descriptions in Chapter 6 are self-contained.

Operational technology and SCADA. Industrial systems such as power grids are controlled by supervisory control and data acquisition (SCADA) systems, a form of operational technology distinct from ordinary information technology. Their security is critical because a compromise can have physical consequences, and their traffic and

devices differ from those of ordinary networks, which is the investigative context of the first mission.

Internet of Things. The Internet of Things comprises everyday devices, from cameras to appliances, connected to networks. Such devices are frequently shipped with weak default configurations and are a common weak link, which the second mission explores, along with the mitigating principle of network segmentation, dividing a network so that a compromise in one segment cannot easily spread to another.

Social engineering and phishing. Not all attacks are technical; many exploit human psychology. Phishing, the third mission's subject, uses deceptive messages to induce victims to reveal information or take harmful actions, combining technical cues such as spoofed senders and malicious links with psychological pressure such as false urgency.

Authentication and hashing. The fourth mission concerns authentication weakness, connecting the strength of a password to the feasibility of guessing it and demonstrating how weak passwords and weak hashing compound, before introducing the defenses of multi-factor authentication and passphrases.

Embedded and automotive security. Modern vehicles contain internal networks, notably the controller-area-network bus, over which components communicate. Because this bus historically lacked strong message authentication, forged messages can be injected, the subject of the fifth mission, which also introduces the integrity mechanisms that detect such forgery.

Denial of service and incident response. A distributed denial-of-service attack overwhelms a service with traffic from many sources. Defending against one requires a structured incident-response process, and classifying the attack requires a taxonomy such as MITRE ATT&CK [11] and an understanding of the process codified in guidance such as the NIST incident-handling guide [10]. These concepts underpin the sixth and most advanced mission.

Together, these concepts span the breadth of the CyberQuest curriculum and provide the technical grounding for the detailed mission designs that follow in Chapter 6.

3.6 Common Web Application Vulnerabilities

Because CyberQuest is itself a web application, it must guard against the vulnerabilities that commonly afflict such applications, several of which are also relevant to the security concepts it teaches. The most pertinent are briefly reviewed here.

Injection. Injection vulnerabilities arise when untrusted input is incorporated into a command or query without proper handling, allowing an attacker to alter the intended operation. The classic example is SQL injection, in which crafted input changes the meaning of a database query. The standard defense, used throughout CyberQuest, is to use parameterized queries, in which input is bound to placeholders rather than concatenated into the query text; the object-relational mapper applies this defense automatically.

Broken authentication. Weaknesses in how credentials are handled or sessions are managed can allow attackers to impersonate users. Defenses include storing only salted, adaptive hashes of passwords, maintaining session state on the server, and using secure, non-guessable session identifiers, all of which CyberQuest applies.

Sensitive data exposure. Applications can inadvertently reveal sensitive data, whether through insecure storage or through exposure in responses. CyberQuest's most relevant instance is the answer keys for missions, which are deliberately retained on the server and never sent to the client before submission.

Cross-site scripting. Cross-site scripting arises when an application includes untrusted input in a page without proper encoding, allowing an attacker's script to run in another user's browser. The framework's Razor view engine encodes output by default, which mitigates this class of vulnerability for the content CyberQuest renders.

Reviewing these vulnerabilities serves two purposes: it explains the protective measures adopted in the implementation, and it situates the platform's own security within the same body of knowledge that its missions draw upon, reinforcing the principle that a security-education platform should exemplify the practices it teaches.

3.7 Software Testing Concepts

The evaluation of CyberQuest rests on software-testing concepts that are briefly defined here. Functional testing verifies that the system behaves according to its specification, treating the system as a black box and comparing observed outputs with expected ones

for given inputs. A test case specifies a precondition, a sequence of steps, and an expected result, so that the outcome can be judged unambiguously; the detailed test cases in Chapter 8 follow this form. Test coverage refers to the extent to which the tests exercise the system's behavior; the evaluation aims to cover each principal workflow at least once, including both its normal path and its principal alternative paths, such as an incorrect flag or invalid credentials.

Beyond functional testing, the evaluation employs a form of structured walkthrough, in which an evaluator methodically works through a scenario to confirm a property, here, that each mission's learning outcome is reachable. This is complemented by inspection, in which artifacts such as stored data and delivered pages are examined directly to confirm non-functional properties such as the hashing of passwords and the absence of answer keys from client output. Together these methods provide a rounded assessment appropriate to a design-and-build thesis, as elaborated in the evaluation methodology of Chapter 8.

3.8 Chapter Summary

This chapter has provided the technical background for the thesis. It reviewed the ModelView-Controller pattern and its benefits for maintainability, object-relational mapping and its role in enabling the data-driven design, and the security primitives, adaptive password hashing and server-side sessions, on which the platform's own protection rests. It surveyed the common web-application vulnerabilities the platform must guard against, several of which also feature in its curriculum, and reviewed the software-testing concepts that underpin the evaluation. Finally, it introduced the cybersecurity concepts, from SCADA and IoT to social engineering, authentication, automotive security, and denial-of-service defense, that the missions draw upon. With this grounding in place, the following chapters specify, design, and implement the platform in detail.

Chapter 4

Requirements Analysis

This chapter specifies what the CyberQuest platform must do and the qualities it must exhibit. It first describes the stakeholders and the overall approach to requirements elicitation, then presents the functional requirements grouped by subsystem, followed by the non-functional requirements grouped by quality attribute. It then presents the use-case model that operationalizes these requirements, and closes with a requirements traceability summary. The requirements decomposition is shown in Figure 4.1.

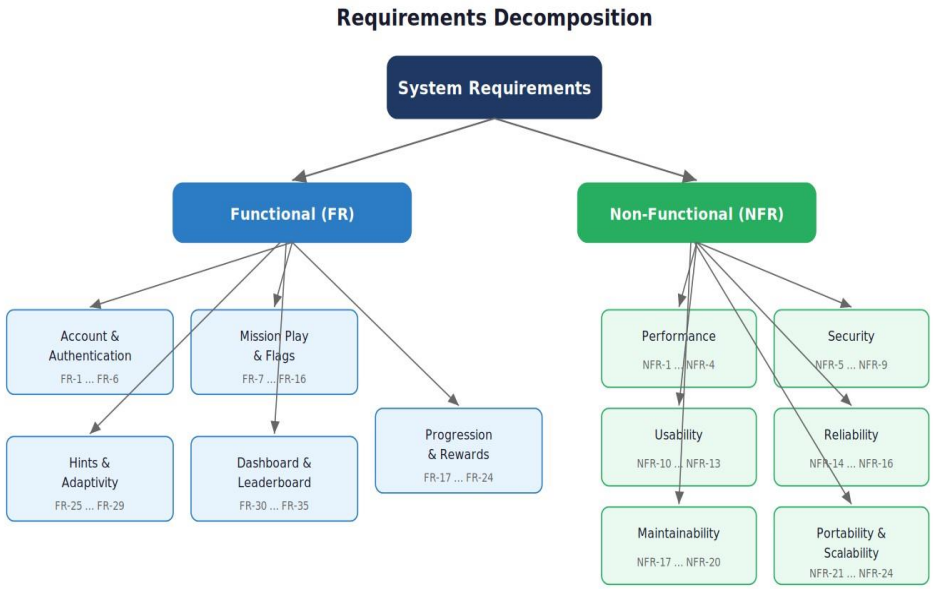


Figure 2 Decomposition of system requirements into functional and non-functional categories

4.1 Stakeholders and Elicitation

Two primary stakeholders shape the requirements of the platform. The learner is a student who uses the platform to develop cybersecurity skills; the learner's needs drive the majority of the functional requirements, particularly those concerned with mission play, feedback, and motivation. The administrator is responsible for curating content and monitoring activity; because the platform is fully database-driven, the administrator interacts with the system through database management tooling rather than a dedicated interface, and their needs drive requirements around content maintainability and the capture of telemetry.

As Figure 4.1 shows, the requirements were derived from the objectives stated in Chapter 1, from the gaps identified in the literature in Chapter 2, and from an analysis of comparable platforms. They were refined iteratively as the system was built, in keeping with the incremental development approach described in Chapter 5.

The elicitation process combined several sources. The primary source was the set of research objectives, which fix the high-level capabilities the platform must provide: narrative missions, staged flag capture, a progression economy, adaptive support, and a database-driven architecture. Each objective was decomposed into the concrete behaviors needed to realize it, yielding the functional requirements. A second source was the analysis of comparable platforms in Chapter 2, which surfaced both features to emulate, such as scaffolded challenges and clear progression, and pitfalls to avoid, such as fragmentation and unmaintainable content. A third source was the set of quality attributes expected of any credible web application, and of a security-education platform in particular, which yielded the non-functional requirements around security, performance, usability, and maintainability. Because the platform was built iteratively, the requirements were not fixed in advance but were refined as each increment revealed practical constraints and opportunities, a process that is natural in a design-and-build project and that kept the requirements grounded in what the running system could actually support.

4.2 Functional Requirements

Functional requirements specify the behavior the system must provide. They are grouped into five subsystems: account and authentication, mission play, progression and rewards, hints and adaptivity, and dashboard and reporting. Each requirement is given a unique identifier, a priority (High, Medium, or Low), and a concise description.

4.2.1 Account and Authentication

Table 4.1 lists the requirements of the account and authentication subsystem. These requirements ensure that learners can create and access accounts securely and manage their own profile.

Table 5 Functional requirements: account and authentication subsystem

ID	Priority	Description
FR-1	High	The system shall allow a new user to register an account with a full name, email address, and password.
FR-2	High	The system shall validate that the email address is well-formed and not already registered.
FR-3	High	The system shall store passwords only as salted one-way hashes, never in plaintext.
FR-4	High	The system shall authenticate a returning user by verifying the submitted password against the stored hash.
FR-5	Medium	The system shall allow an authenticated user to log out, terminating the session.
FR-6	Medium	The system shall allow an authenticated user to update their display name, university, and password.

4.2.2 Mission Play

Table 4.2 lists the requirements of the mission-play subsystem, which constitutes the core learning loop of the platform.

Table 6 Functional requirements: mission-play subsystem

ID	Priority	Description
FR-7	High	The system shall present a catalog of available missions, each showing its title, domain, difficulty, and reward.
FR-8	High	The system shall display a detail view for each mission describing its scenario, objectives, and stages.
FR-9	High	The system shall enforce mission unlocking so that a mission becomes available only when its prerequisite is complete.
FR-10	High	The system shall allow a learner to start or resume an available mission.
FR-11	High	The system shall present each mission as an ordered sequence of stages, revealing one stage at a time.
FR-12	High	The system shall provide a simulated command-line terminal supporting commands such as listing, reading files, and scanning.
FR-13	High	The system shall expose stage-specific evidence (files, scan output) through the terminal.
FR-14	High	The system shall accept a flag submission for the current stage and validate it against the expected answer.
FR-15	High	The system shall advance the learner to the next stage upon a correct submission and reject an incorrect one with feedback.
FR-16	High	The system shall mark a mission complete when its final stage is cleared.

4.2.3 Progression and Rewards

Table 4.3 lists the requirements governing the experience economy, levels, and achievements.

Table 7 Functional requirements: progression and rewards subsystem

ID	Priority	Description
FR-17	High	The system shall award a defined quantity of experience points when a mission is completed.
FR-18	High	The system shall maintain a cumulative experience total for each learner.
FR-19	High	The system shall map a learner's experience total to a named level using defined thresholds.
FR-20	High	The system shall award achievement badges when their unlock conditions are met.
FR-21	Medium	The system shall award mission-specific badges automatically on completion of the corresponding mission.
FR-22	Medium	The system shall unlock cosmetic rewards as the learner reaches higher levels.
FR-23	Medium	The system shall record an activity feed of significant events such as completions and badge awards.
FR-24	Low	The system shall track a daily activity streak for each learner.

4.2.4 Hints and Adaptivity

Table 4.4 lists the requirements of the support subsystem that helps learners who become stuck while keeping the challenge meaningful.

Table 8 Functional requirements: hints and adaptivity subsystem

ID	Priority	Description
FR-25	High	The system shall associate each stage with an ordered set of progressively more explicit hints.
FR-26	High	The system shall reveal the next hint on demand when the learner requests it.
FR-27	Medium	The system shall deduct a small amount of experience for each hint revealed.
FR-28	Medium	The system shall record each hint request as telemetry.
FR-29	Low	The system shall compute an adaptive-difficulty recommendation from the learner's accuracy and hint usage.

4.2.5 Dashboard and Reporting

Table 4.5 lists the requirements for the views that summarize a learner's state and compare learners.

Table 9 Functional requirements: dashboard and reporting subsystem

ID	Priority	Description
FR-30	High	The system shall present a personal dashboard summarizing experience, level, rank, missions completed, and badges.
FR-31	High	The system shall show per-mission progress on the dashboard.
FR-32	High	The system shall present a global leaderboard ranking learners by experience.
FR-33	Medium	The system shall highlight the current learner's own row on the leaderboard.
FR-34	Medium	The system shall present a profile page showing the learner's level, badges, and level progression.
FR-35	Low	The system shall allow a visitor to submit a contact message that is persisted for the administrator.

4.3 Non-Functional Requirements

Non-functional requirements specify the quality attributes the system must exhibit. They are grouped into six categories: performance, security, usability, reliability, maintainability, and portability and scalability.

4.3.1 Performance

Table 4.6 lists the performance requirements.

Table 10 Non-functional requirements: performance

ID	Description
NFR-1	Interactive page requests shall complete within approximately two seconds under normal single-user conditions on typical hardware.
NFR-2	Flag submission and hint requests shall return a response promptly so that the learner perceives the interaction as immediate.
NFR-3	Database queries shall be supported by appropriate indexes so that per-user and per-mission lookups remain efficient as data grows.
NFR-4	The simulated terminal shall respond to commands without perceptible delay, since it executes on the client.

4.3.2 Security

Table 4.7 lists the security requirements, which are especially important for a platform in the security domain.

Table 11 Non-functional requirements: security

ID	Description
NFR-5	Passwords shall be hashed with a salted, adaptive one-way function and never stored or logged in plaintext.
NFR-6	All database access shall use parameterized queries via the object-relational mapper to prevent injection.
NFR-7	Authentication state shall be maintained in server-side sessions rather than in client-modifiable storage.
NFR-8	Input from authentication and contact forms shall be validated before use.
NFR-9	The platform shall not expose the expected flag values or other answer keys to the client before submission.

4.3.3 Usability

Table 4.8 lists the usability requirements.

Table 12 Non-functional requirements: usability

ID	Description
NFR-10	The interface shall present all elements needed to attempt a stage, brief, terminal, and submission, on a single screen.
NFR-11	Feedback on submissions and progression shall be clear and immediate.
NFR-12	The visual theme shall be consistent across pages and reinforce the security-operative narrative.
NFR-13	The layout shall adapt to common screen sizes so the platform is usable on laptops and desktops.

4.3.4 Reliability, Maintainability, Portability

Table 4.9 lists the remaining quality requirements covering reliability, maintainability, portability, and scalability.

Table 13 Non-functional requirements: reliability, maintainability, portability, and scalability

ID	Description
NFR-14	Learner progress and experience shall persist reliably across sessions.
NFR-15	Completing a mission shall apply its side effects (experience, badges, unlocks) atomically so state remains consistent.
NFR-16	The system shall degrade gracefully, presenting a clear message rather than failing silently on error.
NFR-17	All learning content shall reside in the database so that new missions can be added without modifying application code.
NFR-18	Data access shall be confined to a single context and mapping layer to localize change.

NFR-19	The codebase shall follow the conventional Model-View-Controller structure to aid comprehension.
NFR-20	The database schema shall be reproducible from a single setup script.
NFR-21	The platform shall run on any host supporting the .NET runtime and SQL Server, independent of operating system specifics.
NFR-22	Although the present evaluation is limited to single-user validation, the architecture will support a growing catalogue of missions and a modular user base in future additions.
NFR-23	The client shall require only a modern web browser, with no plug-ins or local installation.
NFR-24	The schema shall separate content, user, and telemetry concerns to support future analytics without disruption.

4.4 System Constraints and Assumptions

The requirements are shaped by a number of constraints and assumptions that bound the design space. These are stated explicitly so that the design decisions in later chapters can be understood in context.

4.4.1 Design Constraints

- Self-contained operation. The platform must run without external services or live target systems, so that it is safe and deployable in a resource-constrained educational setting. This constraint is why the security scenarios are simulated within the application.
- Single-machine deployability. The platform must be deployable on a single development machine, with the application and database co-located, while retaining the ability to distribute the tiers if desired. This constraint shapes the three-tier but loosely coupled deployment model.
- Browser-only client. The client must require nothing beyond a modern web browser, with no plug-ins or installation, which constrains the simulated terminal to a client-side JavaScript implementation.
- Standard technology stack. The platform must be built on a mainstream, well-supported technology stack to ensure maintainability and to make the work reproducible, which motivates the choice of ASP.NET Core and SQL Server.

4.4.2 Assumptions

- It is assumed that learners access the platform through a modern browser with JavaScript enabled, as required for the simulated terminal.

- It is assumed that an administrator with database access is available to run the setup script and to curate content, since there is no in-application administrative interface.
- It is assumed that the platform operates in a trusted educational context; while standard protections are applied, the threat model does not include a determined adversary attacking the platform itself, which would be a concern for a public production deployment.
- It is assumed that the learning content, being simulated, is sufficient to develop the targeted conceptual and investigative skills, while acknowledging that transfer to live tooling is a separate concern identified as future work.

These constraints and assumptions are revisited where relevant in the design and are among the limitations discussed in Chapter 9.

4.5 Use-Case Model

The functional requirements are operationalized through a use-case model. The overall model, with both actors and the principal use cases, was introduced conceptually in the design chapter; here the two most important subsystems are detailed. Figure 4.2 shows the use cases of the account and authentication subsystem, and Figure 4.3 shows the use cases of the mission-play subsystem.

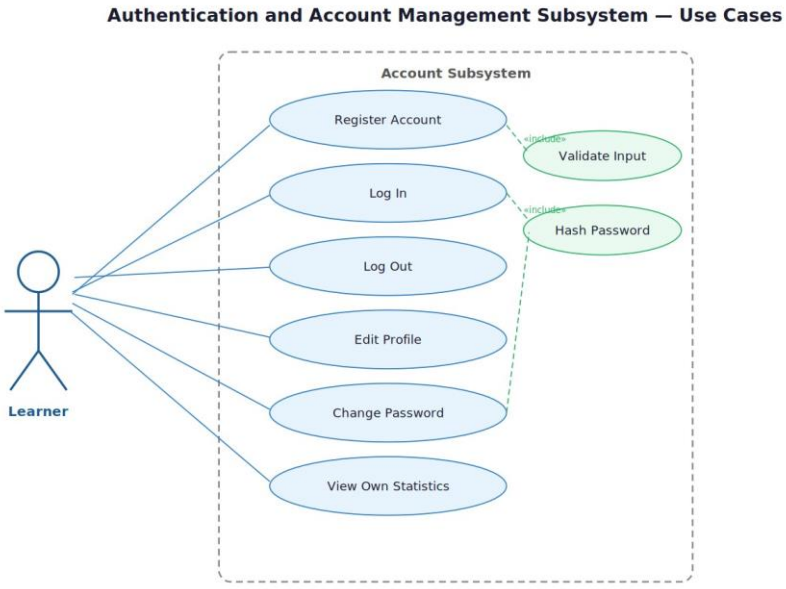


Figure 3 Use cases of the account and authentication subsystem

As Figure 4.2 shows, the learner's account interactions, registering, logging in and out, editing the profile, and changing the password, include supporting behaviors of input

validation and password hashing, which realize requirements FR-1 through FR-6 and the security requirements NFR-5 and NFR-8.

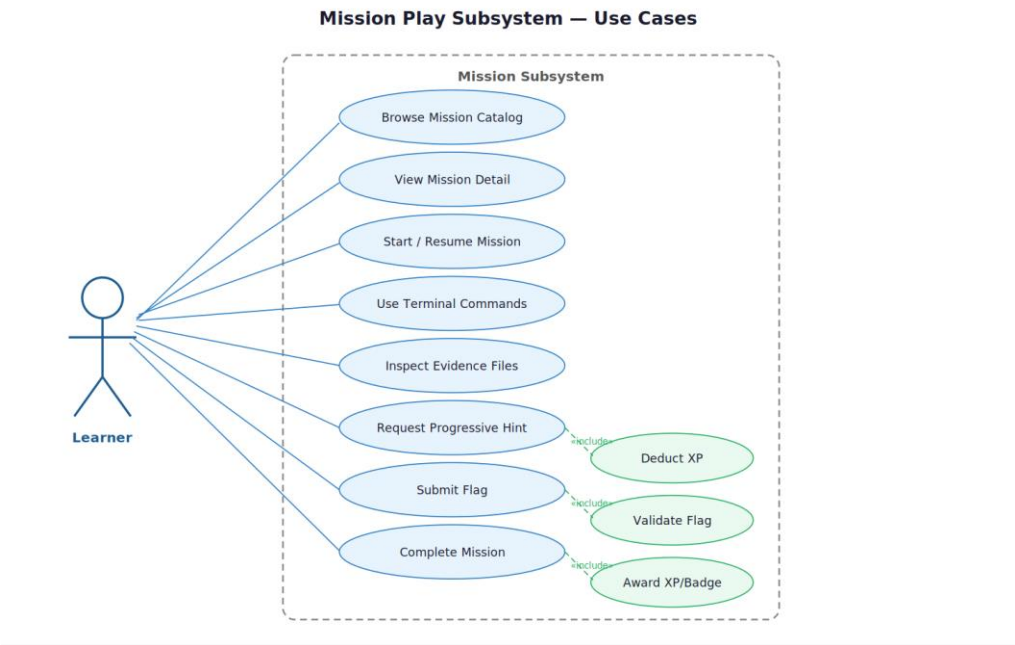


Figure 4 Use cases of the mission-play subsystem

As Figure 4.3 shows, the mission-play interactions form a chain from browsing the catalog to completing a mission, with the flag-submission and hint-request use cases including the supporting behaviors of validation, experience deduction, and reward awarding.

These use cases realize requirements FR-7 through FR-16 and FR-25 through FR-27.

4.6 Detailed Use-Case Specifications

Beyond the diagrams, the two most important use cases are specified textually so that their normal and alternative flows are explicit. Table 4.10 specifies the login use case and Table 4.11 specifies the flag-submission use case. Each specification names the actor, the preconditions, the main success scenario, and the principal alternative flow.

Table 14 Use-case specification: Log In

Element	Description
Use case	Log In
Actor	Learner
Precondition	The learner has a registered account and is not currently authenticated.
Main success scenario	(1) The learner opens the login panel. (2) The learner enters an email and password. (3) The system locates the account and verifies the password against the stored hash. (4) The system establishes a session and redirects the learner to the dashboard.
Alternative flow A	At step 3, if no account matches or the password does not verify, the system rejects the attempt with a generic message and no session is created.
Alternative flow B	At step 2, if a field is empty, the system prompts the learner to complete both fields before submission.
Postcondition	On success, the learner is authenticated and a session is active.
Use case	Submit Flag
Actor	Learner
Precondition	The learner is authenticated and is playing an available stage of a mission.
Main success scenario	(1) The learner investigates the stage evidence. (2) The learner enters a flag and submits it. (3) The system records the attempt and compares the flag with the expected value. (4) On a match, the system advances the learner to the next stage. (5) If the cleared stage was the last, the system marks the mission complete, awards experience, grants the badge, and unlocks the next mission.
Alternative flow A	At step 3, if the flag does not match, the system records the attempt, rejects the submission with feedback, and returns the learner to investigation.
Alternative flow B	At step 2, if the learner requests a hint instead, the Request Hint use case is invoked and experience is deducted.
Postcondition	On success, the learner's progress and, for a final stage, experience and rewards are updated and persisted.

4.7 Data Requirements

Beyond behavior and quality, the platform has data requirements that specify what information must be stored and the constraints upon it. These requirements drive the database design of Chapter 5 and are summarized in Table 4.12.

Table 15 Principal data requirements of the platform

Data category	Requirement
Mission content	Each mission shall store its title, domain, difficulty, reward, ordering, and unlock prerequisite.
Stage content	Each stage shall store its ordering within its mission, its brief, and its expected flag, with the flag never exposed to the client.
Hint content	Each hint shall store its ordering within its stage and its text.
Evidence content	Each item of terminal evidence shall store the stage it belongs to, its name, and its content.
Progression content	The levels and badges and their unlock conditions shall be stored as data.
User account	Each user shall store identity, a hashed credential, and cumulative statistics.
User progress	The system shall store one progress record per user per mission, capturing status and the highest stage reached.
Telemetry	The system shall store an append-only record of flag submissions, hint requests, and play sessions.

As Table 4.12 makes explicit, every category of information the platform depends upon is required to be stored as data, which is the concrete expression of the database-driven principle at the level of requirements. The integrity constraints implied here, such as the uniqueness of a progress record per user and mission, are realized by the schema constraints described in Chapter 5.

4.8 Requirements Traceability

To ensure that every requirement is addressed by the design and validated by testing, a traceability summary is maintained. Table 4.13 maps each subsystem's functional requirements to the design element that realizes them and the evaluation activity that validates them; the detailed test results appear in Chapter 8.

Table 16 Requirements traceability summary

Requirements	Realizing component	Validation
FR-1 – FR-6	AUTH Controller, Users table	Authentication functional tests
FR-7 – FR-16	Mission Controller, Missions/Sub Challenges	Mission-play tests and walkthroughs
FR-17 – FR-24	Dashboard Controller, progression tables	Progression functional tests
FR-25 – FR-29	Hint engine, Hints table	Hint functional tests

FR-30 – FR-35	Dashboard/Leaderboard/Profile Reporting functional tests controllers
NFR-5 – NFR-9	BCrypt, EF Core, Security review sessions
NFR-17 – NFR-20	Database-driven Design inspection architecture

The complete set of requirements defined in this chapter drives the design presented in Chapter 5 and the implementation described in Chapter 7, and is validated by the evaluation reported in Chapter 8.

4.9 Chapter Summary

This chapter has specified what the CyberQuest platform must do and the qualities it must exhibit. Thirty-five functional requirements were defined across five subsystems, account and authentication, mission play, progression and rewards, hints and adaptivity, and dashboard and reporting, each with an identifier and a priority. Twenty-four nonfunctional requirements were defined across six quality attributes, performance, security, usability, reliability, maintainability, and portability and scalability. The requirements were complemented by explicit constraints and assumptions, by data requirements that make the database-driven principal concrete, and by detailed use-case specifications for the two most important interactions. A traceability summary linked each group of requirements to the component that realizes it and the activity that validates it. These requirements form the specification against which the design, implementation, and evaluation of the following chapters are measured.

Chapter 5

System Design

This chapter presents the design of CyberQuest. It describes the development approach, the overall system architecture, the mission and progression models, the flow of data through the system, the database design, and the deployment model. The design is expressed through standard software-engineering models, including an architecture diagram, use case, data-flow, entity-relationship, class, component, state, and deployment diagrams, each of which is referenced from the text.

5.1 Development Approach

The platform was developed using an iterative and incremental approach. Rather than specifying the entire system in advance and building it in a single pass, the work proceeded through repeated cycles in which a slice of functionality was designed, implemented, tested, and refined before the next slice was added. This cycle is illustrated in Figure 5.1.

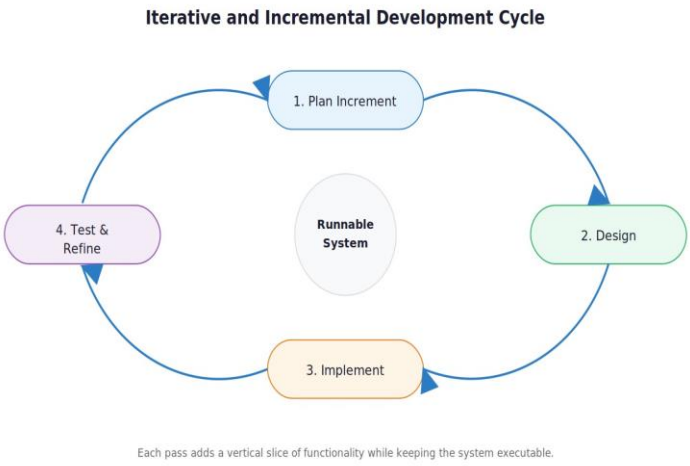


Figure 5 The iterative and incremental development cycle followed in the project

As Figure 5.1 shows, each pass through the cycle plans an increment, designs and implements it, and then tests and refines it against the running system. Early iterations established the core mission-play loop and the Data-Access and Data Layers; later iterations added the progression economy, the achievement and leaderboard subsystems, and the adaptive-support features. This approach suited a single-developer research project because it kept the system runnable at every stage and allowed design decisions

to be validated continuously against a working artifact. The overall schedule of these iterations is shown in the project timeline in Figure 5.2.

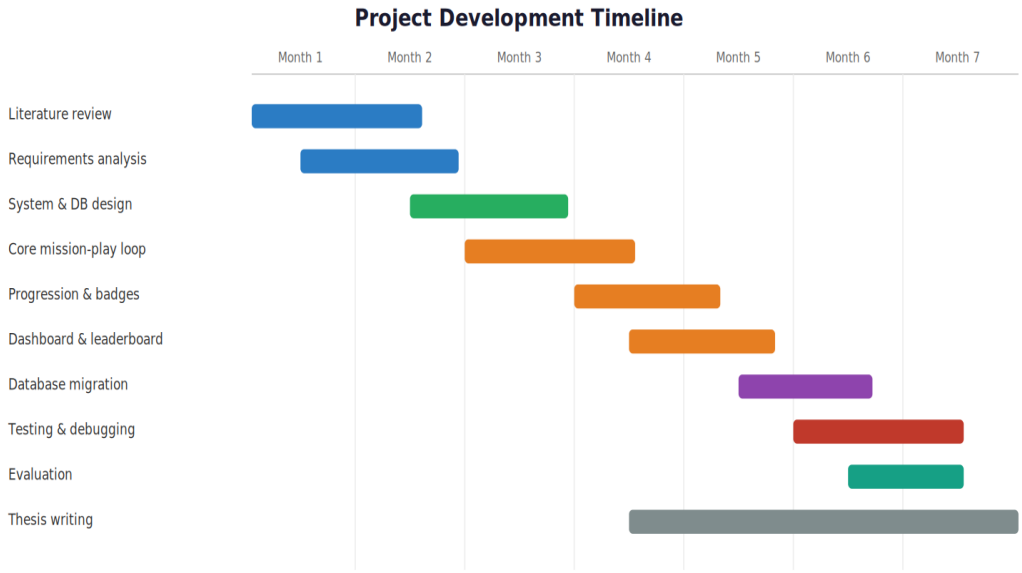


Figure 6 Project development timeline across the major work packages

5.2 System Architecture

CyberQuest follows a classic four-layer web architecture. To avoid the ambiguity that can arise when layers are referred to loosely, this thesis fixes four canonical layer names and uses them consistently throughout: the Presentation Layer, the Application Layer, the Data-Access Layer, and the Data Layer. The word “persistence” is used in this thesis only as a general adjective for the storing of state, never as the name of a layer; the layer that translates between objects and the database is always called the Data-Access Layer, and the database itself is always called the Data Layer. The four layers and their principal components are shown in Figure 5.3, and their exact responsibilities and boundaries are stated in Table 5.1.

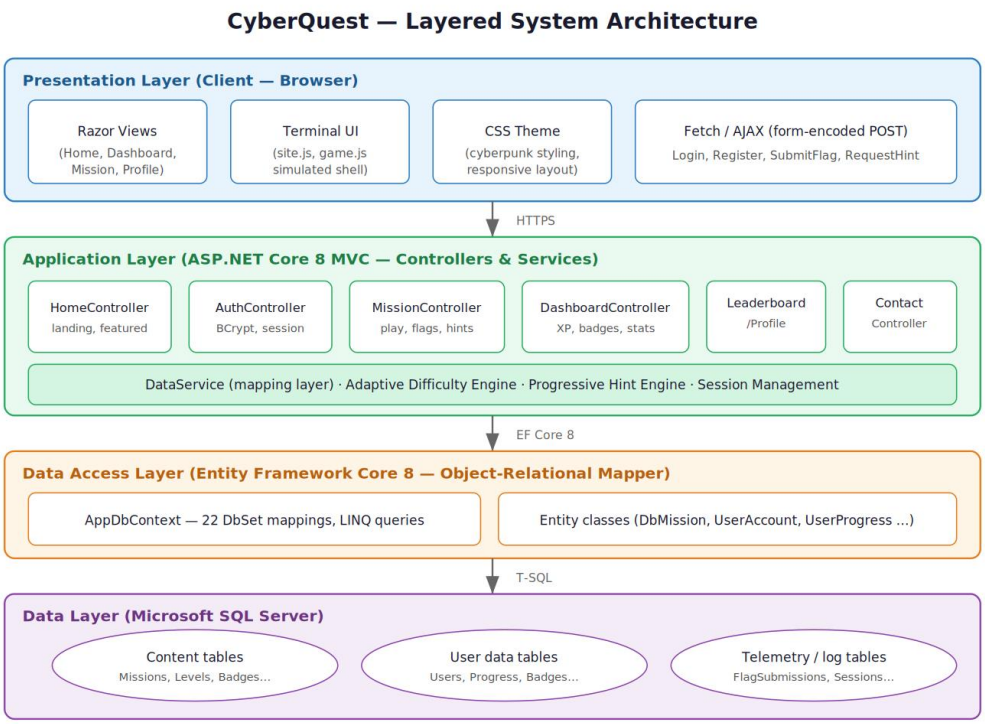


Figure 7 Layered system architecture of the CyberQuest platform

Each layer has a single, well-defined responsibility and depends only on the layer immediately below it; no layer reaches past its neighbor, and no lower layer depends on a higher one. Table 5.1 states, for each layer, what it is responsible for, what it explicitly does not do, and the concrete artifacts that realize it in the implementation described in Chapter 7.

Table 17 Responsibilities and boundaries of the four architectural layers

Layer	Responsible for	Explicitly not responsible for
Presentation	Rendering HTML from view models; running the client-side scripts that drive the simulated terminal and asynchronous requests; applying the visual theme. Realized by the Razor views and the two client scripts.	Any business rule, scoring, unlocking, or data access; it holds no authoritative state and never queries the database.
Application	Interpreting requests and enforcing rules (authentication, mission unlocking, flag validation, scoring, hint costing), then coordinating the response. Realized by the six controllers with the mapping service and adaptive-support helpers.	Rendering markup or issuing SQL directly; it obtains data only through the Data-Access Layer and returns view models to the Presentation Layer.

Data-Access	Translating between the application's objects and relational rows, generating parameterized queries, and tracking changes. Realized by the Entity Framework Core database context and the entity classes.	Deciding *what* data to fetch or *why*; it exposes typed collections but contains no game logic.
Data	Durable storage of all content, user state, and telemetry, and enforcement of integrity constraints. Realized by the Microsoft SQL Server database of 22 tables.	Application behavior; it stores and constrains data but computes no scores, levels, or unlocks.

As Table 5.1 makes explicit, the Presentation Layer runs in the learner's browser and is purely presentational; the Application Layer is an ASP.NET Core Model-View-Controller application that holds all business rules; the Data-Access Layer uses an object-relational mapper to bridge objects and the relational database; and the Data Layer is the Microsoft SQL Server database that stores all content, user data, and telemetry. As Figure 5.3 indicates, requests flow downward through the layers and results flow back upward, with each layer depending only on the one immediately below it. This assignment of responsibilities is not merely diagrammatic: it corresponds directly to the folders and classes of the delivered codebase, as confirmed in the diagram-to-implementation crosscheck of Section 5.12.

5.2.1 Rationale for the Layered Architecture

The choice of a layered architecture over alternatives deserves explicit justification. A monolithic design, in which presentation, logic, and data access are intermingled, would have been quicker to write initially but would have become progressively harder to change, precisely the maintainability problem the thesis sets out to avoid. A more elaborate architecture, such as a microservice decomposition, would have introduced operational complexity, multiple deployable units, inter-service communication, and distributed data, that is unwarranted for a single-developer platform of this scope and would have conflicted with the constraint of single-machine deployability. The layered architecture occupies a sensible middle ground: it provides a clean separation of concerns that keeps the system understandable and changeable, without incurring the overhead of distribution.

Within this layered structure, a further design decision is the confinement of all data access to a single context and of all entity-to-view-model translation to a single mapping service. This decision localizes the two kinds of change that occur most frequently in a

data-driven application, changes to the database structure and changes to the shape of the presented data, so that each can be made in one place. The benefit of this localization was borne out during the database migration described in Chapter 9, when the move to a fully data-driven design touched the persistence and mapping layers extensively but left the controllers and views largely intact.

The way an individual request travels through the application layer is shown in more detail in the pipeline of Figure 5.4. A request from the browser passes through a series of middleware components, is routed to the appropriate controller action, which invokes the mapping service and the database context to obtain data, and finally renders a Razor view that is returned to the browser.

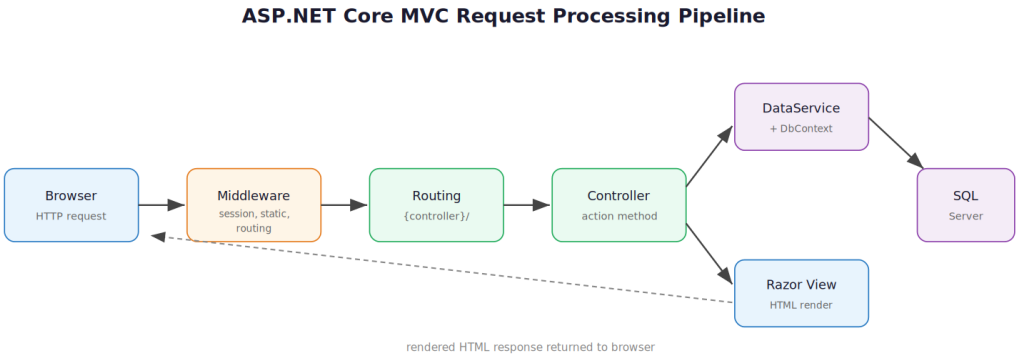


Figure 8 ASP.NET Core MVC request-processing pipeline

5.3 Component Design

At a finer grain than the four layers, the system is organized into cooperating components, shown in the component diagram of Figure 5.5. These components map onto the layers of Table 5.1: the Views and Client-Scripts components realize the Presentation Layer; the Controllers and Services components realize the Application Layer; and the Data Access component realizes the Data-Access Layer. The Controllers component depends on the Services component, which in turn depends on the Data-Access component; the Data-Access component communicates with the Data Layer (the SQL Server database). The dependency direction in the component diagram is therefore identical to the layer dependencies of Figure 5.3, and the two diagrams are consistent by construction.

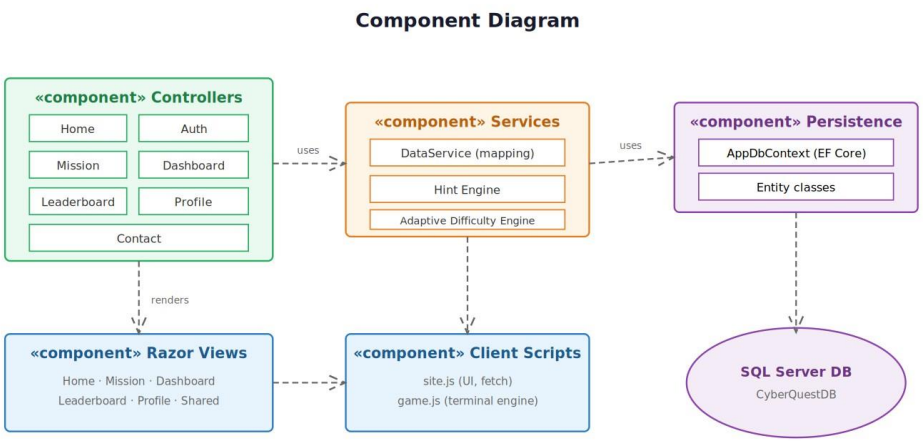


Figure 9 Component diagram showing the principal software components and their dependencies

As Figure 5.5 shows, confining all data access to the persistence component and all entity-to-view-model translation to the services component localizes the two kinds of change that occur most often, database structure and presentation shape, and keeps the controllers focused on coordinating requests. This directly serves the maintainability requirements NFR-17 through NFR-20.

5.4 Actors and Use Cases

The system has two actors, as established in Chapter 4. The overall use-case model, combining both actors and all principal use cases, is shown in Figure 5.6. The primary actor is the learner, whose interactions center on the mission-play loop and the surrounding progression and feedback features; the secondary actor is the administrator, who curates content and monitors analytics through the database.

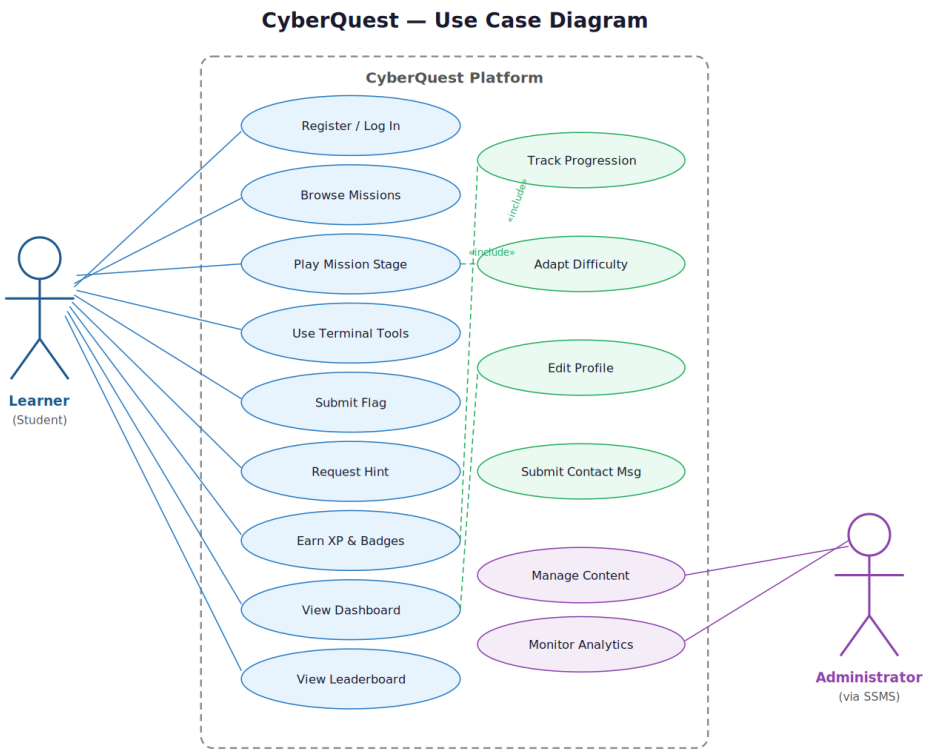


Figure 10 Overall use-case diagram showing both actors and the main system interactions

As Figure 5.6 shows, several learners use cases include supporting system behaviors such as tracking progression and adapting difficulty, and the administrator's use cases act on the same underlying data through the database. The subsystem-level use cases were detailed in Chapter 4.

5.5 Mission and Progression Model

The central abstraction in CyberQuest is the mission. Each mission targets a defined learning outcome within a cybersecurity domain and is decomposed into an ordered sequence of stages, referred to internally as sub-challenges. A stage presents a short brief that frames an investigative task, exposes evidence through the simulated terminal, and is cleared when the learner submits the correct flag. Progressing through all stages completes the mission.

5.5.1 Progression Economy

Completing a mission awards a fixed quantity of experience points. Accumulated experience advances the learner through a series of named levels, each with an associated threshold, and unlocks cosmetic rewards and achievement badges. Requesting a hint deducts a small amount of experience, which creates a meaningful trade-off between

independent effort and assisted progress. Missions unlock in a defined order, so that later, more demanding missions become available only after their prerequisites are complete.

The unlock structure and the experience economy are illustrated in Figure 5.7.

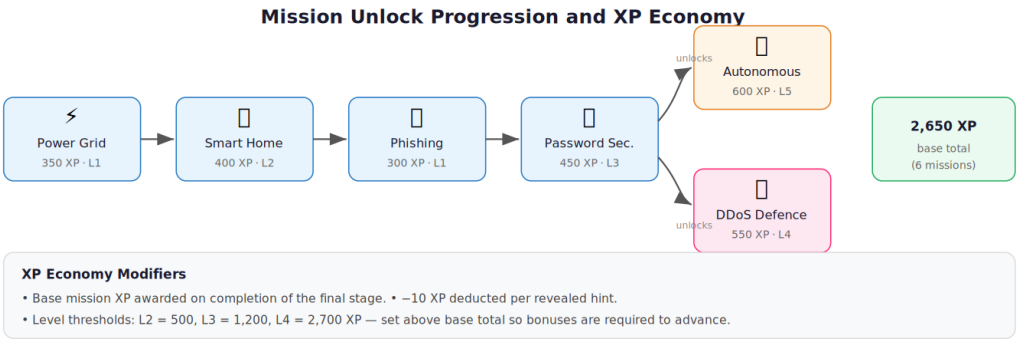


Figure 11 Mission unlock progression and the experience-point economy

The relationship between cumulative experience and the level thresholds is shown in Figure 5.8. The thresholds are deliberately set so that the base experience available from completing all six missions does not by itself carry the learner to the highest early levels; bonus experience, earned by avoiding hints and by maintaining activity streaks, is required.

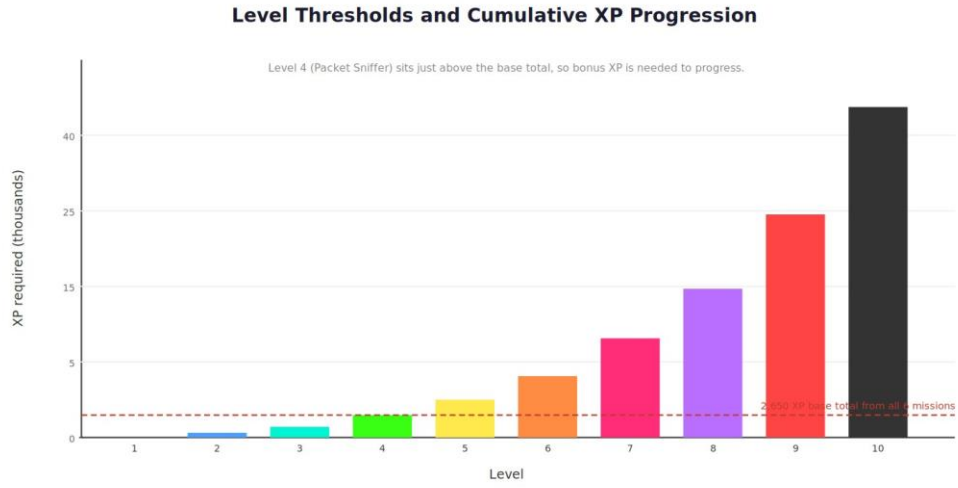


Figure 12 Level thresholds relative to the cumulative experience available from the mission catalog

As Figure 5.8 illustrates, the Level 4 threshold sits just above the base total of experience from all missions, which keeps the progression meaningful and rewards mastery rather than mere completion.

5.5.2 Mission Lifecycle

From the learner's perspective, each mission moves through a well-defined lifecycle of states, shown in the state diagram of Figure 5.9. A mission begins locked, becomes available when its prerequisite is met, enters an in-progress state as stages are cleared, and finally reaches a completed state; a completed mission may be replayed to improve a score.

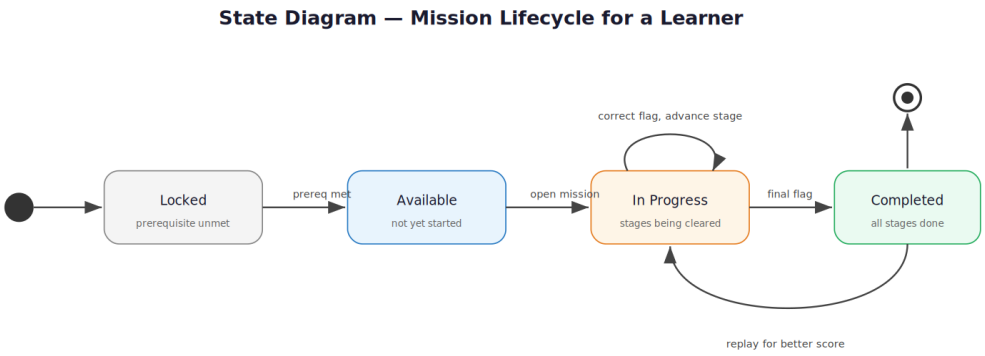


Figure 13 State diagram of the mission lifecycle for a learner

As Figure 5.9 shows, the self-transition on the in-progress state represents the clearing of successive stages, while the transition to completed triggers the progression side effects of awarding experience, granting badges, and unlocking the next mission.

5.5.3 Progressive Hints and Adaptive Difficulty

Each stage is associated with an ordered list of hints that move from a gentle nudge toward progressively more explicit guidance, with the final hint effectively revealing the answer. Hints are revealed on demand and are individually costed in experience, so the learner controls the balance between challenge and support. This scaffolding structure and its experience trade-off are illustrated in Figure 5.10.

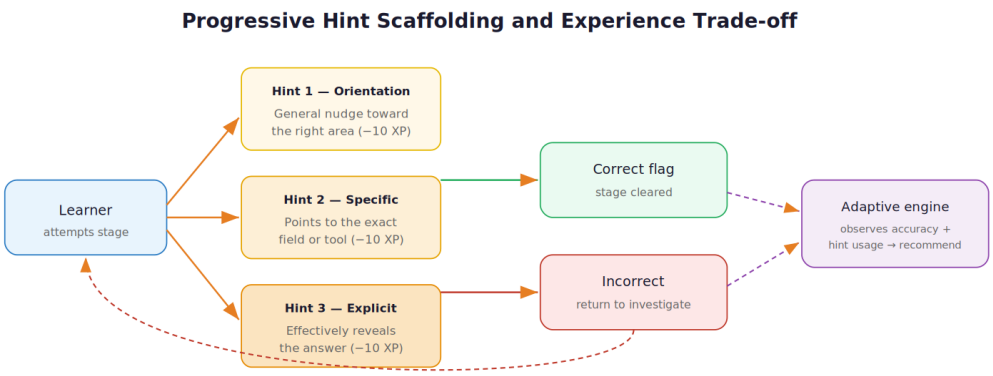


Figure 14 Progressive hint scaffolding and the experience trade-off within a stage

As Figure 5.10 shows, a learner who becomes stuck can reveal successively more explicit hints, each at a small experience cost, until able to submit the correct flag; an incorrect submission returns the learner to investigation. In addition, a lightweight adaptive component observes performance signals, such as accuracy and the rate of hint usage, and recommends whether the challenge should be maintained, eased, or increased, with the aim of keeping the learner within a productive range. The decision logic of this component is shown in Figure 5.11.

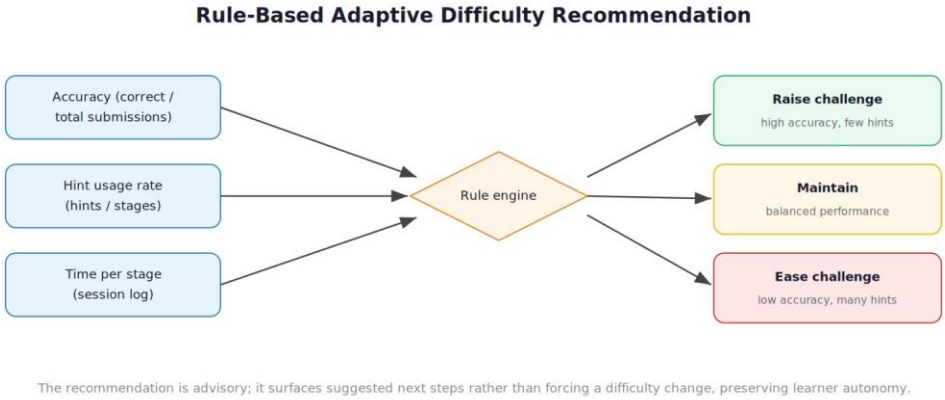


Figure 15 Rule-based adaptive-difficulty recommendation from observed performance signals

As Figure 5.11 indicates, the component is rule-based and operates as a recommendation layer rather than as a trained model: it derives a suggestion from accuracy, hint usage, and time signals but leaves the learner in control, preserving the autonomy that supports intrinsic motivation as discussed in Chapter 2.

5.6 Data Flow

The movement of data through the system is summarized in the data-flow diagram in Figure 5.12, which shows the external entities, the principal processes, and the data stores. The learner supplies credentials and, during play, commands and flags; the system returns evidence, feedback, and progression information; and the administrator curates the content that the processes read.

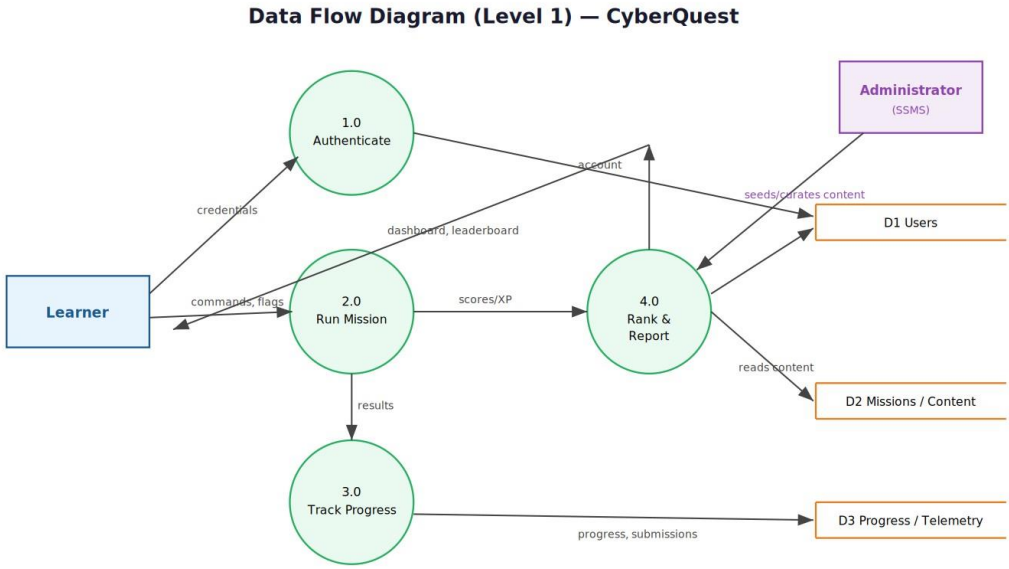


Figure 16 Level-1 data-flow diagram for the CyberQuest platform

As Figure 5.12 shows, the authentication process mediates access and writes to the user data store; the mission-play process reads content and records submissions; the progress tracking process updates the learner's state; and the ranking-and-reporting process derives the dashboard and leaderboard from stored progress. Centralizing state in the data stores, rather than in the application, is what allows the same information to drive multiple views consistently.

The data-flow view also clarifies why the telemetry stores are valuable. Every interaction of pedagogical interest, a flag submission, a hint request, the duration of a play session, flows into an append-only store as a side effect of normal operation. These stores therefore accumulate a complete record of how learners engage with the platform without any additional instrumentation, and because they are append-only they never interfere with the operational data that drives the live views. This is the concrete mechanism by which the platform prepares for the instructor-facing analytics identified

as future work: the data those analytics would need is already being captured, faithfully and unobtrusively, by the flows shown in the diagram.

5.7 Database Design

A defining design decision of CyberQuest is that all content and state reside in the relational database; nothing that a maintainer might reasonably wish to change, such as a mission's text, a stage's flag, a hint, a badge, or a level threshold, is embedded in program code. The conceptual schema is expressed in the entity-relationship diagram in Figure 5.13, which shows the core entities and the relationships among them.

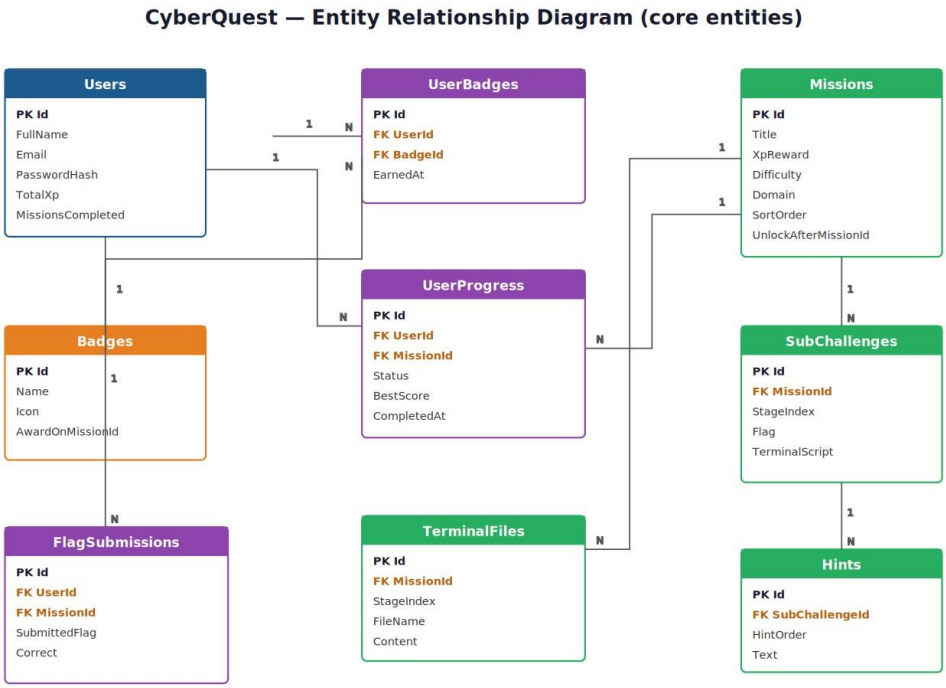


Figure 17 Entity-relationship diagram of the core CyberQuest database

As Figure 5.13 shows, the schema separates content entities from user entities. Content entities include missions, their stages, the hints and terminal files belonging to each stage, and the catalogs of levels and badges. User entities include the user account and the per-user records of progress, earned badges, and activity. The relationships capture the natural structure of the domain: a mission contains many stages, a stage contains many hints and terminal files, a user has one progress record per mission, and a user earns many badges.

The complete physical schema, comprising all twenty-two tables grouped into content, user, and telemetry categories, is shown in Figure 5.14. This normalized structure ensures that content can be extended, by inserting new rows, without any change to the application, and that each learner's state is recorded precisely and queried efficiently.

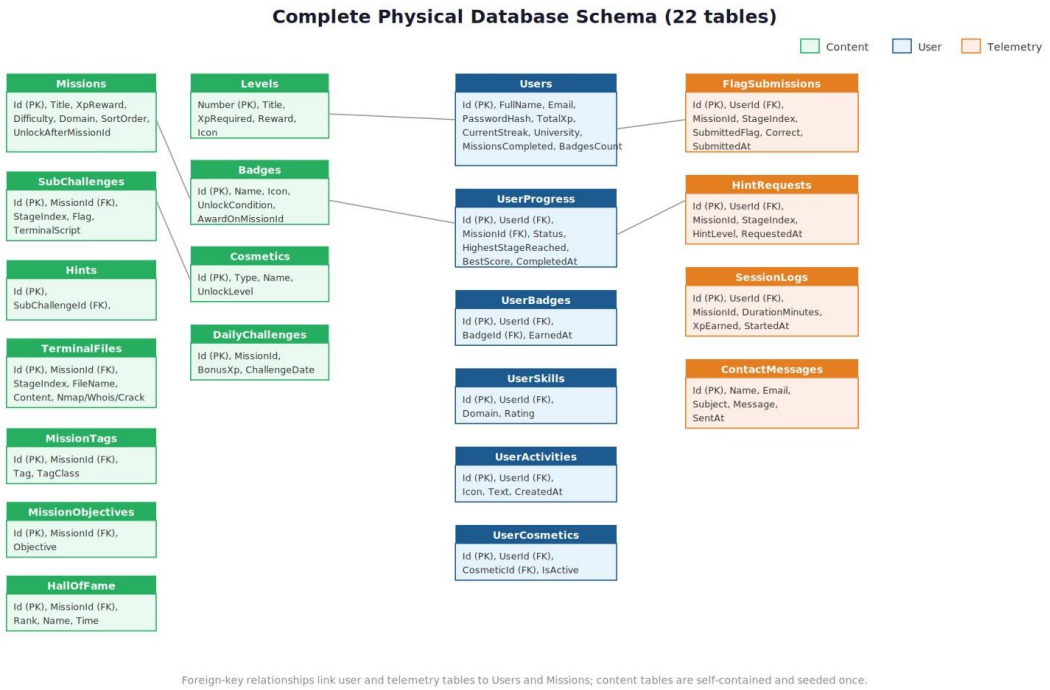


Figure 18 Complete physical database schema showing all twenty-two tables

As Figure 5.14 shows, the content tables are self-contained and are seeded once from a setup script, while the user and telemetry tables reference the users and missions through foreign keys and grow as learners interact with the platform. This separation directly realizes requirement NFR-24.

5.7.1 Schema Design Rationale

The schema is normalized to avoid redundancy and the update anomalies that redundancy causes. Each fact is stored in exactly one place: a hint belongs to exactly one stage, a stage to exactly one mission, and a progress record to exactly one pairing of a user and a mission. The uniqueness constraints introduced in the model configuration, on the pairing of user and mission in the progress table, on the pairing of user and badge in the earned-badges table, and on the ordering of stages within a mission and hints

within a stage, enforce this integrity at the database level, so that inconsistent data cannot be written even if application logic were to err.

The separation of the schema into content, user, and telemetry groups is not merely organizational; it reflects the different lifecycles and access patterns of the three kinds of data. Content is written once during seeding and thereafter read frequently but changed rarely. User data is written and read throughout a learner's activity and is the authoritative record of their state. Telemetry is append-only: submissions, hint requests, and session logs are added but never modified, which makes the telemetry tables a faithful audit trail from which analytics can later be derived without disturbing the operational data. Designing the schema around these distinct lifecycles is what allows future analytics features to be added without disrupting the operational parts of the system, satisfying requirement NFR-24.

5.7.2 Indexing and Query Efficiency

To satisfy the performance requirement NFR-3, the schema is indexed to support its most frequent queries. The per-user, per-mission progress lookup, performed whenever a mission is opened, is supported by the unique index on the user-mission pairing. The leaderboard ordering, performed whenever the leaderboard is viewed, is supported by ordering on the experience column of the user's table. The stage and hint lookups, performed during mission play, are supported by the composite indexes on their ordering columns. Because these indexes align with the actual query patterns of the application, the common operations remain efficient as the volume of user and telemetry data grows.

5.8 Class Design

The principal application, domain, and persistence classes and their relationships are shown in the class diagram of Figure 5.15. The diagram shows the mission and sub challenge domain classes, the user account and progress classes, the static mapping service, the database context, and the mission controller that coordinates the core loop.

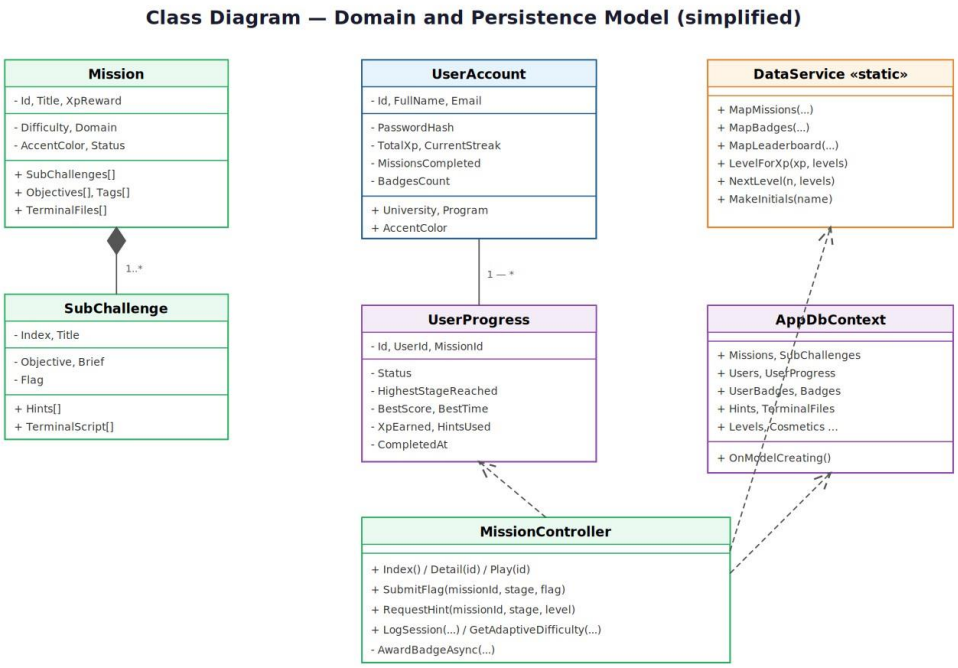


Figure 19 Class diagram of the principal domain, persistence, and controller classes

As Figure 5.15 shows, the mission controller depends on both the mapping service and the database context; the mapping service transforms persistence entities into the domain objects used by the views; and the database context exposes the full set of content and user tables. A mission is composed of many sub-challenges, and a user account is associated

with many progress records, reflecting the multiplicities in the entity-relationship model.

5.9 Security Architecture

Security is addressed at the architectural level rather than as an afterthought, which is fitting for a platform in the security domain. The design places three protective boundaries between an attacker and the system's sensitive assets. The first is the authentication boundary: no learner-specific data is accessible without a valid session, and sessions are established only through password verification against a salted, adaptive hash. The second is the data-access boundary: all interaction with the database passes through the object relational mapper, which parameterizes queries and thereby closes the injection vector that would otherwise be the most likely avenue of attack on a data-driven application. The third is the content boundary: the answer keys, that is, the expected flag values, never cross from the server to the client before a submission, so a learner cannot obtain them by inspecting the delivered page.

These boundaries correspond directly to the security requirements NFR-5 through NFR-9 and are realized by concrete implementation choices described in Chapter 7: BCrypt hashing, server-side sessions, exclusive use of the mapper for data access, and server-side retention of flag values. Designing these protections into the architecture, rather than bolting them on, ensures that the platform models the good security practice it aims to teach.

5.10 User Interface and Interaction Design

The interaction design of CyberQuest follows directly from its usability requirements and its narrative premise. The guiding principle is that everything a learner needs to attempt a stage should be visible at once, so that investigation is not interrupted by navigation. This principle produces the three-part mission-play layout, brief and hints, terminal, and submission, that realizes requirement NFR-10 and is shown in the implementation chapter.

The visual language is deliberately consistent across the platform. A dark, high contrast theme with accent colors evokes the security-operative narrative and, more practically, keeps the terminal, the focus of attention during play, legible. Consistent placement of navigation, status, and content across pages reduces the cognitive load of moving between the catalog, a mission, the dashboard, and the leaderboard, realizing requirement NFR-12. Feedback is immediate and unambiguous: a correct flag produces a clear success indication and visible progress, an incorrect flag produces a clear rejection that returns the learner to investigation, and experience and badge awards are surfaced promptly, realizing requirement NFR-11.

The information architecture of the platform is shallow and task-oriented. From any page the learner can reach the catalog, the dashboard, the leaderboard, and their profile in a single step, so that the structure never obscures the content. This shallow structure suits a learning tool, where the goal is to spend attention on the missions rather than on navigating the application. Together, these interaction-design choices ensure that the platform's engagement derives from the missions themselves rather than from any novelty of the interface, which is the appropriate emphasis for an educational system.

5.11 Deployment Model

The platform is deployed as three logical nodes, shown in the deployment diagram of Figure 5.16: a client device running a web browser, an application server running the .NET runtime and the CyberQuest application, and a database server running SQL Server. The client communicates with the application server over HTTPS, and the application server communicates with the database server using the SQL Server protocol.

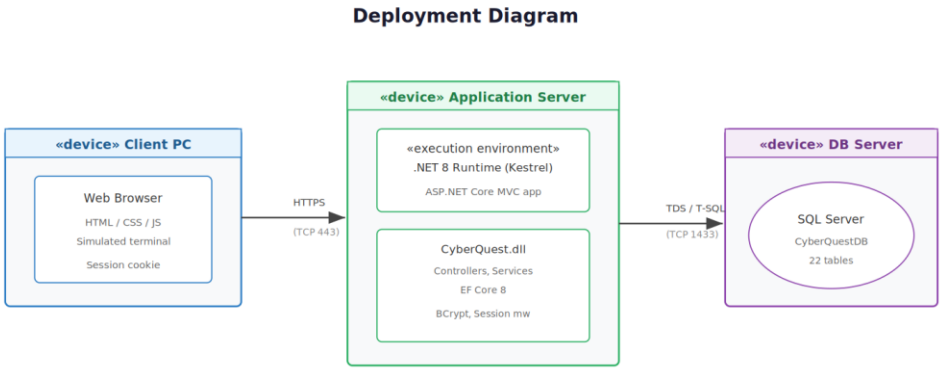


Figure 20 Deployment diagram showing the client, application, and database nodes

As Figure 5.16 shows, the three nodes may in practice be co-located on a single development machine, as in the validation environment for this thesis, or distributed across separate hosts in a production setting; because the tiers communicate only through well-defined protocols, the physical arrangement can change without altering the software. This satisfies the portability requirement NFR-21. The full physical schema described in this chapter is realized by the implementation presented in Chapter 7.

5.12 Model Consistency and Correspondence with the Implementation

Because a design is expressed through several complementary models, it is important that those models agree with one another and with the system actually built. This section makes that agreement explicit, first by cross-checking the models against each other, and then by mapping the architecture onto the concrete artifacts of the codebase.

5.12.1 Mutual Consistency of the Models

The models in this chapter describe the same system from different viewpoints, and the following correspondences hold across them. The four architectural layers of Figure 5.3

correspond one-to-one to the component groups of the component diagram in Figure 5.5, with identical dependency directions, as noted in Section 5.3. The entities of the class diagram in Figure 5.15 correspond to the tables of the entity-relationship diagram in Figure 5.13 and of the physical schema in Figure 5.14: each domain and persistence class named in the class diagram is realized by a table, and the multiplicities on the class associations (a mission has many stages; a user has many progress records) match the cardinalities of the corresponding entity relationships. The processes of the data-flow diagram in Figure 5.12 are performed by the controllers of the Application Layer, and its data stores are the tables of the Data Layer, so the DFD is a behavioral view of the same structure the class and entity-relationship diagrams describe. The states of the mission lifecycle in Figure 5.9 are exactly the values of the Status field of the User Progress table, so the state model and the data model agree. Finally, the three nodes of the deployment diagram in Figure 5.16 host, respectively, the Presentation Layer

(client device), the Application and Data-Access Layers (application server), and the Data Layer (database server), so the deployment view is a physical allocation of the same four layers. Table 5.2 summarizes these correspondences.

Table 18 Cross-model consistency: how the design views correspond

Models compared	Correspondence that holds
Architecture ↔ Component	Four layers map one-to-one to component groups; dependency directions identical.
Class ↔ ER ↔ Schema	Each class is realized by a table; association multiplicities equal relationship cardinalities.
DFD ↔ Class / Schema	DFD processes are Application-Layer controllers; DFD data stores are Data-Layer tables.
State ↔ Schema	Mission-lifecycle states equal the values of the User Progress Status column.
Deployment ↔ Architecture	Each node hosts a defined subset of the four layers.

5.12.2 Correspondence with the Implementation

The architecture is not only internally consistent but also faithful to the code that was built, and this correspondence can be checked directly. Table 5.3 maps each architectural layer to the concrete folders and classes of the delivered project, whose structure is shown in Chapter 7. The mapping is one-to-one: there is no architectural element without a counterpart in the code, and no major code artifact that falls outside the architecture. This

cross-check was performed by inspecting the codebase against the architecture diagram; where an early draft of the diagram and the code disagreed, the diagram was corrected so that it documents the system as actually implemented rather than as first envisaged.

Table 19 Correspondence between architectural layers and implementation artifacts

Layer	Realizing code artifacts
Presentation	Views/ (Razor views for Home, Mission, Dashboard, Leaderboard, Profile, Shared) and wwwroot/js/ (site.js, game.js).
Application	Controllers/ (Home, Auth, Mission, Dashboard, Leaderboard, Contact) and the Data Service mapping and adaptive-support helpers.
Data-Access	Data/AppDbContext.cs and the entity classes it exposes.
Data	The SQL Server database created and seeded by setup_database.sql.

Together, Tables 5.2 and 5.3 establish that the design models agree with one another and with the implementation, so the diagrams in this chapter can be read as an accurate specification of the delivered system rather than as an idealized sketch.

5.13 Chapter Summary

This chapter has presented the design of CyberQuest across its several views. The development approach was iterative and incremental, keeping the system runnable at every step. The architecture is a classic layered design, justified against monolithic and microservice alternatives, with data access and mapping confined to single components for maintainability. The mission and progression models were presented, including the experience economy, the level thresholds set above the base total to reward mastery, the mission lifecycle, and the progressive-hint and adaptive-difficulty mechanisms. The data flow, the conceptual and physical database schemas, the class model, the security architecture, the interaction design, and the deployment model were each described and related to the requirements they satisfy. Taken together, these views specify a coherent system in which a normalized, fully data-driven database underpins every feature, which is the design's central and unifying idea. The realization of this design is the subject of Chapter 7, and the detailed design of the individual missions is the subject of Chapter 6.

Chapter 6

Detailed Mission Design

While Chapter 5 presented the general mission model, this chapter examines the design of each of the six missions individually. For each mission it states the learning outcome, describes the narrative frame, details the stages and the investigative action each requires, explains the evidence exposed through the terminal, and notes the hints that scaffold the learner. This level of detail both documents the delivered content and illustrates how the general model accommodates a range of cybersecurity domains.

6.1 Design Principles Common to All Missions

Every mission in CyberQuest is constructed according to a shared set of principles that follow from the pedagogical foundations of Chapter 2. First, each mission targets a single, clearly stated learning outcome, so that the learner's effort is focused. Second, each mission is decomposed into ordered stages that build toward that outcome, so that difficulty rises gradually. Third, each stage is cleared only by performing the investigative action it is designed to teach, so that success constitutes evidence of the skill. Fourth, each stage provides evidence that is faithful in form to real practice, so that the experience transfers. Fifth, each stage is scaffolded by an ordered set of hints that guide without simply revealing, so that a stuck learner can progress while preserving the challenge. These principles are applied consistently across the six missions described below.

A further principle concerns narrative coherence. Each mission is framed as a short story with a role, a situation, and a goal, and the stages are arranged so that clearing one advances the story toward its resolution. This narrative framing is not decorative: it supplies the context that transforms an abstract exercise into a situated task, and it gives the learner a reason to care about the outcome beyond the accumulation of points. The narrative also provides a natural ordering for the stages, since the story dictates that one must, for example, detect an intrusion before one can contain it. In this way the narrative and the pedagogy reinforce one another, which is the central design idea of the platform applied at the level of an individual mission.

A final shared principle is proportionality of reward. The experience awarded by a mission is scaled to its difficulty and to the number and complexity of its stages, so that the progression economy remains fair and the incentive to attempt harder missions is preserved. This proportionality, visible in the catalog of Chapter 8, ensures that the reward structure reinforces rather than distorts the intended learning path.

6.2 Mission 1: Smart City Power Grid Hack

Learning outcome. On completing this mission, the learner is able to: (i) *identify* a rogue host within an industrial control network from scan output; (ii) *analyze* a captured control command to determine the attacker's intent; and (iii) *select* the containment action that isolates the compromised segment. Each verb names an action the learner performs to clear a stage, so the outcome is directly observable rather than a claim about internal understanding.

Narrative frame. The learner is an incident responder for a municipal utility. An operator has noticed that a section of the smart-grid supervisory control network is behaving erratically, and the learner is dispatched to investigate before the anomaly can affect physical infrastructure.

Stages. In the first stage the learner uses the simulated terminal to run a network scan and must read its output to distinguish legitimate control devices from a rogue host that has appeared on the network. In the second stage the learner inspects a captured control command associated with that host and must interpret it to determine the attacker's intent. In the third stage the learner selects, from the available network operations, the containment action that isolates the compromised segment without disrupting the wider grid.

Evidence and scaffolding. The terminal exposes a realistic scan listing, a set of device descriptors, and a captured command log. The hints move from a general reminder about how rogue hosts differ from known devices, through a pointer to the specific field that reveals malicious intent, to an explicit identification of the containment action. This mission is placed first because its investigative pattern, scan, interpret, act, recurs throughout the catalog.

Design notes. The power-grid scenario was chosen as the opening mission for several reasons. Its subject, the protection of critical infrastructure, immediately conveys why security matters, which helps engage a learner at the outset. Its investigative pattern,

scanning a network, interpreting the evidence, and choosing a response, is the fundamental pattern that later missions elaborate, so establishing it here pays dividends throughout the catalog. And its operational-technology setting introduces, gently, the idea that security extends beyond conventional information technology into systems that control the physical world, a theme that recurs in the automotive mission. The mission is kept to three stages of modest difficulty so that a newcomer can succeed and gain confidence before the catalog's demands increase.

6.3 Mission 2: Smart Home IoT Hack

Learning outcome. On completing this mission, the learner is able to: (i) *enumerate* the devices on a home network and read their descriptors; (ii) *identify* the device running on default or weak credentials as the point of compromise; and (iii) *recommend* the network-segmentation strategy that isolates untrusted devices from sensitive ones.

Narrative frame. The learner is asked to help a household whose smart-home network has been behaving suspiciously since a new device was added. The learner must find the weak device and recommend a response.

Stages. The first stage requires enumerating the devices on the home network and reading their descriptors. The second stage requires recognizing which device is running on default or weak credentials, the entry point exploited by the attacker. The third stage requires selecting the segmentation strategy that isolates untrusted devices from sensitive ones.

Evidence and scaffolding. The terminal exposes a device inventory with configuration details, including a device whose credentials are conspicuously weak to an attentive reader. The hints progress from a reminder that IoT devices often ship with default credentials, to a prompt to compare the configurations, to an explicit statement of the segmentation principle. The mission reinforces the reading-of-evidence skill from Mission 1 while introducing a distinct domain.

Design notes. Placing the IoT mission second consolidates the evidence-reading skill introduced in the power-grid mission while moving to a domain, the connected home, that is familiar and concrete to most learners. This familiarity lowers the barrier to engagement and makes the lesson, that convenience devices are often the weak link, immediately relatable. The mission also introduces the defensive concept of

segmentation, which recurs as a general principle throughout security, so its inclusion early in the catalog seeds an idea that later missions can assume.

6.4 Mission 3: Phishing Awareness Challenge

Learning outcome. On completing this mission, the learner is able to: (i) *inspect* a sender address and message headers to detect spoofing; (ii) *compare* an embedded link's displayed text with its actual destination to identify a malicious URL; and (iii) *name* the psychological pressure tactic the message employs.

Narrative frame. The learner receives a suspicious message purporting to come from a trusted institution and must decide whether it is genuine, examining it as a security-aware user would.

Stages. The first stage requires inspecting the sender address and message headers to detect spoofing. The second stage requires examining an embedded link, comparing its displayed text with its actual destination, to identify a malicious URL. The third stage requires naming the psychological pressure tactic, such as false urgency or authority, that the message employs.

Evidence and scaffolding. The terminal exposes the raw message, including headers and the true target of the link. The hints move from a reminder that the visible sender can be forged, to a prompt to hover conceptually over the link and compare destinations, to an explicit naming of the manipulation tactic. This mission develops a skill, resistance to social engineering, that complements the more technical missions.

Design notes. The phishing mission is deliberately independent of the others in its prerequisites, so that it can be attempted early regardless of a learner's progress on the technical missions. Its inclusion reflects the reality that many real compromises begin not with a technical exploit but with a human being deceived, and that resistance to social engineering is as much a security skill as any technical competence. By requiring the learner to name the specific manipulation tactic, the mission moves beyond a vague sense that a message is "suspicious" toward a precise, transferable ability to classify how such messages work.

6.5 Mission 4: Social Media Password Guessing

Learning outcome. On completing this mission, the learner is able to: (i) *explain* why a specific weak credential is guessed quickly, relating password strength to the feasibility

of guessing; (ii) *recover* a weak hashed password by matching it against a dictionary; and

(iii) *recommend* multi-factor authentication and passphrases as defenses, justifying each.

Narrative frame. The learner investigates how an attacker gained access to a social media account and must both analyze the weakness and recommend the defense.

Stages. The first stage requires reasoning about why a particular weak credential was guessed quickly, connecting password strength to the feasibility of guessing. The second stage requires recovering a weak hashed password from provided material, giving concrete experience of how weak hashing and weak passwords compound. The third stage requires articulating the protective value of multi-factor authentication and passphrases.

Evidence and scaffolding. The terminal exposes account metadata, a weak password hash, and a small dictionary. The hints progress from a reminder about common weak password patterns, to guidance on matching the hash against the dictionary, to an explicit statement of the defensive measures. This mission is positioned as a prerequisite for the two most advanced missions because the authentication concepts it develops recur in them.

Design notes. The password mission occupies a pivotal position in the catalog. It is the first mission marked intermediate rather than beginner, signaling a step up in demand, and it is a prerequisite for the two most advanced missions. This positioning reflects the centrality of authentication to security: a learner who can explain why weak credentials fail and can recover one in practice is better prepared for the more specialized scenarios that follow. By having the learner actually recover a weak password from a hash and a dictionary, rather than merely reading that weak passwords are risky, the mission makes the lesson concrete and memorable.

6.6 Mission 5: Self-Driving Car Hijack

Learning outcome. On completing this mission, the learner is able to: (i) *establish* a baseline of normal messages from a capture of controller-area-network bus traffic; (ii) *identify* a forged message by its identifier and timing; and (iii) *select* the integrity mechanism that would allow the vehicle to detect such forgery.

Narrative frame. The learner analyzes anomalous behavior in an autonomous vehicle whose internal network appears to be carrying forged messages.

Stages. The first stage requires inspecting a capture of bus traffic to establish a baseline of normal messages. The second stage requires identifying a forged message by its identifier and timing, distinguishing it from legitimate traffic. The third stage requires selecting the integrity mechanism that would allow the vehicle to detect such forgery.

Evidence and scaffolding. The terminal exposes a controller-area-network traffic capture with message identifiers and payloads, among which the forged message can be identified by careful comparison. The hints move from an explanation of how the bus normally behaves, to a prompt to look for a message that violates the expected pattern, to an explicit statement of the integrity mechanism. This mission has the highest experience reward, reflecting its difficulty and its position late in the progression.

Design notes. The autonomous-vehicle mission is the most advanced in terms of domain specialization, introducing embedded and automotive security concepts that most learners will not have encountered. Its high experience reward reflects both its difficulty and its role as an aspirational capstone of the technical missions. The mission builds directly on the evidence-reading skill established from the first mission onward, applying it to an unfamiliar kind of evidence, bus traffic, which demonstrates that the investigative pattern the platform teaches generalizes across domains. Like the power-grid mission, it reinforces the theme that security increasingly concerns systems that act on the physical world, closing a thematic loop opened at the start of the catalog.

6.7 Mission 6: DDoS Defense Simulation

Learning outcome. On completing this mission, the learner is able to: (i) *triage* live traffic telemetry to detect a volumetric attack; (ii) *classify* each attack vector by assigning its correct MITRE ATT&CK identifier; (iii) *sequence* countermeasures in a technically correct order to restore service; and (iv) *compose* a structured post-incident report. These four abilities correspond one-to-one to the mission's four stages, justified below.

Narrative frame. The learner is the on-call defender for an online service that comes under a large, multi-vector denial-of-service attack, and must bring the service back within its performance target while following professional process.

Stages, and why there are four. This mission has four stages, one more than the others. The number four is not arbitrary: the stages are placed in direct correspondence with the phases of the NIST SP 800-61 computer-security incident-handling process [10], which is the recognized professional standard for responding to an incident of this kind. That process defines detection and analysis, containment, eradication and recovery, and post-incident activity; the mission realizes these as *triage*, *classification*, *mitigation*, and *documentation* respectively, collapsing containment and eradication-and recovery into a single mitigation stage because, for a denial-of-service incident, containing and recovering from the attack are performed by the same countermeasures. Each stage also exercises a distinct cognitive skill, so the four-way decomposition is pedagogically as

well as procedurally justified:

- Stage 1 — Triage (detection and analysis). The learner reads live traffic telemetry to detect that a volumetric attack is under way. Skill: evidence reading under noise.
- Stage 2 — Classification (analysis). The learner assigns each of three attack vectors its correct MITRE ATT&CK identifier, distinguishing network-layer from application layer techniques. Skill: applying a professional taxonomy.
- Stage 3 — Mitigation (containment, eradication, and recovery). The learner deploys countermeasures in the technically correct order while a service-level score recovers. Skill: ordered decision-making under a recovery objective.
- Stage 4 — Documentation (post-incident activity). The learner assembles a structured post-incident report. Skill: technical communication.

Dropping any stage would omit a phase of the standard process and a distinct skill; adding a fifth would split a phase that professional practice treats as one. Four stages is therefore the smallest decomposition that faithfully mirrors the standard while giving each phase its own learning objective, which is why this mission, uniquely, departs from the three-stage pattern of the others.

Evidence and scaffolding. The terminal exposes traffic telemetry, a taxonomy reference, a set of available countermeasures, and a recovering service-level indicator, as illustrated in the case study of Section 8.5. The hints for each phase scaffold that phase specifically: the triage hints point toward the volumetric signature, the classification hints recall the layer-based split of the taxonomy, the mitigation hints indicate ordering

constraints, and the documentation hints recall the required elements of a report. As the most elaborate mission, it integrates skills, evidence reading, taxonomy application, ordered decision-making, and documentation, that the earlier missions develop individually.

Design notes. The DDoS mission is the capstone of the catalog, and its four-phase structure is a deliberate departure from the three-stage pattern of the other missions, reflecting its greater scope. It is positioned to unlock after the authentication mission because, while it does not depend on authentication concepts directly, it presupposes the investigative maturity that the earlier missions build. Its alignment with the NIST incident-handling process and the MITRE ATT&CK taxonomy is intentional: by the time a learner reaches this mission, they are ready to encounter the professional frameworks that structure real practice, and the mission gives them a first, guided experience of applying those frameworks end to end. The service-level indicator, discussed at length in Section 8.5, is the mission's distinctive feedback device and exemplifies how the platform can make the consequences of a learner's decisions vivid and immediate.

6.8 Coverage and Progression Rationale

Taken together, the six missions cover network and operational-technology security, IoT security, social engineering, authentication, embedded and automotive security, and network defense and incident response. Their ordering, encoded in the unlock structure of Chapter 5, is deliberate: the earlier missions establish the fundamental investigative pattern and the authentication concepts on which the later, more demanding missions depend. This coverage and ordering ensure that a learner who progresses through the catalog encounters a coherent and cumulative curriculum rather than a set of disconnected exercises, directly addressing the fragmentation problem identified in Chapter 1.

6.8.1 A Rubric for Difficulty Levels

The difficulty label attached to each mission is not assigned by intuition but by a consistent rubric, stated in Table 6.1, so that the three levels, Beginner, Intermediate, and Advanced, mean the same thing across the catalog. The rubric rates a mission on three objective criteria: the domain prior knowledge it assumes, the highest cognitive operation it requires (using the measurable verbs of the learning outcomes), and the interdependence of its stages. A mission's level is the highest level for which it meets the

criterion on any dimension, which prevents a superficially short mission from being under-rated when its reasoning is demanding.

Table 20 Rubric defining the three difficulty levels

Level	Prior knowledge assumed	Highest cognitive operation	Stage interdependence
Beginner	None beyond everyday computer literacy	Identify / read evidence	Stages largely independent
Intermediate	A prior mission's concepts (e.g. authentication)	Analyze / recover / sequence	Later stages depend on earlier findings
Advanced	A specialized domain most learners have not met	Select a defense in an unfamiliar domain	Stages form a single connected investigation

Applying this rubric yields the level assignments in Table 6.2. Note that the number of stages is deliberately *not* a criterion in the rubric: the DDoS mission has four stages yet is rated Intermediate rather than Advanced, because although it integrates several skills, each individual skill has already been met in an earlier mission and it assumes no new specialized domain, whereas the Autonomous Vehicle mission is rated Advanced on the strength of the unfamiliar embedded domain it introduces despite having only three stages. Separating stage count from difficulty in this way keeps the rubric honest and explains what might otherwise look like an inconsistency.

Table 21 Difficulty progression and skill dependencies across the mission catalog

Mission	Level	Stages	Rubric justification
Power Grid Hack	Beginner	3	No prior knowledge; identify/read; independent stages
Smart Home IoT	Beginner	3	No prior knowledge; identify/recommend; independent stages
Phishing Challenge	Beginner	3	No prior knowledge; inspect/compare/name; independent stages
Password Security	Intermediate	3	Analyze/recover; introduces authentication concepts reused later
DDoS Defense	Intermediate	4	Sequence/compose; integrates prior skills; no new domain
Autonomous Vehicle	Advanced	3	Select a defense in an unfamiliar embedded domain

As Table 6.2 shows, the catalog is not a flat set of equivalent challenges but a graded sequence, ordered by the rubric of Table 6.1, in which later missions presuppose the

competencies developed earlier. This graded structure is the hallmark of a coherent curriculum.

6.9 Chapter Summary

This chapter has detailed the design of each of the six missions, showing how the general mission model of Chapter 5 accommodates a range of cybersecurity domains. Each mission was described in terms of its learning outcome, narrative frame, stages, evidence, scaffolding, and design rationale, and the catalog as a whole was shown to form a graded, cumulative curriculum rather than a set of disconnected exercises. The consistent application of shared design principles, a single focused outcome, staged decomposition, action-requiring flags, faithful evidence, progressive hints, and narrative coherence, across six distinct domains demonstrates both the discipline of the design and the expressiveness of the underlying model. The implementation that brings these missions to life is described in Chapter 7, and their validation is reported in Chapter 8.

Chapter 7

Implementation

This chapter describes how the design presented in Chapter 5 was realized. It covers the technology stack, the organization of the application, the structure of the domain and persistence classes, the principal runtime workflows, the database schema and its seeding, the user interface, and the security measures adopted. Representative code and schema listings are provided where they clarify the implementation, with fuller listings in the appendix.

7.1 Development Environment

The platform was developed and validated in a self-contained environment comprising the .NET 8 software development kit, an integrated development environment for editing and debugging, a local instance of SQL Server for persistence, and SQL Server Management Studio for administering the database and running the setup script. The application and the database were co-located on a single machine, which is consistent with the single-machine deployability constraint and which simplified the debugging of the interactions between the application and the database during development. Version control was used to track changes across the iterative increments, so that the state of the system at each stage was recoverable and the effect of each change was isolable, a discipline that proved valuable during the database migration.

This environment is modest and reproducible, requiring only freely available or standard tooling, which is consistent with the goal of a platform that can be adopted in resource-constrained educational settings. The same environment that suffices for development also suffices for a small-scale deployment for a class, since the application and database can run on a single machine; larger deployments would distribute the tiers as described in Chapter 5 but would require no change to the application itself.

7.2 Technology Stack

The platform is built on Microsoft's .NET 8 platform using the ASP.NET Core Model View-Controller framework for the web application and Entity Framework Core as the object-relational mapper. The persistent store is Microsoft SQL Server, administered

through SQL Server Management Studio. Views are authored in the Razor markup language, and client-side behavior is implemented in plain JavaScript without a heavyweight front-end framework, which keeps the delivery lightweight. Passwords are protected using a widely used implementation of the BCrypt adaptive hashing function [32]. The principal technologies and their roles are summarized in Table 7.1.

Table 22 Technology stack and the role of each component

Component	Technology	Role in the platform
Runtime / language	.NET 8 / C#	Application platform and language
Web framework	ASP.NET Core MVC	Routing, controllers, server-rendered views
ORM	Entity Framework Core 8	Object-relational mapping and LINQ queries
Database	Microsoft SQL Server	Persistent storage of content and user data
View layer	Razor	Server-side HTML templating
Client scripting	JavaScript	Simulated terminal and asynchronous requests
Password hashing	BCrypt	Adaptive one-way hashing of credentials
Administration	SQL Server Management Studio	Content curation and monitoring

7.3 Application Structure

The application follows the conventional structure of an ASP.NET Core MVC project. Controllers handle incoming requests and coordinate the response: a home controller renders the landing page and featured missions; an authentication controller handles registration, login, and logout; a mission controller drives the mission-play loop, flag submission, and hint requests; a dashboard controller assembles the learner's personalized overview; and further controllers serve the leaderboard, the profile pages, and the contact form.

A static mapping service, DataService, translates database entities into the view models consumed by the Razor views and provides pure helper computations such as mapping an experience total to a level. Keeping this logic in a single service, and confining all database access to the AppDbContext class, localizes the two concerns that change most often. The overall organization of the project into these components is shown in Figure 7.1.



Figure 21 Directory structure of the CyberQuest project

As Figure 7.1 shows, the project follows the conventional ASP.NET Core layout: controllers, models, and views occupy their standard folders; the data-access concerns are gathered in a data folder; the client assets, including the terminal script, reside under the web-root folder; and the entry point, configuration, and database setup script sit at the root. This conventional structure aids comprehension, realizing requirement NFR-19, and makes the project immediately navigable to anyone familiar with the framework. The entry point of the application configures the services, the database connection, and the request pipeline, as shown in Listing 7.1.

```
1  var builder = WebApplication.CreateBuilder(args);
2  builder.Services.AddControllersWithViews();
3  builder.Services.AddSession(o => {
4          o.IdleTimeout = TimeSpan.FromHours(8);
5          o.Cookie.HttpOnly = true;
6  });
7  builder.Services.AddDbContext<AppDbContext>(opt =>
8          opt.UseSqlServer(builder.Configuration
9                  .GetConnectionString("DefaultConnection")));
10 var app = builder.Build();
11 app.UseHttpsRedirection();
12 app.UseStaticFiles();
13 app.UseRouting();
14 app.UseSession();
15 app.MapControllerRoute("default",
16         "{controller=Home}/{action=Index}/{id?}");
17 app.Run();
```

Listing 7.1: Application startup: service and pipeline configuration

7.4 Persistence and the Data Context

All persistence is mediated by Entity Framework Core through the AppDbContext class, which exposes one strongly typed collection for each table and configures the entity model. A notable configuration detail is that the level entity uses a non-default primary key, and several relationships require unique indexes to preserve integrity; these are declared in the model-building method shown in Listing 7.2.

```
protected override void OnModelCreating(ModelBuilder b)
{
    // Level uses Number as its primary key (not the default Id)
    b.Entity<DbLevel>().HasKey(l => l.Number);

    b.Entity<UserProgress>()
        .HasIndex(p => new { p.UserId, p.MissionId }).IsUnique();
    b.Entity<UserBadge>()
        .HasIndex(u => new { u.UserId, u.BadgeId }).IsUnique();
    b.Entity<DbSubChallenge>()
        .HasIndex(s => new { s.MissionId, s.StageIndex }).IsUnique();
    b.Entity<DbHint>()
        .HasIndex(h => new { h.SubChallengeId, h.HintOrder }).IsUnique();
}
```

Listing 7.2: Entity model configuration in the database context

Because all content is read from the database at request time and mapped into domain objects, the application contains no hard-coded mission data. Listing 7.3 shows a representative fragment of the mapping service that assembles a domain mission from its constituent database rows.

```
return dbMissions.OrderBy(m => m.SortOrder).Select(m => new Mission {
    Id = m.Id, Title = m.Title, XpReward = m.XpReward,
    Difficulty = m.Difficulty, Domain = m.Domain,
    Objectives = dbObjectives.Where(o => o.MissionId == m.Id)
                                        .Select(o => o.Objective).ToList(),
    SubChallenges = dbSubs.Where(s => s.MissionId == m.Id)
        .OrderBy(s => s.StageIndex).Select(s => new SubChallenge {
            Index = s.StageIndex, Title = s.Title, Flag = s.Flag,
            Hints = dbHints.Where(h => h.SubChallengeId == s.Id)
                                    .OrderBy(h => h.HintOrder)
                                    .Select(h => h.Text).ToList()
```

```
12                  }).ToList()
13  }).ToList();
```

Listing 7.3: Mapping database entities to a domain mission (excerpt)

7.5 Key Workflows

7.5.1 Authentication

Registration and login are handled asynchronously. When a learner submits the registration form, the request is posted to the authentication controller, which validates the input, hashes the chosen password with BCrypt, and stores the new account. On subsequent logins, the submitted password is verified against the stored hash, and, on success, the learner's identity and summary state are placed in the session. The sequence of interactions for registration and login is shown in Figure 7.2.

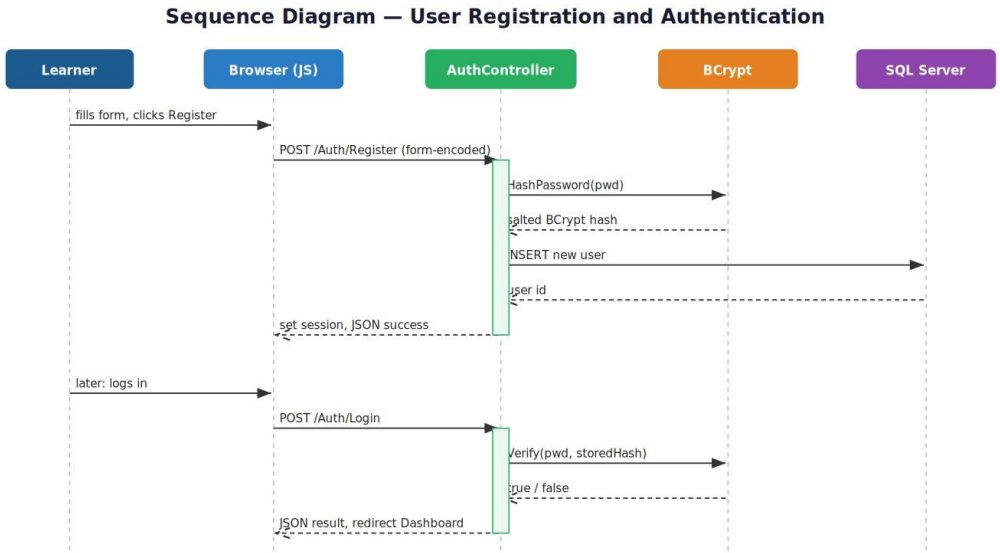


Figure 22 Sequence diagram for user registration and authentication

As Figure 7.2 illustrates, the plaintext password never reaches the database; only the salted BCrypt hash is stored, and verification is performed by the hashing library. The corresponding server-side login handler is shown in Listing 7.4.

```
1  var user = await _db.Users.FirstOrDefaultAsync(
2          u => u.Email.ToLower() == model.Email.ToLower());
3  if (user == null ||
4          !BCrypt.Net.BCrypt.Verify(model.Password, user.PasswordHash))
5          return Json(new { success = false,
6                  message = "Invalid email or password." });
```

```
user.LastLoginAt = DateTime.UtcNow;
await _db.SaveChangesAsync();
SetSession(user);
return Json(new { success = true,
        message = $"Welcome back, {user.FullName.Split(' ')[0]}." });
```

Listing 7.4: Login handler verifying credentials against the stored hash

7.5.2 Playing a Mission and Submitting a Flag

The core loop of the platform is playing a stage and submitting a flag. When a learner opens a mission, the mission controller loads the mission, its stages, and the associated terminal files from the database, applies the learner's progress to determine which stages are available, and renders the play view together with the stage data needed by the client. When the learner submits a flag, the controller compares it against the expected value, records the attempt, and, if the submission clears the final stage, awards experience, updates progress, unlocks the next mission, and grants any earned badges. This sequence is shown in Figure 7.3.

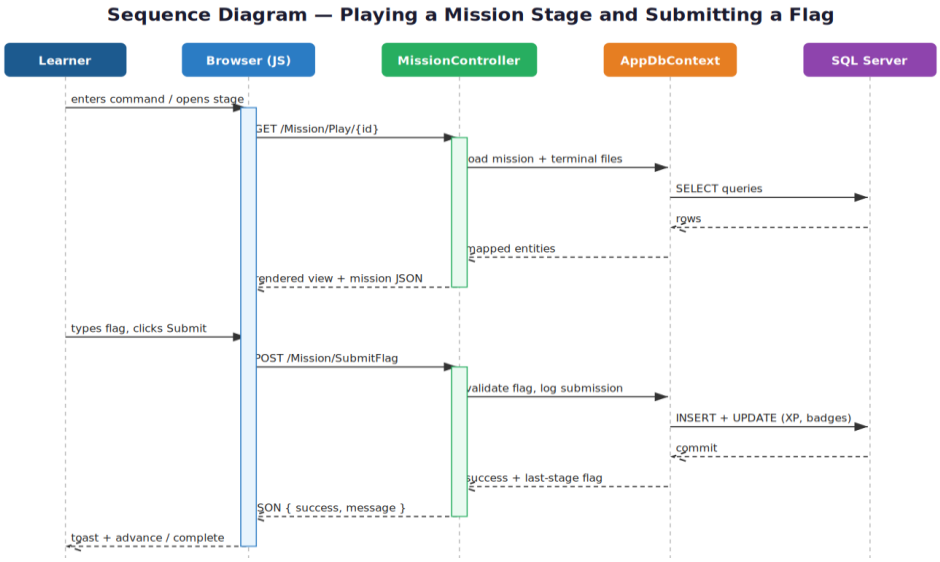


Figure 23 Sequence diagram for playing a mission stage and submitting a flag

As Figure 7.3 shows, every submission is logged whether or not it is correct, which provides the raw telemetry from which learning analytics can later be derived. The essence of the submission handler, including the application of progression side effects, is shown in Listing 7.5.

```
var correct = submitted == expected;
bool isLast = correct && stage == totalStages;
_db.FlagSubmissions.Add(new FlagSubmission {
        UserId = userId, MissionId = missionId,
        StageIndex = stage, Correct = correct });
```

```
6  if (isLast) {
7          prog.Status = "completed";
8          user.TotalXp += mission.XpReward;
9          await AwardBadgeAsync(userId, badge.Id, user);
10 }
11 await _db.SaveChangesAsync();
```

Listing 7.5: Flag submission with progression side effects (excerpt)

7.5.3 Access Control: Preventing Access to Locked Missions

A learner must be prevented both from acting without being logged in and from reaching a mission that their progress has not yet unlocked. The platform enforces this on the server, at every action, rather than relying on the client to hide unavailable options. Two checks are applied in sequence whenever a mission action is invoked.

First, an *authentication* check confirms that the request belongs to a logged-in learner. Every action that touches learner-specific state begins by reading the learner's identity from the server-side session; if no identity is present, the action returns an unauthorized response and does no further work. Because the identity is held in the session and not in any client-supplied field, it cannot be forged by editing the request.

Second, an *authorization* check confirms that the requested mission is actually unlocked for that learner. A mission is unlocked only if it has no prerequisite or if the learner's progress record for its prerequisite mission has the status completed. The controller recomputes this condition from the database on every request; it never trusts a mission identifier simply because it arrived in a URL. If a learner requests a mission whose prerequisite is not complete, whether by guessing its address or by manipulating the page, the controller returns a "mission locked" response and loads no stage data. Listing 7.6 shows the guard that both actions apply before any mission content is loaded.

```
1  var userId = CurrentUserId();
2  if (userId == null)
3          return Unauthorized();                    // not logged in
4
5  var mission = await _db.Missions.FindAsync(      missionId);
6  if (mission == null) return NotFound();
7
8  if (mission.UnlockAfterMissionId is int prereq) {
```

```
        bool unlocked = await _db.UserProgress.AnyAsync(p =>
            p.UserId == userId && p.MissionId == prereq &&
            p.Status == "completed");
        if (!unlocked)
            return Json(new { success = false,
                              message = "This mission is locked. Complete its prerequisite first." });
    }
    // only now is the mission's stage data loaded
```

Listing 7.6: Server-side guard enforcing login and mission unlocking

Because this guard runs on the server for every mission action, the locking shown visually in the catalog (Figure 7.6) is a convenience for the learner, not the mechanism of enforcement: even a learner who bypasses the interface cannot open or submit to a locked mission. This places the access-control decision in the Application Layer, consistent with the responsibilities assigned in Table 5.1.

7.5.4 Hints and Adaptive Difficulty

The overall control flow of a stage attempt, including the interplay between investigation, hint requests, flag submission, and the adaptive-difficulty recommendation, is captured in the activity diagram in Figure 7.4. The diagram shows the loop a learner follows within a stage and how the system responds at each decision point.

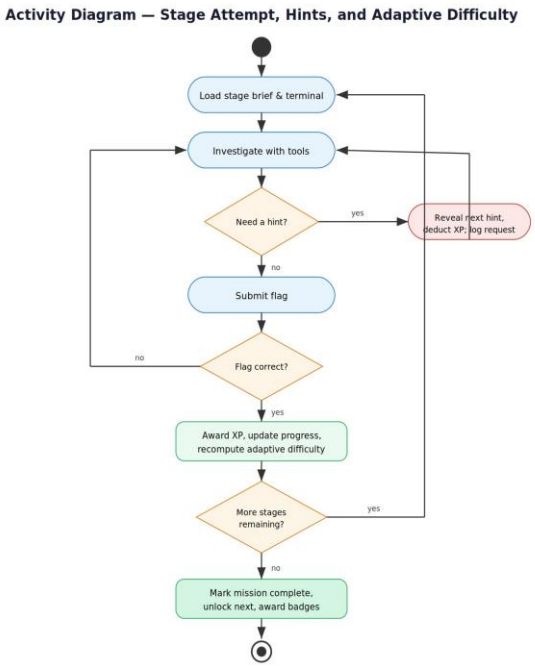


Figure 24 Activity diagram for a stage attempt, hints, and adaptive difficulty

As Figure 7.4 shows, a learner may investigate with the terminal tools, optionally request the next hint at a cost in experience, and submit a flag; an incorrect flag returns the learner to investigation, while a correct flag advances the stage and triggers the recompilation of the adaptive-difficulty recommendation. When the final stage is cleared, the mission is marked complete, the next mission is unlocked, and badges are awarded.

7.6 User Interface

The interface adopts a dark, high-contrast visual theme in keeping with the security operative narrative. The screenshots that follow are presented at a consistent width and use a uniform caption form, "Screenshot of the *X* screen: ...", so that they can be compared readily; the interface text in each is rendered at a size chosen to remain legible in print. Entry to the platform is through a combined login and registration panel, shown in Figure 7.5, which posts credentials asynchronously to the authentication controller.

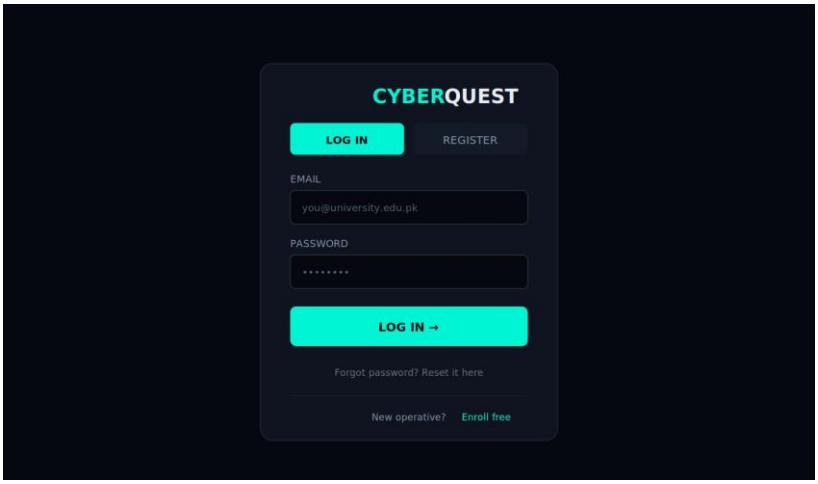


Figure 25 Screenshot of the login screen: the combined login and registration panel

After signing in, the learner arrives at the mission catalog, shown in Figure 7.6, which presents each mission as a card annotated with its domain, difficulty, reward, and completion state, and visibly locks missions whose prerequisites are unmet.

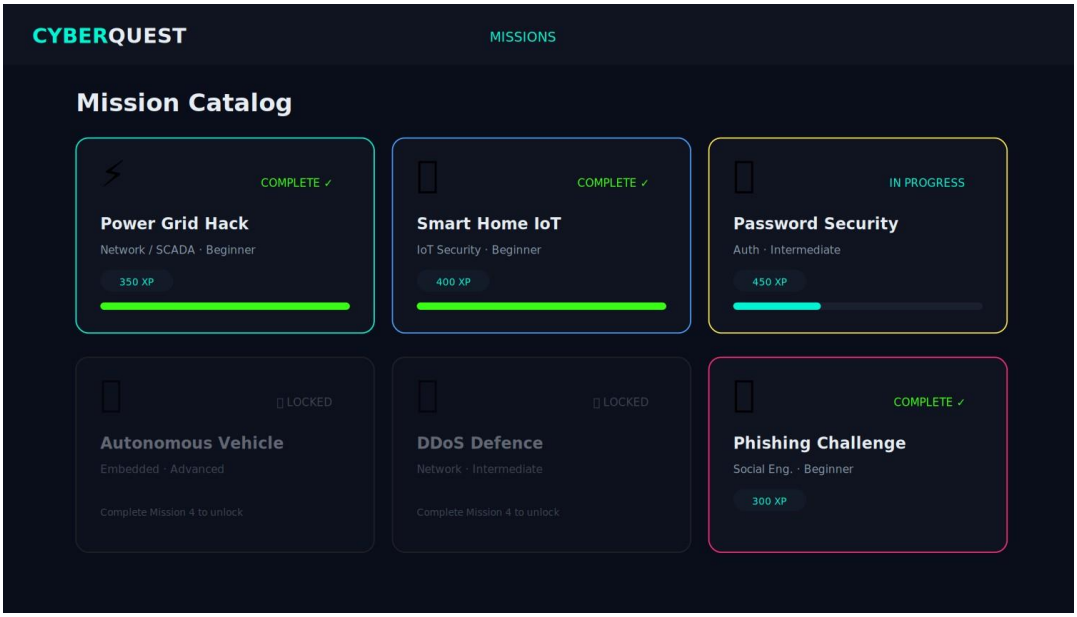


Figure 26 Screenshot of the mission-catalog screen: mission cards with domain, difficulty, reward, completion state, and locked missions

The mission-play screen, shown in Figure 7.7, presents the stage brief and progressive hints on the left, a simulated command-line terminal in the center, and a flag-submission control below it, so that all the elements needed to investigate and answer are visible at once, satisfying usability requirement NFR-10.



Figure 27 Screenshot of the mission-play screen: the stage brief and hints, the simulated terminal, and the flag-submission control

The personal dashboard, shown in Figure 7.8, summarizes the learner's experience, rank, completed missions, and badges, and displays per-mission progress and the current activity streak. Because these values are derived from the database, the dashboard reflects the learner's true state at all times.

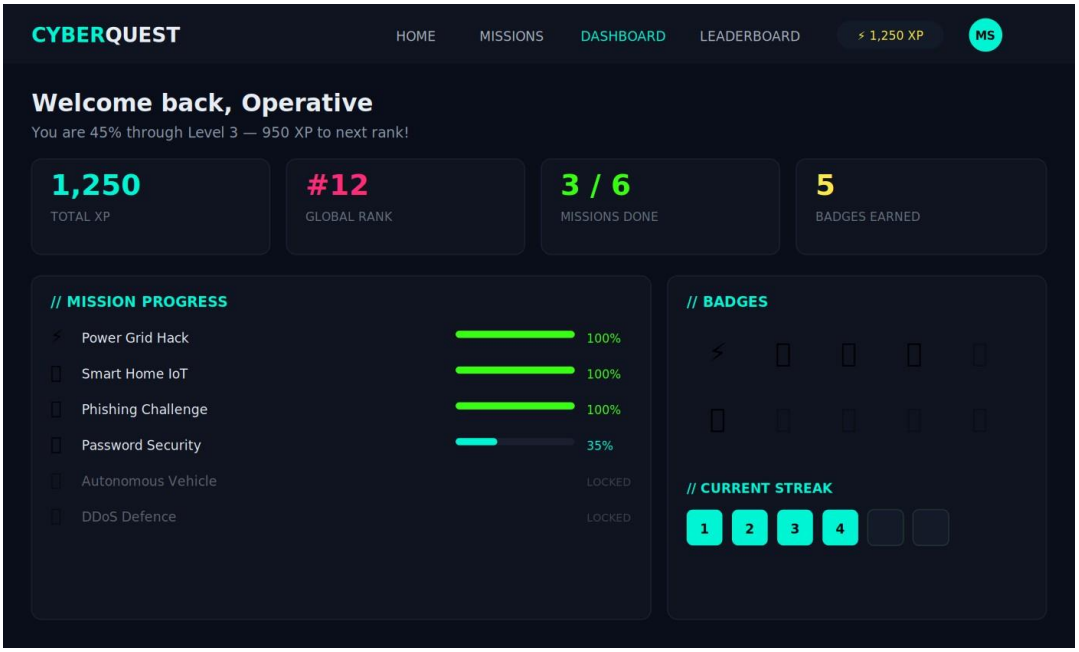


Figure 28 Screenshot of the dashboard screen: experience, rank, completed missions, badges, per-mission progress, and streak

The global leaderboard, shown in Figure 7.9, ranks learners by experience, presents the top three on a podium, and highlights the current learner's own row, realizing requirements FR-32 and FR-33.

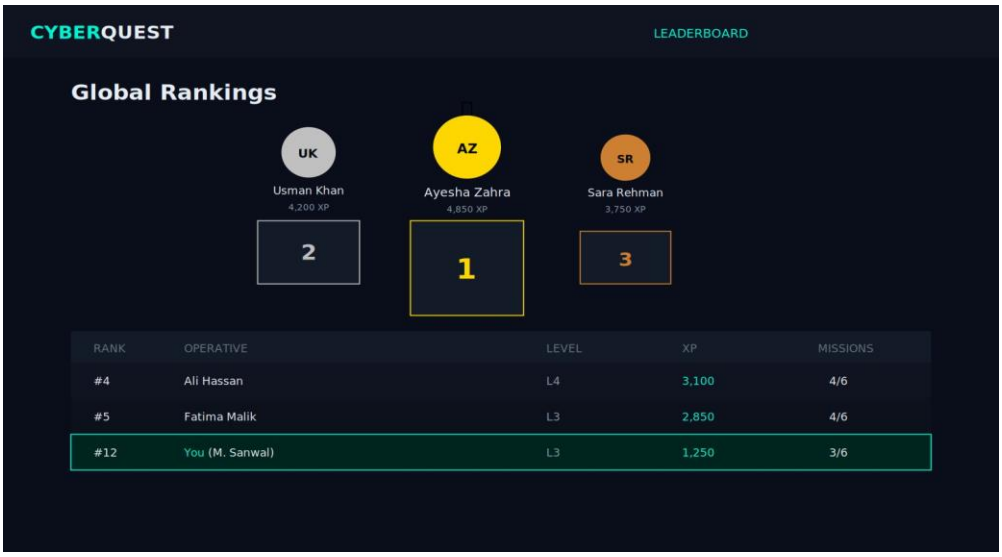


Figure 29 Screenshot of the leaderboard screen: the podium and the ranked table with the current learner highlighted

Figure 7.9: Screenshot of the leaderboard screen: the podium and the ranked table with the current learner highlighted

Finally, the profile page, shown in Figure 7.10, presents the learner's identity, level progression, badge showcase, and recent activity in a single view, realizing requirement FR-34. Like the dashboard, all of its content is derived from stored data.

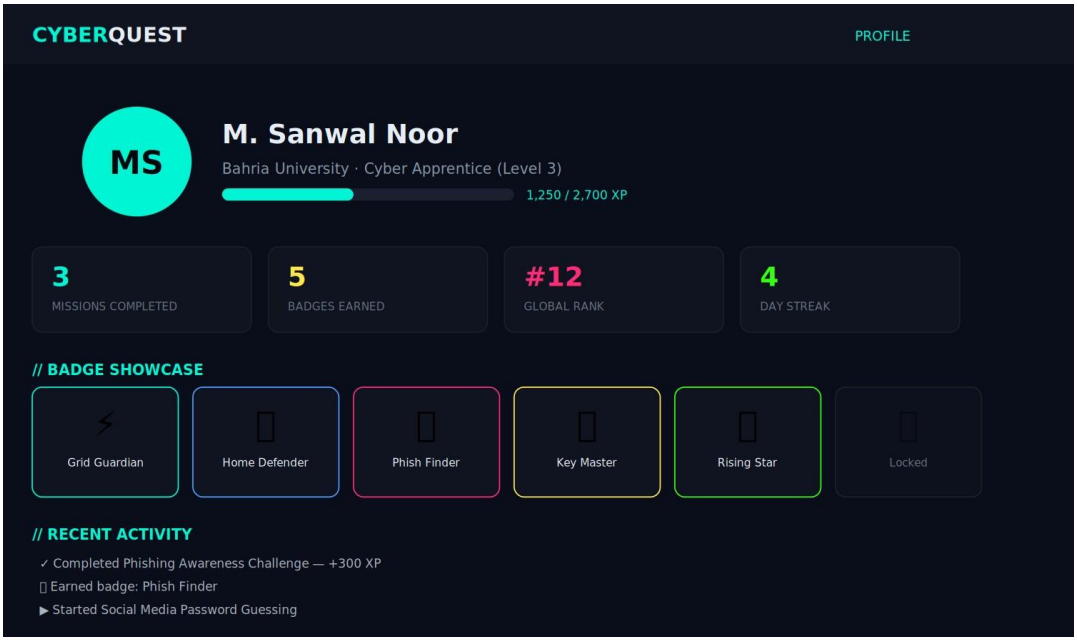


Figure 30 Screenshot of the profile screen: level progression, badge showcase, and recent activity

7.7 The View Layer

The presentation is implemented as a set of Razor views, one for each principal page, together with shared layout and partial views that factor out common structure such as the navigation bar and the status display. A Razor view combines static HTML with embedded server-side expressions that insert values from the view model, so that the page reflects the learner's current state. Because the view engine encodes inserted values by default, the views are protected against the cross-site scripting class of vulnerability described in Chapter 3.

The views are deliberately thin: they contain presentation logic only, such as choosing which badge icon to show or how to render a progress bar, and delegate all substantive computation to the controllers and the mapping service. This discipline keeps the boundary between presentation and logic clear and is the reason the appearance of a page can be changed without risk to the underlying behavior, as noted in the discussion of the layered architecture. Shared layout views ensure that the consistent visual

language required by NFR-12 is applied uniformly, since every page inherits the same surrounding structure and styling.

Client behavior is added by two small scripts. The first drives the asynchronous requests, submitting forms and flag attempts without a full page reload and updating the page with the structured responses the controllers return. The second implements the simulated terminal described below. Keeping the client logic in these two focused scripts, rather than adopting a large front-end framework, keeps the delivery lightweight and the behavior easy to follow, consistent with the browser-only constraint of NFR-23.

7.8 Client-Side Terminal

A distinctive element of the implementation is the simulated command-line terminal. It is important to be precise about what this component is and is not. It is *not* a real command line interface: there is no operating-system shell behind it, no process is spawned, no real file system is read, and no command the learner types is ever executed by any interpreter. It is, rather, a mock interface, implemented entirely in client-side JavaScript, that *imitates the appearance and behavior* of a terminal for a fixed, small vocabulary of commands. When a mission is played, the server serializes the stage data, the mission's evidence files, canned scan outputs, and the recognized command list, into the page. The client script parses the learner's typed input as plain text, matches it against the recognized commands, and prints a pre-supplied or looked-up string in response. The effect is faithful in form to real tools such as directory listings, file contents, and network scans, but nothing is computed or executed; the terminal only selects and displays text that the server already provided. This mock-interface design keeps the component responsive, satisfying requirement NFR-4, while preserving the investigative *feel* of a terminal. Throughout this thesis the component is referred to consistently as the *simulated terminal* to keep this distinction from a real command line clear.

The terminal maintains a small command vocabulary appropriate to each mission, typically including commands to list available evidence, to read a named file, to run a simulated scan, and to display help. Each command is dispatched to a handler that looks up the relevant stage data and formats a response. Listing 7.7 sketches the dispatch logic in simplified form.

```
1  function runCommand(input, stageData) {
2      const [cmd, ...args] = input.trim().split(/\s+/);
```

```
    switch (cmd) {
        case "ls":
            return stageData.files.map(f => f.name).join(" ");
        case "cat":
            const file = stageData.files.find(f => f.name === args[0]);
            return file ? file.content : `cat: ${args[0]}: not found`;
        case "scan":
            return stageData.scanOutput;
        case "help":
            return "Commands: ls, cat <file>, scan, clear, help";
        case "clear":
            clearScreen();
            return "";
        default:
            return `command not found: ${cmd}`;
    }
}
```

Listing 7.7: Client-side terminal command dispatch (simplified)

The security properties of this component should be stated carefully rather than claimed absolutely. The simulated terminal is safe *primarily because it is a restricted simulation*: it exposes no shell, performs no system or file-system access, spawns no process, and initiates no network activity beyond the ordinary requests the application already makes. Because the set of recognized commands is fixed and each merely returns text, there is no command-injection surface of the kind a real command interpreter would present; input that does not match a recognized command simply yields a "command not found" string. This is the sense in which the terminal "poses no security problem": the claim is not that client-side code is inherently safe, but that this particular component cannot reach any sensitive resource because it has no mechanism to do so.

Two residual considerations are worth noting for completeness. First, because the stage data is delivered to the client, any evidence placed in it is visible to a learner who inspects the page; the design accommodates this by keeping the expected flag values *out* of the serialized stage data entirely, so that answers cannot be recovered from the page source, in keeping with requirement NFR-9. Second, the client-side terminal is a convenience for the learner and is never trusted as an authority: the authoritative check of whether a stage is cleared is performed on the server when a flag is submitted (Section 7.5.2), so nothing a learner could do to the client-side script can advance their real progress. With these qualifications, the terminal achieves the pedagogical goal of an

authentic-feeling investigative interface while remaining a safe, self-contained simulation.

7.9 Configuration and Connection Management

The application externalizes its configuration, most importantly the database connection string, into a configuration file rather than embedding it in code, so that the same build can be deployed against different databases without recompilation. The connection string identifies the database server, the database name, and the authentication mode. During development, the application and the SQL Server instance were co-located, and integrated authentication was used together with a setting that trusts the local server certificate, which is convenient for a development environment. In a production deployment, the connection string would be adjusted to point at the production database and to use appropriate credentials, with no change to the application code. This externalization of configuration is a small but important contributor to the portability required by NFR-21.

7.10 Database Schema and Seeding

The physical database comprises content tables, user tables, and telemetry tables, as detailed in the schema of Figure 5.14. The complete schema is created and populated by a single SQL script, which an administrator runs once to provision the database. A representative excerpt, creating and seeding the levels table, is shown in Listing 7.8.

```
CREATE TABLE Levels (
NumberINT NOT NULL PRIMARY KEY,
TitleNVARCHAR(50) NOT NULL,
XpRequiredINT NOT NULL,
        AccentColor NVARCHAR(20) NOT NULL,
RewardNVARCHAR(100) NOT NULL
);
GO
INSERT INTO Levels VALUES
(1,'Ghost',0,'#7a9cc8','Starter terminal theme'),
(2,'Script Kiddie',500,'#4d9fff','Blue navbar accent'),
(3,'Cyber Apprentice',1200,'#00f5d4','Cyan profile frame'),
(4,'Packet Sniffer',2700,'#39ff14','Matrix terminal theme');
GO
```

Listing 7.8: Creating and seeding a content table (excerpt)

Because all content is inserted as data, extending the platform, for example by adding a new mission, requires only new rows rather than any change to the compiled application, directly realizing requirement NFR-17.

7.10.1 The Seeding Procedure and Its Reproducibility

Because the platform is fully data-driven, the database is not merely a store but part of the deliverable, and provisioning it must be reproducible: any administrator, on any suitable SQL Server instance, must be able to recreate an identical database. The single setup script, setup_database.sql, achieves this, and its structure is designed for repeatable execution. The script performs the following steps in a fixed order:

1. Create the database if it does not already exist, and select it for the remaining statements.
2. Create the content tables (missions, sub-challenges, hints, terminal files, levels, badges, and the other catalog tables), then the user tables, then the telemetry tables. This order respects foreign-key dependencies, so that a table is always created before any table that references it.
3. Seed the content tables with the fixed catalog: the six missions, their ordered stages, the hints and evidence for each stage, the level thresholds, and the badge definitions. The user and telemetry tables are intentionally left empty, since they hold runtime data rather than content.

Two properties make the procedure reproducible rather than merely runnable. First, it is *deterministic*: the script contains literal values, not randomly generated or environment dependent data, so two runs produce byte-for-byte identical content. The primary keys of content rows are fixed explicitly rather than left to the database, so that cross-references between content rows, a stage's mission, a hint's stage, are stable across runs and across machines. Second, the script is written to be *idempotent* at the level of a fresh provision: it guards object creation so that re-running it against an already-provisioned database does not duplicate content or error out, and an administrator who wishes to start clean can drop the database and re-run the single script to obtain an identical result. Because seeding is expressed entirely in this one ordered, deterministic, idempotent script, the exact database used for the evaluation in Chapter 8 can be reconstructed by any reader with the script and a SQL Server instance, which is precisely the reproducibility the non-functional requirement NFR-20 calls for.

7.11 Progression and Reward Implementation

The progression subsystem is implemented as a set of pure helper computations in the mapping service, combined with persistent updates applied by the controllers. Mapping an experience total to a level is a pure function over the levels table: the level is the highest one whose threshold does not exceed the learner's experience. Because the levels and their thresholds are stored as data, this computation adapts automatically if the thresholds are retuned, as they were during development. Listing 7.9 shows the level-computation helper.

```
public static DbLevel LevelForXp(int xp, List<DbLevel> levels)
{
    // levels are ordered ascending by threshold
    DbLevel current = levels.First();
    foreach (var lvl in levels.OrderBy(l => l.XpRequired))
        if (xp >= lvl.XpRequired) current = lvl;
    return current;
}
```

Listing 7.9: Computing a learner's level from experience (excerpt)

Badge awarding is applied when a mission is completed. Each mission has an associated badge, and additional badges recognize cross-mission accomplishments such as completing a whole domain or reaching a level. The award operation is idempotent: because the user-badge relationship carries a unique index, an attempt to award a badge that the learner already holds has no effect, which prevents duplicate awards even if a completion is somehow processed twice. Activity-feed entries and streak updates are applied in the same operation, so that the learner's recorded history remains consistent with their experience total.

7.12 Mission Loading and Rendering

When a learner opens a mission, the mission controller assembles a complete view model from the database. It loads the mission row, its ordered stages, the hints for each stage, and the terminal files that constitute the evidence, and it applies the learner's progress record to determine which stage is current and which stages have been cleared. The assembled model is passed to the Razor view, which renders the static structure of the play screen, and a compact serialization of the stage data is embedded in the page for the client script to interpret. Only the data needed for the current interaction is exposed; in

particular, the expected flag values are retained on the server and are never serialized into the page, in keeping with requirement NFR-9.

This loading logic is the point at which the fully database-driven design is most visible. No mission text, flag, hint, or item of evidence appears anywhere in the compiled application; all of it is read from the database at request time. Adding a new mission is therefore a matter of inserting rows into the mission, stage, hint, and terminal-file tables, after which the new mission appears in the catalog and plays correctly with no change to the code, as was verified during the non-functional evaluation reported in Chapter 8.

7.13 Dashboard and Leaderboard Implementation

The dashboard and leaderboard are both derived views: they compute their content from the stored user and progress data rather than maintaining any separate state. The dashboard controller loads the learner's account, progress records, and earned badges, computes the current level and the experience required for the next level, and assembles the per-mission progress summary. Because every value is derived, the dashboard cannot drift out of step

with the learner's true state.

The leaderboard is derived by ordering the users by experience and computing each learner's rank, as shown in the appendix. To keep this efficient as the user base grows, the ordering is performed by the database rather than in application code, and the users table is indexed to support it, in keeping with the performance requirement NFR-3. The current learner's own row is identified during rendering so that it can be highlighted, realizing requirement FR-33.

7.14 Error Handling and Robustness

The application applies consistent error handling so that failures are surfaced clearly rather than silently, in keeping with requirement NFR-16. The approach has two levels. At the first level, each controller action validates its inputs and the current state before acting, and returns a meaningful, structured response when a request cannot be satisfied; because the asynchronous client requests expect structured responses, these errors arrive in the same form as successes and the client can present a clear message to the learner. At the second level, any exception not handled within an action, for example a transient loss of the database connection, is caught by the framework's error-handling middleware,

which logs the detail server-side and returns a generic failure message, preventing the leakage of internal information such as stack traces or connection strings to the client.

To make this concrete, Table 7.2 lists the principal failure cases the application anticipates, the condition that triggers each, and the response the learner receives. These cases were exercised during the functional testing reported in Chapter 8.

Table 23 Principal failure cases and the application's handling of each

Failure case	Trigger	Response
Unauthenticated request	No learner identity in the session	HTTP 401; redirect to login
Locked mission access	Prerequisite mission not completed	Structured "mission locked" message; no data loaded
Unknown mission or stage	Identifier not found in the database	HTTP 404; "not found" message
Invalid registration input	Missing field or malformed email	Field-level validation message; no account created
Duplicate email at registration	Email already registered	"account already exists" message
Incorrect credentials at login	Password fails to verify against the hash	Generic "invalid email or password" message
Incorrect flag submission	Submitted flag does not match	"incorrect, re-examine the evidence" message; attempt logged
Database unavailable	Connection or query exception	Logged server-side; generic failure message to the client

The two failure cases with security significance, the unauthenticated request and the locked-mission access, are handled by the server-side guard described in Section 7.5.3, so that error handling and access control reinforce one another. This disciplined, two-level handling also contributed to the smooth resolution of the integration defects encountered during the database migration described in Chapter 9.

7.15 Security Considerations

Although CyberQuest is a learning platform whose scenarios are simulated, the platform itself is a real web application and applies standard protective measures in line with the security requirements NFR-5 through NFR-9. Passwords are never stored in plaintext; they are hashed with BCrypt, which incorporates a per-password salt and a tunable work

factor, as discussed in Chapter 3. Authentication state is maintained in server-side sessions. Data access is performed exclusively through the object-relational mapper, which uses parameterized queries and thereby mitigates injection risks. Input from authentication and contact forms is validated before use, and expected flag values are held server-side and never sent to the client before submission. These measures ensure that the platform does not itself become a source of the vulnerabilities it teaches learners to recognize.

7.16 Chapter Summary

This chapter has described the realization of the CyberQuest design on a modern web stack. The platform is built on ASP.NET Core with Entity Framework Core and SQL Server, following the conventional Model-View-Controller structure. The application layer comprises focused controllers coordinated with a single mapping service and a single database context, so that the concerns most subject to change, presentation shape and database structure, are each localized. The core workflows, authentication, mission play with flag submission, and hint provision, were presented with representative code and related back to the sequence and activity models of the design. The user interface was shown to present all elements needed for a stage on one screen and to derive its dashboard, leaderboard, and profile views entirely from stored data. The simulated terminal was shown to provide an authentic investigative interface entirely on the client, without any real execution and without exposing answer keys. Finally, the database schema and its seeding were shown to embody the database-driven principle, and the security measures were shown to apply the protections the platform teaches. Together, these elements constitute a complete, working realization of the design, whose validation is the subject of the next chapter.

Chapter 8

Results and Evaluation

This chapter reports the evaluation of CyberQuest. Because the contribution of this thesis is the design and construction of a working platform rather than a controlled study of learning gains, the evaluation focuses on two complementary questions: whether the system functions correctly across its principal workflows, and whether each mission's intended learning outcome is actually reachable through play. The chapter first describes the evaluation methodology, then the mission catalog that constitutes the platform's content, then reports functional testing, then presents the outcome-reachability walkthroughs, and finally examines the DDoS Defense mission in detail as a representative case.

8.1 Evaluation Methodology

The evaluation follows an artifact-centered methodology appropriate to a design-and-build thesis. Rather than testing a hypothesis about a population, the evaluation asks whether the constructed artifact meets its specification and achieves its purpose. Three complementary methods are used. First, functional testing exercises each workflow against its specified behavior, validating the functional requirements of Chapter 4. Second, structured walkthroughs play each mission from start to finish to confirm that its intended learning outcome is reachable and that success requires the intended skill, validating the pedagogical construction of the content. Third, an inspection-based assessment examines the non-functional qualities, security, maintainability, persistence, and performance, against their requirements.

This methodology has the advantage of directly addressing the objectives and requirements defined earlier in the thesis, and its limitations, principally the absence of a controlled study of learning gains with human participants, are acknowledged explicitly in Chapter 9 and framed as future work. The tests were conducted in a development environment comprising the application and database co-located on a single machine, which is sufficient for validating functional correctness and single-user behavior. It is crucial to state clearly that the assessments carried out in this chapter are solely concerned with outcome-reachability walkthroughs, functional correctness, and system

architecture verification. Since no controlled user-learning research involving human participants was carried out as part of this work, they do not quantify student learning improvements.

8.2 Mission Catalog

The delivered platform contains six missions spanning distinct cybersecurity domains, each decomposed into ordered stages and each targeting a specific learning outcome. The catalog is summarized in Table 8.1, which also serves as the dataset description for this project, since the missions, stages, and their embedded evidence constitute the content on which the platform operates. The catalog as presented to the learner was shown earlier in Figure 7.6.

Table 24 Summary of the mission catalog (the platform's content dataset)

Mission	Domain	Stages	XP	Target learning outcome
Smart City Power Grid Hack	Network / SCADA	3	350	Detect and contain an intrusion in an industrial control network
Smart Home IoT Hack	IoT security	3	400	Identify a weak device and apply network segmentation
Phishing Awareness Challenge	Social engineering	3	300	Recognize spoofed senders, malicious URLs, and pressure tactics
Social Media Password Guessing	Authentication	3	450	Recover a weak credential and recommend MFA and passphrases
Self-Driving Car Hijack	Embedded / automotive	3	600	Detect a forged CAN signal and restore message integrity
DDoS Defense Simulation	Network security	4	550	Triage, classify, mitigate, and document a multi-vector DDoS attack

As Table 8.1 shows, the catalog covers six domains and comprises nineteen stages in total, with experience rewards scaled to mission difficulty. The most advanced mission, the DDoS Defense Simulation, is decomposed into four phases, triage, classification, mitigation, and documentation, mirroring the structure of a real incident-response process.

8.3 Functional Testing

The platform's principal workflows were exercised through systematic functional testing, in which each workflow was executed under representative conditions and its

observed behavior compared against the expected outcome. The results are summarized in Table 8.2. Testing covered authentication, the mission-play loop, the progression and achievement subsystems, the persistence of state across sessions, and the derived dashboard and leaderboard views, thereby validating the functional requirements defined in Chapter 4.

Table 25 Functional test results for the principal workflows

Req.	Workflow tested	Expected behavior	Result
FR-1	Registration with new credentials	Account created, password hashed, session started	Pass
FR-4	Login with valid / invalid credentials	Valid accepted; invalid rejected with message	Pass
FR-10	Open unlocked mission	Mission and stage evidence load correctly	Pass
FR-14	Submit correct flag	Stage cleared; progress advanced	Pass
FR-15	Submit incorrect flag	Rejected; learner returned to investigation	Pass
FR-16	Complete final stage	XP awarded; badge granted; next mission unlocked	Pass
FR-26	Request progressive hint	Next hint revealed; XP deducted; request logged	Pass
FR-9	Locked mission access attempt	Access prevented until prerequisite complete	Pass
NFR-14	Persistence across sessions	Progress and XP retained after logout / login	Pass
FR-30	Dashboard derivation	Level, rank, badges reflect stored state	Pass
FR-32	Leaderboard ranking	Users ordered correctly by experience	Pass
FR-6	Profile update and password change	Changes persisted; new password verifies	Pass
FR-35	Contact message submission	Message persisted for the administrator	Pass

As Table 8.2 records, all of the tested workflows behaved as intended. Defects encountered during development, including several that arose from the migration to a fully database-driven design, were resolved before this testing was carried out; these are discussed in Chapter 9.

8.4 Learning-Outcome Reachability

Functional correctness establishes that the platform works, but not that it teaches. To address the latter, each mission was subjected to a structured walkthrough in which the mission was played from start to finish while verifying that clearing its stages requires the learner to exercise the intended skill and that the target learning outcome is genuinely reachable through investigation rather than guessing. The results are summarized in Table 8.3.

Table 26 Learning-outcome reachability walkthrough for each mission

Mission	Skill exercised to clear stages	Reachable
Power Grid Hack	Reading scan output to isolate a rogue host and its malicious command	Yes
Smart Home IoT	Enumerating devices, spotting default credentials, choosing segmentation	Yes
Phishing Challenge	Inspecting headers and URLs, naming the manipulation tactic	Yes
Password Security	Cracking a weak hash, reasoning about MFA and passphrases	Yes
Autonomous Vehicle	Identifying a forged CAN ID and the integrity mechanism	Yes
DDoS Defense	Triaging traffic, classifying vectors, sequencing mitigations, documenting	Yes

As Table 8.3 shows, in every mission the correct flag can be obtained only by performing the investigative action the mission is designed to teach, and the progressive hints guide a stuck learner toward that action without simply bypassing it. This provides evidence that the platform's content is pedagogically sound in construction, while acknowledging that measuring actual learning gains in human participants remains future work.

8.5 Case Study: The DDoS Defense Mission

The DDoS Defense Simulation is examined in detail here because it is the most elaborate mission and best illustrates how a realistic professional workflow is embedded in the platform. Its four phases, shown in Figure 8.1, correspond to the detection-and-analysis, containment, and post-incident stages of the NIST incident-handling process discussed in Chapter 2.

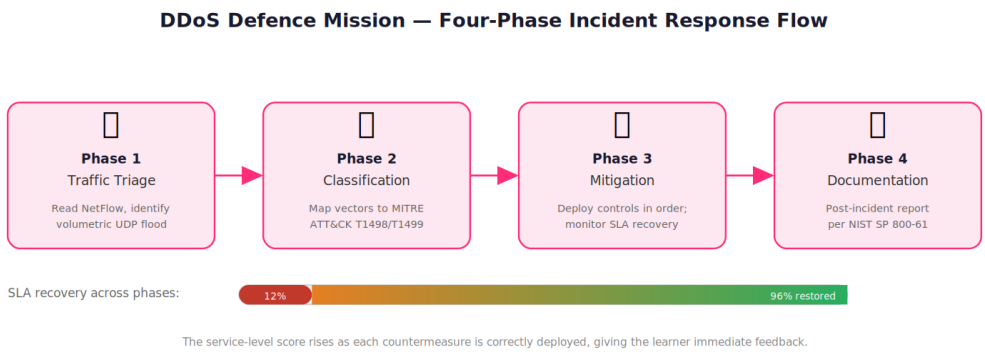


Figure 31 The four-phase incident-response flow of the DDoS Defense mission

As Figure 8.1 illustrates, the learner first triages live traffic telemetry to identify a volumetric attack, then classifies each of the three attack vectors and assigns the correct MITRE ATT&CK identifier, then deploys countermeasures in the technically correct sequence while a service-level score recovers, and finally documents the incident in a structured report. The recovering service-level score gives the learner immediate, quantitative feedback on the effect of each correct decision, which reinforces the connection between action and consequence that underpins experiential learning. The mission thus exercises, in a single coherent scenario, the reading of evidence, the application of a professional taxonomy, ordered decision-making under a recovery objective, and formal documentation, a combination that isolated exercises rarely achieve.

The design of this mission repays closer examination because it illustrates several of the platform's principles at once. The four phases embody the principle that a mission should mirror a real professional workflow: they follow the structure of the incident handling process discussed in Chapter 2, so that a learner who completes the mission has rehearsed, in miniature, the shape of a genuine response. The requirement to assign the correct MITRE identifiers embodies the principle that content should use the vocabulary of practice, so that the learner's terminology transfers to professional settings. The ordering constraint on the mitigations, in which countermeasures must be deployed in a technically sensible sequence, embodies the principle that a stage should require genuine reasoning rather than recognition: a learner cannot simply select all countermeasures but must determine which to apply first. And the documentation phase embodies the often-neglected principle that communication is part of security work, requiring the learner to articulate what happened in a structured form. The mission therefore demonstrates that

the general mission model is expressive enough to capture a sophisticated, multi-faceted professional task, not merely a simple puzzle.

The service-level indicator deserves particular note as a design device. By expressing the effect of the learner's decisions as a single, continuously visible number that rises toward a target as correct countermeasures are applied, it converts an abstract notion of "defending the service" into concrete, moment-to-moment feedback. This is a direct application of the flow principle from Chapter 2: the indicator keeps the learner oriented toward the goal and provides the immediate feedback that sustains engagement, while the ordering constraint ensures that the challenge remains genuine.

8.6 Detailed Test Cases

To complement the summary of Table 8.2, this section documents representative test cases in the standard form of precondition, steps, expected result, and observed result. Table 8.4 through Table 8.6 present three such cases covering registration, login, and flag submission; the remaining workflows were tested analogously.

Table 27 Test case TC-01: successful registration

Field	Detail
Requirement	FR-1, FR-2, FR-3
Precondition	No account exists for the chosen email.
Steps	Open the registration panel; enter a valid name, unused email, and password; submit.
Expected result	Account is created, the password is stored as a BCrypt hash, a session is started, and the learner is directed to the dashboard.
Observed result	As expected, a subsequent database inspection confirmed the stored value was a hash, not plaintext.
Status	Pass
Requirement	FR-4
Precondition	An account exists for the chosen email.
Steps	Open the login panel; enter the correct email with an incorrect password; submit.
Expected result	Authentication is rejected with a clear message, and no session is started.
Observed result	As expected; the generic rejection message did not disclose whether the email existed.
Status	Pass

Requirement	FR-14, FR-16, FR-17, FR-20
Precondition	The learner is on the final stage of an in-progress mission.
Steps	Enter the correct flag for the final stage; submit.
Expected result	The stage is cleared, the mission is marked complete, experience is awarded, the mission badge is granted, and the next mission is unlocked.
Observed result	As expected, the dashboard and leaderboard reflected the new experience total immediately.
Status	Pass

As these cases illustrate, the tests were specified with explicit preconditions and expected results so that the observed behavior could be judged unambiguously. The consistency between expected and observed results across all workflows, summarized in Table 8.2, gives confidence in the functional correctness of the platform.

8.7 Evaluation of Non-Functional Qualities

Beyond functional behavior, the non-functional requirements were assessed by inspection and observation during testing. The security measures of NFR-5 through NFR-9 were verified by confirming that stored passwords are BCrypt hashes, that all data access passes through the parameterized object-relational mapper, and that flag values are not present in the client-delivered page before submission. The maintainability requirements NFR-17 through NFR-20 were verified by adding a trial mission entirely through database inserts and confirming that it appeared and played correctly without any code change or redeployment. The persistence and consistency requirements NFR-14 and NFR-15 were verified through the cross-session and completion tests reported in Table 8.2. Interactive responses were observed to be prompt under single-user conditions, satisfying the performance expectations of NFR-1 and NFR-2 for the validation environment.

8.8 Demonstration of Extensibility

A specific claim of this thesis is that the database-driven design makes the platform extensible without code changes. To substantiate this claim concretely, a trial mission was added to the platform entirely through database inserts during the evaluation. New rows were inserted into the mission table, the stage table, the hint table, and the terminal-file table, describing a short additional scenario with its stages, hints, and evidence. No application code was modified and the application was not recompiled or redeployed.

On reloading the catalog, the trial mission appeared alongside the existing six, with its correct title, domain, difficulty, and reward. It could be opened and played, its stages advanced on correct flag submission, its hints were revealed on request with the appropriate experience deduction, and its completion awarded experience and unlocked the next mission in sequence. Every feature of the platform operated on the new mission exactly as it does on the original six. This demonstration provides direct evidence that requirement NFR-17 is satisfied and that the maintainability objective of the thesis is achieved in practice, not merely in principle. The trial mission was subsequently removed, again through database operations alone, leaving the delivered catalog at its intended six missions.

8.9 Discussion of Results

Taken together, the functional testing, the reachability walkthroughs, and the non-functional assessment indicate that CyberQuest meets the objectives set out in Chapter 1. The platform delivers a complete, database-driven, gamified learning environment in which six narrative missions are playable end to end, progression and achievements function correctly, learner state persists reliably, each mission's intended skill is genuinely required to succeed, and the quality attributes required of the system are satisfied. These results substantiate the central claim of the thesis: that narrative-driven, flag-based practice, an explicit progression economy, lightweight adaptive support, and a maintainable content architecture can be integrated into a single coherent platform.

Chapter 9

Discussion

This chapter reflects on the outcomes of the project. It interprets the results in relation to the original objectives and research questions, examines the strengths of the design, discusses the challenges encountered during development and how they were resolved, considers the threats to the validity of the evaluation, and states the limitations of the work honestly so that the conclusions drawn are appropriately bounded.

9.1 Interpretation of Findings

The evaluation shows that the platform achieves what it set out to achieve. Each of the six research objectives stated in Chapter 1 is reflected in a working feature of the delivered system: the narrative mission model, the staged flag-capture loop, the progression economy, the progressive-hint and adaptive-difficulty support, the fully database-driven architecture, and the functional validation are all present and operational. Revisiting the research questions of Section 1.5, RQ1 is answered by the mission and stage model whose reachability was confirmed in Chapter 8; RQ2 by the experience, level, and badge economy tied to genuine learning actions; RQ3 by the progressive hints and rule-based adaptivity; RQ4 by the database-driven content architecture; and RQ5 by the functional testing and walkthroughs.

The most significant finding is not any single feature but their integration: the same normalized database simultaneously drives mission play, progression, achievements, the dashboard, and the leaderboard, which demonstrates that a data-centric design can unify features that are often implemented in isolation. This integration is what distinguishes CyberQuest from the categories of existing systems compared in Chapter 2.

9.2 Strengths of the Approach

Three strengths of the design are worth emphasizing. First, the database-driven content architecture makes the platform genuinely extensible: a new mission, with its stages, hints, and evidence, can be added purely as data, without recompiling or redeploying the application. This was demonstrated directly during the non-functional evaluation and addresses the maintainability problem identified in the literature.

Second, the narrative framing of each mission provides context that isolated challenges lack, which is intended to improve engagement and the transfer of skills to realistic situations. The DDoS case study of Section 8.5 shows how a single scenario can weave together evidence reading, taxonomy application, ordered decision-making, and documentation, a combination that fragmented exercises rarely achieve.

Third, the layered architecture and the confinement of data access to a single context and mapping service make the system understandable and maintainable, which matters for a platform intended to grow over time. The clear separation of concerns also made the system easier to test, since each workflow could be exercised through a well-defined controller action.

9.3 Challenges Encountered

Several engineering challenges arose during development, and the way they were handled shaped the final design.

The most substantial was the transition from an early version in which mission content was embedded in code to the final version in which all content resides in the database. This migration required introducing a full set of content entities, moving every mission's text, flags, hints, and terminal evidence into tables, and rewriting the controllers to read from those tables through the mapping service. The change eliminated a class of maintainability problems but introduced, temporarily, a number of integration defects, such as mismatches between the object model and the schema and an omitted primary-key configuration on the level entity, which were identified and resolved during testing. The primary-key configuration in particular, shown in Listing 7.2, had to be declared explicitly because the level entity uses a non-default key.

A second challenge concerned authentication: an early client-server data-format mismatch prevented credentials from binding correctly on the server, because the client submitted data in one encoding while the server expected another. This was resolved by aligning the request encoding with the server's model binding, after which the login workflow behaved as specified.

A third challenge was tuning the progression economy so that levels felt meaningful. Initial thresholds allowed a learner to reach high levels simply by completing missions once; the thresholds were subsequently raised, as reflected in Figure 5.8, so that bonus

experience from avoiding hints and maintaining streaks became necessary to advance, aligning the reward structure with mastery rather than mere completion.

These experiences reinforced the value of the iterative approach, since each problem surfaced against a runnable system and could be isolated quickly.

9.4 Threats to Validity

Several threats to the validity of the evaluation should be acknowledged. With respect to construct validity, the reachability walkthroughs were conducted by the developer, who is familiar with the intended solution path; an independent evaluator might encounter difficulties that the developer does not anticipate. With respect to external validity, the functional testing was performed in a single-user development environment, so the results do not speak to behavior under concurrent multi-user load. With respect to internal validity, the absence of a controlled study means that any claim about learning effectiveness rests on the construction of the content rather than on measured outcomes. These threats motivate the future work described in the next chapter and are the reason the claims of this thesis are framed in terms of functional correctness and outcome reachability rather than measured learning gains. Additionally, whereas NFR-22 describes architectural preparedness for scalability, this single-developer MS thesis did not include actual multi-user load testing. In a similar vein, the lack of an empirical study prevents quantitative learning benefits from being quantified, leaving them as important avenues for further research.

9.5 Lessons Learned

Several general lessons emerged from the project that may be of value to similar efforts. The first concerns the timing of the move to a data-driven design. Embedding content in code was expedient in the earliest iterations, when the shape of a mission was still uncertain, but it became a liability as the catalog grew. In retrospect, the boundary between code and content could have been drawn earlier, once the mission model had stabilized, which would have avoided some of the migration effort. The lesson is that content-heavy applications benefit from treating content as data as soon as its structure is understood, but not before.

A second lesson concerns the value of keeping the system runnable at every step. Because each increment left a working system, defects could be attributed to a specific, recent change and isolated quickly. This discipline was especially valuable during the

database migration, when a large amount of code changed; the ability to test each migrated mission against a running system kept the process manageable.

A third lesson concerns the alignment of reward with mastery. The initial progression economy inadvertently rewarded mere completion, which risked encouraging shallow engagement. Retuning the thresholds so that meaningful advancement required avoiding hints and sustaining activity aligned the incentives with the learning goal. The lesson is that gamification mechanics must be tuned deliberately against the intended behavior, not merely added.

9.6 Detailed Comparison with Related Systems

The high-level comparison of Chapter 2 can be sharpened in light of the completed system. Compared with open capture-the-flag frameworks, CyberQuest sacrifices some flexibility, its content model is fixed to the mission-stage-hint structure, in exchange for a coherent narrative, an integrated progression economy, and a lower barrier to content authoring, since new content is added as database rows rather than as configuration files and challenge binaries. Compared with guided commercial learning sites, CyberQuest is far more modest in scale and polish but is fully self-contained, open to institutional extension, and free of any dependence on external infrastructure or subscription. Compared with cyber ranges, CyberQuest deliberately trades fidelity for accessibility: it cannot develop the muscle memory of operating real tools, but it can be deployed on a single machine and run in any browser, which suits the introductory and intermediate learning it targets.

The distinctive position of CyberQuest, then, is as an integrated, low-infrastructure platform that unifies narrative, progression, scaffolding, and maintainable content. No single category of existing system occupies exactly this position, which is the sense in which the thesis fills a gap. This is a claim about design integration rather than about any individual feature, each of which has precedent in the literature.

9.7 Generalizability of the Approach

Although CyberQuest is a cybersecurity-education platform, the design principles it embodies generalize to other domains in which applied, procedural skills must be developed through practice. The core pattern, decomposing a domain into narrative scenarios, representing each as a sequence of stages that require a genuine action to clear,

scaffolding each stage with progressive support, motivating progress with an economy of points and achievements, and storing all of this as data rather than code, is not specific to security. The same pattern could structure a platform for teaching data analysis, network administration, digital forensics, or any field where the goal is competence at a procedure rather than recall of facts.

What would change from domain to domain is the content and the nature of the evidence a learner investigates, not the architecture. Because the architecture treats content as data, adapting the platform to a new domain would be primarily a content-authoring effort rather than a re-engineering effort, which is precisely the property that the database-driven design was chosen to provide. In this sense the contribution of the thesis is not only a cybersecurity-education platform but a reusable design for skill-development platforms more broadly, of which CyberQuest is one instantiation. This generality strengthens the case that the integration demonstrated here is a genuine contribution rather than a one-off construction.

9.8 Ethical Considerations

A platform that teaches security techniques raises ethical questions that merit explicit consideration. The foremost concern is dual use: knowledge of how attacks work can be used to defend or to harm. CyberQuest addresses this concern through the framing and scope of its content. Every mission is cast from the defender's perspective, the learner responds to, contains, or documents an attack rather than perpetrating one, and the scenarios are simulated entirely within the platform, so that no capability to attack a real system is developed or exercised. The skills the platform builds, reading evidence, recognizing attack patterns, reasoning about defenses, are precisely those a defender needs and are not operational attack techniques.

A second consideration is data and privacy. The platform records learner telemetry, submissions, hint requests, and sessions, for the legitimate purpose of supporting the learner's own progress and enabling future analytics. This telemetry concerns the learner's interaction with a simulated environment, not any sensitive personal data beyond the account information required to operate the platform, and the account credential is stored only as a salted hash. In any real deployment, the collection and use of this telemetry

would be governed by an appropriate privacy policy and by the institution's data-protection obligations, and the append-only design of the telemetry tables supports transparency about what is recorded.

A third consideration is fairness in the gamified design. Competitive elements such as a leaderboard can, if poorly designed, discourage learners who are not near the top. CyberQuest mitigates this by making the core progression individual, experience, levels, and badges reflect a learner's own accomplishments rather than their standing relative to others, and by treating the leaderboard as a secondary, optional motivator rather than the central mechanic. This keeps the emphasis on personal mastery, which is both pedagogically sound and more inclusive.

9.9 Limitations

The work has limitations that bound its conclusions. First, the evaluation establishes functional correctness and outcome reachability but does not measure learning gains in human participants; a controlled study would be required to quantify the platform's educational effectiveness. Second, the adaptive-difficulty component is rule-based and lightweight rather than a trained model, so its personalization is coarse. Third, the security scenarios are simulated within the application; while this is appropriate and safe for a learning context, it means the platform does not develop the tool-operation skills that a live environment would. Fourth, the content catalog, though covering six domains, is modest in size, and the platform has not yet been exercised at scale with a large concurrent user base. Each of these limitations corresponds directly to an opportunity for future work, as outlined in the next chapter.

Chapter 10

Conclusion and Future Work

This final chapter summarizes the thesis, restates its contributions, and sets out concrete directions in which the work can be extended.

10.1 Conclusion

This thesis set out to address a persistent gap in cybersecurity education: the distance between conceptual knowledge and the applied, decision-making competence that practitioners need. It argued that existing hands-on learning environments tend to emphasize one of narrative, engagement, adaptivity, or maintainability at the expense of the others, and that an integrated approach would be more effective. In response, it presented CyberQuest, a gamified, story-driven web platform in which learners develop practical security skills by investigating simulated incidents and capturing flags across six narrative missions.

The platform was specified through a detailed requirements analysis, designed using standard software-engineering models, and implemented as an ASP.NET Core application backed by a Microsoft SQL Server database in which all content, progression rules, and learner telemetry are stored rather than hard-coded. It provides a staged mission-play loop, an experience-and-badge progression economy, progressive hints, a lightweight adaptive-difficulty recommendation, a personal dashboard, and a global leaderboard. Functional testing confirmed that every principal workflow behaves correctly and satisfies the requirements defined for it, and structured walkthroughs confirmed that each mission's intended learning outcome is genuinely reachable through play. Together, these results demonstrate that narrative-driven practice, a motivating progression economy, adaptive support, and a maintainable, database-driven architecture can be integrated into a single coherent and extensible platform.

The principal contribution of this work is therefore not a new learning mechanic but the demonstration of a unified, data-centric design that brings these elements together in a low-infrastructure system suitable for educational use, together with a working implementation that can serve as a foundation for further research and development.

10.2 Revisiting the Objectives

Each objective stated in Chapter 1 has been met. A gamified, story-driven platform was designed and implemented; a staged mission model with flag capture was developed and shown to be reachable; a progression system of experience, levels, and badges was implemented and tested; progressive hints and a rule-based adaptive component were provided; a fully database-driven architecture was designed and its extensibility demonstrated; and the platform was validated through functional testing and walkthroughs. The correspondence between objectives, realizing features, and validating evidence was traced in Chapter 4 and confirmed in Chapter 8.

10.3 Summary of Answers to Research Questions

Table 10.1 summarizes how each research question posed in Chapter 1 has been answered by the work reported in this thesis, with a pointer to the chapter that provides the evidence.

Table 28 Research questions and the answers established by this thesis

RQ	Answer	Evidence
RQ1	Realistic scenarios can be represented as narrative, staged missions whose flags require genuine investigation, as shown by the six-mission catalog and its reachability.	Ch. 6, Ch. 8
RQ2	An experience, level, and badge economy tied to mastery sustains motivation without displacing the learning goal, provided rewards are aligned with accomplishment.	Ch. 5, Ch. 9
RQ3	Support can be adapted through progressive, costed hints and a lightweight rule-based recommendation, keeping personalization maintainable.	Ch. 5
RQ4	A fully database-driven architecture allows all content to reside in data, enabling extension without code change, as demonstrated.	Ch. 5, Ch. 8
RQ5	The platform functions correctly and each mission's outcome is reachable, as established by functional testing and walkthroughs.	Ch. 8

10.4 Future Work

Several directions would extend the value of the platform. The most important is an empirical study with human participants to measure learning gains, comparing learners who use CyberQuest against those who use conventional material, so that the platform's educational effectiveness can be quantified rather than argued from construction. A second direction is to replace the rule-based adaptive component with a data-driven

model that estimates each learner's knowledge state from the telemetry the platform already records and personalizes mission and hint selection accordingly.

- Richer content. Expanding the mission catalog to cover additional domains, and introducing branching scenarios in which a learner's choices change the course of a mission, would deepen both coverage and replay value.
- Analytics for instructors. The submission, hint, and session telemetry already captured could drive an instructor-facing analytics dashboard that highlights where learners struggle, enabling targeted intervention.
- Collaborative and competitive modes. Team-based missions and time-boxed competitive events would add social learning dynamics that the current single-player design does not exploit.
- Integration with real tooling. For advanced learners, selected missions could be connected to isolated, containerized environments so that skills practiced in the simulated terminal transfer to genuine tools.
- Deployment and scale. Hosting the platform for a real cohort and studying its behavior under concurrent load would validate its operational characteristics and surface further engineering requirements.
- Accessibility and internationalization. Broadening the interface to meet accessibility standards and to support multiple languages would widen the platform's reach.

These directions can be organized into a roadmap by the horizon over which they are likely to be pursued, as shown in Figure 10.1. Near-term work centers on evaluation and instructor-facing analytics, which build directly on the telemetry the platform already captures; mid-term work introduces data-driven adaptivity and richer branching content; and long-term work integrates live tooling, collaborative modes, and deployment at scale.

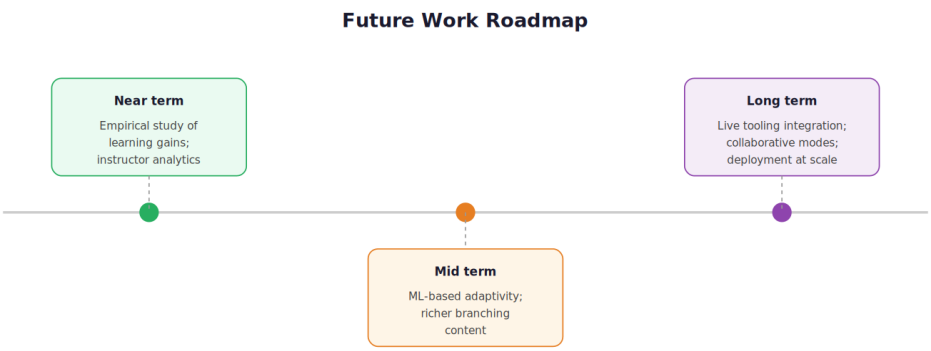


Figure 32 A roadmap organizing the future work by horizon

As Figure 10.1 suggests, the near-term items are the natural next step because they require no new architecture, only the analysis of data the platform already records. Pursuing these directions would transform CyberQuest from a validated prototype into a continually improving educational platform, while preserving the data-centric design principle that makes such growth feasible.

10.5 Concluding Remarks

The work presented in this thesis shows that careful integration of well-understood ideas, experiential learning, gamification, scaffolding, and a maintainable architecture, can yield a learning environment that is greater than the sum of its parts. None of the individual ingredients is novel in isolation; the contribution lies in bringing them together coherently, grounding each in an established principle, and realizing the result in a working, extensible system. In doing so, the thesis provides both a design and a concrete artifact on which future cybersecurity-education research can build.

More broadly, the project illustrates a design philosophy that may apply beyond cybersecurity education: that treating content as first-class data, rather than embedding it in code, is what makes an educational platform sustainable over time. A platform built this way can grow with its subject and its audience, because extending it is a matter of authoring content rather than engineering software. As the demands on cybersecurity education continue to grow, platforms that can evolve quickly and cheaply will be at a premium, and the data-centric design demonstrated here offers one path toward that goal. It is the author's hope that CyberQuest, and the design principles it embodies, can contribute in a small way to closing the persistent gap between knowing about security and being able to do it.

The broader lesson of this thesis is that the difficulty in building effective educational technology often lies not in any single component but in their integration. Experiential learning, gamification, scaffolding, and maintainable architecture are each well understood in isolation; what has been less common is a system that brings them together coherently, so that each reinforces the others and none is compromised for the sake of another. CyberQuest is offered as one demonstration that such integration is achievable within the scope of a focused project, and that the discipline of treating content as data is what keeps the resulting system open to the continual growth that any living educational platform requires. It is the hope of this work that both the design and the reflections accompanying it will be of use to those who take up the task of preparing the next generation of cybersecurity practitioners.

Appendix A

Supporting Code and Schema Listings

This appendix provides fuller extracts of the artifacts described in the main text, for readers who wish to see the concrete realization of the design. The listings are representative rather than exhaustive; they illustrate the structure and conventions of the codebase and the database.

A.1 Database Context

Listing A.1 shows the shape of the database context class, including the collections that expose each table and the model configuration that establishes keys and unique constraints.

```
public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> o) : base(o) {}

    // Content tables
    public DbSet<DbMission> Missions => Set<DbMission>();
    public DbSet<DbSubChallenge> SubChallenges => Set<DbSubChallenge>();
    public DbSet<DbHint> Hints => Set<DbHint>();
    public DbSet<DbTerminalFile> TerminalFiles => Set<DbTerminalFile>();
    public DbSet<DbLevel> Levels => Set<DbLevel>();
    public DbSet<DbBadge> Badges => Set<DbBadge>();

    // User tables
    public DbSet<UserAccount> Users => Set<UserAccount>();
    public DbSet<UserProgress> UserProgress => Set<UserProgress>();
    public DbSet<UserBadge> UserBadges => Set<UserBadge>();
    public DbSet<UserActivity> UserActivities => Set<UserActivity>();

    // Telemetry tables
    public DbSet<FlagSubmission> FlagSubmissions => Set<FlagSubmission>();
    public DbSet<HintRequest> HintRequests => Set<HintRequest>();
    public DbSet<SessionLog> SessionLogs => Set<SessionLog>();
    public DbSet<ContactMessage> ContactMessages => Set<ContactMessage>();

    protected override void OnModelCreating(ModelBuilder b)
    {
        b.Entity<DbLevel>().HasKey(l => l.Number);
        b.Entity<UserProgress>()
            .HasIndex(p => new { p.UserId, p.MissionId }).IsUnique();
```

```
            b.Entity<UserBadge>()
                .HasIndex(u => new { u.UserId, u.BadgeId }).IsUnique();
            b.Entity<DbSubChallenge>()
                                .HasIndex(s => new { s.MissionId, s.StageIndex }).IsUnique();
            b.Entity<DbHint>()

                .HasIndex(h => new { h.SubChallengeId, h.HintOrder }).IsUnique();

    }

}
```

Listing A.1: The AppDbContext class (excerpt)

A.2 Registration Handler

Listing A.2 shows the registration handler, which validates the submitted data, ensures the email is unique, hashes the password, and creates the account.

```
[HttpPost]
public async Task<IActionResult> Register(RegisterViewModel model)
{
    if (!ModelState.IsValid)
        return Json(new { success = false,
            message = "Please complete all fields correctly." });

    var exists = await _db.Users.AnyAsync(
        u => u.Email.ToLower() == model.Email.ToLower());
    if (exists)
        return Json(new { success = false,
            message = "An account with that email already exists." });

    var user = new UserAccount {
        FullName = model.FullName.Trim(),
        Email = model.Email.Trim().ToLower(),
        PasswordHash = BCrypt.Net.BCrypt.HashPassword(model.Password),
        University = model.University,
        TotalXp = 0, CreatedAt = DateTime.UtcNow
    };
    _db.Users.Add(user);
    await _db.SaveChangesAsync();

    SetSession(user);
```

```
25          return Json(new { success = true, redirect = "/Dashboard" });
26      }
```

Listing A.2: Registration handler (excerpt)

A.3 Flag Submission Handler

Listing A.3 shows the flag-submission handler in fuller form, including the loading of progress, validation of the flag, logging of the attempt, and the application of progression side effects when the final stage is cleared.

```
1   [HttpPost]
2   public async Task<IActionResult> SubmitFlag(
3           int missionId, int stage, string flag)
4   {
5           var userId = CurrentUserId();
```

```
6           if (userId == null) return Unauthorized();
7
8           var sub = await _db.SubChallenges.FirstOrDefaultAsync(
9                   s => s.MissionId == missionId && s.StageIndex == stage);
10          if (sub == null) return NotFound();
11
12          var submitted = (flag ?? "").Trim();
13          var correct = string.Equals(submitted, sub.Flag,
14                  StringComparison.OrdinalIgnoreCase);
15
16          _db.FlagSubmissions.Add(new FlagSubmission {
17                  UserId = userId.Value, MissionId = missionId,
18                  StageIndex = stage, SubmittedFlag = submitted,
19                  Correct = correct, SubmittedAt = DateTime.UtcNow });
20
21          if (!correct) {
```

```
            await _db.SaveChangesAsync();
            return Json(new { success = false,
                message = "Incorrect. Re-examine the evidence." });
        }

        var totalStages = await _db.SubChallenges
            .CountAsync(s => s.MissionId == missionId);
        var prog = await GetOrCreateProgress(userId.Value, missionId);
        prog.HighestStageReached = Math.Max(prog.HighestStageReached, stage);

        bool isLast = stage == totalStages;
        if (isLast) {
            var mission = await _db.Missions.FindAsync(missionId);
            var user = await _db.Users.FindAsync(userId.Value);
            prog.Status = "completed";
            prog.CompletedAt = DateTime.UtcNow;
            user.TotalXp += mission.XpReward;
            user.MissionsCompleted += 1;
            await AwardMissionBadge(userId.Value, missionId, user);
            await UnlockNextMission(userId.Value, missionId);
        }
        await _db.SaveChangesAsync();
        return Json(new { success = true, last = isLast });
    }
```

Listing A.3: Flag submission handler (excerpt)

A.4 Schema Creation Script

Listing A.4 shows a representative portion of the setup script that creates the core content and user tables and their relationships.

```
CREATE TABLE Missions (
    Id              INT IDENTITY(1,1)   PRIMARY KEY,
    Title           NVARCHAR(120) NOT   NULL,
    Domain          NVARCHAR(60) NOT    NULL,
    Difficulty      NVARCHAR(20) NOT    NULL,
    XpReward        INT NOT             NULL,
    SortOrder       INT NOT             NULL,
UnlockAfterMissionId INT NULL
);
GO
CREATE TABLE SubChallenges (
    Id              INT IDENTITY(1,1) PRIMARY KEY,
    MissionId       INT NOT NULL FOREIGN KEY REFERENCES Missions(Id),
    StageIndex      INT NOT NULL,
    Title           NVARCHAR(120) NOT NULL,
    Brief           NVARCHAR(MAX) NOT NULL,
    Flag            NVARCHAR(120) NOT NULL,
    CONSTRAINT  UQ_Sub UNIQUE (MissionId, StageIndex)
);
GO
CREATE TABLE Users (
IdINT IDENTITY(1,1)                     PRIMARY KEY,
FullNameNVARCHAR(120) NOT               NULL,
EmailNVARCHAR(160) NOT                  NULL UNIQUE,
        PasswordHash NVARCHAR(200) NOT  NULL,
TotalXpINT NOT NULL DEFAULT 0,
    MissionsCompleted INT NOT NULL DEFAULT 0,
CreatedAtDATETIME2 NOT NULL DEFAULT GETUTCDATE()
);
GO
CREATE TABLE UserProgress (
IdINT IDENTITY(1,1) PRIMARY KEY,
UserIdINT NOT NULL FOREIGN KEY REFERENCES Users(Id),
MissionIdINT NOT NULL FOREIGN KEY REFERENCES Missions(Id),
StatusNVARCHAR(20) NOT NULL DEFAULT 'available',
```

```
        HighestStageReached INT NOT NULL DEFAULT 0,
CompletedAtDATETIME2 NULL,
        CONSTRAINT UQ_Prog UNIQUE (UserId, MissionId)
);
GO
```

Listing A.4: Schema creation script (excerpt)

A.5 Leaderboard Query

Listing A.5 shows how the leaderboard ranking is derived directly from the users table, ordering by experience and computing each learner's rank.

```
var ranked = await _db.Users
        .OrderByDescending(u => u.TotalXp)
        .ThenBy(u => u.CreatedAt)
        .Select(u => new LeaderboardRow {
                Name = u.FullName,
                TotalXp = u.TotalXp,
                MissionsCompleted = u.MissionsCompleted
        })
        .ToListAsync();

for (int i = 0; i < ranked.Count; i++)
        ranked[i].Rank = i + 1;
```

Listing A.5: Deriving the leaderboard ranking (excerpt)

Appendix B

Mission Content Summary

This appendix provides a concise narrative summary of each mission's scenario and stages, complementing the tabular catalog of Table 8.1.

B.1 Smart City Power Grid Hack

The learner is cast as an incident responder for a municipal utility whose supervisory control network shows anomalous activity. Across three stages the learner inspects a network scan to locate a rogue host, reads a captured control command to determine the attacker's intent, and identifies the containment action that isolates the affected segment. The mission develops the ability to read operational-technology evidence and reason about containment.

B.2 Smart Home IoT Hack

The learner investigates a compromised smart-home network. The stages require enumerating connected devices, recognizing a device left on default credentials, and selecting an appropriate network-segmentation response. The mission develops awareness of the weak-link nature of consumer IoT and the value of segmentation.

B.3 Phishing Awareness Challenge

The learner triages a suspicious message. The stages require inspecting the sender and message headers, examining an embedded link for signs of spoofing, and naming the psychological pressure tactic employed. The mission develops the ability to recognize social-engineering cues.

B.4 Social Media Password Guessing

The learner examines an account-compromise scenario. The stages require reasoning about why a weak credential fell quickly, recovering a weak hashed password, and articulating

the protective value of multi-factor authentication and passphrases. The mission develops an understanding of authentication weakness and defense.

B.5 Self-Driving Car Hijack

The learner analyzes anomalous behavior in an autonomous vehicle. The stages require inspecting bus traffic, identifying a forged controller-area-network message by its identifier, and selecting the integrity mechanism that would detect the forgery. The mission introduces embedded and automotive security concepts.

B.6 DDoS Defense Simulation

The learner defends a service under a multi-vector distributed denial-of-service attack, as detailed in Section 8.5. The four stages, triage, classification, mitigation, and documentation, mirror a professional incident-response process and culminate in a structured post-incident report.

Appendix C

Glossary of Key Terms

This glossary defines the principal domain and technical terms used in the thesis, for readers who wish to consult a concise reference.

Adaptive difficulty

: The adjustment of challenge to the individual learner; in this platform, a lightweight rule-based recommendation derived from performance signals.

Badge

: An achievement token awarded when a defined condition, such as completing a mission, is met.

Capture the flag (CTF)

: A hands-on security exercise in which a task is completed by recovering a hidden token that proves success.

Experience points (XP)

: The numeric reward accumulated by completing missions, which drives level progression.

Flag

: The token a learner submits to clear a stage, obtained by performing the stage's investigative task.

Gamification

: The use of game design elements, such as points and badges, in non-game contexts to increase engagement.

Level

: A named rank reached when cumulative experience crosses a defined threshold.

Mission

A narrative learning unit targeting a single learning outcome, decomposed into ordered stages.

Chapter C. Glossary of Key Terms

Object-relational mapping (ORM)

A technique that bridges object-oriented programs and relational databases automatically.

Progressive hint

A hint from an ordered sequence that becomes more explicit as later hints are revealed, each at an experience cost.

Scaffolding

Temporary support that helps a learner accomplish a task they could not yet manage independently, gradually withdrawn as competence grows.

Stage

An ordered step within a mission, cleared by submitting the correct flag; referred to internally as a sub-challenge.

Telemetry

The record of learner actions, submissions, hint requests, and sessions, captured for later analysis.

Appendix D

Glossary of Terms

This glossary defines the principal technical terms used in the thesis, for readers who may be unfamiliar with some of them.

Table 29 Glossary of Terms

Adaptive difficulty	The adjustment of a task's challenge to the individual learner, here implemented as a lightweight rule-based recommendation.
BCrypt	An adaptive, salted, one-way password-hashing function whose cost can be tuned upward as hardware improves.
Capture the flag (CTF)	A style of security exercise in which completing a task yields a hidden token that proves completion.
Controller-area network (CAN)	An internal vehicle network over which components communicate; the subject of the autonomous-vehicle mission.
Distributed denial of service (DDoS)	An attack that overwhelms a service with traffic from many sources; the subject of the most advanced mission.
Entity Framework Core	The object-relational mapper used to translate between the application's objects and the relational database.
Experience points (XP)	The primary progression currency, awarded for completing missions and deducted for revealing hints.

Flag	A token a learner submits to prove a stage has been solved; validated server-side against an expected value.
Gamification	The use of game-design elements such as points and badges in a non-game context to increase engagement.
MITRE ATT&CK	A knowledge base of adversary tactics and techniques, each with an identifier, used in the classification stage of the DDoS mission.
Model-View-Controller (MVC)	An architectural pattern separating data, presentation, and control logic, followed by the application.
Multi-factor authentication (MFA)	An authentication method requiring more than one form of evidence; discussed in the password mission.
Object-relational mapping (ORM)	The automated translation between object-oriented programs and relational databases.
Operational technology (OT)	The hardware and software that controls physical processes, such as the SCADA systems in the power-grid mission.
Progressive hints	An ordered sequence of increasingly explicit hints that scaffold a learner while preserving challenge.
SCADA	Supervisory control and data acquisition; the class of industrial control system featured in the first mission.

Scaffolding	Temporary instructional support that is gradually withdrawn as competence grows.
Session	Server-side state that maintains a learner's authenticated identity across stateless HTTP requests.
Sub-challenge	The internal term for a mission stage, cleared by submitting a correct flag.
Telemetry	The append-only record of learner actions, submissions, hints, requests and sessions, captured for future analytics.

Appendix E

Complete Requirements Validation Matrix

This appendix provides a complete matrix mapping every functional and non-functional requirement to its validation status, complementing the traceability summary of Chapter 4 and the test results of Chapter 8. Every requirement defined in the thesis is listed with the method by which it was validated and its status.

Table 30 Complete requirements validation matrix

Requirement	Validation method	Status
FR-1	Functional test (registration)	Validated
FR-2	Functional test (duplicate email)	Validated
FR-3	Inspection of stored hash	Validated
FR-4	Functional test (login)	Validated
FR-5	Functional test (logout)	Validated
FR-6	Functional test (profile update)	Validated
FR-7	Functional test (catalog)	Validated
FR-8	Functional test (mission detail)	Validated
FR-9	Functional test (locked mission)	Validated
FR-10	Functional test (start/resume)	Validated
FR-11	Walkthrough (staged reveal)	Validated
FR-12	Walkthrough (terminal commands)	Validated
FR-13	Walkthrough (evidence access)	Validated
FR-14	Functional test (flag submission)	Validated
FR-15	Functional test (correct/incorrect)	Validated
FR-16	Functional test (completion)	Validated
FR-17	Functional test (XP award)	Validated
FR-18	Inspection (cumulative XP)	Validated
FR-19	Functional test (level mapping)	Validated
FR-20	Functional test (badge award)	Validated
FR-21	Functional test (mission badge)	Validated
FR-22	Inspection (cosmetic unlock)	Validated
FR-23	Inspection (activity feed)	Validated

FR-24	Inspection (streak)	Validated
FR-25	Walkthrough (hint ordering)	Validated
FR-26	Functional test (hint reveal)	Validated
FR-27	Functional test (XP deduction)	Validated
FR-28	Inspection (hint telemetry)	Validated
FR-29	Inspection (adaptive recommendation)	Validated
FR-30	Functional test (dashboard)	Validated
FR-31	Functional test (per-mission progress)	Validated
FR-32	Functional test (leaderboard)	Validated
FR-33	Functional test (own-row highlight)	Validated
FR-34	Functional test (profile)	Validated
FR-35	Functional test (contact message)	Validated
NFR-1	Observation (page response)	Validated
NFR-2	Observation (submission response)	Validated
NFR-3	Inspection (indexes)	Validated
NFR-4	Observation (terminal response)	Validated
NFR-5	Inspection (BCrypt hashing)	Validated
NFR-6	Inspection (parameterized queries)	Validated
NFR-7	Inspection (server-side session)	Validated
NFR-8	Functional test (input validation)	Validated
NFR-9	Inspection (no flag in client)	Validated
NFR-10	Inspection (single-screen layout)	Validated
NFR-11	Walkthrough (feedback clarity)	Validated
NFR-12	Inspection (visual consistency)	Validated
NFR-13	Inspection (responsive layout)	Validated
NFR-14	Functional test (persistence)	Validated
NFR-15	Functional test (atomic completion)	Validated
NFR-16	Functional test (error handling)	Validated
NFR-17	Extensibility demonstration	Validated
NFR-18	Design inspection (single context)	Validated
NFR-19	Design inspection (MVC structure)	Validated
NFR-20	Inspection (setup script)	Validated
NFR-21	Design inspection (portability)	Validated
NFR-22	Design inspection (extensible schema)	Validated
NFR-23	Inspection (browser-only client)	Validated
NFR-24	Design inspection (schema separation)	Validated

As Table E.1 shows, every one of the thirty-five functional and twenty-four nonfunctional requirements defined in Chapter 4 was validated, by functional testing, structured walkthrough, inspection, observation, or the extensibility demonstration, and every one was found to be satisfied. This complete matrix substantiates the claim that the delivered platform meets its specification in full.

References

[1] Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011). From game design elements to gamefulness: Defining "gamification." In *Proceedings of the 15th International*

[2] Chung, K., & Cohen, J. (2014). Learning obstacles in the capture the flag model. In *2014*

[3] Kolb, D. A. (2015). *Experiential learning: Experience as the source of learning and development* (2nd ed.). Pearson Education.

[4] Csikszentmihalyi, M. (1990). *Flow: The psychology of optimal experience*. Harper & Row.

[5] Prensky, M. (2001). *Digital game-based learning*. McGraw-Hill.

[6] Hamari, J., Koivisto, J., & Sarsa, H. (2014). Does gamification work? —A literature review of empirical studies on gamification. In *Proceedings of the 47th Hawaii*

[7] Cowan, C., Arnold, S., Beattie, S., Wright, C., & Viega, J. (2003). Defcon capture the flag: Defending vulnerable code from intense attack. In *Proceedings DARPA*

[8] Chapman, P., Burket, J., & Brumley, D. (2014). PicoCTF: A game-based computer security competition for high school students. In *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. USENIX Association.

[9] Petersen, R., Santos, D., Smith, M. C., Wetzel, K. A., & Witte, G. (2020). *Workforce framework for cybersecurity (NICE Framework)* (NIST Special Publication 800181 Revision 1). National Institute of Standards and Technology. https://do

[10] Cichonski, P., Millar, T., Grance, T., & Scarfone, K. (2012). *Computer security incident handling guide* (NIST Special Publication 800-61 Revision 2). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.80 0-61r2

[11] Strom, B. E., Applebaum, A., Miller, D. P., Nickels, K. C., Pennington, A. G., & Thomas, C. B. (2018). *MITRE ATT&CK: Design and philosophy* (MP180360). The MITRE

[12] Namukasa, M., Chaparro Osman, M., Ficke, C., Piasecki, I., O'Connor, T. J., & Carroll, M. (2024). Evaluation of an Internet of Things device-based educational approach to engage a more diverse cybersecurity workforce. *International Journal of Human Computer Interaction, 41*(2), 1364–1380. https://doi.org/10.1080/10

[13] Alothman, B. (2024). Cyber gamification: Implementing gamified adaptive learning environments for effective cyber security teams education. In *Proceedings of the 2024 5th International Conference on Education Development and Studies*

[14] Karagiannis, S., & Magkos, E. (2024). Leveraging gamification and game-based learning in cybersecurity education: Engaging and inspiring non-cyber students. *Journal of*

[15] Yasin, A., Fatima, R., JiangBin, Z., Afzal, W., & Raza, S. (2024). Can serious gaming tactics bolster spear-phishing and phishing resilience? Securing the human hacking in information security. *Information and Software Technology, 170*, 107426.

[16] Bellemans, J., Van Landuyt, D., Sion, L., & Desmet, L. (2024). Cybersecurity serious games development: A systematic review. *Computers & Security, 150*, 104307. https://doi.org/10.1016/j.cose.2024.104307

[17] Zhang, Y. (2024). Enhancing educational gamification through AI in higher education. In

[18] James, W., Oates, G., & Schonfeldt, N. (2024). Improving retention while enhancing student engagement and learning outcomes using gamified mobile technology.

[19] Ofte, H. (2024). The awareness of operators: A goal-directed task analysis in SOCs for critical infrastructure. *International Journal of Information Security, 23*(5),

[20] Pramod, D. (2024). Gamification in cybersecurity education: A state of the art review and research agenda. *Journal of Applied Research in Higher Education, 17*(4),

[21] Namukasa, M., Chauhan, B. B., Swords, C., et al. (2024). Diversifying cybersecurity: Evaluation of an Internet of Things (IoT)-based cybersecurity training course designed to bridge the diversity gap. *Journal of Cybersecurity Education, Research and Practice, 2025*(1), Article 9.

[22] Schafeitel-Tähtinen, T., Helenius, M., & Seppänen, V. (2025). Teaching and learning cybersecurity using capture the flag: Effectiveness comparison between university students in Finland and Czechia. *Computer Applications in Engineering Education,*

[23] Ali, F. (2025). Unlocking cybersecurity: A game-changing framework for training and awareness—A systematic review. *Human Behavior and Emerging Technologies,*

[24] González-López, O. R., Buenadicha-Mateos, M., Tato-Jiménez, J. L., & SánchezHernández, M. I. (2025). Adaptive gamification in higher education: Tailoring game elements to player profiles and gender differences. *SAGE Open, 15*(3). https://doi.org/10.1177/21582440251377738

[25] Rahartomo, A., Ghaleb, A. T. A., & Ghafari, M. (2025). Phishing awareness via game-based learning. In *2025 IEEE/ACM 37th International Conference on Software*

[26] Yasin, A., Fatima, R., Wen, L., JiangBin, Z., & Niazi, M. (2025). What goes wrong during phishing education? A probe into a game-based assessment with unfavourable results. *Entertainment Computing, 52*, 100815. https://doi.org/10.101 6/j.entcom.2024.100815

[27] Lazarov, W., Schafeitel-Tähtinen, T., Squillace, J., Martinasek, Z., Coufalikova, A., Helenius, M., Gallus, P., & Fujdiak, R. (2025). Lessons learned from using cyber range to teach cybersecurity at different levels of education. *Technology,*

[28] Kavvadias, A., & Kotsilieris, T. (2025). Understanding the role of demographic and psychological factors in users' susceptibility to phishing emails: A review. *Applied Sciences, 15*(4), 2236. https://doi.org/10.3390/app15042236

[29] Zulqarnain, M., Ismail, M. H., & Ab Aziz, N. A. (2025). Adaptive gamification in collaborative virtual classroom: A systematic review. *PeerJ Computer Science, 11*,

[30] Bahrini, M., Burdina, V., Sohr, K., & Malaka, R. (2025). Game elements in cybersecurity education: Hype or help? In *Entertainment Computing—ICEC 2025* (pp. 15–31).

[31] Idika, S., & Saihi, S. (2025). Enhancing student engagement through AI-driven adaptive learning and gamification. *British Journal of Education, 13*(12), 57–103.

[32] Provos, N., & Mazières, D. (1999). A future-adaptable password scheme. In *Proceedings of the USENIX Annual Technical Conference* (pp. 81–91). USENIX Association.

e3146. https://doi.org/10.7717/peerj-cs.3146