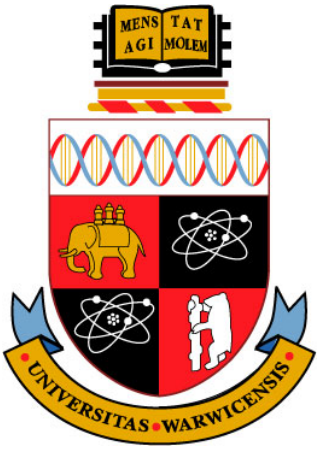

A Cross-Platform App for Interactive Learning on Security Attacks

CS310: Computer Science Project

Aman Hamidsha

Supervisor: Dr. Ligang He

Department of Computer Science

University of Warwick

2025-2026

Abstract

Everyday technology users remain largely underserved by existing cybersecurity education tools, which are predominantly designed for corporate or career-oriented audiences. This project presents the design and development of a cross-platform mobile and web application that delivers structured, interactive cybersecurity and privacy awareness training to non-expert users. The application features interactive simulations, cybersecurity lessons, quizzes and tips, gamified progress tracking, and live threat news integration. The project demonstrates that effective, engaging cybersecurity education can be made accessible to the general public without prior technical knowledge or institutional context.

Keywords: cybersecurity awareness, phishing simulation, cross-platform application, Flutter, gamified learning, privacy education, social engineering, non-technical users, NIST SP 800-50r1

Acknowledgements

Thanks to:

- **Dr. Ligang He and Dr. Kaihua Qin**: for your invaluable guidance
- **My girlfriend Thu**: for your sincere and boundless encouragement and support
- **My family and friends**: for your support and belief in my abilities

Contents

List of Figures

List of Tables

List of Code Snippets

1 Introduction

1.1 Motivations

Cybersecurity threats targeting individual users represent a growing and largely underserved challenge. While organisational security has benefited from decades of structured training frameworks and compliance-driven investment, the everyday technology user has been left with comparatively little support. The UK Cyber Security Breaches Survey 2025 reports that 43% of UK businesses and 30% of charities experienced a cybersecurity breach or attack in the preceding 12 months [5], yet this figure captures only the formally reported organisational surface. Individual users, who interact daily with phishing emails, fraudulent SMS messages, and deceptive websites, remain largely outside the scope of structured cyber education.

The consequences of this gap are tangible. Research by Bada, Sasse, and Nurse [1] highlights that awareness campaigns consistently fail to translate knowledge into behavioural change, particularly when material is perceived as impersonal or overly technical. This failure of engagement is not merely an academic concern. UK Finance reported that authorised push payment fraud, predominantly enabled by social engineering, resulted in losses of £450.7 million in 2024 alone [7], with individuals accounting for a significant proportion of victims.

Existing cybersecurity awareness tools do not adequately address this population. Most platforms are designed for corporate compliance or for individuals already pursuing cybersecurity careers, and consequently assume prior motivation and institutional context. Many suffer from poor engagement due to being overly theoretical, outdated, or structured in the style of academic or corporate training modules rather than accessible, practical learning tools. Interactive, scenario-based approaches (such as simulated phishing environments) are widely recognised as effective within organisations [5], yet no mainstream cross-platform application makes these available to the general public in an engaging, user-centred format. This is discussed at length, and citing specific examples, in section 2.2.

This project addresses that gap directly. The application is designed to bring structured, interactive, and practically grounded cybersecurity education to everyday, non-expert technology users. By combining gamified learning mechanics, real-world simulations, and

a curriculum grounded in established frameworks from the National Institute of Standards and Technology (NIST) [2] and the National Cyber Security Centre (NCSC), the application aims to make cybersecurity awareness salient, actionable, and genuinely engaging for users who have previously been overlooked by the industry.

1.2 Client

This application has been developed independently as an academic project. There is no external commercial client. The intended end users are everyday, non-technical technology users: individuals who interact regularly with digital services but who have not received formal cybersecurity training and are unlikely to seek out specialist resources independently. This target population is intentionally broad: the application is designed to be accessible regardless of prior technical knowledge, with an explicit objective of justifying the value of engagement to users who may not yet perceive themselves as potential victims of cyber threats.

1.3 Scope

The application is scoped as a cross-platform educational tool for individual users, built using Flutter to support simultaneous deployment on web, Android, and even Linux. The scope encompasses a structured learning curriculum, interactive quiz and challenge systems, four distinct simulation environments (email, SMS, crypto, and WiFi), a gamified progress system, and integration of real-world cybersecurity news for contextual learning.

The scope explicitly excludes: tools or content that could facilitate offensive cybersecurity activity; any technically detailed replication of attack mechanics; and features oriented toward organisational compliance or enterprise deployment. Simulations are constrained to recognition and decision-making exercises, and no exploit code, payloads, or actionable offensive techniques are included under any circumstance. This boundary is both an ethical requirement and a design principle: the application is for protection, not exploitation.

***Note**: Some ethical and managerial concerns are outlined in this report since covering them in the project management report would be infeasible. The scope of those discussions are limited.*

2 Background Research

This section presents the research underpinning the design and curriculum decisions of the application. It covers the landscape of existing cybersecurity education tools, the evidence base for individual-level cyber risk, established frameworks for effective learning programme design, and key findings from the academic literature on behaviour change and security awareness.

2.1 The Individual User Gap in Cybersecurity Education

The cybersecurity education landscape is heavily skewed toward organisational contexts. UK government reporting, NCSC publications, and NIST guidance are framed predominantly around businesses, public sector bodies, and charities. While this reflects the scale at which measurable institutional harm is reported and documented, it leaves a meaningful gap in provision for individual users.

This does not imply that individual users face negligible risk. Ofcom's Online Nation report [4] found that, consistently, from June 2024 to January 2025, 80% of the UK's adult internet users find 'Scams, fraud or phishing' amongst the top 10 areas of 'highest concern' when it comes to potential harms as a result of internet activity. Further, the report states that, across the same time frame, 34% of these users have actually encountered such content. Over 7000 people were surveyed. The NCSC's annual review consistently identifies phishing as the dominant attack vector across all user types [3], and UK Finance data confirms that individuals bear significant financial losses attributable to social-engineering-enabled fraud [7]. These harms are often undercounted in formal breach surveys because individual incidents rarely meet the reporting thresholds or institutional visibility of organisational incidents, not because they are genuinely less frequent.

Critically, the attack vectors most relevant to individuals such as phishing, smishing, pharming, fake websites, and social engineering are the same vectors identified as responsible for the majority of successful cyberattacks more broadly. Avast's Q1 2024 Threat Report [6] found that over 90% of successful cyberattacks begin with social engineering or human error. This underscores that the technical surface being addressed by this application is not marginal; it is central to the cybersecurity threat landscape.

2.2 Shortcomings of Existing Tools

Existing cybersecurity awareness tools aimed at general users suffer from consistent and well-documented design shortcomings. Many are overly theoretical, structured as academic courses, or derived from corporate compliance training that assumes a captive, institutionally motivated audience. Reviews of freely available platforms frequently identify dated content, lack of interactivity, and absence of cross-platform support as core limitations.

Bada et al. [1] identify a fundamental failure mode in awareness campaigns: the assumption that communicating risk information is sufficient to produce behaviour change. In practice, users must first accept that the information is personally relevant, then understand the appropriate response, and finally be motivated to act despite competing demands on their attention. Applications that present security information without addressing personal relevance are typically treated as obligatory and ignored. Fear-based or technically intimidating framing compounds this problem by increasing stress and encouraging disengagement rather than action; a phenomenon documented as security fatigue [1].

Mock phishing exercises and similar simulation-based approaches are well-established as effective training mechanisms within organisations [5]. The central insight driving this project is that these techniques, which are demonstrably effective at building practical recognition skills are not currently available to individual users in an accessible, self-directed format. The design of this application is therefore guided by the principle that the method of learning matters as much as the content.

Illustrating this point with some concrete examples would be quite relevant at this stage. Several apps come close to filling the said market gap. However, all of them lack the completeness in features, or scope, or both to truly address this issue.

Whilst tryhackme is generally an excellent platform, and is a signficantly better web application than this project in its current state, and has a more exhaustive library of resources, it is still geared towards cyber security students and aspiring cybersecurity professionals, and not towards the general public.

The android app Cyber Warriors does earnestly attempt to resolve

this gap in the market. However, the quality of the application in general, does seriously warrant questions about its effectiveness. Apart from there being a lot of paywalled content, AI-generated descriptions, and a mere 100 downloads with no reviews, the application is not cross platform, and does not present a hint of interactivity. Our testing team notes that this application makes cyber awareness training feel like a chore, which is the last thing the general public wants to experience.

Table 2.1: Comparing App to Potential Competitors.

App \ Benefit	Geared to General Public	No Paywalled Content	Cross-Platform	Exhaustive Resource Library	High Interactivity
tryhackme	✗	✗	✗	✓	✓
Cyber Warriors	✓	✗	✗	✗	✗
This Project	✓	✓	✓	✓	✓

2.3 Curriculum Design: NIST SP 800-50r1

The curriculum structure of the application draws directly from NIST Special Publication 800-50 Revision 1 [2], which provides a comprehensive framework for designing Cyber Security and Privacy Learning Programmes (CPLPs). Although this publication is oriented toward organisational deployment, its principles for structuring learning goals, objectives, outcomes, and assessments are generalisable and directly applicable to self-directed individual learning.

Three principles from NIST SP 800-50r1 are of particular relevance. First, the framework emphasises the importance of integrating privacy awareness alongside cybersecurity training, recognising that many threats to individuals (including data harvesting, account compromise, and identity theft) sit at the intersection of security and privacy. This informed the decision to incorporate privacy-oriented content as a core strand of the curriculum rather than an optional extension.

Second, the publication provides detailed guidance on the types of learning programme data that should be collected to evaluate effectiveness. This directly shaped the application's data collection design, which includes quantitative metrics (streak completion, leaderboard engagement, login frequency, time-on-content, module completion rates) and qualitative mechanisms (open-ended feedback, surveys, and provision for focus group evaluation). Behavioural

competence is assessed through performance in interactive simulations (such as correctly identifying phishing content) rather than through knowledge-recall testing alone, in line with the framework's emphasis on demonstrated capability over declarative knowledge.

Third, the framework's goal-objective-outcome structure, was used to define the learning hierarchy of the application's modules, ensuring that each topic area has measurable outcomes and appropriate assessment mechanisms.

2.4 Behaviour Change and Presentation Design

A further critical strand of the research concerns how information must be presented to produce genuine behaviour change rather than passive consumption. Bada, Sasse, and Nurse [1] argue that three conditions must be satisfied for a security awareness intervention to be effective: the user must believe the risk is personally relevant; they must know how to respond; and they must be willing to do so. Failing to satisfy the first condition, namely personal relevance, renders even technically accurate and well-structured content ineffective.

This finding has direct implications for the design of an application targeting everyday users who have not chosen to engage with cybersecurity content and may not perceive themselves as likely victims. The application must not merely present information; it must justify, in accessible and concrete terms, why that information matters to the specific user engaging with it. This is treated in this project as a core design problem, not a cosmetic consideration.

The authors also highlight that awareness material perceived as impersonal or overly complex tends to be treated as a box-ticking exercise, and that users can often answer awareness quiz questions correctly while failing to act accordingly in real contexts. This motivates the application's emphasis on scenario-based, interactive learning over knowledge-recall quizzes, and its use of realistic simulation environments as the primary vehicle for developing practical recognition and decision-making skills.

3 Requirements and Risks

This section formalises the functional and non-functional requirements of the application, followed by an analysis of technical, managerial, and ethical risks.

Requirements are prioritised using the MoSCoW framework: **M**ust Have, **S**hould Have, **C**ould Have, and **W**on't Have (in this version).

Risks are outlined through risk registers which are structured as a tabular log where each risk is assigned an identifier and described alongside its likelihood, impact, and corresponding mitigation strategy.

3.1 Functional Requirements

Functional requirements describe what the system must do from the user and system perspective, these requirements are especially important because the application is not a single-purpose tool, but a multi-feature learning platform combining authentication, educational content delivery, assessment, simulation, and progress tracking. The functional requirements therefore define the minimum set of behaviours needed for the system to achieve its educational objective. Without these clearly defined functions, the project would risk becoming a collection of disconnected pages rather than a coherent cybersecurity training application.

Table 3.1: MoSCoW prioritisation of functional requirements

Priority	ID	Functional Requirement
Must Have	FR1	The system must allow users to register, log in, log out, and switch account.
Must Have	FR2	The system must restrict protected areas of the application to authenticated users and redirect unauthenticated users appropriately.
Must Have	FR3	The system must provide a dashboard that summarises learning activity, xp, streaks, and progress indicators.
Must Have	FR4	The system must provide structured lessons organised into topic chapters.
Must Have	FR5	The system must provide interactive lesson checks such as fill-in-the-blank, scenario selection, and matching activities.
Must Have	FR6	The system must store and update lesson progress across sessions.
Must Have	FR7	The system must provide quiz banks for multiple cybersecurity topics.
Must Have	FR8	The system must generate quiz sessions and display results after completion.
Must Have	FR9	The system must provide realistic simulators for email, sms, wi-fi, and crypto scam awareness.
Must Have	FR10	The system must evaluate simulator responses and provide a score, verdict, and feedback.
Must Have	FR11	The system must award xp for relevant learning activities and update streak-related progress.
Should Have	FR12	The system should provide a leaderboard view to compare users through progress-related metrics.
Should Have	FR13	The system should support backend communication for selected features such as keyword briefing and simulator response persistence.
Should Have	FR14	The system should store simulator response history and user progress on the backend where supported.
Should Have	FR15	The system should support registration verification and password reset email workflows through the backend.
Could Have	FR16	The system could synchronise all progress data fully across multiple platforms and devices.
Could Have	FR17	The system could extend backend support so that more local-only features become fully server-backed.
Won't Have	FR18	The system will not provide full real-time multiplayer or real-time competitive interactions between users in the current version.
Won't Have	FR19	The system will not provide advanced ai-generated personalised tutoring or natural-language explanation generation in the current version.

3.2 Non Functional Requirements

The non-functional requirements focus on the quality of the learning experience and the quality of the software architecture. Since the system is intended as an educational tool, usability and interface clarity are especially important.

Table 3.2: MoSCoW prioritisation of non-functional requirements

Priority	ID	Non-Functional Requirement
Must Have	NFR1	The system must provide a clear, usable, and educationally appropriate user interface.
Must Have	NFR2	The system must remain responsive across supported layouts and screen sizes.
Must Have	NFR3	The system must provide rapid feedback after quiz and simulator interactions.
Must Have	NFR4	The system must persist essential local data such as theme settings and progress where required.
Must Have	NFR5	The system must be maintainable through modular code organisation separating app, core, feature, and server responsibilities.
Must Have	NFR6	The system must be reliable enough to support repeated learning sessions without frequent crashes or broken navigation.
Must Have	NFR7	The system must support cross-platform execution through a shared Flutter codebase.
Should Have	NFR8	The system should maintain consistent visual design, interaction patterns, and feedback styles across major features.
Should Have	NFR9	The system should be extensible so additional lessons, quizzes, simulators, and backend features can be added later.
Should Have	NFR10	The system should use typed backend communication and structured persistence to improve data integrity and maintainability.
Should Have	NFR11	The system should degrade gracefully when optional backend resources or external data sources are unavailable.
Should Have	NFR12	The system should support a reasonable level of security for authentication and stored backend data within the scope of the project.
Could Have	NFR13	The system could provide stronger automated testing coverage across ui, domain, and backend components.
Could Have	NFR14	The system could provide fuller cross-device synchronisation and consistency for all learning progress data.
Could Have	NFR15	The system could include more advanced accessibility refinements and platform-specific optimisation in future iterations.
Won't Have	NFR16	The system will not target production-grade horizontal scalability or enterprise deployment requirements in the current version.
Won't Have	NFR17	The system will not implement full offline-first synchronisation with conflict resolution in the current version.

3.3 Technical Risks

Technical risks tend to arise as a result of the size of the codebase, however, they are mostly manageable. This project combines a Flutter front end, local persistence, simulator scoring engines, and a Serverpod backend with database and email configuration, meaning that failures can emerge not only from coding errors but also from integration points between components.

Table 3.3: Technical Risk Register

ID	Risk	Likelihood	Impact	Mitigation
TR1	Backend startup may fail due to incorrect database, smtp, or password configuration.	Low	High	Use example config files, setup scripts, documented startup steps, and early environment validation.
TR2	PostgreSQL may be unavailable or incorrectly configured during development and demonstration.	Low	High	Use containerised setup and migration scripts, and test backend availability early in each sprint.
TR3	Different platforms may fail to connect to the same backend because localhost behaves differently on desktop, web, and mobile.	High	Medium	Use configurable backend urls through dart defines and document platform-specific launch commands.
TR4	Linux desktop builds may fail because of missing native packages or compiler toolchain issues.	Medium	Medium	Document required dependencies, test build environment early, and keep a fallback demonstration target available. Medium impact because Linux build is supplementary.
TR5	Local-only progress and backend-supported data may diverge, causing inconsistent behaviour across devices.	High	Medium	Clearly separate local and backend responsibilities and identify full synchronisation as staged future work.
TR6	Simulator scoring logic may produce misleading or inconsistent feedback if rules are poorly calibrated.	Medium	Medium	Analyzers are deterministic, scenario-specific, and easy to review and refine during testing. An AI powered analyzer is an extension task. The analyzers are currently accurate enough to not make big mistakes.
TR7	External data fetches such as keyword news or vulnerability lookups may fail or return poor-quality data.	Medium	Medium	Fail safely, and return empty/default results when needed.
TR8	Data stored in `SharedPreferences` may become invalid, stale, or inconsistent after updates or resets.	Medium	Medium	Use defensive parsing, default fallbacks, and reset pathways for corrupted local state.
TR9	Port conflicts or stale running processes may stop the backend from launching during development.	Low	Medium	Use startup and stop scripts, check for running processes, and document restart steps.
TR10	Partial backend integration may leave some features prototype-level rather than fully production-ready.	High	Medium	Treat local persistence as an interim implementation choice and scope future backend migration clearly.

3.4 Managerial Risks

There are many possible directions in which this project could develop, which creates a risk of tunnel vision or of spending too much time pursuing interesting ideas that do not directly support the core objectives.

Table 3.4: Managerial Risk Register

ID	Risk	Likelihood	Impact	Mitigation
MR1	Project scope may become too broad because the system includes lessons, quizzes, multiple simulators, dashboard features, and backend work.	High	Low	Prioritise must-have features using MoSCoW and manage delivery through sprint goals. Although, more lesson content is generally welcomed.
MR2	Time may be lost to setup and integration work rather than visible feature development.	Low	Medium	Track enabling tasks explicitly in Kanban and **address technical blockers early.**
MR3	Delays in backend integration may compress time available for testing and refinement.	Medium	Medium	Develop frontend modules incrementally and treat backend expansion as staged rather than all-at-once.
MR4	Estimation may be inaccurate because feature complexity is discovered gradually during implementation.	Medium	Medium	Use short sprint cycles and regular supervisor review to re-estimate and reprioritise.
MR5	Excessive focus on visual polish could reduce time available for core learning functionality.	High	Medium	Tie UI work to educational goals and protect time for functional completion and testing. This presents a potential risk, as the lead engineer has a strong inclination toward UI design.
MR6	A solo development structure increases dependency on one person for implementation, documentation, testing, and deployment.	High	Medium	Use Trello, version control, commit history, and documentation to maintain traceability and discipline.
MR7	Requirement changes or clarification from supervision may disrupt sprint plans.	Low	Low	Use bi-weekly reviews as controlled reprioritisation points rather than changing direction continuously.
MR8	Testing coverage may be insufficient if too much effort is consumed by feature implementation.	Medium	Medium	Prioritise testing of highest-risk flows such as auth, progress persistence, and simulator evaluation.
MR9	Documentation may lag behind implementation, reducing report quality and maintainability.	Low	Low	Update documentation incrementally during feature completion rather than leaving it until the end.

3.5 Ethical Risks

Due to the nature of this project, and the industry in which it lies, being wary of potential ethical risks is non-negotiable. The application deals directly with cybersecurity awareness, simulated scams, account handling, and user progress data, all of which carry ethical implications even in an educational setting. For example, realistic phishing and fraud simulations must be handled carefully so that they educate users without causing unnecessary distress or unintentionally normalising harmful behaviour.

Table 3.5: Ethical Risk Register

ID	Risk	Likelihood	Impact	Mitigation
ER1	Realistic scam simulations may unintentionally distress or confuse some users.	Medium	Low	Frame scenarios clearly as educational exercises and provide explanatory feedback after each interaction.
ER2	Simulated examples may accidentally normalise scam behaviour if they are shown without enough corrective context.	Low	High	Pair every scenario with red flags, safe actions, verdicts, and recommended next steps.
ER3	Storing user credentials locally in a simplified prototype auth system may not meet strong real-world security expectations.	Medium	High	Clearly present the auth implementation as prototype-level and avoid claiming production-grade security.
ER4	Progress, response history, or other learner data may create privacy concerns if stored without sufficient transparency.	Medium	Medium	Minimise stored personal data, document what is stored, and separate prototype local storage from future secure deployment claims.
ER5	Leaderboards and streak systems may create unhealthy pressure or discourage weaker users.	Low	Low	Use these systems as motivational aids rather than punitive measures and balance them with supportive feedback. Low impact since established applications with leaderboards report exceptional user satisfaction (based on reviews), e.g. Duolingo.
ER6	Educational scoring may oversimplify complex cybersecurity judgement and give users false confidence.	Medium	High	Present analyzers as guided training tools, not perfect security authorities, and explain limitations in documentation.
ER7	External keyword briefing content may include inaccurate, biased, or outdated material.	Medium	Medium	Use trusted sources where possible and treat external content as supplementary rather than authoritative core teaching material.
ER8	Cross-platform inconsistencies may lead users to believe their progress or account state is universally synced when it is not.	High	High	Communicate current system limitations clearly and avoid overstating backend completeness.
ER9	The use of persuasive UI elements such as XP, streaks, and rewards may shift focus from learning to reward chasing.	Medium	Medium	Ensure rewards support, rather than replace, the educational purpose of the application.

4 Design

This section outlines the system design, covering the tech stack and component structure, as well as the design decisions for the overall app and the parts that make it up.

4.1 Tech Stack

Flutter is the frontend framework used to build the application's user interface, including the lessons, quizzes, dashboard, and simulator screens, while Riverpod is the state management library that handles shared app state such as authentication, theme mode, and routing-related dependencies. On the backend, Serverpod provides the server framework, including endpoints, generated protocol models, and integration with authentication and persistence features. PostgreSQL is the relational database used to store structured backend data such as simulator responses, user progress, and authentication-related records.

Table 4.1: Technology Choices and Justification

Technology	**Justification**
Flutter	A single codebase supporting a polished multi-platform UI. Suitable for lessons, quizzes, dashboards, and highly customised simulator interfaces due to its flexible widget system.
Riverpod	Provides reactive shared state for authentication, theme mode, routing, and app bootstrap. More scalable and cleaner than manual state propagation.
Serverpod	Enables backend development within the Dart ecosystem, reducing friction and simplifying typed client–server communication.
PostgreSQL	Mature relational database suited to structured data such as user progress, simulator responses, and authentication. Integrates cleanly with Serverpod.

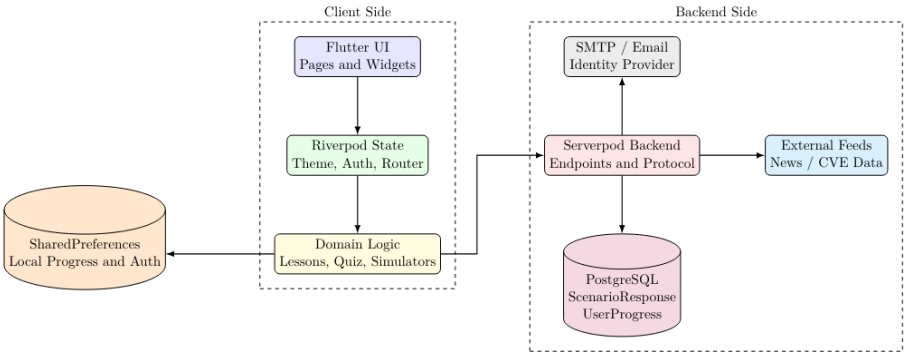

Figure 4.1: System Architecture Diagram

The system architecture was the single biggest influence on the choice of tech stack. As shown in the figure above, the system follows a clear client-server separation. On the client side, the Flutter application is structured into three layers: the UI layer, which handles pages and widgets; the state management layer, which manages authentication, theming, and routing via Riverpod; and the domain logic layer, which encapsulates the core learning features including lessons, quizzes, and simulators. Local session data and progress are persisted on-device using SharedPreferences. On the backend side, a Serverpod server exposes typed endpoints and handles all communication with a PostgreSQL database, an email identity provider, and external feeds for news and CVE data.

4.2 UI

The Figma UI design for this application was developed with established UI design principles in mind to ensure that the application was not only visually appealing, but also clear, usable, and appropriate for its educational purpose.

Table 4.2: Design Principles and Justification

Principle	Justification
Hierarchy	Ensures users can immediately identify key elements such as titles, actions, and progress indicators through contrast in font size, weight, spacing, and colour. This guides attention and clarifies information priority.
Progressive Disclosure	Content is revealed step by step in lessons, quizzes, and simulators to avoid overwhelming users. This reduces cognitive load and improves clarity in an educational context.
Consistency	Repeated layouts, navigation patterns, and visual styles create familiarity across the application. Strategic contrast highlights important actions, warnings, and feedback, especially in cybersecurity scenarios.

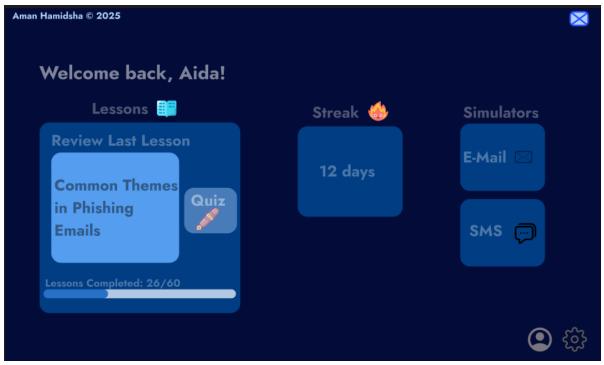

Figure 4.2: Web app dashboard wireframe

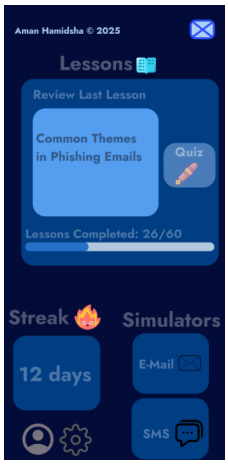

Figure 4.3: Mobile dashboard wireframe

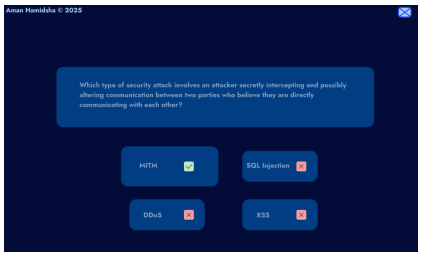

Figure 4.4: Web Quiz Interface Wireframe

Figure 4.5: Mobile SMS Simulator Wireframe

The components of the UI are linked together to form a consistent and effective learning experience, with the navigation structure built around a single guiding (UX) principle: user-centricity. Every navigational decision prioritises the experience of the user above all else.

The component tree shown below was thus refined well in order to form a complete navigation map for the app.

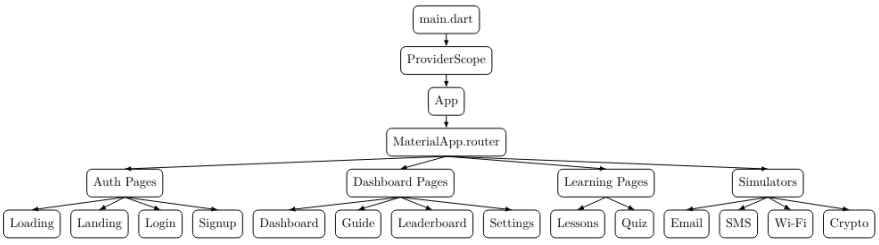

Figure 4.6: Component Tree

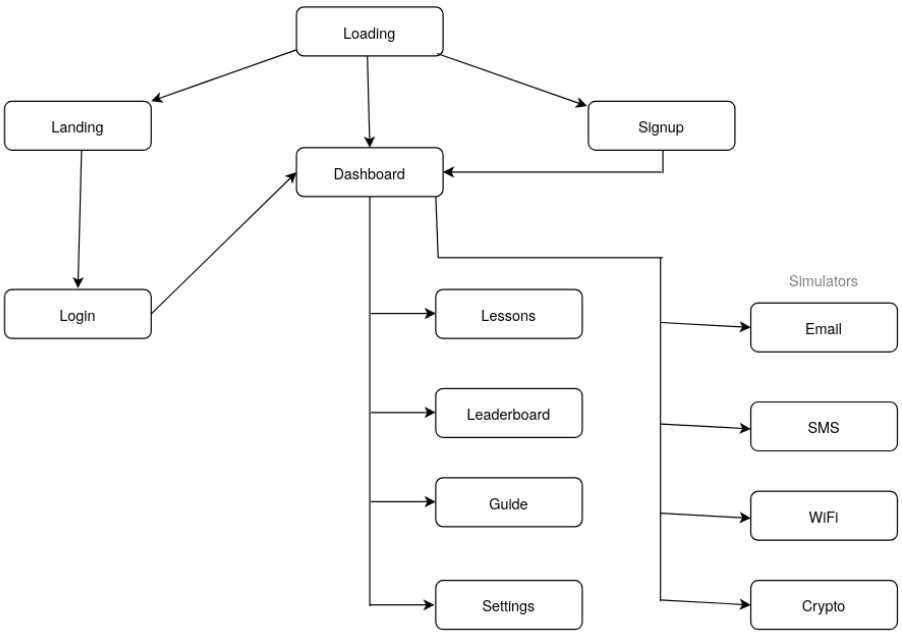

Figure 4.7: Navigation Map

4.3 Overall Learning Procedure

Based off of the research conclusions from section 2, the learning program was designed with interactivity as the main priority, whilst also taking care to not sacrifice rigour and lesson quality. The flowchart below helps illustrate the learning procedure.

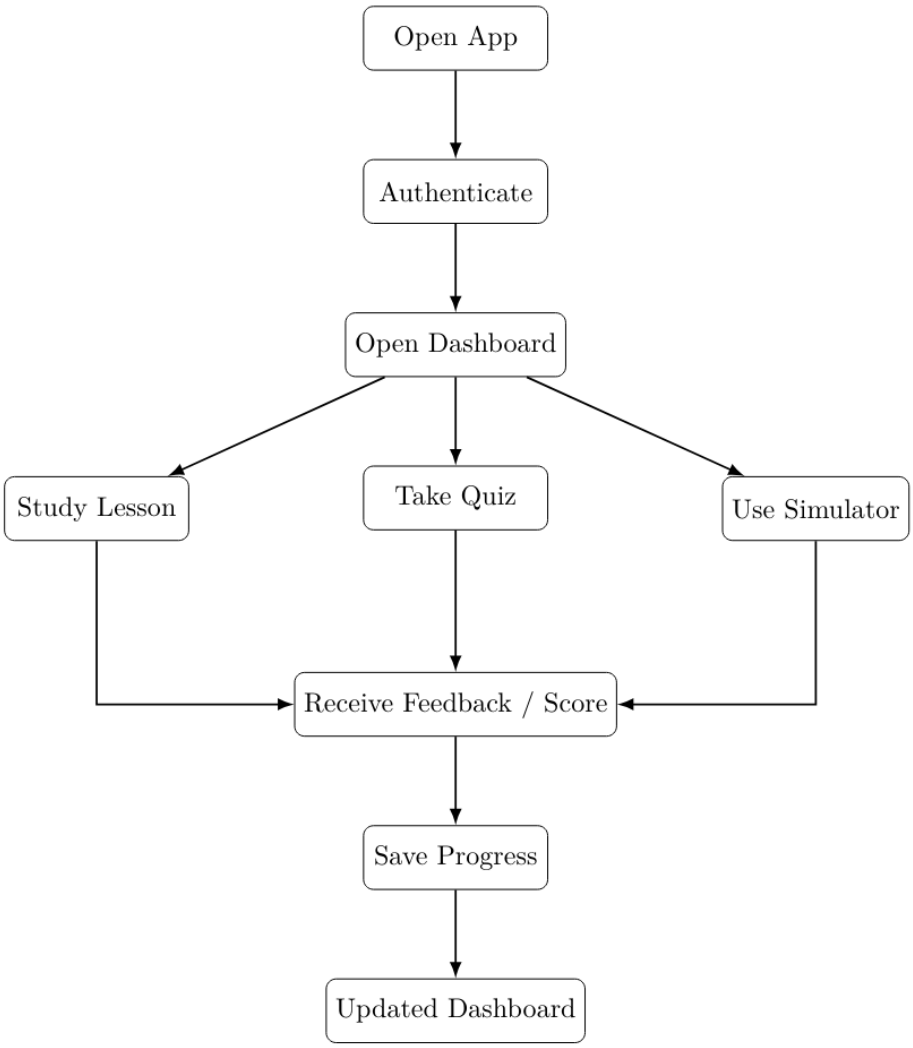

Figure 4.8: Learning Procedure Flowchart

4.4 Lessons and Quizzes

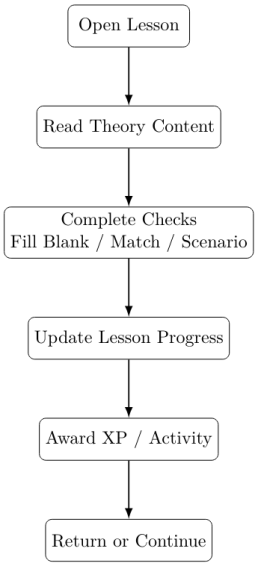

Figure 4.9: Lesson Flowchart

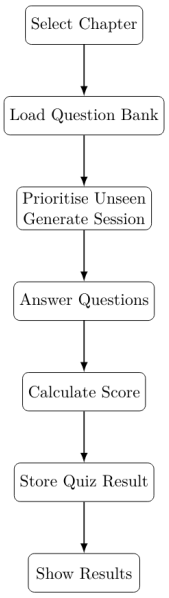

Figure 4.10: Quiz Flowchart

The lessons structure is quite straightforward, but there will also be a list of keywords at the start of each lesson, and when the keywords are pressed, latest cyber security news and CVEs related to said keyword will be displayed. There will also be a button in each lesson that redirects the user to its corresponding quiz.

Quizzes are structured differently from lessons. Each quiz draws from a large question bank, prioritising questions the user has not yet seen. Sessions are deliberately capped at 10 questions, a decision grounded in cognitive load theory, which suggests that shorter, focused practice sessions improve information retention more effectively than exhaustive ones.

4.5 Simulators' Design

All four simulators have a lot in common. At a structural level, they combine presentation, domain logic, and feedback generation in a consistent way. The user-facing pages are responsible for displaying scenarios and collecting decisions, while the supporting domain models and response engines handle the representation and evaluation of those decisions. This creates a coherent simulator framework

across the application, even though the specific threats being modelled differ between modules.

4.5.1 Email

The email simulator presents the user with realistic phishing and scam-style email scenarios and asks them to decide how to respond. It is designed to help users recognise suspicious message patterns, unsafe requests, and safer verification habits in an email context.

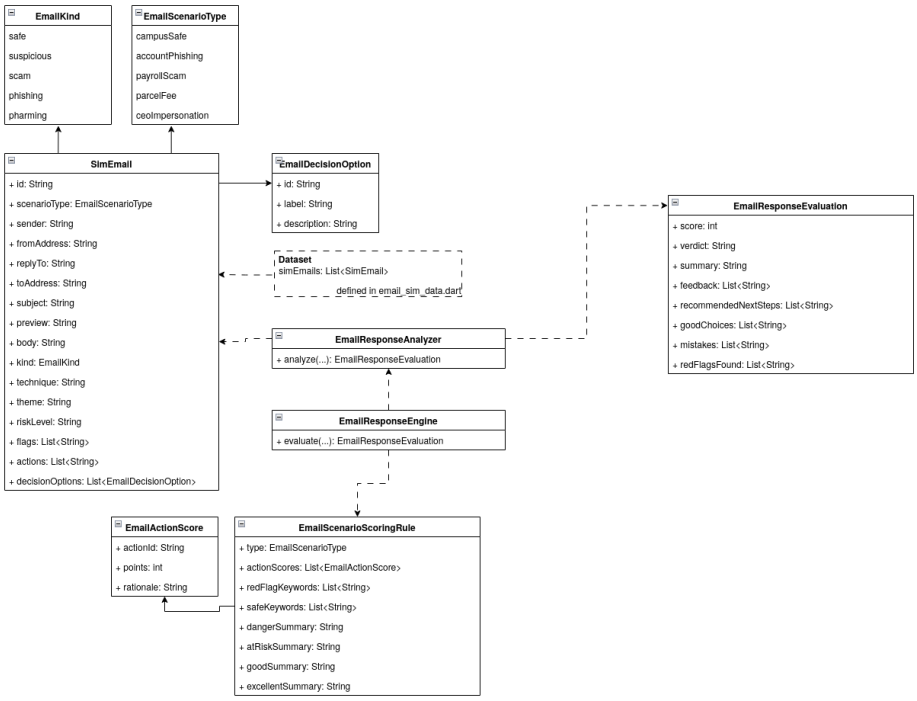

Figure 4.11: Email Simulator Class Diagram

The diagram above models the full structure of the email simulator. A `SimEmail` is the core entity, representing a simulated phishing or safe email scenario. It carries all the fields you would expect of a real email (sender, subject, body, etc.) alongside simulation-specific metadata such as risk level, flags, and a list of `EmailDecisionOption` objects representing the choices presented to the user. Each `SimEmail` has an `EmailKind` (safe, suspicious, scam, phishing, or pharming) and an `EmailScenarioType` (such as `accountPhishing` or `ceoImpersonation`), which together classify the nature of the scenario. All emails are stored in a static `Dataset` defined in `email_sim_data.dart`.

When the user submits their response, the `EmailResponseAnalyzer`

takes the selected actions and reply text and delegates the actual scoring to the `EmailResponseEngine`. The engine loads the relevant `EmailScenarioScoringRule` for the scenario type, which defines per-action point values via `EmailActionScore`, as well as red-flag and safe keywords to scan the reply text against. The engine produces an `EmailResponseEvaluation` containing the final score, a verdict, a summary, and detailed feedback including good choices made, mistakes, red flags found, and recommended next steps.

4.5.2 SMS

The SMS simulator performs the same educational role in a text-message setting, using short conversational threads to model smishing, impersonation, and fraud-alert scenarios. It helps the user practise identifying suspicious urgency, risky links, and unsafe replies in a format that reflects real mobile communication.

The SMS class diagram is almost one-to-one with the email version, with only minor differences to reflect the message-thread structure and SMS-specific scenario details.

4.5.3 Wi-Fi

The Wi-Fi simulator teaches users how to judge the trustworthiness of public wireless networks and understand the risks of connecting to unsafe hotspots. Its system design is more self-contained than the email and SMS simulators because the learning experience depends heavily on the visual transition between a network-selection view and an attacker-perspective reveal. Rather than using a separate analyzer and scoring-rule structure, it is built around scenario-rich network objects, captured-data examples, and a stateful UI flow that exposes the consequences of a poor decision in a more visual and environmental way.

4.5.4 Crypto Trading Environment

The crypto simulator teaches users how to evaluate suspicious investment opportunities and recognise patterns commonly associated with crypto fraud, hype, and unsafe wallet behaviour. Its design combines scenario data, user decision options, evaluation logic, and outcome tracking into a more game-like flow than the communication-based simulators. Instead of modelling a single message exchange, it presents a sequence of projects with warning signs, asks the user whether to invest or walk away, and then evaluates

both the action taken and the reasoning given. This makes it well suited to teaching slower, higher-stakes judgement in a financial-risk context.

4.6 Response Analyzers

Both the SMS, and email simulators have response analyzers that work the exact same way.

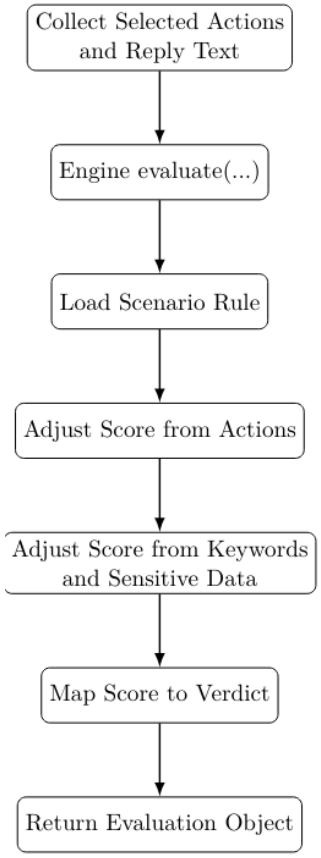

Figure 4.12: Analyzer Flowchart

The response analyzers were designed as deterministic, rule-based systems that convert a user's selected actions and written response into a structured evaluation. In the email and SMS simulators, this design is based around scenario-specific scoring rules, keyword checks, and fixed verdict bands, which makes the analysis process transparent and easy to follow. From a system design perspective, this is useful because it keeps the logic explainable, testable, and closely aligned with the educational aims of the project. It also

reduces implementation complexity, since the analyzer can be fully contained within the app's own codebase without relying on external services or additional model infrastructure.

4.7 Data Models

The application uses several different groups of data models, each serving a specific purpose within the system. At the frontend level, there are models that represent educational content and user progress, such as `LessonCatalogEntry` and `LessonProgressSnapshot` for the lessons feature, and `DashboardSocialSnapshot`, `LeaderboardEntry`, and `SocialActivityDay` for dashboard and streak-related functionality.

A second major set of data models supports the simulator system. In the email and SMS simulators, models such as `SimEmail`, `EmailDecisionOption`, `EmailResponseEvaluation`, `SmsThread`, `SmsMessage`, `SmsDecisionOption`, and `SmsResponseEvaluation` define the scenarios, user choices, and evaluation results that drive the learning experience. The Wi-Fi and crypto simulators use similar domain-specific models, such as `SimWifiNetwork`, `WifiCapturedDatum`, `CryptoProject`, `InvestmentDecision`, and `CryptoDecisionEvaluation`, although these are implemented more directly within their feature files.

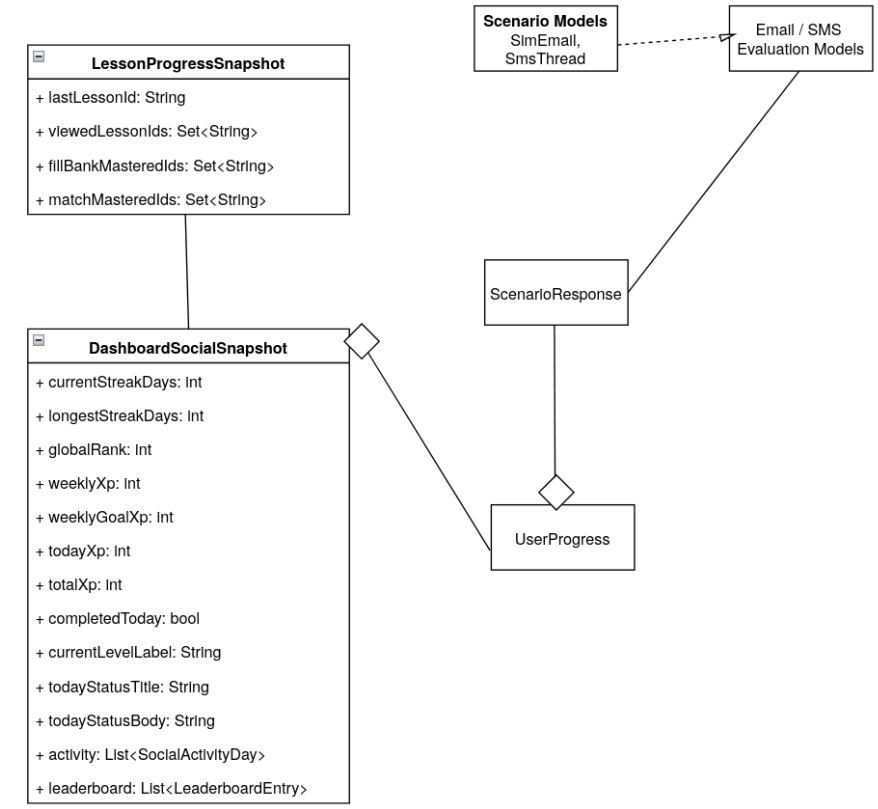

Figure 4.13: Front End Data Model Diagram

4.8 Database and Backend

The backend system was designed as a typed Serverpod service layer responsible for persistence, endpoint handling, and structured communication between the Flutter client and the database.

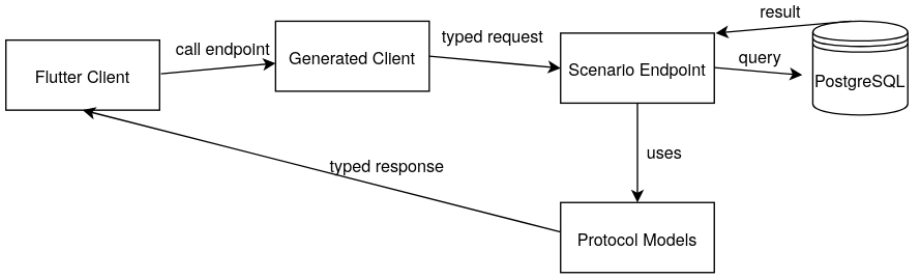

Figure 4.14: Backend Endpoint Flow Diagram

The endpoint diagram illustrates how this backend flow operates in practice. A request begins in the Flutter client, passes through

the generated client layer, and is then handled by ScenarioEndpoint, which contains the custom server-side logic for operations such as response analysis storage, progress retrieval, and keyword briefing generation. The endpoint works with protocol models to keep request and response data strongly structured, and where required it reads from or writes to the PostgreSQL database.

5 Implementation

This section explains how the design of the project was translated into a working software system.

5.1 Development Structure

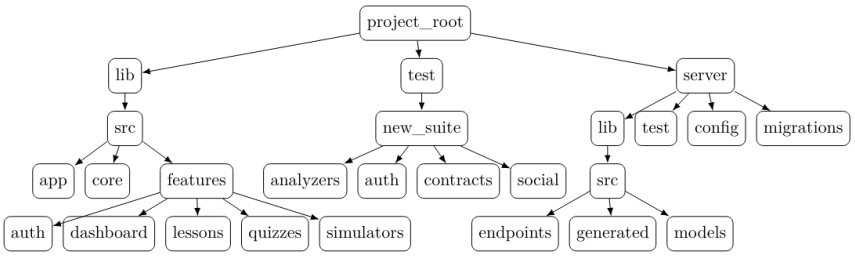

Figure 5.1: Directory Tree

At the top level, the codebase is separated into the Flutter client, automated tests, and the Serverpod backend, which immediately distinguishes user-facing code from backend infrastructure. Within the client, the `app` directory contains global concerns such as routing, theming, and app-wide setup, while the `core` directory holds shared services such as backend configuration and bootstrap logic. The `features` directory then contains the main functional areas of the system, including authentication, dashboard, lessons, quizzes, and simulators. This makes the structure easier to understand because related code stays grouped by feature rather than being scattered across the project.

Inside those feature folders, there is a further separation between presentation and domain responsibilities. Presentation files contain pages and widgets, while domain files contain models, scoring logic, canned data, and persistence helpers. This is especially clear in the simulator modules, where the user interface is separated from the response engines and analyzers that actually perform the educational evaluation. The directory tree is therefore the way it is because the app needs to support multiple distinct learning systems while still remaining maintainable. By separating app-wide concerns, shared infrastructure, feature modules, and backend code, the project reduces coupling, improves readability, and makes it easier to extend or test individual parts of the system without affecting everything else.

5.2 Flutter Client

```dart
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(const ProviderScope(child: App()));
}
```

Listing 5.1: `main.dart` Implementation

The Flutter client begins in `main.dart`, which serves as the entry point of the frontend application. Here, Flutter bindings are initialized and the root app is launched inside a `ProviderScope`, making Riverpod state management available throughout the widget tree from the very start. This is an important implementation detail because it establishes the application as a reactive frontend system rather than a collection of isolated screens.

```dart
class App extends ConsumerWidget {
  const App({super.key}); // constructor with optional key

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final GoRouter router = ref.watch(
      appRouterProvider,
    ); // watching the router to get GoRouter instance
    final themeMode = ref.watch(themeModeProvider);
    return MaterialApp.router(
      debugShowCheckedModeBanner: false,
      title: 'CS310',
      theme: appLightTheme,
      darkTheme: appDarkTheme,
      themeMode: themeMode,
      routerConfig: router, // attach GoRouter config for
          navigation
    );
  }
}
```

Listing 5.2: `app.dart` Implementation

From there, the `App` widget in `app.dart` constructs the main application shell using `MaterialApp.router`. This centralises the app's theme configuration, title, and route handling in one place, allowing the frontend to operate as a single coherent system. The use of route-driven navigation is particularly important in this project because the user must move between multiple functional areas, including authentication pages, the dashboard, lessons, quizzes, and simulators.

5.3 Authentication

Authentication is implemented through a local controller-based flow built around `AuthController`, `AuthState`, `SharedPreferences`, and the app router. When the application starts, the controller initializes by loading any stored account data and setting the authentication state, after which login and registration requests are handled by validating credentials against locally stored account records and updating the current session key. Once the state changes, the route system reacts automatically and redirects the user to the appropriate area of the app, such as the dashboard for authenticated users or the login page for unauthenticated users. The authentication sequence diagram illustrates this clearly by showing how user input from the login page passes through the controller, updates local persistence, and then triggers the routing logic that controls access to protected sections of the application.

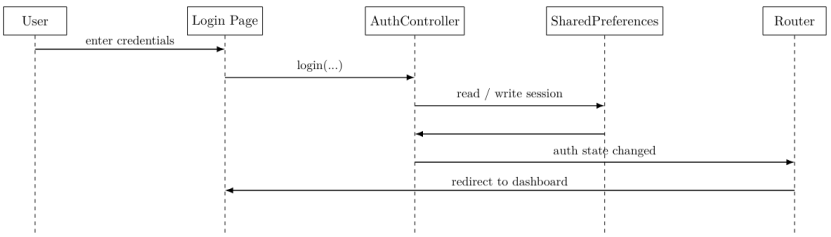

Figure 5.2: Authentication Sequence Diagram

5.4 Lesson System, Quiz, and Dashboard

The lesson system is implemented around a structured progress model and a persistent local storage layer. `LessonCatalogEntry` defines the lesson metadata used to populate the lesson list, while `LessonProgressSnapshot` stores the user's progress across three tracked lesson steps: viewing the lesson, completing the fill-in-the-blank task, and completing the match activity. `LessonProgressStore` is responsible for loading, updating, and saving this snapshot through `SharedPreferences`, which allows lesson state to persist between sessions.

The quiz system is implemented as a chapter-based assessment feature that works alongside the lessons rather than independently from them. Quiz logic is organised around question banks and the generation of quiz sessions from selected chapters, allowing the user to answer a set of questions, receive a score, and then review their re-

sult.

The dashboard is implemented as the app's main progress aggregation layer. It works by combining learning activity information into a structured `DashboardSocialSnapshot`, which contains data such as streak length, XP totals, daily completion state, recent activity, and leaderboard entries. This snapshot is built by `DashboardSocialActivity.loadSnapshot()`, which reads and derives stored profile information from local persistence.

The leaderboard is implemented as part of this same social-data system. Internally, user profiles are loaded and sorted according to total XP, current streak, and username, producing a list of `LeaderboardEntry` objects that can be shown directly in the UI. This gives the app a simple but effective ranking system that can compare different local users consistently. Technically, the leaderboard is not a separate persistence structure of its own, but rather a derived output generated from the stored social profile data, which keeps the design compact and avoids redundant stored values.

XP is handled through recording a specific learning action, checking whether it has already been counted for that day, updating streak status, incrementing total XP, and storing the resulting state. Different activities are represented through `UserActivityType`, each of which carries a default XP reward.

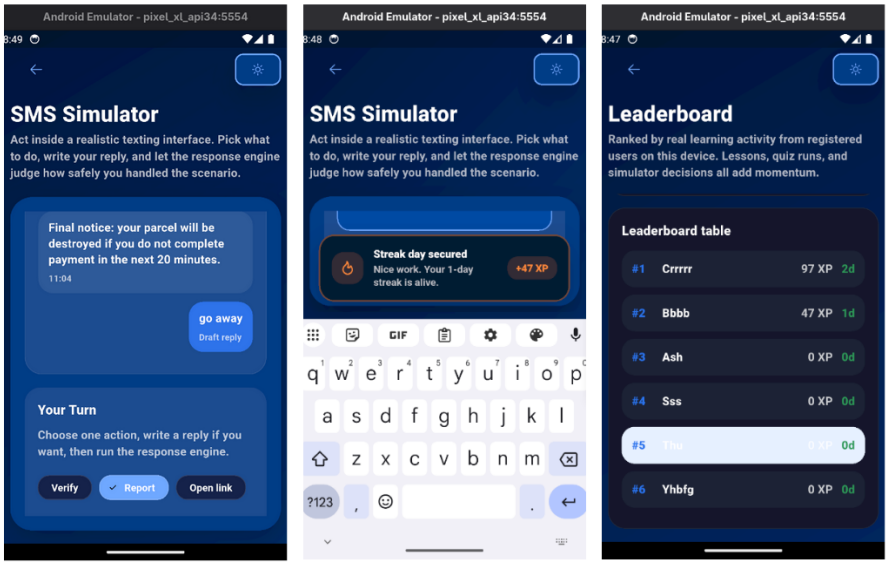

Figure 5.3: Mobile UI Screenshots showing SMS simulator, streaks, and leaderboard.

5.5 Live Updates for Key Words

The live keyword update feature is implemented through the backend keyword briefing flow, where the frontend requests additional information for a selected cybersecurity term and the server responds with a bundled briefing containing an overview, recent vulnerabilities, and related news items. Technically, this is handled through `ScenarioEndpoint.getKeywordBriefing(...)`, which normalises the keyword, generates an appropriate search phrase, and then fetches external data before packaging the result into a typed `KeywordBriefing` model. This allows the application to supplement its static educational content with more current contextual information, making the learning experience feel more relevant and dynamic without requiring the lesson pages themselves to hard-code live content.

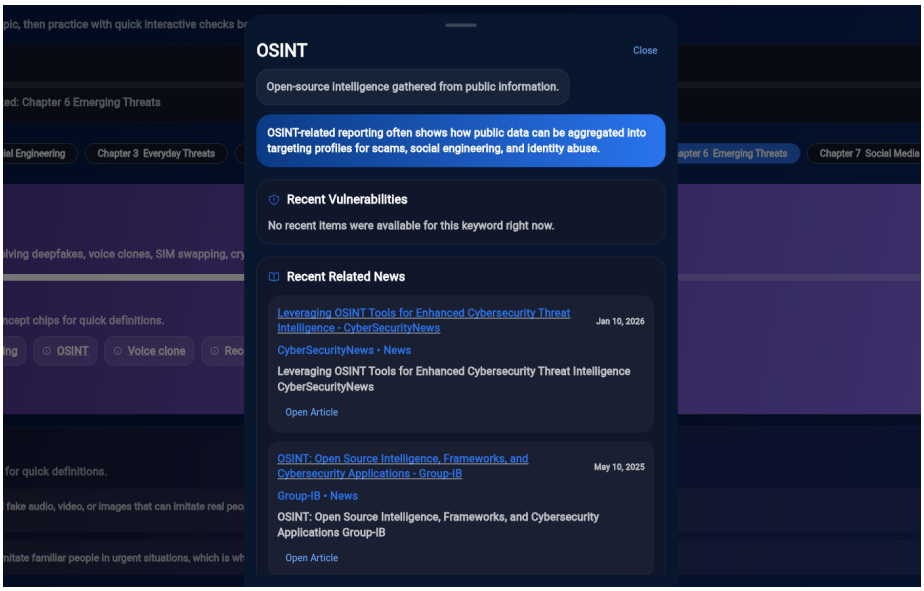

Figure 5.4: Latest Sources about OSINT information (since OSINT is a keyword in Chapter 6)

5.6 Simulators

The simulators share a common technical interaction pipeline built around scenario data, user input capture, evaluation logic, and feedback generation. In each case, the simulator page first loads a predefined scenario model, such as `SimEmail`, `SmsThread`, `SimWifiNetwork`, or `CryptoProject`, which contains the structured content and metadata needed for that activity. The page then collects the user's selected actions and, where relevant, free-text input, before passing that data into an evaluation layer. In the email and SMS simulators, this is handled through dedicated response engines that forward the input into rule-based analyzers, while in the crypto simulator the decision engine performs this evaluation directly within the feature file, and the Wi-Fi simulator uses a more state-driven reveal model centred on the chosen network object. The resulting evaluation object contains fields such as score, verdict, summary, feedback, and recommended next steps, which are then rendered by the UI and can also feed into dashboard activity tracking or backend persistence. The simulator sequence diagram reflects this overall architecture by showing the movement of data from the presentation layer, into the domain scoring layer, and then back into user-facing feedback and stored progress.

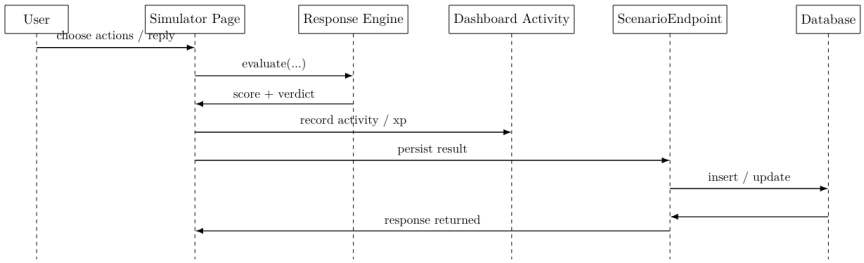

Figure 5.5: Simulator Sequence Diagram

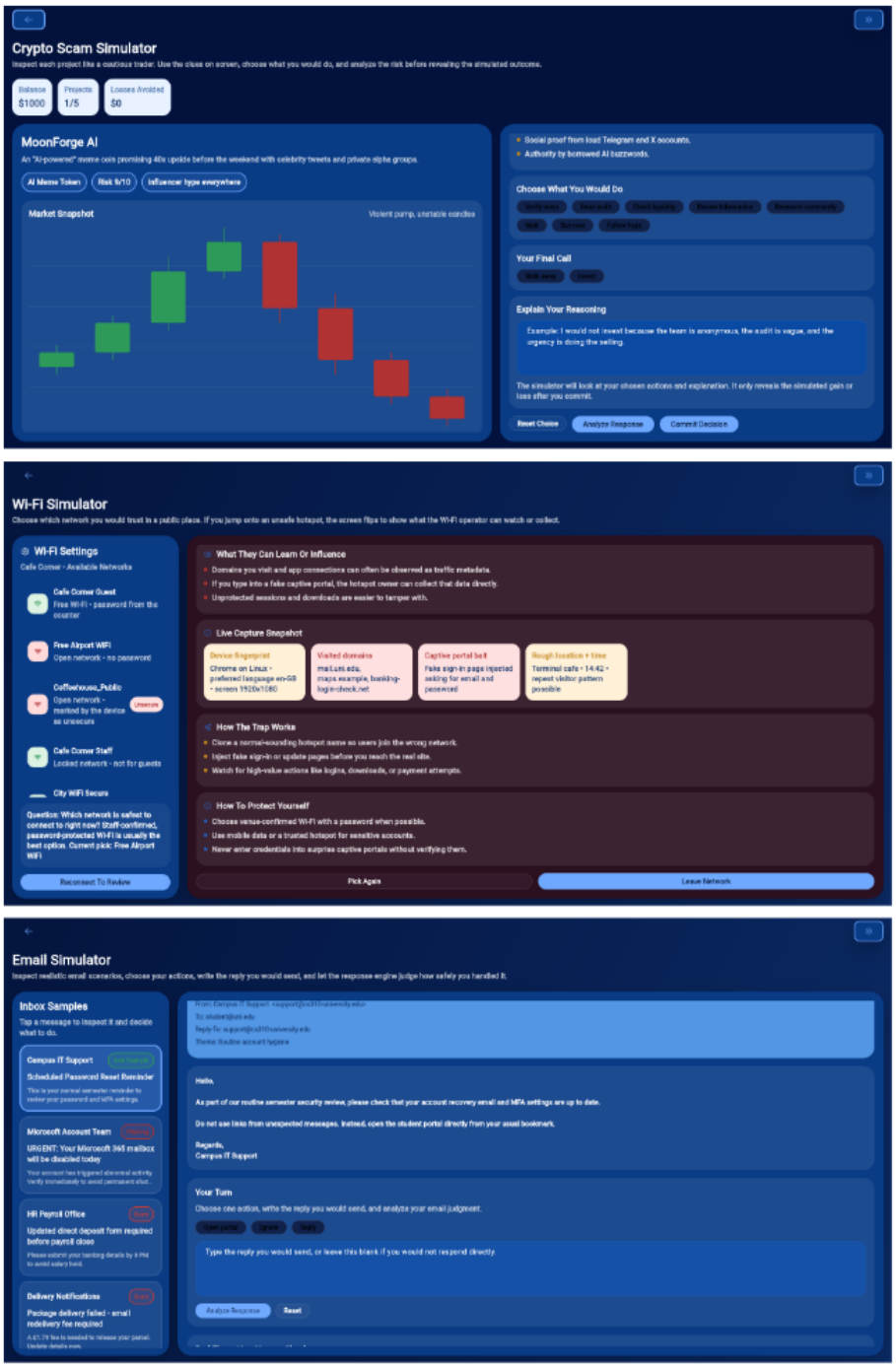

Figure 5.6: Crypto, Wi-Fi, and Email Simulators

5.6.1 Email Simulator Implementation

The email simulator is implemented using a clear separation between scenario data, domain models, and presentation logic. Its scenarios are defined as `SimEmail` objects, each containing message content, sender details, classification metadata, warning flags, suggested actions, and a set of `EmailDecisionOption` objects for the user to choose from. The page layer is responsible for displaying these scenarios in an inbox-style interface and tracking the user's selected response actions and any typed reply text. This makes the email simulator technically well structured, because the simulator content is stored independently from the UI, allowing the page to focus on interaction while the underlying scenario objects provide the data required to drive the experience.

5.6.2 SMS Simulator Implementation

The SMS simulator follows a very similar technical pattern, but it is adapted to a conversation-thread format rather than an inbox-and-detail view. Its scenarios are represented by `SmsThread` objects, which contain thread-level metadata such as contact identity, phone number, scam type, and decision options, as well as a list of `SmsMessage` objects that define the individual message bubbles shown in the interface. This allows the simulator to model a realistic text-message exchange while keeping the scenario content structured and reusable. From a system perspective, this design works well because it keeps the thread data separate from the page code, making the simulator easier to extend with additional threads or message structures later.

5.6.3 Wi-Fi Simulator Implementation

The Wi-Fi simulator is implemented in a more self-contained way inside its page file, with the core data model represented by `SimWifiNetwork` and supporting captured-data objects such as `WifiCapturedDatum`. Each network scenario contains descriptive metadata about trust level, security indicators, exposure risks, attacker behaviour, and safer habits, which the page uses to build both the selection interface and the consequence view. Unlike the email and SMS simulators, this feature relies less on a separate domain-layer pipeline and more on stateful UI transitions, because the main learning effect comes from the user selecting a hotspot and then seeing the consequences unfold visually. Technically, this makes it a more presentation-centric simulator, but still one that is driven by structured scenario

data rather than hard-coded text scattered throughout the widget tree.

5.6.4 Crypto Trading Environment Simulator Implementation

The crypto simulator is implemented as a sequential decision-based environment centred around `CryptoProject`, `InvestmentDecision`, `CryptoDecisionOption`, `CryptoDecisionEvaluation`, and `CandlePoint`. Each `CryptoProject` object contains a large amount of structured scenario information, including social signals, liquidity information, audit status, tokenomics, red flags, safer checks, outcome values, and chart data. The page state uses this information to move the user through a series of investment decisions while keeping track of balance changes and decision history. Compared with the other simulators, the crypto feature is more self-contained and combines its scenario data, decision engine, and state-tracking logic more tightly in one file. This suits the feature technically because it behaves less like a single-message interaction and more like a multi-step simulation of financial judgement over time.

5.7 Response Analyzers Implementation

The analysis functions are implemented as deterministic scoring routines that transform structured user input into evaluation objects containing a score, verdict, summary, and feedback. In the email and SMS simulators, this begins with scenario-specific rule definitions such as `EmailScenarioScoringRule` and `SmsScenarioScoringRule`, which map selected action identifiers to point values and store keyword lists and summary text for each scenario family. The analyzer function then takes the current scenario, the selected actions, and the user's typed reply, normalises the input, applies score changes based on the chosen actions, checks the reply text for safe or risky keywords, and adjusts the score further if the response appears cooperative, contains sensitive information, or contradicts an otherwise safe action. After all adjustments have been applied, the final numeric score is clamped into a bounded range and converted into a verdict band such as safe, at risk, or dangerous. This makes the analyzers technically straightforward to follow and easy to test, because the full decision process is expressed explicitly in code rather than hidden behind opaque model behaviour.

```
1 // each scenario type has its own scoring table so the same
      action can be
2    // judged differently depending on the email being
       simulated.
```

```
    final rule = _emailScoringRules[email.scenarioType]!;
    final normalizedReply = replyText.trim().toLowerCase();

    // safe examples start slightly higher because there is less inherent danger,
    // while scam scenarios begin from a more neutral baseline.
    var rawScore = email.kind == EmailKind.safe ? 60 : 50;
    final goodChoices = <String>[];
    final mistakes = <String>[];
    final feedback = <String>[];

    for (final actionId in actionsSelected) {
      // selected decision chips are translated into point changes using the
      // rule table above, then mirrored into positive or negative feedback.
      final actionScore = rule.actionScores.firstWhere(
        (score) => score.actionId == actionId,
        orElse: () => const EmailActionScore('', 0, ''),
      );
      rawScore += actionScore.points;
      if (actionScore.points > 0 && actionScore.rationale.isNotEmpty) {
        goodChoices.add(actionScore.rationale);
      }
      if (actionScore.points < 0 && actionScore.rationale.isNotEmpty) {
        mistakes.add(actionScore.rationale);
      }
    }
```

Listing 5.3: Portion of Email Response Analyzer Logic

The crypto simulator uses the same overall idea but implements it more directly inside `CryptoDecisionEngine.evaluate(...)` rather than through a separate analyzer file. Here, the function evaluates the chosen trade decision, due-diligence actions, and written rationale against predefined safe and risky signals, then derives a score and final judgement from those combined inputs. Across all of these analysis functions, the technical pattern remains consistent: take structured simulator input, apply rule-based scoring logic, and return a typed evaluation object that the UI can render as educational feedback.

```
  final normalizedRationale = rationale.trim().toLowerCase();
    final feedback = <String>[];
    final nextSteps = <String>{};
    final isSaferProject = project.outcomeIfInvested >= 0;
    var score = isSaferProject ? 58 : 48;

    switch (selectedDecision) {
```

```
      case SimTradeDecision.walkAway:
        if (isSaferProject) {
          score += 6;
          feedback.add(
            'Walking away is conservative. That can still be reasonable when you are unsure, even if the project is not showing obvious scam markers.',
          );
        } else {
          score += 26;
          feedback.add(
            'Avoiding a high-risk project is a strong protective choice when the warning signs outweigh the upside story.',
          );
        }
      case SimTradeDecision.invest:
        if (isSaferProject) {
          score += 22;
          feedback.add(
            'Choosing to invest can be reasonable here because the project shows more verifiable fundamentals than the typical scam setup.',
          );
        } else {
          score -= 28;
          feedback.add(
            'Investing into a project with strong scam markers exposes you to the exact trap the simulator is trying to teach you to catch.',
          );
          nextSteps.add(
            'Pause before buying when the project is selling urgency, hype, or unverifiable trust signals.',
          );
        }
    }
```

Listing 5.4: Portion of Crypto Analyzer Logic

5.8 Database and Backend

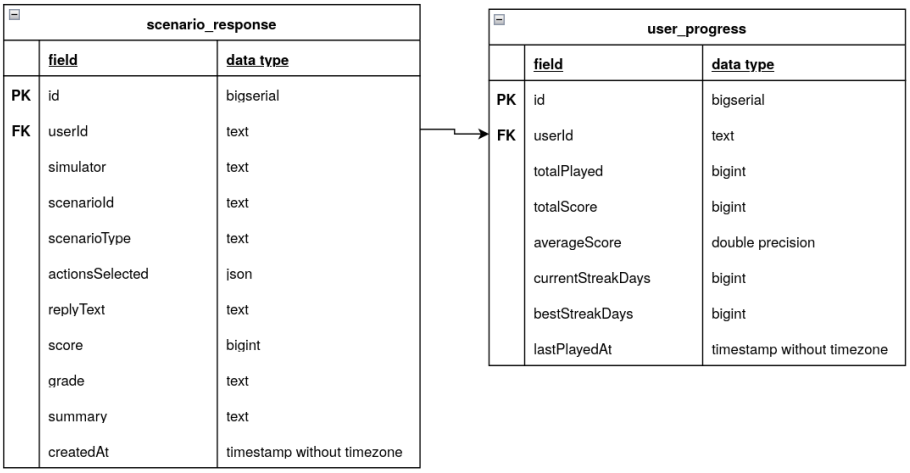

Figure 5.7: Database Schema

The main application-specific schema is centred around the relationship between `scenario_response` and `user_progress`, as these two tables capture the most important backend data produced by the system's simulator features. `scenario_response` stores individual simulator submissions, including the user identifier, simulator name, scenario identifier, selected actions, reply text, score, grade, summary, and creation time. This makes it the detailed history table, preserving each discrete interaction as a standalone record. `user_progress`, by contrast, acts as an aggregate table. Rather than storing every event individually, it keeps rolling summary values for each user, including total sessions played, total score, average score, current streak length, best streak length, and last played time. Technically, the relationship between them is logical rather than enforced as a direct foreign-key join: both tables use `userId`, and responses from `scenario_response` are rolled up into the matching `user_progress` row whenever new activity is recorded. This allows the backend to support both detailed history and efficient summary retrieval without recalculating everything on every request.

This is the main schema of the application because it directly supports the custom educational functionality that distinguishes the project from the underlying framework. Although the database also contains many additional tables created by Serverpod and its authentication modules, those largely provide framework-level infrastructure such as session handling, refresh tokens, email account man-

agement, profiles, migration tracking, and logging. These tables are important to the backend as a whole, but they are not the central schema from the perspective of the application's own learning features. By contrast, `scenario_response` and `user_progress` are the schema elements that represent the project's domain logic, namely tracking simulator behaviour and measuring a user's progress over time. For that reason, they are the most important schema to discuss when explaining what the backend does technically and how it supports the core aims of the app.

6 Testing

This section outlines the range of testing approaches applied during development to verify correct system behaviour.

Table 6.1: Classification of Automated Tests

Test File	**Classification**	**Purpose**
`test/new_suite/auth/auth_controller_test.dart`	Unit test	Tests local authentication logic such as registration, login validation, username normalisation, and sign-out behaviour in isolation.
`test/new_suite/analyzers/email_response_engine_test.dart`	Unit test	Verifies that the email simulator scoring engine produces the expected verdicts and scores for safe and unsafe phishing responses.
`test/new_suite/analyzers/sms_response_engine_test.dart`	Unit test	Verifies that the sms simulator scoring engine correctly evaluates safe, unsafe, and mixed-response scenarios.
`test/new_suite/social/dashboard_social_activity_test.dart`	Integration-style test	Tests dashboard social activity logic together with local persistence through `SharedPreferences`.
`test/new_suite/contracts/server_endpoint_contract_test.dart`	Contract test	Checks that the frontend-facing backend contract still exposes the expected scenario endpoint methods and related structures.

Test File	Classification	Purpose
`test/new_suite/contracts/server_database_contract_test.dart`	Contract test	Verifies that the server migration and schema definition files still contain the expected application tables, columns, and indexes.
`server/test/database_contract_test.dart`	Contract test	Confirms that the backend migration registry and migration sql files define the expected database structures.
`server/test/endpoint_contract_test.dart`	Contract test	Verifies that the backend endpoint implementation and dispatch wiring still expose the required scenario methods.
`server/test/protocol_contract_test.dart`	Protocol contract test	Verifies serialization behaviour and confirms that generated protocol models remain registered and consistent.

6.1 UI Testing

UI testing is important in this project because the application is heavily interaction-driven and much of its educational value depends on how clearly information is presented to the user. Features such as lessons, quizzes, and simulators are not only judged by whether the underlying logic works, but also by whether the interface loads correctly, routes properly, and presents the intended structure to the learner. In a cybersecurity education app, realism and clarity are especially important, since confusing layouts or broken screens would directly reduce the effectiveness of the learning experience.

For that reason, UI testing is justified as a way of confirming that the application shell and key interface flows behave as expected from the user's point of view.

The UI testing procedure involved functional testing, visual testing, as well as usability testing. The following checklist outlines the aspects that were tested in detail.

Table 6.2: UI Testing Checklist

Category	**Checklist Item**	**Status**
Functional Testing	Dashboard, lessons, quizzes, and simulator pages load correctly through the app router.	✓
Functional Testing	Protected routes redirect appropriately depending on authentication state.	✓
Functional Testing	Core UI interactions such as buttons, cards, chips, and navigation actions behave as intended.	✓
Functional Testing	Simulator interfaces accept user input and return feedback, scores, and verdicts correctly.	✓
Functional Testing	Theme switching and app boot behaviour function correctly within the application shell.	✓
Visual Testing	Overall visual design is polished, modern, and appropriate for an educational cybersecurity application.	✓
Visual Testing	Layout and alignment of major UI elements are consistent across pages.	✓
Visual Testing	Colour palette is coherent and consistent between dashboard, lessons, quizzes, and simulators.	✓
Visual Testing	Typography and font hierarchy clearly separate titles, body content, warnings, and feedback.	✓
Visual Testing	Icons and graphical elements support comprehension without overwhelming the interface.	✓

Category	Checklist Item	Status
Visual Testing	UI component styling is consistent across inputs, cards, buttons, chips, and feedback panels.	✓
Visual Testing	Responsive layouts adapt well across wide and narrow screen sizes, especially in simulator pages.	✓
Visual Testing	The visual experience remains consistent across the supported Flutter targets.	✓
Usability Testing	Navigation is straightforward, with the dashboard acting as a clear central hub.	✓
Usability Testing	Instructions and feedback are understandable in lessons, quizzes, and simulator activities.	✓
Usability Testing	Users are guided clearly through task completion without unnecessary confusion.	✓
Usability Testing	Progress indicators such as xp, streaks, and leaderboard data help orient and motivate the user.	✓
Usability Testing	Simulator layouts resemble real-world contexts closely enough to support intuitive use.	✓
Usability Testing	The app balances realism with clarity, making cybersecurity learning approachable rather than intimidating.	✓

The results of these tests show that the application demonstrates a very strong user interface overall. The UI is not only visually consistent and functional, but also well suited to the educational purpose of the project. Its layouts, navigation structure, simulator realism, and feedback mechanisms work together effectively to create an experience that is clear, engaging, and easy to use. Taken together, these findings suggest that the app has a very good UI, with a level of polish and usability that supports both the learning aims of the project and its potential as a real-world product with further refinement. Additionally, a focus group was informally surveyed (with open ended questions) to test the UI as well, and they were satisfied with the performance of the application overall.

6.2 Unit Testing

The significance of unit testing to this project can be highlighted by noting that several core features rely on isolated logic that must behave consistently and predictably. This is especially true for the simulator response engines, analyzers, authentication controller, and related domain-level functionality, where small logic errors could lead to incorrect scoring, misleading feedback, or broken user flows. Since the project is educational, the correctness of these components matters more than it would in a purely cosmetic feature, because the user is expected to learn from the outcomes they receive. Unit testing is therefore justified as a way of verifying that the most important pieces of business logic work correctly in isolation before being relied on by the wider application.

As outlined in table 6.1, three separate, automated unit tests were conducted. Key, isolable, testable metrics such as analyser response accuracy, as well as other core functionality like the local authetntication logic were the clear and obvious candidates for unit testing.

6.2.1 Analyser Unit Testing

```
group('EmailResponseEngine', () {
  SimEmail findEmail(String id) =>
      simEmails.firstWhere((email) => email.id == id);

  test('rewards reporting and separate verification for
      phishing email', () {
    final email = findEmail('microsoft_phish');

    final result = EmailResponseEngine.evaluate(
      email: email,
      reply: 'This looks suspicious. I will report it and
          open the real site from my bookmark.',
      selectedActionIds: {'open_real_site', 'report_delete'
          },
    );

    expect(result.score, greaterThanOrEqualTo(70));
    expect(
      result.verdict,
      anyOf(equals('Excellent judgment'), equals('Good
          response')),
    );
    expect(result.goodChoices, isNotEmpty);
  });

});
```

Listing 6.1: Example of Email Response Engine Test

The tests for `EmailResponseEngine` and `SmsResponseEngine` are implemented as isolated unit tests that validate the behaviour of the simulator scoring layer against predefined scenarios. In each case, a known scenario object is loaded from `email_sim_data.dart` or `sms_sim_data.dart`, then passed into `EmailResponseEngine.evaluate(...)` or `SmsResponseEngine.evaluate(...)` together with a user reply string and a set of selected action identifiers. The returned evaluation object is then checked using assertions on fields such as `score`, `verdict`, `goodChoices`, `mistakes`, and `redFlagsFound`. This means the tests are not simply checking whether the method runs, but whether the rule-based analyzer produces the correct classification and feedback for both safe and unsafe user behaviour. Technically, this is important because these engines act as the main decision layer behind the email and sms simulators.

6.2.2 Authentication Logic Unit Testing

The `AuthController` tests verify the core local authentication logic in isolation. They check that methods such as `register(...)`, `login(...)`, and `signOut()` update the authentication state correctly, persist account data through `SharedPreferences`, and reject invalid credentials when expected. This is important because `AuthController` is responsible for the app's login state and protected-route access behaviour.

6.3 Contract Testing

These tests are done since the system includes a Flutter client, generated protocol models, and a Serverpod backend that must remain aligned with one another. If the client expects one endpoint structure or model shape while the backend exposes another, features may fail even when both sides appear individually correct. This is particularly relevant here because the project uses custom endpoints, generated client code, migrations, and typed protocol classes. Contract testings thus serves as a way of checking that these boundaries remain consistent, helping to prevent integration issues caused by changes in backend structure, model definitions, or endpoint wiring.

```
1 import 'dart:io';
2
3 import 'package:test/test.dart';
```

```dart

void main() {
  group('Scenario endpoint contract', () {
    test('endpoint persists responses and updates aggregates', () async {
      final source = File('lib/src/endpoints/scenario_endpoint.dart');
      expect(await source.exists(), isTrue);

      final content = await source.readAsString();
      expect(content, contains('class ScenarioEndpoint extends Endpoint'));
      expect(content, contains('ScenarioResponse.db.insertRow'));
      expect(content, contains('UserProgress.db.findFirstRow'));
      expect(content, contains('UserProgress.db.updateRow'));
      expect(content,
          contains('Future<List<ScenarioResponse>> listRecentResponses'));
    });

    test('dispatch registry exposes the scenario endpoint methods', () async {
      final dispatch = File('lib/src/endpoints.dart');
      expect(await dispatch.exists(), isTrue);

      final content = await dispatch.readAsString();
      expect(content, contains("'scenario': _i4.ScenarioEndpoint()"));
      expect(content, contains("'getKeywordBriefing'"));
      expect(content, contains("'analyzeResponse'"));
      expect(content, contains("'getUserProgress'"));
      expect(content, contains("'listRecentResponses'"));
    });
  });
}
```

Listing 6.2: Endpoint Contract Tests

The contract tests are used to verify that key backend-facing structures remain consistent with what the rest of the system expects. In the case of `endpoint_contract_test.dart`, the test reads the source files directly and checks that the `ScenarioEndpoint` still exposes the required methods and persistence calls, such as `analyzeResponse(...)`, `getUserProgress()`, and `listRecentResponses(...)`, as well as the expected database interactions like `ScenarioResponse.db.insertRow` and `UserProgress.db.updateRow`. This helps confirm that the backend contract has not drifted away from the assumptions made elsewhere in the application.

More broadly, the other contract tests follow the same principle. Rather than testing business logic outputs in isolation, they verify that important structural contracts still hold between files, generated models, migrations, and backend definitions. For example, the database contract tests confirm that the expected tables, indexes, and schema fields still exist, while the protocol contract test checks that generated models still serialize correctly and remain registered in the protocol layer. Together, these tests provide confidence that the boundaries between the frontend, backend, and persistence layers remain stable even as the codebase evolves.

6.4 Integration Testing

These tests are crucial for this project because many features depend on multiple parts of the system working together rather than on isolated logic alone. Examples include local persistence with SharedPreferences, progress tracking, social dashboard updates, and the wider interaction between frontend state, stored data, and backend-supported features. Because the application combines UI behaviour, domain logic, local storage, and server-side components, there is a real risk that each part may work in isolation while the full feature still behaves incorrectly when combined. Integration testing ensures that these connected parts function together as intended, which is especially important in a project with both client-side and backend elements.

```
1  test('records XP and creates a streak day for the current
       user', () async {
2    final award = await DashboardSocialActivity.
         recordCurrentUserActivity(
3      type: UserActivityType.quizCompletion,
4      activityId: 'quiz:basics',
5      xp: 60,
6    );
7    final snapshot = await DashboardSocialActivity.
         loadSnapshot();
8
9    expect(award, isNotNull);
10   expect(award!.tracked, isTrue);
11   expect(award.unlockedStreakDay, isTrue);
12   expect(snapshot.currentStreakDays, 1);
13   expect(snapshot.todayXp, 60);
14   expect(snapshot.weeklyXp, 60);
15   expect(snapshot.completedToday, isTrue);
16   expect(snapshot.leaderboard.first.isCurrentUser, isTrue)
         ;
17 });
18
```

```
19  test('deduplicates the same activity on the same day', ()
        async {
20    final first = await DashboardSocialActivity.
          recordCurrentUserActivity(
21      type: UserActivityType.simulatorDecision,
22      activityId: 'sms:delivery_fee',
23      xp: 50,
24    );
25    final second = await DashboardSocialActivity.
          recordCurrentUserActivity(
26      type: UserActivityType.simulatorDecision,
27      activityId: 'sms:delivery_fee',
28      xp: 50,
29    );
30    final snapshot = await DashboardSocialActivity.
          loadSnapshot();
31
32    expect(first!.tracked, isTrue);
33    expect(second!.tracked, isFalse);
34    expect(snapshot.todayXp, 50);
35    expect(snapshot.weeklyXp, 50);
36  });
```

Listing 6.3: Dashboard Social Activity Tests

The dashboard social activity tests verify that progress-related behaviour is persisted and aggregated correctly when a user completes learning actions. In the first example, `DashboardSocialActivity.recordCurrentUserActivity(...)` is called with a quiz completion event and an xp value, after which `loadSnapshot()` is used to confirm that the stored dashboard state reflects the change correctly. The assertions check that the activity was tracked, that a streak day was unlocked, and that the resulting snapshot updates fields such as `currentStreakDays`, `todayXp`, `weeklyXp`, `completedToday`, and the leaderboard state. In the second example, the same simulator activity is recorded twice on the same day, and the test verifies that the first event is accepted while the second is rejected as a duplicate. This confirms that the dashboard logic does not incorrectly award repeated xp for the same event and that the aggregated progress totals remain accurate. Together, these tests act as integration-style checks because they validate the interaction between activity tracking logic, local persistence, and the derived dashboard snapshot used by the user interface.

7 Evaluation

This section outlines a systematic evaluation of the requirements and risks associated with the project post development.

7.1 Evaluation of Requirements

Overall, most of the project requirements were met successfully. The main functional areas, including authentication, lessons, quizzes, simulators, and progress-related features, were implemented to a solid standard, and the wider non-functional expectations such as usability, structure, and cross-platform intent were also considered throughout development. While some areas remain more complete than others, the project as a whole satisfies the majority of the requirements set out at the beginning.

7.1.1 Evaluation of Functional Requirements

Table 7.1: Functional Requirements Checklist

ID	Status
FR1	✓
FR2	✓
FR3	✓
FR4	✓
FR5	✓
FR6	✓
FR7	✓
FR8	✓
FR9	✓
FR10	✓
FR11	✓
FR12	✓
FR13	✓
FR14	✗
FR15	✗
FR16	✗
FR17	✗
FR18	✗
FR19	✗

As illustrated above, most of the core functional requirements were met during development. The application supports the main user flows expected of it, including account access, navigation to protected pages, educational content delivery, simulator interaction,

quiz participation, and progress feedback. Although some backend-backed behaviour is not yet fully complete across every feature, the core system functionality is present and working.

7.1.2 Evaluation of Non Functional Requirements

Table 7.2: Non Functional Requirements Checklist

ID	Status
NFR1	✓
NFR2	✓
NFR3	✓
NFR4	✓
NFR5	✓
NFR6	✓
NFR7	✓
NFR8	✓
NFR9	-
NFR10	✓
NFR11	✓
NFR12	✓
NFR13	✓
NFR14	✗
NFR15	✗
NFR16	✗
NFR17	✗

Most of the non-functional requirements were also addressed well throughout the project. Consideration was given to usability, layout clarity, responsiveness, maintainability, and consistency of design, particularly because the app is intended to function as an educational tool rather than just a technical prototype. Some platform-specific limitations remain, but the overall quality expectations were taken into account during implementation. Note: NFR9 was neither successful, nor a failure, since the codebase has moderate scalability. As mentioned in NFR16, it is acknowledged that scalability is a key contributing factor to the soundness of this application and improving the codebase's scalability is a serious consideration for future work.

7.2 Evaluation of Risks

Overall, the major risks associated with the project were identified early and managed reasonably well throughout development. Consideration was given to technical, managerial, and ethical risk areas

rather than treating them as afterthoughts. While not every risk could be removed entirely, most were mitigated effectively enough to prevent them from seriously undermining the project.

7.2.1 Technical Risk Evaluation

Table 7.3: Analysis of Technical Risks and Mitigations

ID	**Analysis**
TR1	This risk was identified early on, and clear documentation on running the code has been produced, rendering this issue virtually solved.
TR2	Same solution and analysis as TR1.
TR3	The mitigation has been applied successfully, and the risk factor posed by this particular risk has thus been significantly lowered.
TR4	Same solution and analysis as TR1.
TR5	This can only be solved in future iterations. Can be considered a failed mitigation.
TR6	Accuracy of the analysis function can be improved through methods involving machine learning. This is a high priority task for future iterations, but for now, the risk of users getting provided highly inaccurate responses is naught.
TR7	Risk has been managed and mitigated well, fail safe features were appropriately implemented.
TR8	Partially unsuccessful, as only defensive parsing was implemented, however, it is to be noted that the likelihood of this is low.
TR9	Mitigation plan successfully executed.
TR10	Mitigation plan successfully executed.

Most of the technical risks were handled well during the development process. Issues around backend setup, configuration, local persistence, and platform-specific behaviour did arise, but they were generally manageable and did not prevent the project from progressing. In most cases, the risks were either reduced through practical solutions or clearly understood and documented.

7.2.2 Managerial Risk Evaluation

Table 7.4: Analysis of Managerial Risks and Mitigations

ID	Analysis
MR1	Developer motivation hinged on delivering a complete application, albeit not intensely feature rich application, and not a series of incomplete features. Risk successfully mitigated.
MR2	The project management strategy that was employed help take care of this risk quite effectively. Further discussion is provided in the project management report.
MR3	The incremental development strategy has helped mitigate this risk quite well.
MR4	Supervisor meetings were helpful in mitigating this risk.
MR5	Risk was managed moderately well, it does appear as though more focus was given to the UI, but the app is complete enough to justify such focus.
MR6	Same solution and analysis as MR2.
MR7	Same solution and analysis as MR4.
MR8	Priority-based testing proved to be highly effective to mitigate, and even eliminate this risk.
MR9	Mitigation strategy was applied appropriately.

Managerial risks were also considered throughout the project, particularly in relation to scope control, prioritisation, and maintaining focus on the core deliverables. Because the project had several possible directions it could have taken, there was a real risk of becoming sidetracked. However, this was largely mitigated by keeping attention on the key requirements and continuing to evaluate progress against the overall aims of the project.

7.2.3 Ethical Risk Evaluation

Table 7.5: Analysis of Ethical Risks and Mitigations

ID	**Analysis**
ER1	The app endorses positive reinforcement, and gives the user a sense of control over the situation. This boosts morale, and does the opposite of increasing distress.
ER2	The likelihood of this occuring is too low to warrant serious deliberation into sophisticated mitigation techniques, however, the app has enough disclaimers to dissuade such behaviour.
ER3	Moderately succesful implemenatation of mitigation. App will be made more secure in future iterations.
ER4	Storage of personal data is minimised and passwords are salted and hashed. Succesful mitigation.
ER5	A survey of focus group testers indicated that this was not a concern. The risk is therefore considered fully mitigated.
ER6	The content of the app, and transparency to the user as to what their expectation of the app should be, significantly lowered the risk of this occurring.
ER7	Only trusted materials have been sourced for the news feed. The burden is now on the user to not misinterpret very obvious and clear information. Given that they will have completed some training from the app before then, it is likely that they will come to sensible conclusions from the articles and documents.
ER8	Moderately successful. Full cross platform integration (especially backend) is part of future work.
ER9	Same solution and analysis as ER5.

Ethical risks were taken seriously throughout the development process. Given that the project deals with cybersecurity scenarios, simulated scams, and user-related data, it was important to consider how content was presented and how the system could affect users. These risks were not ignored, and reasonable care was taken to ensure that the application remained educational, responsible, and appropriate in its overall design.

8 Future Work

The clearest area for future work is the completion of backend integration across the full system. At present, some features already interact with the Serverpod backend, but major areas such as authentication, lesson progress, and dashboard social data still rely primarily on local storage. A logical next step would therefore be to migrate these features to backend-backed persistence so that users can experience consistent progress and account state across devices and platforms.

A second area for future development would be the extension and refinement of the learning content itself. The lessons, quizzes, and simulators already provide a solid foundation, but the project could be expanded with additional topic coverage, more scenario variation, and a broader range of question banks.

Finally, the biggest drawback of the current version of the app is the performance and accuracy of the response analyzers. In future, a machine learning model could be trained to evaluate responses, which would result in a significantly more accurate and insightful response generation, leading to increased educational value provided by the app.

References

[1] M. Bada, A. M. Sasse, and J. R. Nurse. Cyber security awareness campaigns: Why do they fail to change behaviour? https://arxiv.org/pdf/1901.02672, 2019.

[2] M. Merritt, S. Hansche, D. B. Ellis, K. Sanchez-Cherry, J. N. Snyder, and D. Walden. Building a cybersecurity and privacy learning program: NIST special publication 800-50r1. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-50r1.pdf, September 2024.

[3] National Cyber Security Centre. NCSC annual review 2025. https://www.ncsc.gov.uk/sites/default/files/documents/ncsc-annual-review-2025.pdf, October 2025.

[4] Ofcom. Online nation report 2025. https://www.ofcom.org.uk/siteassets/resources/documents/research-and-data/online-research/online-nation/2025/online-nations-report-2025.pdf, December 2025.

[5] S. Rizvi and E. Fordham. Cyber security breaches survey 2025. https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2025, April 2025.

[6] Threat Research Team. Avast Q1/2024 threat report. https://www.gendigital.com/blog/insights/reports/avast-q1-2024-threat-report, May 2024.

[7] UK Finance. Annual fraud report 2025. https://www.ukfinance.org.uk/system/files/2025-05/UK%20Finance%20Annual%20Fraud%20report%202025.pdf, 2025.