Software Requirements Specification

For

Axamise

Department of Electrical and Computer Engineering

North South University

Prepared By: Group 10:

Name	Student ID
Adnan Abdullah	2221345642
Kinga Tshering	2233710042
Md Shihabul Alam Shihab	2233115642
Monjurul Islam	2232339042

Version: 1.0
Created on: 2nd November, 2025

Table of Contents

Software Requirements Specification

Axamise Project

Version :

Revision History

Date	Version	Description	Author
02/11/2025	1.0	SRS 1.0	Contributed by all the group members.
–/–/–	2.0	SRS 2.0	Group-1
–/–/–	3.0	SRS 3.0	Group-1
–/–/–	4.0	SRS 4.0	Group-1

1. Introduction

The **Software Requirements Specification (SRS)** defines the complete set of functional and non-functional requirements for the **Axamise Competitive Programming Platform**, an intelligent, AI-augmented environment designed to elevate programming proficiency through

interactive problem-solving and global competitions.
Axamise combines the precision of automated evaluation, the accessibility of a web-based IDE, and the engagement of competitive ranking systems. It aims to support individual learning, peer comparison, and large-scale contest management, forming a seamless bridge between practice and performance.

The document outlines **technical specifications**, **user expectations**, and **system behaviors**, ensuring that developers, designers, and stakeholders share a unified understanding of the project's vision and deliverables

1.1.Purpose

The core purpose of this application is to serve as a comprehensive **competitive programming and skill development platform**.

User-Focused Objective

The primary purpose of Axamise is to create a **comprehensive, interactive, and scalable competitive programming platform** that supports both self-paced learning and structured contests.

- Provide diverse types of coding challenges covering various algorithms, data structures, and problem complexities.
- Offer an integrated IDE for writing, compiling, and testing code directly within a browser, eliminating external dependencies.
- Establish a transparent and dynamic **global ranking system** that motivates users by showcasing performance metrics and worldwide leaderboards.
- Ensure continuous learning by offering real-time performance feedback and analytics.

Engineering and Quality Objectives

- Develop a **high-performance and modular architecture**, enabling fast iterations and easy scalability.

- Maintain robust **fault tolerance**, ensuring uninterrupted contest experiences even under heavy load.
- Use **secure, modern technologies** (React, Firebase, Docker) to ensure code isolation, data integrity, and user trust.
- Incorporate **AI-driven evaluation** via Gemini 2.5 Pro to intelligently assess solution quality, efficiency, and clarity.
-

1.2.Scope

The **Axamise project** encompasses the end-to-end design and deployment of a web-based competitive programming ecosystem.

Inclusions

- A feature-rich web-based IDE supporting compilation, execution, and testing.
- A sandboxed execution engine with automated validation against hidden test cases.
- A **proprietary global ranking and scoring system** to foster competitive learning.
- Integrated AI evaluation for code correctness, efficiency, and style.
- A role-based system for learners, competitors, and administrators.
- Cloud-based infrastructure ensuring 24/7 availability and seamless updates.

Exclusions

- Real-time code collaboration or live debugging.
- Financial transactions, paid events, or premium memberships.
- In-person proctoring or webcam monitoring for contests.

System Boundaries

Axamise interacts with the user through a browser-based interface. All code execution and scoring occur on secure remote servers through REST APIs. The front-end handles data

visualization and interaction, while the backend processes submissions, evaluates code, and updates leaderboards.

1.3. Definitions, Acronyms, and Abbreviations

Term	Definition
IDE	Integrated Development Environment – a browser-based coding interface.
CES	Code Execution Service – a secure, containerized sandbox for executing user-submitted code.
AI	Artificial Intelligence – Gemini 2.5 Pro model used for evaluation.
WCAG	Web Content Accessibility Guidelines – ensures accessibility for all users.
UID	Unique Identifier assigned to every user and submission.
DB	Firestore NoSQL database for real-time data persistence.
TLE/MLE	Time Limit Exceeded / Memory Limit Exceeded – performance constraints.
UI	User Interface of the Axamise web application.
CM	Configuration Management – the process of maintaining code repositories (Git).

1.4. References

The references are:

- Axamise Structural Model
- Axamise Behavioural Model
- Axamise Non-Functional Requirement Model
- Vision Document (Draft 5)
- Project Plan, Traceability Matrix, and Test Strategy

1.5. Overview

The **Axamise** project is the specification for a global competitive programming platform designed to significantly enhance the programming proficiency of users. This application provides a modern, integrated environment where coders can access diverse coding challenges, write solutions using a web-based IDE, and immediately test their code against comprehensive, automated test cases. Beyond skill development, Axamise establishes a competitive arena by featuring a real-time, worldwide ranking system. The technical foundation prioritizes **smooth performance, scalability, and maintainability** to ensure a high-quality, long-lasting product experience for its global user base.

2. Overall Description

2.1. Product Perspective and General Functions

Axamise is a **self-contained, web-hosted ecosystem** that brings together multiple services under a single unified framework. Its architecture integrates the following components:

- **Front-End Interface:** Built with React and TailwindCSS, providing responsive and accessible navigation across devices.
- **Back-End Services:** TypeScript-based factory pattern for modular functionality (authentication, events, evaluation).
- **Database Layer:** Real-time Firestore database ensuring data persistence and synchronization.

- **AI Evaluation Layer:** Gemini 2.5 Pro API integration for code quality and correctness analysis.
- **Execution Engine:** Docker-based sandbox (CES) ensuring isolated and secure code compilation.

This combination makes Axamise an intelligent, scalable, and adaptive system capable of supporting thousands of simultaneous users.

2.1.1. System Summary

Axamise acts as both a skill development tool and a standardised competitive arena. The platform's functional focus is categorised below:

Functional Area	Purpose and Key Capabilities
Integrated Coding Environment	Provides users with an in-browser IDE supporting multiple languages (starting with C++). Offers live syntax highlighting, auto-save, and quick execution of custom test inputs.
Automated Testing Engine	Validates code against hidden test cases within isolated Docker containers. Enforces TLE/MLE constraints for fairness.
AI Evaluation Engine	Uses Gemini AI to evaluate correctness, efficiency, and readability, generating intelligent feedback for continuous improvement.
Competitive Scoring & Ranking	Maintains dynamic global rankings based on problem-solving accuracy, execution time, and overall efficiency.

User & Content Management	Enables user registration, authentication, and role-based access (Learner/Competitor/Admin). Facilitates question approval and event management for administrators.

2.2. User Classes and Characteristics

The **Axamise** platform is built to cater to a diverse range of users, each with distinct needs and expectations from the system. The application must accommodate three primary classes of users:

2.2.1. Learner/Novice Programmers

- **Description:** Beginners learning to code or new to competitive programming.
- **Needs:** Simple challenges, clear problem statements, visual feedback, and progress tracking.
- **Access:** Practice area, personal history, submission review, and leaderboard viewing.
- **Goal:** Develop algorithmic thinking in a non-intimidating environment.

2.2.2. Competitive/Advanced Programmers

- **Description:** Skilled users seeking advanced problems and global recognition.
- **Needs:** Complex challenges, high-speed evaluation, performance insights, and detailed analytics.
- **Access:** Full access to contests, global leaderboards, and timed tournaments.
- **Goal:** Compete globally while tracking personal improvement through detailed metrics.

2.2.3. Platform Administrators and Content Managers

- **Description:** Internal users who manage platform health, challenges, and competition logistics.
- **Needs:** Admin portal with secure logins, approval dashboards, performance monitoring, and notification management.

- **Access:** Elevated permissions for event scheduling, question moderation, and system configuration.
- **Goal:** Maintain integrity, fairness, and content quality across the Axamise ecosystem

2.3. Operating Environment

The Axamise application is designed to operate primarily as a **cloud-hosted web service** accessible via standard internet browsers.

- **Platform Type:** Cloud-hosted web service accessible via standard browsers (Chrome, Firefox, Edge, Safari).
- **Supported Devices:** Responsive interface supporting desktop, tablet, and mobile use.
- **Execution Environment:** Dedicated, containerized (Docker) sandbox ensuring safe code compilation and isolation.
- **Database:** Firestore — real-time, scalable, and distributed database.
- **Hosting:** Firebase Hosting / Cloud Run ensuring minimal latency and high uptime.
- **Dependencies:** Node.js, React, Firebase SDK, Gemini AI API, TailwindCSS, Framer Motion.
- **Resilience:** Automatic scaling based on user load and redundancy for fault tolerance.

3. Specific Requirements

The following section defines the **functional**, **interface**, and **non-functional** requirements for the Axamise platform. Each requirement is designed to ensure **traceability**, **testability**, and **alignment** with the system's business goals.

The system architecture supports modular development, where each feature — from authentication to ranking — functions as an independent, testable unit integrated through REST APIs and global state management.

3.1. Functionality

The Axamise platform offers all major functions needed for a **complete competitive programming ecosystem**, including authentication, integrated coding, automated evaluation, and leaderboard management.

The system follows an MVC-style architecture where user interactions (controllers) trigger specific service modules (factories) that execute business logic and return structured results to the UI layer.

3.1.1. User Authentication and Profile Management

- 3.1.1.1. The system shall allow users to register an account using a valid email and password.
- 3.1.1.2. The system shall allow registered users to log in and maintain a secure, authenticated session
- 3.1.1.3. The system shall display a dedicated user profile page containing submission history and performance metrics.
- 3.1.1.4. The system shall enable the user to securely update their display name, password, and optional demographic data (e.g., country).
- 3.1.1.5. The system shall display the user's current global rank and historical ranking progression on their profile.

Rationale:
Authentication forms the foundation for personalized learning and global competition, ensuring data security while enabling user-specific analytics.

3.1.2. Integrated Coding Environment (IDE)

- **3.1.2.1. The system shall support code input and execution in C++ for now.**
- **3.1.2.2. The system shall provide syntax highlighting and basic code formatting features within the IDE for all supported languages.**
- **3.1.2.3. The system shall allow the user to input custom test data and execute their code against this input before official submission.**
- **3.1.2.4. The system shall display the output and any standard error messages (e.g., compilation failure) immediately following local execution.**
- **3.1.2.5. The system shall automatically save the user's current code draft locally in the browser.**

  Rationale:

  The IDE eliminates environment setup friction, allowing users to focus on logic and performance optimization. Its modular design supports integration with AI-based code assistants in future releases.

3.1.3. Submission and Automated Testing

- 3.1.3.1. The system **shall** allow the user to submit their code solution to the contest engine for official scoring.
- 3.1.3.2. The system **shall** execute the submitted code in an **isolated sandbox environment**.
- 3.1.3.3. The system **shall** test the submission against a hidden, comprehensive suite of test cases.
- 3.1.3.4. The system **shall** enforce predefined **Time Limit Exceeded (TLE)** and **Memory Limit Exceeded (MLE)** constraints on all submissions.
- 3.1.3.5. The system **shall** provide the user with a submission result status, indicating: **Accepted**, **Wrong Answer**, **Runtime Error**, **TLE**, or **MLE**.

  Rationale:

  Automated testing and AI grading provide consistent and fair evaluations at scale. It reduces manual checking and offers deep insights into coding behavior.

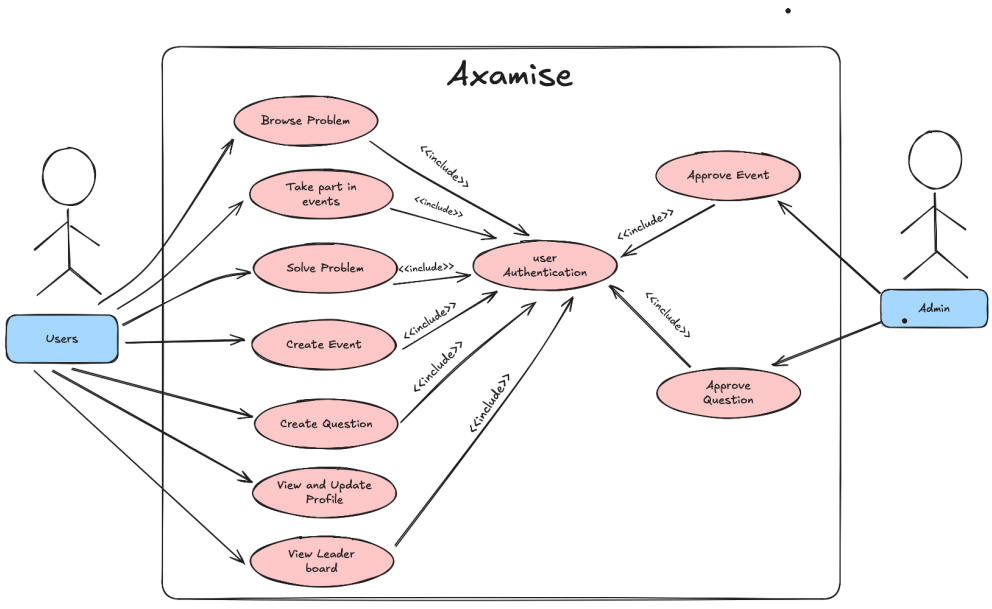
3.1.4. Competitive Ranking and Leaderboard

- 3.1.4.1. The system **shall** implement the proprietary scoring algorithm to determine user points for completed problems.
- 3.1.4.2. The system **shall** maintain and display a **global leaderboard**, sortable by current rank, total score, and recent activity.
- 3.1.4.3. The system **shall** enable users to search for and view the rank of any other registered user.
- 3.1.4.4. The system **shall** provide filtering options for the leaderboard (e.g., filtering by country, specific competition, or skill tier).
- 3.1.4.5. The system **shall** update the leaderboard data in **near real-time** following the successful evaluation of a user's submission.

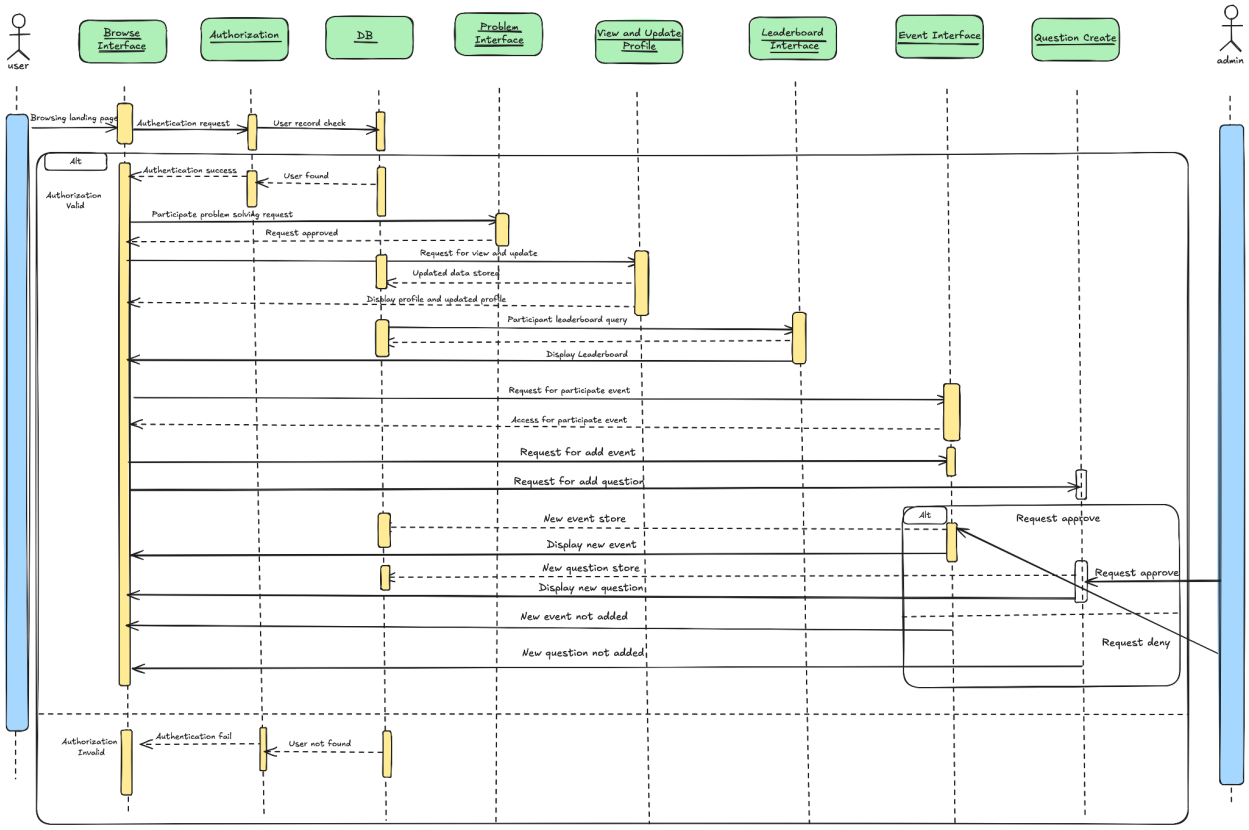
Rationale:

The competitive component fosters user retention, goal orientation, and community growth while showcasing individual improvement over time.

3.1.5. Use Case Diagram



3.1.6. Sequence Diagram



3.2. External Interface Requirements

This subsection defines all required interactions between the Axamise system and the external environment.

3.2.1. User Interfaces

- 3.2.1.1. The system **shall** provide a responsive web interface optimised for both desktop and mobile viewports.
- 3.2.1.2. The IDE interface **shall** provide a clear separation between the problem description, code editor, and execution output windows.
- 3.2.1.3. All navigation elements **shall** be intuitive and provide consistent access to core features (Practice, Contests, Leaderboard).
- 3.2.1.4. Visual elements shall support dark/light themes for readability.
- 3.3.1.5. Animations and transitions (via Framer Motion) shall enhance UX without affecting performance.

3.2.2. Software Interfaces

- 3.2.2.1. The system **shall** interface with a robust third-party or custom Code Execution Service (CES) via a secure API (REST/JSON) for submitting code and retrieving execution results.
- 3.2.2.2. The system **shall** integrate with a persistent data storage solution (e.g., Firestore) to manage all application data.
- 3.2.2.3. The system **shall** utilise an external or integrated Authentication Service for managing user credentials securely.
- 3.2.2.4. Notifications shall be managed using Firebase Cloud Messaging (FCM).
- 3.2.2.5 All third-party API keys shall be securely stored in environment variables.

3.2.3. Communications Interfaces

- 3.2.3.1. All data transmission between the client (user browser) and the server **shall** be secured using **HTTPS/TLS encryption**.
- 3.2.3.2. The Code Execution Service communication **shall** utilise secure, low-latency protocols to minimise submission processing time.
- 3.3.3.3. The system shall log all API communications for auditing and debugging.

3.3. Data Requirements

This section describes the essential data entities that must be managed and preserved by the Axamise system.

3.3.1. Persistent Data Entities

- 3.3.1.1. **User Profile Data:** Must store immutable user ID, email, display name, password hash, registration date, and current score/rank.
- 3.3.1.2. **Challenge Data:** Must store problem statement, input/output specifications, creation date, and difficulty rating.
- 3.3.1.3. **Test Case Data:** Must store the hidden input and corresponding expected output for each challenge, accessible only to administrators and the CES.
- 3.3.1.4. **Submission Data:** Must store the submitted code, language used, submission timestamp, result status (e.g., Accepted, TLE), and measured execution metrics (time/memory).

3.3.2. Data Integrity and Security

- 3.3.2.1. The system **shall** encrypt all sensitive user data, including passwords and personally identifiable information, **both in transit and at rest**.
- 3.3.2.2. All submitted code and its results **shall** be recorded as **immutable history** linked to the user and challenge UID.
- 3.3.2.3. The ranking algorithm **shall** be protected from manipulation and must recalculate rankings accurately following every official submission.

3.4. Usability

Usability requirements ensure that the system is intuitive and accessible, minimizing the cognitive load on the user.

3.4.1. Graphical User Interface (GUI)

- 3.4.1.1. The system **shall** provide a uniform look and feel (consistent fonts, color scheme, and layout) across all practice and competition pages.
- 3.4.1.2. The IDE **shall** provide context-aware feedback (e.g., highlighted compilation errors) directly within the editor window..
- 3.4.1.3. The navigation structure **shall** limit the number of clicks required to reach any core feature (Practice, Leaderboard, Profile) to a **maximum of two** from the homepage.

3.4.2. Accessibility

- 3.4.2.1. The system **shall** comply with **WCAG 2.1 Level AA** standards for visual contrast and text size.
- 3.4.2.2. The system **shall** be fully navigable via keyboard commands to support users who cannot use a mouse.

3.5. Reliability & Availability

These requirements define the system's ability to operate without failure under stated conditions and for a specified period of time, especially during competitive events.

3.5.1. System Availability

- 3.5.1.1. The core Axamise platform (excluding scheduled maintenance) **shall** maintain an uptime of **99.9%**.
- 3.5.1.2. The Code Execution Service (CES) **shall** maintain an uptime of **99.99%** during all scheduled competitive events.

3.5.2. Fault Tolerance and Recovery

- 3.5.2.1. The system **shall** provide storage of all submission and ranking databases on redundant computers with automatic switchover.
- 3.5.2.2. In the event of a Code Execution Service failure, the system **shall** queue submissions and automatically process them upon CES recovery, ensuring no data loss.

3.6. Performance

Performance requirements govern the speed, responsiveness, and capacity of the Axamise platform

3.6.1. System Capacity

- 3.6.1.1. The platform **shall** be able to handle a minimum of **5,000 concurrent active users** during a peak-hour competitive event.
- 3.6.1.2. The Code Execution Service (CES) **shall** be capable of processing a sustained rate of **20 submissions per second** during peak load.

3.6.2. Response Time

- 3.6.2.1. The global leaderboard **shall** load and display within a maximum of **2 seconds** under normal operating load.
- 3.6.2.2. The time required to receive a final result status (excluding TLE duration) after a code submission **shall not exceed 10 seconds** for standard practice problems.
- 3.6.2.3. Page-to-page navigation within the main application (e.g., from Home to a Challenge page) **shall not exceed 1.5 seconds**.

3.7. Security

Security requirements protect the system and user data from unauthorised access and ensure the integrity of the competitive environment.

3.7.1. Data Transfer Security

- 3.7.1.1. The system **shall** use **Secure Sockets Layer (HTTPS/TLS)** for all transactions that include any confidential customer information.
- 3.7.1.2. The system **shall** automatically log out all customers after a configurable period of inactivity (e.g., 30 minutes).

3.7.2. Execution and Competition Integrity

- 3.7.2.1. The execution environment **shall** prevent submitted code from accessing network resources or the local file system outside of the designated sandbox.
- 3.7.2.2. The system **shall** use **encrypted password storage (salting and hashing)** for all user credentials.
- 3.7.2.3. The system **shall** implement **rate limiting** on code submissions to prevent denial-of-service or brute-force attack attempts against the CES.

3.8. Supportability

Supportability requirements define the attributes of the software that bear on the effort needed to maintain and evolve the system.

3.8.1. Maintainability

- 3.8.1.1. The source code developed for this system **shall** be maintained in a standard Configuration Management (CM) tool (e.g., **Git**).
- 3.8.1.2. All system components **shall** be delivered with developer-level documentation, including API usage and deployment guides.
- 3.8.1.3. System administrators **shall** be able to easily add or modify supported programming languages with minimal downtime.

3.8.2. Diagnosability

- 3.8.2.1. The system **shall** implement **centralised logging** for all critical operations (submission, execution, authentication) to facilitate debugging and auditing.

3.9. Design Constraints

These are factors that restrict the development team's design choices.

3.9.1. Technical Stack

- 3.9.1.1. The primary front-end must be implemented using a modern, component-based framework (e.g., **React**).
- 3.9.1.2. The system **shall** utilise a containerised sandboxing approach (e.g., **Docker**) for the Code Execution Service.

3.9.2. Platform Constraints

- 3.9.2.1. The product **must be stored in such a way that allows easy client access** via a standard URL.
- 3.9.2.2. Client computers must be equipped with modern web browsers (e.g., Chrome 100+, Firefox 90+, Edge).

3.10. On-line User Documentation and Help System Requirements

The online help system is a critical component for guiding users through the complex environment of competitive coding.

- The system **shall** provide **contextual online help** (tooltips, help icons) for all major IDE features and submission processes.
- The system **shall** include a comprehensive **FAQ** section covering common technical and competition-rule questions.
- The system **shall** include a dedicated section detailing the Time Limit and Memory Limit specifications for each supported programming language.

4. Other Nonfunctional Requirements

4.1. Scalability

- The system shall automatically scale horizontally to handle increased workloads without degrading performance.
- The architecture shall support the addition of new modules (languages, AI evaluators, or features) with minimal codebase changes.
- The database and backend services shall be capable of handling a growing user base exceeding 100,000 registered users.
- The Code Execution Service shall dynamically allocate containers based on incoming submission load to maintain evaluation speed.

4.2. Interoperability

- The system shall provide standardised RESTful APIs for integration with third-party learning management systems (LMS).
- Axamise shall ensure that its API layer follows OpenAPI standards for easy future extension.
- The platform shall support integration with AI coding assistants and external event management tools without major architectural changes.
- All services shall use consistent data serialisation (JSON format) to ensure compatibility between client and server modules.

4.3. Portability

- The application shall be deployable across multiple cloud environments (Google Cloud, AWS, Azure) with minimal configuration changes.
- The client interface shall remain functional across operating systems (Windows, macOS, Linux, Android, iOS) through browser-based access.
- The codebase shall use containerization (Docker) to ensure portability between development, testing, and production environments.

4.4. Extensibility

- The architecture shall follow modular design principles, allowing new features like “Team Contests,” “AI Code Review,” or “Gamified Learning” to be integrated with minimal effort.
- The plugin system shall allow external developers to contribute new problem categories or language runtimes.
- Future AI upgrades (e.g., switching from Gemini 2.5 Pro to newer models) shall not require rewriting core backend logic.

4.5. Auditability

- All user actions, including submissions, edits, and administrative changes, shall be logged and timestamped in a secure audit trail.
- Logs shall be immutable and accessible only to authorised administrators.
- The system shall provide data export capabilities for compliance verification and academic audits.

4.6. Compliance and Ethics

- Axamise shall comply with **GDPR**, **COPPA**, and relevant data protection regulations for handling user data.
- AI components shall operate transparently — users shall be informed when AI is involved in evaluation or scoring.
- Collected data shall be used solely for educational and analytical purposes, never for marketing without explicit consent.
- All communications and stored user content shall comply with ethical AI standards and OpenAI's responsible use guidelines.

4.7. Localization and Internationalization

- The platform shall support multi-language UI (starting with English, with future extensions to other languages).
- Number, date, and currency formats shall adapt to user's locale.
- Unicode compliance shall ensure compatibility with multilingual text (e.g., variable names in non-Latin scripts).

4.8. Performance Optimisation

- The platform shall use caching (Cloud CDN / Redis) to reduce server response times for frequently accessed data.

- Problem lists, leaderboards, and user profiles shall be served through optimised queries and prefetching mechanisms.
- Front-end scripts and assets shall be minified and lazy-loaded to ensure page load times under 2 seconds on broadband.

4.9. Environmental and Energy Efficiency

- The system shall optimise resource utilisation to minimise CPU and memory usage per container.
- Idle containers or processes shall auto-terminate to reduce energy consumption.
- Cloud deployments shall favour data centres using renewable energy sources when available.

4.10. Disaster Recovery and Business Continuity

- The platform shall maintain off-site backups replicated across multiple geographic regions.
- A disaster recovery plan shall ensure restoration of services within 1 hour (RTO) and minimal data loss (RPO $\leq$ 5 minutes).
- Regular system health checks and failover simulations shall be performed quarterly.

4.11. AI-Specific Non-Functional Requirements

- The Gemini AI integration shall respond within 2 seconds for small code evaluations and within 5 seconds for complex submissions.
- AI-generated feedback shall maintain a minimum accuracy threshold of 90% in identifying logical or syntax-related issues.
- The AI system shall continuously learn from anonymised user submissions to enhance evaluation quality.
- AI recommendations shall remain explainable and traceable to their reasoning steps (AI transparency).

4.12. Continuous Deployment and Monitoring

- CI/CD pipelines shall automate testing, linting, and deployment of new builds.
- The system shall integrate with monitoring tools (e.g., Firebase Analytics, Google Cloud Monitoring) to track uptime, latency, and error rates.
- Developers shall be notified automatically upon build failures or performance degradation.

5. Safety Requirements

- All user-submitted code shall be executed in an **isolated sandbox environment** to prevent any harmful access to system resources.

- The system shall prevent user code from accessing the **file system, network, or external APIs**.

- Automatic backups of **submission history, rankings, and user data** shall occur daily to prevent data loss.

- The system shall comply with general data protection and operational safety standards applicable to educational web platforms.

5.2. Legal, Copyright, and Other Notices

These requirements define the legal obligations of the platform.

- The website footer **shall** display disclaimers, copyright notices, and a link to the **Terms of Service (TOS)** and Privacy Policy.
- The system **shall** require users to explicitly accept the **Terms of Service** upon initial registration.

- The platform **shall** adhere to industry standards regarding data privacy and protection (e.g., **GDPR principles**).

6. Glossary of Terms

6.1. System and Architecture Terms

Term	Definition
Axamise	The competitive programming and learning platform is being developed to enhance user coding proficiency through practice, AI evaluation, and global ranking.
Module	A self-contained functional unit in the Axamise system, e.g., Authentication, Event Management, or Submission Evaluation.
Factory Pattern	A design pattern used to modularise backend services for flexibility and scalability.
Service Layer	Backend modules are responsible for communication between controllers and models (e.g., _factory.approval.service.ts).
Controller	The logic layer that handles user requests and communicates with the models and services (e.g., event_create.controller.js).
Model	Database schema and business entities used for managing persistent data (e.g., User_Model.js, Event_Model.js).

6.2. User and Role Terms

Term	Definition
Learner/Novice Programmer	A user new to programming who practices beginner-level problems to improve skills.
Competitive Programmer	An advanced user participating in time-bound contests and global leaderboards.
Administrator	A platform manager responsible for event creation, problem approval, and user moderation.
Content Manager	A specialised admin role managing challenge curation and test case verification.

6.3. AI and Evaluation Terms

Term	Definition
Gemini 2.5 Pro	AI model integrated into Axamise for intelligent evaluation, feedback, and scoring.
AI Evaluator	Module responsible for analysing correctness, performance, and coding style.
AI Feedback Report	A structured result generated by Gemini AI indicating efficiency, time complexity, and improvement suggestions.
AI Explainability	Transparency in AI scoring — ability to trace reasoning steps behind feedback.

6.4. Development and Execution Terms

Term	Definition
Code Execution Service (CES)	A secure, containerised (Docker-based) sandbox that executes and tests user-submitted code.
Sandbox	An isolated environment prevents user code from accessing unauthorised system resources.
Docker	A containerization tool used for sandbox creation.
Firestore	A cloud-based NoSQL database used for real-time storage of users, submissions, and rankings.
Firebase Authentication	Google-provided authentication system for secure login and account management.
Framer Motion	A React-based animation library used for UI transitions in the Axamise front-end.
Tailwind CSS	Utility-first CSS framework used for rapid UI styling.

6.5. General Technical Terms

Term	Definition
IDE	Integrated Development Environment — browser-based editor for writing and executing code.
TLE / MLE	Time Limit Exceeded / Memory Limit Exceeded — standard programming evaluation constraints.
UI/UX	User Interface / User Experience design principles guiding layout and usability.
CI/CD	Continuous Integration and Continuous Deployment pipelines used to automate testing and deployment.
REST API	Web service interface allowing communication between client and server over HTTP.

7. Resources and References

7.1. Technical References

Type	Reference / Resource
Frameworks & Libraries	ReactJS, Firebase SDK, Tailwind CSS, Framer Motion, OpenAI Gemini API, Node.js, Express.js, TypeScript
Backend Architecture	Factory Pattern in TypeScript, MVC Design Pattern
Database	Google Firestore (NoSQL, real-time sync)
Execution Environment	Docker (sandboxing), Node Runtime
Cloud Infrastructure	Google Cloud Platform (GCP) / Firebase Hosting
AI & Evaluation	Gemini 2.5 Pro documentation and API Reference
Accessibility Standards	W3C Web Content Accessibility Guidelines (WCAG 2.1 Level AA)
Security Standards	OWASP Top 10, GDPR Compliance Documentation
Design Guidelines	Google Material Design, Tailwind UI Principles

2.2. Documentation References

Document	Purpose
Axamise Vision Document	Outlines project goals, stakeholders, and high-level objectives.
Use Case Analysis Document	Provides detailed behavioral modeling for user interactions.
System Structural Model	Defines the architecture and module-level relationships.
Behavioral Model	Describes sequence and state transitions for user activities.
Non-Functional Requirements Model	Specifies performance, reliability, and security expectations.
Project Plan	Provides timelines, milestones, and development sprints.
Traceability Matrix	Maps functional requirements to test cases for validation.
Test Strategy and Evaluation Report	Describes testing methodologies for IDE, AI evaluation, and performance.

7.3. Research & Standards References

- ISO/IEC 25010:2011 — Software Product Quality Model
- IEEE 830-1998 — Recommended Practice for Software Requirements Specifications
- OpenAI Developer Documentation — Gemini Integration Guidelines
- Firebase Developer Guide — Firestore and Authentication
- Docker Documentation — Containerization for Secure Execution

8. Appendices

8.1. API Reference Summary

Endpoint	Controller	Functionality
/signup	signup.controller.js	Register a new user
/login	login.controller.js	Authenticate user
/createEvent	event_create.controller.js	Create a new competition event
/submit	solving_section.controller.js	Submit and evaluate code
/questionList	question_list.controller.js	Retrieve all available questions

8.2. System Architecture Diagram

- Client (React Frontend)
- Server (Node/TypeScript Factory Services)
- Firestore (Database)
- Firebase Auth
- Gemini AI Evaluator
- Docker Sandbox (CES)
- Inter-service communication via REST APIs.