Tecnológico de Monterrey Campus Santa Fe

Software Requirement Specification

Gabriel Rodríguez De Los Reyes A01027384

Mariel Gómez Gutiérrez A01275607

Pablo Banzo Prida A01782031

Santiago Rodriguez Palomo A01025232

Group 501

Final Assessment

June 6th, 2025

1. Introduction

This document will provide an overview of the project, detailing features, constraints and other design considerations that were taken to create our platform.

1.1 Purpose and Document Convention

This document defines the software requirements for the final assessment project, which requires hosting a service of our choice on the designated private cloud infrastructure. The requirements were gathered from a comprehensive kickoff session where we, as project stakeholders, identified essential features, technical constraints, and implementation considerations necessary for successful development and deployment.

To maintain professional standards and ensure document consistency, all content is formatted using Times New Roman typeface at 12-point font size. Section and subsection headers utilize Times New Roman typeface at 14-point font size and provide clear, descriptive titles that accurately reflect the content addressed in each respective section.

1.2 Project Scope

The project scope encompasses the deployment and configuration of a selected service on the designated private cloud infrastructure, establishing secure communication protocols with the provided server environment. The project requires selection between two primary service options: implementing our proprietary AI model for number recognition services or integrating a C-compiler solution.

Key deliverables include the implementation of comprehensive security frameworks, proper network architecture configuration, and seamless integration between the hosted service and existing server infrastructure. The project must demonstrate secure data transmission, appropriate access controls, and optimal performance within the private cloud environment while meeting all specified technical and operational requirements.

2. Overall description

At Compilo, we've built the ultimate platform for learning how to code because we believe programming education should be accessible, engaging, and effective for everyone. Whether you're taking your first steps into the world of coding or you're a developer looking to sharpen your existing skills, we provide all the tools and resources you need to succeed in one comprehensive platform. Our approach centers around three core pillars that set us apart: our interactive browser-based editor delivers real-time feedback so you can code seamlessly, our instant testing feature lets you run your code against test cases and see results immediately to accelerate your learning, and our beginner-friendly design ensures new coders get the helpful hints and clear explanations they need to build confidence. We've designed

Compilo to eliminate the barriers that often prevent people from learning to code, creating an environment where anyone can start their programming journey and truly conquer the world of software development.

2.1 Product Perspective

Compilo is a comprehensive educational platform designed to facilitate coding skill development for users at all proficiency levels. The platform leverages a proprietary C- compiler to address the current market gap in available compilers for this programming language. The solution focuses on delivering foundational programming exercises that emphasize core syntax comprehension and fundamental programming logic concepts. The platform features an intuitive user interface designed to ensure accessibility and user engagement across diverse skill levels. This user-centric design approach aims to create a supportive learning environment that transforms the coding education experience into an enjoyable and sustainable journey.

2.2 User Characteristics

The platform primarily targets aspiring programmers and coding enthusiasts seeking to develop proficiency in the C- programming language. These users demonstrate a strong motivation to engage with structured learning experiences and require access to a comprehensive repository of programming challenges specifically designed for C- implementation. Users value detailed diagnostic information that identifies specific areas for improvement, enabling targeted skill development and iterative learning. The platform caters to users who prioritize intuitive design in their learning tools. These individuals seek distraction-free environments that facilitate deep focus on problem-solving activities. The target audience appreciates clean, professional interfaces that minimize cognitive overhead.

2.3 Operating Environment

The service deployment operates within a designated server infrastructure (Server3), allocated to our development team. The architecture utilizes five distinct virtual instances deployed across the private cloud infrastructure to support distributed service components including web services, application programming interfaces (APIs), backend processing systems, and load balancing mechanisms.

The underlying cloud infrastructure is built upon OpenStack, an industry-standard open-source cloud computing platform. OpenStack manages over 40 million cores globally and maintains widespread adoption across diverse industry sectors, demonstrating its enterprise-grade reliability and scalability. Key implementation components include Next.js for frontend development, Go programming language for backend services, MongoDB for database management, and Flask for our compiler. This technology combination provides a scalable, maintainable, and high-performance foundation for the platform's operational requirements.

2.4 Design and Implementation Constraints

- The compiler must operate on the designated server infrastructure.
- The platform requires deployment within the private cloud environment.
- Project completion timeline is set at seven days.
- Development team consists of four qualified developers.
- No budgetary constraints limit project implementation.

2.5 Assumptions

Due to the project's scope and critical constraints, several key assumptions were established to ensure seamless platform integration and successful implementation.

- Framework selection and service integration were unrestricted, allowing optimal technology choices for project requirements.
- Architectural design freedom was granted, with the requirement to maintain communication between server infrastructure and cloud instances.
- Instance provisioning was flexible, permitting resource scaling based on justified architectural needs.
- Platform functionality could be customized provided integration with either the compiler or AI project.
- Development language selection was unrestricted, allowing implementation in either English or Spanish.

3. System Features

3.1 Session Management

This functionality encompasses user account creation and session handling capabilities. User registration requires three essential fields: username, email address, and password. Upon successful validation, the system creates a new user record in the database, enabling subsequent authentication. The login process requires only the registered email and password credentials. The API manages user authentication and generates a JSON Web Token (JWT) with a 24-hour expiration period to maintain secure session control.

Response Sequence

User Registration: Users input personal credentials including username, email, and password. Following data validation, the system creates the new user account in the database.

User Login: Users provide their registered email and password. After credential validation, the system authenticates the user and grants platform access.

User Logout: Users initiate session termination through the logout function, which redirects them to the landing page for future authentication as needed.

Functional Requirements

- The system must provide registration and login interfaces enabling user account creation and access.
- The system must validate user data and securely create new user accounts in the database.
- The system must authenticate user credentials and grant access upon successful verification.
- The system must include logout functionality that securely terminates user sessions.

3.2 Problem Selection

This functionality provides users access to a comprehensive problem repository. Problems are displayed with essential metadata including problem number, descriptive title, and difficulty classification. Users can search for specific problems, apply sorting mechanisms, and filter content by difficulty level to locate problems that align with their skill development needs. Problem titles are intentionally descriptive to provide clear expectations regarding content and complexity before users begin implementation.

Response Sequence

View Problems: The system displays a scrollable interface presenting all available problems with corresponding difficulty indicators.

Problem Search: Users enter problem names in the search bar, and the system returns matching results based on the searched name.

Difficulty Filtering: Users utilize filtering controls to select from three difficulty classifications: easy, medium, and hard. The system displays only problems matching the selected difficulty level.

Problem Sorting: Users access sorting options to organize problems by difficulty level, problem title, or problem number. The system reorganizes and displays the updated problem list accordingly.

Functional Requirements

- The system must display a comprehensive list of available problems from the problem repository.
- The system must provide search functionality enabling users to locate specific problems by name.
- The system must offer filtering capabilities allowing users to view problems by difficulty classification.

- The system must include sorting functionality for organizing problems by number, title, and difficulty level.

3.3 Problem Solving

This functionality enables users to develop and test solutions for selected programming problems. The system evaluates submitted solutions and provides comprehensive feedback regarding correctness. Each problem includes detailed descriptions and illustrative examples to enhance user comprehension. All solutions are processed through the integrated C-compiler for execution and validation.

Response Sequence

View Problem: Users access comprehensive problem descriptions, including explanatory examples to facilitate understanding of requirements and expected outcomes.

Write Solution: Users input their solution code in the designated coding area, focusing on implementing the function body that addresses the problem requirements.

Run Code: The system compiles and executes the submitted code using the C- compiler infrastructure.

Receive Feedback: Users receive detailed test case results indicating passed or failed scenarios. Solutions achieving all test case requirements receive "successful" status, while partially correct solutions receive "partial" status for continued improvement.

Access Past Solutions: Users can review their previous solution attempts for any given problem to track progress and learning development.

Functional Requirements

- The system must display comprehensive problem descriptions and associated test case specifications.
- The system must provide a dedicated code input interface for users to write and submit their solutions.
- The system must compile and execute user code to generate expected output results.
- The system must evaluate solution correctness against test cases and communicate success or failure status to users.
- The system must maintain and display historical solution attempts for each problem per user.

4. Data Requirements

This section provides a comprehensive overview of the logical data structure underlying the platform architecture. The following subsections detail database implementation strategies, data interaction patterns, and system integration approaches that support the platform's core functionality and user experience requirements.

4.1 Data Dictionary

We present the list of entities and fields stored on our MongoDb

User

Field	Description
Username (string)	Unique identifier.
Email (string)	User's email address.
Password (string)	User 's password.

Problem

Field	Description
ID	Unique identifier.
Title (string)	Problem's descriptive title.
Description (string)	Problem's detailed description (might include an example).
Difficulty (string)	Problem's difficulty (easy, medium, hard).
TestCases (list of int)	List of test cases that will be used to evaluate the user's solution.
CreatedAt (Timestamp)	Timestamp of when the problem was created.
Params (list of string)	List of params needed for the function.

Logs

Field	Description
ID	Unique identifier.
UserID (string)	User's unique identifier.
Method (string)	Allows to know what kind of request the user made.
Path (string)	Allows to know what kind of request the user made.
ResponseStatus (int)	Able to know the problem status.
Duration (Timestamp)	Duration of the user interaction.
Body (string)	Request.
Problem (object)	Problem ID.
IP (string)	IP checker (frontend IP)
CreatedAt (Timestamp)	Timestamp of the log creation.

4.2 Data Acquisition, Integrity, Retention, and Disposal

Problems are loaded directly to the database by CompπBs. We ensure quality on our problems and are causally thought to develop user skill. The test cases will also be uploaded by us. The user data will be given by the user when account creation, and the system will save its logs such as submission on a specific problem. Nevertheless we implemented verification techniques to verify that the data is consistent and reliable.

5. External Interface Requirements

5.1 User Interface

The user interface is designed as an intuitive learning platform featuring a clean, modern aesthetic that provides users with a distraction-free environment for learning C- programming and developing coding skills. The interface adopts the brand's signature color palette of pale pink and purple tones, creating a distinctive and engaging visual experience.

The platform features a fully responsive design that adapts seamlessly across devices, with device-specific optimizations to enhance the learning experience on desktop, tablet, and mobile platforms. Navigation follows clear, descriptive naming conventions and logical routing structures, enabling users to begin their coding journey without confusion.

The interface prioritizes usability through standard, recognizable elements including login, logout, and registration functionality. Users receive immediate feedback upon code submission, and the coding environment is designed for intuitive interaction from the first use. This streamlined approach ensures that learners can focus entirely on skill development rather than navigating complex interface elements.

5.2 Software Interfaces

We use diverse service to provide best solution, each with a specific functionality and great advantages, it includes:

MongoDB: MongoDB serves as our primary database solution, offering a flexible NoSQL, open-source, document-oriented architecture. This database system scales efficiently from simple, small-scale implementations to complex, enterprise-level deployments. MongoDB provides advanced capabilities including interactive search, vector search, and real-time stream processing. Its inherent flexibility enables seamless integration with diverse services and applications, while its intuitive graphical user interface simplifies database management. The extensive MongoDB community contributes comprehensive documentation, tutorials, and implementation guides, facilitating straightforward deployment and configuration.

Docker: Docker provides an open platform for application development, deployment, and execution through containerization technology. This platform enables applications to be packaged with their dependencies into portable containers, ensuring consistent performance across diverse environments. Docker containers operate seamlessly on developer workstations, physical servers, virtual machines, cloud platforms, or hybrid infrastructure configurations. The platform integrates natively with CI/CD workflows, standardizing development environments and enabling efficient application distribution and testing. Container lifecycle management is simplified through Docker's comprehensive API and command-line interface.

Go: Go is an open-source programming language developed and maintained by Google, featuring a robust standard library and extensive ecosystem of development tools. The language excels in multiple implementation scenarios, including command-line interface development, web application creation, DevOps automation, and API construction—the latter being critical for our project requirements. Go delivers rapid compilation times that support iterative development workflows while maintaining low memory overhead. The language integrates seamlessly with containerization platforms like Kubernetes and Docker. Go's standard library provides native database connectivity with support for most SQL databases, and MongoDB offers official Go driver support with advanced features including client-side field encryption and comprehensive error handling through the ServerError interface.

Next.js: Next.js is a comprehensive full-stack framework built on React that enables the development of high-performance web applications. The framework includes built-in

optimizations for enhanced user experience and Core Web Vitals compliance, featuring dynamic HTML streaming and support for the latest React capabilities. Next.js automatically configures underlying development tools and provides essential features such as advanced data fetching mechanisms and server-side code execution. The framework supports Incremental Static Regeneration (ISR) on a per-page basis when static content optimization is required.

NGINX: NGINX functions as a high-performance HTTP web server, reverse proxy, content cache, load balancer, TCP/UDP proxy server, and mail proxy server. In our architecture, Nginx serves primarily as a load balancer to distribute incoming traffic efficiently across multiple backend servers.

Flask: Flask is a lightweight Web Server Gateway Interface (WSGI) framework that has become the standard for Python web application development. The framework maintains a minimalist core design while remaining highly scalable and supporting extensive functionality through its plugin ecosystem. Flask's emphasis on code readability and developer-friendly design makes it an optimal choice for implementing our Python-based compiler service, providing the necessary web interface for our compilation scripts.

OpenStack Cloud: OpenStack represents the most widely adopted open-source cloud computing platform globally, managing extensive pools of compute, storage, and networking resources. The platform comprises multiple integrated components including Nova (compute), Zun (container management), Swift (object storage), and Octavia (load balancing), among others. OpenStack supports diverse programming languages including Go and Python, and offers dual management interfaces: a comprehensive web-based dashboard for administrative control and RESTful APIs for programmatic resource provisioning and management.

5.3 Communication Interface

The system employs Hypertext Transfer Protocol Secure (HTTPS) as its primary communication protocol for data transmission between web browsers and the application servers. HTTPS represents the industry standard for secure web communication, providing essential security features that protect sensitive data during transmission.

The implementation of HTTPS is critical for our platform due to its robust encryption capabilities, which ensure the secure transmission of sensitive information including user credentials, authentication tokens, and personal data. HTTPS utilizes asymmetric cryptography, also known as public key infrastructure (PKI), which employs a pair of mathematically related keys, a public key and a private key, to establish secure communication channels between client and server endpoints. This encryption mechanism ensures data integrity, authentication, and confidentiality throughout the communication process. When a client initiates a connection, the server presents its digital certificate

containing the public key, which the client uses to encrypt data sent to the server. Only the server, possessing the corresponding private key, can decrypt this information, thereby preventing unauthorized access and man-in-the-middle attacks.

6. Quality Attributes

6.1 Usability

Our system features an intuitive interface that provides comprehensive access to all functionalities. The design prioritizes simplicity with clear, direct navigation paths that enable users to quickly locate desired features and services. Navigation is facilitated through visual interface elements, including graphical buttons that direct users to specific pages and functionalities. This approach ensures users can efficiently access different areas of the system without confusion.

To maintain transparency throughout user interactions, the system provides feedback through contextual messaging. Users receive clear, human-readable notifications that communicate system status, including both error alerts when issues occur and confirmation messages when tasks complete successfully. Rather than displaying technical error codes, the system presents understandable explanations that help users understand what is happening and how to proceed.

6.2 Security

We have implemented several critical security measures to protect both the system and user data.

Secure Communication Protocol HTTPS: serves as the primary communication protocol for all data transmission between web browsers and application servers. This implementation ensures encrypted communication channels, leveraging cryptographic hashing algorithms to maintain data integrity and confidentiality for both the service and end users.

Authentication and Session Management: JSON Web Tokens (JWT) provide robust session management and user authentication. This token-based system enables secure session identification while implementing automatic expiration policies. When tokens expire, users must re-authenticate to regain access credentials, ensuring that unauthorized access through compromised or stale sessions is prevented.

Code Execution Security: given that our system executes user-submitted code, we recognize the inherent risks of code injection attacks. While our containerized architecture adds an additional security layer, we have implemented Bubble Wrap as an extra safeguard. Bubble Wrap creates an isolated sandbox environment featuring a temporary filesystem and restricted

Linux namespaces for executing processes, effectively containing potential security threats and preventing system-level access.

6.3 Portability

Our system is designed primarily for web desktop browsers, ensuring broad accessibility across computing platforms. We maintain compatibility with all major browsers, including Google Chrome, Safari, Mozilla Firefox, and Microsoft Edge, providing consistent functionality regardless of the user's preferred browsing environment.

Responsive Design Architecture The system implements responsive design principles that automatically adapt to different screen sizes and resolutions. Users can resize interface components without compromising visibility or functionality, ensuring an optimal user experience across various display configurations.

Cross-Device Compatibility Beyond desktop environments, Compilo extends full functionality to mobile and tablet devices. The system operates seamlessly on iPad and iPhone platforms, delivering the same comprehensive feature set available on desktop browsers. This cross-device compatibility ensures users can access and utilize all system capabilities regardless of their chosen device or platform.