Vietnam National University - HCMC University of Science

Faculty of Information Technology

---□🕮□---

Project Proposasl: Programming Practice System

Course: Introduction to Software Engineering

Students that participated in this project:

Đinh Ngọc Anh Dương – 23127039

Trần Thị Xuân Tài – 23127256

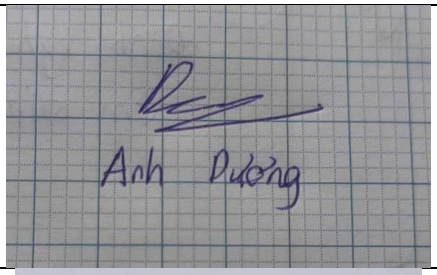
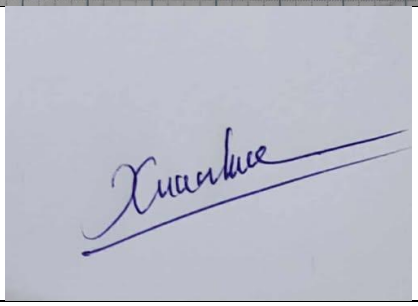
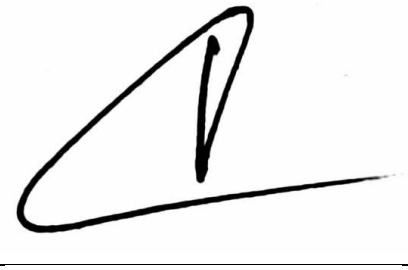
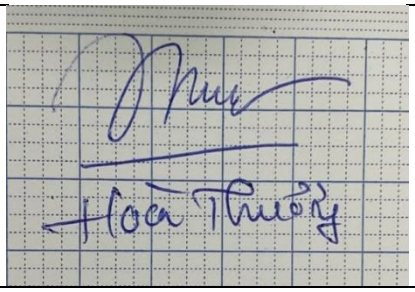
Trần Thọ - 23127264

Nguyễn Tuấn An – 23127316

Đỗ Thị Hoài Thương - 23127492

Instructor: Trương Phước Lộc

Member Contribution Assessment

ID	Name	Contribution(%)	Signature
23127039	Đinh Ngọc Anh Dương	100%	
23127256	Trần Thị Xuân Tài	100%	
23127264	Trần Thọ	100%	
23127316	Nguyễn Tuấn An	100%	
23127492	Đỗ Thị Hoài Thương	100%	

Preliminary Problem Statement for Programming Practice System

I. Overview

The "Programming Practice System" is the web-based platform designed to help users improve their programming skills by using a variety of coding challenges, competitions, and learning paths. Inspired by platforms like HackerRank, LeetCode, and Codeforces, the target of the system is to create an engaging and interactive learning environment for both newbie programmers and seasoned developers. The system supports a wide variety of programming languages and provides real-time feedback, detailed problem explanations, and performance tracking. Its goal is to create a comprehensive ecosystem where users can learn, practice, and prepare for technical interviews or competitive programming contest

II. FLOW

Users open the website on any modern browser, then register or log in using their credentials to access the system. After logging in, users are directed to their personal dashboard, which displays their statistics, recommended problems, and quick links to other features such as contests and code storage.

Users can explore the problem list categorized by topic, difficulty, or tags. Each problem provides a detailed description, input/output format, and sample test cases. Users write code directly in the integrated online

editor. They can select a programming language, run their code with sample inputs, and view outputs immediately.

When submitting, the system automatically evaluates the solution and returns feedback (Accepted, Wrong Answer, Runtime Error, …) along with runtime and memory details.

All submissions are automatically saved in the user's personal repository. Users can view previous versions, compare differences, and restore old code—similar to Git version management.

Users may participate in scheduled programming contests or assignments. The system enforces timing rules, evaluates submissions in real time, and updates leaderboards accordingly.

The profile section allows users to track their progress, achievements, and rankings. Statistics such as total solved problems and accuracy rate help measure learning outcomes.

After completing their activities, users can safely log out. All session data is securely handled, ensuring privacy and protection of personal information.

III. Key Features

1. **Problem Library:**
    - Thousands of programming problems categorized by difficulty/topic (for example: algorithms, data structures, machine learning), and company-specific interview questions
    - Problems are available in multiple languages (Python, Java, C++, JavaScript)
2. **Code Editor with Real-Time Feedback:**

- Integrated browser-based code editor with syntax highlighting, auto-completion, and debugging tools
- Real-time feedback for code submissions, including test case results, execution time, and memory usage

3. **Learning Paths:**

- Courses and challenges focused on specific skills like programming, web development, database, or data science which is hot-trend major
- Step-by-step problem-solving guides and video explanations

4. **Analytics Dashboard:**

- Personalized performance tracking, including statistics like problem-solving accuracy, time complexity improvements, and skill trends
- Comparison tools to benchmark performance against other developers

5. **Mock Interviews:**

- Simulations of real technical interviews with timed coding problems and behavioral interview questions.
- Customizable interview experiences for specific companies (Google, Amazon, Microsoft)

IV. Operating Environment

Client-Side Environment:

- Web Browser: Modern HTML5-supported browsers like Google Chrome, Firefox, Safari, and Microsoft Edge
- sCode Editor: Browser-based editor using Monaco Editor for rich coding experiences.

V. Design and Implementation Constraints

1. **Programming Language**:
    - Backend: **Java**
    - Frontend: **React.js**
    - Support for multiple languages in the code editor (Python, C++,...).
2. **Database**:
    - Use of **SQL** databases like **PostgreSQL** for structured data.
    - NoSQL databases like **MongoDB** for storing unstructured data
3. **Documentation**:
    - User manuals and developer guides formatted using **Markdown** and hosted on **GitHub Pages** or **ReadTheDocs**
4. **Security**:
    - All user data and transactions are encrypted using **SSL/TLS**.
    - Regular vulnerability testing and compliance with **GDPR**
5. **File Storage**: GG Drive
6. **Project management tool**: JIRA
7. **Communicate tool**: GG Meet, Messenger
8. **Code storage:** Github

VI. Conclusion

The Programming Practice System is a scalable, secure, and feature-rich platform that addresses the growing need for developers to practice and improve their coding skills. By combining structured learning, real-time feedback, global competitions, and detailed analytics, the platform aims to become an essential tool for individuals preparing for technical interviews, competitive programming contests, or simply improving their problem-solving abilities

Proposed Solution

I. Software

1. Features

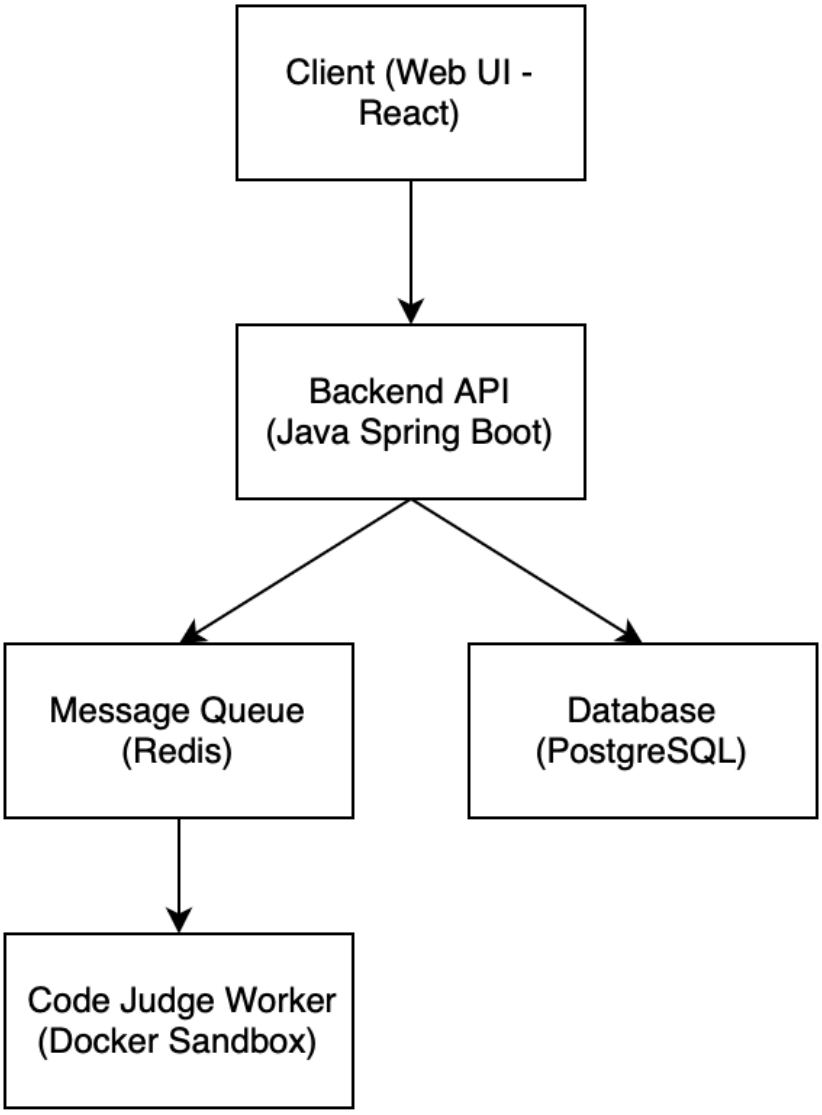
The project team applied the User Story Mapping approach to derive the actual software features required to satisfy user needs. The following table summarizes the mapping between user stories and the corresponding system features:

User Story	**Software Feature**
As a new user, I want to register an account so that I can participate in solving programming problems.	User Account Registration
As a registered user, I want to log in so that I can access problem sets and view my submission history.	User Login & Authentication
As a user, I want to view problem lists categorized by difficulty (Easy, Medium, Hard) so that I can select suitable problems.	Problem Browsing & Difficulty Filtering
As a user, I want to write code directly on the website and run it to see the results.	Online Code Editor & Code Execution
As a user, I want to see the execution results (correct/incorrect, runtime, errors) so that I know if my solution is correct.	Automatic Judging & Result Feedback

User Story	Software Feature
As a user, I want to view the history of submissions and the result of each attempt to track my learning progress.	Submission History & Progress Tracking
As a user, I want the code judging to respond within 10 seconds so that I do not have to wait too long.	High Performance Response Time (≤ 10s)
As a registered user, I want my password to be protected and not easily guessed.	Security & Password Hashing
As an administrator, I want to create, update, and delete problems to maintain and update the system content.	Admin Panel for Problem Management
As an administrator, I want to view the number of users and the statistics of submissions over time.	System Analytics & User Activity Statistics

2. Software Architecture

The software system follows a Client–Server architecture combined with asynchronous job processing for running and judging user code. The high-level architecture is illustrated as follows:



Main components:

- **Frontend Web (React):** provides user interface for problem browsing, code editing, and submission.
- **Backend API (Java Spring Boot):** handles authentication, business logic, problem management, and submission processing. Spring Boot was selected to better support enterprise-grade development practices and layered architecture.
- **PostgreSQL Database:** stores users, problem metadata, test cases, and submission history.
- **Redis Queue:** acts as a message broker to queue submissions and support non-blocking code execution.
- **Docker Judge Worker (Python):** executes user-submitted code inside isolated containers, evaluates correctness, and returns the result to the backend.

II. Hardware

1. Hardware Requirements

In order to run all software components (Frontend, Backend API, Database, Redis Queue, and Code Execution Worker), the system requires a computing environment with the following minimum specification:

Hardware Component	Minimum Requirement
CPU	2 cores
RAM	8 GB
Storage	at least 50 GB available
Operating System	Windows / Linux / macOS (Ubuntu recommended for worker execution)

A machine with the above specification is sufficient to run the full system locally for implementation, testing, and demonstration purposes.

2. **Required Equipment for Operation**

The following equipment is required to operate the system and demonstrate it during the project presentation:

Equipment	Purpose
Personal laptops (each group member)	Used for software development and local testing
One dedicated laptop (acting as local server)	Runs Backend API, PostgreSQL, Redis Queue, and Code Execution Worker
Internal Wi-Fi Router (LAN network)	Allows other devices to access the system during the live demo
Classroom projector / display screen	Used to present and demonstrate the system to the instructor and evaluators

Development Plan

I. Requirements Analysis

1. Objective

To understand what the web application should do, who will use it, and what core features should be included in the first version.

2. Activities

- Research similar coding platforms such as *LeetCode* and *HackerRank* to understand their main features and layout.
- Discuss in the team which features are suitable for a beginner student project.
- Identify two main user types:
    - **User (Student):** can register, log in, view and solve coding problems.
    - **Administrator (for later development):** can add and manage problems.
- Define basic functions for version 1.0:
        - User registration and login
        - Problem list page
        - Problem details page (problem description, example input/output)
        - Simple text-based code editor (for practice only)
        - Display of example output or simulated results

3. Deliverables

- Requirement list (functional and non-functional)
- Simple use case diagram
- User stories (e.g., “As a user, I want to log in so I can track my problem-solving progress.”)

II. Software Design

1. Objective

To prepare a simple overview of the system’s architecture and interface before coding starts.

2. Activities

- Learn and practice the basics of:
  - **ReactJS** for building the user interface.
  - **Spring Boot (Java)** for developing RESTful APIs.
  - **MySQL** for storing user and problem data.
  - **GitHub** for version control and collaboration.
- Create a simple architecture diagram showing data flow:
  - ReactJS (Frontend) → Spring Boot (Backend) → MySQL (Database).
- Draft basic UI wireframes for:
  - Login/Register page
  - Problem List page
  - Problem Details / Code Editor page
- Design an initial database schema including three main tables:

- User
- Problem
- Submission

3. **Deliverables**

- Basic system architecture diagram
- UI wireframes or sketches
- Draft ERD (Entity Relationship Diagram)
- Description of main database tables

III. Implementation

1. **Objective**

To prepare the working environment and organize the team structure before starting development.

2. **Activities**

- Create a GitHub repository for version control.
- Install and configure development environments:
  - **ReactJS:** Node.js, npm, and basic project setup.
  - **Spring Boot:** IntelliJ IDEA or Eclipse, Maven/Gradle configuration.
  - **MySQL:** Local database setup.
- Assign team roles:
  - Frontend developer(s): ReactJS UI
  - Backend developer(s): Java (Spring Boot APIs)

    - Database designer
    - Documentation manager
- Plan small development milestones:
    - Implement login and register API
    - Display problem list from database
    - Show problem details page

3. **Deliverables**

- GitHub repository created and organized
- Local development setup instructions
- Member task assignment list

IV. Testing

1. **Objective:**

Ensure that all components of the web application function correctly, meet the defined requirements, and provide a good user experience before release.

2. **Activities:**

- **Unit Testing:**
  Each developer tests their own modules independently (e.g., frontend components, backend API endpoints).
    - Tools: Jest (for React), JUnit (for Spring Boot).
- **Integration Testing:**
  Verify the interaction between frontend, backend, and database

(e.g., data fetched from MySQL displayed properly on React interface).

- **Functional Testing:**
  Check that each feature behaves according to requirements:
  - User registration and login flow
  - Viewing problem list
  - Viewing problem details and examples
  - Code editor displaying and submitting simulated output
- **Usability Testing:**
  Team members act as end users to ensure the interface is intuitive and responsive.
- **Bug Tracking:**
  All discovered issues are logged and managed using GitHub Issues or JIRA until resolved.

3. **Deliverables:**
   - Test plan and test case documentation
   - Bug report and resolution logs
   - Verified stable version for deployment

V. Deployment and Maintenance

1. **Objective:**
   Deploy a working version of the coding practice website and ensure long-term usability through regular updates. The entire

system will be hosted locally on a single machine acting as the main server. Each software component — frontend, backend, database, queue, and judge worker — will run as independent services using Docker containers to ensure consistency and easy setup across different devices.

2. **Activities:**

- **Deployment Setup:**
    - Host backend (Spring Boot API) locally for testing and demonstration.
    - Host frontend (ReactJS) using **Vercel** or **Netlify**.
    - Database hosted on local server for demo.
- **Environment Configuration:**
    - Use environment variables for API URLs and database credentials.
    - Configure CORS policy between frontend and backend.
- **Version Control and Continuous Integration:**
    - Use GitHub Actions for automatic testing before merging new commits.
- **Maintenance:**
    - Monitor uptime and fix issues from user feedback.
    - Plan feature updates (e.g., adding submission grading system or admin features).

Human Resources & Costing Plan

Role	Member	Responsibility	Tools/Resources
Project Manager	Nguyen Tuan An	Schedule deliverables, team coordination	Jira, Discord
Backend Dev	Tran Tho	API, judge, DB integration	NodeJS, MongoDB, PostgreSQL/MySQL
Frontend Dev	Xuan Tai, Hoai Thuong	UI/UX, code editor, problem list	ReactJS, Figma
DevOps	Nguyen Tuan An	Setup CI/CD, Docker, deployment	Docker, Github
QA/Tester	Duong	Manual + automated testing	Katalon, Postman
Documentation	Multiple students	Report	Google Docs, Latex, PDF

Estimated cost:

Item	Description	Cost (VND)
Cloud VM (for 1 month)	1 CPU, 1-2GB RAM	1-200.000
Doman (free/1 month)	hostinger.com	0-100.000
Tools (free)	Github, Jira, Docker,…	
Total	1-300.000	

Tool Setup

Tool	Purpose
Moodle	Assignment submission
Discord	Team coms
Jira/Trello	Task management and sprint tracking
Github	Version control and code hosting
Figma	UI/UX
Kalaton	Automated testing
Google Docs & Drive	Shared documentation

Folder structure:

/src : source code

/docs: documentation

/management → Planning & weekly reports

/requirements → SRS (PA2)

/design → Architecture & UI design (PA3)

/test → Test plan, test cases (PA4–PA5)

/pa → Submissions (PA1–PA5)