Software Requirements Specification

for

CodeSpace

Version 1.0

Prepared by Kaviselvan SJ

01-10-2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) defines the functional and non-functional requirements for **CodeSpace Platform – Version 1.0**, an online platform designed for coding practice and task-based problem solving. This document encompasses the system's complete scope, including authentication, user role management, task assignment, code compilation, evaluation mechanisms, and reporting functionalities.

It serves as a foundational document for the system's **design**, **development**, **testing**, and potential **future improvements**, ensuring that all **team members and evaluators** have a clear and consistent understanding of the platform's goals and requirements.

1.2 Document Conventions

Requirements are categorized using the following format:
- FR for Functional Requirements
- NFR for Non-Functional Requirements

Bold text is used to highlight key system components.
Monospace font is used for code snippets or API endpoints.
Priorities are indicated as:
- [H] – High
- [M] – Medium
- [L] – Low

1.3 Intended Audience and Reading Suggestions

Contestents – Use the platform to practice coding and solve tasks assigned by companies.
Companies – Assign coding challenges to developers and evaluate their performance.
Admins – Manage users, tasks, and monitor the platform for performance and security.

1.4 Product Scope

CodeSpace is a web-based platform designed to connect developers with companies through coding challenges and real-time evaluation. It functions both as a learning environment for users to improve their coding skills and as a recruitment tool for companies to assess and hire based on actual performance.

Key Features:
- **Role-based Secure Login** for Developers, Admins, and Company Users
- Task Assignment Engine with tags for difficulty, deadline, and language preference
- **Integrated Code Editor** supporting multiple programming languages

- **Online Compiler** Integration via Judge0 API
- Leaderboard and **Progress Tracking** per user and organization
- **Admin Dashboard** for analytics, usage tracking, and user management

The product supports the broader mission of bridging the gap between skilled talent and hiring companies by providing measurable, real-time coding performance data.

1.5 References

- Judge0 API Documentation - https://judge0.com/docs
- IEEE Standard for SRS Documents - https://ieeexplore.ieee.org/document/720574
- Express.js Documentation – https://expressjs.com/
- React.js Official Documentation – https://react.dev/
- Node.js Official Documentation – https://nodejs.org/en/docs
- MongoDB Documentation – https://www.mongodb.com/docs/

2. Overall Description

2.1 Product Perspective

CodeSpace is a standalone, web-based application developed specifically for academic and demonstration purposes. It is not derived from any existing system, nor is it part of a broader enterprise suite. This platform is being built from scratch and incorporates modern web development technologies, secure user management, and real-time code execution using third-party compiler APIs. The platform is designed to simulate real-world coding assessment environments by enabling users to solve programming tasks, receive performance feedback, and interact through role-based interfaces. CodeSpace **provides three distinct user portals — Contestent , Company, and Admin** - each with tailored access and functionality.

External Components and Technologies:

- **Online Compiler API** (e.g., Judge0): Used to compile and execute code securely in various programming languages.
- **Database**: Stores information such as user accounts, problem statements, code submissions, rankings, and system logs.
- **Authentication Service**: Manages secure login, role-based access, and session handling (using Firebase).

2.2 Product Functions

At a high level, CodeSpace must provide the following major functions:

User Authentication:

Secure registration and login for all user classes (Admin, Candidate, Company).

Problem Management:

Admins can create, update, and remove coding problems.

Candidates can view and practice coding problems.

Code Execution and Evaluation:

Integration with Judge0 API to compile, run, and evaluate code submissions in multiple programming languages.

Submission Review and Leaderboard:

Companies can review candidate submissions.

Candidates can view their scores and ranks in leaderboards.

Platform Monitoring and Administration:

Admins manage users, oversee activities, and ensure system integrity.

2.3 User Classes and Characteristics

CodeSpace identifies three primary user classes:

1. Candidates (Students/Job Seekers)

Characteristics: Frequent users, primarily interested in practicing coding problems and participating in contests. Moderate technical expertise (basic programming knowledge).

Functions Used: Registration/Login, Practice Problems, Contest Participation, View Leaderboards, Review Own Submissions.

Importance: High — main user base of the platform.

2. Companies (Recruiters/Organizations)

Characteristics: Occasional users, higher technical and domain expertise. Use platform mainly for creating and managing contests, reviewing submissions, and evaluating candidate performance.

Functions Used: Create Contest, Review Submissions, Access Leaderboards.

Importance: High — their engagement drives recruitment-related contests.

3. Admins (Platform Managers)

Characteristics: Limited user group with full technical and functional control. Higher technical expertise.

Functions Used: Manage Users, Manage Problems, Monitor Activities, Oversee Contests.

Importance: Very High — critical for platform maintenance, security, and reliability.

2.4 Operating Environment

- **Hardware Platform**: Any modern device (PC, laptop, or mobile) with internet access.
- **Operating System**: Cross-platform (Windows, macOS, Linux, android for web access)
- **Browser Support**: Latest versions of Chrome, Firefox, Edge, Safari.
- **Backend Environment**: Node.js with Express.js framework.
- **Database**: MongoDB (cloud-hosted).
- **Third-party Services**: Judge0 API for code execution.

2.5 Design and Implementation Constraints

Technology Constraints:

- Must use Node.js + Express.js for backend.
- Must use React.js for frontend.
- Must use MongoDB as database.
- Must integrate Judge0 API for code compilation and execution.

Regulatory/Security Constraints:
- Secure handling of user credentials using encryption (using firebase).
- Ensure data privacy for candidate submissions and company-created contests.

Performance Constraints:
- Real-time execution feedback depends on Judge0 API availability and response time.

Scalability Constraints:
- Limited by hosting resources; may require cloud scaling for heavy contest loads.

2.6 User Documentation

The following documentation will be delivered with the platform:
- **User Manual** (PDF/online) explaining system navigation, features, and usage for all user classes.
- **Quick Start Guide** for candidates (focused on registration, solving problems, and participating in contests).
- **Admin and Company Guide** covering contest creation, user management, and submission review.
- **Online Help/FAQ Section** accessible from within the application.

2.7 Assumptions and Dependencies

Assumptions:
- Users have a stable internet connection and modern browser.
- Candidates have basic programming knowledge.
- Judge0 API will remain available and stable for code execution.
- MongoDB cloud services will provide sufficient uptime.

Dependencies:
- Judge0 API for code execution (external dependency).
- Hosting services for backend and database.
- Browser compatibility for frontend rendering.
- Security libraries for authentication and encryption.

3. External Interface Requirements

3.1 User Interfaces

The system will provide a web-based graphical user interface (GUI) accessible via modern browsers. Key characteristics:
- **General Layout**:
  - **Navigation Bar**: Present on every page, includes Home, Practice, Contests, Leaderboard, Profile, and Logout options.
  - **Standard Buttons**: Submit, Cancel, Back, Next, Help.

    - **Consistency**: Uniform style using React.js components with Tailwind CSS for responsiveness and design consistency.
- **Error Handling**:
    - Error messages displayed inline (near the form fields) with red highlights.
    - Global errors displayed at the top of the page.
- **Keyboard Shortcuts**:
    - Ctrl + Enter for submitting code.
    - Esc to close modals or dialogs.
- **Screen Constraints**:
    - Responsive design (desktop, tablet, mobile).
    - Minimum resolution supported: 1024x768.

3.2 Hardware Interfaces

The software is web-based and requires only general-purpose computing devices.

- **Supported Devices**:
    - Desktops, laptops, and mobile devices with internet access.
- **Input Devices**:
    - Keyboard (primary input for coding).
    - Mouse or touch interface (for navigation).
- **Output Devices**:
    - Standard displays with minimum resolution 1024x768.
- **No specialized hardware interfaces** are required since execution is handled via external API (Judge0).

3.3 Software Interfaces

The system interacts with multiple software components:

1. **Database Interface**
    - **MongoDB (latest stable version)**
    - Data stored: user credentials, problems, submissions, contests, leaderboard data.
    - Communication: Mongoose ODM library.
2. **Backend Environment**
    - **Node.js + Express.js** framework.
    - Provides RESTful APIs for frontend–backend communication.
    - JSON used for request/response data.
3. **Frontend Environment**
    - **React.js** framework.
    - Interacts with backend APIs via Axios/Fetch.
4. **Third-Party API**
    - **Judge0 API** (code execution engine).
    - Input: code, language, and input test cases.
    - Output: execution result, errors, memory usage, execution time.
    - Protocol: HTTPS with RESTful requests.
5. **Operating System**
    - Runs on Windows/Linux-based hosting servers.
    - Compatible with browsers on Windows, macOS, Linux, Android, and iOS devices.

3.4 Communications Interfaces

Web Communication:
- Frontend and backend communicate using **HTTPS** requests.
- REST API endpoints deliver JSON responses.

Browser Requirement:
- Supports Chrome, Firefox, Safari, Edge (latest 2 versions).

Data Transfer Protocols:
- **HTTP/HTTPS** for client-server communication.
- **TLS/SSL** encryption for secure data transfer (especially for login and submissions).

Security:
- Firebase for session authentication.

Performance:
- API requests should complete within 3 seconds under normal server load.
- Code execution latency depends on Judge0 API availability.

4. System Features

4.1 User Authentication & Profile Management

4.1.1 Description and Priority

This feature allows users (Admin, Candidate, Company) to securely register, log in, and manage their profiles.
Priority: High(9)

4.1.2 Stimulus/Response Sequences

Stimulus: User enters registration details (name, email, password).
Response: System validates inputs, encrypts password, and creates a new user account.
Stimulus: User attempts login with credentials.
Response: System verifies credentials and grants access based on role.
Stimulus: User updates profile information.
Response: System validates and saves updated information.

4.1.3 Functional Requirements

REQ-1: The system shall allow new users to register with name, email, and password.
REQ-2: The system shall hash all passwords before storing them.
REQ-3: The system shall provide login authentication.
REQ-4: The system shall differentiate access rights based on roles (Admin, Candidate, Company).
REQ-5: The system shall allow users to update personal details such as name and profile picture.
REQ-6: The system shall reject invalid inputs (e.g., weak passwords, duplicate email).

4.2 Contest Management

4.2.1 Description and Priority

This feature allows admins (or companies) to create, update, and manage coding contests.
Priority: High

4.2.2 Stimulus/Response Sequences

Stimulus: Admin fills in contest details (name, description, start time, end time).
Response: System saves contest and makes it visible to users before the start time.
Stimulus: User clicks “Participate” during active contest time.
Response: System allows user to access problems and submit solutions.

4.2.3 Functional Requirements

REQ-7: The system shall allow admins to create contests with name, description, start time, end time.
REQ-8: The system shall not allow contests with invalid or past start dates.
REQ-9: The system shall make the “Participate” button active only during contest time.
REQ-10: The system shall allow admins to edit or delete contests before they start.
REQ-11: The system shall store contest data persistently in the database.

4.3 Problem Management

4.3.1 Description and Priority

This feature allows admins to create, update, and assign coding problems to contests.
Priority: High

4.3.2 Stimulus/Response Sequences

Stimulus: Admin creates a problem with title, description, input/output format, constraints, and test cases
Response: System stores the problem and associates it with a contest.
Stimulus: User views a problem during a contest.
Response: System fetches and displays the problem details.

4.3.3 Functional Requirements

REQ-12: The system shall allow admins to add problems with problem statement, sample input/output, and constraints.
REQ-13: The system shall allow test cases (hidden and sample) to be uploaded for each problem.
REQ-14: The system shall allow problems to be linked to one or more contests.
REQ-15: The system shall prevent deletion of problems if they are already in an active contest.

4.4 Code Submission & Evaluation

4.4.1 Description and Priority

This feature enables candidates to submit code solutions, which are evaluated using the Judge0 API.
Priority: High

4.4.2 Stimulus/Response Sequences

Stimulus: Candidate submits code in a selected programming language.
Response: System sends code to Judge0 API, retrieves output, and stores results.
Stimulus: Candidate submits invalid code.
Response: System returns compilation/runtime error

4.4.3 Functional Requirements

REQ-16: The system shall allow code submissions in multiple programming languages.
REQ-17: The system shall send submitted code to Judge0 API for execution.
REQ-18: The system shall return execution results (Accepted, Wrong Answer, TLE, Runtime Error, Compilation Error).
REQ-19: The system shall store all submissions for future review.
REQ-20: The system shall restrict submissions to one per problem at a time but allow resubmission before contest end.

4.5 Leaderboard & Results

4.5.1 Description and Priority

This feature displays rankings based on user submissions, accuracy, and speed.
Priority: Medium

4.5.2 Stimulus/Response Sequences

Stimulus: User solves a problem correctly.
Response: Leaderboard updates with points and submission time.
Stimulus: User views leaderboard during contest.
Response: System fetches and displays current rankings.

4.5.3 Functional Requirements

REQ-21: The system shall calculate points based on correctness and time of submission.
REQ-22: The system shall update leaderboard in real time or near-real time.
REQ-23: The system shall freeze leaderboard 30 minutes before contest end.
REQ-24: The system shall allow users to view final results after contest completion.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- The system shall handle at least 1,000 concurrent users without noticeable degradation in response time.
- API response times for standard requests (contest retrieval, leaderboard updates, problem fetch, etc.) shall be < 300 ms under normal load.
- Code execution via Judge0 API shall return results within 5 seconds for 90% of submissions, excluding intentionally long-running programs.
- The platform shall support scaling horizontally (using load balancing and containerization) to maintain consistent performance during peak contest hours.

5.2 Safety Requirements

- All user code shall be executed in isolated sandboxes (via Judge0) to prevent malicious access to the server or other users' data.
- The system must have **automatic backup** of contest data, user submissions, and leaderboards every 24 hours to prevent accidental loss.
- In case of server failure, the system shall recover from backup within **2 hours** with minimal data loss.
- No user code shall have access to the **host machine filesystem, network, or external APIs** beyond what is permitted.

5.3 Security Requirements

- Users shall authenticate using secure login mechanisms
- All communications between client and server shall be encrypted using **HTTPS/TLS 1.2 or higher**.
- User data (profile details, submissions, leaderboard stats) shall be stored in compliance with **GDPR-like privacy standards**.
- Admin features (contest creation, problem management, user management) shall only be accessible to accounts with **admin privileges**.
- The system shall enforce **role-based access control (RBAC)** to ensure separation between admin, company, and candidate functionalities.

5.4 Software Quality Attributes

- **Availability**: The platform shall maintain 99.5% uptime during contest periods.
- **Maintainability**: The codebase shall follow **modular architecture (React frontend, Express backend, MongoDB database)** for easier upgrades and bug fixes.
- **Usability**: The UI shall support **dark mode, mobile responsiveness, and intuitive navigation** for ease of use.
- **Reliability**: The platform must ensure that **submissions are never lost** even during server restarts (queue-based submission storage).

- **Scalability**: The system shall be deployable on **cloud infrastructure (AWS/Heroku/Vercel)** to handle growing user base.

5.5 Business Rules

- Only admins can create, edit, or delete contests and problems.
- **Candidates** can register, practice problems, participate in contests, and view leaderboards.
- **Companies** can create hiring contests and review candidate submissions but cannot modify system-wide settings.
- A user may only submit **one solution per problem per contest**, but can update it multiple times until the contest ends.

6. Other Requirements

6.1 Database Requirements

- The system shall use a relational database (e.g., PostgreSQL/MySQL) for managing user accounts, submissions, and question data.
- Database must ensure **referential integrity** between users, questions, and submissions.
- Submissions history should be **retained per user per question** with support for deletion and updates.
- Indexing must be applied to frequently queried fields such as user_id, question_id, and submission_id for performance optimization.

6.2 Internationalization Requirements

- The platform shall initially support **English**.
- Future extensions should allow localization (e.g., multi-language support for questions and explanations).
-

6.3 Legal and Compliance Requirements

- User data shall be stored securely in compliance with applicable data protection regulations (e.g., **GDPR-like principles** for data privacy).
- The platform shall not store sensitive payment information (as CodeSpace is an educational/coding platform).
- Code execution shall be sandboxed via Judge0 API to avoid **security risks**.

6.4 Reuse and Extensibility Objectives

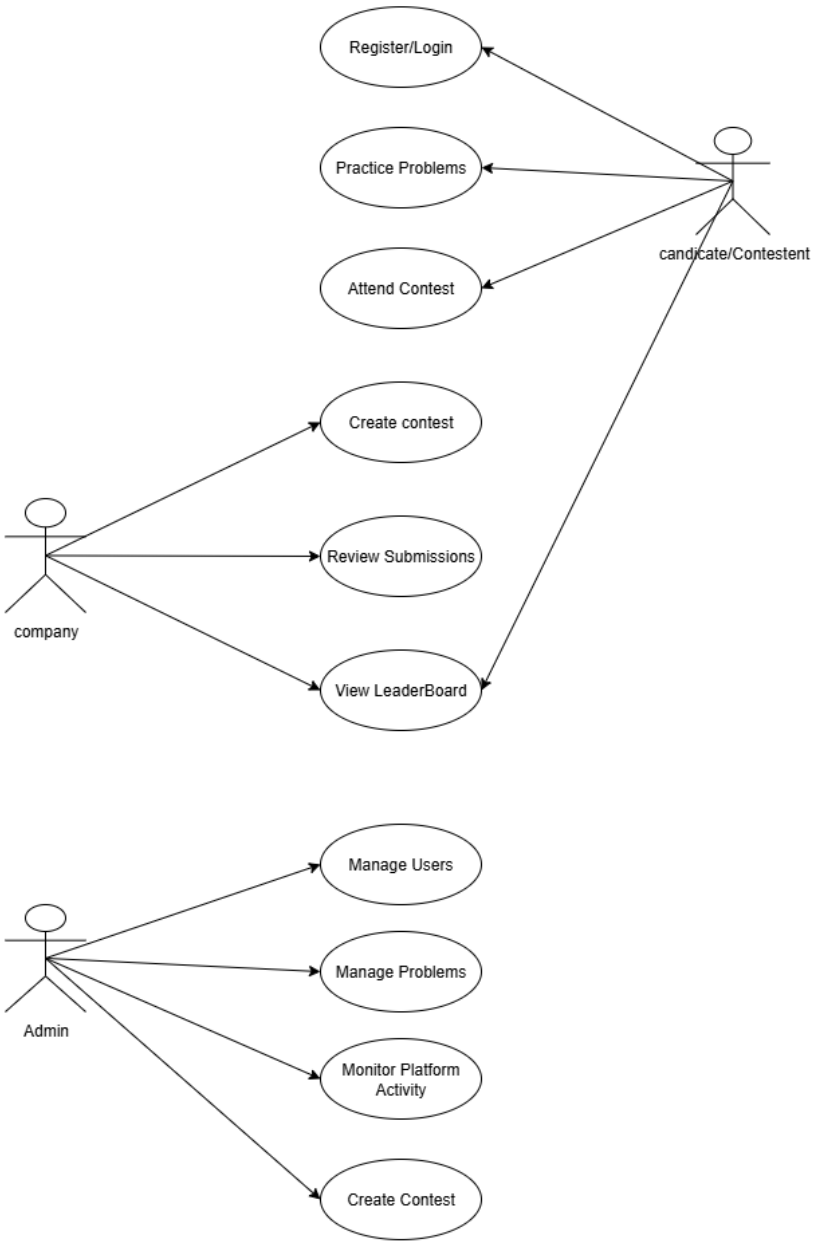
- The coding editor and Judge0 API integration shall be modular to allow reuse in other educational or competitive coding platforms.
- The question management system and submission tracking shall be extensible for **multi-course or multi-organization deployments**.
- The frontend design should follow **component-based architecture (React + Tailwind)** for easier reuse across projects.

Appendix A: Glossary

Term / Acronym	Definition
SRS	Software Requirements Specification – a formal description of the software system.
IDE	Integrated Development Environment – a tool where developers write and test code.
Monaco Editor	A web-based code editor used in CodeSpace, developed by Microsoft (also powers VS Code).
Judge0 API	An open-source API for secure code execution in multiple programming languages.
Frontend	The user-facing part of the system (React + Tailwind in CodeSpace).
Backend	The server-side part of the system (handles questions, submissions, authentication, etc.).
Submission	A user's code answer to a coding question, which is stored and validated against test cases.
Test Case	Input-output pair used to validate whether a submitted solution is correct.

Appendix B: Analysis Models

1.USECASE DIAGRAM:



Online Coding Platform – Admin/company/Candidate

UC1: Register / Login

- **Primary Actor:** Candidate / Contestant / Company
- **Secondary Actor:** Admin
- **Precondition:** Candidate must access the CodeSpace platform.
- **Trigger:** Candidate selects "Register" or "Login".
- **Main Success Scenario:**
  1. Candidate enters credentials.
  2. System validates the input.
  3. If new user, system creates account; else, logs in.
  4. Dashboard is loaded with user-specific data.
- **Exception Scenarios:**
  - a) Invalid credentials
    ➔ System shows error and prompts retry.
  - b) Email not verified
    ➔ System requests verification step.
  - c) Server down
    ➔ System displays fallback message.

UC2: Practice Problems

- **Actor:** Candidate / Contestant
- **Precondition:** Candidate must be logged in.
- **Trigger:** Candidate navigates to "Practice Problems" section.
- **Main Success Scenario:**
  1. Candidate selects a problem.
  2. System loads problem statement and editor.
  3. Candidate writes and submits code.
  4. Code is compiled and evaluated.
  5. System shows verdict and updates progress.
- **Exception Scenarios:**
  - a) Compilation error
    ➔ System displays error output.
  - b) Server timeout
    ➔ Retry mechanism or message shown.
  - c) Test cases not reachable
    ➔ System notifies internal issue.

UC3: Attend Contest

- **Primary Actor:** Candidate / Contestant
- **Secondary Actor:** Admin
- **Precondition:** Registered for an active contest.
- **Trigger:** Candidate selects an ongoing contest.
- **Main Success Scenario:**
  1. Contestant joins contest lobby.
  2. System loads problems with timer.
  3. Code submissions are accepted and judged in real-time.
  4. Leaderboard is updated accordingly.
- **Exception Scenarios:**
  - a) Contest expired
    ➔ System denies access and informs reason.

- b) Network issues
  ➔ System allows reconnection with session resume.

UC4: Create Contest

- **Primary Actor:** Admin
- **Secondary Actor:** Admin
- **Precondition:** Must be logged in as authorized user.
- **Trigger:** User clicks “Create Contest”.
- **Main Success Scenario:**
  1. User enters contest title, rules, date, problems.
  2. System validates input and stores contest.
  3. Contest becomes available on scheduled time.
- **Exception Scenarios:**
  - a) Missing problem set
    ➔ System alerts the user.
  - b) Overlapping contest time
    ➔ System prevents scheduling.

UC5: Review Submissions

- **Actor:** Company
- **Precondition:** Candidate must have submitted solutions.
- **Trigger:** Company selects “Review Submissions”.
- **Main Success Scenario:**
  1. Company browses candidate submissions.
  2. System shows code, test results, and timestamps.
  3. Company reviews and marks evaluations.
- **Exception Scenarios:**
  - a) Submission corrupted
    ➔ System flags issue to admin.
  - b) Access rights mismatch
    ➔ System blocks access and logs event.

UC6: View Leaderboard

- **Primary Actor:** Candidate / Company
- **Secondary Actor:** Admin
- **Precondition:** Contest must be completed or ongoing.
- **Trigger:** Actor clicks on “View Leaderboard”.
- **Main Success Scenario:**
  1. System fetches sorted scores.
  2. Leaderboard is displayed with rankings, times, and attempts.
- **Exception Scenarios:**
  - a) Contest data not available
    ➔ System shows maintenance notice.

UC7: Manage Users

- **Actor:** Admin
- **Precondition:** Must be logged in as admin.
- **Trigger:** Admin navigates to “User Management”.
- **Main Success Scenario:**
  1. Admin views user list.
  2. Admin activates/deactivates accounts or changes roles.
- **Exception Scenarios:**
  - a) Unauthorized action
    ➔ System restricts access.

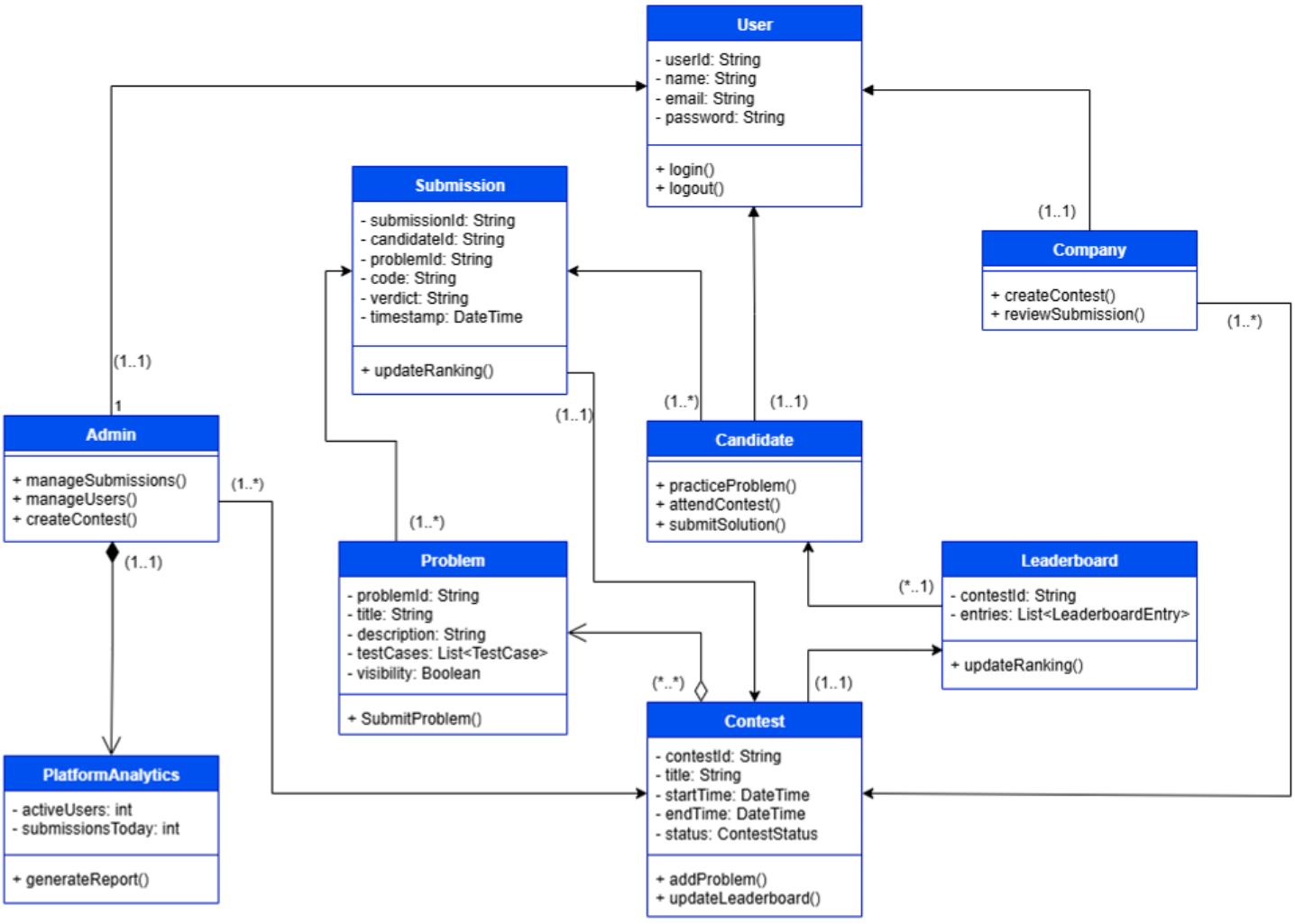
UC8: Manage Problems

- **Actor:** Admin
- **Precondition:** Must be logged in.
- **Trigger:** Admin selects “Manage Problems”.
- **Main Success Scenario:**
  1. Admin adds, edits, deletes problems.
  2. System validates test cases and formats.
  3. Problems get published to practice or contests.
- **Exception Scenarios:**
  - a) Test case format error
    ➔ System prompts correction.
  - b) Problem already used in live contest
    ➔ Action restricted.

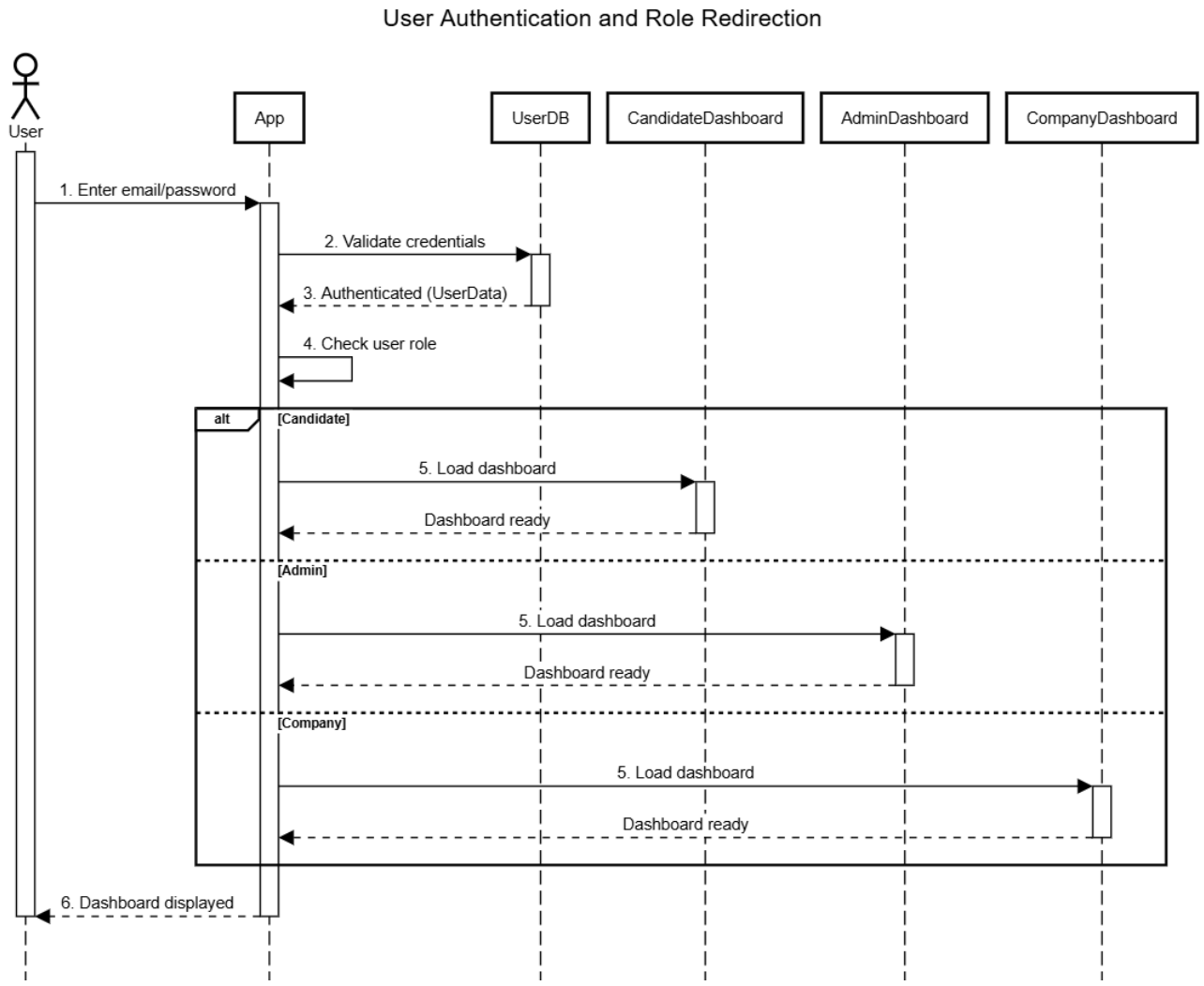
UC9: Monitor Platform Activity

- **Actor:** Admin
- **Precondition:** Admin must be authenticated.
- **Trigger:** Admin selects “Platform Analytics”.
- **Main Success Scenario:**
  1. System displays real-time stats: logins, submissions, errors.
  2. Admin can filter by time, user type, or contest.
- **Exception Scenarios:**
  - a) Database issue
    ➔ System shows degraded analytics.
  - b) Unauthorized access attempt
    ➔ Logged and blocked automatically.

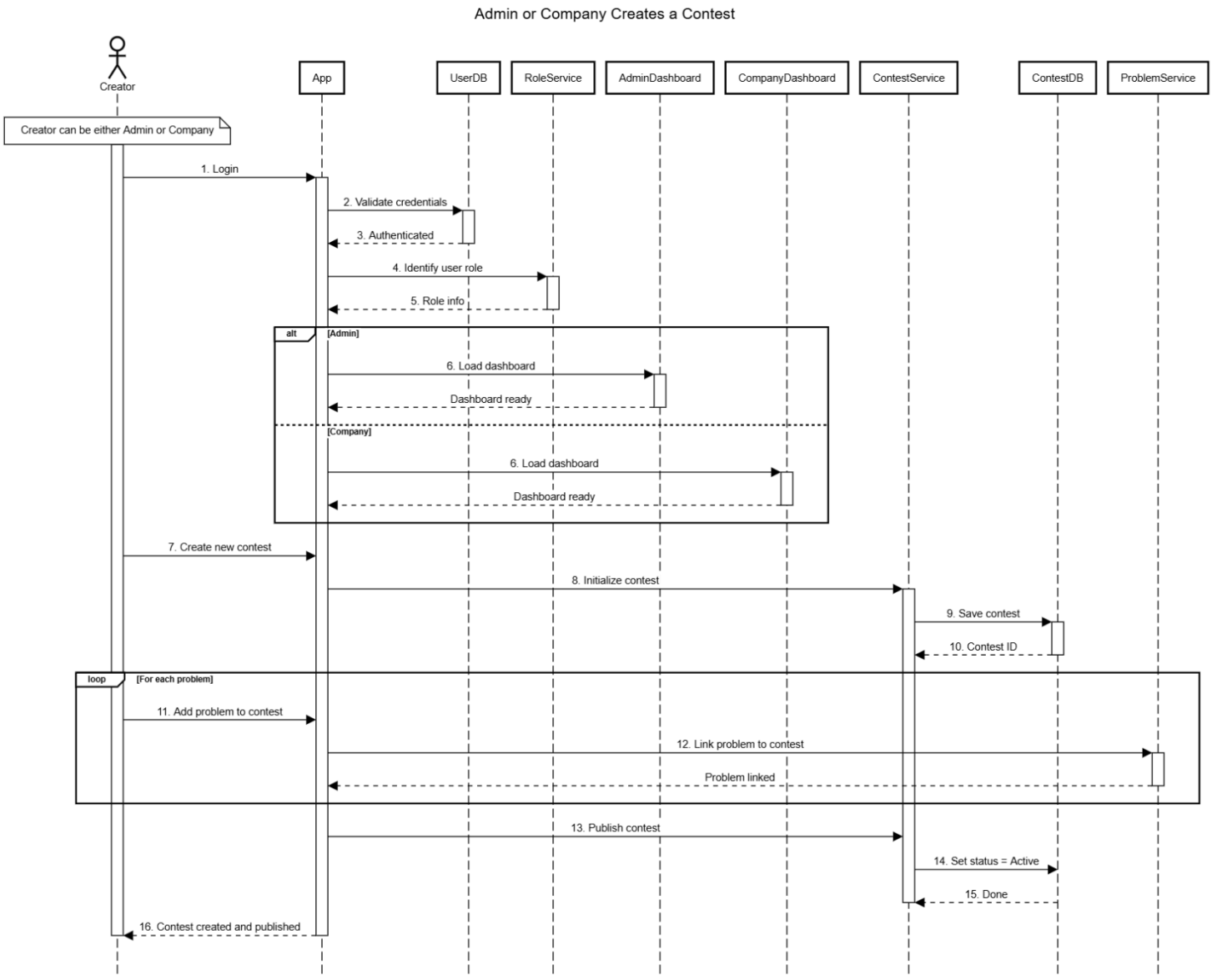
2. CLASS DIAGRAM



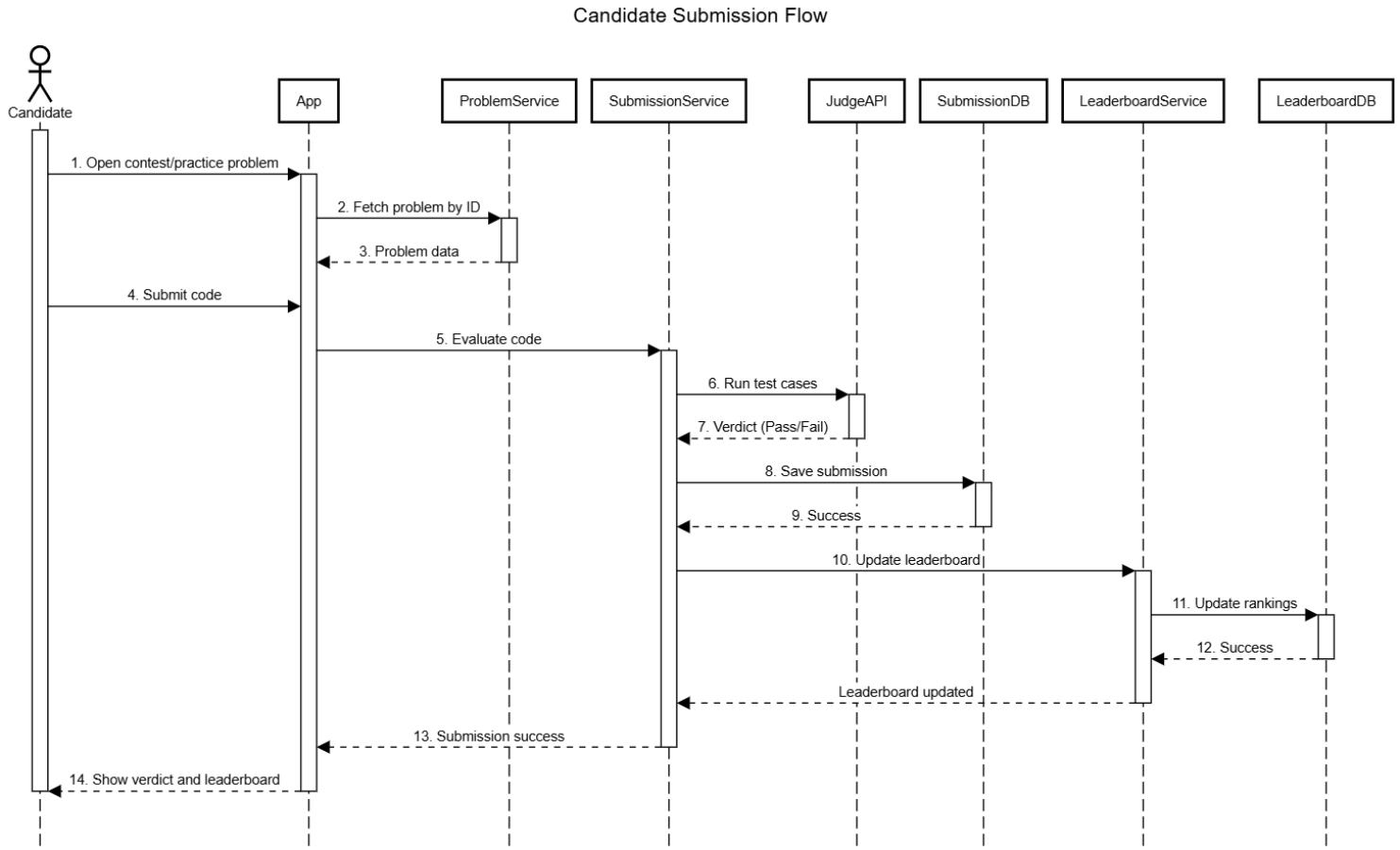
3.SEQUENCE DIAGRAM



a) Login Sequence

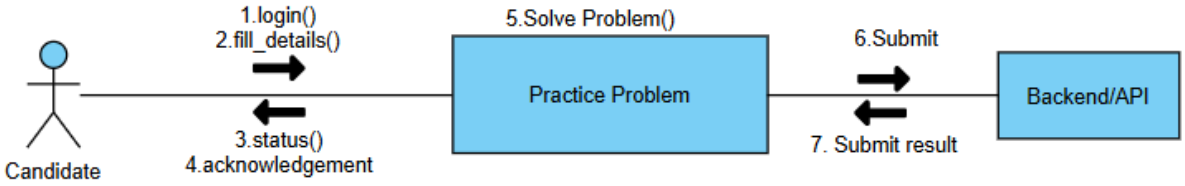


c) Contest Creation Flow

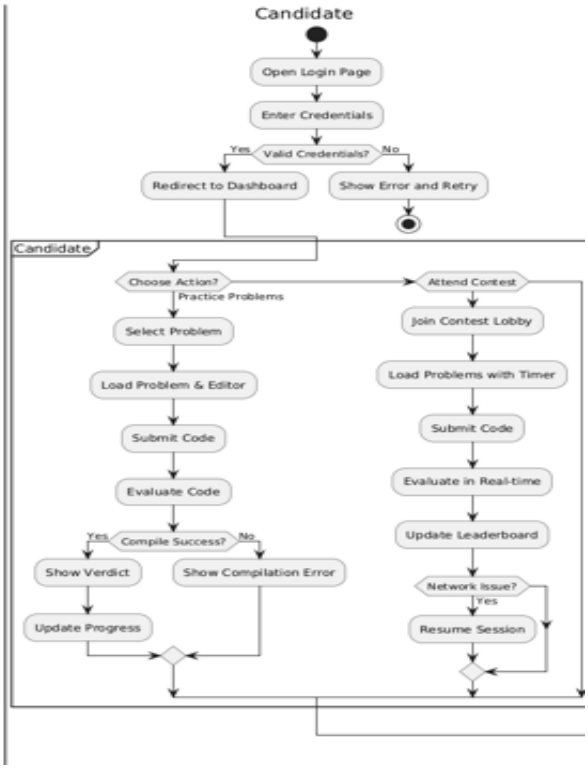


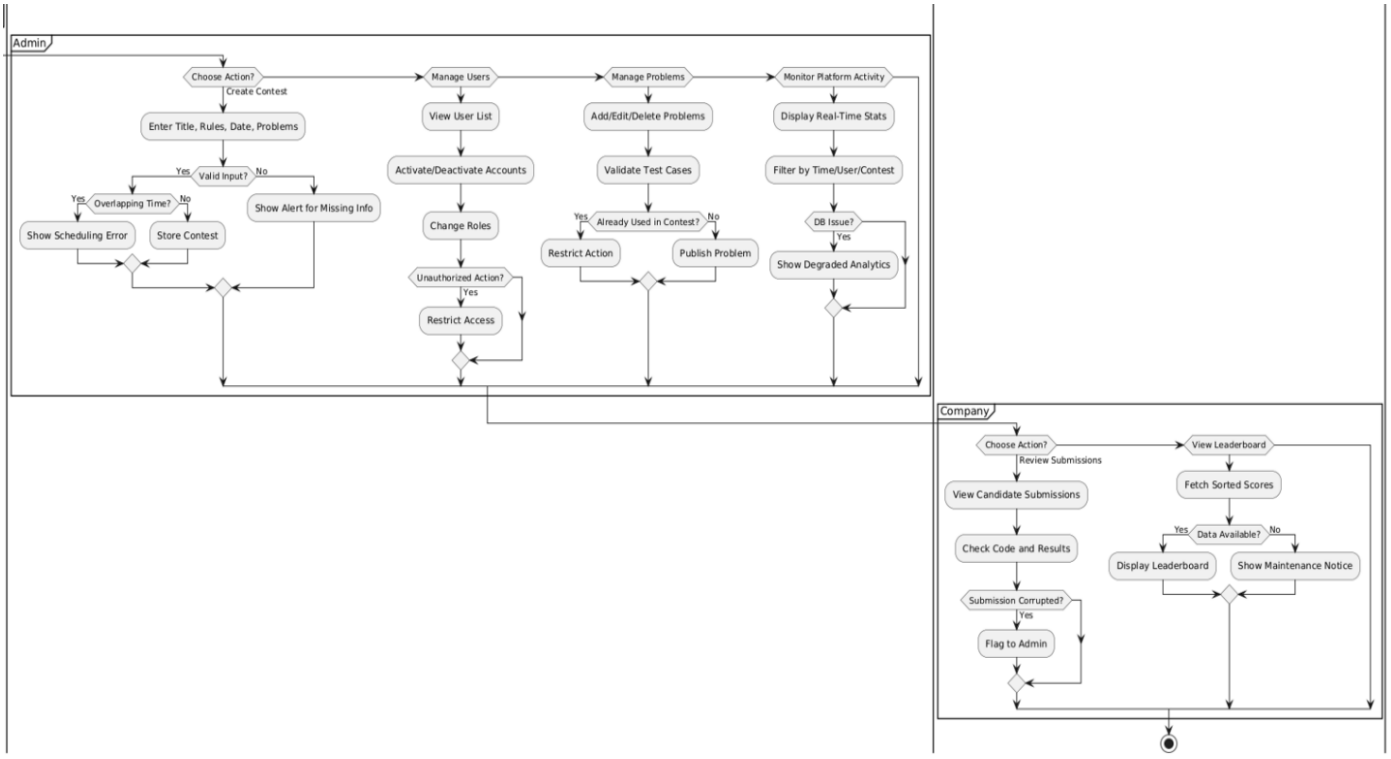
b) Candidate Submission Flow

4. Colloboration diagram

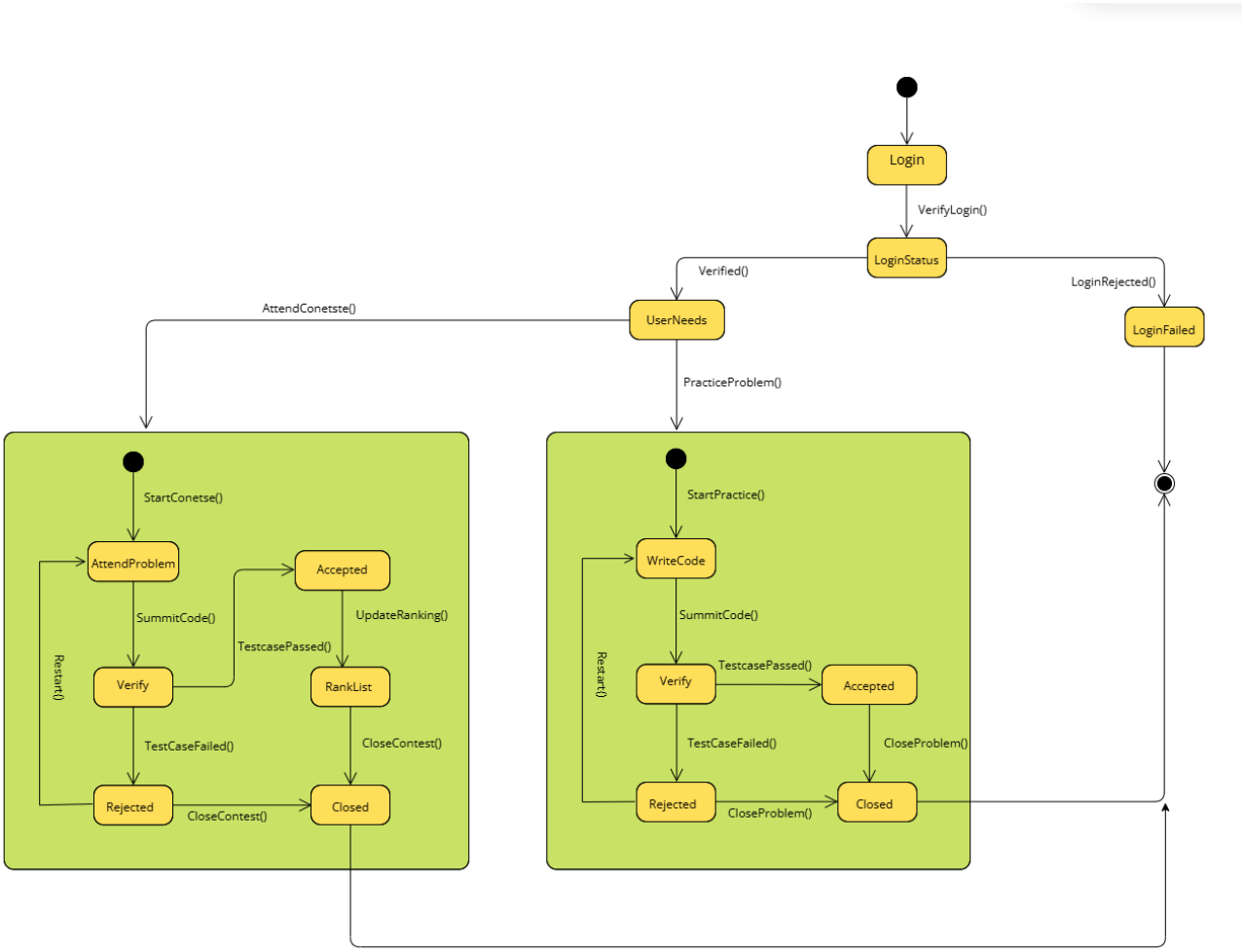


5. Activity diagram

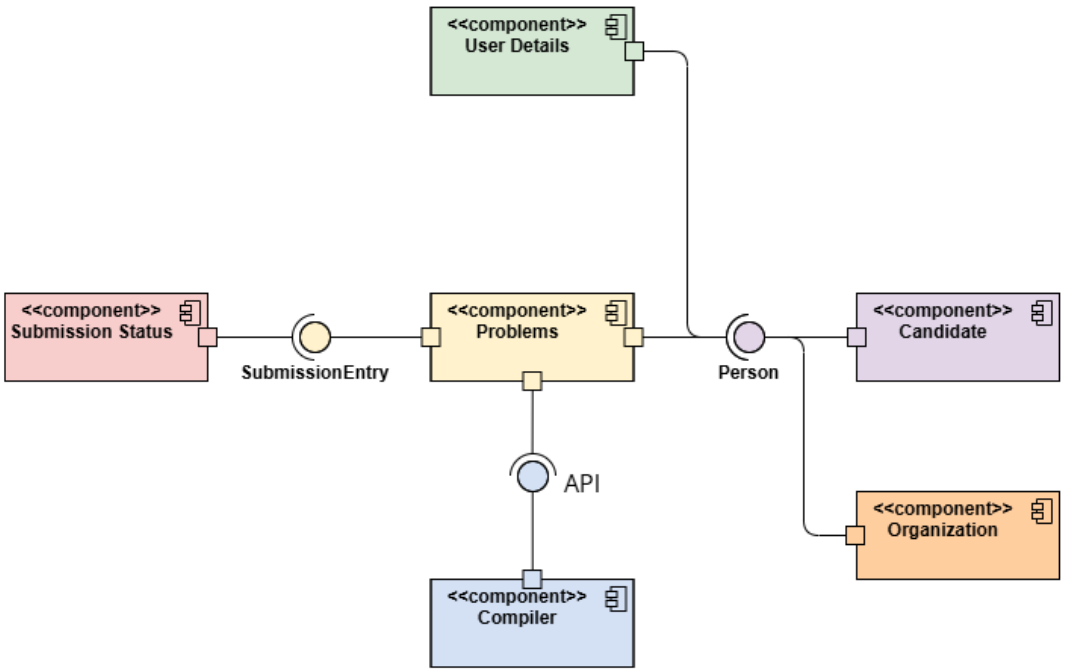




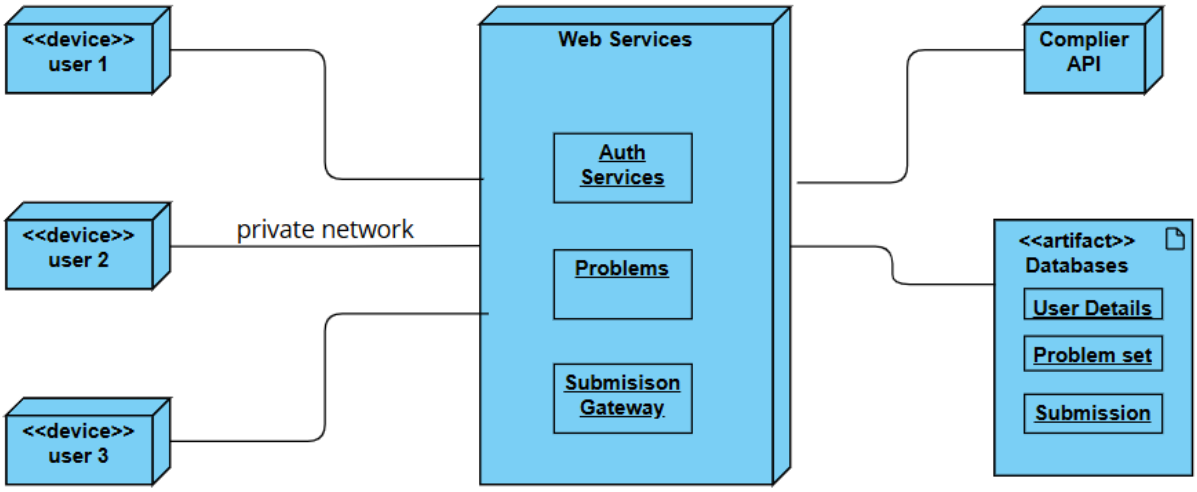
6. State chart diagram



7. Component diagram



8. Deployment diagram



Appendix C: To Be Determined List

- **Maximum Execution Time per Submission** – The exact timeout limit for code execution on Judge0 API.
- **Storage Limit per User** – Maximum number or size of submissions each user can store.
- **Concurrent User Limit** – Exact number of simultaneous users that the system must reliably support.
- **Authentication Methods** – Whether social logins (Google, GitHub) or only email/password will be supported.
- **Advanced Security Measures** – Final decision on features such as 2FA, IP-based restrictions, and submission sanitization.
- **Scoring & Grading Policy** – How submissions are scored (pass/fail, partial credit, percentage, etc.).
- **Future Internationalization** – Languages and regions for potential localization support.
- **Subscription / Monetization Plans** – TBD if platform will implement premium features.
- **Backup and Disaster Recovery Policy** – Exact strategy for database backup frequency and restoration procedure.

SAMPLE CODE

//App.jsx

```
import { Home } from "./pages/Home"
import { NotFound } from "./pages/NotFound"
import Login from "./components/Login"
import SignUp from "./components/SignUp"
import CompanyLogin from "./components/CompanyLogin"
import CompanySignUp from "./components/CompanySignUp"
import { useEffect } from "react"
import { CodeEditorPage } from "./pages/CodeEditorPage"
import PracticePage from "./pages/Practice"
import PracticeQuestions from "./pages/PracticeQuestions"
import { Admin } from "./pages/Admin"
import {  Route, Routes,HashRouter } from "react-router-dom"
import { Profile } from "./pages/Profile"
import { CreateContestPage } from "./pages/CreateContestPage"
import { Contest } from "./pages/Contest"
import { Contact } from "./pages/ContactPage"

function App() {
  useEffect(() => {
    const theme = localStorage.getItem("theme");
    if (theme === "dark") {
      document.documentElement.classList.add("dark");
    } else {
      document.documentElement.classList.remove("dark");
    }
  }, []);

  return (
    <HashRouter>

      <Routes>
        <Route path="/" element={<SignUp />} />
        <Route path="/login" element={<Login />} />
        <Route path="/company-signup" element={<CompanySignUp />} />
        <Route path="/company-login" element={<CompanyLogin />} />
        <Route path="/home" element={<Home />} />
        <Route path="/contest" element={<Contest />} />
        <Route path="/editor" element={<CodeEditorPage />} />
        <Route path="/practice" element={<PracticePage />} />
        <Route path="/practice/:id" element={<PracticeQuestions />} />
        <Route path="/admin" element={<Admin />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="/profile" element={<Profile />} />
        <Route path="/admin/create-contest" element={<CreateContestPage />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </HashRouter>
  )
}
```

```
export default App;
```

//Home.jsx

```
import { Navbar } from "../components/NavBar";
import { ThemeToggle } from "../components/ThemeToggle";
import { Footer } from "../components/Footer";
import { MainSection } from "../components/MainSection";
import { UserProfile } from "../components/userProfile";

export const Home = () => {
  return (
    <div>
      <ThemeToggle />
      <UserProfile />
      <Navbar />
      <MainSection />
      <Footer />
    </div>
  );
};
```

//MainSection.jsx

```
import { useNavigate } from "react-router-dom";
import { Code, Target, Rocket } from "lucide-react";

export const MainSection = () => {
  const navigate = useNavigate();

  const handleGetStarted = () => {
    navigate("/editor");
  };

  return (
    <div className="relative min-h-screen pt-30 flex flex-col items-center justify-center px-6 text-
center gap-10 bg-gradient-to-br ">
      {/* Hero Section */}
      <div>
        <h1 className="text-6xl font-extrabold drop-shadow-md text-indigo-700">
          Code Space
        </h1>
        <h3 className="text-2xl font-semibold mt-3">
          code → submit → repeat
        </h3>
```

```
      <p className="max-w-2xl mt-4 text-lg ">
        Want to master <span className="font-medium text-indigo-600">Data Structures </span>
        and <span className="font-medium text-indigo-600">Algorithms</span>? <br />
        Want to improve your <span className="font-medium text-indigo-600">problem-solving
skills</span>?
        This platform is built for you.
      </p>

      <div className="flex gap-4 justify-center mt-6">
        <button
          onClick={handleGetStarted}
          className="bg-blue-600 hover:bg-blue-700 text-white font-semibold px-6 py-3 rounded-lg
transition duration-300"
        >
          Get Started
        </button>
        <a
          href="#/login"
          className="border border-blue-600 hover:bg-blue-200 text-blue-600 font-semibold px-6
py-3 rounded-lg transition duration-300"
        >
          Already have an account?
        </a>
      </div>
    </div>

    {/* Features Section */}
    <div className="grid grid-cols-1 sm:grid-cols-3 gap-6 max-w-4xl mt-12">
      <div className="p-6 bg-blue-300 rounded-2xl shadow-md hover:shadow-lg transition">
        <Code className="mx-auto text-indigo-600 w-10 h-10 mb-3" />
        <h4 className="text-lg font-semibold text-gray-800">Practice Coding</h4>
        <p className="text-sm text-gray-600 mt-2">
          Solve a wide range of coding challenges from beginner to advanced level.
        </p>
      </div>
      <div className="p-6 bg-blue-300 rounded-2xl shadow-md hover:shadow-lg transition">
        <Target className="mx-auto text-indigo-600 w-10 h-10 mb-3" />
        <h4 className="text-lg font-semibold text-gray-800">Track Progress</h4>
        <p className="text-sm text-gray-600 mt-2">
          Monitor your performance, improve weak areas, and stay motivated.
        </p>
      </div>
      <div className="p-6 bg-blue-300 rounded-2xl shadow-md hover:shadow-lg transition">
        <Rocket className="mx-auto text-indigo-600 w-10 h-10 mb-3" />
        <h4 className="text-lg font-semibold text-gray-800">Level Up</h4>
        <p className="text-sm text-gray-600 mt-2">
          Sharpen your problem-solving skills and get interview ready faster.
        </p>
      </div>
    </div>

    {/* Footer-style CTA */}
    <div className="mt-16 text-center">
      <h2 className="text-2xl font-bold ">
```

```
          Ready to start your coding journey?
        </h2>
        <button
          onClick={handleGetStarted}
          className="mt-4 bg-indigo-600 hover:bg-indigo-700 text-white px-8 py-3 rounded-lg font-
semibold transition"
        >
          Start Practicing
        </button>
      </div>
    </div>
  );
};

//navbar.jsx

import { cn } from "../lib/utils";
import { Menu, X } from "lucide-react";
import { useEffect, useState } from "react";
import { useNavigate, Link } from "react-router-dom";
import { UserProfile } from "./userProfile";
import { ThemeToggle } from "./ThemeToggle";

const navItems = [
  { name: "Home", to: "/home" },
  { name: "Code Editor", to: "/editor" },
  { name: "Practice", to: "/practice" },
  { name: "Contest", to: "/contest" },
  { name: "Contact", to: "/contact" },
];

export const Navbar = () => {
  const [isScrolled, setIsScrolled] = useState(false);
  const [isMenuOpen, setIsMenuOpen] = useState(false);
  const navigate = useNavigate();

  useEffect(() => {
    const storedName = localStorage.getItem("userName");
    if (!storedName) {
      navigate("/login");
    }
  }, [navigate]);

  useEffect(() => {
    const handleScroll = () => {
      setIsScrolled(window.scrollY > 10);
    };

    window.addEventListener("scroll", handleScroll);
    return () => window.removeEventListener("scroll", handleScroll);
  }, []);

  return (
    <nav
```

```
      className={cn(
        "fixed w-full z-40 transition-all duration-300",
        isScrolled
          ? "py-3 bg-background/80 backdrop-blur-md shadow-xs"
          : "py-5"
      )}
    >
      <div className="container flex items-center justify-between">
        {/* Logo */}
        <Link
          to="/home"
          className="text-xl font-bold text-primary flex items-center"
        >
          <span className="relative z-10">
            <span className="text-glow text-foreground">Code</span> Space
          </span>
        </Link>

        {/* Desktop Nav */}
        <div className="hidden md:flex space-x-8 items-center pr-15">
          {navItems.map((item, key) => (
            <Link
              key={key}
              to={item.to}
              className="text-foreground/80 hover:text-primary transition-colors duration-300"
            >
              {item.name}
            </Link>
          ))}
        </div>

        {/* Mobile Menu Button */}
        <button
          onClick={() => setIsMenuOpen((prev) => !prev)}
          className="md:hidden p-2 text-foreground z-50"
          aria-label={isMenuOpen ? "Close Menu" : "Open Menu"}
        >
          {isMenuOpen ? <X size={24} /> : <Menu size={24} />}
        </button>

        {/* Mobile Nav Menu */}
        <div
          className={cn(
            "fixed inset-0 bg-background/95 backdrop-blur-md z-40 flex flex-col items-center justify-
center",
            "transition-all duration-300 md:hidden",
            isMenuOpen
              ? "opacity-100 pointer-events-auto"
              : "opacity-0 pointer-events-none"
          )}
        >
        <div className="flex flex-col space-y-8 text-xl">
          {navItems.map((item, key) => (
            <Link
```

```
            key={key}
            to={item.to}
            className="text-foreground/80 hover:text-primary transition-colors duration-300"
            onClick={() => setIsMenuOpen(false)}
          >
            {item.name}
          </Link>
        ))}

        <button
          onClick={() => {
            navigate("/profile");
            setIsMenuOpen(false);
          }}
          className="text-foreground hover:text-primary transition"
        >
          View Profile
        </button>
        <button
          onClick={() => {
            localStorage.clear();
            navigate("/");
          }}
          className="text-red-500 hover:text-red-600 transition"
        >
          Logout
        </button>
      </div>

      </div>
    </div>
  </nav>
 );
};


//Footer.jsx

import { Link } from "react-router-dom";
import {
  Instagram,
  Linkedin,
  Github,
  Contact2,
  ArrowUp,
} from "lucide-react";

const navItems = [
  { name: "Home", to: "/home" },
  { name: "Code Editor", to: "/editor" },
  { name: "Practice", to: "/practice" },
  { name: "Contest", to: "/contest" },
  { name: "Contact", to: "/contact" },
];
```

```
export const Footer = () => {
  const year = new Date().getFullYear();

  return (
    <footer className="py-12 px-4 bg-card relative border-t border-border mt-12 flex flex-col
items-center gap-6">
      <h2 className="text-2xl font-bold text-primary">CodeSpace</h2>

      {/* Navigation Links */}
      <nav className="flex flex-wrap justify-center gap-6 text-muted-foreground font-medium">
        {navItems.map((item) => (
          <Link
            key={item.name}
            to={item.to}
            className="hover:text-primary transition-colors"
          >
            {item.name}
          </Link>
        ))}
      </nav>

      {/* Admin Login Button */}
      <Link
        to="/admin"
        className="px-4 py-2 bg-primary text-primary-foreground rounded-xl font-medium shadow
hover:bg-primary/90 transition"
      >
        Admin Login
      </Link>

      {/* Copyright */}
      <p className="text-sm text-muted-foreground text-center">
        &copy; {year} <span className="text-primary font-medium">CodeSpace</span> – All rights
reserved.
      </p>
    </footer>
  );
};


//adminDashboard.jsx

import { useEffect, useState } from "react";
import axios from "axios";
import { useNavigate } from "react-router-dom";

const API_URL = import.meta.env.VITE_BACKEND_URL;

export default function AdminDashboard() {
  const navigate = useNavigate();
  const [stats, setStats] = useState({
    users: 0,
    submissions: 0,
```

```
  companies: 0,
  contests: 0,
});

const [testCaseCount, setTestCaseCount] = useState(1);

const [newQuestion, setNewQuestion] = useState({
  title: "",
  description: "",
  constraints: "",
  sampleTestCase: {
    input: "",
    output: "",
    explanation: "",
  },
  topics: [],
  testCases: Array(1).fill({ input: "", expectedOutput: "" }),
});

useEffect(() => {
  axios.get(`${window.API_URL}/api/admin/stats`).then((res) => setStats(res.data));
}, []);

const handleTestCaseCountChange = (e) => {
  const count = parseInt(e.target.value);
  setTestCaseCount(count);
  const updatedTestCases = Array.from({ length: count }, (_, i) =>
    newQuestion.testCases[i] || { input: "", expectedOutput: "" }
  );
  setNewQuestion({ ...newQuestion, testCases: updatedTestCases });
};

const handleQuestionSubmit = async (e) => {
  e.preventDefault();
  try {
    const res = await axios.post(`${window.API_URL}/api/questions`, newQuestion);
    const addedQuestion = res.data;

    alert("✅ Question added successfully");

    console.log("New question ID:", addedQuestion._id);

    setNewQuestion({
      title: "",
      description: "",
      constraints: "",
      sampleTestCase: { input: "", output: "", explanation: "" },
      topics: [],
      testCases: Array(testCaseCount).fill({ input: "", expectedOutput: "" }),
    });
  } catch (err) {
    alert("❌ Failed to add question");
  }
};
```

```
 return (
   <div className="max-w-6xl mx-auto p-20 space-y-8">
     <h1 className="text-3xl font-bold mb-6">Admin Dashboard</h1>



     <div className="grid grid-cols-2 md:grid-cols-4 gap-4">
       <StatCard label="Users" value={stats.users} />
       <StatCard label="Submissions" value={stats.submissions} />
       <StatCard label="Companies" value={stats.companies} />
       <StatCard label="Contests" value={stats.contests} />
     </div>

      {/* Create Contest Button */}
     <button
       onClick={() => navigate("/admin/create-contest")}
       className="bg-blue-600 text-white px-4 py-2 rounded-xl shadow hover:bg-blue-700"
     >
       + Create Contest
     </button>

     <form
       onSubmit={handleQuestionSubmit}
       className="bg-card p-6 rounded-xl shadow space-y-4"
     >
       <h2 className="text-xl font-semibold text-primary">Add New Question</h2>

       <input
         className="w-full p-2 rounded border border-border placeholder-gray-700 dark:placeholder-
gray-400"
         placeholder="Title"
         value={newQuestion.title}
         onChange={(e) => setNewQuestion({ ...newQuestion, title: e.target.value })}
       />

       <textarea
         className="w-full p-2 rounded border border-border placeholder-gray-700 dark:placeholder-
gray-400"
         placeholder="Description"
         value={newQuestion.description}
         onChange={(e) => setNewQuestion({ ...newQuestion, description: e.target.value })}
       ></textarea>

       <textarea
         className="w-full p-2 rounded border border-border placeholder-gray-700 dark:placeholder-
gray-400"
         placeholder="Constraints"
         value={newQuestion.constraints}
         onChange={(e) => setNewQuestion({ ...newQuestion, constraints: e.target.value })}
       ></textarea>

       <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
         <input
```

```
        className="p-2 rounded border border-border placeholder-gray-700 dark:placeholder-gray-
400"
        placeholder="Sample Input"
        value={newQuestion.sampleTestCase.input}
        onChange={(e) =>
          setNewQuestion({
            ...newQuestion,
            sampleTestCase: {
              ...newQuestion.sampleTestCase,
              input: e.target.value,
            },
          })
        }
      />

      <input
        className="p-2 rounded border border-border placeholder-gray-700 dark:placeholder-gray-
400"
        placeholder="Sample Output"
        value={newQuestion.sampleTestCase.output}
        onChange={(e) =>
          setNewQuestion({
            ...newQuestion,
            sampleTestCase: {
              ...newQuestion.sampleTestCase,
              output: e.target.value,
            },
          })
        }
      />

      <input
        className="p-2 rounded border border-border placeholder-gray-700 dark:placeholder-gray-
400"
        placeholder="Explanation"
        value={newQuestion.sampleTestCase.explanation}
        onChange={(e) =>
          setNewQuestion({
            ...newQuestion,
            sampleTestCase: {
              ...newQuestion.sampleTestCase,
              explanation: e.target.value,
            },
          })
        }
      />
    </div>

    <input
      className="w-full p-2 rounded border border-border placeholder-gray-700 dark:placeholder-
gray-400"
      placeholder="Topics (comma separated)"
      value={newQuestion.topics.join(",")}
      onChange={(e) =>
```

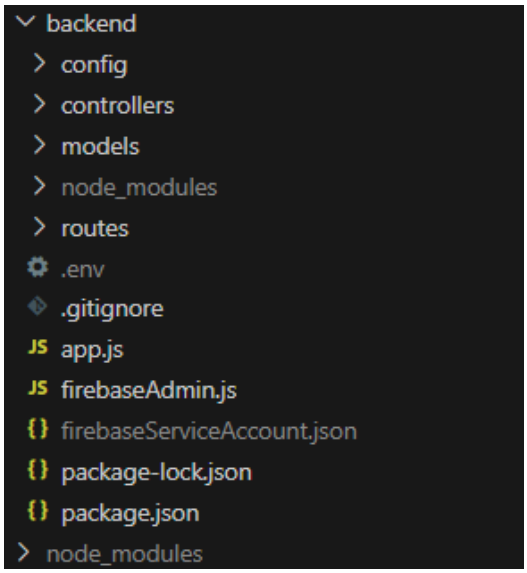
```
          setNewQuestion({
            ...newQuestion,
            topics: e.target.value.split(",").map((t) => t.trim()),
          })
        }
      />

      <label className="block font-medium mt-4 text-gray-700 dark:text-gray-400">Number of
Test Cases</label>
      <select
        className="w-20 p-2 pl-3 pr-8 rounded border border-border bg-background text-gray-400
dark:text-gray-400 relative"
        value={testCaseCount}
        onChange={handleTestCaseCountChange}
      >
        {Array.from({ length: 10 }, (_, i) => i + 1).map((num) => (
          <option key={num} value={num}>{num}</option>
        ))}
      </select>

      <div className="space-y-4">
        {newQuestion.testCases.map((tc, index) => (
          <div key={index} className="grid grid-cols-2 gap-4">
            <input
              className="p-2 rounded border border-border placeholder-gray-700 dark:placeholder-
gray-400"
              placeholder={`Test Input #${index + 1}`}
              value={tc.input}
              onChange={(e) => {
                const updated = [...newQuestion.testCases];
                updated[index].input = e.target.value;
                setNewQuestion({ ...newQuestion, testCases: updated });
              }}
            />
            <input
              className="p-2 rounded border border-border placeholder-gray-700 dark:placeholder-
gray-400"
              placeholder={`Expected Output #${index + 1}`}
              value={tc.expectedOutput}
              onChange={(e) => {
                const updated = [...newQuestion.testCases];
                updated[index].expectedOutput = e.target.value;
                setNewQuestion({ ...newQuestion, testCases: updated });
              }}
            />
          </div>
        ))}
      </div>

      <button type="submit" className="bg-primary text-primary-foreground px-4 py-2 rounded-
xl">
        Add Question
      </button>
    </form>
```

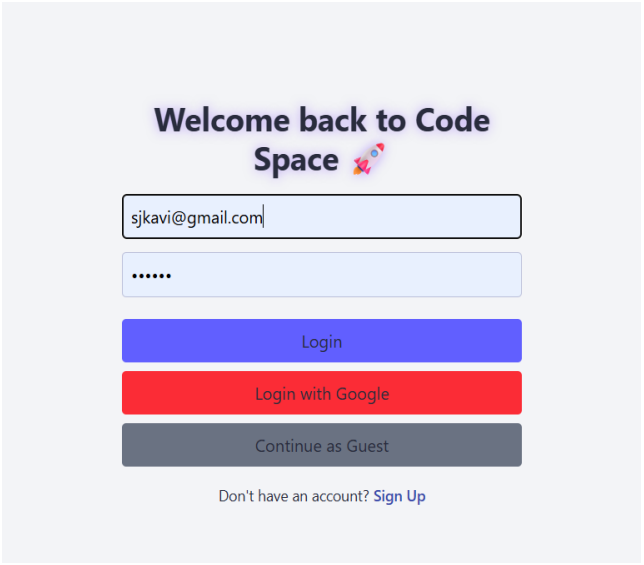
```
    </div>
  );
}

function StatCard({ label, value }) {
  return (
    <div className="bg-card border border-border rounded-xl p-4 text-center shadow">
      <p className="text-sm text-foreground/70">{label}</p>
      <p className="text-2xl font-bold text-primary mt-1">{value}</p>
    </div>
  );
}

//userProfile.jsx

import { useState, useEffect, useRef } from "react";
import { useNavigate } from "react-router-dom";
import { auth } from "../firebase/firebaseConfig";
import { signOut } from "firebase/auth";
import { CircleUserRound } from "lucide-react";

export const UserProfile = () => {
  const [isOpen, setIsOpen] = useState(false);
  const [userName, setUserName] = useState("");
  const dropdownRef = useRef(null);
  const navigate = useNavigate();

  useEffect(() => {
    const name = localStorage.getItem("userName");
    if (name) setUserName(name);
  }, []);

  useEffect(() => {
    const handleClickOutside = (event) => {
      if (dropdownRef.current && !dropdownRef.current.contains(event.target)) {
        setIsOpen(false);
      }
    };
    document.addEventListener("mousedown", handleClickOutside);
    return () => document.removeEventListener("mousedown", handleClickOutside);
  }, []);

  const handleLogout = async () => {
    try {
      await signOut(auth);
    } catch (e) {
      // guest fallback
    }
    localStorage.clear();
    navigate("/");
  };

  return (
    <div
```

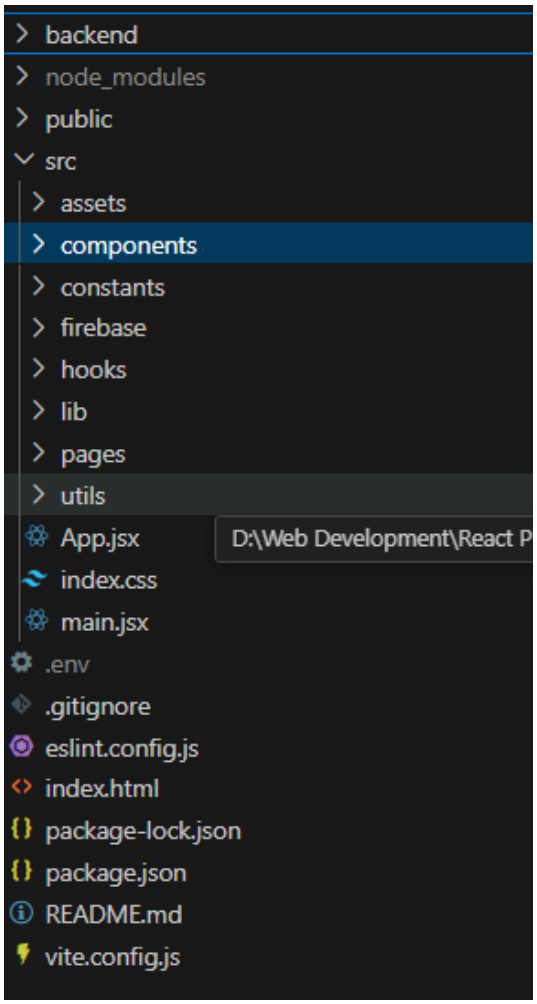
```
    className="fixed top-4 right-16 z-50 max-sm:hidden" // right-16 puts it left of ThemeToggle
(which is right-5)
    ref={dropdownRef}
   >
    <button
     onClick={() => setIsOpen(!isOpen)}
     className="w-10 h-10 rounded-full flex items-center justify-center bg-background/80
dark:bg-muted shadow-md hover:scale-105 transition"
     title={userName}
    >
     <CircleUserRound className="w-7 h-7 text-primary" />
    </button>

    {isOpen && (
     <div className="mt-2  shadow-lg rounded-md py-2 w-40 absolute right-0 text-sm border">
      <button
       onClick={() => {
        navigate("/profile");
        setIsOpen(false);
       }}
       className="w-full text-left px-4 py-2 transition"
      >
       View Profile
      </button>
      <button
       onClick={handleLogout}
       className="w-full text-left px-4 py-2 text-red-500 hover:bg-red-100 dark:hover:bg-red-900
transition"
      >
       Logout
      </button>
     </div>
    )}
   </div>
  );
};
```

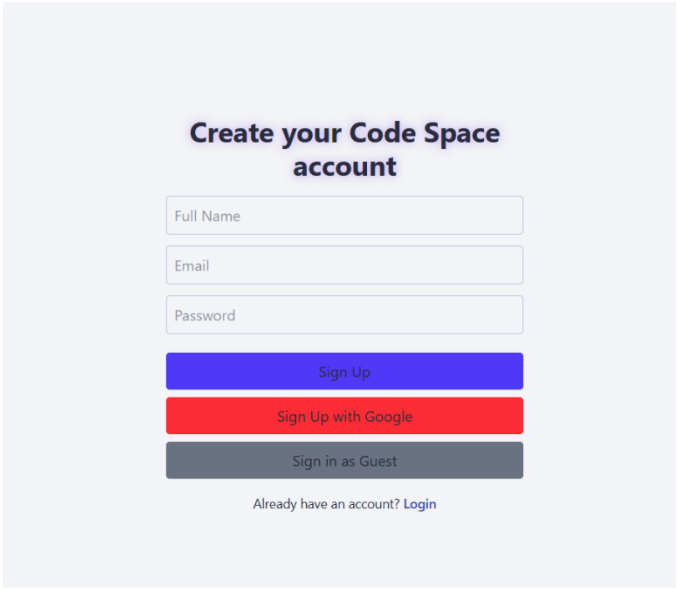
6.1

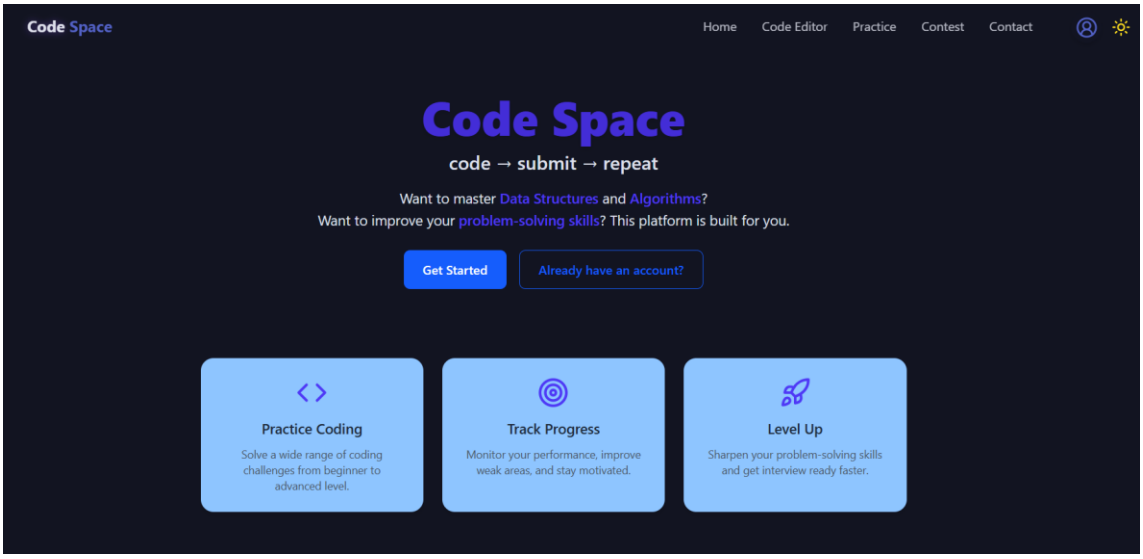
OUTPUT

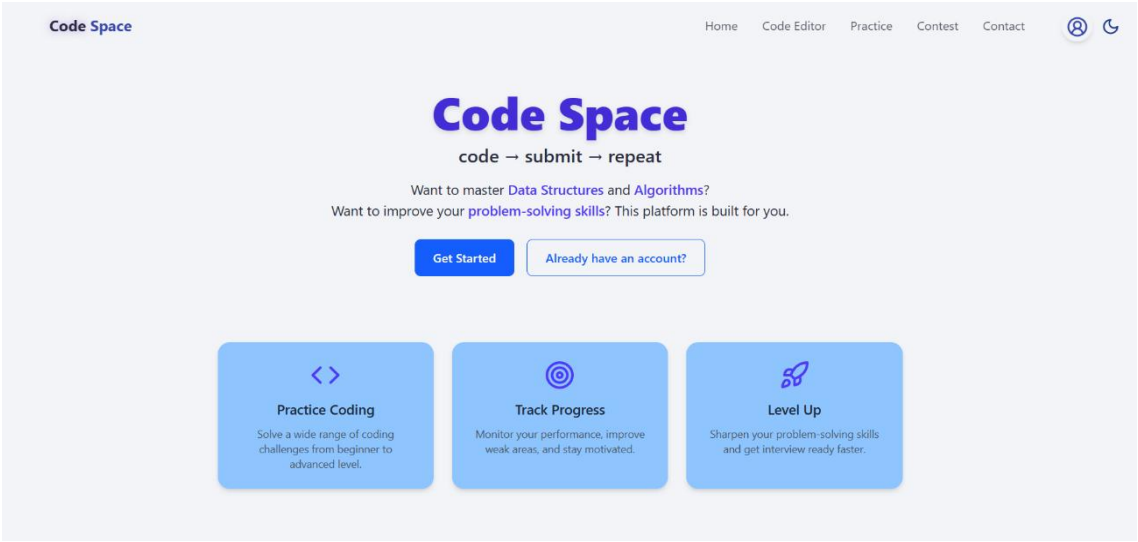

backend
config
controllers
models
node_modules
routes
.env
.gitignore
app.js
firebaseAdmin.js
firebaseServiceAccount.json
package-lock.json
package.json
node_modules

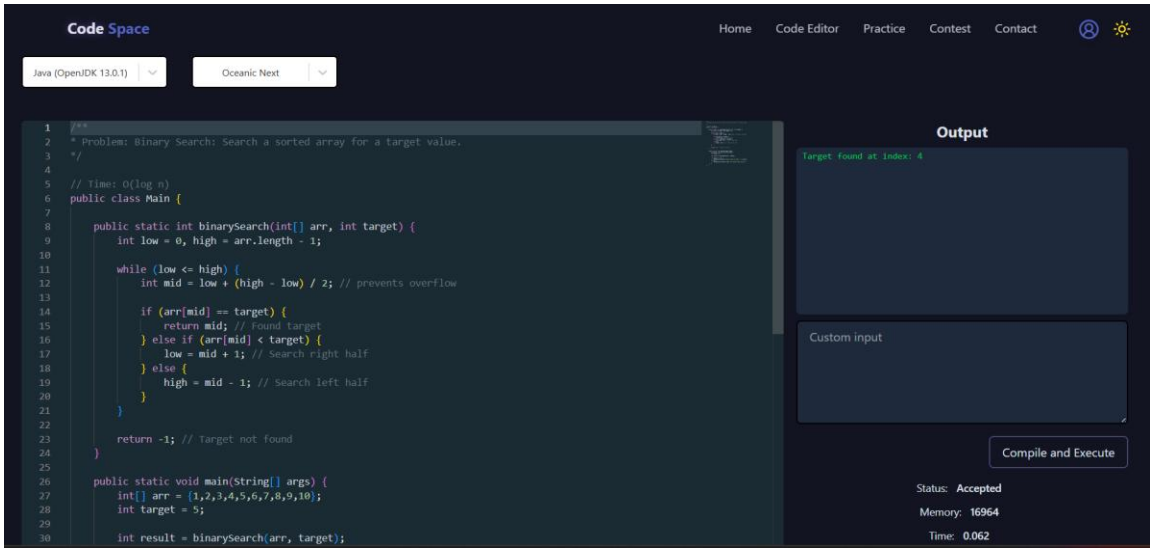

Welcome back to Code Space
sjkavi@gmail.com
Login
Login with Google
Continue as Guest
Don't have an account? Sign Up

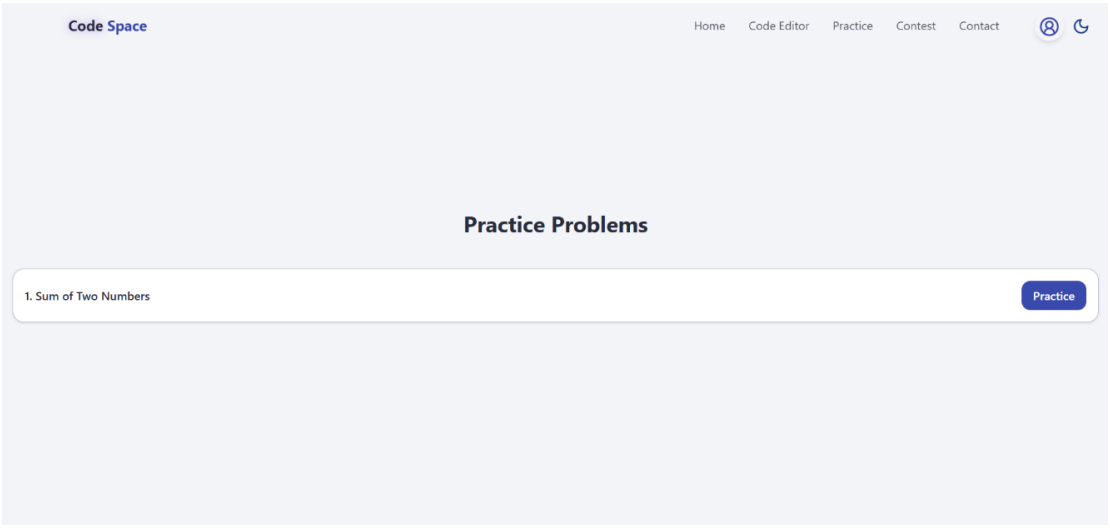

backend
node_modules
public
src
assets
components
constants
firebase
hooks
lib
pages
utils
App.jsx
D:\Web Development\React P
index.css
main.jsx
.env
.gitignore
eslint.config.js
index.html
package-lock.json
package.json
README.md
vite.config.js

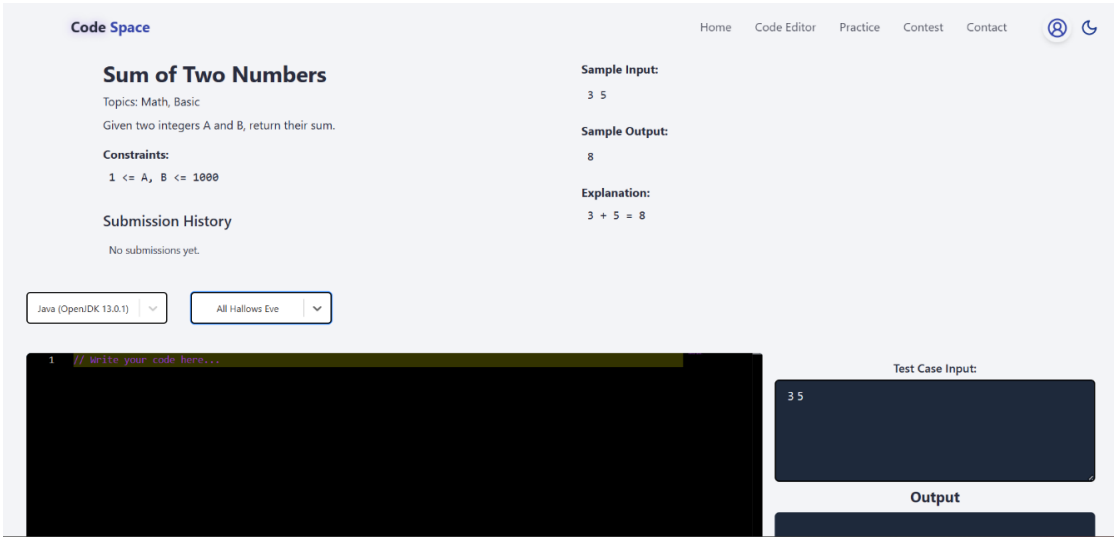

Create your Code Space account
Full Name
Email
Password
Sign Up
Sign Up with Google
Sign in as Guest
Already have an account? Login

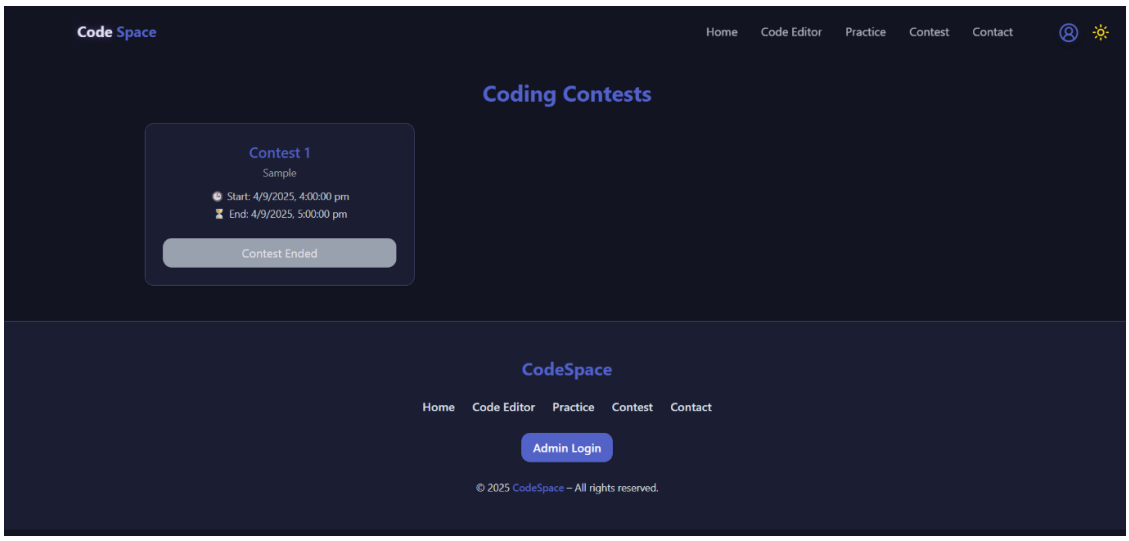

Code Space
Home
Code Editor
Practice
Contest
Contact
Code Space
code → submit → repeat
Want to master Data Structures and Algorithms?
Want to improve your problem-solving skills? This platform is built for you.
Get Started
Already have an account?
Practice Coding
Solve a wide range of coding challenges from beginner to advanced level.
Track Progress
Monitor your performance, improve weak areas, and stay motivated.
Level Up
Sharpen your problem-solving skills and get interview ready faster.

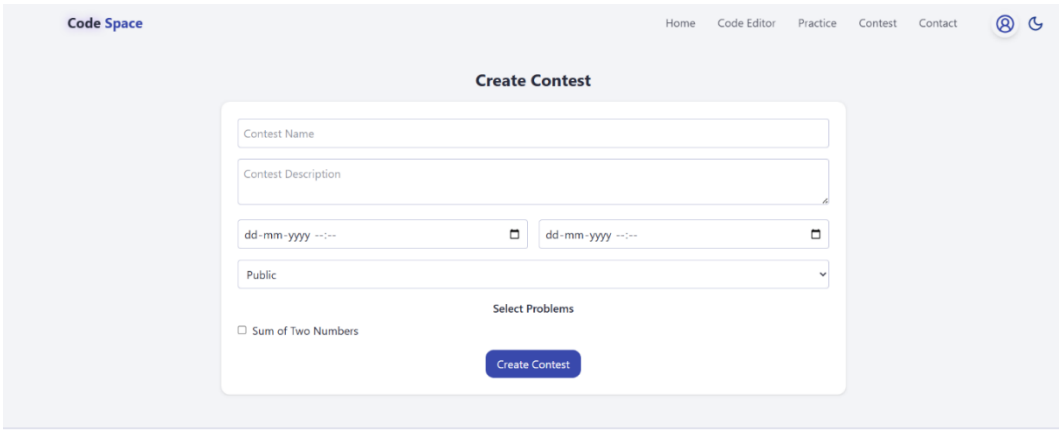

Code Space
Home
Code Editor
Practice
Contest
Contact
Code Space
code → submit → repeat
Want to master Data Structures and Algorithms?
Want to improve your problem-solving skills? This platform is built for you.
Get Started
Already have an account?
Practice Coding
Solve a wide range of coding challenges from beginner to advanced level.
Track Progress
Monitor your performance, improve weak areas, and stay motivated.
Level Up
Sharpen your problem-solving skills and get interview ready faster.


Code Space
Home
Code Editor
Practice
Contest
Contact
Java (OpenJDK 13.0.1)
Oceanic Next
/**
* Problem: Binary Search: Search a sorted array for a target value.
*/
// Time: O(log n)
public class Main {
public static int binarySearch(int[] arr, int target) {
int low = 0, high = arr.length - 1;
while (low <= high) {
int mid = low + (high - low) / 2; // prevents overflow
if (arr[mid] == target) {
return mid; // Found target
} else if (arr[mid] < target) {
low = mid + 1; // Search right half
} else {
high = mid - 1; // Search left half
}
}
return -1; // Target not found
}
public static void main(String[] args) {
int[] arr = {1,2,3,4,5,6,7,8,9,10};
int target = 5;
int result = binarySearch(arr, target);
Output
Target found at index: 4
Custom input
Compile and Execute
Status: Accepted
Memory: 16964
Time: 0.062

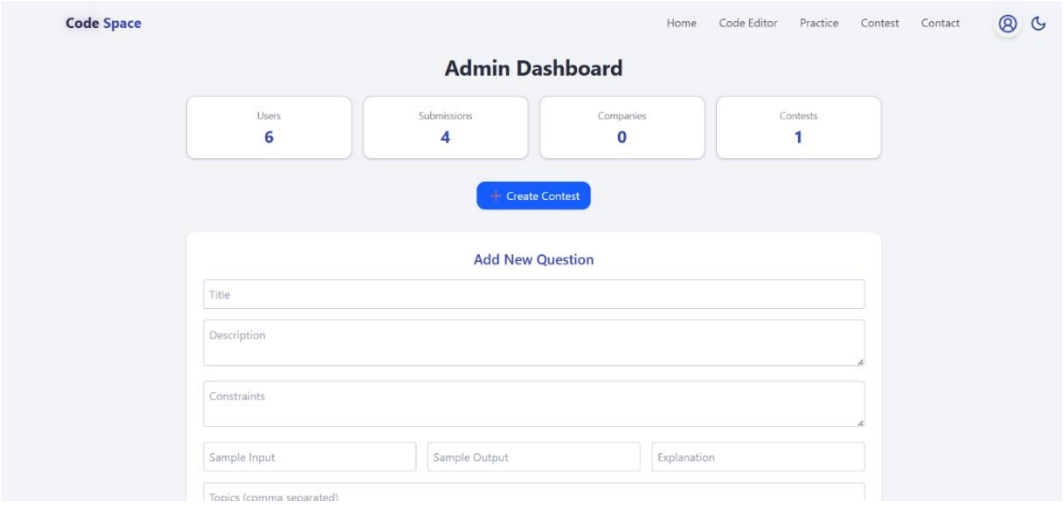

Code Space
Home
Code Editor
Practice
Contest
Contact
Practice Problems
1. Sum of Two Numbers
Practice

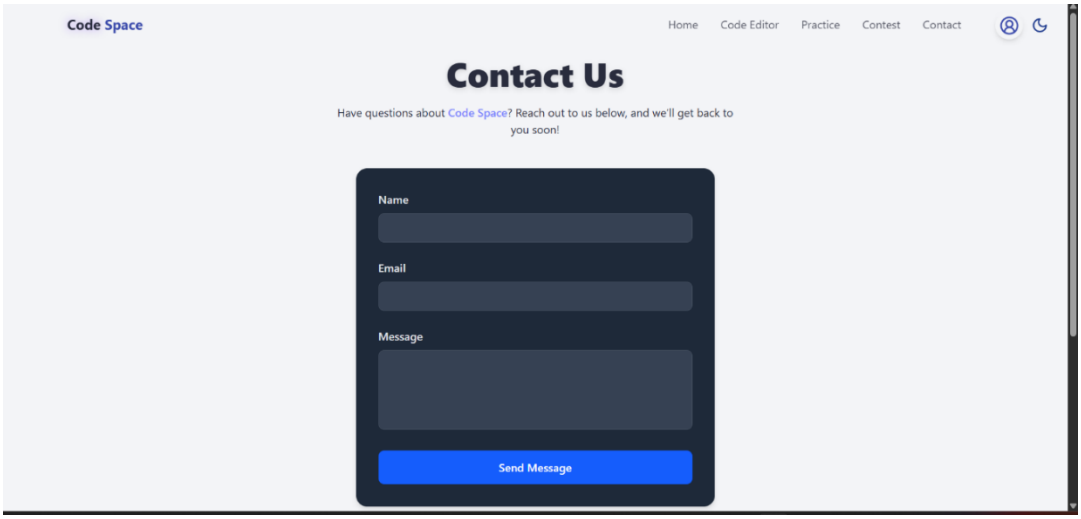

Code Space
Home
Code Editor
Practice
Contest
Contact
Sum of Two Numbers
Topics: Math, Basic
Given two integers A and B, return their sum.
Constraints:
1 <= A, B <= 1000
Submission History
No submissions yet.
Sample Input:
3 5
Sample Output:
8
Explanation:
3 + 5 = 8
Java (OpenJDK 13.0.1)
All Hallows Eve
// Write your code here...
Test Case Input:
3 5
Output


Code Space
Home
Code Editor
Practice
Contest
Contact
Coding Contests
Contest 1
Sample
Start: 4/9/2025, 4:00:00 pm
End: 4/9/2025, 5:00:00 pm
Contest Ended
CodeSpace
Home
Code Editor
Practice
Contest
Contact
Admin Login
© 2025 CodeSpace – All rights reserved.


Code Space
Home
Code Editor
Practice
Contest
Contact
Create Contest
Contest Name
Contest Description
dd-mm-yyyy --:--
dd-mm-yyyy --:--
Public
Select Problems
Sum of Two Numbers
Create Contest

CodeSpace


Code Space
Home
Code Editor
Practice
Contest
Contact
Admin Dashboard
Users
6
Submissions
4
Companies
0
Contests
1
Create Contest
Add New Question
Title
Description
Constraints
Sample Input
Sample Output
Explanation


Code Space
Home
Code Editor
Practice
Contest
Contact
Contact Us
Have questions about Code Space? Reach out to us below, and we'll get back to you soon!
Name
Email
Message
Send Message