DSA VS

Design Document

Team #18

Lawrence Liu

Josh Cho

Tyler Gentry

Joshua Jackson

1. Purpose

DSA VS is an online platform that will transform practicing data structures and algorithms into real time online versus coding matches. Users will solve interview style problems with a limited amount of time to make learning DSA more engaging and interactive, helping users improve their problem solving speed while competing with others at a similar skill level.

1.1 Functional Requirements

1. As a user,

a. I would like to create an account and be able to set a username on DSA.VS.
b. I would like to be able to login using a password on DSA.VS.
c. I would like to be able to change the password connected to my account on DSA.VS.
d. I want the ability to customize my profile for my account.
e. I want the ability to "friend" other users for later contact.
f. I want the ability to "block" other users from communication.
g. I want the ability to message other users.
h. I would like to be able to enter the queue for matchmaking on DSA.VS.
i. I would like to be able to challenge a specific user.
j. I would like to be able to see my current Elo on DSA.VS.
k. I would like to be able to get matched with another user with a similar Elo score in real time.
l. I would like to view the history of my previous matches to review and see how I've improved.
m. I would like the matches to last no longer than 25 minutes and be able to see a timer of my remaining time left.
n. I would like to be able to sort my past problems by category.
o. I would like to be able to see the number of test cases my opponent has passed at any given time during a match on DSA.VS.
p. I would like the code that I have written to execute properly when given test cases.
q. I would like for my Elo to increase or decrease depending on the outcome of a match.
r. I would like a variety of problems within the problem set that are correct and able to be solved within the given time period.
s. I would like to be able to queue for an "AI free for all mode".

t. I would like a working report system to ban users that are abusing the platform.
u. I want the ability to see my statistics for different modes and types of problems.
v. I want the ability to sort and visualize my different stats in different formats.
w. I would like to see how my current Elo stacks up to others with a ranking system through an active leaderboard.
x. I would like to report any issues/bugs I find to ensure they're fixed.
y. I would like to be able to spectate a match (if time allows).
z. I would like a whiteboard-like scratch area to help visualize my solution (if time allows).
aa. I would like the ability to select a preferred language before a match.
bb. After a match I would want the ability to view problem solutions.
cc. I would like the ability to report suspected cheating.
dd. I would want the ability to receive notifications when a new chat pops up, a friend comes online, or I am challenged by a user.
ee. I would like to earn achievements or profile badges when certain milestones are passed.
ff. If I disconnect during a match I would want to be able to reconnect and continue working if the match is still ongoing.
gg. I would like the ability to save my current work if I am disconnected or the page refreshes.
hh. I would like the ability to see the difficulty rating of current problems.
ii. I would want the ability to create private matches with a password

2. As an employer,

a. I would like to see the current level of programmers to be able to gauge the requirements for jobs.
b. I would like to evaluate a student's coding performance without the use of AI.
c. I would like to see the relative focus students have on certain problems.
d. I would like to see what programming languages students excel at.

3. **As a contributor**
    a. I would like to be able to submit problems to the DSA.VS problem set.
    b. I would like to submit individual test cases for a given problem.
    c. I would like the ability to edit or update previous problems submitted.
4. **As an administrator,**
    a. I would like the ability to monitor chats.
    b. I would like the ability to review submitted problem sets and solutions before they are added to the pool.

1.2 Non-Functional Requirements

Security

- All sensitive user data (emails, match results, etc.) should be encrypted to protect privacy and secure data
- Multi-factor authentication should be implemented for 95% to prevent unauthorized access, ensuring user accounts are secure
- User-submitted code should be properly sanitized in order to prevent vulnerabilities like SQL injection
- Sessions should time out after 15 minutes of inactivity to prevent unauthorized access

Architecture

- Code execution should be done in a sandboxed environment to ensure any malicious code has no effect

Performance

- The site should be able to handle up to 20 simultaneous users and matches at a time without lags or crashes
- The platform should have < 1 second of latency to prevent disruptions to the competitive experience
- The system should be able to handle up to 20 concurrent matches or more, allowing performance to remain stable as user demand increases

Usability

- User satisfaction with the interface should be 4.5/5 or higher to allow for easy navigation and coding without issues
- Accessibility features like a screen reader will allow the platform to be used by individuals with disabilities

2 Design Outline

DSA VS will be a client/server based system with real time communication and isolated code execution. The client will handle UI and frontend, while the backend will handle authentication, matchmaking, tracking game state, and code execution. Importantly, code execution will be done with sandboxed workers to ensure security.

2.1 Components:

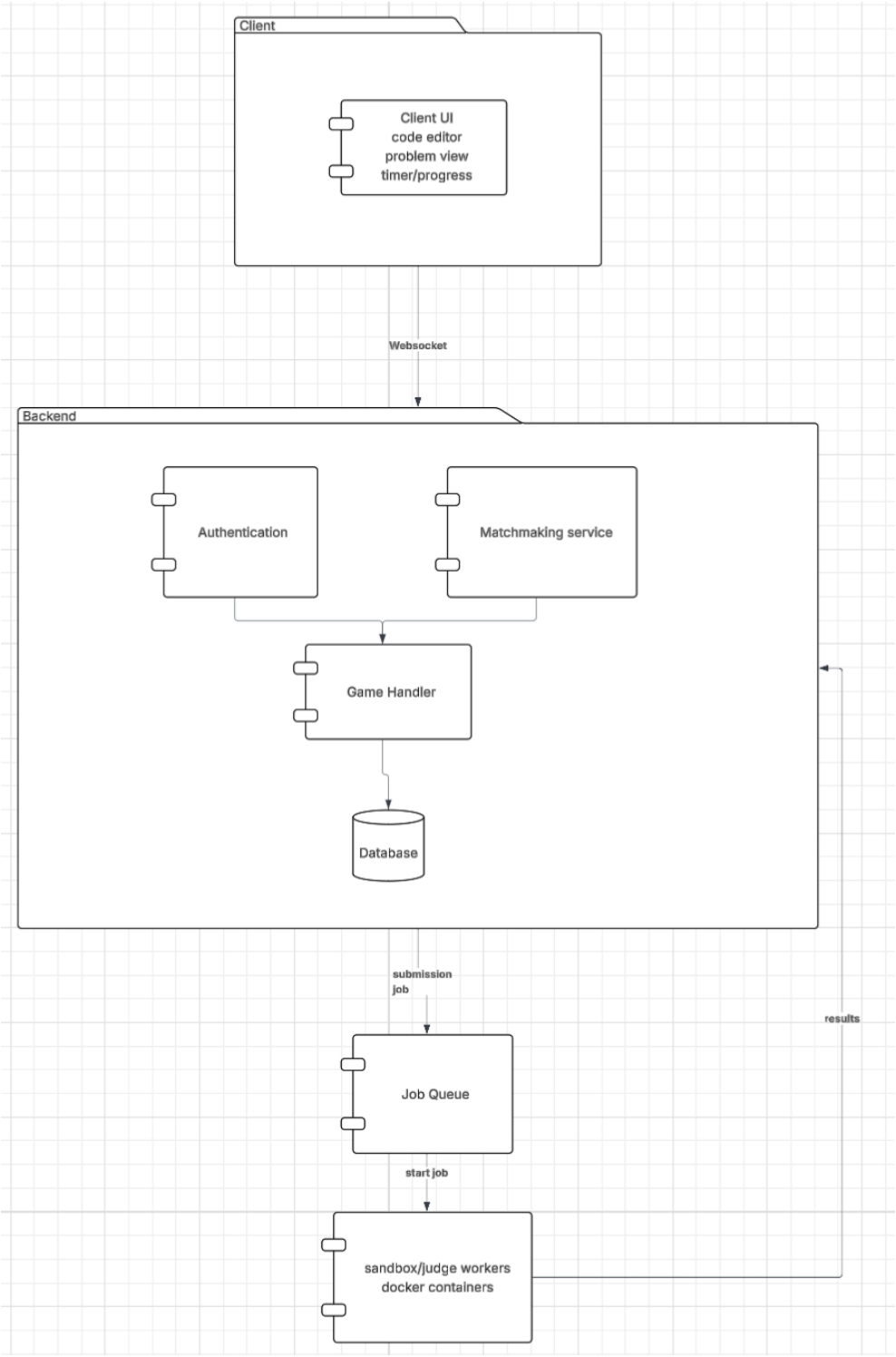
- Client: provides UI for matches
- Backend: executes all app logic
- Authentication: ensures user identity
- Matchmaking service: finds matches
- Game handler: manages active matches
- Database: stores accounts, elo, problems, etc.
- Job queue: prevents code submissions from affecting user experience
- Judge workers: execute and test code securely

Interactions:

- User authentication
    - Client sends login info to the backend, which validates it and creates a new session
- Matchmaking
    - Client requests a match, and the matchmaking service puts them in a queue, where the game handler then initializes a new game
- Match execution
    - The game handler selects a problem and sends data to clients, including the timer and opponent test case completion
- Code submission/evaluation
    - The client submits code to the game handler, which enqueues the submission as a job
    - Sandbox workers pull jobs, execute code, then send test case results back to the game handler

- Match completion
    - Game handler determines the winner, updating the database with the results

2.3 Server Diagram



3 Design Issues

3.1 Functional Issues:

- **Matchmaking and Elo Rating**
    - Solution options:
        - Option 1: Simple queues with nearest Elo match, where a match is based on picking opponents with a fixed close Elo
        - Option 2: An expanding search window over time to prevent long queues
        - Option 3: Multivariable based matchmaking, including Elo, recent performances, and problem categories
    - Our Choice:
        - Option 2: Expanding Elo search, it preserves fairness by prioritizing close-skill matches while preventing long queues when there aren't that many players online. It's also less complex to implement than option c, and will be more flexible than option a, where matches might not be found

- **Real time Match State Tracking**
    - Options
        - Option 1: Polling, where the client asks the server after a set amount of time

    - Option 2: Websockets, with the server pushing updates
    - Option 3: Server-sent events, where the server decides when to update
- Our choice
    - Option 2: We'll be using websockers, as matches require fast updates (syncing timers, checking passed test cases, submissions, full match progress, etc). Websockets allow for bi-directional communication and reduce the traffic required from polling

- Code Execution and Test Case Evaluation

- Solutions
    - Option 1: Run code directly on app server, which is fast but could execute malicious code
    - Option 2: A third-party judge service for code execution

        - Option 3: Isolated code execution through docker containers
    - Our choice
        - Option 3:Container based sandbox environments. Running arbitrary code on our app is risky, whereas a sandboxed judge can isolate execution in the case of malicious code. It also allows us to scale to multiple languages and increase the number of judges in case usage of our app grows.

- **Problem Selection**
    - Solutions
        - Option 1: Pick a random problem from the whole pool
        - Option 2: Pick problems based on Elo (higher Elo = harder problems)
        - Option 3: Personalize problems to avoid recently seen problems and to account for category preferences
    - Our Choice
        - We'll combine options 1 and 2, creating a challenge for users no matter their level, while also allowing them to engage with questions they want to practice on.

3.2 Non-Functional Issues:

What web service should we use?

- Option 1: Proprietary VM ran on Node.js
- Option 2: Docker/Docker Compose
- Option 3: AWS/Kubernetes
- Our Choice:

- Option 2 and 3: We want to use Kubernetes with Docker, however the option of Docker Compose is still on the table
-

What backend language/framework should we use?

- Option 1: C++
- Option 2: Node.js
- Option 3: Go
- Our Choice:
- Option 2: We decided to use a Typescript/Node.js for a backend as it is most well documented

What database should we use?

- Option 1: PostgreSQL
- Option 2: MongoDB
- Option 3: Postgres + Redis
- Our Choice:
- Option 3: We want to use Postgres for its data storage and Redis for fast caching.

What frontend language/framework should we use?

- Option 1:html and JavaScript
- Option 2:React
- Option 3:Qt
- Our Choice:

- We decided to use Qt as it allows interactions with C++ and QML via WebAssembly. It is easier to learn than pure html and JavaScript and has more modularity compared to React.

- **User Account Security**
    - Options
        - Option 1: Store passwords with a simple hash
        - Option 2: Use complex password hashing and reset sessions after inactivity
        - Option 3: Add 2FA by default
    - Our Choice:
        - We'll be going with a strong password hash and offer/encourage 2FA to allow for high security while adding minimal friction to the user experience

- **Latency and Performance**
    - Options
        - Option 1: Host one server for everything
        - Option 2: Split into services (web app, service, judge workers)
        - Option 3: Third party cloud service
    - Our Choice:

        - We’ll split our system into a couple services to prevent code execution from interfering with the match experience, and to allow other components to work apart from the coding experience.

- **Scalability for concurrent matches**
    - Options
        - Option 1: One judge worker handles all submissions
        - Option 2: Make a queue for jobs and add multiple workers
        - Option 3: Add one worker per match
    - Our Choice:
        - We’ll go with a queue with multiple workers, which will allow us to scale based on user load and keep the app running smoothly

4 Design Details

4.1 Classes Outline

Server (Class)

- Sandbox (Class)
    - Containerize user written code on a Docker, send said container to Kubernetes Instance which will run test cases and return amount passed
- Problem set (Class)
    - Basic queries in the database based on account elo etc.
    - Will interact with the sandbox class by
- UI backend (Class) - *basically any server-side stuff that needs to happen when a user clicks something on their screen*
    - Handle page navigation
    - Account authentication
    - Retrieving client side user interactions and routing said interactions to correct code
        - Ex. Player adds friend on client, UI backend class will make sure that the user's intended action is complete
- Account (Class)
    - Some internal identifier/GUID
    - Username
    - Password
    - Profile picture

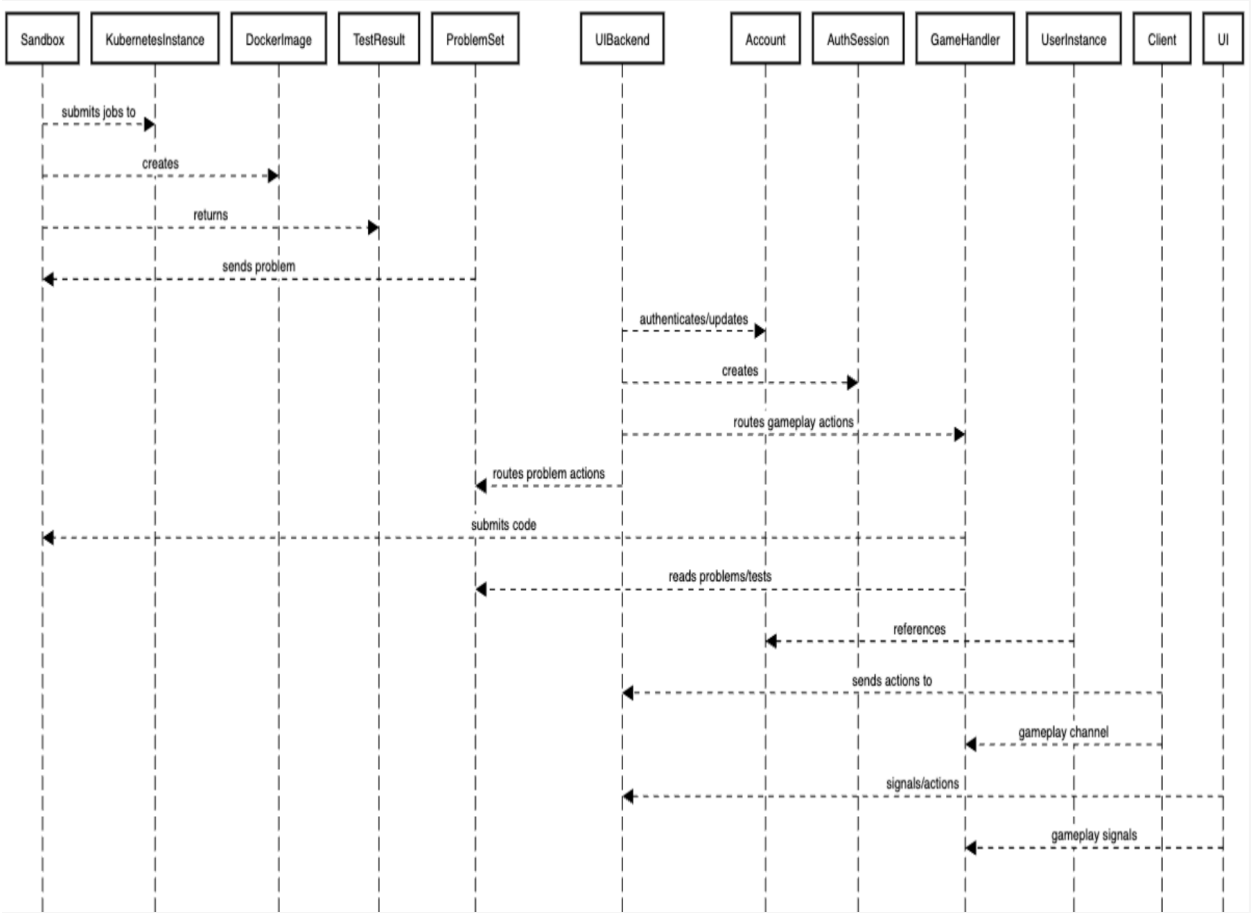
        - ELO
    - Game Handler (Class)
        - Makes sure that the 2 clients versing each other have the same game state
        - 1 or more references to a User Instance
        - User instance authentication statuses
        - Reference to Sandbox class
        - Reference to Problem set class

Client (Class)

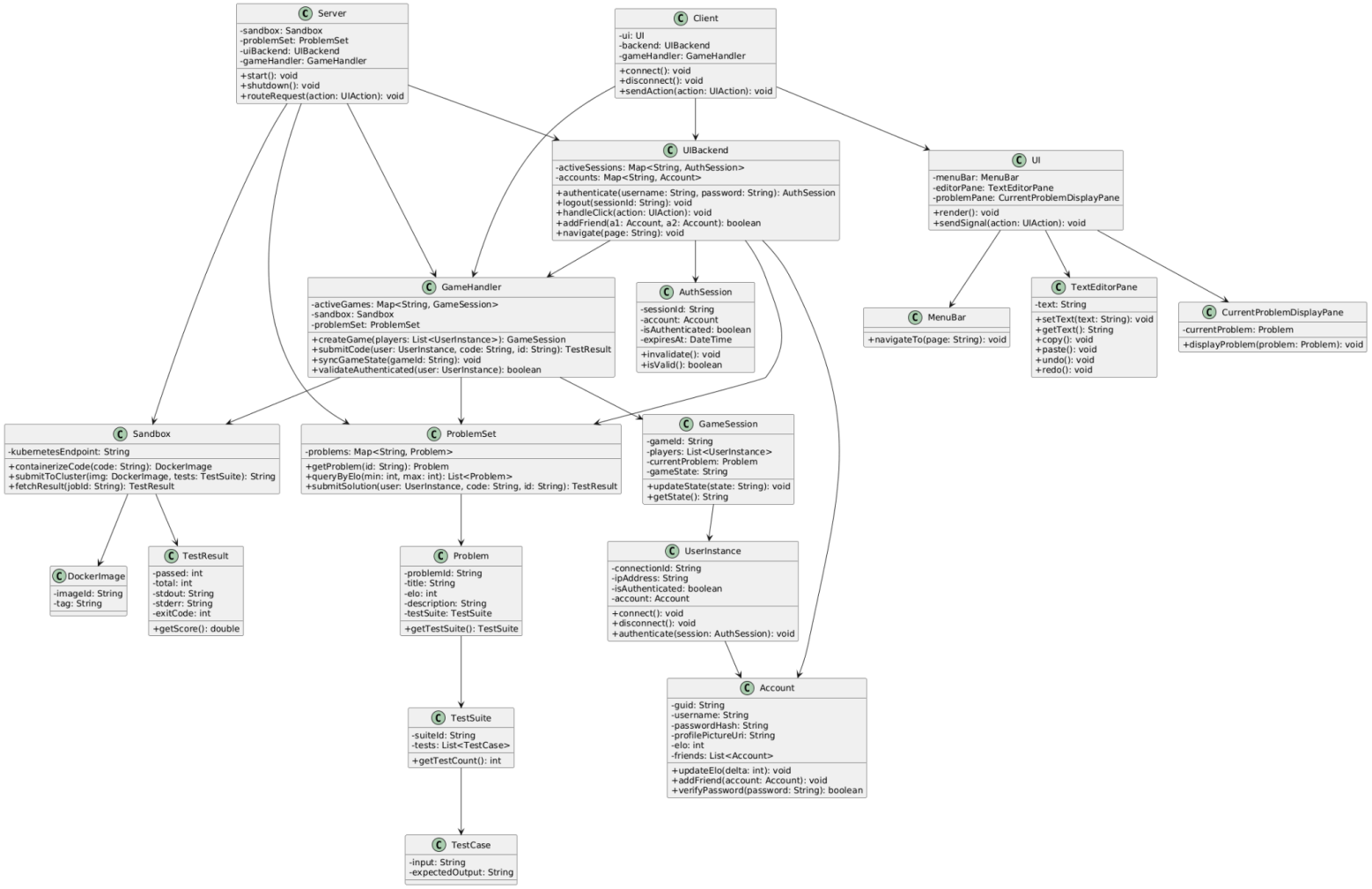
    - UI (Class)
        - Reference to UI backend
        - References to GUI elements (using Qt)
        - Reference to Game Handler
        - Handler functions to send signals to UI backend
        - Menu Bar (Class, GUI element)
            - Global page navigation controls
            - Button to access
        - Text Editor Pane (Class, GUI element)
            - Toolbar for font settings and searching the editor
            - Text input
            - Support for clipboard
            - Undo/redo functionality
        - Current Problem Display Pane (Class, GUI element)
            - Toolbar for zooming and searching the problem contents

        - Problem description
        - Any diagrams/images associated with description
- User instance (Class)
    - Client/server connection details
    - Reference to an Account

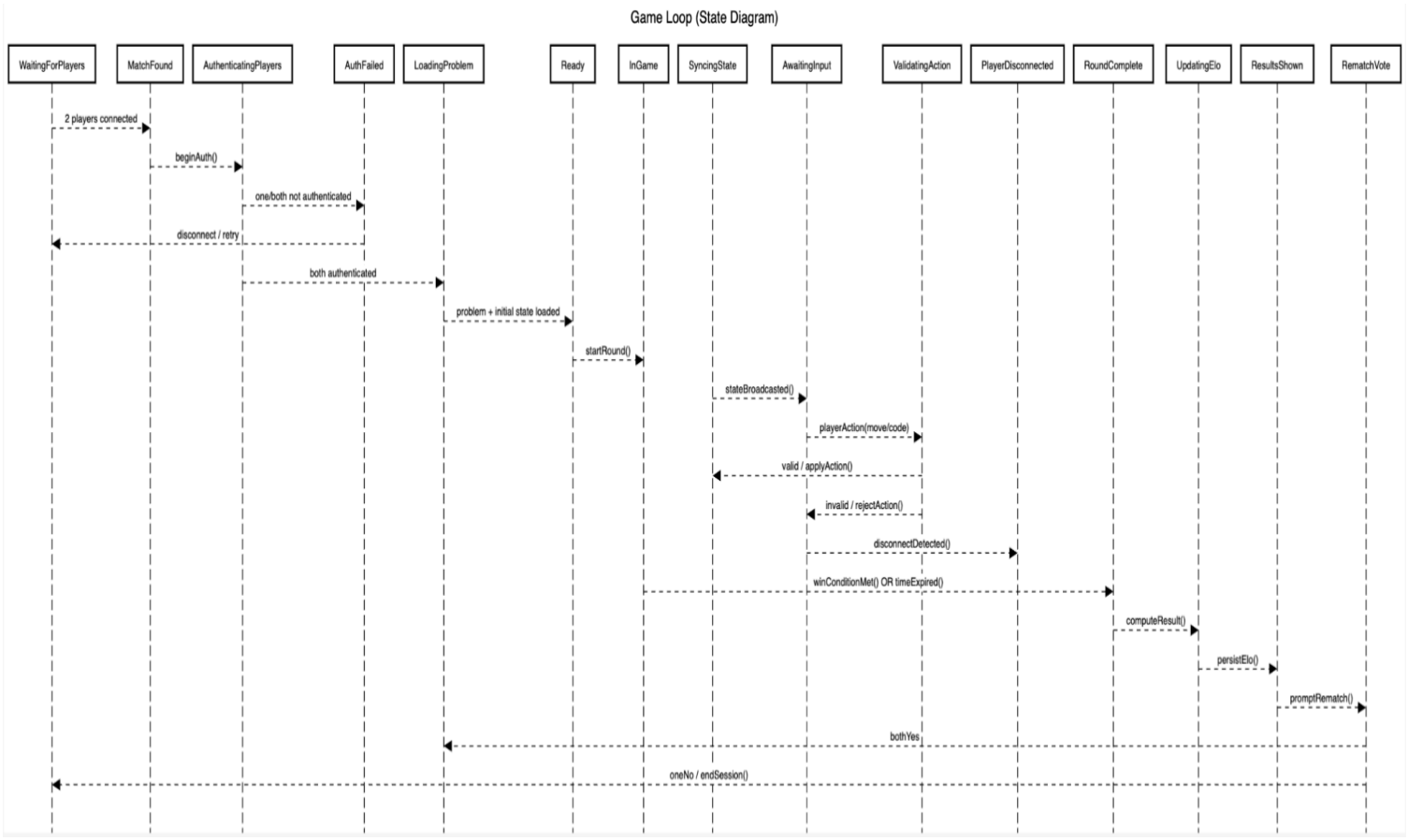
4.2 Sequence Diagram:



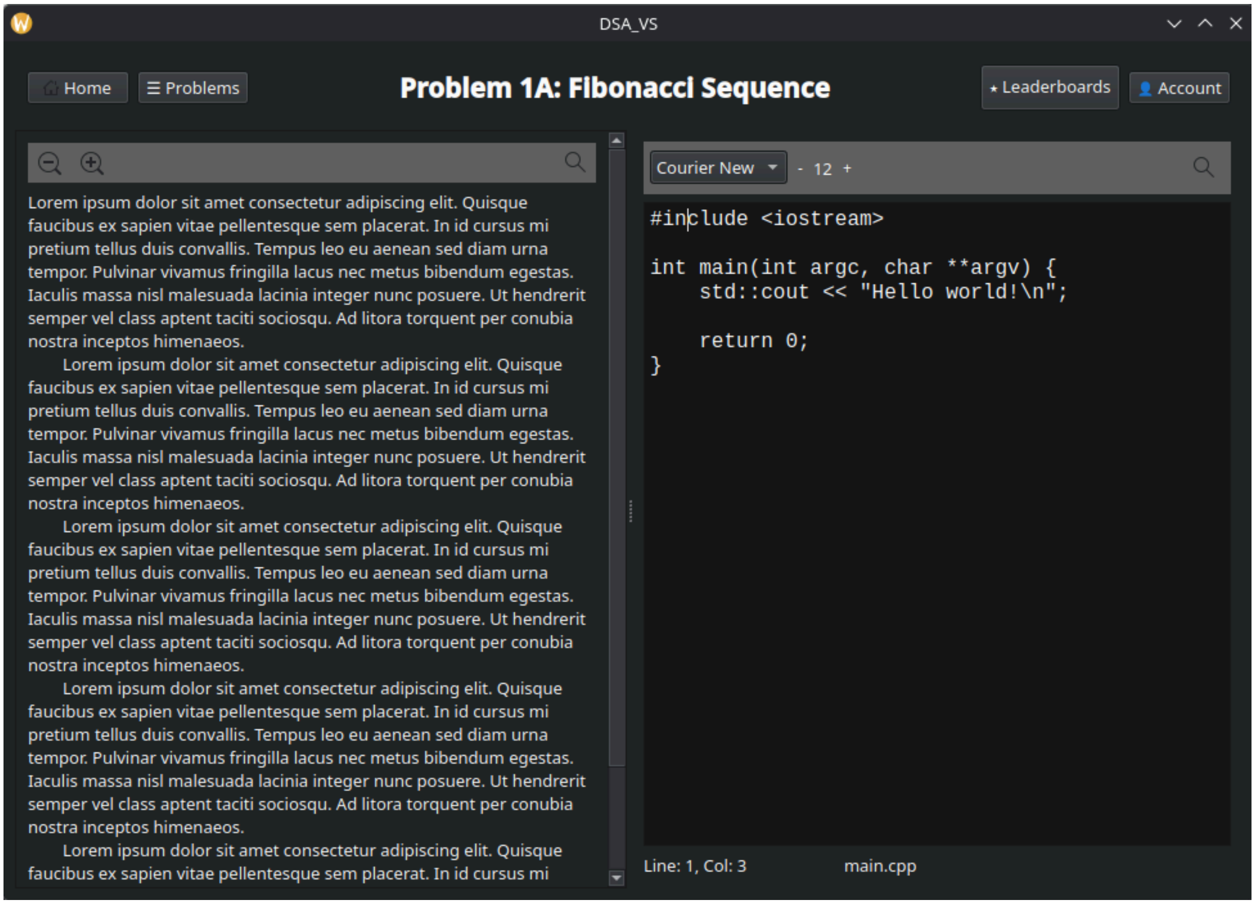
4.3 Class Diagram:



4.4 Game Loop state diagram:



4.5 UI Mockup



This mockup was implemented using the Qt desktop framework, written in C++. Qt allows easy deployment to a variety of platforms, including WebAssembly, which will allow a flexible development environment for our team. The UI consists of three main sections: the Main Toolbar (top), the Problem Pane (left), and the Editor Pane (right).

The Main Toolbar is implemented as a QWidget which holds a QHBoxLayout, which allows the controls at the top of the window to be properly aligned. These buttons will be used for site navigation and account management, as well as social interactions.

The Problem and Editor Panes are contained within a QSplitter object, and each pane has its own QToolBar object with relevant functions. The QSplitter allows the user to resize each

pane to their liking. The Problem Pane consists of a toolbar for text navigation and an area for text and image content. The Editor Pane consists of a toolbar for font settings and an area for code input. This pane also has a small info bar at the bottom, which currently displays the cursor position and filename.