

Codelingo

System Design Document

Frontend CRC.....	1
Backend CRC.....	3
System Architecture.....	4
System Decomposition.....	5
Microservice Architecture.....	5
Microservices Overview.....	5
1. Auth Service.....	5
CREATE Functionality:.....	5
GET Functionality:.....	5
2. User Service.....	6
3. Lesson Service.....	6
4. Match-Making Service.....	6
5. Problem Service.....	6

Frontend CRC

Class Name: LandingPage	
Sub Classes: N/A Parent Class: N/A	
Responsibilities: <ul style="list-style-type: none">- Greets new users with an introduction to the app.- Displays an overview of key features.- Provides Login/Register buttons for authentication.	Collaborators: NavBar

Class Name: Register	
Sub Classes: Parent Class: Login	
<ul style="list-style-type: none">- Allows users to create an account by entering credentials.- Sends data to Authenticator for verification.- Handles validation (password strength, duplicate emails).- Displays success or error messages.	Collaborators: Authenticator

Class Name: Login	
Sub Classes: Register Parent Class: N/A	
<ul style="list-style-type: none">- Authenticates existing users by verifying credentials.- Establishes a cookie-based session.- Handles login errors (incorrect credentials,	Collaborators: User

unverified accounts). - Redirects authenticated users to Dashboard .	
--	--

Class Name:Navbar	
Sub Classes: Parent Class: N/A	
- Provides navigation for authenticated users. - Updates dynamically based on user login status. - Contains links to Dashboard, Lessons, Forum, Match Service, and Logout . - Ensures consistent UI across pages.	Collaborators:

Class Name:Dashboard	
Sub Classes: Parent Class: N/A	
- Acts as the authenticated user's homepage . - Displays lesson progress, match rank, and user details . - Provides access to Lessons, Forum, and Match Service . - Loads personalized content dynamically.	Collaborators: User, Navbar, Login

Class Name: Discussion Board	
Sub Classes: Parent Class: N/A	
- Provides a platform for users to discuss topics. - Allows users to create, view, and respond to discussion threads. - Moderates discussions based on rules and guidelines.	Collaborators: User

Backend CRC

Class Name: User	
Sub Classes: Parent Class: N/A	
- Stores and manages all user account details. - Contains attributes such as email, rank, lesson progress, and unique user ID (UID) . - Provides methods to retrieve and update user details.	Collaborators: User

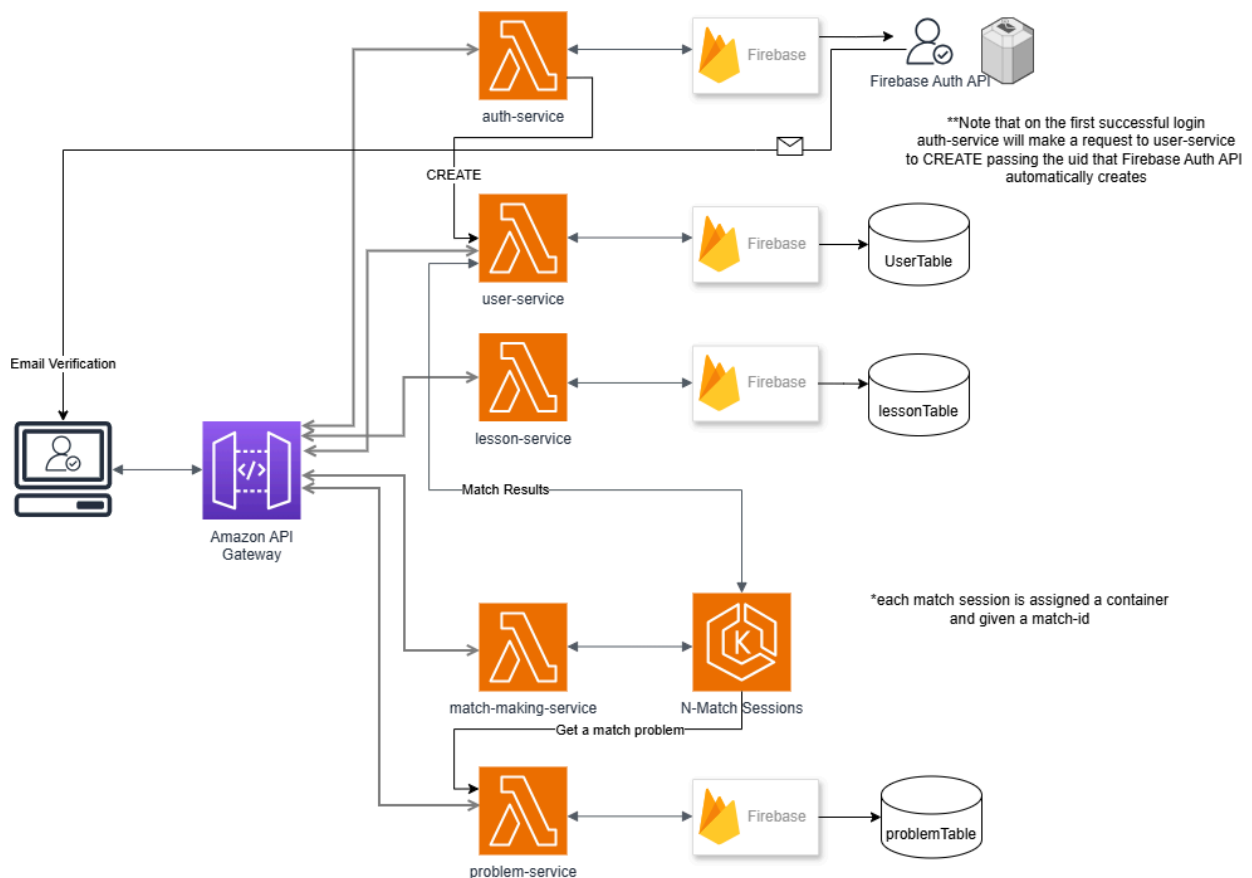
Class Name: Discussion	
Sub Classes: Parent Class: N/A	
- Stores data relating to a specific problem discussion post so Authenticated Users can discuss problems. - Contains attributes such as PID, title, content, upvotes, downvotes, UID .	Collaborators: User, Problem (TBD)

Class Name: Authenticator	
Sub Classes: Parent Class: N/A	
- Handles user authentication and verification. - Stores UID, email, hashed password, verification status, discussion board . - Integrates with Firebase email API for authentication. - Ensures secure login and registration processes.	Collaborators: Firebase email api User

Class Name: Problem	
Sub Classes: Parent Class: N/A	
- includes a description of a problem and its associated tags and difficulties	Collaborators: Firebase, Lesson

Class Name: Lesson	
Sub Classes: Parent Class: N/A	
- includes a description of a lesson along with its title and references to related problems.	Collaborators: Firebase, Problem

System Architecture



System Decomposition

Microservice Architecture

The system follows a **microservice architecture**, utilizing **five (5) Node.js APIs** hosted on **AWS Lambda** serverless functions. Services requiring persistent data interact with the **Firebase API** using **PUT**, **GET**, **DELETE**, and **UPDATE** operations. The **presentation layer** is decoupled and delivered to users via **Amazon API Gateway**.

Microservices Overview

1. Auth Service

A basic API handling authentication operations.

CREATE Functionality:

- Registers a new user if the email is unique using the **Firebase Auth API**.
- Sends a **verification email** containing a link.
- When a **GET** request is received on the verification link's endpoint, the user is marked as verified.

- Upon successful registration, the **auth service** makes a **CREATE** request to the **user-service**, passing the **uid** (generated by Firebase Auth) and email.

Successful registration: User is created and a verification email is sent.

Error (Non-unique email): Displays a popup message:

- *"This email is associated with an existing account."*

GET Functionality:

- Logs in users using a valid email and password.
- Sends a **GET** request to the **user-service** with the **uid** to retrieve account details.

Successful login: User data is retrieved for display.

Error (Invalid credentials): Displays a popup message:

- *"The email or password was incorrect. Try again or reset your password."*
 - **Password Reset:** Handled externally via Firebase Auth API.
-

2. User Service

A basic API managing user records.

- **CREATE:** Adds a user record to the **Firestore Users table**.
 - **GET:** Retrieves user data based on **uid**.
 - You can also specify a post id to include a specific post **pid**.
 - **UPDATE:** Modifies user data based on **uid**.
-

3. Lesson Service

Handles lesson-related operations.

- **GET:** Retrieves lesson data based on **lid** (lesson ID).
-

4. Match-Making Service

Facilitates user match making for sessions.

- **POST:**

- Accepts a `uid`.
 - Requests user data.
 - Finds a valid match.
 - Hosts the match session on a **K-Match server**, which establishes connections between clients and the **problem-service**.
-

5. Problem Service

Generates and provides problem-based challenges.

- **GET:**
 - Accepts a `problem` type (`enum`).
 - Accepts a `difficulty rating` (1, 2, or 3).
 - Returns a **seeded random problem** based on the provided parameters.