High Level Design

Overview

A web-based Online Judge application that allows users to submit code, get it evaluated automatically, and view results.

Database Design :

Collection 1 : **users**

Document Structure :

_id: ObjectId
username:String
passwordHash:String
role:user | admin
joinedAt:Date

Collection 2 :**problems**

Document Structure :

_id:ObjectId
title:String
description:String
difficulty:Easy | Medium |Hard
tags:String **type**:rated |practice |contest **points**:Number
contestId:ObjectId (ref:contests,optional)
testCases:[{ **input**:String
expectedOutput:String }]
createdBy:ObjectId (ref:users)
createdAt:Date

Collection 3 :**submissions**

Document Structure :

_id:ObjectId
userId:ObjectId (ref:users)
problemId :ObjectId(ref:problems)
code :String
contestId:ObjectId(ref:contests,optional)
language:String //c,cpp,etc **verdict**:String **score**:Number
status:Pending |Running |Evaluated **submittedAt** :Date

Collection 4:**contests**

Document Structure :

_id:ObjectId
name:String
startTime:Date
endTime:Date
problems:[ObjectId(ref:problems)]
participants:[ObjectId(ref:users)]

User Interface :

Screen 1 : Home Screen(Landing page)

- Information about platform
- Sign up/Login button

Screen 2 : Authentication Screens

- Log in Screen:
    - form for email and password
- Sign up Screen :
    - Registration form

Screen 3 :Dashboard Screen
Serves as user's main landing page after Login

- view/edit their profile
- Access practice problems
- List and join contests
- Track progress and see rated problems

Screen 4:Specific Problem Screen

- Problem info section(title,description,constraints,sample input/expectedOutput)
- Code editor(language selector,editor,buttons for run and submit)
- Output / verdict section

Screen 5: Global leaderboard

- Shows users ranked based on their cumulative performance on rated problems across the platform

Screen 6 :Contest leaderboard

- Shows participants ranked within a specific contest.

Route Design

- **api/auth/signup** : register a new user
- **api/auth/login** : login existing user

- **api/user/me** :get user profile
- **api/problems/practice** :get all practice problems
- **api/problems/rated** : get all rated problems
- **api/contests/:contestId/problems** : get all problems from a specific contest
- **api/problems/:id**:fetches individual problem details
- **api/submissions/run**: run code with sample inputs
- **api/submissions/submit** :submit solution for evaluation
- **api/leaderboard/global**:fetch global leaderboard
- **api/leaderboard/contest/:contestId**:fetch contest specific leaderboard

Controllers Design

1. authController :

- **signup(req, res)**:Registers a new user, hashes password, stores user.
- **login(req,res)**: Validates credentials, returns JWT token.

2. userController:

- **getProfile(req, res)**: Returns current logged-in user's profile.
- **updateProfile(req, res)** : (optional)Updates user info.

3. problemController:

- **getPracticeProblems(req, res)**:Returns all practice problems.
- **getRatedProblems(req, res)**:Returns all rated problems.
- **getProblemById(req, res)**:Returns full problem details.
- **getContestProblems(req, res)**:Returns problems tied to a contest.

4.submissionController:

- **runCode(req, res)**: Executes code with sample input and returns output.
- **submitSolution(req, res)**: Evaluates submission, stores verdict and score.

5.leaderboardController:

- **getGlobalLeaderboard(req, res)**: Returns users ranked by rated problem performance.
- **getContestLeaderboard(req, res)**:Returns contest-specific rankings.

Application workflow :

1.User Signup/Login:

- user visits landing page->clicks signup/login
- fills the respective form and submits it

Backend:

- Hashes password(signup) using bcrypt
- Validates credentials (login)
- Returns JWT token on success

- **Frontend** stores token for authenticated requests

2. Dashboard Access :

- After login, user is redirected to Dashboard
- **Frontend** sends token to /api/user/me
- **Backend** verifies JWT and returns profile data
- User can now :
    - View/edit profile
    - Access practice/rated problems
    - Join contests

3.Viewing a Problem:

- User selects a problem (from any category)
- **Frontend** calls /api/problems/:id
- **Backend** fetches and returns full problem details
- User sees:
    - Title, description, sample input/output
    - Language selector and code editor

4. Code Submission (Run or Submit):

User clicks run:

- **Frontend** sends code and language to /api/submissions/run
- **Backend**:
    - Accepts code + sample input
    - Sends an execution job to the job queue with:
        - Code
        - Sample input
        - Language
        - type: run (not submission)

    Worker service picks up the job
    - Spins up a Docker container for the specified language
    - Runs the code with sample input
    - Captures output/errors

Worker returns result to the main server

Backend sends output back to the frontend

User clicks Submit:

- **Frontend** sends code, language, and problem ID to /api/submissions/submit
- **Backend**:
    - Creates a new submission document in DB with:
    - status: Pending
    - Adds a submission job to the queue with:
      submissionId, problemId, userId, language, code

Worker service (in the background):

- Retrieves the submission and corresponding test cases
- Spins up a Docker container to run the code for each test case
- Collects verdicts, score, runtime, and errors
- Updates submission document:

  verdict, score, status: Evaluated, submittedAt

Frontend is notified (e.g., via WebSocket) when submission is evaluated and verdict is ready.

5. Leaderboard:

- **Global**: **frontend** calls /api/leaderboard/global
  Backend aggregates top users from rated problems
- **Contest**: **frontend** calls /api/leaderboard/contest/:id

  Backend aggregates scores from that contest's submissions