HLD (High Level Design) for Online Coding Judge

ABOUT THE PROJECT

This project aims to develop a web-based online coding judge that allows users to solve programming challenges, submit their code, and receive instant feedback. The application includes user authentication, code submission and evaluation, leaderboards, and various user-specific functionalities. It is built using the MERN stack and follows best practices for design and security.

FRONTEND (React.js)

1. Home Page
- Purpose: To provide an overview of the coding judge and navigation to other sections.
- Description: The home page includes a welcome message, featured coding challenges, and links to login or register.
2. Login/Register Page
- Purpose: To authenticate users and allow new users to create accounts.
- Description: The page includes fields for username, email, and password, with options to login or create a new account.
3. Dashboard
- Purpose: To display user-specific data and quick access to main features.
- Description: The dashboard shows an overview of the user's solved problems, current rank, and recent submissions.
4. Problem List
- Purpose: To display available coding challenges.
- Description: A list of coding problems with difficulty levels, success rates, and tags.
5. Problem Page
- Purpose: To show problem details and allow code submission.
- Description: Displays problem statement, input/output examples, and provides a code editor for submission.
6. Submissions Page
- Purpose: To show user's submission history and results.
- Description: Lists all submissions with their status (Accepted, Wrong Answer, Time Limit Exceeded, etc.).
7. Leaderboard
- Purpose: To display top performers.
- Description: Shows rankings based on problems solved and efficiency.

BACKEND (Node.js & Express.js)

REST APIs

1. /api/auth • POST /register: Register a new user • POST /login: Authenticate a user
2. /api/problems • GET /: Retrieve list of problems • GET /:id: Retrieve a specific problem • POST /: (Admin) Create a new problem
3. /api/submissions • POST /: Submit a solution • GET /: Retrieve user's submissions
4. /api/leaderboard • GET /: Retrieve leaderboard data
5. /api/user • GET /profile: Retrieve user profile • PUT /profile: Update user profile

DATABASE (MongoDB)

Collections:

1. Users • Fields: id, username, email, passwordHash, solvedProblems, submissions, createdAt, updatedAt

2. Problems • Fields: id, title, description, difficulty, testCases, solutionCode, tags, createdAt, updatedAt
3. Submissions • Fields: id, userId, problemId, code, language, status, executionTime, memory, submittedAt
4. Leaderboard • Fields: userId, score, rank, updatedAt

DOCKER

Overview Docker is used to containerize the application components, ensuring consistency across different environments and simplifying deployment.

Containers:

1. Frontend Container (React.js)
2. Backend Container (Node.js & Express.js)
3. Database Container (MongoDB)
4. Judge Container (for code execution and evaluation)

Security Considerations:

1. Isolated Execution Environment: • Use separate containers for code execution to prevent malicious code from affecting the main system.
2. Resource Limitations: • Implement strict resource limits (CPU, memory, network) for the judge container to prevent DoS attacks.
3. Input Sanitization: • Thoroughly sanitize and validate all user inputs, especially submitted code.
4. Secure Communication: • Use HTTPS for all API communications. • Implement proper authentication and authorization for API endpoints.
5. Regular Updates: • Keep all components (Node.js, MongoDB, React.js) and dependencies up to date to patch known vulnerabilities.

JUDGE SYSTEM

The judge system is a crucial component that securely executes submitted code and evaluates it against test cases.

Key Features:

1. Support for multiple programming languages
2. Secure code execution in isolated environments
3. Time and memory limit enforcement
4. Comparison of output with expected results
5. Detailed feedback on code performance and correctness

ANTI-CHEATING PROTOCOLS

Purpose: To maintain the integrity of coding contests by detecting and flagging potential cheating attempts.

Key Features:

1. Tab Switch Detection
2. Copy-Paste Monitoring
3. Screen Recording Prevention
4. Browser Focus Tracking

5. Automated Plagiarism Check

Implementation Details:

1. Tab Switch Detection Function: Detect when a user switches away from the contest tab. Implementation: • Use browser's Page Visibility API to detect when the page loses focus. • Record the number and duration of tab switches. Challenges: • Distinguishing between intentional cheating and accidental tab switches. • Handling multi-monitor setups where the contest tab might not always be in focus.
2. Copy-Paste Monitoring Function: Track and limit copy-paste actions within the coding environment. Implementation: • Monitor clipboard events (copy, cut, paste) within the code editor. • Set limits on the number of allowed paste actions. • Flag suspicious patterns of copy-paste, especially large blocks of code. Challenges: • Balancing between preventing cheating and allowing legitimate use of copy-paste for coding efficiency. • Handling browser extensions that might interfere with clipboard monitoring.
3. Screen Recording Prevention Function: Detect and prevent screen recording attempts. Implementation: • Use browser APIs to detect if screen sharing or recording is active. • Warn users and potentially end the contest session if screen recording is detected. Challenges: • Limited ability to detect all forms of screen recording, especially hardware-based solutions. • Potential false positives with certain browser configurations or OS settings.
4. Browser Focus Tracking Function: Ensure the user remains engaged with the contest environment. Implementation: • Use JavaScript events to track mouse movements and keyboard inputs. • Flag prolonged periods of inactivity or unusual activity patterns. Challenges: • Distinguishing between thinking time and potential cheating attempts. • Accommodating different coding styles and speeds.
5. Automated Plagiarism Check Function: Detect code similarities with existing solutions or between contestants. Implementation: • Use code similarity algorithms to compare submitted solutions with a database of known solutions and other contestants' submissions. • Flag submissions with suspiciously high similarity scores for manual review. Challenges: • Differentiating between common coding patterns and actual plagiarism. • Handling cases where similar solutions might be coincidental or expected.

FLAGGING AND REPORTING SYSTEM

When potential cheating is detected:

1. Log the incident with details (timestamp, type of violation, relevant data).
2. Implement a strike system, where multiple minor violations or a single major violation triggers a review.
3. Provide warnings to the user for minor violations.
4. For severe or repeated violations, automatically end the contest session and mark the submission for manual review.

API Additions:

1. POST /api/contest/log-violation • Log a potential cheating incident
2. GET /api/admin/violation-reports • Retrieve violation reports for admin review

Frontend Modifications:

1. Add a secure browser mode that attempts to prevent tab switching and external copying.
2. Implement warning modals for users when violations are detected.
3. Include a user agreement page before starting the contest, explaining the anti-cheating measures.

Challenges and Considerations:

1. Privacy Concerns: Balancing cheat detection with user privacy.
2. False Positives: Ensuring legitimate actions aren't flagged as cheating.
3. Technical Limitations: Browser and OS differences may affect consistent implementation.
4. User Experience: Implementing strict measures without significantly hindering the coding experience.
5. Evolving Cheat Methods: Continuously updating the system to detect new cheating techniques.