**CODE-ACE**

**PROJECT REPORT OF CODE ACE –AN INTERVIEW AND DSA PROBLEM SOLVE PLATEFORM**

**2025-26**

**Name - Sandeep Kumar**

**Course - MCA**

**Institute - N.I.T Jamshedpur**

**State - Uttarakhand (India)**

**E-mail - Skumar131431@gmail.com**

****

**WHY I BUILT THIS PROJECT**

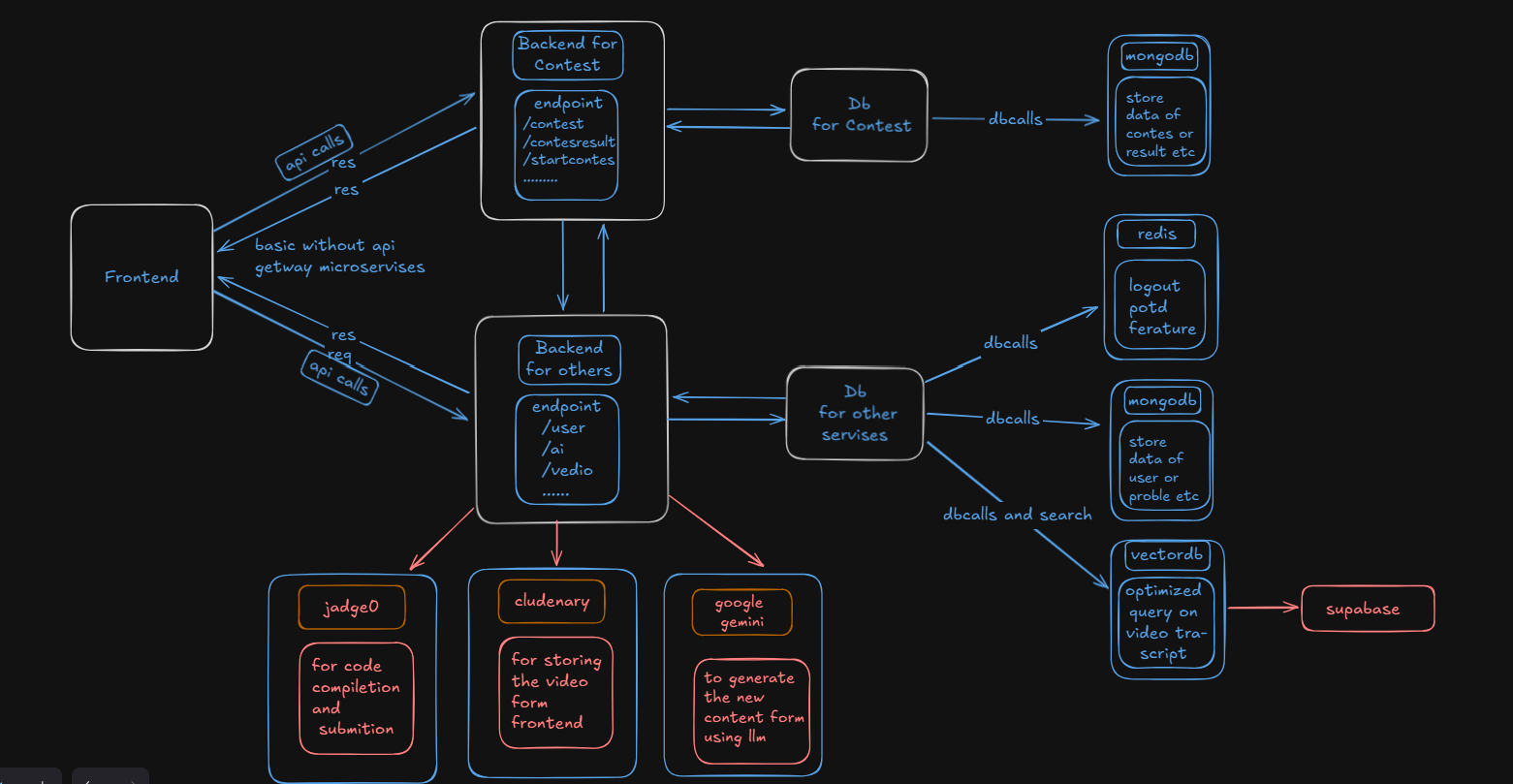
I built this project to **solve the limitations I personally experienced while using existing coding platforms** like LeetCode and HackerRank. While they are great for practicing problems, I felt there was **room for improvement** in the following areas:

1. **Lack of Real Interview Simulation**  
   Most platforms only offer coding problems but **don’t simulate real interviews**. I wanted to create a system where users could **practice mock interviews with an AI**, receive **instant feedback**, and **track their progress** — just like a real-world interview experience.
2. **No AI-Powered Help in Videos**  
   Sometimes, even after watching problem explanation videos, certain concepts are hard to understand. I wanted to **build a feature where users can ask doubts about any part of the video**, and the AI will give answers using **Vector DB** for context — something current platforms lack.
3. **Premium Features without Real Money**  
   Most platforms require payment for premium problems. With LeetCoins earned from daily practice or contests, users can **unlock premium content without paying**, encouraging **fair access**.
4. **Friendly UI and Community-Like Feel**  
   I wanted a **user-friendly, clean interface** with the modern look of **Tailwind CSS + DaisyUI**, making the platform both **aesthetically pleasing and easy to navigate**.
5. **Quick Access to Doubts with Gen AI**  
   Often, learners are stuck on small issues. Instead of Googling or asking on forums, they can **instantly ask the chatbot** on the problem page and get context-aware solutions using **Generative AI**.
6. **Practice + Contest + Learning All-in-One**  
   My goal was to **combine practice, contests, interviews, video learning, and AI help — all under one platform** — something no single platform fully offers today.

**PROJECT OVERVIEW**

**CodeAce** is an advanced LeetCode-style coding platform that combines AI-driven interview preparation with gamified learning and real-time contests, offering users a comprehensive solution for mastering coding interviews. The platform features 1000+ problems across DSA, system design, and algorithms, categorized by difficulty (Easy, Medium, Hard) and topics (Arrays, Graphs, DP, etc.). Users can participate in AI-powered mock interviews with performance scoring, detailed feedback, and historical review, while earning **1 LeetCoin daily** by solving the **Problem of the Day (POTD)**—redeemable for premium problems. Each problem includes **video explanations** with an AI-powered **"Ask Me"** feature (Vector DB-backed) for doubt resolution, alongside a **GenAI chatbot** for context-aware assistance. Real-time coding contests with leaderboards enhance competitive practice, supported by a **microservices backend** using **Redis** for session management, POTD tracking, and rate limiting The platform boasts a **modern UI** built with **DaisyUI + Tailwind CSS**, ensuring a seamless and intuitive user experience.

**Project Architecure**

****

**TECH STACK---->**

**Frontend**

* **React.js**: javascript laibrary for building the user interface
* **Tailwind CSS**: Utility-first CSS framework for styling
* **Zod & React Hook Form**: For form validation
* **Redux**: Global state management
* **WebRTC & WebSockets**: For real-time communication features
* **DaisyUI :** for perfect and creating clean UI

**Backend**

* **Express.js**: Node.js framework for API development
* **JWT**: For authentication and authorization
* **Supabase**: For database and authentication services
* **MongoDB**: NoSQL database for flexible data storage
* **Redis**: For session management and potd tracking

**AI & Media Services**

* **Gemini API**: AI-powered coding assistance and interview feedback
* **Cloudinary**: For media storage and management
* **Web Speech API**: For voice recognition during interviews

**FEATURE AND FUNCTIONALITY**

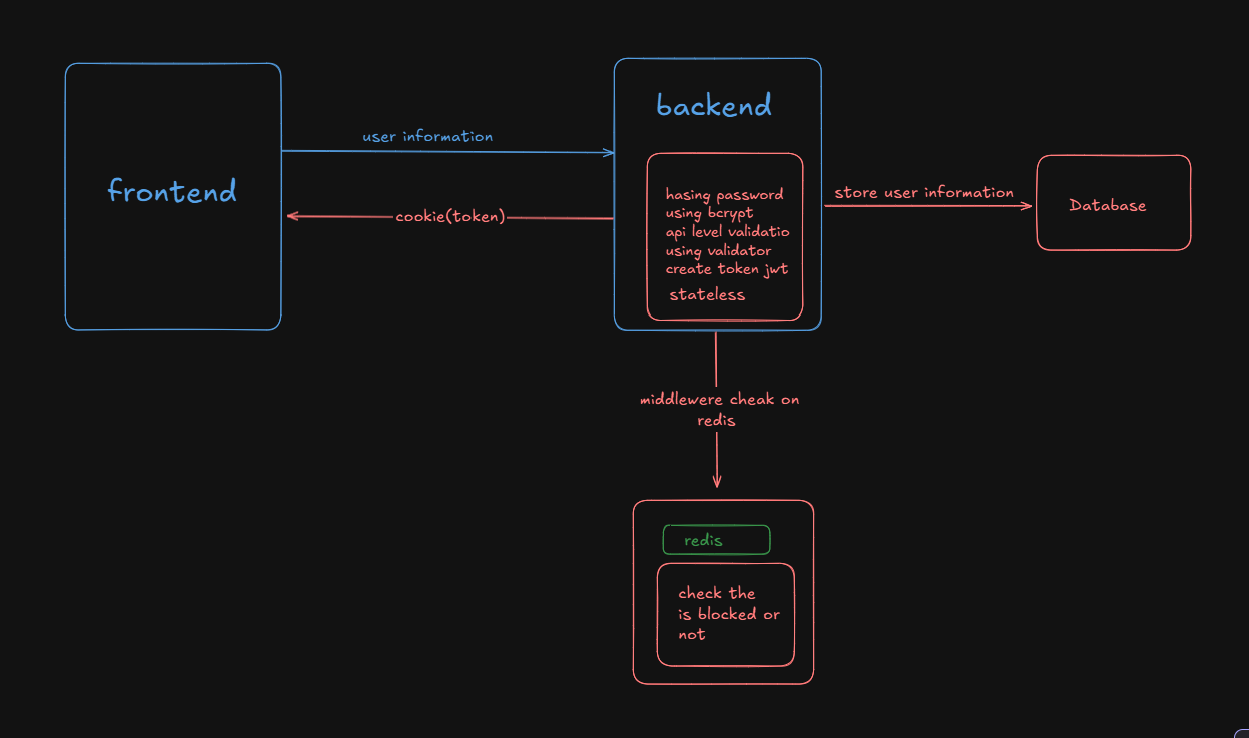
**1. Authentication System**

## ****Overview****

The authentication system provides **secure user registration, login, and session management** for CodeAce platform. It includes:

* **Password hashing** (bcrypt)
* **Input validation** (validator package)
* **JWT-based authentication** (stateless sessions)
* **Token blacklisting** (Redis for revoked tokens)
* **Redux state management** (to avoid frequent DB calls)

**Authentication System**

****

## ****Key Features & Workflow****

**1. User Registration**

* **Process**: Users sign up with email, username, and password.
* **Security**:
  + Password hashed using **bcrypt** (with auto-generated salt).
  + Email & password validated using **validator** package.
* **Storage**:
  + User details stored in DB (except plain password).
  + JWT token generated and sent to client.

**2. User Login**

* **Process**:
  + User provides credentials → Server validates → Issues JWT.
* **Security Checks**:
  + Password matched against hashed DB entry (bcrypt.compare).
  + JWT signed with secret key (expiry set, e.g., 1hour).

**3. JWT (JSON Web Token) Implementation**

* **Token Structure**:

{

"userId": "123",

"email": "user@example.com",

"iat": 1620000000,

"exp": 1620086400

}

* How it works:
  + Generated on login (jsonwebtoken package).
  + Stored in **HTTP-only cookie** (secure, not accessible via JS).
  + Verified on each protected API call.

**Note**-(why token structure not contaion password or any confidential details)-> JWTs can be **decoded** (not encrypted), so sensitive data like **passwords, credit cards, or personal info** can be exposed if intercepted .Even **hashed passwords** in JWTs are risky because attackers can compare them with stolen database hashes or rainboo table .

**4. Redis for Token Blacklisting**

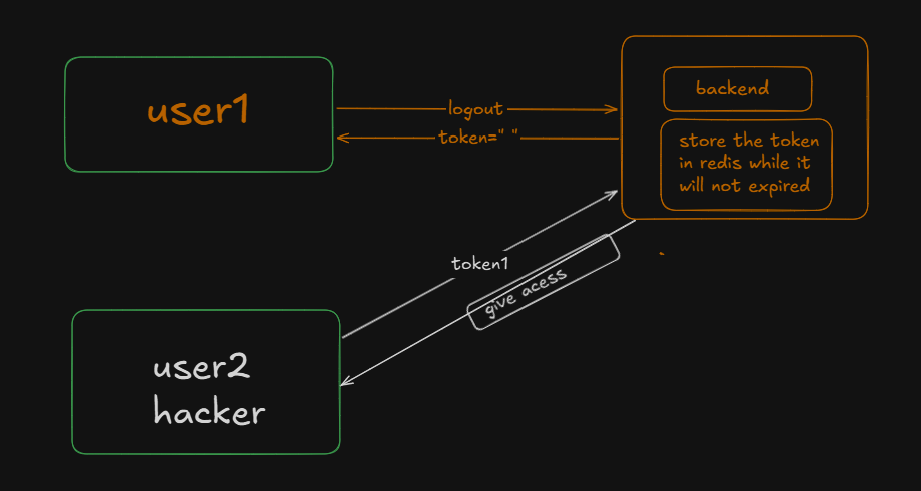
* **Why?** To handle logout & token revocation.
* **Implementation**:
  + When user logs out → Token added to Redis blocklist.
  + Middleware checks Redis before allowing access or authorization

**2. Logout Feature Using Redis-**

**Without Redis ->**

1. **Delete the client-side cookie** – Remove the authentication token from the browser.
2. **Send an expired/empty cookie in response** – Overwrite the existing cookie with an empty value.
3. **Server ignores future requests** – Without a valid cookie, the server treats the user as logged out.
4. **No session tracking needed** – Since Redis isn’t used, cookie deletion alone enforces logout.

Problem**-If the authentication token isn't expired and someone copies/stores it, they can reuse it to log back in even after logout.**

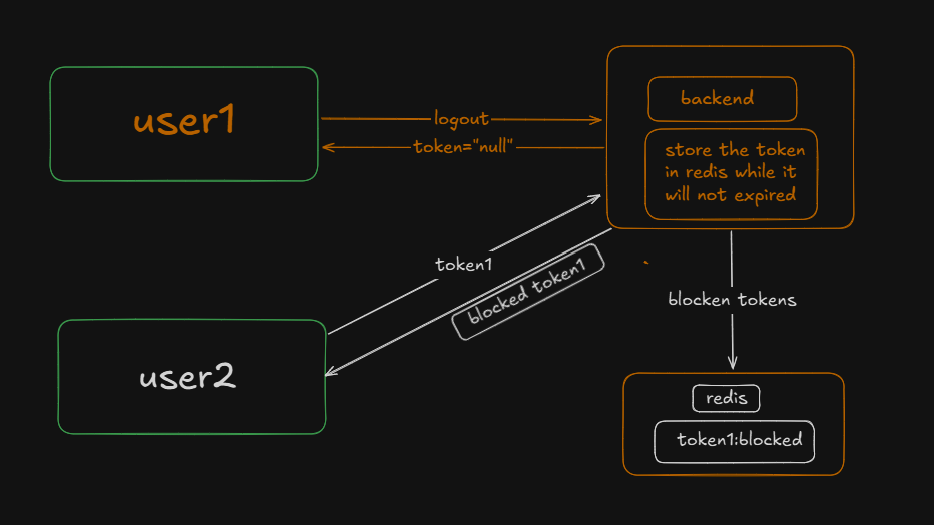
****

**With redis->**

In authentication systems, Redis acts as a **fast, in-memory database** to store and manage active or revoked tokens. Here’s how it works:

When a user logs in, their JWT is stored in Redis with an expiration time (e.g., 24 hours). If the user logs out or the token is revoked (like a passport being flagged), Redis **blacklists the token** by marking it as invalid. Any subsequent request using that token triggers a Redis check—if the token is found in the blacklist, the system rejects the request with an error (e.g., "Token revoked"). This ensures security similar to passport verification, where an invalidated document is immediately detected and denied access. Redis’s speed makes this process seamless, maintaining performance while enforcing strict session control.

**Bonus**: Redis also helps in my project to implement **rate-limiting, Problem of the day feature tracking .**

****

**3. DSA Problem and Jadge0**

This project is a **Data Structures and Algorithms (DSA) practice platform** designed for students and programmers to solve coding problems efficiently. It provides an integrated coding environment with problem filtering, search, pagination, and code compilation functionality.

### Key Features->

#### 1. ****Code Editor****

* Uses **Monaco Editor** (same as used in VS Code).
* Supports syntax highlighting, autocompletion, etc.
* Real-time code writing experience in the browser.

#### 2. ****Code Compilation****

* Integration with **Judge0 API**.
* Supports multiple programming languages (based on Judge0 support).
* User can write code, submit, and see the output or errors.

#### 3. ****Problem Management****

* Each problem includes:
  + Title
  + Description
  + Difficulty (Easy, Medium, Hard)
  + Tags (e.g., Array, Graph, DP)
* Problems stored in database (MongoDB).

#### 4. ****Search Functionality****

* Users can search problems by title or keywords.
* Implements fuzzy or partial matching for better results.

#### 5. ****Filter System****

* Users can filter problems by:
  + **Difficulty** (Easy, Medium, Hard)
  + **Tags** (Array, Graph, Linked List, etc.)

#### 6. ****Pagination****

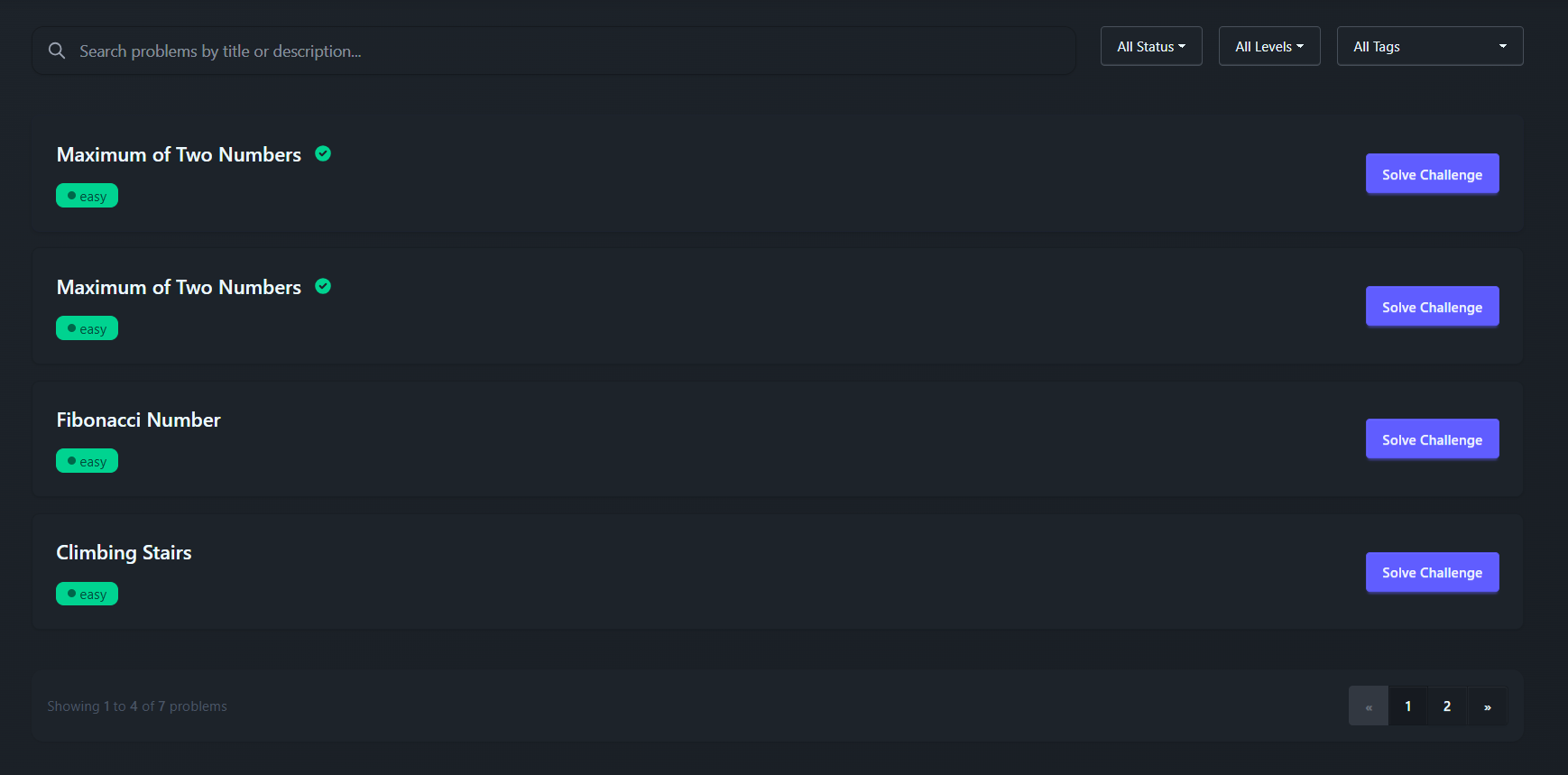
* Problems are paginated for better UI performance.
* User can browse page-wise rather than loading all at once.

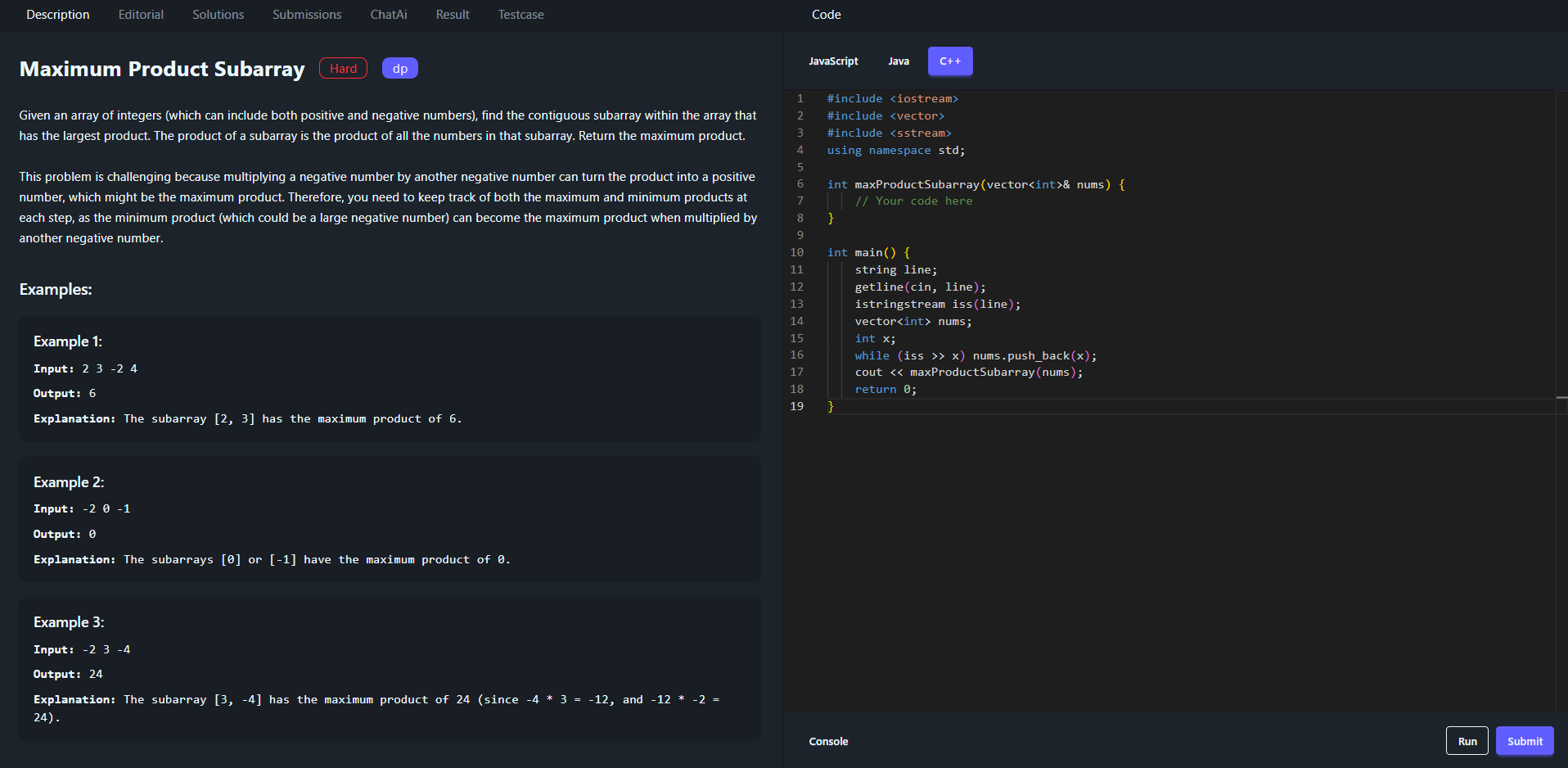
**How jadge0 works->**

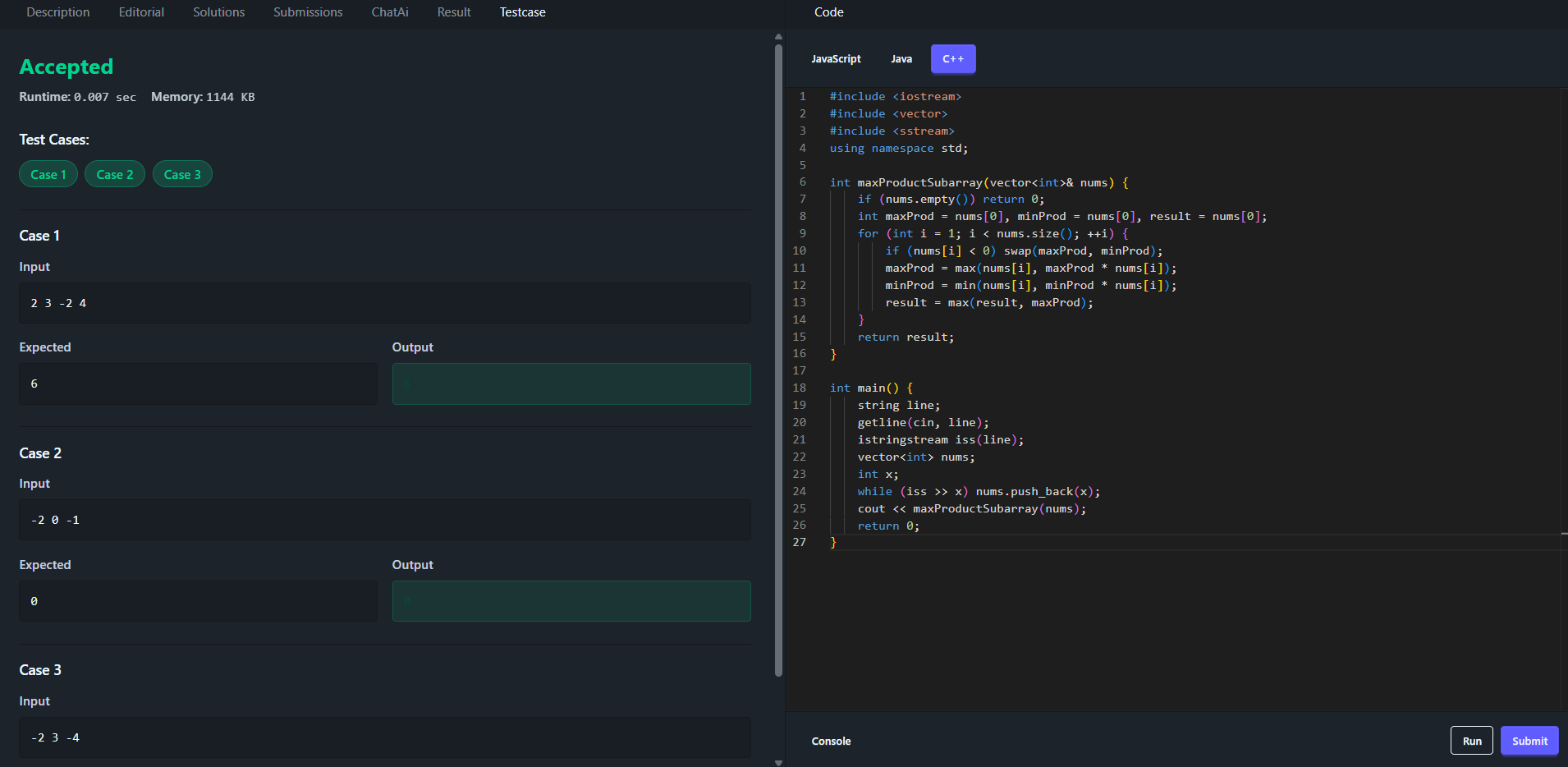
Judge0 is an open-source API that allows remote code compilation and execution. In this project, the code written by the user in the Monaco Editor is first sent to the Judge0 API. The API requires the complete source code, the corresponding language ID, input (if any), and expected output. Once the code is submitted, Judge0 returns a unique **token** instead of the result immediately, which helps reduce the server load. Later, this token is used to fetch the execution status and output. The returned status code indicates whether the code was successfully compiled and executed, or if there were any errors.

**Tokens : ["f5e9c06a-a9d0-4b5c-8b2f-3497f4c4f7aa", "b7a18265-f1ce-4574-bec9-0cbf460e13d6 ]**

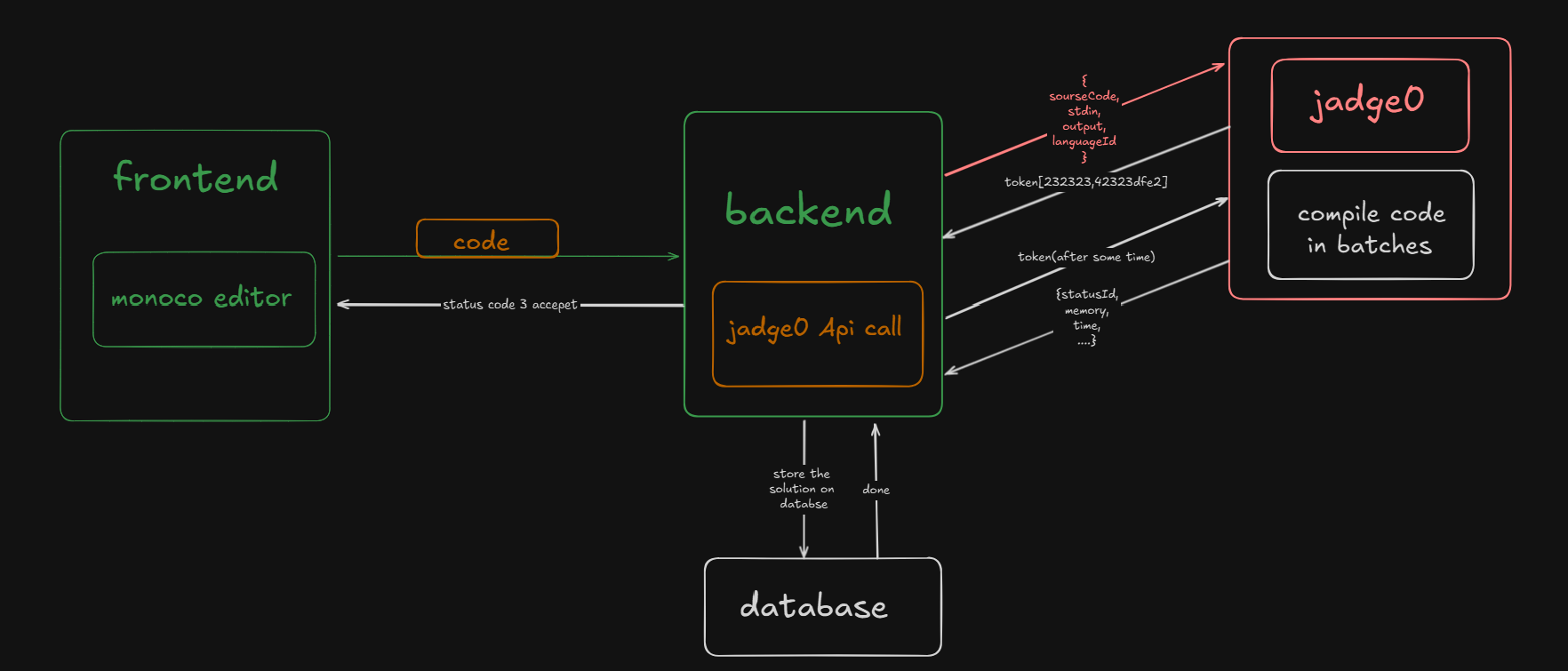
**\**

****

****

****

**Jadge0 Architecture**

****

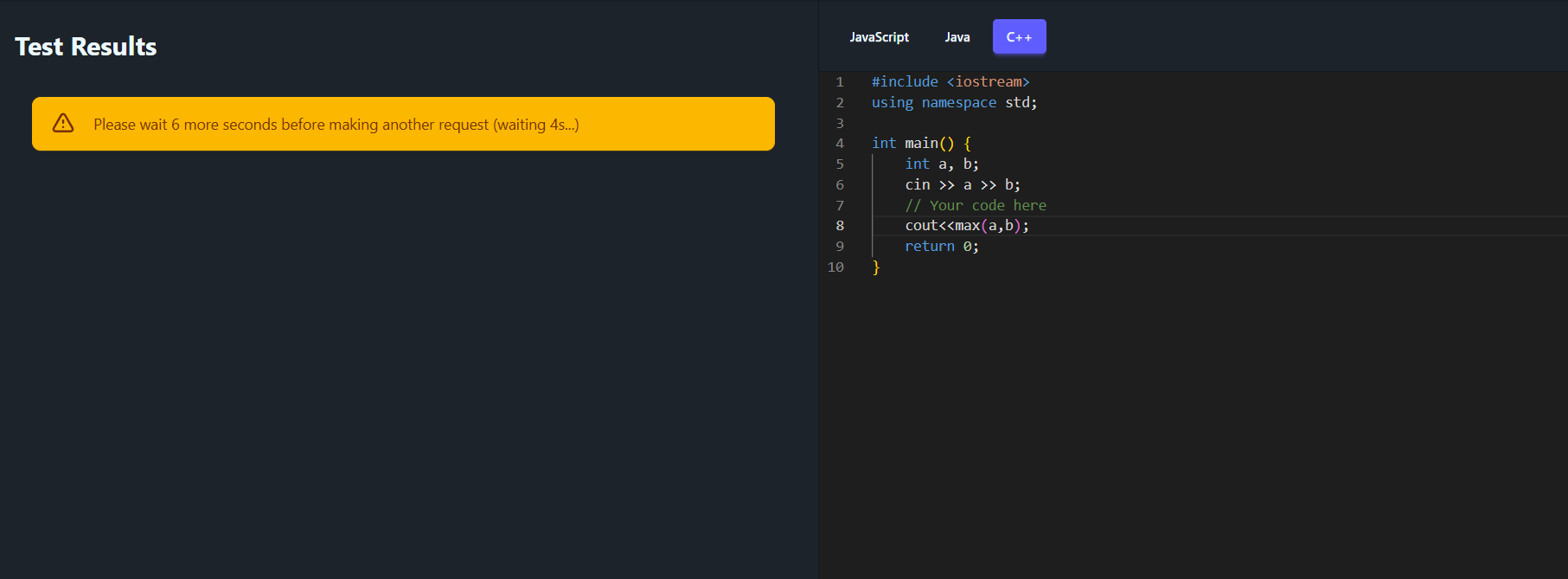
**4. Rate Limmiter**

A **Rate Limiter** is a mechanism that controls the number of requests a client (user, IP, or service) can make to a server within a specified time window. It helps prevent abuse, overuse, and ensures fair resource allocation.

**Common Rate Limiting Strategies**

* **Fixed Window:** Allows X requests per Y seconds (e.g., 100 requests/minute).
* **Sliding Window:** More precise, tracks requests in real-time.
* **Token Bucket:** Grants tokens at a fixed rate; requests consume tokens.
* **Leaky Bucket:** Queues requests and processes them at a fixed rate.

**In my project, I have implemented a rate limiter that enforces a 5-second delay between two consecutive 'Run' or 'Submit Code' actions.**

****

***How Rate Limmiter works in my Plateform:***

* It captures the user's IP address and current timestamp (in seconds).
* It checks Redis for the timestamp of the user's **last request**.
* If a previous request was made within the last 5 seconds, it blocks the request and responds with a message like:  
  **"Please wait X more seconds before making another request."**
* If enough time has passed, it updates Redis with the current timestamp and allows the request to proceed.

This helps prevent users from spamming actions like "Run Code" or "Submit Code" too frequently.

**5. Interview And Feedback and Score**

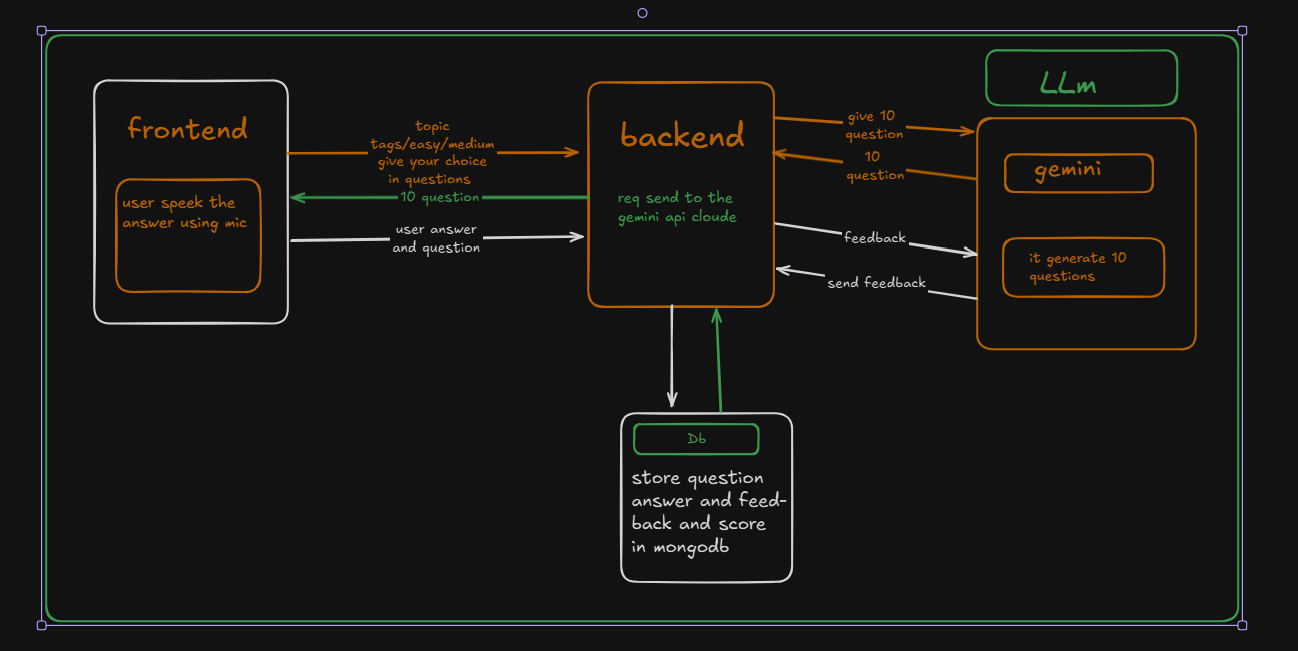
I've developed an **AI-powered Interview feature** on my platform that allows users to simulate real interview experiences tailored to their interests.

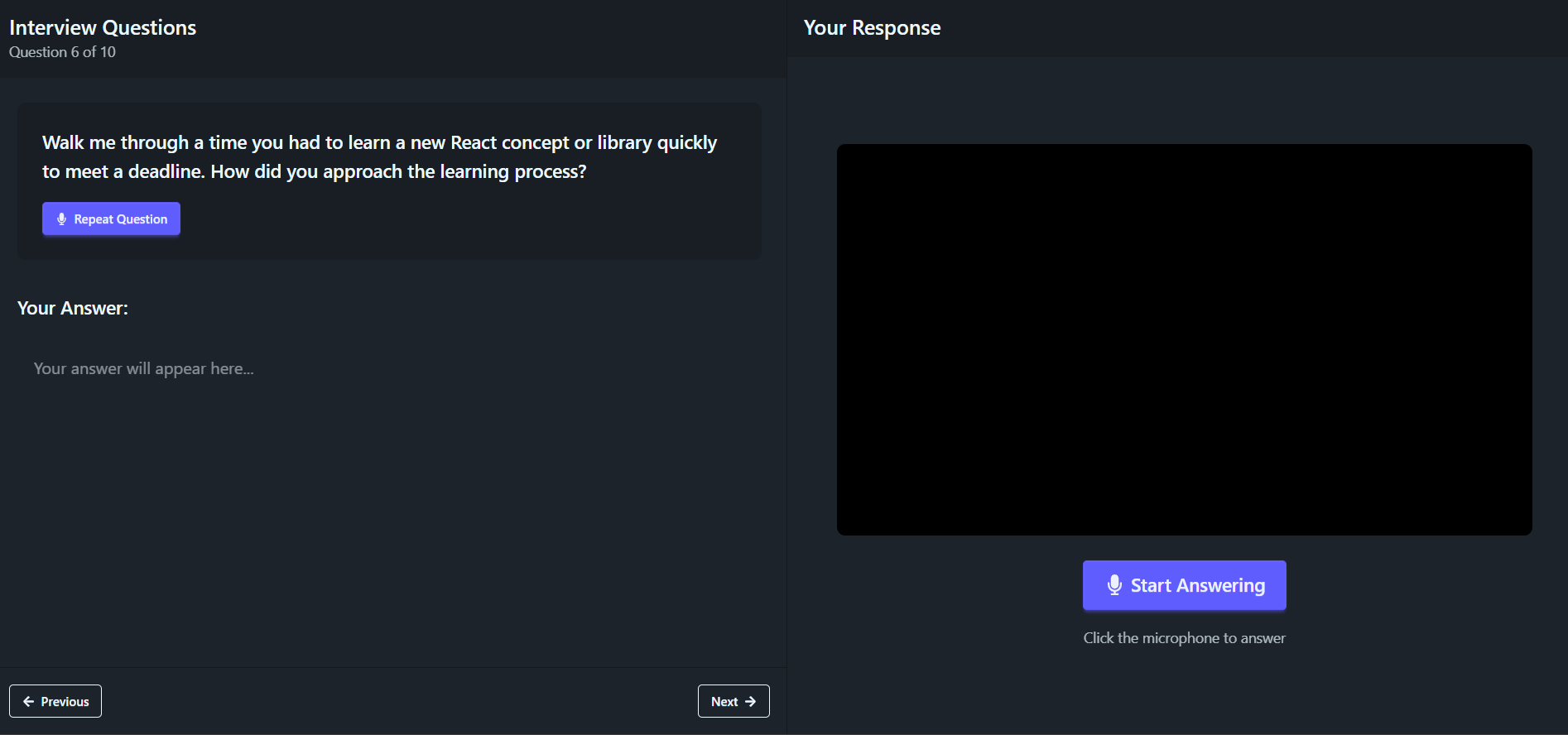
**Problem-(why this feature)**

Many people practice DSA (Data Structures & Algorithms) extensively and have a strong grasp of core concepts. However, they often struggle during **real interviews** — not because they lack knowledge, but because they haven't practiced **communicating their thoughts under pressure**.

This issue is especially common among **introverts** or those who haven’t had mock interview experience. To help users overcome this gap and build confidence in expressing their solutions verbally, I created this **AI-driven interview simulation feature**.

It gives users a safe environment to **practice speaking, answering technical questions, and receiving feedback**, just like in a real interview — helping them become well-prepared and interview-ready.

****

****

#### How It Works:

* **Topic Selection:**  
  The user selects a topic of their choice (e.g., Arrays, Trees, OOP, DBMS, javascript,node js etc.).
* **Question Generation (via Gemini API):**The selected topic is sent to the backend, where Gemini generates **10 interview questions**.
* **Interactive Interview Mode:**
  1. The AI speaks each question aloud.
  2. The user responds to each question verbally.
  3. User responses are **stored locally** during the session.
* **Final Submission:**  
  After all 10 questions, the user clicks on **“Final Submit.”**
* **Results Page:**  
  A new page opens showing:
  1. All **user answers**
  2. The original **questions**
  3. These are sent back to Gemini for **evaluation**.
* **Feedback & Scoring:**  
  Gemini analyzes the answers and provides:
  1. **Score out of 10**
  2. **Detailed feedback**
* **Rewards:**  
  If the user scores **more than 5**, they are rewarded with **3 LeetCoins**
* **Track your Progress:**
  + 1. **You can check your progress because every interview all information a saved in db and show in form or card in our app**

**6. Premium Problem Unlock Using LeetCoin**

Platforms like LeetCode and others often require users to pay money to access premium problems—those frequently asked by top MNCs. However, my platform is different. Instead of charging users, we offer a unique currency called **LeetCoin*.***

With **LeetCoin**, you can unlock premium problems by:

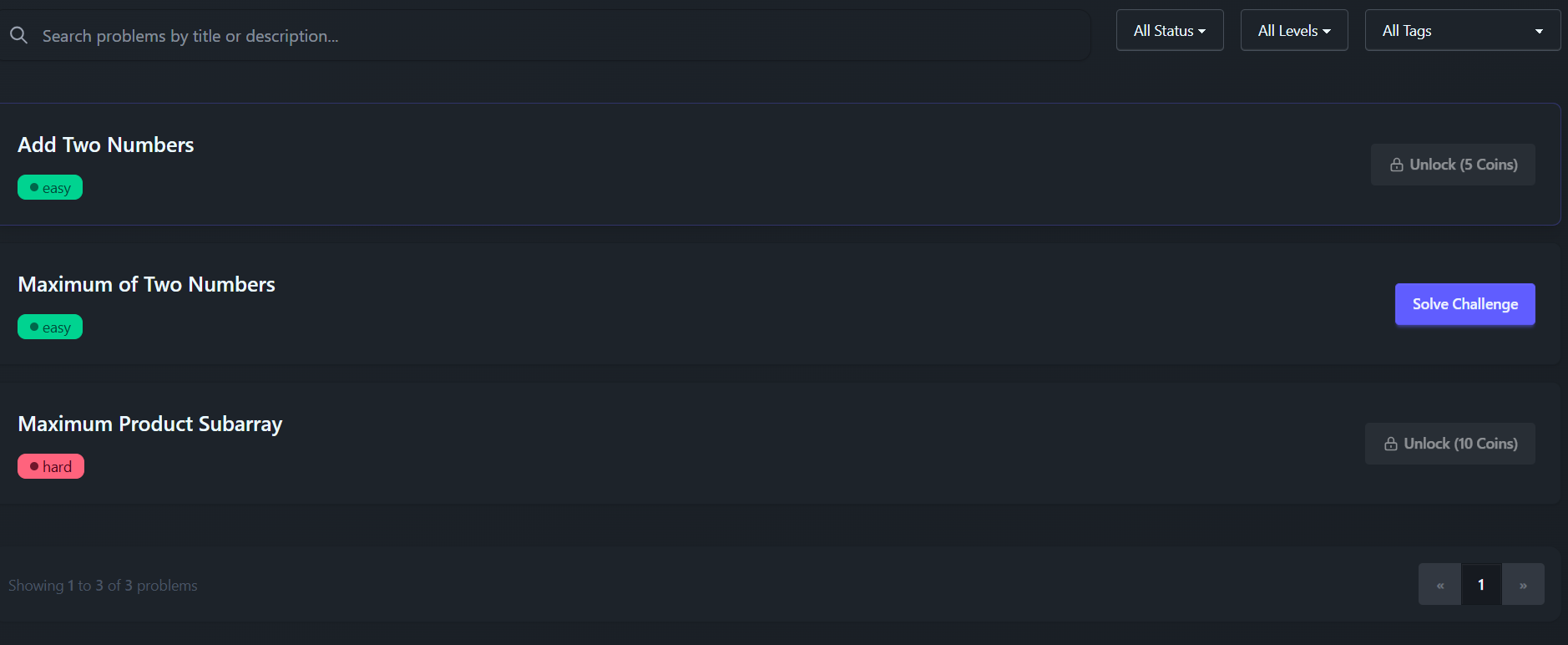
* Solving the Problem of the Day (POTD),
* Scoring **5+ points** in mock interviews, or
* Ranking **within the top 10** in contests.

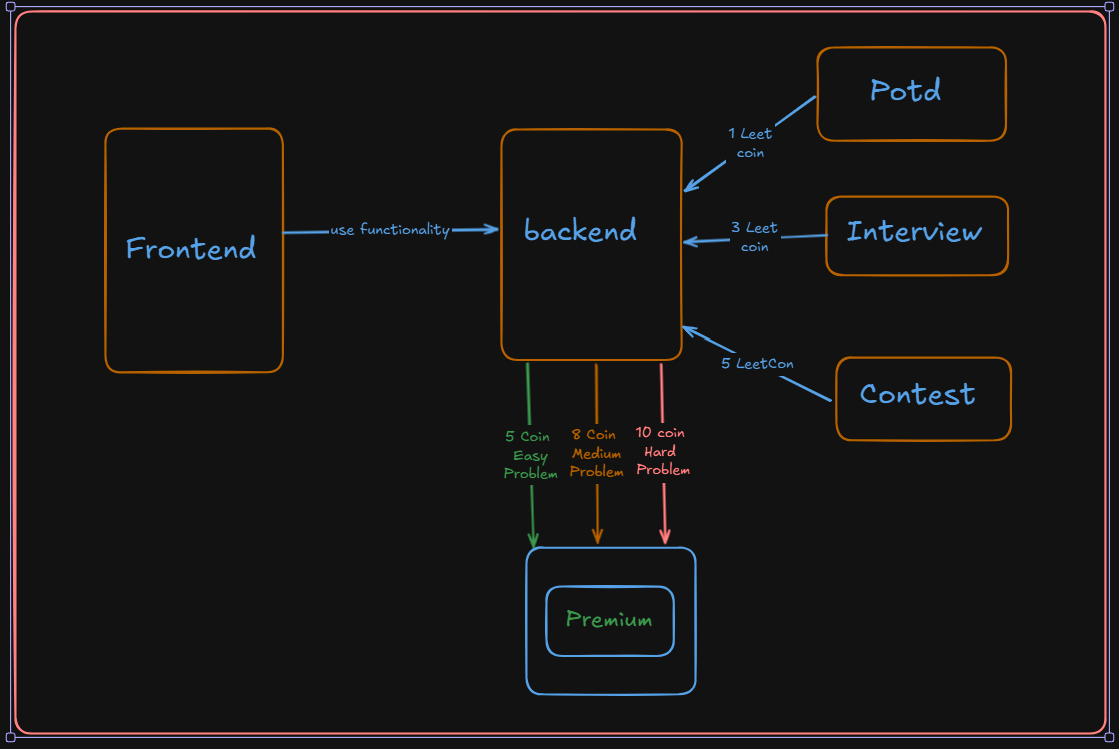
**No payments, just skill-based rewards!**

#### ****Enhanced User Experience with Smart Features:****

✔ **Search Functionality** – Quickly find problems by title, company, or difficulty.  
✔ **Pagination** – Browse seamlessly with organized problem sets.  
✔ **Advanced Filtering** – Sort by:

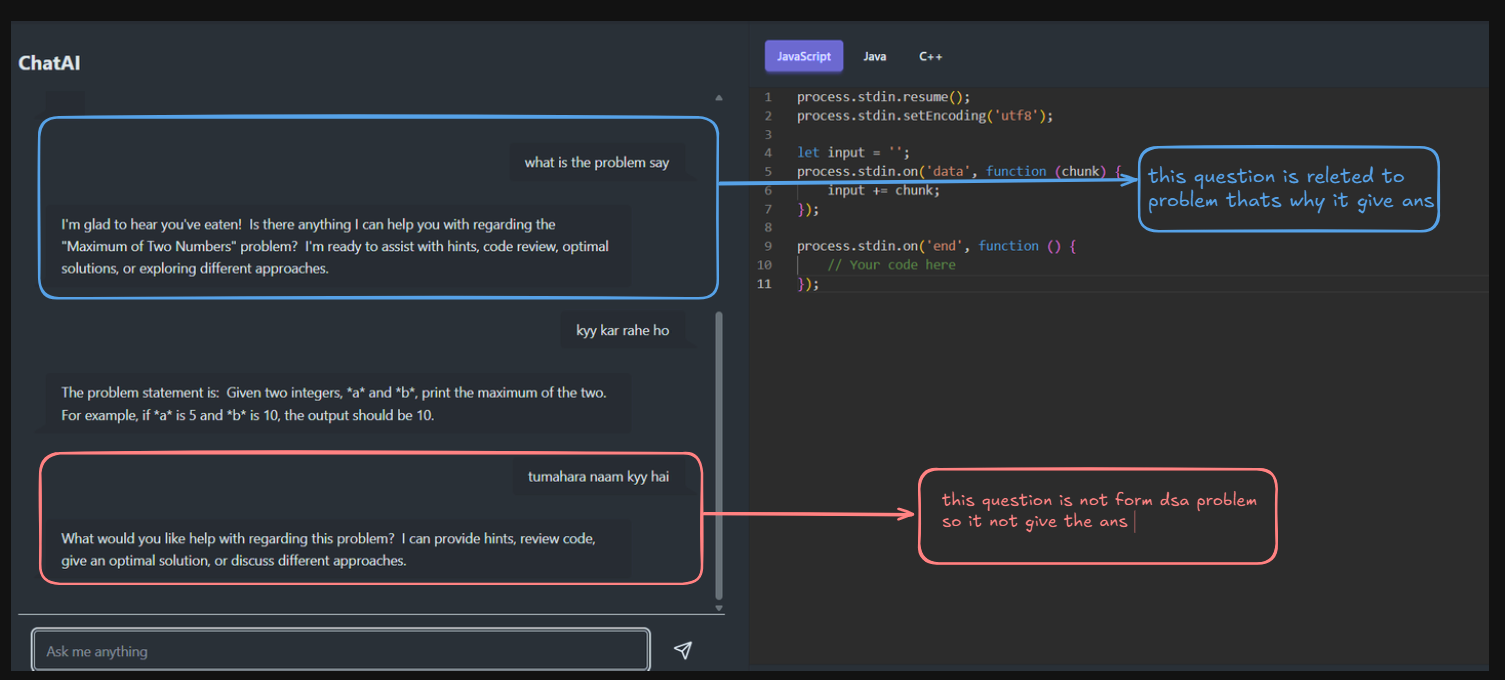
* **Company** (Google, Amazon, Meta, etc.)
* **Difficulty** (Easy, Medium, Hard)
* **Topic** (DP, Graphs, Arrays, etc.)
* **Frequency** (Most asked in interviews)

****

****

**7. AI BoT**

While solving DSA problems, it's common for users to get stuck at certain points and then switch to other platforms like Google, YouTube, or discussion forums to find help. This not only breaks the flow but also wastes valuable time and causes distractions. To solve this problem, I have integrated an AI bot directly into my own platform. This bot is specifically designed to assist with the problem currently being solved. If the user asks any question related to that particular problem—whether it's about the logic, a specific test case, or a concept—the bot will respond with helpful guidance. However, if the user asks something unrelated, the bot will not respond. This ensures focused, context-aware assistance without distractions, enhancing the overall problem-solving experience.

****

**How this give answer only Dsa proble**

In my platform, I have configured the AI bot with a specific system instruction that guides its behavior. The instruction clearly defines that the bot is dedicated solely to answering questions related to DSA (Data Structures and Algorithms) problems. It has been explicitly told that it is a problem-solving assistant and should only respond when the user asks something related to the current DSA problem. If any unrelated query is made—whether it's about general knowledge, coding outside the current problem, or any other topic—the bot has been instructed to ignore it and not provide any response. This focused setup ensures that users remain on track, avoid distractions, and receive precise, relevant help related only to the problem they're solving.

**Why I do this**

When we ask a question to any LLM (Large Language Model), the input is first tokenized, and each token contributes to the total usage cost. Most free APIs come with strict token limits, and even in paid plans, the number of available tokens is limited. This means that if users are allowed to ask any kind of question, it can quickly consume a large number of tokens, leading to increased costs and inefficient usage. To avoid this problem, I have restricted the AI bot on my platform to respond **only** to queries related to the current DSA problem. This approach ensures that unnecessary or unrelated interactions are filtered out, significantly reducing token usage and keeping API costs under control while still providing targeted and valuable assistance to users.

**8. VIDEO FOR EVERY PROBLEM**

In my project, I have implemented a feature where each problem includes an editorial video to help users understand the solution better. These videos are stored on Cloudinary, and the metadata is saved in the database. The upload process is handled directly from the frontend, but only after proper authentication from the backend. When a user initiates a video upload, the frontend first sends a request to the backend, which responds with the necessary upload credentials including: signature, timestamp, public\_id, api\_key, cloud\_name, and upload\_url. Using this secure response, the frontend then uploads the video directly to Cloudinary, ensuring a fast, efficient, and secure upload process.

**Why Not Use Backend for Upload video-**

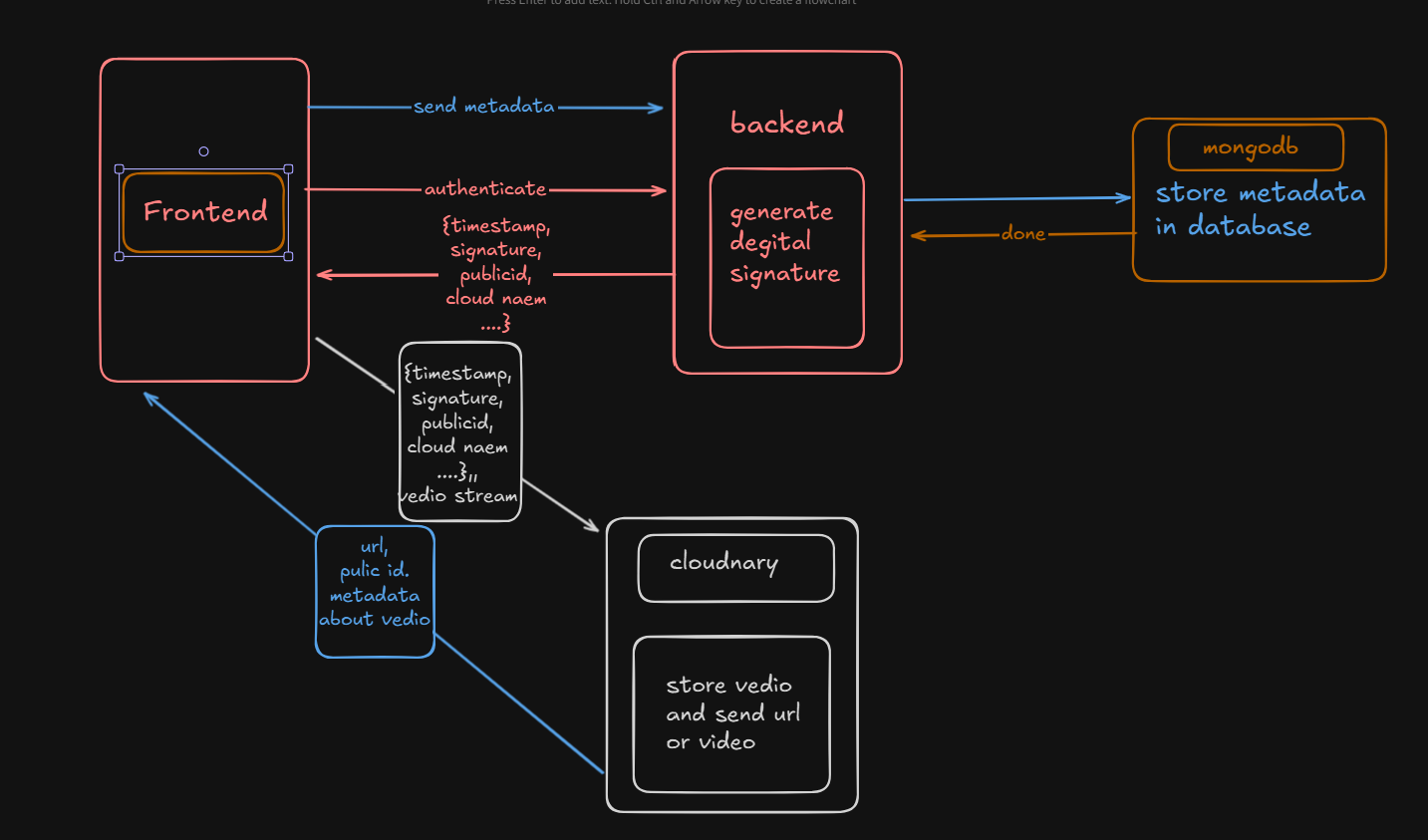
I chose to upload videos directly to Cloudinary from the frontend instead of routing the file through the backend to optimize performance and reduce server load. Uploading large video files via the backend would require the server to handle heavy file processing, which could lead to increased memory usage, slower response times, and higher infrastructure costs. By generating a secure upload signature and other required credentials on the backend, and then sending them to the frontend, the actual file upload happens directly between the client and Cloudinary. This approach is secure—since uploads are authenticated—and much more efficient, as it bypasses the server entirely for the heavy lifting, keeping the backend lightweight and focused only on authorization and token generation.

**I used a digital signature in the Cloudinary upload process to ensure the security, integrity, and authenticity of the upload request.**

**Why Not Use Database for Store Video**

If I store videos directly in the database, it would significantly increase the storage size of each row due to the large binary data. As a result, the overall size of the table would grow rapidly. This impacts performance—where earlier the database engine could load, for example, 100 rows into RAM for processing, it may now only be able to load 10 rows due to the increased size. This drastically reduces efficiency and leads to slower query performance, especially during read-heavy operations. Therefore, to keep the database lightweight and optimized for structured data access, I chose to store only the video metadata (like the Cloudinary URL or public\_id) in the database, while offloading the actual video storage to Cloudinary, which is built specifically to handle large media files.

**Video store on Cloudnary**

****

**9. ASK ME IN VIDEO**

The **"Ask Me" feature** is one of the most interesting and unique aspects of my platform. While a user is watching an editorial video, if they have any doubts or questions related to a specific part of the explanation, they can instantly ask their query using this feature. It's similar to an AI-powered assistant but highly optimized for this use case. Instead of general-purpose responses, the "Ask Me" feature is designed to understand the context of the current video and provide accurate, relevant answers. This makes the learning experience more interactive and personalized, helping users clear their doubts in real time without needing to switch platforms or search externally.

The core idea behind the **"Ask Me"** feature is that I provide the **video content** (or its transcript/context) directly to the AI. This allows the AI to fully understand what is being explained in the video. When a user asks a question, the AI doesn't answer generally—instead, it analyzes the context of the video and responds based on that specific explanation.

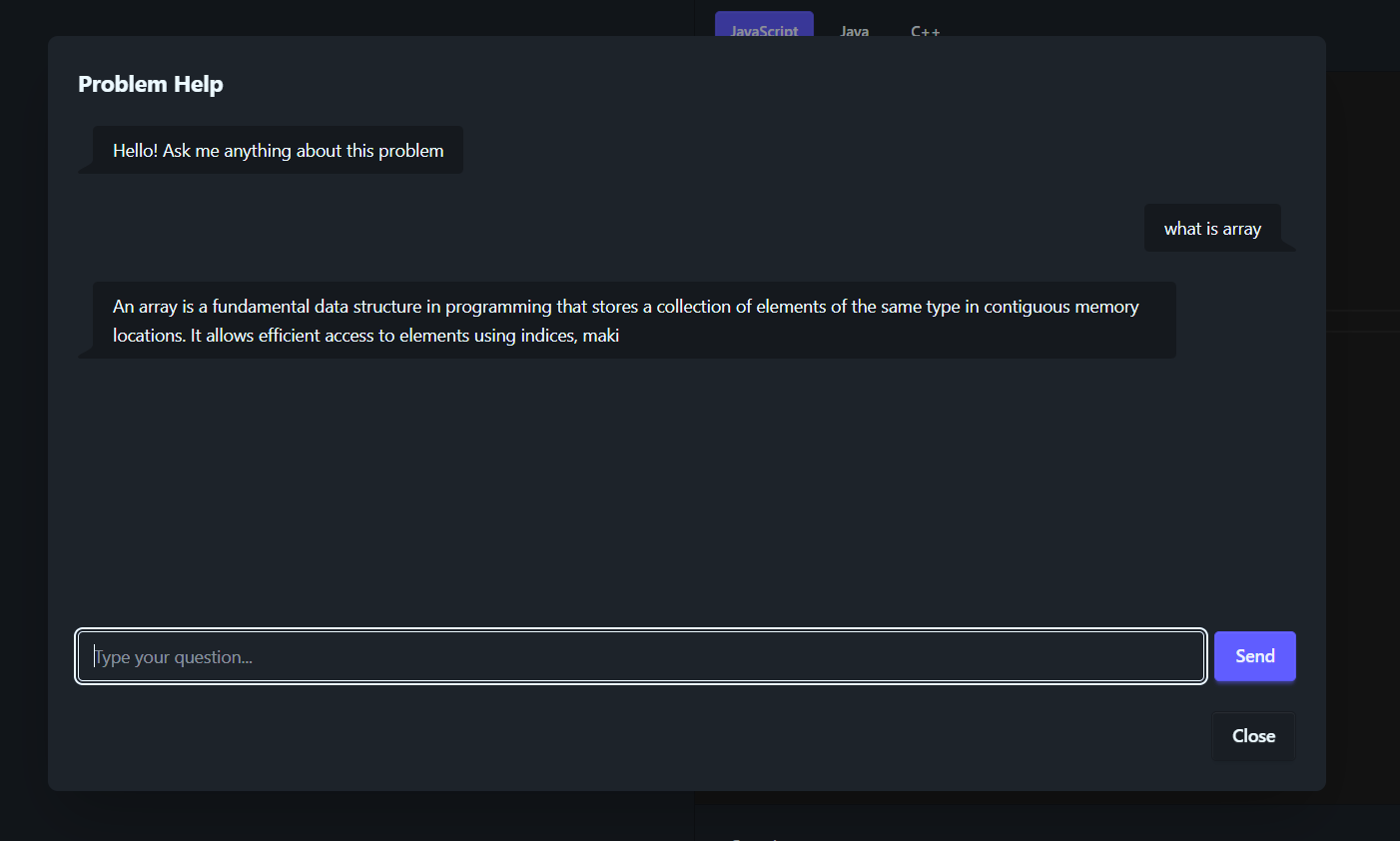
**What Problem I faced**

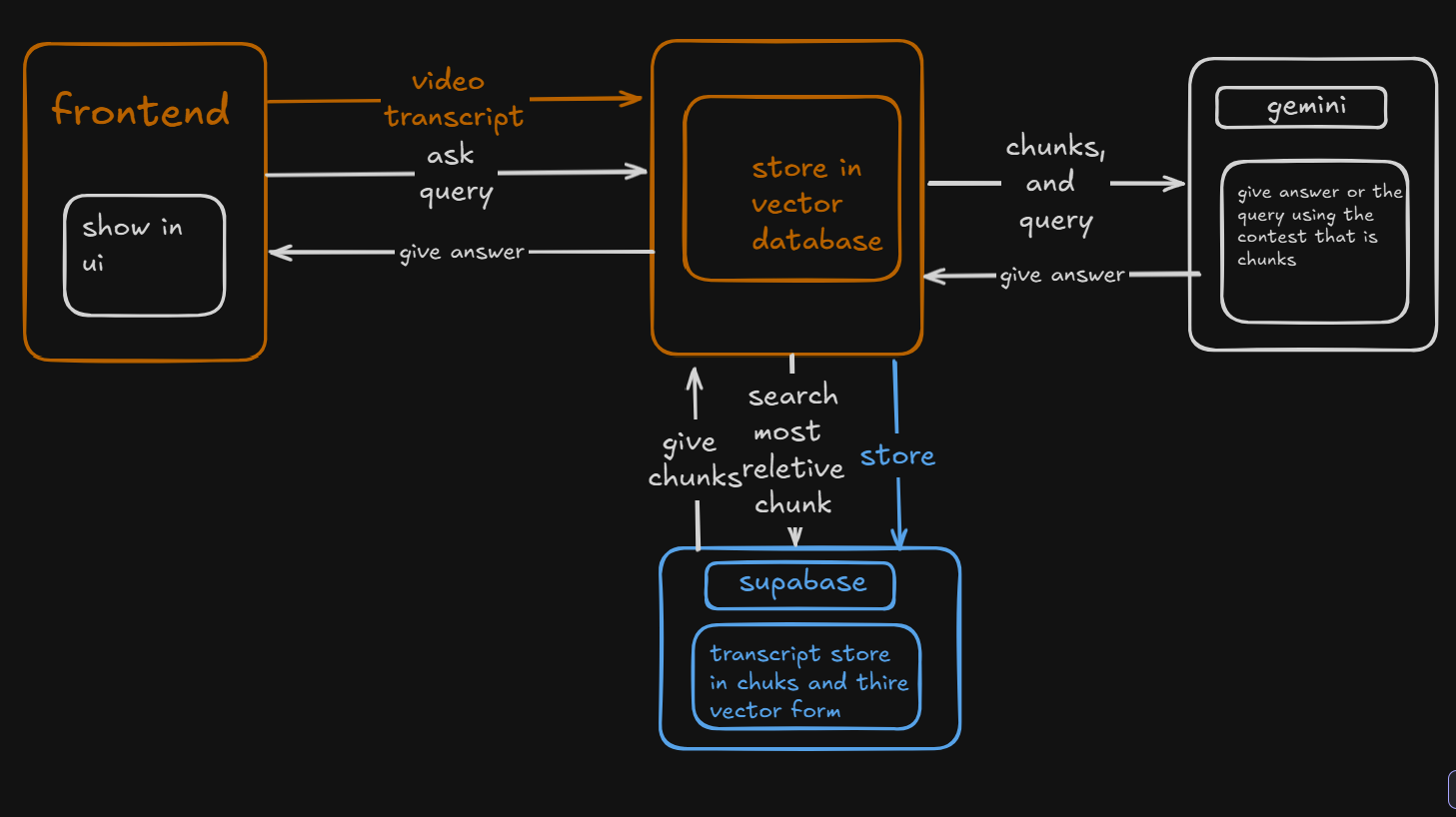
While building the "Ask Me" feature, one major challenge I faced was managing the **size of the video transcript**. Editorial videos often have long explanations, and sending the **entire transcript to the AI model** every time a user asked a question was highly inefficient. It resulted in unnecessary **token consumption**, which is costly—especially when using LLMs with limited token limits in both free and paid plans.

**How I Solve This**

To solve this, I conducted some research and discovered the concept of **vector databases**. These databases allow storing large text data in smaller **chunks**, each represented as a **2D vector (embedding)** generated using an LLM. Instead of sending the entire transcript, I now perform a **semantic search** in the vector database to retrieve only the most relevant chunks based on the user's question. These relevant chunks are then sent to the AI as context.

This optimization helped me **dramatically reduce token usage**, maintain fast response times, and still provide highly accurate, context-aware answers. Now, the AI doesn’t process the full transcript repeatedly—it only sees what’s most relevant, thanks to vector search.





**10. PROBLEM OF THE DAY**

The **POTD (Problem of the Day)** feature is designed to encourage daily problem-solving and build consistent coding habits among users. Every day, a new problem is added by the admin, and users can attempt to solve it within that day. Upon successfully solving the POTD, the user is rewarded with **1 LeetCoin**, a virtual currency on the platform. These LeetCoins can be accumulated and later used to unlock **premium problems** that are otherwise restricted. This gamified reward system not only increases user engagement but also provides an incentive for regular practice while giving value to solving problems consistently.

**How I Solve if User Multiple time solve same the problem then only one time take leetcoin**

To prevent users from earning multiple LeetCoins by solving the same POTD more than once a day, I implemented a tracking mechanism using **Redis**. Each time a user solves a problem, the system creates a unique key in Redis using the format:  
**userId:problemId**

Before awarding a LeetCoin, the system checks whether this key exists in Redis:

* If the key **exists**, it means the user has already solved the problem today — so no coin is awarded.
* If the key **does not exist**, the user is rewarded with **1 LeetCoin**, and the key is stored in Redis.

To automate daily resets, I set an **expiry time of 24 hours** for each key using Redis's built-in TTL (Time-To-Live) feature. This ensures that the same user can earn a coin for that problem again **only on the next day**, keeping the system fair and efficient without manual cleanup.

**11. CONTEST FEATURE**

I have also implemented a **Contest feature** on my platform to create a more competitive and engaging environment for users. Each contest consists of **3 randomly selected DSA problems**, and users are given **1 hour** to solve them. Based on their performance, users are rewarded with **LeetCoins**, adding a gamified incentive to participate actively.

**The workflow is as follows:**

* All potential contest problems are first added and managed by the **admin** in a dedicated problem database.
* When the admin wants to create a new contest, the system automatically picks **3 random problems** from this database.
* These selected problems are then shown to users as part of the contest interface, along with a countdown timer of **1 hour**.
* Users must attempt and solve as many of the problems as they can within the given time.
* After submission, LeetCoins are awarded based on the number of correct solutions.

This feature not only encourages time-bound problem solving but also gives users a chance to earn extra LeetCoins, which they can use to unlock premium problems on the platform. It brings a competitive edge while promoting consistency and learning.

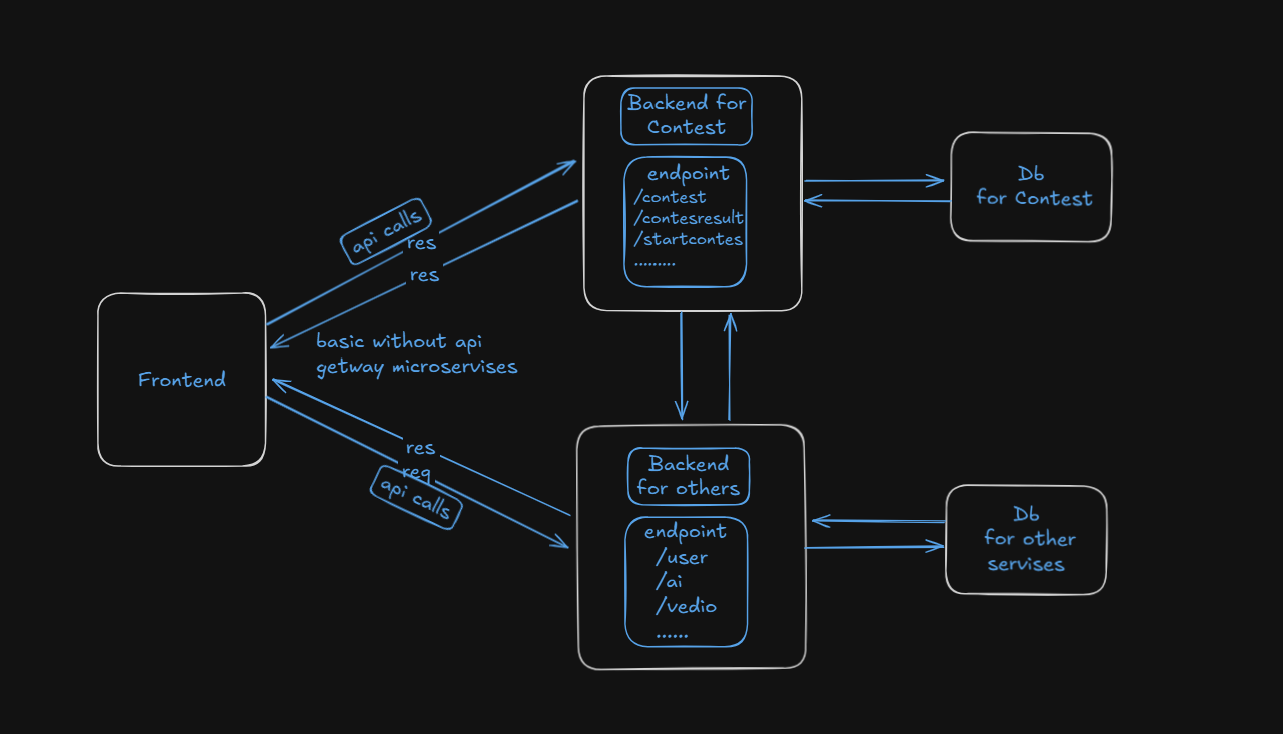
**11. MICRO SERVICES ARCHITECTURE**

To ensure scalability and modularity in my platform, I have implemented a **basic microservices architecture**. Although I haven't yet integrated an API Gateway (as I'm still exploring its complexity), I’ve structured the system into **separate services**, each with its **own dedicated database**, which aligns with the microservices principle of decentralized data management.

One key example is the **Contest Service**, which is isolated from other parts of the system. Since contests are expected to handle higher traffic—especially during live events—I designed this service independently so it can scale separately when needed. It manages everything from contest creation, random question selection, and user participation to time tracking and reward distribution.

Other services, such as user authentication, problem management, and the POTD system, are also separated logically. Each service handles a specific domain of functionality, ensuring that the platform remains maintainable, modular, and ready for future expansion.

This architecture forms a solid foundation for transitioning into a full-fledged microservices system with API Gateway and service-to-service communication in the future.



**DATABASES->**

**1.MONGO dB**

For data storage, I have used **MongoDB** as the primary database in my platform. MongoDB's flexible, **schema-less design is well-suited for the dynamic and evolving structure of a platform like mine, which** includes features such as DSA problems, POTD, contests, user submissions, rewards, and editorial videos. Each of these modules has its own data structure, and MongoDB allows me to store and manage this data efficiently without rigid table definitions.

In the context of my **microservices architecture**, each service (like the Contest Service, POTD Service, etc.) has its **own dedicated MongoDB database or collection**, ensuring loose coupling and independent scalability. MongoDB also makes it easy to store nested data like problem tags, user submissions, and editorial metadata, which is perfect for a platform focused on coding practice and learning.

Using MongoDB has helped me keep development fast, storage scalable, and services independent — aligning well with the architecture and vision of the platform.

**Schemas->**

**For other**

* **users** – Stores user profile, progress, and wallet (e.g., LeetCoins).
* **problems** – Contains all DSA problems (title, description, difficulty, tags, etc.).
* **solutionvideos** – Stores metadata for editorial videos linked to each problem.
* **potds** – Daily featured problems for POTD (Problem of the Day).
* **submissions** – Tracks each user's code submissions, verdicts, and timestamps.
* **feedbacks** – Stores user feedback on problems, videos, or the platform itself.

**For contest**

 **contestdetails** – Stores metadata for each contest (start time, duration,endtime.).

 **contest-**in this I store the problem that comes in contest

 **designcontests** – stores contest participants and there solve problems and time.

**2.REDIS**

Redis is used as a **fast, in-memory data store** to handle use cases that require quick access, temporary storage, and automatic expiration.

I have integrated **Redis** into my platform to enhance performance and handle temporary, time-sensitive features efficiently. Redis is used in several key areas. Firstly, for the **logout functionality**, I implemented token blacklisting using Redis. When a user logs out, their JWT is stored in Redis with an expiry time equal to the token’s remaining validity. This ensures that even though JWTs are stateless, logged-out tokens cannot be reused, providing a secure logout experience. Secondly, Redis is used in the **POTD (Problem of the Day)** feature to prevent users from earning multiple LeetCoins by solving the same problem more than once in a day. A unique key combining the user ID, problem ID, and date is stored in Redis with a 24-hour expiry. Lastly, Redis is utilized across the platform for various purposes such as contest participation tracking, rate limiting, and caching leaderboard data. By offloading these responsibilities to Redis, I’ve kept the system fast, secure, and scalable without putting unnecessary load on the main database.

**3.SUPABASE**

In my platform, I have used **Supabase as a vector database** to implement **optimized semantic search** on video transcripts. Since editorial videos often contain long and detailed explanations, storing and processing the entire transcript for every user query would be inefficient and lead to high token usage in LLM interactions. To solve this, I used Supabase's **pgvector extension**, which allows storing video transcript chunks as **embeddings (vector representations)** generated using an LLM. When a user asks a question during a video, their query is also converted into a vector, and a similarity search is performed in Supabase to retrieve only the most relevant transcript chunks. These selected chunks are then passed to the AI for answering, ensuring both accuracy and efficiency. This approach significantly reduces token consumption, speeds up response time, and provides highly contextual and meaningful answers based on the video content.

**RESOURCES->**

While building this platform, I followed a variety of resources to ensure each feature was thoughtfully designed and well-implemented. I applied the **First Principles Thinking** approach to break down problems and build solutions from the ground up, rather than relying on pre-built templates or shallow implementations. For the frontend, I referred to **DeepSeek** for design guidance and implementation help, which allowed me to maintain a clean and user-friendly UI. On the **backend**, I wrote all the logic independently, relying solely on official documentation for libraries, frameworks, and tools. I intentionally avoided using shortcuts or copying code, to deepen my understanding of how everything works under the hood. Additionally, **AI (like ChatGPT)** played a significant role in assisting with architectural decisions, refining ideas, and solving complex problems efficiently. This blend of independent learning, documentation, and AI-assisted development helped me build a robust and scalable platform.

**Thank You**