Name: Vishnu Vardhan Reddy
RollNo:2300031042

Project Overview

We are planning to build **LogicFox**, a modern, scalable code evaluation and learning platform focused on strengthening both conceptual understanding and hands-on coding skills.

The primary objective of LogicFox is to help students and developers improve problem-solving abilities across **Data Structures & Algorithms (DSA), SQL, Object-Oriented Programming (OOP), Aptitude, and Web Development**.

The platform also aims to simulate real-world **competitive programming and technical interview environments** through features such as contests, leaderboards, and performance-based evaluation.

LogicFox is designed with scalability, performance, and real-time feedback in mind, making it suitable for both academic learning and competitive practice.

Tech Stack

Frontend

- **Next.js (v16)** – Used to build a fast, SEO-friendly, and scalable web application with server-side rendering and modern routing.
- **Tailwind CSS (v4)** – Enables rapid UI development with a consistent, responsive, and utility-first design system.
- **Monaco Editor** – Provides an IDE-like coding experience directly in the browser.
- **Framer Motion** – Adds smooth animations and improves overall user experience.
- **React Hook Form + Zod** – Manages form state and validation in a clean, scalable, and type-safe manner.

- **Lucide React** – Supplies lightweight, clean, and customizable icons.

Backend

- **Next.js API Routes / Server Actions** – Handles backend logic while keeping frontend and backend tightly integrated.
- **Prisma ORM** – Ensures type-safe database access and simplifies schema management.
- **PostgreSQL** – Serves as the primary relational database for users, problems, submissions, contests, and coin balances.

Queue & Execution System

- **Redis** – Used for high-speed queuing, caching, and pub/sub communication.
- **BullMQ** – Manages job queues reliably for handling code submissions and retries.
- **Judge0** – Executes user-submitted code in isolated environments and returns execution results securely.

Authentication & State

- **Better Auth** – Manages authentication, sessions, and role-based access control.
- **Next Themes** – Handles dark and light mode preferences.

The Features & Modules

1. Data Structures & Algorithms (DSA)

Experience:

A split-screen interface where:

- Left side displays the problem description written in Markdown.
- Right side contains the Monaco Code Editor.

Verification:

Solutions are evaluated using standard input/output-based test cases.

UI Enhancement:

Resizable panels implemented using react-split for a better coding experience.

2. SQL & Database Logic

Challenge:

SQL problems cannot be validated using standard output alone.

Implementation Strategy:

- Provide users with a read-only PostgreSQL database instance.
- Execute the user's SQL query.
- Compare the resulting dataset with the dataset produced by the official solution query.
- Validate correctness based on result matching.

3. Object-Oriented Programming (OOP) & Aptitude

Format:

- Multiple Choice Questions (MCQs)
- Fill-in-the-blanks questions

Storage:

Questions are stored in PostgreSQL using JSON fields for flexibility.

State Management:

Use **Dexie (IndexedDB)** to store quiz progress locally so users do not lose progress on page refresh or accidental navigation.

4. Web Development Evaluation

Core Challenge:

Evaluating complete frontend or backend applications such as React, Node.js, or Spring Boot projects.

Evaluation Strategy:

- **Visual Validation:**

Render user output inside a sandboxed iframe.

- **Automated Testing:**

Use unit tests (Jest/Vitest) executed inside the Judge0 container to verify:

  - Component rendering
  - Event handling
  - Expected outputs and behaviors

The Gamification Layer

The Coin System

A reward-based system designed to encourage consistent practice and engagement.

Coin Allocation Logic:

- Easy Problems → +5 Coins
- Medium Problems → +10 Coins
- Hard Problems → +15 Coins

Coins act as an incentive mechanism and can later be extended for rankings, badges, or unlockable features.

Contests & Leaderboards

Real-Time Updates:

Leaderboards are updated live during contests using polling.

Performance Optimization:

- Leaderboard scores are not recalculated from SQL on every request.
- Each successful submission updates a **Redis Sorted Set (ZADD)**.
- Fetching top users is efficient and fast (O(log N)).

Action Plan

Phase 1: The Skeleton (Weeks 1–2)

- Configure authentication using Better Auth with GitHub and Google providers.
- Design and implement the Prisma schema:
  - User
  - Problem
  - Submission
  - TestCase
- Build the execution pipeline:
  - Set up a local Redis instance.
  - Create a BullMQ producer inside Next.js.
  - Implement a separate BullMQ consumer service.
  - Connect the consumer to Judge0 for code execution.

Phase 2: Core DSA Engine (Weeks 3–4)

- Develop the problem-solving UI using Monaco Editor and react-split.
- Implement the **Run Code** feature to trigger the full execution pipeline.
- Display execution results such as:
  - Success
  - Runtime Error
  - Time Limit Exceeded

using toast notifications (sonner).

Phase 3: Expanding Domains (Weeks 5–6)

- **Aptitude & OOP:**
  - Build an MCQ interface.
  - Validate answers using react-hook-form and zod.
- **Web Development Problems:**
  - Introduce a new problem type that accepts HTML, CSS, and JavaScript.
  - Render output in a secure, sandboxed iframe.

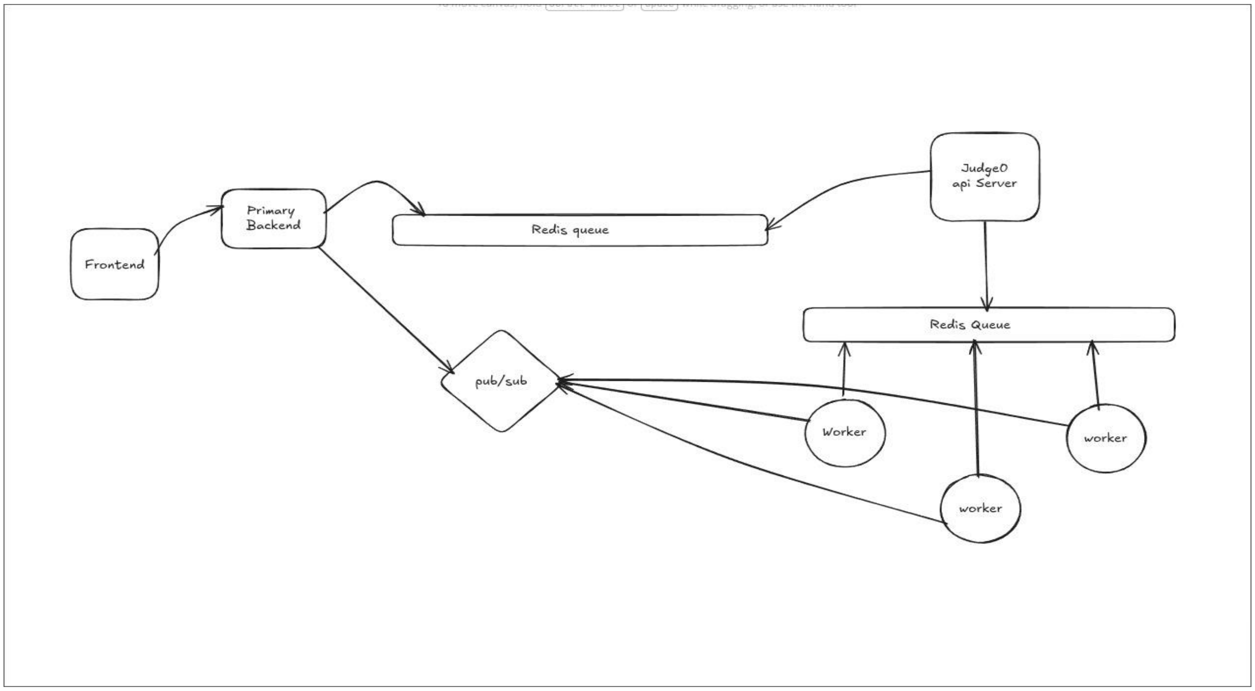
Phase 4: Community & Scale (Weeks 7–8)

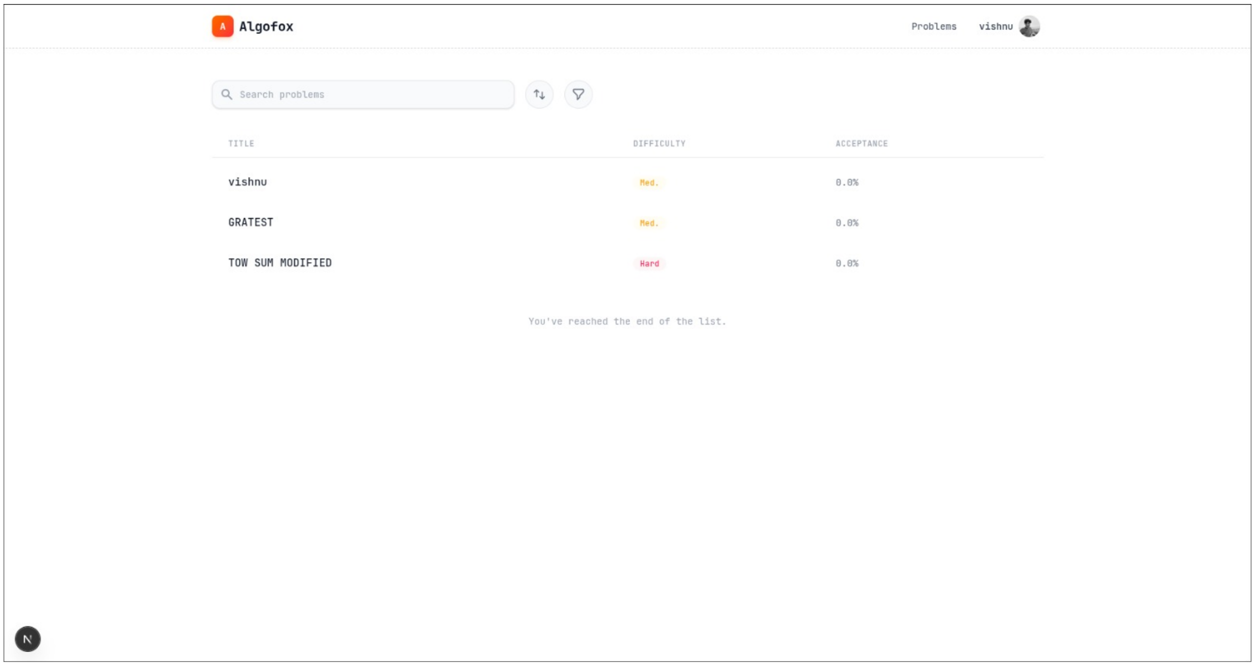
- Implement contest scheduling with start and end times.
- Build leaderboard logic using Redis Sorted Sets.
- Integrate the coin economy by incrementing user balances on successful submissions.

Phase 5: Polish & Deployment (Week 9+)

- Add dark/light mode toggle using next-themes.
- Improve performance with skeleton loaders and optimized assets.
- Deploy:
  - Frontend to Vercel or AWS.
  - Workers and execution services to a VPS (DigitalOcean or EC2), as serverless platforms cannot maintain persistent worker connections efficiently.

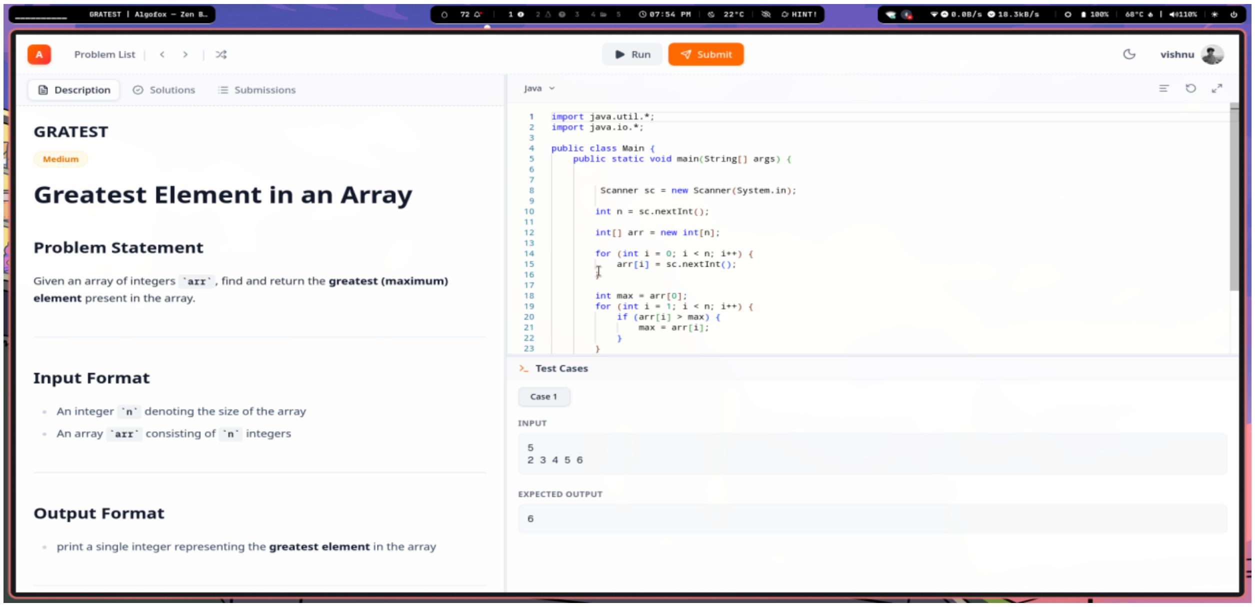
Architecture && Prototype





TITLE	DIFFICULTY	ACCEPTANCE
vishnu	Med.	0.0%
GRATEST	Med.	0.0%
TOW SUM MODIFIED	Hard	0.0%

You've reached the end of the list.



GRATEST

Medium

Greatest Element in an Array

Problem Statement

Given an array of integers `arr`, find and return the **greatest (maximum) element** present in the array.

Input Format

- An integer `n` denoting the size of the array
- An array `arr` consisting of `n` integers

Output Format

- print a single integer representing the **greatest element** in the array

```java
import java.util.*;
import java.io.*;

public class Main {
    public static void main(String[] args) {


        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        int max = arr[0];
        for (int i = 1; i < n; i++) {
            if (arr[i] > max) {
                max = arr[i];
            }
        }
```

Test Cases

Case 1

INPUT

```
5
2 3 4 5 6
```

EXPECTED OUTPUT

```
6
```