Software Engineering Assignment: System Design Document

Himanshu Shete - IIT Bombay - 23B0770 - 23b0770@iitb.ac.in

Project Overview

- **Project Name**: **DSA Prep Agent**
- **Purpose**: An AI agent that automates the manual task of analyzing Data Structures and Algorithms (**DSA**) submissions and generating personalized learning recommendations.

Manual Task Being Automated

Original Manual Process

Task: Analyzing Codeforces submission history to identify learning gaps and recommend next problems.

Step	Description
1	Visit Codeforces profile
2	Review recent submissions
3	Identify patterns in failures (wrong answers, timeouts)
4	Determine topics that need practice
5	Search for appropriate problems to practice next
6	Manually curate a list of recommended problems

-

  Time Investment: 30-60 minutes per analysis session
- **Frequency**: Weekly or bi-weekly

Automated Solution

The DSA Prep Agent automates this entire workflow, reducing the analysis time from 30-60 minutes to **30 seconds**.

1. Fetches submission history via **Codeforces API**.
2. Analyzes each submission using **AI**.
3. Identifies learning gaps and patterns.
4. Generates personalized problem recommendations.
5. Provides structured insights and action items.

System Architecture

High-Level Architecture

The system uses a microservices approach with a multi-agent structure.

Component	Technology	Port	Role
Frontend	Next.js	3000	User Interface
API Gateway	Node.js (Express)	5000	AI Chat Proxy
Core Orchestrator	Python (FastAPI)	8000	Multi-agent coordination
Core AI	Gemini API	N/A	General AI and Fallback

Component Breakdown

Component	Technology	Purpose & Key Responsibilities
1. Frontend	Next.js 13, React 18	User interface: AI Chat and DSA Recommendations interface.
2. Node.js Backend	Express.js, Axios	Simple API gateway for AI chat; Proxies to Gemini API.

3. Python FastAPI Backend	FastAPI, Python	**Core orchestrator**: Coordinates multi-agent workflow and complex reasoning.
4. Analyzer Agent	Fine-tuned LLM (LoRA) + Gemini	Analyze individual submissions, extract topics, infer issues/difficulty.
5. Planner Agent	OpenAI GPT-4o-mini / Gemini API	Synthesize analysis, identify learning patterns, generate tailored recommendations.
6. Codeforces Client	Python requests	External integration: Fetch user submissions and problem metadata.

Multi-Agent Collaboration

Agent Communication Flow

1. **User Input** (Codeforces Handle) → **FastAPI Backend** (Orchestrator).
2. Orchestrator invokes: **Codeforces Client** fetches submissions.
3. **Analyzer Agent** (Parallel) analyzes *each* submission, returning structured analysis.
4. **Planner Agent** receives all analyses, synthesizes patterns, and generates recommendations.
5. **(Optional) Evaluator** measures quality and tracks reliability.
6. **Response to User**.

Reasoning, Planning, and Execution

- **1. Reasoning (Analyzer Agent)**: Analyzes submissions, reasons about likely failure causes, and identifies knowledge gaps.
- **2. Planning (Planner Agent)**: Synthesizes patterns across submissions, plans learning trajectory, and selects appropriate problems.
- **3. Execution**: Fetches data, coordinates agent workflows, and generates actionable recommendations.

Data Design

Data Flow

Codeforces API → Submissions → Analyzer → Analysis → Planner → Recommendations → User

Key Data Structures (Examples)

- **Submission Object**: {"id": 123456, "verdict": "OK", "tags": ["math", "brute force"]}
- **Analysis Object**: {"topics": ["math", "brute force"], "likely_issue": "No issues, solved correctly", "difficulty_inference": "easy"}
- **Recommendations Object**: An array of objects with **tailored problem recommendations** including title, link, difficulty, and reason.

Storage

- **Logs**: JSONL files for interaction history.
- **Evaluation Results**: JSONL files for metrics tracking.
- **Training Data**: JSONL files for fine-tuning.

Technology Choices and Rationale

Component	**Technology**	**Rationale**
Frontend	**Next.js**	Fast development, server-side rendering (SSR), robust React ecosystem.
Backend	**Express + FastAPI**	Express for basic endpoints; **FastAPI** for async support, parallel agent processing, and Python's AI/ML ecosystem.
AI Models	**Gemini 1.5 Flash**	Fast, cost-effective for general chat and fallback.
Specialized Model	**LoRA Fine-tuning**	Parameter-efficient, memory-efficient specialization for structured DSA analysis.

External Integrations

- **Codeforces API**: Used to fetch user submissions and problem metadata. Rate limiting is handled with timeout and retry logic.
- **Gemini API**: General AI chat and primary model for planning/analysis fallback.
- **OpenAI API (Optional)**: Alternative for the Planner Agent. Uses graceful fallback to Gemini if unavailable.

User Interface Design

Design Philosophy

The UI prioritizes **modern UX principles**: Visual Hierarchy, Accessibility, Responsiveness, and clear Feedback.

Main Page (Recommendations)

- **Focus**: DSA recommendations feature prominently.
- **Elements**: Large input field for Codeforces handle, prominent call-to-action, **color-coded difficulty badges**, and direct links to Codeforces problems.

Floating Chat Interface

- **Location**: Bottom-right corner as a floating dialog.
- **Purpose**: General AI chat for DSA questions.
- **Design**: Expandable dialog with conversation-style message bubbles.

Operational and Security Features

Operational Features

- **Monitoring**: Automatic tracking of evaluation metrics and logging.
- **Scalability**: **Stateless backend** design, parallel agent processing, and caching strategies.

Security Considerations

- API keys stored in **environment variables**.
- CORS configured strictly for the frontend domain.
- Input validation on all endpoints.

Deployment Considerations

- Production will require **Containerization (Docker)**, a Reverse Proxy (nginx), and dedicated monitoring infrastructure.

Social Impact and Originality

Social Impact

- **Educational**: Makes DSA learning more accessible and efficient.
- **Time Savings**: Reduces manual analysis time by up to 95%.
- **Accessibility**: Democratizes high-quality, personalized DSA coaching.

Originality

- Pioneering a **Multi-agent collaboration** model specialized for competitive programming.
- Developing a **Fine-tuned model** for structured DSA analysis of failed submissions.
- **Automated analysis** of submission patterns leading to personalized recommendations.

0) Initial Thoughts

like i feel like building something that doesn't take too much input from the user, eg i feel the something like daily water intake calculator is useless, the user will have to input every time they drink water, lets think of something i want a problem to be solved, the dsa prep agent is soooo good infact i need one now, but it seems complex like i will have to develop a oj with a online complier which make learning dsa like a game it learns from your submitted code what can be optimised teaches you that then next question you implement what you learned , ok maybe oh yeah it can be done that my website shows the recommended list of next problem to do(which then redirect to codeforces) or topics to learn but i was thinking of a game like interface which maybe is a overkill, so lets goooo,
//i feel like i did lie 300 problems so for it to analyse all that problems it will be too big so i think like analysing how i solved last few questions also fetching the number of questions solved per topic will help i think there should be simpler method than extracting all the questions like there are some chrome extensions like cf analytics

1) Final product summary (one sentence)

User provides a Codeforces handle (or uploads recent code). The agent fetches recent submissions, automatically analyzes mistakes and topic-weaknesses, and produces a personalized ranked list of next 5 problems (with reasons) and a short learning plan. Frontend shows dashboard + links to CF problems.