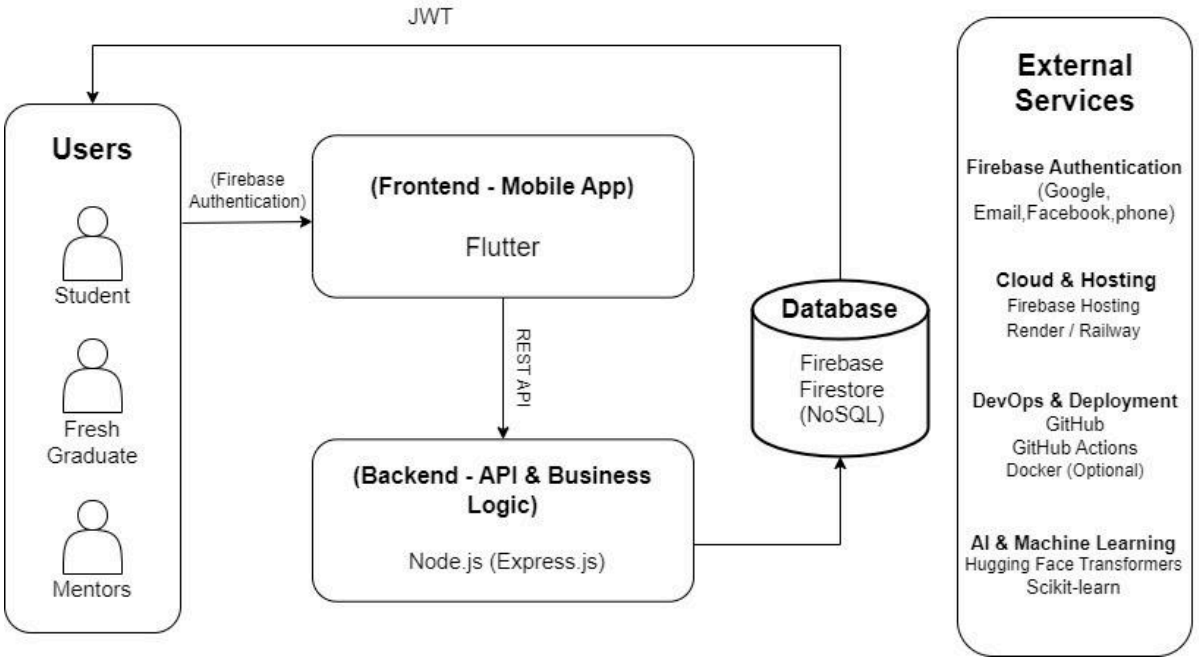
1/ High level architecture diagram



2/ Selected design patterns & justification

1. Observer Pattern

Why Observer?

The **Observer Pattern** is useful when multiple objects need to be notified when another object changes state. It helps maintain a **loosely coupled** system where components interact without being directly dependent on each other.

Justification in Our Project:

In our project, we need real-time updates between different components. Possible use cases include:

- **Notification System:** When an event happens (e.g., new course added, update in schedule), all subscribed users (students, instructors) should be **automatically notified**.

- **User Activity Tracking:** If a student completes a lesson or added a skill closer to his chosen career , an observer can notify the system to update

progress.

- **Live Updates:** If instructors or admins modify content, all relevant users (students, assistants) should be updated in real-time.

---

2. Singleton Pattern

Why Singleton?

The **Singleton Pattern** ensures that **only one instance** of a class exists throughout the application. It is typically used for managing shared resources like database connections, logging, or system-wide configurations.

Justification in Our Project:

Our project requires components that must be **globally accessible** and **unique**. Possible use cases include:

- **Database Connection Manager:** Ensures that only **one** database connection instance is used throughout the system.
- **Logging System:** A single instance of a `Logger` class can be used to track system events.
- **Configuration Manager:** A centralized class managing settings such as **API keys, user preferences, and permissions**.

---

3. Factory Method Pattern

Why Factory?

The **Factory Method Pattern** provides a way to create objects **without exposing the instantiation logic**. This is useful when a system has multiple related objects that share behavior but require different implementations.

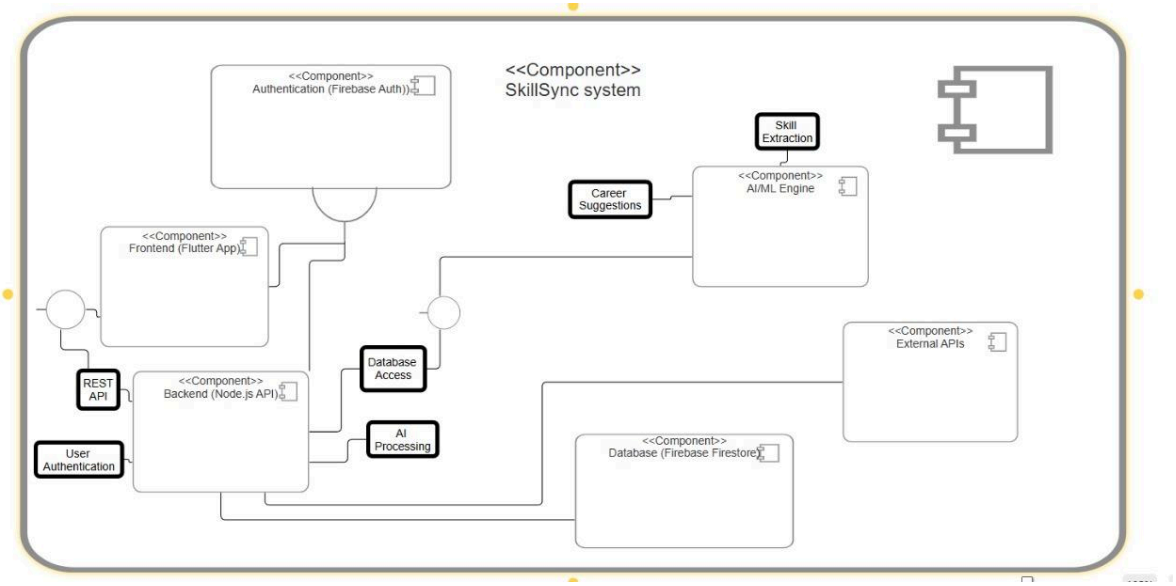
Justification in Our Project:

Our project involves **different types of objects** that should be **created dynamically** based on conditions. Possible use cases include:

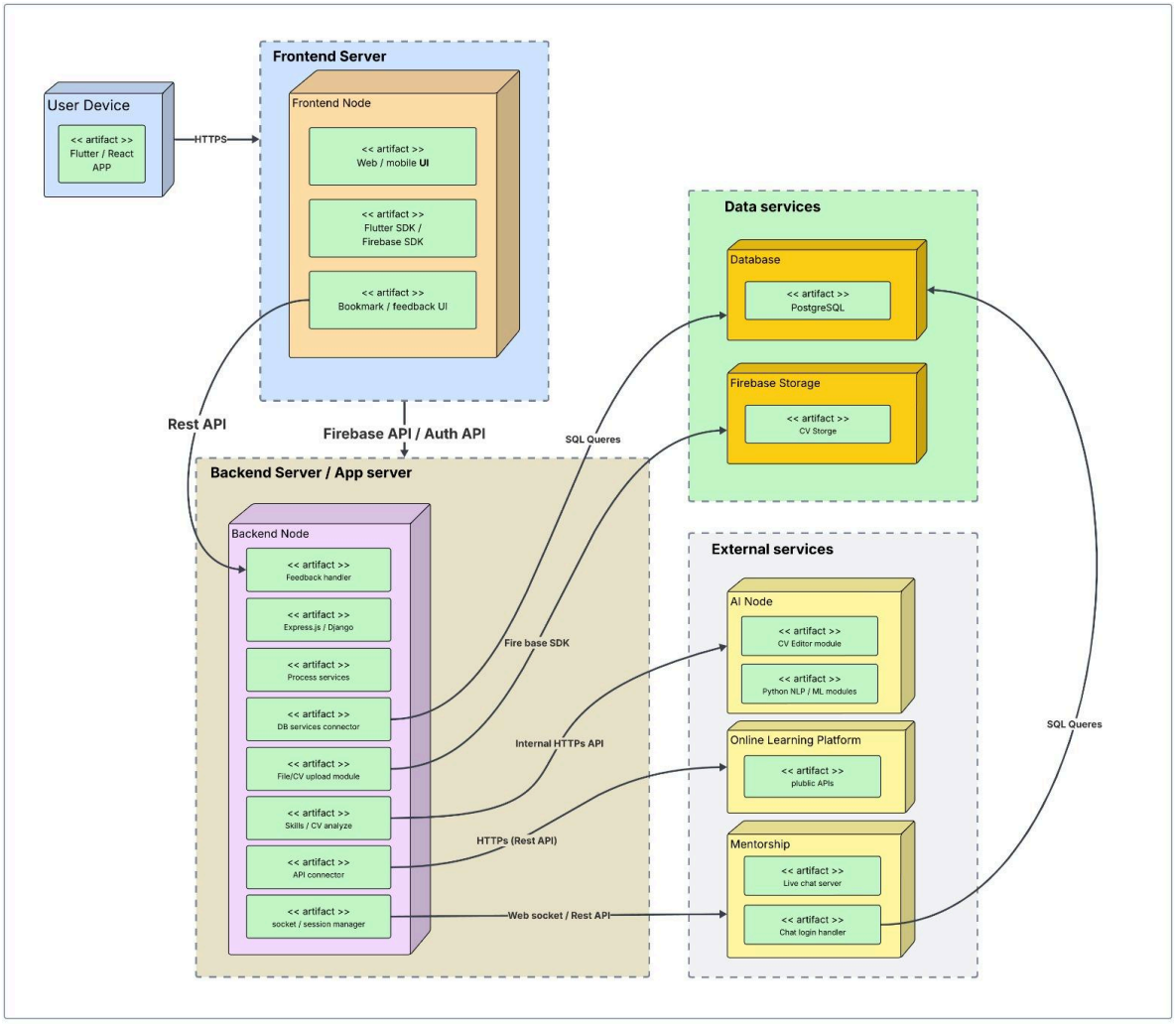
- **User Creation System:** Different types of users (`Student`, `Instructor`, `Admin`) need to be instantiated based on role.
- **Notification System:** Instead of manually creating **email, SMS, or push notifications**, a Factory can determine which type to generate.
- **Exam/Assignment Management:** Different question formats (`MCQ`, `Essay`, `Coding Challenge`) can be created dynamically.

3/ UML diagrams:

Component diagram:

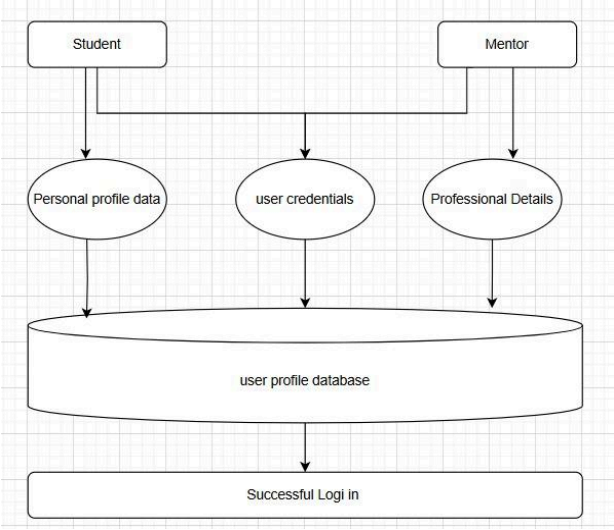


Deployment diagram:

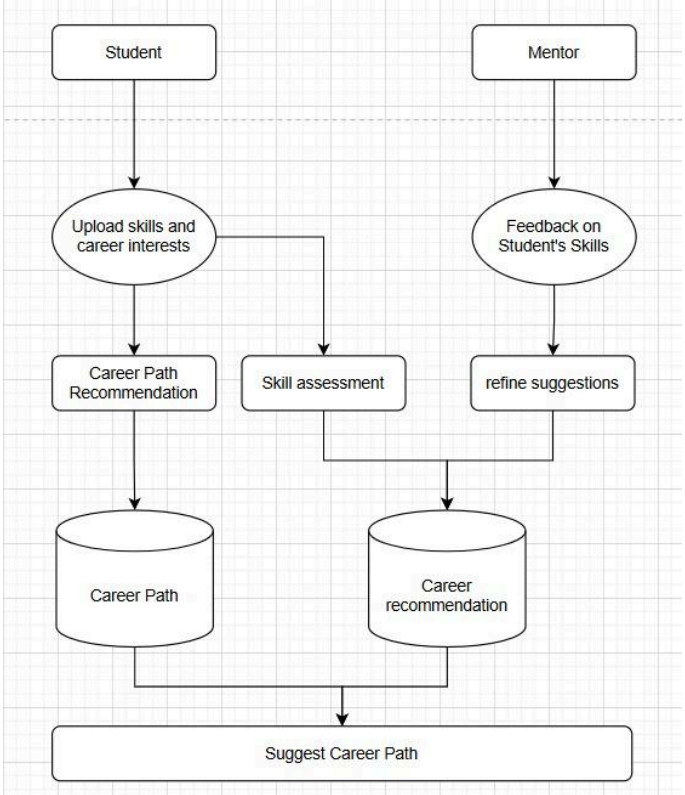


4/ Data flow overview:

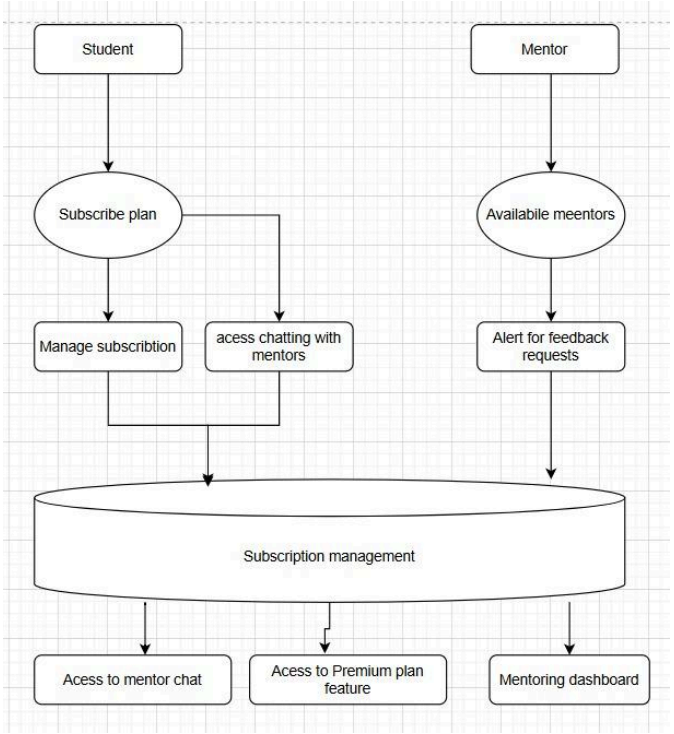
Profile creation:



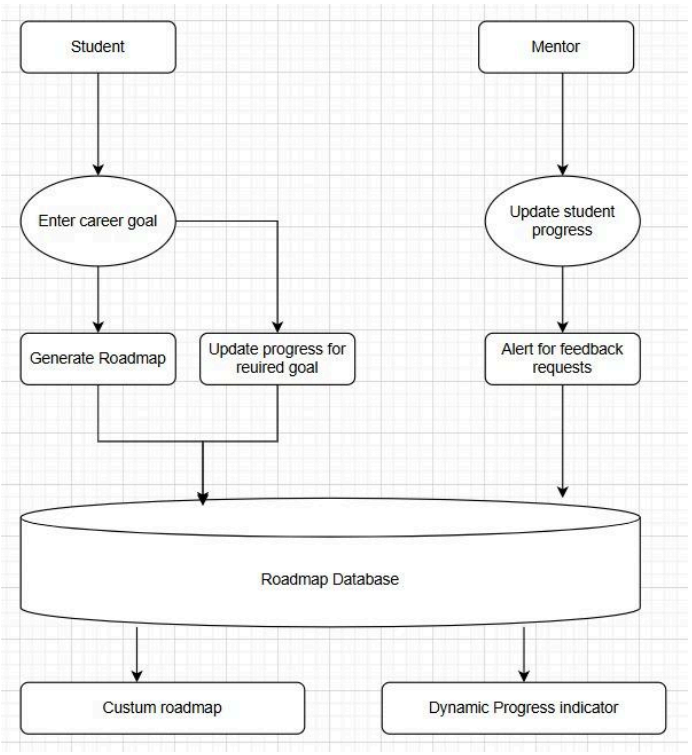
Career path:



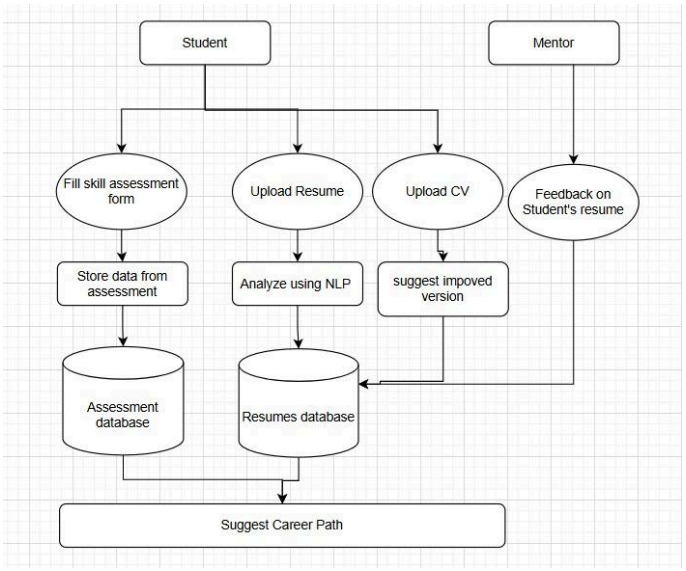
Subscription plan:



Roadmap suggestion:



Career path:


Student
Mentor
Fill skill assessment form
Upload Resume
Upload CV
Feedback on Student's resume
Store data from assessment
Analyze using NLP
suggest impoved version
Assessment database
Resumes database
Suggest Career Path