Career Path App

Deliverable 2:

Architectural Design Document

Authors:	Bobe Andreea Radu Denisa Pakcan Merve Stoicescu Stefan
Status	Draft 1

Summary

Introduction

Design Quality Criteria

The design quality criteria for the Career Path App reflect the non-functional requirements defined in the URD and ensure that the system remains feasible. These criteria focus on performance, reliability, maintainability, usability, and cost-effectiveness.

The system is expected to provide quick and smooth interactions for end users. The **response time** for the main actions—loading assessment pages, submitting questionnaires, generating recommendations—should remain under two seconds for typical workloads. Since the system will not handle high-volume traffic, the **throughput** requirements are modest: it should be capable of serving several dozen simultaneous users without noticeable degradation. **Resource usage** should remain lightweight, relying on common web technologies and a small relational or document-based database, allowing the application to run comfortably on a standard cloud instance or university server with limited CPU and memory resources.

As the system serves an advisory purpose rather than life-critical operations, the necessary level of reliability is moderate but important. The application should operate consistently during normal usage, with no frequent crashes or data loss. Basic **robustness** is required to handle invalid inputs—for example, missing form fields or incorrect data formats—by returning user-friendly error messages rather than failing abruptly. The expected **availability** is approximately 95–98%. Minimal **fault tolerance** is implemented by validating inputs, handling unexpected exceptions gracefully, and maintaining consistent database transactions. Given the system's limited scope, **security** is focused on essential protections such as secure authentication, encrypted connections, and safe storage of personal data. Since the application does not control hardware or physical processes, safety concerns are minimal.

The system must be reasonably easy to maintain. **Extensibility** is supported by a modular architecture with separate components for user management, assessments, recommendation logic, and learning paths. **Modifiability** is facilitated by clear coding practices, separation of concerns, and adherence to standard frameworks. **Adaptability** is moderate: although the system is built specifically for career guidance, the underlying assessment and recommendation modules could be adapted to similar domains such as academic advising or employee training. **Portability** is achieved by using widely supported technologies (e.g., Python/Node back-end and a standard web front-end), making it possible to deploy the system on various platforms with minimal changes. **Code clarity** is ensured through comments, straightforward logic, and consistent naming conventions. **Traceability of requirements** is maintained through documentation mapping features to URD requirements and linking code modules to specific functionalities.

The system focuses on delivering meaningful value to students and professionals. **Usefulness** is demonstrated through personalized career recommendations, structured learning paths, and a clear presentation of strengths and improvement areas. The interface emphasizes **ease of use**, with simple navigation, short forms, visual progress indicators, and concise feedback so users can quickly understand their results and next steps.

The cost of designing and implementing the system is kept minimal, as expected. Development tools are open-source, hosting can rely on a low-cost cloud instance or institutional server, and no specialized hardware or paid APIs are required. The overall cost primarily involves developer time and basic infrastructure, making the system feasible.

Architecture

The system is structured into **four main subsystems**:
User & Profile Management, **Career Guidance**, **Mentorship & Scheduling**, and **Admin & System Management**.

Subsystems

1. User & Profile Management Subsystem

This subsystem handles user registration, authentication and personal profile management. Users can manually enter education, skills, experience and interests. Advanced CV parsing is not required for the prototype, manual input is considered sufficient.

Network:
Secure communication via HTTPS between client and server.

Communication / Interfaces:
REST API endpoints such as /register, /login, /profile using JSON format.

Technologies (could be used):

- React (web interface)
- Optional prototype mobile app with React Native
- Node.js with Express for backend
- JWT-based authentication

Server:
Could run on a local / university development server. Docker may be used only for deployment simulation.

Database:
PostgreSQL (or MySQL) could be used to store user data. Passwords should be hashed, and sensitive fields encrypted.

2. Career Guidance & Learning Subsystem

This subsystem analyzes the user profile and generates a personalized career pathway. Instead of machine learning, a **rule-based recommendation method** can be used to keep the prototype achievable. A roadmap is generated and the user's progress is tracked.

Network:
Data can be transferred securely from the profile subsystem via HTTP requests.

Communication / Interfaces:
REST API endpoints such as /recommendations, /roadmap, /progress.

Technologies (could be used):

- Node.js / Express backend
- Rule-based matching logic (no ML required)
- Simple task management structure for roadmap tracking

Server:
Runs on the same backend server as other subsystems.

Database:
Tables for job roles, skills, roadmap tasks and user progress (PostgreSQL/MySQL).

3. Mentorship & Scheduling Subsystem

This subsystem connects users with available mentors and simulates session scheduling. Real calendar integrations are not required; simple scheduling logic is sufficient for the prototype.

Network:
All mentor-related communication could be performed over HTTPS.

Communication / Interfaces:
REST API endpoints: /mentors, /request-session, /schedule.

Technologies (could be used):

- React for frontend
- Node.js / Express for backend
- Simple scheduling logic

Server:
Runs on the same backend server as other subsystems.

Database:
Relational tables for mentor profiles, availability slots and session bookings.

4. Admin & System Management Subsystem

This subsystem allows authorized administrators to manage users, mentors and system settings. Instead of complex analytics tools, simple log tracking may be used for monitoring.

Network:
Admin access should be restricted and secured via HTTPS.

Communication / Interfaces:
Protected endpoints such as /admin/users, /admin/settings (with authorization).

Technologies (could be used):

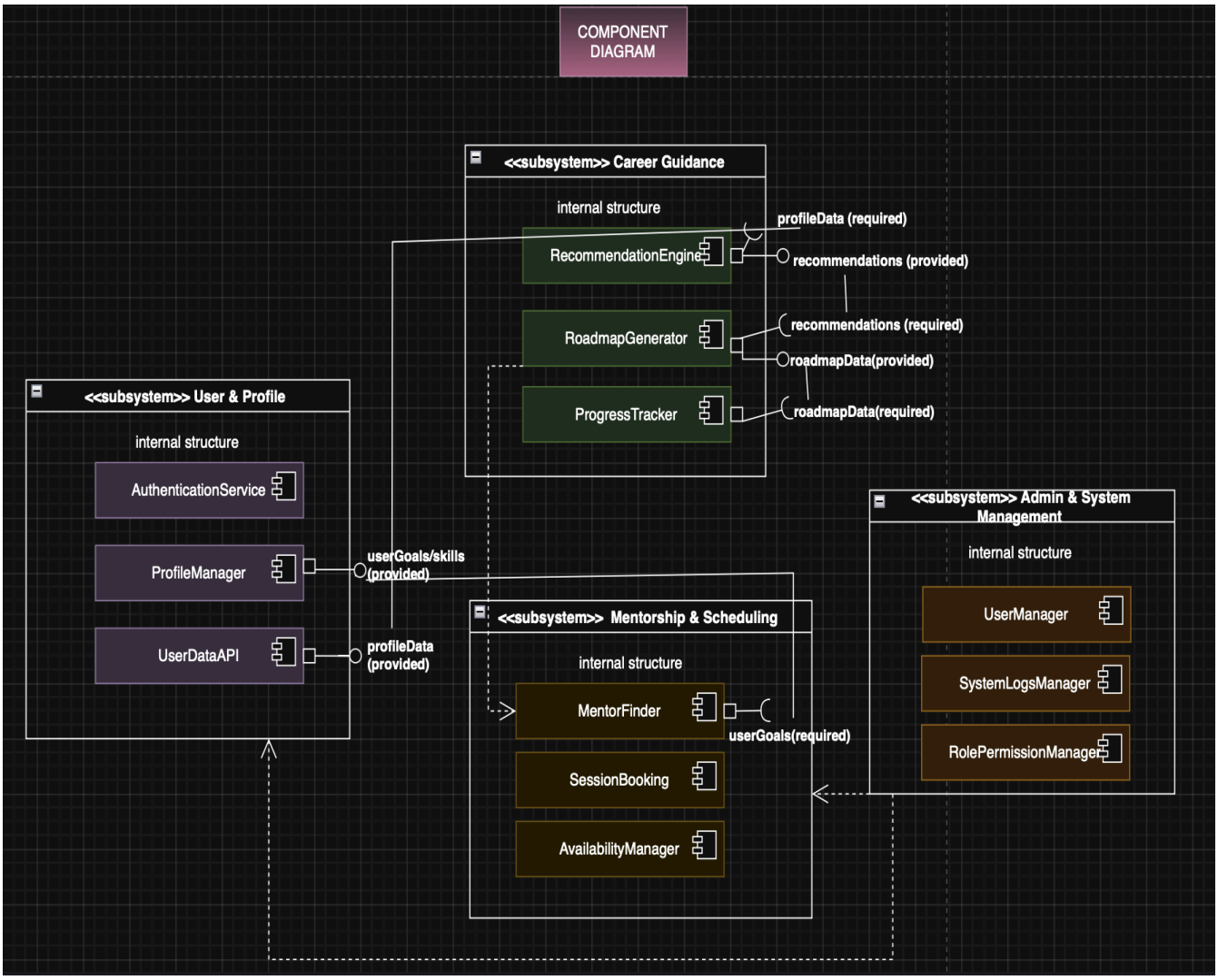
- Basic React-based admin dashboard
- Node.js / Express backend
- JWT + role-based authorization

Server:
Runs on the same backend server as other components.

Database:
Admin roles, logs and configuration data may be stored. Sensitive information must be encrypted and access restricted.

Components & deployment

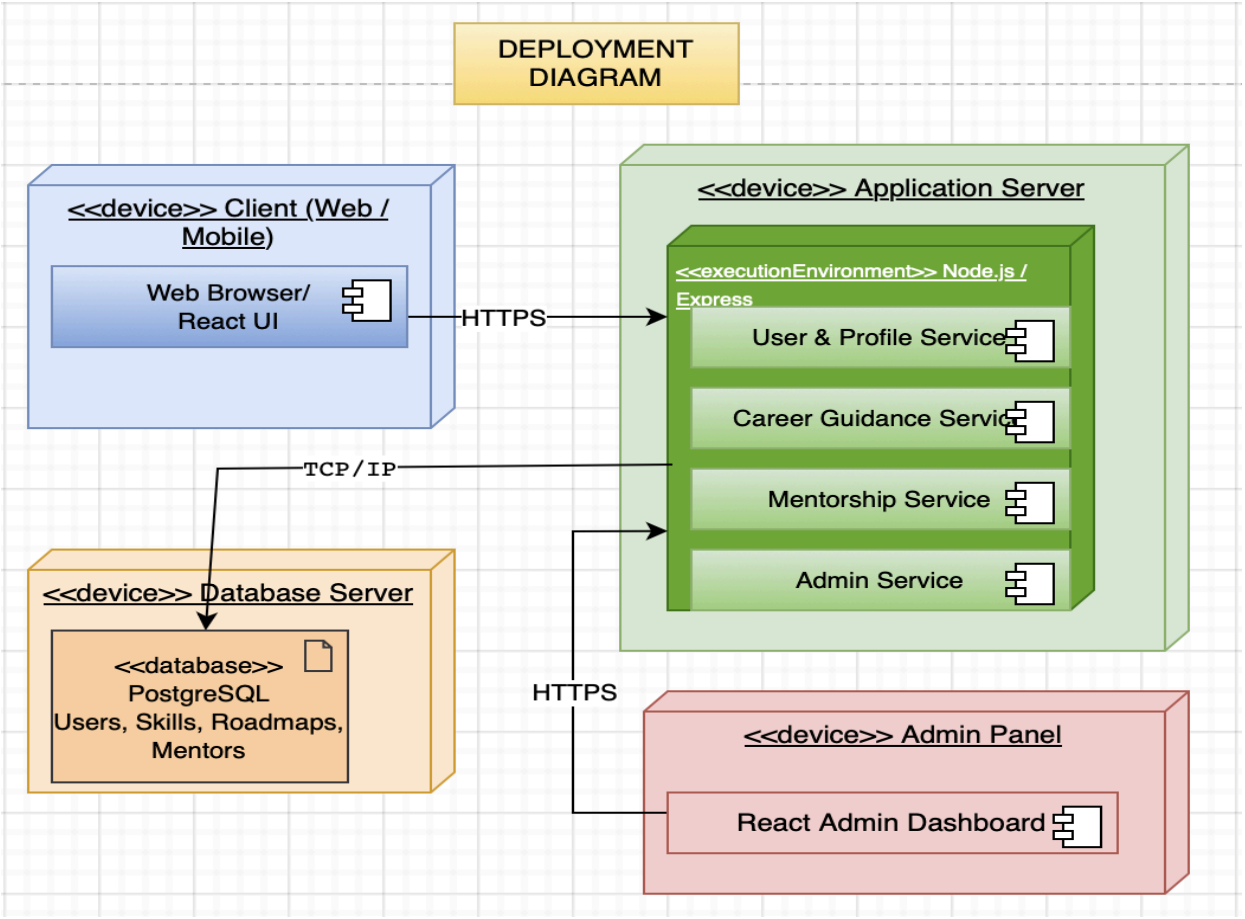
1.Component Diagram



This component diagram shows the main structure of the system and how the subsystems communicate with each other. The architecture is divided into four subsystems: **User & Profile**, **Career Guidance**, **Mentorship & Scheduling**, and **Admin & System Management**. Each subsystem includes its own components, and they interact through provided and required interfaces. For example, the **UserDataAPI** provides profileData, which is required by the **RecommendationEngine** in the Career Guidance subsystem. Also, the **userGoals/skills** information from the ProfileManager is used by both the Career Guidance and Mentorship subsystems.

The diagram helps to clearly show **what each part of the system does** and **how data flows between subsystems**. It also supports a modular and scalable design, meaning that components can be updated or replaced without affecting the whole system. Overall, the component diagram demonstrates a well-structured architecture with clear responsibilities and organized communication between components.

2.Deployment Diagram

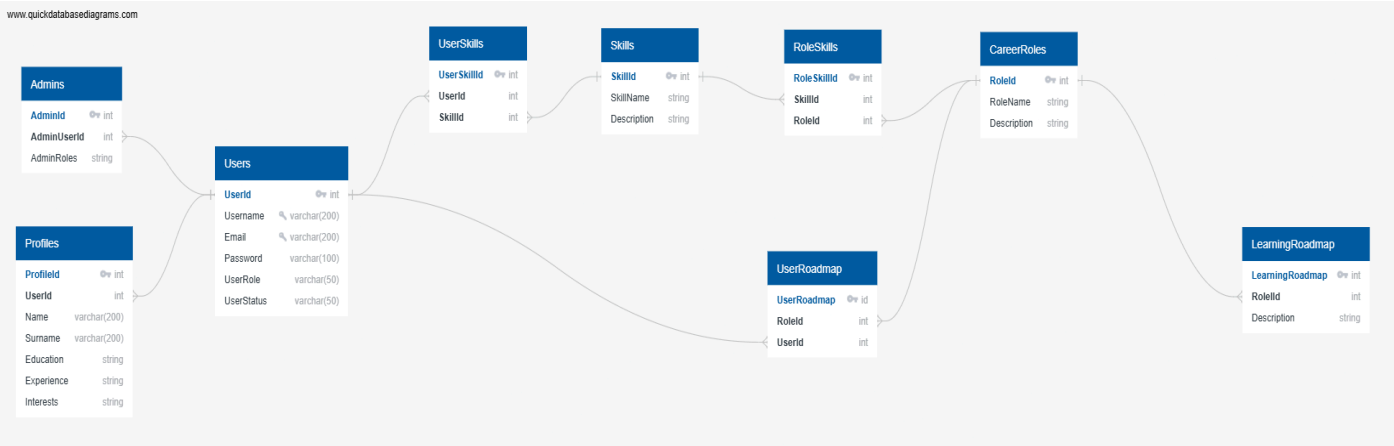


The deployment diagram illustrates how the system is physically deployed across different devices and servers. The **Client (Web/Mobile)** communicates with the backend through HTTPS using a **React-based user interface**, while all core services are hosted on a single **Application Server** running **Node.js and Express**. This server contains four main services: User & Profile, Career Guidance, Mentorship, and Admin, which work together through REST APIs.

A separate **Database Server** (PostgreSQL) stores user profiles, skills, roadmaps, and mentor data. It communicates with the application server via **TCP/IP**, ensuring secure and structured data access. Additionally, there is an **Admin Panel**, which is implemented as a React dashboard and connects to backend services over HTTPS for management and monitoring tasks.

Each software component interacts with the database through REST API calls.
The **User & Profile Subsystem** works with the Users, Profiles and UserSkills tables.
The **Career Guidance Subsystem** uses the Skills, CareerRoles and LearningRoadmap tables to generate recommendations.
The **Mentorship Subsystem** reads mentor information from the Profiles table..
The **Admin Subsystem** accesses all database tables to manage users and system data.

Database diagram



User authentication & authorization

Describe how the user authentication & authorization will be handled.

User Authentication

1. A user opens the app and presses **Register**.
2. They enter: **name, email, password, and role** (Student or Professional).
3. The server checks if the email is new.
4. The password is hashed so nobody can read it.
5. The user is saved in the database.

Login (enter the app)

6. The user presses **Login**.
7. They enter **email and password**.
8. The server checks if the password matches the stored hashed password.
9. If correct ⟶ the server creates a **login session / token**.
10. The user is now logged in and can use the app.

Authorization

11. Every user receives a **role badge** after login. The system checks the role **before opening a page or doing an action**.

- **Student & Professional**
    - They can **see and edit only their own profile**.
    - They can **view recommendations**, create a roadmap, track progress, and request a mentor session.
    - They cannot access admin panels or other users' data.

- **Mentor**
    - They can see only the sessions they are part of.
    - They can add notes after a session.
    - They cannot delete users or change system settings.
- **Admin**
    - They can manage everything:
      - create/update/suspend/delete users
      - edit career roles, skills, learning roadmaps, mentors
      - view logs and system status
    - They are blocked from anything outside their admin role.

Security (extra protection)

13. The app always uses **HTTPS secure connection** (safe internet tunnel).

14. Password input field only shows dots or stars.

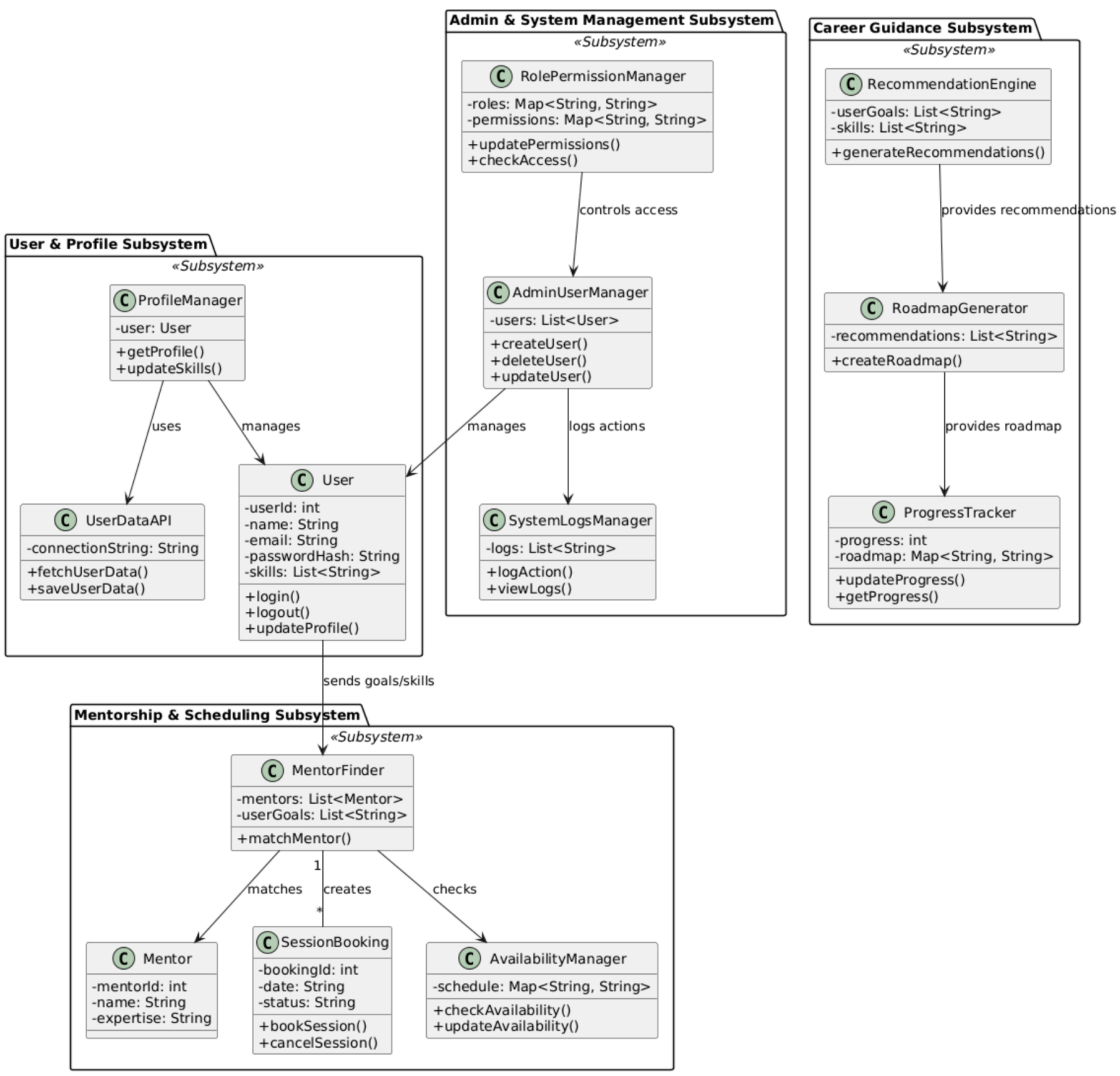
15. Tokens/sessions expire after a short time if the password is changed, so users must login again.

16. If a user tries something forbidden:

- the system sends **401 (not logged in)** or **403 (no permission)** messages.

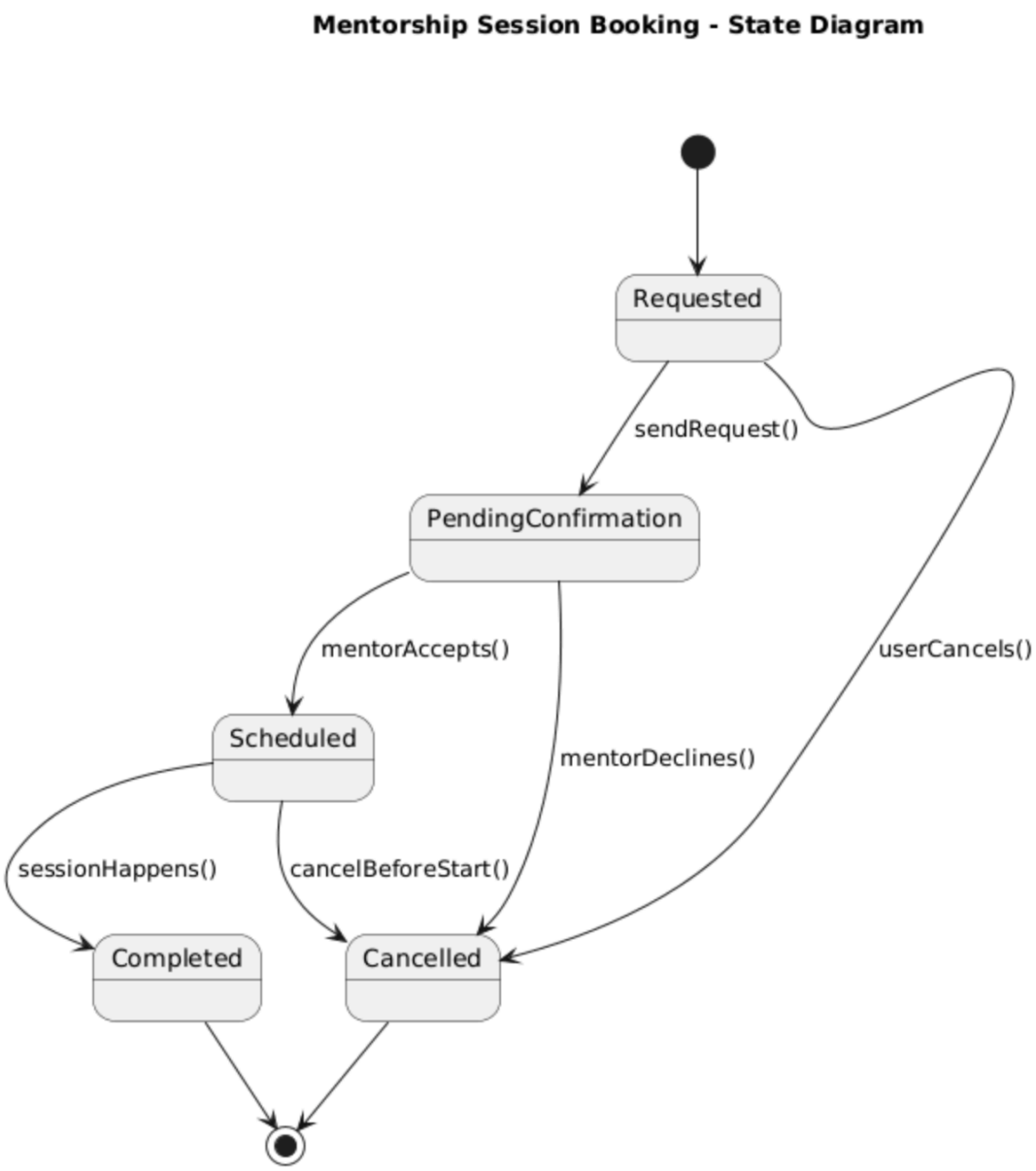
Additional workflow details

CLASS DIAGRAM



The **Class Diagram** shows the main building blocks of the app, grouped into four subsystems: **User & Profile Management, Career Guidance, Mentorship & Scheduling, and Admin & System Management**. Each subsystem contains classes that either store data (like User and Mentor) or handle specific features (like ProfileManager, RecommendationEngine, RoadmapGenerator, SessionBooking, and AdminUserManager). The relations between the classes show how the system works—for example, only one user can have many roadmaps, mentors can be matched by MentorFinder, and admins control access using RolePermissionManager and log actions using SystemLogsManager.

STATE DIAGRAM



The **State Diagram** explains how a mentorship session booking behaves over time. It starts when a user creates a request (**Requested**), moves to **PendingConfirmation** while waiting for a mentor response, then becomes **Scheduled** if the mentor accepts. After the meeting happens, it reaches **Completed**, or it can reach **Cancelled** if the user or mentor stops it before the session. Both final states (**Completed** and **Cancelled**) lead to the end point, meaning the booking process is finished.

Plan & management

Team responsibilities

1. Project Lead & Backend Architect

   This development role focuses on the Backend architecture, designing the database, ensuring proper authentication and system coordination.

   Responsibilities:

   -Implement the core backend components
   -Defining the API contracts to align with use cases from UC-1 to UC-7
   -Plan the sprints and monitor meeting the requirements
   -Implement the authentication in a secure way and give role-based access depending on the user type

2. Frontend Architect & UI/UX Designer

   This role is responsible for creating the Web interface, user experience and providing accessibility.

   Responsibilities**:**

   -Create the main pages of the application, Login, Profile, Dashboard, Recommendations
   -Monitor the design and usability requirements (text resizing, contrast modes,interface)
   -Manage frontend state and integrate REST APIs from backend
   -Design profile forms and dashboards according to the use cases

3. Data Engineer & Roadmap Recommendation Logic Developer

   This role will be responsible for creating the career-matching logic, data modeling, and roadmap generation.

   Responsibilities**:**

   -Design the complete dataset for career roles, required skills, learning resources
   -Implement the recommender system for careers using rule-based matching
   -Generate the career roadmap using the user's skills
   -Focus on preparing the job roles, skills, interest tags

4. QA Specialist & Integration Mobile Coordinator

   This development role focuses on the Testing flows, backend–frontend integration, mobile and DevOps.

   Responsibilities:

   -Create the test plans for the authentication process, the profile editing and recommendation accuracy
   -Manage the pull-request workflow, GitHub repository and branches
   -Test end to end the processes
   -Set up GitHub actions for automated builds

Team organization

The team will follow the Agile structure with a 2 weeks sprint duration followed by review and demo, weekly sync meetings and GitHub for backlog.

For development, the following tools will be used:

-Frontend: React.js
-Backend: Node.js
-Database: MySQL
-IDE: Visual Studio Code
-Testing Tools
-Documentation: Markdown, Swagger

The code collaboration workflow will consist of GitHub with Gitflow branching model (main production code, development versions, features).

The pull requests will be thoroughly reviewed, the backend will be monitored by Data or QA members and frontend, by QA or backend members.

The main collaboration channels will be Teams for meetings and calls, GitHub Projects for tasks and shared API contract file by Backend Lead.

Tasks

Backend

-Create the backend project structure
-Set up MySQL and create the initial database tables
-Configure environment variables and basic server routes

-Implement user registration API (POST /register)
-Implement user login API with JWT (POST /login)
-Implement profile endpoints (GET /profile, PUT /profile)

Frontend

-Initialize the React frontend project
-Create basic page structure (Home, Login, Register, Profile)
-Set up React Router and navigation.

-Build Login and Register pages
-Connect authentication pages to backend API
-Build the Profile page (education, experience, skills fields)

Data Logic

-Create initial datasets: skills list and career roles
-Prepare data models for career recommendations

-Expand skills and job-role datasets

-Develop a prototype for career-matching logic

QA & Integration

-Set up GitHub repository and branching workflow
-Prepare initial test plan for main features

-Test end-to-end flow: Register - Login - Profile Update
-Verify backend–frontend connectivity