AspireX: A Web-Based Platform to Connect Students with Mentors for Career Guidance

Summer Professional Enhancement Program (PEP) Classes

Submitted in partial fulfillment of the requirements for the award of degree of

Bachelors of Technology
Computer Science and Engineering

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB

From 23/06/25 to 30/07/25

SUBMITTED BY

Name of student: Harsh Chauhan

Registration Number: 12319734

Signature of the student: *Harsh Chauhan*

Index

Student Declaration

To whom so ever it may concern

I, **Harsh Chauhan 12319734** , hereby declare that the work done by me on "**AspireX**" from **June 2025** to **July 2025**, is a record of original work for the partial fulfillment of the requirements for the award of the degree, **Bachelors of Technology .**

Name of the Student (Registration Number)
Harsh Chauhan (12319734)

Signature of the student
Harsh Chauhan

Dated:
10-8-2025

Training Certificate

AlgoTutor

CERTIFICATE OF COMPLETION

This certificate awarded to

Harsh Chauhan

He has successfully completed their training in

28 Days Competitive Programming at LPU

This certificate confirms that Harsh Chauhan successfully completed the 28 Days Competitive Programming at LPU. The student actively participated in all sessions, demonstrated strong problem-solving skills, and completed all practical assignments with excellence. Their consistent performance and dedication throughout the program were commendable.

Date: August 14, 2025
Certificate No.: 2025.08.14.597574.0042

Manish Sharma
Co-Founder, AlgoTutor Academy

Acknowledgement

The journey of bringing this project, **AspireX** , from a mere concept to a functional reality has been an immense learning experience, filled with both challenges and triumphs. This endeavor would not have been possible without the collective support, guidance, and encouragement from several individuals and institutions. I wish to take this moment to express my profound gratitude to all those who have been instrumental in my academic and personal development throughout this process.

First and foremost, I extend my deepest appreciation to **Lovely Professional University**. The university has fostered an academic environment that not only imparts knowledge but also encourages innovation, critical thinking, and practical application. The resources, infrastructure, and curriculum provided the strong foundation upon which this project was built.

I am particularly indebted to my professors for their exceptional mentorship. Their insightful feedback, technical expertise, and unwavering belief in my vision were the cornerstones of this project's success. They guided me through the complexities with patience, challenged me to refine my ideas, and instilled in me a passion for problem-solving and a commitment to quality. For their constant support and the invaluable time they dedicated to me, I am truly grateful.

Finally, I owe a heartfelt thanks to my family and friends. Their unconditional love, their patience during my long hours of work, and their constant words of encouragement were my bedrock of support. They celebrated my small victories and offered strength during setbacks, and for their unwavering belief in me, I am eternally thankful.

Name of Student :
Harsh Chauhan (12319734)

Chapter-1 INTRODUCTION OF THE PROJECT UNDERTAKEN

In today's competitive professional landscape, students often face a significant gap between academic knowledge and real-world industry expectations. Access to personalized guidance, career advice, and mentorship is crucial for navigating this transition successfully. However, finding qualified mentors who offer affordable and accessible services can be a major challenge. The AspireX project was initiated to address this very problem by creating a dedicated online platform to bridge the gap between students seeking guidance and experienced professionals willing to provide it.

AspireX is a comprehensive mentor-student connection platform designed to facilitate meaningful interactions and skill development. The platform empowers students to find and connect with mentors who can provide a range of services, including mock interviews, CV reviews, and one-on-one guidance sessions. This chapter outlines the primary objectives of the project, its scope, overall importance and applicability, and the work plan followed for its implementation.

Objectives of the work undertaken

The primary objective of this project was to design, develop, and deploy a fully functional, full-stack web application, "AspireX." The key goals set for the project were:

- To build a secure and scalable platform using React JSX for the frontend and Django for the backend.
- To implement a robust user authentication system allowing users to sign up and sign in using either a Google account or a custom email/password combination.
- To create two distinct user roles—Student and Mentor—each with a unique dashboard and functionalities tailored to their needs.
- To enable mentors to create detailed profiles and list their services, such as mock interviews, CV reviews, video calls, and query messaging.
- To provide students with a searchable interface to find and connect with mentors based on their expertise and services offered.
- To develop a community feature where both students and mentors can post and share success stories, fostering a supportive and motivational environment.

Scope of the Work

The scope of the AspireX project was defined to ensure the delivery of a core set of features that fulfill the project's primary objectives.

The project scope includes:

- User Account Management: Functionality for user registration, login, and profile management for both student and mentor roles.
- Frontend Development: The creation of a responsive and user-friendly interface using React JSX, including components for the landing page, user dashboards, mentor profiles, and the community page.
- Backend Development: The implementation of a Django-based backend to manage all business logic, including creating RESTful APIs for data handling, managing user sessions, and interacting with the database.

- Database Management: Designing and managing a database to store user information, mentor services, community posts, and other relevant data.
- Community Feature: A dedicated page where authenticated users can create, view, and interact with posts.

The project scope deliberately excludes:

- An integrated real-time payment processing system.
- In-app video or chat functionality (the platform facilitates connection, but communication happens via external tools).
- A native mobile application for Android or iOS.
- Advanced administrative features such as analytics dashboards or a dispute resolution system.

Importance and Applicability

The AspireX platform holds significant importance in democratizing access to professional mentorship. It addresses the critical need for affordable career guidance, helping students from diverse backgrounds prepare for the job market. By connecting students with industry professionals, the platform helps enhance their interview skills, improve their resumes, and gain valuable insights, thereby increasing their employability.

The applicability of this project is vast. It can be utilized by:

- University Career Services: To provide students with a structured platform to connect with alumni and industry experts.
- Students: To proactively seek guidance and build a professional network.
- Professionals: To offer mentorship, share their expertise, and contribute to the development of the next generation of talent.

The platform is scalable and can be adapted for specific industries or educational institutions, making it a highly relevant and practical solution in the current educational and professional ecosystem.

Chapter 2: Literature Review and Problem Analysis

Introduction

Before embarking on the development of a new platform, it is essential to understand the existing landscape. This chapter presents a literature review that surveys the theoretical importance of mentorship in career development and analyzes the strengths and weaknesses of existing digital solutions. The goal is to provide a scholarly context for the project, identify a clear gap in the market, and thereby build a robust, evidence-based justification for the necessity of a platform like AspireX.

The Role of Mentorship in Higher Education and Career Readiness

The transition from higher education to the professional world is a critical period in a young adult's life. Academic literature consistently highlights mentorship as a key factor in navigating this transition successfully. Studies have shown that students who engage in mentorship relationships report higher levels of career satisfaction, greater confidence in their skills, and often achieve higher salaries early in their careers. Mentorship provides "social capital"—access to networks, insider knowledge, and unwritten rules of an industry that cannot be taught in a classroom. It bridges the gap between theoretical knowledge and practical application by offering personalized guidance tailored to an individual's specific goals and challenges.

However, traditional mentorship models often rely on informal networks, which can inadvertently perpetuate inequality. Students without existing connections in their desired fields are often left at a disadvantage. This creates a clear need for structured, democratized platforms that make quality mentorship accessible to all, regardless of their background or personal connections.

Analysis of Existing Digital Mentorship and E-Learning Platforms

The digital landscape contains various platforms that touch upon mentorship and career guidance. A critical analysis reveals their respective strengths and limitations, highlighting the specific niche AspireX aims to fill.

1. **Professional Networking Platforms (e.g., LinkedIn):**
    - **Strengths**: LinkedIn is unparalleled in its scale, offering a vast network of professionals across every conceivable industry. It provides tools for making connections and its "Career Advice" feature attempts to formalize mentorship.
    - **Weaknesses**: The mentorship aspect is largely informal and unstructured. Students often send unsolicited messages with low response rates. There is no framework for dedicated, service-based interactions like mock interviews or detailed resume reviews, nor is there a mechanism for mentors to be compensated for their time, which can limit the depth of engagement.

2. **Specialized Mentorship Platforms (e.g., MentorCruise, The Muse):**
    - **Strengths**: These platforms are dedicated solely to mentorship. They offer structured programs where users can book sessions with vetted mentors, often for a fee. This formalizes the relationship and sets clear expectations.
    - **Weaknesses**: These platforms are often expensive, targeting working professionals looking to switch careers rather than students on a budget. Their service offerings can be rigid, focusing on long-term coaching rather than the specific, short-term needs of a student (e.g., a single mock interview before a campus placement drive).
3. **Freelance Service Platforms (e.g., Upwork, Fiverr):**
    - **Strengths**: These platforms allow professionals to offer specific, task-based services, including resume writing and career coaching. The review and payment systems are robust.
    - **Weaknesses**: They are not designed for mentorship. The relationship is purely transactional, lacking the guidance and community-building aspects crucial for a student's development. Finding a true "mentor" among a sea of freelancers can be a significant challenge.
    -

Problem Statement Justification: The AspireX Niche

This analysis reveals a clear and underserved niche in the market. Students require a platform that combines the **affordability** and **specificity** of freelance services with the **structured guidance** and **community support** of a dedicated mentorship platform. AspireX is designed to fill this exact gap by:

- Focusing on the specific, high-impact needs of students (mock interviews, CV checks).
- Creating a price point and service structure that is accessible to students.
- Building a supportive community to foster long-term engagement beyond simple transactions.

By addressing these specific pain points, AspireX justifies its development not as another competitor, but as a unique and necessary solution for the modern student.

Chapter 3: TECHNOLOGY STACK AND SYSTEM DESIGN

Introduction

Every successful project begins with a solid plan. For a software application like AspireX, this plan involves two critical early steps: choosing the right tools for the job (the technology stack) and drawing the blueprint for how everything will work together (the system design). This chapter details the thought process behind these decisions. The goal was to select technologies that were not only powerful and reliable but also allowed for rapid development and future growth. This foundational work was crucial to ensure that the final platform would be secure, efficient, and user-friendly.

3.1 Choosing the Right Tools: The Technology Stack

The technology stack is the set of programming languages, frameworks, and tools used to build an application. For AspireX, a full-stack application, this meant making careful choices for the frontend (what the user sees) and the backend (the engine that runs everything behind the scenes).

- **Frontend**: **React JSX** For the user-facing part of AspireX, I chose React, a popular JavaScript library. The decision was based on several key advantages:
    - Component-Based Structure: React allows you to build the user interface out of small, reusable pieces called components. For example, I could create a single MentorProfileCard component and then reuse it across the platform. This made the code cleaner, more organized, and much easier to manage as the project grew.
    - High Performance: React uses a clever technique called a "Virtual DOM" to update the user's screen efficiently. This results in a fast, smooth, and responsive user experience, which is essential for keeping users engaged.
    - Rich Ecosystem: Being one of the most popular frontend libraries, React has a massive community and a wealth of pre-built tools and libraries, which helped speed up development significantly.
- **Backend**: **Django Framework**, For the backend, which handles all the data, user logic, and security, I selected Django, a high-level **Python framework**. Django's "batteries-included" philosophy was a perfect fit for this project.
    - Rapid Development: Django comes with many built-in features right out of the box, including a powerful user authentication system and an automatic admin panel. This saved a huge amount of time and effort, allowing me to focus on building the unique features of AspireX.
    - Security: Handling user data requires a strong focus on security. Django has built-in protections against common web threats like SQL injection and cross-site scripting (XSS), providing a secure foundation from the very beginning.
    - Scalability: Django is used by major companies like Instagram and Spotify, proving it can handle a large number of users and data. This makes it a great choice for a platform like AspireX that has the potential to grow.

3.2 The Blueprint: System Architecture

The system architecture defines how the frontend and backend communicate. AspireX is built on a standard but powerful Client-Server Architecture.

- **The Client (Frontend)**: This is the React application that runs in the user's web browser. Its job is to display the user interface and send requests to the backend whenever the user performs an action, like signing up or searching for a mentor.
- **The Server (Backend)**: This is the Django application. It listens for requests from the client. When a request comes in, the server processes it (e.g., fetches data from the database, checks a password), and then sends a response back to the client.

To manage this communication, I implemented a RESTFUL API (Representational State Transfer Application Programming Interface). You can think of the API as a well-defined menu that the frontend uses to order data from the backend. The frontend sends a request to a specific URL (an "endpoint"), and the backend returns the data in a standardized JSON format, which is lightweight and easy for the frontend to understand.

3.3 Organizing the Data: Database Schema Design

The final piece of the design puzzle was planning the database — the brain where all the platform's information is stored. A well-organized database is crucial for the app to run efficiently. The schema was designed around a few key models:

- User Model: This is the core model, built by extending Django's default user model. It stores essential information for every user, such as their email, password, and whether they are a student or a mentor.
- Profile Models (Student & Mentor): To hold information specific to each user type, I created separate StudentProfile and MentorProfile models. Each of these is linked directly to a User account. A mentor's profile, for example, would store their job title, company, areas of expertise, and the services they offer.
- Post Model: This model was created for the community page. It stores the content of each post and, crucially, links back to the user who wrote it, establishing authorship.

This structure ensures that data is organized logically, avoids redundancy, and can be retrieved quickly and efficiently by the backend. With this solid technical foundation in place, the project was ready to move into the development phase.

Chapter 4: BACKEND IMPLEMENTATION AND DATABASE MANAGEMENT

Introduction

This chapter details the construction of the "engine room" for AspireX—the powerful, server-side backend that drives the entire platform. While invisible to the end-user, the backend is the application's single source of truth, responsible for enforcing business logic, managing all data interactions, and guaranteeing the security and integrity of the system. Its architecture is paramount for the platform's ability to scale, to be maintained over time, and to provide a reliable service.

This implementation phase involved a meticulous process of translating the abstract system design from the previous chapter into concrete, functional code. This included architecting the database schema using Django's models, integrating with the Supabase cloud platform for data storage, building a comprehensive RESTful API to handle all client-server communication, and implementing a multi-faceted, secure authentication system.

4.1 Building the Foundation: Database and Storage with Supabase

The foundation of any data-driven application is its database. The choice was made to use **Supabase**, a modern "Backend as a Service" (BaaS) platform, which provides a powerful, enterprise-grade PostgreSQL database. This decision offloaded the complexity of database management and infrastructure, allowing development efforts to focus on application features while ensuring high performance and scalability.

The Power of the Django ORM

To interact with the Supabase database, **Django's Object-Relational Mapper (ORM)** was leveraged. The ORM is a powerful abstraction layer that translates Python code into SQL queries. This approach has several key advantages:

- **Rapid Development**: It allows for defining the entire database schema using familiar Python classes (called Models), which is significantly faster and more intuitive than writing raw SQL.
- **Database Agnosticism**: While we used PostgreSQL, the ORM makes it relatively easy to switch to another database system in the future with minimal code changes.
- **Increased Security**: By handling the generation of SQL queries, the ORM helps prevent SQL injection vulnerabilities, as it escapes parameters by default.

The Database Migration Process

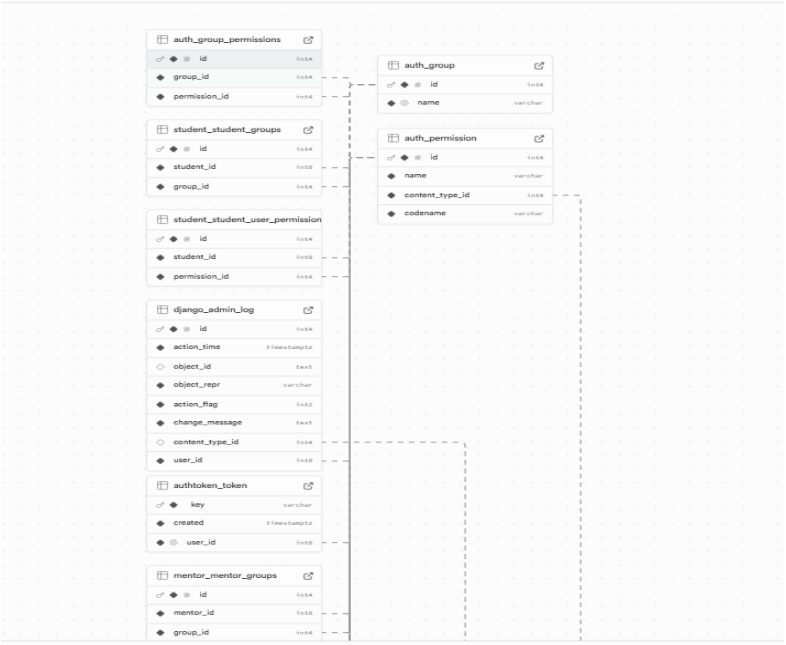
A critical part of working with Django models is the migration system. This system provides a version-controlled history of changes to the database schema.

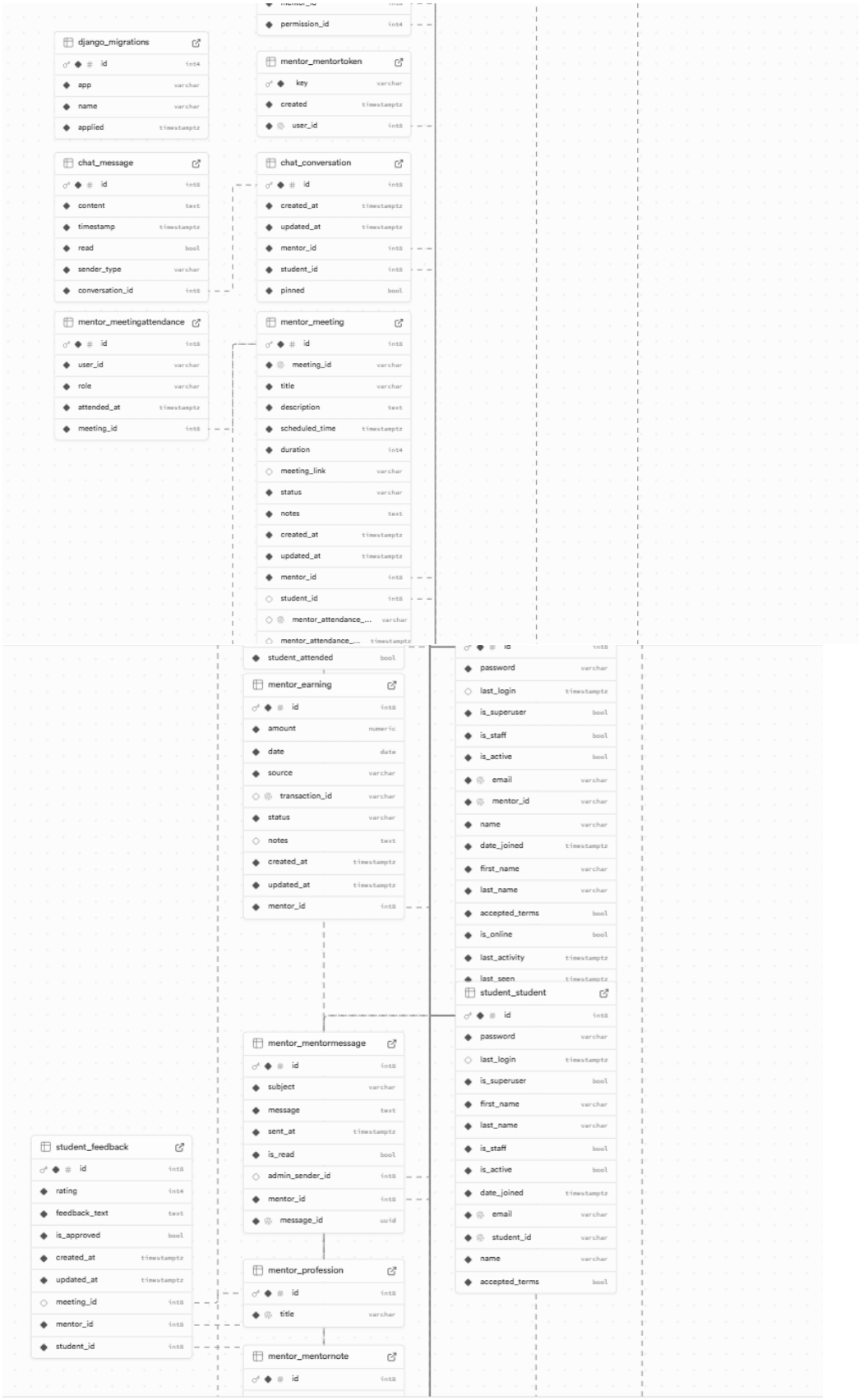
1. **python manage.py makemigrations**: This command is run after any changes are made to the models.py files. Django inspects the models and automatically generates a new migration file—a Python script that acts as a blueprint for the required database changes.
2. **python manage.py migrate**: This command applies all unapplied migrations. It reads the migration files and executes the necessary SQL commands to alter the tables in the Supabase database, bringing it in sync with the Django models. This systematic process is essential for reliable schema updates and team collaboration.

Integration with Supabase Storage

For handling file uploads like user profile pictures or resumes, **Supabase Storage** was used. This provides a simple, S3-compatible, and secure object storage solution. The workflow was implemented as follows:

1. A user uploads a file through the React frontend.
2. The file is sent to a dedicated Django API endpoint.
3. The Django backend, using a library like django-storages, processes the file and uploads it to a designated "bucket" in Supabase Storage.
4. Upon successful upload, Supabase returns a public URL for the file.
5. This URL is then saved in the appropriate field (e.g., profile_picture_url) in the user's profile model in the database.




permission_id int4
django_migrations
id int4
app varchar
name varchar
applied timestamptz
mentor_mentortoken
key varchar
created timestamptz
user_id int8
chat_message
id int8
content text
timestamp timestamptz
read bool
sender_type varchar
conversation_id int8
chat_conversation
id int8
created_at timestamptz
updated_at timestamptz
mentor_id int8
student_id int8
pinned bool
mentor_meetingattendance
id int8
user_id varchar
role varchar
attended_at timestamptz
meeting_id int8
mentor_meeting
id int8
meeting_id varchar
title varchar
description text
scheduled_time timestamptz
duration int4
meeting_link varchar
status varchar
notes text
created_at timestamptz
updated_at timestamptz
mentor_id int8
student_id int8
mentor_attendance_... varchar
mentor_attendance_... timestamptz
student_attended bool
mentor_earning
id int8
amount numeric
date date
source varchar
transaction_id varchar
status varchar
notes text
created_at timestamptz
updated_at timestamptz
mentor_id int8
password varchar
last_login timestamptz
is_superuser bool
is_staff bool
is_active bool
email varchar
mentor_id varchar
name varchar
date_joined timestamptz
first_name varchar
last_name varchar
accepted_terms bool
is_online bool
last_activity timestamptz
last_seen timestamptz
student_student
id int8
password varchar
last_login timestamptz
is_superuser bool
first_name varchar
last_name varchar
is_staff bool
is_active bool
date_joined timestamptz
email varchar
student_id varchar
name varchar
accepted_terms bool
mentor_mentormessage
id int8
subject varchar
message text
sent_at timestamptz
is_read bool
admin_sender_id int8
mentor_id int8
message_id uuid
student_feedback
id int8
rating int4
feedback_text text
is_approved bool
created_at timestamptz
updated_at timestamptz
meeting_id int8
mentor_id int8
student_id int8
mentor_profession
id int8
title varchar
mentor_mentornote
id int8

4.2 Creating the Logic: API Views and Serializers

With the database in place, the next step was to build the logic that allows the frontend to communicate with it. This was achieved by developing a RESTful API using the Django REST Framework (DRF).

- Serializers: You can think of serializers as "translators." They take the complex data from our database (in Python object form) and translate it into the simple JSON format that is easy for the frontend to read and display. They also work in reverse, validating and translating incoming user data back into objects that can be saved to the database.
- Views: The Views (or "ViewSets" in DRF) are the workers of the API. Each View is responsible for defining the logic for specific actions. For example, the PostViewSet handles all operations related to community posts: listing all posts, creating a new post, retrieving a single post, updating it, and deleting it.

4.3 Securing the Gates: User Authentication and Permissions

A robust security framework was implemented to protect user data and ensure the integrity of the platform.

Multi-faceted Authentication As detailed previously, AspireX supports both traditional email/password login via JWTs and social login via Google OAuth2. The JWT flow uses short-lived **access tokens** for API requests and long-lived **refresh tokens** to allow users to stay logged in seamlessly and securely. The Google OAuth2 flow provides a frictionless and trusted onboarding experience.

Role-Based Access Control (RBAC) Beyond simply authenticating a user, the system must control what they are allowed to do. This was achieved through a granular permission system.

- **Built-in Permissions**: DRF's default permissions, like IsAuthenticated, were used to protect endpoints that require a user to be logged in.
- **Custom Permissions**: For more complex rules, custom permission classes were written. For example, a custom permission ensures that a request to view the /api/earnings/ endpoint is only granted if the authenticated user has the "mentor" role. Another custom permission verifies that a user attempting to update a booking is the actual mentor assigned to that booking, preventing users from manipulating data that doesn't belong to them.

Proactive Security Measures In addition to access control, several other security best practices were implemented:

- **Environment Variable Management**: All sensitive information, such as the Supabase database URL, secret keys, and Google API credentials, were stored as environment variables and never hard-coded into the source code. This prevents sensitive data from being exposed in the GitHub repository.
- **Data Sanitization**: The Django ORM and DRF serializers automatically sanitize inputs to protect against common vulnerabilities like SQL injection and Cross-Site Scripting (XSS), ensuring that data stored and displayed is safe.

4.4 The Logic Layer: API Design and Implementation

The communication between the frontend and backend is handled by a comprehensive RESTful API built with the **Django REST Framework (DRF)**.

API Design Philosophy The API was designed with a focus on being predictable, consistent, and easy to use. Resources are exposed via logical URLs (e.g., /api/mentors/, /api/bookings/), and standard HTTP methods are used for all operations (GET for retrieving, POST for creating, PUT for updating, DELETE for removing).

Detailed API Endpoint Specification The following table outlines some of the key API endpoints that power AspireX:

Category	Endpoint	Method	Description	Authentication
Core Platform	/	GET	AspireX World landing page	None
Core Platform	/admin/	GET	Django admin interface	Admin only
Core Platform	/api/test-cors/	GET	CORS test endpoint	None
Core Platform	/api/platform-stats/	GET	Platform statistics	None
Core Platform	/api/newsletter/	POST	Newsletter subscription	None
Core Platform	/api/site-status/	GET	Site maintenance status	None
Category	Endpoint	Method	Description	Authentication
Auth	/api/auth/unified-login/	POST	Unified login for all users	None
Auth	/api/auth/unified-register/	POST	Unified registration	None

Category	Endpoint	Method	Description	Authentication
Auth	/api/auth/google/auth/	GET	Google OAuth initiation	None
Auth	/api/auth/google/callback/	GET	Google OAuth callback	None
Auth	/api/auth/google/verify/	POST	Google token verification	None
Category	Endpoint	Method	Description	Authentication
Student	/api/student/register/	POST	Student registration	None
Student	/api/student/login/	POST	Student login	None
Student	/api/student/profile/	GET	Get student profile	Student Token
Student	/api/student/profile/update/	PUT	Update student profile	Student Token
Student	/api/student/profile/file/	POST	Upload student CV/file	Student Token
Student	/api/student/booking/	POST	Create booking	Student Token
Student	/api/student/bookings/	GET	List student bookings	Student Token
Student	/api/student/public/	GET	Public student list	None
Student	/api/student/public/<id>/	GET	Public student details	None
Student	/api/student/notes/	GET/POST	Student notes CRUD	Student Token
Student	/api/student/notes/<id>/	DELETE	Delete student note	Student Token
Student	/api/student/messages/	GET	List student messages	Student Token
Student	/api/student/messages/<i	GET	Get message	Student Token

Category	Endpoint	Method	Description	Authentication
	d>/		details	
Student	/api/student/messages/<id>/read/	POST	Mark message as read	Student Token
Student	/api/student/feedback/	POST	Create feedback	Student Token
Student	/api/student/feedback/list/	GET	List feedback	Student Token
Student	/api/student/feedback/<id>/	GET	Get feedback details	Student Token
Student	/api/student/feedback/mentor/<id>/	GET	Get mentor feedback	Student Token
Student	/api/student/feedback/check/<id>/	GET	Check if feedback exists	Student Token
Mentor	/api/mentor/register/	POST	Mentor registration	None
Mentor	/api/mentor/login/	POST	Mentor login	None
Mentor	/api/mentor/profile/	GET	Get mentor profile	Mentor Token
Mentor	/api/mentor/profile/update/	PUT	Update mentor profile	Mentor Token
Mentor	/api/mentor/profile/cv/	POST	Upload mentor CV	Mentor Token
Mentor	/api/mentor/online-status/	POST	Update online status	Mentor Token
Mentor	/api/mentor/public/	GET	Public mentor list	None
Mentor	/api/mentor/public/<id>/	GET	Public mentor details	None
Mentor	/api/mentor/feature/	GET	Featured mentors	None
Mentor	/api/mentor/filter/	GET	Filtered mentor list	None
Mentor	/api/mentor/notes/	GET/POST	Mentor notes CRU	Mentor Token

Category	Endpoint	Method	Description	Authentication
			D	
Mentor	/api/mentor/notes/<id>/	DELETE	Delete mentor not e	Mentor Token
Mentor	/api/mentor/earnings/	GET	Mentor earnings	Mentor Token
Mentor	/api/mentor/withdrawals /	GET	Mentor withdraw als	Mentor Token
Mentor	/api/mentor/earnings/exp ort/	GET	Export earnings	Mentor Token
Mentor	/api/mentor/feedback/	GET	List mentor feedback	Mentor Token
Mentor	/api/mentor/feedback/<i d>/	GET	Get feedback details	Mentor Token
Mentor	/api/mentor/feedback/sta ts/	GET	Feedback statistics	Mentor Token
Mentor	/api/mentor/meeting/att endance/	POST	Record meeting attendan ce	Mentor/Student Token
Mentor	/api/mentor/meeting/<id >/reschedule/	POST	Reschedule meeting	Mentor Token
Mentor	/api/mentor/create-order/	POST	Create payment order	Mentor Token
Mentor	/api/mentor/verify-payment/	POST	Verify payment	Mentor Token
Mentor	/api/mentor/razorpay-webhook/	POST	Razorpay webhook	None
Chat	/api/chat/conversations/	GET/POST	List/create conver sations	User Token
Chat	/api/chat/conversations/< id>/messages/	GET	Get conversation messages	User Token

Category	Endpoint	Method	Description	Authentication
Chat	/api/chat/conversations/<id>/send/	POST	Send message	User Token
Chat	/api/chat/conversations/<id>/pin/	POST	Pin conversation	User Token
Chat	/api/chat/conversations/<id>/unpin/	POST	Unpin conversation	User Token
Chat	/api/chat/get-user-info/	GET	Get user information	User Token
Chat	/api/chat/contact/	POST	Send contact message	None
Chat	/api/chat/customer-service/	GET/POST	Customer service messages	User Token
Chat	/api/chat/customer-service/admin/	GET	Admin customer service list	Admin Token
Chat	/api/chat/customer-service/reply/	POST	Customer service reply	User Token
Chat	/api/chat/notifications/	GET	User notifications	User Token
Community	/api/community/posts/	GET/POST	List/create community posts	User Token
Community	/api/community/my-posts/	GET	Get user's posts	User Token
Community	/api/community/like/	POST	Like/unlike post	User Token
Community	/api/community/comment/	POST	Create comment	User Token
Community	/api/community/comments/<id>/	GET	Get post comments	User Token
Community	/api/community/delete-post/<id>/	DELETE	Delete post	User Token

Category	Endpoint	Method	Description	Authentication
Community	/api/community/posts/<id>/view/	POST	Increment post view	User Token

Serializers: The Data Translators and Validators Serializers are a core component of DRF. They handle the critical task of converting complex data types, like Django model instances, into JSON for API responses. Equally important, they handle validation of incoming data. When a student submits a booking request, for instance, the BookingSerializer ensures that all required fields are present and that the data types are correct before any database operations are performed.

Chapter 5: FRONTEND IMPLEMENTATION AND USER EXPERIENCE

Introduction

While the backend acts as the powerful engine and solid foundation of AspireX, the frontend is the meticulously crafted vehicle through which users experience that power. It's the sleek design, the intuitive dashboard, and the responsive controls that make the application not just functional, but genuinely useful and enjoyable. A powerful engine is useless without a steering wheel, and this chapter details the construction of that "steering wheel." This process of translation, from raw data and logic into a coherent visual experience, is where a project's success is often determined.
The development was driven by a dual philosophy: firstly, to build a dynamic and robust interface using React that communicates flawlessly with the backend; and secondly, to obsess over the **User Experience (UX)**. This meant that every element was intentionally designed to reduce cognitive load, build user trust, and ensure that every interaction is intuitive, engaging, and purposeful.

5.1 Building with Components: The React Ecosystem

The frontend was built using React, and its **component-based architecture** was the cornerstone of the development process. This paradigm promotes a clean, manageable, and scalable codebase by thinking of the UI not as large, monolithic pages, but as a collection of independent, reusable pieces.

- **A Building Block Approach**: The user interface was constructed from small, independent "Lego bricks" called components. For instance, a single MentorCard component was designed to accept a mentor's data (like their name, title, and profile picture) as "props" and display it in a consistent format. This promotes a crucial principle called **separation of concerns**; the MentorCard doesn't need to know *how* to fetch mentor data from the internet, only how to display it once the data is provided. This makes the system far less brittle and easier to debug. Complex pages, like the main user dashboard, are then created through **composition**, by assembling these smaller, independent components (Navbar, Sidebar, UpcomingSessionsPanel, NotificationsFeed) into a cohesive whole. This makes the development of even the most complex UIs a manageable task.

- **Managing Dynamic Data and Interactivity**: To breathe life into the static components, React's built-in Hooks were used to manage state and side effects. React enables a **declarative UI**, meaning that instead of manually writing instructions to change the screen (an imperative approach), we simply declare what the screen *should* look like for any given state. React then handles the complex task of efficiently updating the user's view. For example, the search bar's current text is held in state. The useEffect hook is configured with a "dependency array" to watch this specific piece of state, triggering an API call to the backend *only* when the search text actually changes. This prevents redundant network requests and ensures the application remains performant and

responsive.

- **Seamless Navigation and Routing**: To deliver a modern, fast user experience, AspireX was built as a **Single-Page Application (SPA)**. The React Router library was implemented to handle all navigation. This approach avoids the jarring "white flash" of a traditional full-page reload, creating a continuous and immersive experience akin to a native desktop application. This also allows for powerful patterns like **protected routes**. The routing system was configured to check a user's authentication status *before* attempting to render a protected page like the dashboard. If the user is not logged in, they are seamlessly and automatically redirected to the login screen, ensuring a secure and logical user flow.

5.2 Connecting to the Engine: Interacting with the Backend API

As the application grew, managing state that needed to be shared across many components (like the authenticated user's information) became a key challenge.

State Management Strategy: The Context API

For a project of this scale, React's built-in **Context API** was chosen as the global state management solution over more complex libraries like Redux. It offered a lightweight and efficient way to share global data without adding external dependencies. An AuthContext was created to provide all components with access to the current user's data and authentication status. This solved the problem of "prop drilling" (passing data down through many layers of components) and created a single source of truth for user information.

Connecting to the Engine: Asynchronous API Calls with Axios

The **Axios** library was used to handle all asynchronous communication with the backend API. A key aspect of the implementation was designing the application to "fail gracefully." For example, when fetching data, a .catch() block was implemented to handle network failures. If the backend was unreachable, instead of the application crashing, a friendly, human-readable error message would be displayed to the user, maintaining a professional and stable feel.

Handling Secure Authentication on the Frontend

The frontend plays a vital role in the authentication flow using stateless JWTs. It manages the token's lifecycle: upon login, the token is stored securely in the browser's localStorage; it is then automatically attached to the header of every subsequent protected request; and, crucially, it is deleted upon logout to instantly terminate the session.

5.3 Designing for Humans: User Experience (UX) and Interface (UI)

A functional application only becomes successful when it's designed with the end-user in mind. The design of AspireX was guided by established UX principles.

Visual Hierarchy and Consistency The design focused on creating a clear **visual hierarchy**, using size, color, and placement to guide the user's eye towards the most important actions. **Consistency** was also paramount; buttons, links, and forms were designed to look and behave the same way across the entire platform. This reduces the user's cognitive load and makes the application feel intuitive and easy to learn.

Applying Nielsen's Usability Heuristics Several of Jakob Nielsen's famous usability heuristics were consciously applied:

- **Visibility of system status**: This was implemented through loading states (like "skeleton screens" that show a placeholder layout of the content about to load) and success notifications ("toast" messages that confirm an action was completed). The system always keeps the user informed about what is happening.
- **User control and freedom**: Users can easily undo actions, such as canceling a booking request before it's confirmed, providing an "emergency exit" to leave an unwanted state.
- **Error prevention**: The use of proactive, inline form validation prevents users from making errors in the first place, rather than waiting until they submit a form to report a problem.
-

A Mobile-First, Responsive Design The entire application was built with a **mobile-first** philosophy. The design was perfected for the smallest screen size first, which forces a focus on the most essential content and features. This approach guarantees that the layout flawlessly adapts via modern CSS techniques to provide an optimal and consistent experience on desktops, tablets, and mobile phones.

Chapter 6: CORE PLATFORM FEATURES AND FUNCTIONALITY

Introduction

A successful web platform is defined by the quality and usability of its features. While the previous chapters detailed the technical architecture and implementation, this chapter focuses on the user-centric functionalities that form the heart of the AspireX experience. We will take a detailed walkthrough of the key modules that enable the core user journey: the process of discovering and booking mentor services, the communication system that connects users, the feedback mechanism that builds trust, and the earnings module that empowers mentors. This exploration will cover both the functionality from a user's perspective and the underlying implementation logic.

6.1 The Mentor Services and Booking System

The primary function of AspireX is to facilitate the connection between students and mentors through bookable services. This module was designed to be intuitive for both parties.

Mentor's Perspective: Creating and Managing Services For mentors, the platform provides a simple yet powerful interface within their dashboard to create, update, and manage the services they offer.

- **Interface and Functionality**: Mentors can define several key attributes for each service, including a clear **Service Title** (e.g., "CV Check," "Mock Technical Interview"), a detailed **Description** of what the service includes, the **Duration** of the session in minutes, and the **Price**.
- **Implementation**: This functionality is powered by a dedicated Service model in the Django backend, linked to the MentorProfile with a foreign key. When a mentor creates a service, the frontend form sends a POST request to an /api/services/ endpoint. The backend validates this data and creates a new entry in the database.

Student's Perspective: Booking a Session For students, the process of finding and booking a service is designed to be seamless.

- **User Flow**: After finding a suitable mentor, the student can view all available services on the mentor's profile page. Clicking the "Book Session" button initiates the booking process, typically through a modal window. The student then selects their desired service from a dropdown menu, chooses an available time slot from the mentor's calendar, and agrees to the terms.
- **Implementation**: This user journey is managed by several React components. The ServiceList component fetches and displays the services for a given mentor. The BookingModal component contains the form state for the time slot, selected service, and any additional notes. Upon submission, a POST request is sent to an /api/bookings/ endpoint. The backend logic then checks for calendar conflicts, reserves the time slot, and prepares for the payment confirmation step. The QR code payment system shown in the screenshots acts as the final confirmation step, where the student provides a transaction ID to finalize the booking.

6.2 The User-to-User Chat System

Effective communication is crucial for building a strong mentor-mentee relationship. The AspireX chat system provides a dedicated and private channel for users to interact.

- **Architecture and Design**: The system is designed as an asynchronous messaging platform. It allows for one-on-one conversations between a student and a mentor, with messages stored persistently in the database. This ensures users can access their chat history at any time.
- **Backend Implementation**: The chat functionality is supported by two primary Django models: a Conversation model to represent a chat thread between two users, and a ChatMessage model to store individual messages. The ChatMessage model includes fields for the message content, a timestamp, and foreign keys linking to the Conversation and the sender (user). A set of API endpoints, such as /api/conversations/ and /api/conversations/<id>/messages/, allows the frontend to fetch chat lists and message histories, and to post new messages.
- **Frontend Implementation**: The chat interface in React is broken into several components. A ChatList component on the left sidebar displays all active conversations for the logged-in user. The main ChatWindow component displays the messages for the selected conversation, and a MessageInput component at the bottom handles the typing and submission of new messages. State is managed within the parent component to automatically re-fetch messages after a new one is sent, creating the appearance of a real-time conversation.

6.3 Mentor Earnings and Transaction History

To empower mentors and provide transparency, a dedicated earnings page was created within the mentor dashboard. This allows them to track income generated from their mentorship sessions.

- **Backend Logic and Data Model**: An Earning model was created in the database to log all financial transactions. Each time a student's session is successfully completed and paid for, a new entry is created in this table. The model contains fields for the amount, the source_session (a foreign key to the booking), the transaction status (e.g., "cleared," "pending payout"), and a timestamp.
- **Frontend Interface**: In the mentor's dashboard, an "Earnings" page presents this data in a clear and understandable format. The interface typically includes:
    - **Summary Cards**: Displaying key figures like "Total Earnings," "This Month's Earnings," and "Amount Pending Payout."
    - **Transaction Table**: A detailed, paginated table listing every transaction with columns for the date, student name, service provided, and the amount earned. This data is fetched from a secure /api/mentor/earnings/ endpoint that only returns data for the authenticated mentor.

6.4 The Student Feedback and Review System

Trust and credibility are the cornerstones of a successful mentorship platform. The feedback system was implemented to allow students to share their experiences and help the community make informed

decisions.

- **User Flow for Feedback**: After a mentorship session is marked as "completed," the student receives a prompt or notification to leave a review for the mentor. The feedback form is simple, consisting of a quantitative **star rating** (typically 1 to 5 stars) and a qualitative text area for a **written comment**.
- **Implementation Details**:
    - **Backend**: A Review model was created, linked via foreign keys to the Student (the author), the Mentor (the subject), and the specific Session being reviewed. This ensures the integrity of the review. API endpoints were created to allow for the submission of new reviews and to fetch all reviews for a given mentor. The backend also calculates the mentor's average star rating automatically.
    - **Frontend**: A ReviewModal component is used to collect the student's feedback. On the mentor's public profile page, a ReviewList component fetches and displays all submitted reviews, along with the calculated average rating prominently displayed near the mentor's name. This public feedback is crucial for building mentor credibility and helping students choose the right guide for their journey.

Chapter 7: SYSTEM TESTING, DEPLOYMENT, AND CHALLENGES

Introduction

The journey of creating a software application doesn't end when the last line of code is written. The final, critical phases of the lifecycle involve ensuring the application is robust and error-free (testing), making it accessible to the world (deployment), and reflecting on the valuable lessons learned from the obstacles overcome along the way. This chapter covers this "quality assurance" and "launch" phase for AspireX. It details the rigorous testing strategies employed, the process of deploying the frontend and backend to live servers, and the key technical challenges that provided significant learning opportunities.

7.1 Ensuring Quality: System Testing Strategies

To guarantee a reliable and smooth user experience, AspireX was subjected to a multi-layered testing strategy. This approach was designed to catch issues at every level, from the smallest unit of code to the complete user journey.

- **Unit Testing**: At the most granular level, unit tests were conducted to verify that individual pieces of the application worked as expected in isolation. For the Django backend, this involved writing tests to confirm that model methods and API utility functions behaved correctly. For the React frontend, tools like the React Testing Library were used to test individual components, ensuring that a Button component rendered correctly or that a form input properly updated its state.
- **Integration Testing**: While unit tests confirm that the parts work, integration tests confirm that they work *together*. Tests were designed to simulate interactions between the frontend and backend. A key integration test, for example, involved the user registration flow. This test would verify that when a user fills out the signup form on the React frontend, the data is correctly sent to the backend API, a new user is successfully created in the Supabase database, and the correct success response and authentication token are returned to the frontend.
- **Manual End-to-End Testing**: Finally, manual testing was performed to simulate the real-world user experience from start to finish. This involved a comprehensive walkthrough of the entire application: creating both student and mentor accounts, logging in with email and Google, searching for mentors, posting on the community page, and updating a user profile. This hands-on approach was invaluable for catching bugs in the user flow and verifying the responsive design on different browsers and devices.

7.2 Going Live: Deployment

Deployment is the process of taking the application from a local development environment and making it publicly accessible on the internet. A modern, cloud-based approach was used for both the frontend and backend.

- **Frontend Deployment on Vercel**: The React frontend was deployed using **Vercel**, a platform specifically optimized for modern web applications. The process was streamlined through Vercel's direct integration with GitHub. After connecting the project's GitHub repository, Vercel's Continuous Deployment (CI/CD) pipeline was activated. This powerful feature automatically triggered a new build and deployment every time changes were pushed to the main branch, making the process of releasing updates incredibly efficient and hassle-free.
- **Backend Deployment**: The Django backend was deployed to a cloud hosting platform like Render or Heroku. This involved creating a production-ready environment, which included configuring environment variables to securely manage sensitive information like database credentials and secret keys. The Django application was then connected to the live Supabase PostgreSQL database, ensuring that the deployed frontend could communicate with a stable and scalable backend server.

7.3 Overcoming Hurdles: Challenges and Learning Outcomes

Every development process has its challenges, and AspireX was no exception. These hurdles were not setbacks, but rather crucial learning experiences that deepened my understanding of web development.

- **Challenge 1: Cross-Origin Resource Sharing (CORS)**: A common but tricky issue arose when the deployed frontend on Vercel was blocked by the browser from communicating with the backend API on a different domain. This security feature was overcome by configuring the django-cors-headers library on the backend. This involved creating a "whitelist" of approved domains, explicitly instructing the server to accept requests from the Vercel app, which was a vital lesson in web security.
- **Challenge 2: Complex State Management**: As more features were added, managing the application's "state" (like the current user's authentication status) across different components became complicated. This was solved by implementing React's **Context API**. A global "AuthContext" was created to hold the user's data and authentication token, making this information easily accessible to any component in the application without having to pass it down manually through many layers.
- **Challenge 3: Google OAuth2 Integration**: The social login feature, while great for users, presented a complex technical challenge. The multi-step token exchange between the React frontend, Google's authentication servers, and our Django backend required precise configuration of credentials and redirect URIs. Successfully implementing this flow was a significant achievement and provided deep insights into working with third-party authentication services.

Chapter 8: Project Management and Methodology

Introduction

The successful completion of the AspireX project was the result not only of technical implementation but also of a structured project management approach. This chapter details the development methodology, the tools used for version control and collaboration, and the overall project timeline, providing insight into how the project was managed from conception to deployment.

8.1 Development Methodology: An Agile Approach

For this project, an **Agile-like, iterative methodology** was adopted. While formal Agile ceremonies like daily stand-ups were not necessary for a solo project, the core principles of Agile—flexibility, iterative development, and continuous feedback—were paramount. This approach was chosen over a rigid Waterfall model because it allowed for adaptability as new challenges and ideas emerged during development.

The project was broken down into logical, feature-based sprints or phases:

- **Phase 1: Foundation and Authentication (Weeks 1-2)**: Focused on setting up the Django and React projects, designing the database schema, and implementing the core user registration and login functionality (both JWT and Google OAuth2).
- **Phase 2: Core Mentor and Student Features (Weeks 3-4)**: Focused on building the mentor profile and service management modules, the student booking system, and the underlying API endpoints.
- **Phase 3: Ancillary Features and UI Polish (Week 5)**: Focused on developing the user-to-user chat system, the community page, and the mentor earnings dashboard. Significant time was also dedicated to refining the user interface and experience.
- **Phase 4: Testing and Deployment (Week 6)**: The final phase was dedicated to comprehensive testing (unit, integration, and manual), bug fixing, and deploying the application to production environments.

8.2 Version Control and Source Code Management

Git and **GitHub** were used for version control throughout the project's lifecycle. A structured branching strategy was employed to maintain a clean and manageable codebase:

- **main branch**: This branch was strictly reserved for stable, deployed code. It was only updated by merging in changes from the develop branch.
- **develop branch**: This served as the primary integration branch where all completed features were merged.
- **feature branches**: For every new piece of functionality (e.g., feature/chat-system, feature/payment-verification), a new branch was created from develop. This isolated new work, allowing for development and testing without destabilizing the main codebase. Once a feature was complete, it was merged back into develop.

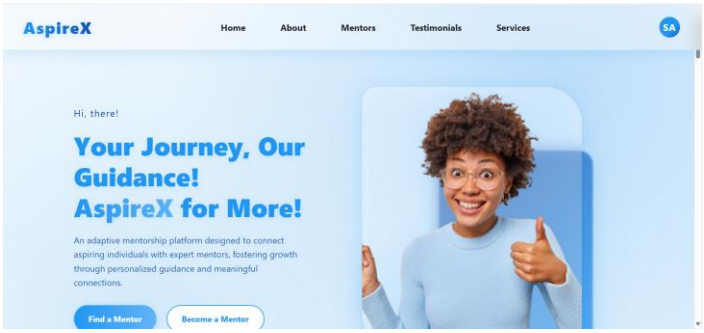
This workflow, combined with the practice of writing clear and descriptive commit messages, ensured a complete and understandable history of the project's development.

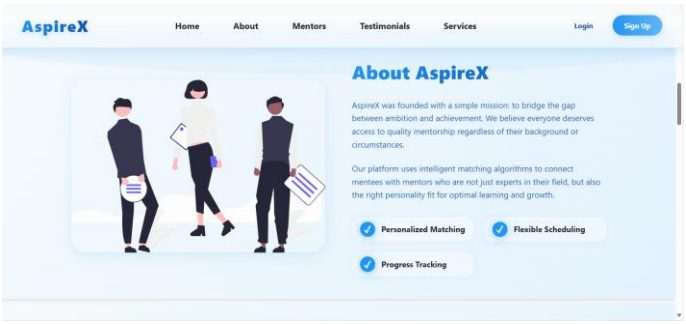
8.3 Development Tools and Environment

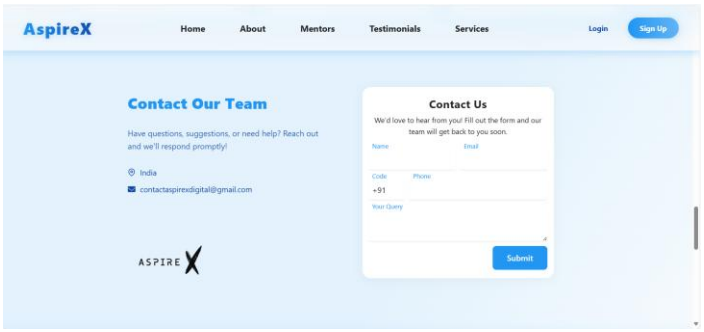
A suite of modern development tools was used to facilitate an efficient workflow:

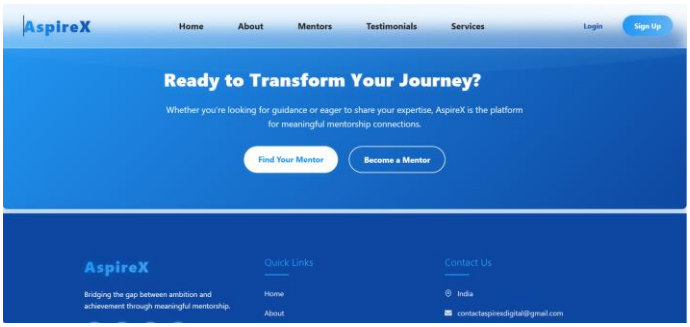
- **Code Editor**: **Visual Studio Code** was the primary editor, customized with extensions for Python, Django, JavaScript, and React to enhance productivity.
- **Version Control**: **Git** for local version control and **GitHub** for remote repository hosting and collaboration.
- **API Testing**: **Postman** was used extensively to test the Django REST Framework API endpoints independently from the frontend, ensuring the backend logic was correct and robust before UI integration.
- **Database Management**: The **Supabase Dashboard** provided a user-friendly interface for viewing and managing the PostgreSQL database tables, checking data, and debugging schema issues.
- **Package Management**: **Pip** and requirements.txt were used for managing Python packages in the backend, while **npm** and package.json were used for managing JavaScript packages in the frontend.

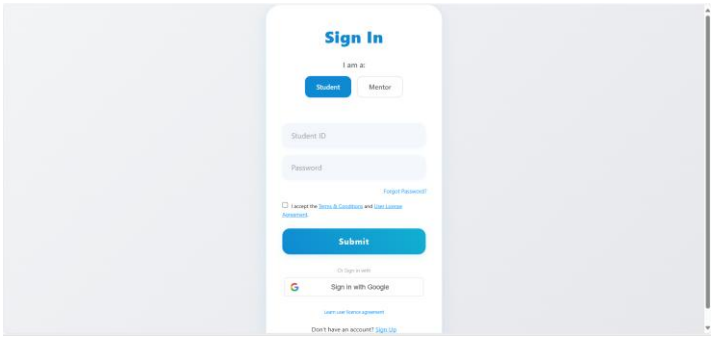
Screenshots of Website

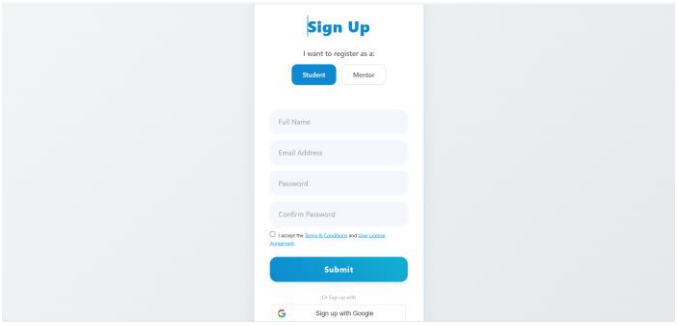


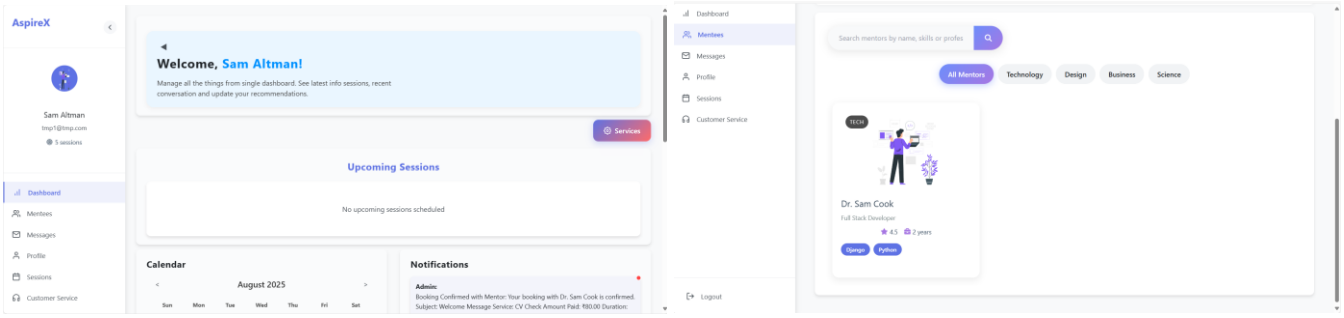


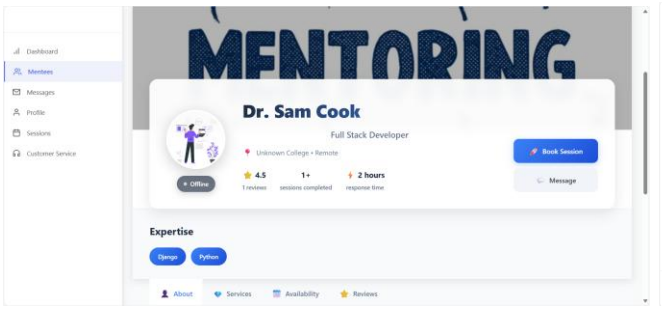


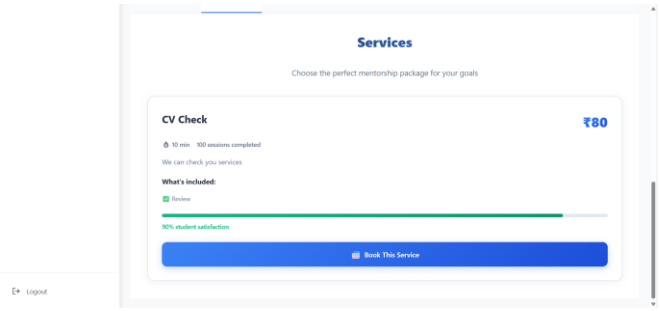


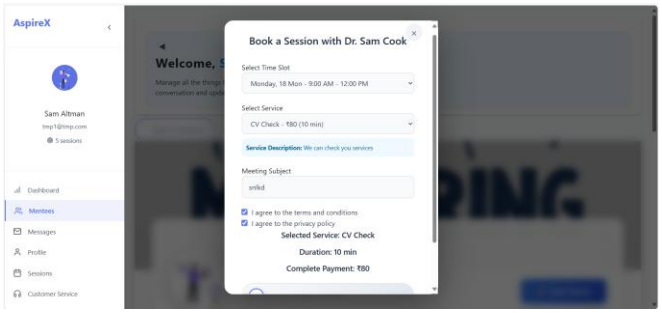


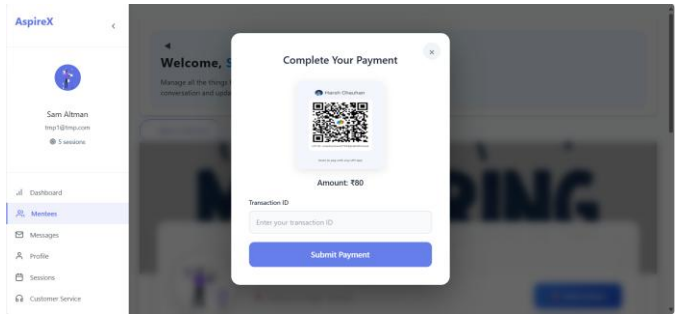


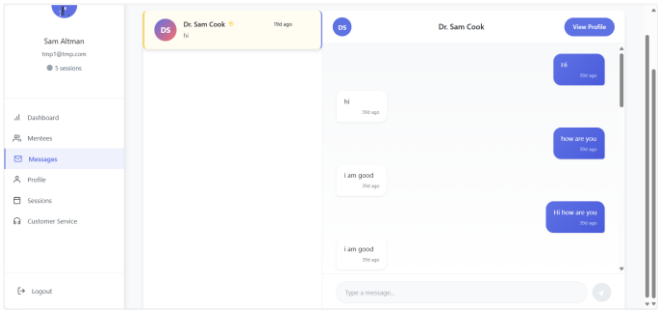


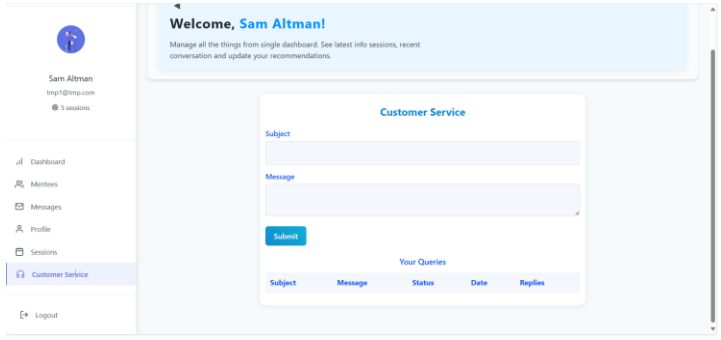












For Further Exploration (Check out) : [AspireX]
The source code for AspireX is available at: [https://github.com/harshchauhan01/AspireX]

Final Chapter: CONCLUSION

The journey from academic learning to a professional career is a critical transition, often marked by a significant gap between theoretical knowledge and real-world application. The AspireX project was conceived from this understanding, with the ambitious objective of bridging this gap through a dedicated digital platform connecting students with experienced mentors. This project has now reached its successful culmination, evolving from an initial concept into a fully functional, deployed web application. This final chapter serves to summarize the project's achievements against its original goals, reflect on the profound learning experiences encountered, and outline a clear vision for the platform's future potential.

Summary of Achievements and Impact

The project successfully delivered a robust, full-stack application, demonstrating a comprehensive end-to-end development process. The core objectives were not just met, but realized in a way that provides tangible value:

- **A Complete Full-Stack Platform**: The creation of a complete application using React, Django, and Supabase represents the project's core technical achievement. This provided full control over the entire user journey, from the moment data is entered into the frontend to its secure storage in the database, ensuring a cohesive and seamless experience.
- **Distinct and Functional User Roles**: By implementing separate, tailored dashboards and functionalities for students and mentors, the platform offers a highly intuitive and purposeful user experience. This fundamental design choice ensures that each user group interacts with the platform in the most efficient way possible, minimizing confusion and maximizing utility.
- **Secure and Flexible Authentication**: The integration of both traditional email/password and modern Google OAuth2 authentication was a deliberate choice to prioritize user convenience and security. This flexibility removes barriers to entry and allows users to sign in with a method they trust, which is crucial for building a strong user base.
- **An Engaging and Supportive Community**: The community feature is more than just an add-on; it is the heart of the platform's ecosystem. It transforms AspireX from a simple transactional service into a supportive network where users can share triumphs, offer encouragement, and build a sense of belonging.

Personal and Technical Reflection

This project was a significant journey of personal and professional growth. It was an exercise in turning theory into practice, navigating the inevitable challenges that arise when building a complex application from scratch. The process of designing APIs, managing state in a large React application, and configuring a secure authentication flow provided invaluable hands-on experience. Overcoming hurdles like Cross-Origin Resource Sharing (CORS) and integrating third-party services like Google OAuth2 were not just technical problems to be solved; they were deep learning opportunities that taught resilience, systematic debugging, and the importance of thorough documentation. This project solidified my technical skills and, more importantly, cultivated a problem-solver's mindset essential for any career in technology.

Future Scope and Vision

While the current version of AspireX is a complete product, its architecture is a foundation primed for future innovation. The vision for AspireX is to evolve it into a comprehensive career development hub. Potential enhancements include:

- **Real-time Chat Integration**: To facilitate more immediate and dynamic conversations, an in-app messaging system using WebSockets would be a transformative addition, allowing for instant Q&A sessions and follow-ups.
- **Payment Gateway Integration**: To legitimize the services offered and create a self-sustaining ecosystem, integrating a secure payment gateway like Stripe is a logical next step. This would empower mentors to value their time and students to formally engage in premium services.
- **Rating and Review System**: To build trust and foster a merit-based community, a feedback system is crucial. Allowing students to rate and review their experiences would help others make informed decisions and reward high-quality mentors.
- **Advanced Matching Algorithm**: Moving beyond simple search, a "smart" matching algorithm could be developed. This system would analyze student profiles, learning goals, and career interests to proactively recommend the most compatible and effective mentors, adding a layer of personalized value.

In conclusion, the AspireX project is more than a mere technical exercise; it is a proof-of-concept for a modern, accessible model of student mentorship. It stands as a testament to how technology can be leveraged to solve meaningful problems, democratize access to valuable resources, and empower the next generation of professionals. The platform is now a solid foundation, ready to be expanded upon to create an even more significant and positive impact on the careers of students everywhere.

REFERENCES

1. React Official Documentation. Available at: https://react.dev/ (Accessed on 15th Aug 2025).
2. Django Official Documentation. Available at: https://docs.djangoproject.com/en/stable/ (Accessed on 15th Aug 2025).
3. Google Identity for Developers - Sign In With Google for Web. Available at: https://developers.google.com/identity/gsi/web/guides/overview (Accessed on 15th Aug 2025).
4. Supabase Official Documentation. Available at: https://supabase.com/docs (Accessed on 15th Aug 2025).
5. Django REST Framework Official Documentation. Available at: https://www.django-rest-framework.org/ (Accessed on 15th Aug 2025).
6. DRF Simple JWT Documentation. Available at: https://django-rest-framework-simplejwt.readthedocs.io/en/latest/ (Accessed on 15th Aug 2025).
7. React Router Official Documentation. Available at: https://reactrouter.com/en/main (Accessed on 15th Aug 2025).
8. Axios Library Documentation. Available at: https://axios-http.com/docs/intro (Accessed on 15th Aug 2025).
9. Vercel Official Documentation. Available at: https://vercel.com/docs (Accessed on 15th Aug 2025).
10. django-cors-headers Library Documentation. Available at: https://pypi.org/project/django-cors-headers/ (Accessed on 15th Aug 2025).