The Backend Development Process for SubscriptPro

Hello! This is a detailed, step-by-step documentation of the entire backend development process for our SubscriptPro project. It covers everything we did, from the first line of code to setting up the admin panel, so you'll have a complete understanding of the system's architecture and logic.

---

1. Project Foundation: Setup and Initial Configuration

The first step in any backend project is to set up the local environment. This process lays the groundwork for all our code.

What I Did:

I created the main project and our specific application to handle all the API logic:

1. I created the project's root folder and core files with **django-admin startproject subscriptpro_backend**.
2. I created our application, named api, with **python manage.py startapp api**. An "app" in Django is a modular component that handles a specific function within the project.
3. I added our new api app and the required rest_framework dependencies to the INSTALLED_APPS list in **subscriptpro_backend/settings.py**.
4. I also explicitly told Django to use our custom user model by adding **AUTH_USER_MODEL = 'api.User'** to our settings file, which was the key fix for the errors we encountered.
5. Finally, I configured Django REST Framework to use **Token Authentication** as our default security method.

---

2. The Database: Models & Migration Logic

This is where we defined the structure of our application's data.

What I Did:

I used Django's Object-Relational Mapper (ORM) to define our database tables. The ORM lets us write database schemas using Python code instead of raw SQL, making it more intuitive and less prone to error.

- **api/models.py**: This file contains the Python classes that represent our database tables.
    - I defined a custom **User** model by extending Django's AbstractUser class. I configured it to use the email as the primary login field instead of a separate username.
    - The **Subscription** model was defined to store details like the service name, cost, and start/end dates.
    - I used a **ForeignKey** field in the Subscription model to link each subscription back to a specific user. This is how the backend knows which subscriptions belong to whom.

Database Migration:

Once the models were defined, I ran the following commands to create the tables in our database:

1. **python manage.py makemigrations api**: This command scanned our models.py file and generated a migration file—a set of instructions that tells Django how to create or update the database schema.
2. **python manage.py migrate**: This command executed the migration instructions, building the actual tables in our database.

---

3. The API: Building the Endpoints

This is the code that handles all the requests from the frontend. It's the "brain" that processes data and communicates with the database.

What I Did:

1. **api/serializers.py**: A serializer is a translator. I created one for our User model and one for our Subscription model. Their job is to take a complex Django model object and convert it into a simple format that the frontend

can understand, which is **JSON**. It also handles the reverse process for data coming in from the frontend.

2. **api/views.py**: This file contains the core logic of our API. A "view" is a class that receives a web request from the frontend and returns a response.
    - I created the **RegisterView** to handle user signups. It creates a new user and gives them a login token immediately.
    - The **LoginView** was created to handle user logins and issue an authentication token.
    - The **SubscriptionListCreateView** handles two things: listing all subscriptions for the logged-in user and creating new ones.
    - The **SubscriptionDestroyView** was created specifically to handle the deletion of a subscription.
3. **subscriptpro_backend/urls.py & api/urls/**: These files handle the routing. They tell Django which view to use when a specific URL is requested. I organized the URLs into separate files for a cleaner structure: one for all authentication-related links (api/urls/auth.py) and one for all subscription links (api/urls/subscriptions.py).

---

4. Authentication and Security

Authentication is a crucial part of a secure backend. It ensures that only authorized users can access or modify their private data.

What I Did:

I implemented **Token Authentication** as our security method. Here's the process:

1. When a user signs up or logs in, the backend generates a unique, single-use authentication **token**.
2. The frontend receives this token and stores it.
3. For every subsequent request to a protected endpoint (like adding a subscription), the frontend includes this token in the request header.

4. Our backend views are configured to require this token. The backend validates the token to make sure the request is coming from a legitimate, logged-in user before processing it.

---

5. Managing Data with the Admin Site

A great part about Django is its built-in admin site. It provides an easy way to manage all your data without writing extra code.

What I Did:

1. I registered our User and Subscription models in the **api/admin.py** file. This tells Django to make them visible and manageable in the admin site.
2. I created an administrator account with **python manage.py createsuperuser**.
3. By running the server, I could then navigate to http://127.0.0.1:8000/admin/ to log in and manage all users and subscriptions directly from the browser. This is a very powerful tool for administrative tasks.