**1. Planning (Day 1)**

This step remains the same—decide on features, plan your UI, and set up version control with Git.

**2. Front-End Development (Day 2 - 10)**

**No changes here:**

* Continue with **React** for the front-end.
* The front-end development process is the same whether you're using Node.js or Django for the back-end.

**3. Back-End Development with Django (Day 11 - 20)**

**Learn:**

* Django framework (Python)
* Django REST framework (for building APIs)
* SQLite (default) or PostgreSQL (for production) as the database

**Steps:**

**Day 11-12: Set Up Django Project**

* Install Django on your system (pip install django).
* Create a new Django project (django-admin startproject project\_gemini).
* Set up the project structure with apps for user management and project generation.

**Day 13: Set Up Django REST Framework (DRF)**

* Install Django REST framework (pip install djangorestframework).
* Create a Django app for handling API routes (python manage.py startapp api).
* Set up serializers and views for user registration, login, and project data.

**Day 14-15: Implement User Authentication**

* Use **Django’s built-in authentication system** or **DRF's TokenAuthentication** to handle user login and registration.
* Set up routes (/register, /login) and token-based authentication using **JWT** or **DRF tokens**.

**Day 16-17: Set Up Database**

* Use Django's ORM to define your models for storing project data:
  + **User model** (can use Django's built-in User).
  + **Project model** (with fields like title, sections, and generated text).
* For development, use **SQLite**, and for production, consider switching to **PostgreSQL** or **MySQL**.

**Day 18-19: Build Project Generation API**

* Create a Django view to handle the project topic submission from the front-end.
* Integrate the **OpenAI API** (or any other text-generation service) to generate text for project sections.
* Use Django to process the topic and return the generated text as a response to the front-end.

**Day 20: Set Up Project Saving and Retrieval**

* Create API routes in Django for saving and retrieving projects (/projects/save, /projects/get).
* Use Django views to interact with the database and store/retrieve user-specific project data.

**4. Integrating Front-End with Django Back-End (Day 21 - 23)**

**Learn:**

* **Axios** for making HTTP requests in React to the Django back-end API.

**Steps:**

**Day 21-22: Connect Front-End to Django Back-End**

* Use Axios in your React app to send HTTP requests (e.g., POST to /api/generate-project).
* Handle responses from Django’s API in your React components and display the generated text.

**Day 23: User Authentication Flow**

* After login, store the authentication token (JWT or DRF token) in local storage.
* Use this token for authenticated API requests in React (e.g., saving projects).

**5. Final Touches & Testing (Day 24 - 27)**

**No major changes here:**

* Continue with styling, testing, and deployment steps.
* For deployment:
  + Deploy the Django back-end to **Heroku** or **PythonAnywhere**.
  + Deploy the React front-end to **Netlify** or **Vercel**.
  + Set up **CORS** (Cross-Origin Resource Sharing) in Django so that your React app can communicate with your Django API.

**Key Concepts to Learn (Specific to Django)**

* **Django:** General understanding of views, models, and templates.
* **Django REST Framework (DRF):** For building APIs in Django.
* **Django ORM:** To interact with your database.
* **Token-Based Authentication:** Using DRF or JWT tokens for user login.