Of course. Here is a complete, in-depth, and sequential project plan designed to guide AI agents in building the Engaze Resume Builder.

This document serves as the master blueprint, detailing every step from foundational setup to final deployment, with a core focus on delivering a minimalist, professional, and highly intuitive user experience inspired by platforms like FlowCV.

Project Blueprint: The Engaze Resume Builder

1. Project Vision & UI/UX Philosophy

- **Guiding Principle:** "Simplicity is the ultimate sophistication." Every design and interaction choice must serve to reduce complexity and cognitive load for the user. The interface should be clean, uncluttered, and predictable.
- **Core UX Tenets:**
    - **Frictionless Editing:** The user should feel like they are directly manipulating the final document. The "on-canvas" experience is paramount.
    - **Clarity and Focus:** At any given time, the UI should only present options relevant to the user's current task (e.g., show text formatting tools only when text is selected).
    - **Professional Aesthetics:** Utilize a muted color palette, generous whitespace, and highly legible typography. The default templates must look polished out-of-the-box.
    - **Responsive and Fast:** The application must be performant and provide a seamless experience on both desktop and mobile devices.

---

2. Milestone 0: Foundation & Environment Setup (ETA: 4 Hours)

This milestone establishes the project's core infrastructure.

Task 0.2: Backend Setup (Django)

1. Navigate to backend. Create a Python virtual environment.
2. Install dependencies: pip install django djangorestframework psycopg2-binary djangorestframework-simplejwt django-cors-headers.
3. Initialize the Django project: django-admin startproject config . and an app django-admin startapp api.
4. Configure settings.py: Add rest_framework, corsheaders, and api to INSTALLED_APPS. Set up CORS_ALLOWED_ORIGINS to accept requests from the frontend's development URL.
5. Configure PostgreSQL database connection in settings.py.

- **Task 0.3: Frontend Setup (React)**
    1. Navigate to frontend. Initialize a React project: npm create vite@latest . -- --template react-ts.

   2. Install dependencies: npm install axios react-router-dom @reduxjs/toolkit react-redux tailwindcss postcss autoprefixer react-beautiful-dnd @react-pdf/renderer.
   3. Initialize Tailwind CSS: npx tailwindcss init -p. Configure tailwind.config.js to scan component files.
- **Task 0.4: Proxy Server Setup (Node.js)**
   1. Navigate to proxy. Initialize a Node.js project: npm init -y.
   2. Install dependencies: npm install express http-proxy-middleware cors.
   3. Create server.js. Implement a basic Express server that uses http-proxy-middleware to forward all requests from /api to the Django backend running on http://127.0.0.1:8000.

---

3. Milestone 1: Backend Architecture - Core Models & Auth (ETA: 8 Hours)

This milestone builds the data backbone and secures the application.

- **Task 1.1: Database Schema Design (in backend/api/models.py)**
   - User: Use Django's built-in AbstractUser for extensibility.
   - Resume:
      - user: ForeignKey to User (on_delete=models.CASCADE).
      - title: CharField(max_length=255).
      - template_name: CharField(max_length=50, default='classic').
      - share_slug: UUIDField(unique=True, null=True, blank=True).
      - created_at, updated_at: DateTimeField with auto_now_add and auto_now.
   - Style:
      - resume: OneToOneField to Resume.
      - primary_color: CharField(max_length=7, default='#000000').
      - font_family: CharField(max_length=100, default='Inter').
      - font_size: IntegerField(default=10).
   - Section:
      - resume: ForeignKey to Resume.
      - type: CharField (e.g., 'experience', 'education', 'skills').
      - content: JSONField to store flexible data structures (e.g., {"title": "...", "company": "..."}).
      - order: PositiveIntegerField to control section sequence.
- **Task 1.2: API Authentication (Simple JWT)**

1. Configure Django REST Framework to use JWT for authentication.
2. Create authentication endpoints in urls.py:
    - POST /api/auth/register/
    - POST /api/auth/token/ (Login)
    - POST /api/auth/token/refresh/
3. Implement a UserSerializer for the registration view.

---

4. Milestone 2: Frontend - Auth Flow & User Dashboard (ETA: 12 Hours)

This milestone creates the user's entry point and home base.

- **Task 2.1: UI Design - Authentication & Dashboard**
    - **Login/Register Pages:** Center-aligned form on a clean background. Focus on clear typography and a single call-to-action button.
    - **Dashboard:** A minimalist navigation bar with a "Logout" button. The main area will be a grid of cards. Each card represents a resume, displaying its title and a thumbnail preview. A prominent "+" card or button will be used to create a new resume.
- **Task 2.2: Redux State Setup**
    - store.js: Configure the Redux store.
    - authSlice.js: Manages user, accessToken, isAuthenticated state. Includes reducers for login, logout, and registration.
    - dashboardSlice.js: Manages resumes (an array of resume objects) and loading states.
- **Task 2.3: Component Implementation**
    - pages/LoginPage.jsx, pages/RegisterPage.jsx: Build the forms.
    - pages/DashboardPage.jsx: Fetches and displays user resumes.
    - components/ResumeCard.jsx: The individual card for each resume on the dashboard.
    - components/Navbar.jsx: Application header.
- **Task 2.4: Routing & API Integration**
    - Set up react-router-dom for public (/login) and private (/dashboard) routes.
    - Create an apiService.js (using Axios) to handle all backend communication via the proxy.
    - Connect the login/register forms and the dashboard to their respective backend endpoints.

---

5. Milestone 3: The Resume Editor - Core Workspace (ETA: 16 Hours)

This milestone builds the heart of the application.

- **Task 3.1: Backend API Extension (CRUD Operations)**
    1. Create serializers for Resume, Section, and Style models.
    2. Implement ModelViewSets for resumes and sections, ensuring they are protected and only accessible by the owner.
    3. Define URL patterns in urls.py:
        - /api/resumes/: GET (list), POST (create).
        - /api/resumes/{id}/: GET (retrieve), PUT/PATCH (update), DELETE.
        - /api/resumes/{id}/sections/: GET, POST.
        - /api/sections/{id}/: PUT/PATCH, DELETE.
- **Task 3.2: UI Design - The Editor**
    - A three-column layout:
        1. **Left Sidebar:** A list of available section types ("Experience," "Education," etc.) that can be dragged or clicked to add to the resume.
        2. **Center Canvas:** A live, interactive preview of the resume (ResumeCanvas).
        3. **Right Sidebar:** Context-aware controls. When editing the document, it shows "Design" (colors, fonts) and "Template" options.
- **Task 3.3: Frontend State & Component Implementation**
    - resumeSlice.js: Manages the entire state of the *active* resume being edited, including its sections, styles, etc.
    - pages/EditorPage.jsx: The main workspace component. It fetches the specific resume data based on the URL parameter (/editor/{resumeId}).
    - components/ResumeCanvas.jsx: The central component that renders the resume sections based on the resumeSlice state.
    - components/sections/ExperienceSection.jsx, etc.: Individual components for each section type.
    - components/Sidebars/LeftSidebar.jsx, components/Sidebars/RightSidebar.jsx.

---

6. Milestone 4: Implementing Core Interactive Features (ETA: 20 Hours)

This is where the application comes to life.

- **Task 4.1: On-Canvas Editing**
    1. Make text elements within section components editable using controlled inputs.

2. On onChange, dispatch an action to update the resumeSlice in real-time.
3. Implement a useDebounce custom hook to automatically call the PATCH API to save changes after the user stops typing for ~1.5 seconds. Provide a visual "Saving..." -> "Saved" indicator.

- **Task 4.2: Section Reordering (Drag-and-Drop)**
  1. Wrap the sections within the ResumeCanvas with react-beautiful-dnd components (DragDropContext, Droppable, Draggable).
  2. On drag end, dispatch an action to reorder the sections array in the resumeSlice.
  3. Send an API request to the backend with the new order of all sections.
- **Task 4.3: Template Switching & Design Customization**
  1. In the RightSidebar, create UI controls (color pickers, font dropdowns, template thumbnails).
  2. When a control is changed, dispatch an action to update the template_name or style object in the resumeSlice.
  3. The ResumeCanvas will dynamically apply CSS variables or conditional classes based on the style state, instantly reflecting the changes.

---

7. Milestone 5: Advanced Features & Finalization (ETA: 16 Hours)

This milestone adds the high-value features required for a senior-level submission.

- **Task 5.1: Text-Based PDF Download**
  1. For each resume template, create a corresponding set of PDF components using @react-pdf/renderer's primitives (<Page>, <View>, <Text>).
  2. The "Download" button will trigger a function that passes the current resumeSlice state as props to these PDF components.
  3. Use the library's PDFDownloadLink or blob provider to generate and download the file.
- **Task 5.2: Public Sharing**
  1. **Backend:** Create an endpoint /api/resumes/{id}/share that generates a unique share_slug for a resume. Create a public, unauthenticated endpoint /api/public/resume/{slug} to fetch resume data.
  2. **Frontend:** Add a "Share" button in the editor that calls the share endpoint and presents the user with a shareable URL (/share/{slug}). Create a new public page component at this route that renders a read-only version of the resume.
- **Task 5.3: Undo/Redo Stack (Bonus)**
  1. Install redux-undo.
  2. In the store, wrap the resumeSlice reducer with the undoable() higher-order reducer.

3. Add "Undo" and "Redo" buttons to the editor UI that dispatch the UndoAction and RedoAction from redux-undo.

---

8. Milestone 6: Deployment & Operations (ETA: 8 Hours)

This final milestone makes the project publicly accessible.

- **Task 6.1: Preparation for Production**
    1. Create a .env file for each service to manage environment variables (DATABASE_URL, SECRET_KEY, DEBUG, etc.).
    2. Configure static file serving and database settings for the production Django environment.
- **Task 6.2: Deployment**
    1. **Backend (Django):** Deploy to Heroku or a similar PaaS. Use Gunicorn as the WSGI server.
    2. **Frontend (React):** Deploy to Vercel or Netlify.
    3. **Proxy (Node.js):** Deploy as a serverless function on the same platform as the frontend (Vercel is ideal for this) to handle API routing.
- **Task 6.3: Final Documentation**
    1. Create a comprehensive README.md in the root repository. It must include:
        - Project overview and live demo link.
        - Detailed setup instructions for local development.
        - An explanation of the architecture and tech stack choices.
        - A brief overview of the API endpoints.
    2. (Bonus) Create an ERD diagram for the database schema.
    3. (Bonus) Record a short video walkthrough of the final application.