**Project Name: TaskFlow - A Full-Stack Team Productivity App**

**Description**: Build a collaborative task management app for teams. The app will allow users to create accounts, form teams, assign tasks, set deadlines, and track progress. This project will mimic the functionality of popular tools like Trello or Asana but at a smaller scale.

**Key Features**

**Authentication & Authorization (Auth.js)**

* **User Accounts**: Allow users to sign up/sign in using email/password or third-party providers like Google.
* **Role-Based Access Control (RBAC)**:
  + Admins: Manage team members and projects.
  + Members: View and update assigned tasks.

**Team Management**

* **Create Teams**: Users can create or join teams.
* **Invite Members**: Admins can send email invites to add users to their team.
* **Member Management**: View and manage team members (only available to admins).

**Task Management**

* **CRUD Operations**:
  + Create, Read, Update, and Delete tasks.
  + Assign tasks to specific team members.
* **Task Status**: Tasks can have statuses (To Do, In Progress, Completed).
* **Deadlines**: Include deadline tracking with alerts for overdue tasks.

**Dashboard**

* A personalized dashboard for users:
  + View tasks assigned to them.
  + See team activity (recent updates, new tasks).

**Real-Time Updates**

* Use WebSockets or a library like Pusher/Ably to implement real-time updates for task changes, comments, and notifications.

**File Uploads**

* Allow users to upload files/images related to tasks (e.g., a design file or document).

**Analytics (Optional)**

* Visualize team productivity with charts showing completed vs. pending tasks.
* Show overdue tasks or bottlenecks.

**Tech Stack**

1. **Frontend**:
   * Next.js (app directory with server components).
   * TypeScript for type safety.
   * Tailwind CSS for styling.
2. **Backend**:
   * Next.js API routes for backend logic.
   * Prisma for database ORM.
   * MongoDB for storing users, teams, and tasks.
3. **Authentication**:
   * Auth.js for secure, scalable authentication.
4. **Extras**:
   * Framer Motion for animations.
   * Axios or TanStack Query for API calls and state management.

**Learning Outcomes**

1. **Complex CRUD Operations**: Handle nested data relationships (e.g., teams, tasks, users) with Prisma.
2. **Role-Based Security**: Implement and test different user permissions using Auth.js.
3. **Real-Time Functionality**: Learn how to integrate real-time updates.
4. **File Handling**: Work with image/file uploads and storage.
5. **Scalability**: Optimize queries with Prisma and use server-side rendering (SSR) or static generation (SSG) where necessary.
6. **User Experience (UX)**: Build a polished UI with interactive elements and animations.
7. **Debugging & Deployment**: Deploy the project using platforms like Vercel, and ensure it works seamlessly in production.

Would you like a more detailed breakdown of any feature or a step-by-step guide to start this project?

The **TaskFlow** project is a comprehensive and challenging solo project that simulates a real-world software engineering experience. Below, I'll break down the steps, features, and technologies you'll need to build it, along with insights to tackle it effectively.

## ****Project Goal****

Create a task management platform where users can:

* Register, log in, and manage their profile securely.
* Create, edit, delete, and prioritize tasks.
* Organize tasks into projects.
* Track progress with statuses (e.g., To Do, In Progress, Done).
* Collaborate with other users by sharing projects.
* Use analytics to gain insights into task completion patterns.

## ****Key Features and Functionalities****

### 1. ****Authentication****

* **What to Implement:**
  + User registration and login using **Auth.js** with email/password and OAuth (e.g., Google, GitHub).
  + Role-based access control (e.g., Admin, Editor, Viewer).
* **Technologies:**
  + Auth.js for authentication.
  + Prisma's built-in adapter for managing user sessions.
  + MongoDB for storing user data.
* **Challenges:**
  + Protecting routes and ensuring only authenticated users access certain pages.
  + Implementing secure password storage and session management.

### 2. ****Task Management****

* **What to Implement:**
  + CRUD operations (Create, Read, Update, Delete) for tasks.
  + Tasks should include:
    - Title, description, due date, priority (e.g., High, Medium, Low).
    - Status (e.g., To Do, In Progress, Done).
  + Drag-and-drop functionality to move tasks between statuses (Kanban-style).
* **Technologies:**
  + Prisma for schema and database operations.
  + MongoDB collections for tasks and their relationships to users.
  + Framer Motion or React Beautiful DnD for drag-and-drop.
* **Challenges:**
  + Designing a scalable database schema for tasks and users.
  + Keeping the drag-and-drop interactions smooth and intuitive.

### 3. ****Project Management****

* **What to Implement:**
  + Group tasks into projects.
  + Allow users to invite collaborators to projects.
  + Assign tasks to specific users within a project.
* **Technologies:**
  + Additional Prisma models for Project and User\_Project relationships.
  + MongoDB for relational mapping.
  + Next.js API routes to manage collaboration invitations and permissions.
* **Challenges:**
  + Handling relational data efficiently with MongoDB.
  + Managing access control for collaborators (e.g., only the owner can delete a project).

### 4. ****Analytics****

* **What to Implement:**
  + Visualize progress using charts (e.g., tasks completed over time, tasks by status).
  + Provide a dashboard with key metrics (e.g., task completion rate, overdue tasks).
* **Technologies:**
  + **Chart.js** or **Recharts** for data visualization.
  + Prisma queries for aggregating task data.
  + Server-side rendering (SSR) or static generation (SSG) for analytics pages.
* **Challenges:**
  + Optimizing database queries for large datasets.
  + Ensuring visualizations are dynamic and user-friendly.

### 5. ****Real-time Features****

* **What to Implement:**
  + Real-time updates when tasks are added, edited, or deleted in a project.
  + Notifications for updates on shared projects.
* **Technologies:**
  + **WebSockets** or **Server-Sent Events** for real-time updates.
  + Tools like **Pusher** or **Socket.IO** to manage WebSocket connections.
* **Challenges:**
  + Synchronizing changes across multiple clients without performance issues.
  + Securing WebSocket connections for shared projects.

### 6. ****UI/UX Design****

* **What to Implement:**
  + Responsive design with Tailwind CSS.
  + Intuitive navigation using Next.js App Router.
  + Modals for task details, forms for adding/editing tasks.
* **Technologies:**
  + Tailwind CSS for styling.
  + Framer Motion for animations (e.g., smooth page transitions, interactive feedback).
  + DaisyUI for pre-styled components (optional).
* **Challenges:**
  + Creating a cohesive design system for the app.
  + Ensuring accessibility (e.g., keyboard navigation, ARIA labels).

### 7. ****Deployment****

* **What to Implement:**
  + Fully deploy the app to a production environment.
  + Use environment variables for sensitive credentials (e.g., database URL, Auth.js secrets).
* **Technologies:**
  + **Vercel** for deploying the frontend and backend (Next.js API routes).
  + MongoDB Atlas for a cloud-based database.
  + Monitor performance using tools like **Sentry** or **LogRocket**.
* **Challenges:**
  + Handling production-specific issues (e.g., cold starts, rate limits).
  + Scaling the app to handle multiple users simultaneously.

## ****Steps to Build****

1. **Plan and Design**
   * Draft user stories for each feature.
   * Design a database schema using Prisma.
   * Create wireframes or mockups for the UI.
2. **Setup the Project**
   * Initialize a Next.js project with TypeScript.
   * Install dependencies:
     + Auth.js, Prisma, Tailwind CSS, Framer Motion, etc.
3. **Build Features Incrementally**
   * Start with authentication and basic CRUD for tasks.
   * Gradually add advanced features like projects, collaboration, and analytics.
4. **Test Thoroughly**
   * Write unit tests for functions and components.
   * Test API routes using tools like Postman.
5. **Deploy and Iterate**
   * Deploy the app to Vercel.
   * Gather feedback and refine the app based on user experience.

## ****Potential Challenges****

* **Database Relationships**: MongoDB is document-based, so handling relationships (e.g., tasks assigned to projects) will require careful schema design and aggregation queries.
* **State Management**: Consider using **TanStack Query** or **Redux Toolkit** for managing global state efficiently.
* **Security**: Protect routes and sensitive data. Use middleware to validate user roles and permissions.
* **Scaling Features**: Start simple, and gradually add complexity as you test and refine the app.

## ****Stretch Goals****

* **Mobile App**: Use **Expo** with React Native to create a companion mobile app.
* **Task Reminders**: Implement push notifications or email reminders for due tasks.
* **Dark Mode**: Add a toggle for dark/light themes using Tailwind CSS.

This project not only tests your knowledge of Next.js, Prisma, MongoDB, and Auth.js but also provides a real-world use case that can be a strong addition to your portfolio. Let me know if you'd like detailed guidance on any specific part!