1. **Project Management Tool**:
   * Knowledge: Basic understanding of web development and databases.
   * Tools: HTML/CSS/JavaScript for frontend, a backend language like Python (Django or Flask), Ruby (Rails), or JavaScript (Node.js), SQL or NoSQL databases.
2. **Inventory Management System**:
   * Knowledge: Web development, databases.
   * Tools: HTML/CSS/JavaScript for frontend, a backend language, SQL or NoSQL databases.
3. **E-commerce Website**:
   * Knowledge: Web development, databases, and security.
   * Tools: HTML/CSS/JavaScript for frontend, a backend language, SQL or NoSQL databases, payment processing APIs.
4. **Cryptocurrency Tracker**:
   * Knowledge: Web development, APIs, basic understanding of financial markets.
   * Tools: Any programming language (Python/JavaScript are commonly used), data visualization libraries.
5. **Real-time Traffic Monitoring System**:
   * Knowledge: Web development, APIs, maps, and GPS data.
   * Tools: Any programming language, data visualization libraries.
6. **Health Monitoring System**:
   * Knowledge: Mobile development, databases.
   * Tools: A mobile development platform (iOS/Swift or Android/Java or Kotlin), SQL or NoSQL databases.
7. **Cloud-based File Storage System**:
   * Knowledge: Cloud storage services, file handling, security.
   * Tools: A backend language, cloud storage services like AWS S3 or Google Cloud Storage.
8. **Smart Parking System**:
   * Knowledge: APIs, maps, GPS data, possibly IoT programming, payment processing.
   * Tools: Any programming language, IoT tools for hardware interaction, payment processing APIs.
9. **AI Chatbot**:
   * Knowledge: Machine learning, natural language processing.
   * Tools: Python, NLP tools like NLTK or SpaCy, machine learning libraries like TensorFlow or PyTorch, chatbot frameworks like Dialogflow or Rasa.
10. **Privacy-Focused Browser Extension**:
    * Knowledge: Web development, web security and privacy.
    * Tools: HTML/CSS/JavaScript, browser extension APIs.
11. **Voice Controlled Home Automation System**:
    * Knowledge: IoT programming, voice recognition technology, home automation technology.
    * Tools: IoT programming tools, voice recognition APIs, home automation technologies like ZigBee, Z-Wave, or SmartThings.
12. **Sentiment Analysis Tool**:
    * Knowledge: Machine learning, natural language processing.
    * Tools: Python, NLP tools, machine learning libraries.
13. **Image Recognition System**:
    * Knowledge: Machine learning, computer vision.
    * Tools: Python, machine learning libraries like TensorFlow or PyTorch, computer vision libraries like OpenCV.
14. **Automated Resume Screening System**:
    * Knowledge: Machine learning, natural language processing, potentially document parsing.
    * Tools: Python, NLP tools, machine learning libraries.
15. **E-Learning Platform**:
    * Knowledge: Web development, databases, potentially video processing and storage.
    * Tools: HTML/CSS/JavaScript for frontend, a backend language, SQL or NoSQL databases, possibly video processing tools.

**Project Management Tool**:

* Knowledge: Basic understanding of web development and databases.
* Tools: HTML/CSS/JavaScript for frontend, a backend language like Python (Django or Flask), Ruby (Rails), or JavaScript (Node.js), SQL or NoSQL databases.

1. **Requirements Gathering and Analysis (1-2 weeks):** Understand and document what your project management tool needs to do. What features will it have? Who will use it and how? Define clear, measurable goals for the project.
2. **Design (1-2 weeks):** Plan your application's architecture and create wireframes or mockups of the user interface. Decide on the technologies you will use. You might also design your database schema at this stage.
3. **Development - Frontend (2-4 weeks):** Implement your designs in code, starting with the user interface. At this stage, you'll be writing HTML, CSS, and JavaScript to create the pages of your application.
4. **Development - Backend (3-5 weeks):** After that, develop the server-side part of your application. This includes setting up your server, defining routes, and implementing the logic for each route. You'll also set up your database and write code to interact with it.
5. **Integration and Testing (2-3 weeks):** Once your frontend and backend are both developed, you'll need to integrate them together and test the entire system. This includes unit tests, integration tests, and functional tests. Be prepared to find bugs and have to go back to your code to fix them.
6. **Deployment (1 week):** Finally, you'll deploy your application to a server so that it's accessible on the internet. This involves setting up a hosting service, uploading your code, and configuring the server.
7. **Maintenance (Ongoing):** After deployment, you'll enter the maintenance phase, where you'll need to monitor the application, fix any bugs that come up, and possibly add new features or improvements.

**Target Users:**

1. **Project Managers:** These are the main users who will be creating projects, assigning tasks, tracking progress, and managing resources.
2. **Team Members:** They need to update the status of their tasks, log hours, and communicate with others.
3. **Stakeholders:** They might not directly interact with the tool but would need access to reports and overviews of the project status.

**Features:**

1. **User Registration and Authentication:** Users should be able to register, log in, and manage their accounts. You might also want different levels of access for different types of users (like administrators, project managers, and team members).
2. **Project Creation and Management:** Project managers should be able to create new projects, set their parameters (like start date, end date, budget), and close them when completed.
3. **Task Management:** Tasks can be created, assigned to team members, and tracked through their lifecycle. Tasks should have properties like title, description, assignee, due date, and status.
4. **Time Tracking:** Team members should be able to log the hours they've spent on each task.
5. **Collaboration Tools:** These could include comments on tasks, file attachments, and even a chat feature.
6. **Notifications:** Users should be notified about relevant updates, like new tasks assigned to them or changes in task status.
7. **Reporting and Analytics:** The system should provide overviews of project progress, budget usage, task status, etc. These could be shown in the form of dashboards and could also be exported as reports for stakeholders.

**1. Design Phase:**

Start with a wireframe of your application. This includes a rough sketch of how your application will look. It will give you a roadmap to follow during the development process. You'll need to design pages for:

* User registration and login
* Logo on top.
* Fields for username and password.
* Buttons for "Log In" and "Register".
* Links for "Forgot Password" and "Register" (on the Login page) or "Log In" (on the Register page).
* Project creation and management

1. Navigation bar on top with links to different pages.
2. Main area of the page divided into two sections:
   * List of existing projects on the left, each with options to view, edit, or delete.
   * Form on the right for creating a new project, with fields for the project name, description, start date, end date, and budget.

* Task creation, assignment, and tracking

1. Navigation bar on top.
2. Dropdown to select a project.
3. Main area of the page divided into two sections:
   * List of existing tasks on the left, each with options to view, edit, or delete.
   * Form on the right for creating a new task, with fields for the task name, description, assignee, due date, and status.

* A dashboard or report page showing project progress

1. Navigation bar on top.
2. Dropdown to select a project.
3. A series of charts or graphs showing project progress, like a Gantt chart, a pie chart of task status, and a bar chart of budget usage.

**2. Frontend Development:**

* **Set Up Your Development Environment:**
* Angular
* **Frontend Framework/Library (Angular):**

1. Setup Routing:

Defining routes for all the pages application will have.

Home page,

User Registration page,

Login page,

Project Management page, and

Task Management page.

Define these routes in the **app-routing.module.ts** file.

1. Create Components:

Components control a patch of screen called a view.

For each of your routes, create a corresponding Angular component using the Angular CLI command **ng generate component componentName**.

1. Develop Components:

Start by creating static versions of each view with sample data, and then gradually add interactivity.

* + User Registration & Login: Create a simple form where users can input their details to register or login. Use Angular Forms for this.
  + Project Creation & Management: Allow users to create new projects, set project details and possibly add other users to the project. Might also have a view to list all the projects a user is involved in.
  + Task Creation, Assignment & Tracking: Within each project, users can create tasks, assign them to different users, and track their progress.

1. **Develop Services:** In Angular, services are used to share data and logic across components. Create services to handle operations like user authentication, project management, and task management. Initially, these services might use static data, but later on you'll modify these to communicate with your backend.
2. **Integrate With Backend:** Once your backend is ready, you'll replace the static data in your services with actual API calls to the backend. Angular's HttpClient module can be used to make these API calls.

* **Develop the Static Components:**

For each page in your wireframe, create the HTML and CSS. Start by creating the layout and then add more details. Here's a rough sequence you could follow:

* User Registration and Login Page: Create the form with fields for username and password, the buttons, and the links. Style it as desired.
* Project Creation and Management Page: Create the navigation bar, the list for projects, and the form for project details.
* Task Creation, Assignment, and Tracking Page: Create the dropdown for selecting a project, the list for tasks, and the form for task details.
* Dashboard or Report Page: Create the dropdown for selecting a project and placeholders for where your charts will go. You'll add the actual charts when you're integrating with your backend data.
* **Implement Interactivity with JavaScript:**

Once you have the static components, you can add interactivity. This can mean different things depending on whether you're using a JavaScript framework or not:

* Without a framework, you'll be using vanilla JavaScript or jQuery to manipulate the DOM based on user actions.
* With a framework, you'll define components, their state, and how they should render based on that state. You'll also define how state should update when certain events happen.
* **Responsive Design (Optional but Recommended):**

Ensure that your application is usable on both desktop and mobile by implementing a responsive design. You might use media queries in your CSS, a CSS framework like Bootstrap that includes a grid system, or a combination of both.

Remember, you won't actually be able to register users, create projects, or create tasks at this stage. You're just creating the structure and appearance of your application, and defining how it should react to user input. You'll make it functional when you integrate the frontend with the backend.

1. **User Registration (RegisterComponent)**: Implement a form for user registration. Use Angular's Reactive Forms to create form controls for fields like username, email, and password. You'll also need to create a method to handle form submissions, which should include form validation and an HTTP request to your backend to create the user.
2. **User Login (LoginComponent)**: Similarly, implement a login form. Upon successful login, the backend should return a token which your front-end should store and use for authenticated requests.
3. **Navigation**: Implement navigation links to allow users to navigate through the application. This could be done in AppComponent.
4. **Displaying Projects (ProjectComponent)**: Fetch and display a list of projects. This will involve sending a GET request to your backend, likely including the user's token in the headers to authenticate the request.
5. **Creating Projects (ProjectComponent)**: Implement a form to create new projects. Similar to the registration form, you'll need a method to handle form submissions, including validation and an HTTP request to your backend.
6. **Project Details (TaskComponent)**: When a user clicks on a project, navigate to a detail view where you display the project's tasks. This will involve parameterized routing and fetching data based on route parameters.
7. **Creating Tasks (TaskComponent)**: Implement a form to create new tasks within a project. Like the other forms, this will involve handling form submissions, validation, and an HTTP request to your backend.
8. **Error Handling**: Implement error handling for all your HTTP requests. This might involve displaying error messages to the user and retrying requests.
9. **Loading States**: Implement loading states for all operations that involve HTTP requests. This could be as simple as displaying a loading spinner and disabling forms/buttons while requests are in progress.
10. **User Logout**: Implement user logout functionality that clears the stored user token and redirects to the login screen.
11. **Styling**: Improve the styling of your application, making sure it's responsive and looks good on all screen sizes.
12. **Testing**: Write unit tests for your components and services to ensure they work as expected.

**3. Backend Development:**

* Integrating with a Backed (Node.js and MongoDB):
  + Setting up the backend: Create a new directory for the backend application.
  + Models set-up
    - User Model: used to store user’s info
    - Project Model: store the project details, such as name of the project, description, the users
    - Task model: store the task details, such as name of the tasks, description, and users it is assigned to, and due date.

Using Postman to test API endpoint:  
- Registration endpoint:  
- <http://localhost:4000/api/users/register>

‘Body’ tab, select ‘raw’ and ‘JSON’ data for new user,   
{ "name": "Test User",

"email": "test@example.com",

"password": "testpass",

"role": "user"

}  
  
{"message":"User registered successfully"}

* Login endpoint:
* [**http://localhost:4000/api/users/login**](http://localhost:4000/api/users/login)
* **JSON data for user:**
* **{ "email": "test@example.com", "password": "testpass" }**
* {"message":"User logged in successfully"}

1. **JWT Authentication**: Implement JSON Web Token (JWT) for maintaining session. It’s a stateless, session-less way of managing user authentication and is quite scalable. On successful login, you would send a JWT to the client which will be used for subsequent requests to access restricted areas of the application.
2. **User Profile**: Build a user profile page where users can update their profile details. This could be a simple form that submits a PATCH request to your '/users/:id' endpoint.
3. **Password Reset Functionality**: It's common for users to forget their password. Implementing a password reset functionality that sends a reset password link to the user's email would improve your application's user experience.
4. **Project and Task Management**: Since this appears to be a project management application, you would want to create other models like **Project** and **Task**. Projects would have many tasks. A **Project** model could contain fields like **name**, **description**, **startDate**, **endDate**, and **status**. A **Task** model could contain fields like **name**, **description**, **status**, **assignee** (user assigned to the task), and **dueDate**.
5. **Authorization**: You should limit what users can do based on their roles. For example, only a project manager can create or delete a project. This can be achieved by adding middleware to your routes that checks the role of the authenticated user.
6. **Frontend**: Build a frontend using a library or framework like React, Angular, or Vue. This frontend would make requests to your backend and present the response to your users. This way, users can interact with your application using a UI rather than making requests with Postman.
7. **Error Handling**: Improve your error handling to provide more accurate error messages. For instance, instead of just returning 'Error registering user', return a more descriptive error message. This can help a lot with debugging in the future.

**4. Integration and Testing:**

After your frontend and backend are both developed, integrate them together. Make sure data flows correctly from the frontend to the backend and back.

Write tests for your application. This should include unit tests for individual components and functions, integration tests for your server routes, and end-to-end tests for user journeys.

**5. Deployment:**

Once you're confident that your application is working as expected, it's time to deploy it to a server so it's accessible on the internet.