Task Tracking Project

# User Stories

**1. User Registration & Login (Functional)**

**As a new user, I want to register and log in so that I can access and manage my tasks securely.**

**Acceptance Criteria:**

* **Registration form requires name, email, and password.**
* **Password is encrypted in the database.**
* **JWT token is issued on successful login.**

**Scope:**

* **Covers user authentication with JWT, backend API endpoints, and frontend login/signup forms.**

**2. Task Creation (Functional)**

**As a logged-in user, I want to create a task with a title, description, estimate time.**

**Acceptance Criteria:**

* **All fields (title, status, estimate) are required.**
* **Task is saved in the PostgreSQL database.**
* **Tasks appear in the dashboard after creation**

**Scope:**

**Implements task creation with React frontend, Node.js/Express API, and database storage.**

**3. Task Viewing (Functional)**

**As a user, I want to view a list of my tasks with their current status and progress so that I can track what needs to be done.**

**Acceptance Criteria:**

* **Task list displays title, description, status, estimated hours, and logged time.**
* **Tasks are retrieved from the database and rendered in React.**
* **Progress bar shows logged time vs. estimated hours**

**Scope:**

**Covers task retrieval from database, API endpoint for listing tasks, and React-based UI rendering**

**4. Task Editing (Functional)**

**As a user, I want to edit an existing task so that I can update its information when things change.**

**Acceptance Criteria:**

* **Form pre-fills with existing task data.**
* **After editing, data is updated in the DB and UI.**

**Scope:**

**Implements task update functionality with React form, API endpoint, and database sync.**

**5. Task Deletion (Functional)**

**As a user, I want to delete tasks I no longer need so that I can keep my dashboard clean.**

**Acceptance Criteria:**

* **User can select a task and confirm deletion via a prompt.**
* **Task is removed from the database and UI.**
* **Only the task owner can delete the task.**

**Scope:**

**Covers task deletion with React UI, API endpoint, and database operation**

**6. Time Logging (Functional)**

**As a user, I want to log the number of hours I spend on a task so that I can track my effort.**

**Acceptance Criteria:**

* **Time log input accepts hours.**
* **Progress bar updates based on estimate vs. actual**

**Scope:**

**Implements time tracking with React UI, API endpoint, and database updates.**

**7. Change Task Status (Functional)**

**As a user, I want to change a task’s status (To-Do, In Progress, Done) so that I can reflect its current state.**

**Acceptance Criteria:**

* **Dropdown menu allows status selection.**
* **Status update is saved to the database.**

**Scope:**

**Covers status updates with React dropdown, API endpoint, and database sync.**

**8. Logout Functionality (Functional)**

**As a user, I want to log out so that I can securely end my session.**

**Acceptance Criteria:**

* **JWT token is removed from local storage.**
* **User is redirected to the login page.**

**Scope:**

**Implements logout functionality with React UI, JWT clearance,**

**9.Input Validation**

* **Summary:**
  + **The system must validate all user inputs to ensure security and data integrity.**
* **Scope:**
  + **Applies input validation across all forms and API endpoints.**
* **Acceptance Criteria:**
  + **All inputs ( task title, email, hours) are validated on frontend and backend.**
  + **Invalid inputs (negative hours, malicious scripts) are rejected with clear error messages.**

**10. Docker Containerization**

* **Summary:**
  + **The application services must be containerized using Docker for consistent deployment.**
* **Scope:**
  + **Covers infrastructure setup for deployment and portability.**
* **Acceptance Criteria:**
  + **Frontend (React), backend (Node.js/Express), and database (PostgreSQL) run in Docker containers.**
  + **Docker Compose file orchestrates all services.**
  + **Containers deploy successfully in development environment.**

# API Definitions

**1. User Registration**

* **Title**: User Registration (User Story 1)
* **Endpoint**: POST /auth/register
* **Description**: Registers a new user with email and password.
* **Request Body**: { "email": "string", "password": "string" }
* **Response: {**{ "user": { "id": "string", "email": "string" }, "message": "User registered successfully}**}**

**2. User Login**

* **Title**: User Login (User Story 1)
* **Endpoint**: POST /auth/login
* **Description**: Authenticates a user and returns a JWT token.
* **Request Body**: : { "email": "string", "password": "string" }
* **Response**: **{**{ "user": { "id": "string", "email": "string" }, "token": "string” }**}**

**3. Create Task**

* **Title**: Task Creation (User Story 2)
* **Endpoint**: POST /tasks
* **Description**: Creates a new task for the authenticated user.
* **Authorization**: Bearer token required.
* **Request Body: {**"title": "string","description": "string","estimate": "number","status": "string" }
* **Response:** {"task": {"id": "string", "title": "string", "description": "string", "estimate": "number","status": "string","loggedTime": "number", "userId": "string", "createdAt": "string" }}

**4. List Tasks**

* **Title: Task List View (User Story 3)**
* **Endpoint: GET /tasks**
* **Description: Retrieves all tasks for the authenticated user.**
* **Response**: {"tasks": [{ "id": "string", "title": "string","description": "string","estimate": "number","status": "string","loggedTime": "number","userId": "string","createdAt": "string"}]}