Group 3 : Taskify Project Documentation

Name: Huzaifa Abdul Rehman (23k-0782)
Name : Syed Ashhad Hassan (23k-0584)

Taskify - MERN Stack Task Management Application

Complete Project Documentation

Table of Contents

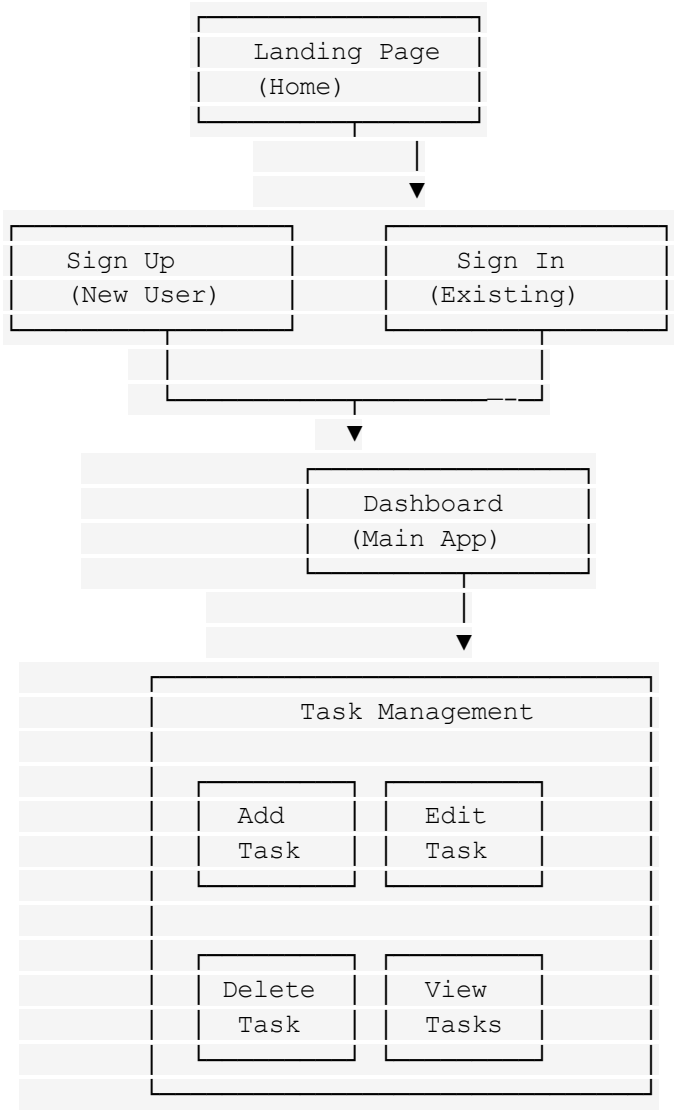
Project Overview

Taskify is a full-stack task management application built with the MERN stack (MongoDB, Express.js, React.js, Node.js). The application allows users to create, manage, and track tasks with features like priority levels, due dates, and status tracking..

Key Features:

- User authentication and authorization
- Task CRUD operations
- Task status management (Todo, In Progress, Done)
- Priority levels (High, Medium, Low)
- Due date tracking
- Responsive design

User Interface Flow Chart



Application Architecture

Application Architecture Overview

Frontend (React.js-based SPA – Single Page Application)

The user interface will be modular and component-based. Core screens and components include:

- **Navigation Header** – for app-wide links and user profile access
- **Form Inputs** – used in login, signup, task creation/editing
- **Error Handling Interface** – to catch and display frontend errors
- **Pages:**
  - **Home Page** – introductory screen or redirect to login/dashboard
  - **Login Page** – email/password authentication
  - **Signup Page** – user registration with basic info
  - **Dashboard** – displays and manages tasks in categories (To-Do, In Progress, Done)

State & Logic Management:

- Centralized state system for:
  - Authentication (logged-in user data, token storage)
  - Tasks (create, update, delete, filter, sort)
  - UI preferences (theme, language)

Backend (Node.js + Express.js REST API)

The backend will expose endpoints for users and task operations, with authentication, access control, and data persistence.

- **User APIs:**
  - Register, login, logout
  - Profile updates
  - Role-based access handling
- **Task APIs:**
  - Create, edit, update status
  - Filter by priority or status
  - Delete or archive
- **Admin APIs:**
  - View all users and tasks
  - Suspend/reactivate accounts
  - Audit user activity
- **Middle-layer Logic:**
  - Authentication with JWT
  - Async error handling
  - Middleware for role-checking and input validation

User Roles & Functions

These assumptions are based on expected use patterns of similar productivity apps

1. End User (Regular User)

Primary Functions:

- **Task Creation** (Usage: 5-10 times/day)
  - Create new tasks with title, description, priority, due date
  - Add detailed descriptions and attachments
- **Task Management** (Usage: 15-20 times/day)
  - Edit existing tasks
  - Update task status (Todo → In Progress → Done)
  - Change priority levels
  - Set/update due dates
- **Task Organization** (Usage: 8-12 times/day)
  - View tasks by status (Todo, In Progress, Done)
  - Filter tasks by priority
  - Sort tasks by due date
- **Task Deletion** (Usage: 2-3 times/day)
  - Remove completed or unnecessary tasks
- **Profile Management** (Usage: 1-2 times/week)
  - Update personal information
  - Change password
  - Upload profile picture

2. Admin User

Administrative Functions:

- **User Management** (Usage: 2-3 times/week)
  - View all registered users
  - Reset user passwords
- **System Monitoring** (Usage: Daily)
  - Monitor application performance
  - View system logs
  - Track user activity
- **Data Management** (Usage: Weekly)
  - Backup user data
  - Export system reports
  - Database maintenance

Planned Database Design (Schema-Level Concepts)

Users

Stores user-related data:

- Basic info: name, email, password, profile picture
- Account settings: theme, language, notification preferences
- Role & status: admin/user, isActive, lastLogin

Tasks

Stores task data:

- Metadata: title, description, due date, priority
- Status flow: todo → in-progress → done
- Relations: linked to a user ID
- Enhancements: tags, attachments, comments, time tracking

Sessions

Manages authentication lifecycle:

- Token info, expiration
- Device/browser/IP info
- Active/inactive status

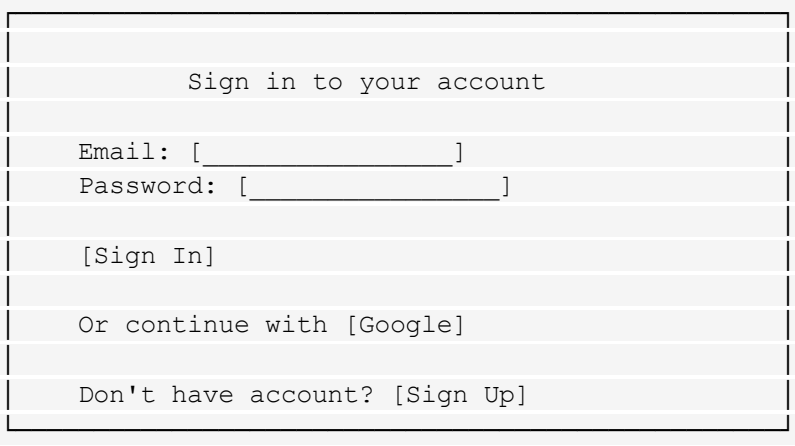
Frontend ↔ Backend ↔ Database Flow

- User logs in from frontend form
- Auth API validates credentials and returns a token
- Token is stored in frontend for session management
- Task Dashboard uses that token to:
  - Fetch tasks from backend
  - Send task edits or creation requests
- Backend stores/retrieves task data in MongoDB

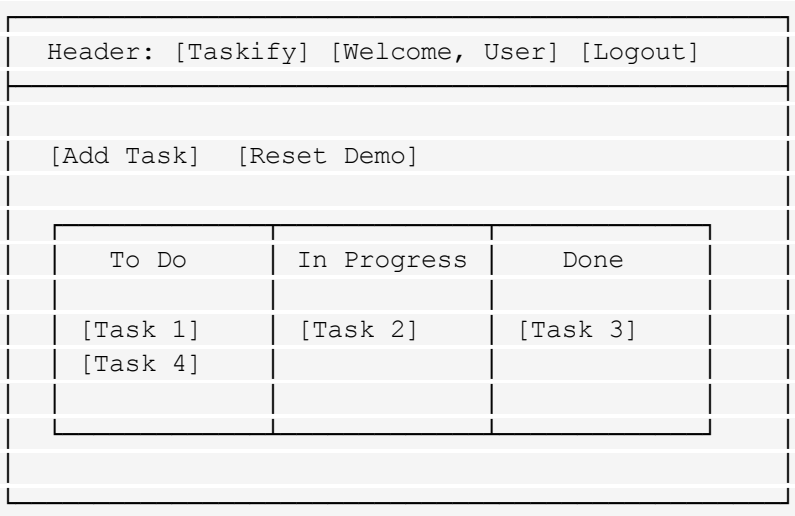
Wireframes & UI Design

1. Landing Page (Home)

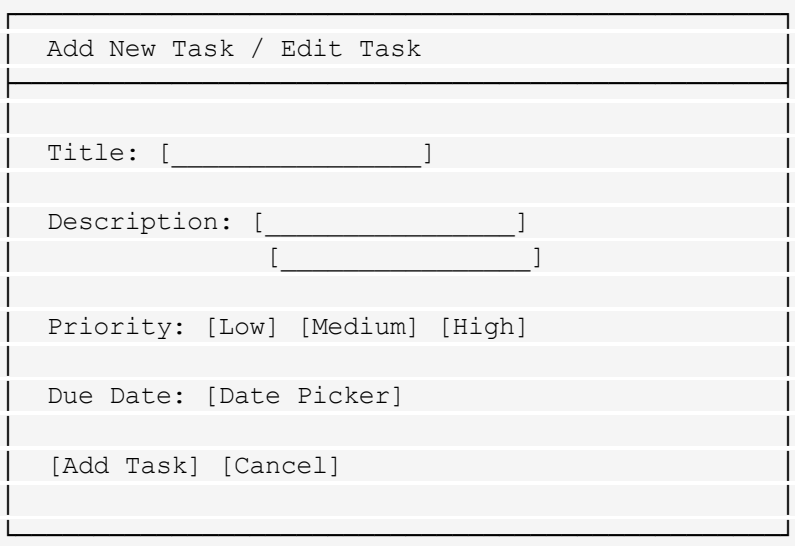
2. Login Page



3. Dashboard (Main Application)



4. Add/Edit Task Modal



Data Validation

Client-Side Validation (Frontend)

This validation is performed in the browser using form controls and logical rules. It improves user experience by catching issues before sending data to the server
.

1. User Registration
First Name / Last Name: Required. Minimum 2, maximum 50 characters.
Email: Must be in valid format (e.g., name@email.com). Should not already be used by another user.
Password: Required. Minimum 8 characters. Must include at least 1 uppercase letter, 1 lowercase letter, 1 number, and 1 special character.
Confirm Password: Must exactly match the entered password.

2. User Login
Email: Required. Must be a valid email format.
Password: Required. Must match the credentials of an existing user.

3. Task Creation / Editing
Title: Required. Minimum 3, maximum 100 characters.
Description: Optional. Maximum 500 characters.
Priority: Must be one of: Low, Medium, or High.
Due Date: Optional. If provided, there must be a date in the future.

Server-Side Validation (Backend)

These validations ensure the integrity and security of data on the server. They are enforced regardless of frontend checks.

1. Input Sanitization
Strip or escape unsafe characters from all inputs. Remove any HTML or JavaScript tags to prevent script injection. Ensure correct data types (e.g., string, date, number). Sanitize file uploads and query parameters.

2. Business Logic Validation
Check if the user is authenticated before allowing sensitive actions. Verify that a task being edited or deleted belongs to the requesting user. Prevent abuse through rate limiting or IP throttling. Manage user sessions to auto-logout expired tokens or block invalid logins.

API Design (Conceptual Overview)

This section describes the main API services that allow the frontend to communicate with the backend securely and efficiently.

Authentication Services

Register User: Collects signup information and stores a new user in the database.
Login User: Validates credentials and returns a token/session for future requests.
Logout User: Ends the session or invalidates the token.
View or Update Profile: Allows users to view and edit their personal information.
Google Sign-In (Planned): Alternate login method using OAuth from Google.

Task Management Services

Get All Tasks: Returns all tasks belonging to the logged-in user.
Create Task: Adds a new task to the user's list.
Get One Task: Retrieves a single task using its ID.
Update Task: Modifies title, description, or other fields.
Delete Task: Removes a task from the user’s list.
Change Task Status: Updates a task from "To-Do" → "In Progress" → "Done".

Admin Services

View All Users: Lists all registered accounts (with filters/search).
Update User Access: Temporarily suspend or reactivate accounts.
System Statistics: Admin-only dashboard to track activity and performance logs.

Authentication & Access

JWT (JSON Web Tokens): Keeps users logged in securely.
bcrypt: Protects passwords by hashing them before saving.
User Roles: Regular users can only manage their tasks, admins have more control.

Data Safety

Form Validation: Inputs are checked before sending to the server.
CORS Setup: Backend only accepts requests from trusted frontend.
Basic Protection: Data is cleaned to avoid risky inputs (like scripts or code).

User Privacy

Passwords are never stored in plain text.
Important data is sent using HTTPS.
Users can update or delete their personal information

Technical Stack

Frontend Technologies

- **React.js 18** - UI framework
- **Redux Toolkit** - State management
- **React Router DOM** - Navigation
- **Tailwind CSS** - Styling
- **Axios** - HTTP client
- **React Hook Form** - Form handling
- **Zod** - Schema validation
- **React Toastify** - Notifications

Backend Technologies

- **Node.js** - Runtime environment
- **Express.js** - Web framework
- **MongoDB** - Database
- **Mongoose** - ODM
- **JWT** - Authentication
- **bcryptjs** - Password hashing
- **cors** - Cross-origin requests
- **cookie-parser** - Cookie handling

Development Tools

- **Vite** - Build tool
- **ESLint** - Code linting
- **Nodemon** - Development server
- **Git** - Version control