Taskify - Technical Design Document

Table of Contents

Introduction and System Overview

Introduction

Taskify is a cloud-based task management and collaboration platform designed to help teams plan, organize, and track their work efficiently. It provides tools for creating, assigning, and managing tasks while enabling seamless collaboration among team members.

With a focus on security and reliability, Taskify ensures that all data is stored securely in the cloud, offering a robust solution for modern teams to work together effectively.

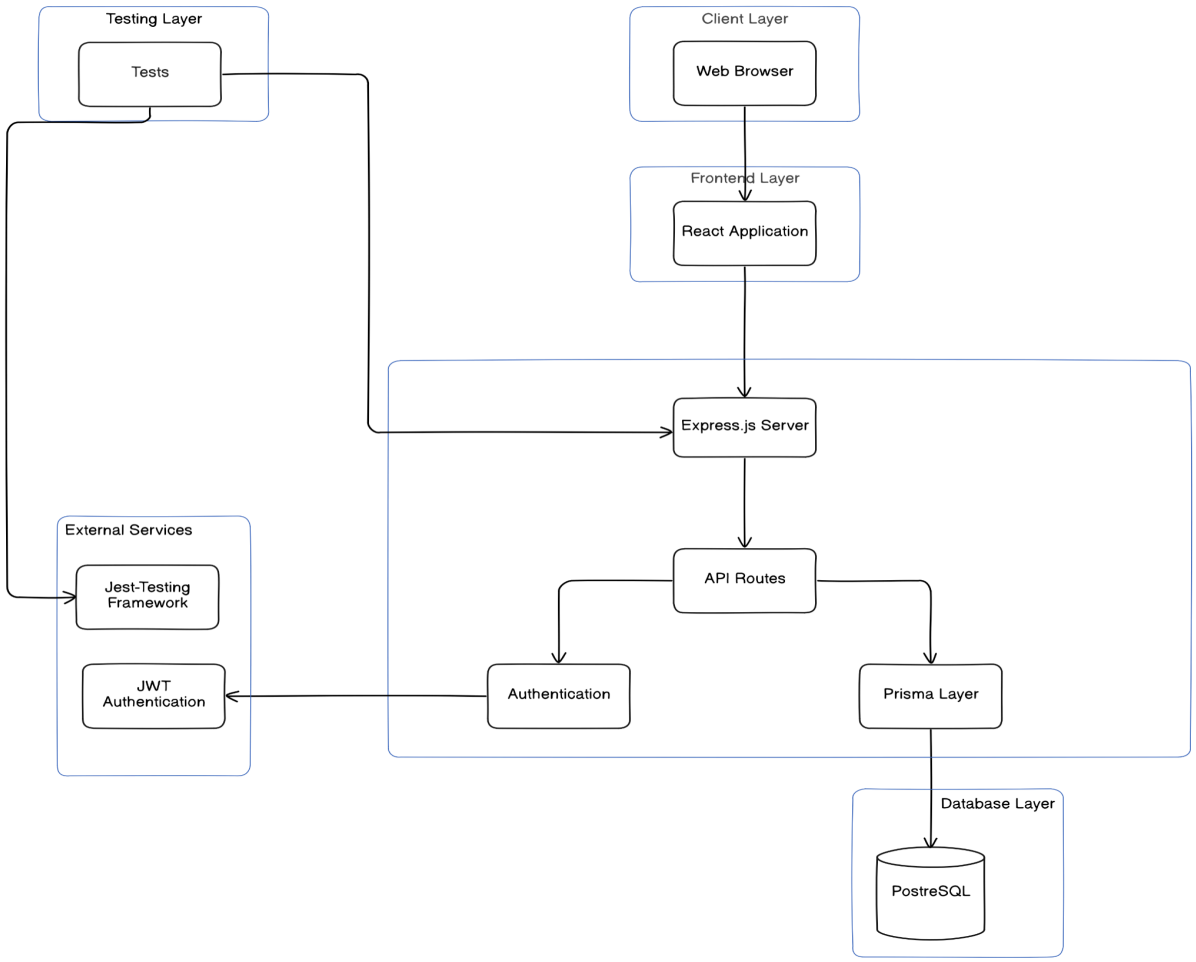
This design document provides a comprehensive overview of the project's architecture, components, technology stack, and design decisions. It aims to serve as a guide for developers, stakeholders, and contributors.

System Overview

Taskify is built using the MERN stack (MongoDB, Express.js, React, Node.js), utilizing modern web development practices. The application is structured to provide scalability, maintainability, and a responsive user experience across devices.

Architecture

High-Level Architecture



The system follows a three-tier architecture, comprising:

1. **Frontend**: Developed with React.js, responsible for the client-side user interface and interactions.
2. **Backend API**: Built with Express.js and Node.js, handling server-side logic, API endpoints, authentication, and business logic.
3. **Testing:** Developed using JS Testing library Jest.

Backend Design

Technologies Used

> Node.js : JavaScript runtime environment.

> Express.js : Web application framework for building APIs.

> MongoDB : NoSQL database for data storage.

> Mongoose : ORM (Object Relational Model) library for MongoDb.

> JWT : JSON Web Tokens for authentication.

> Bcrypt : Hashing Library.

Project Structure

> backend/server.js : Entry point of the server application.

> backend/middlewares/ : Contains middleware functions, including authentication.

> backend/config/ : Mongoose Package for Database access.

> backend/routes/ : Defines Version 1 API endpoints for authentication, users, admin, space.

> backend/controllers/ : All the db logic exported as a module and available for all app use.

Backend Design - API, Database Schema and Middlewares

API Design

The backend exposes RESTful API endpoints categorized under:

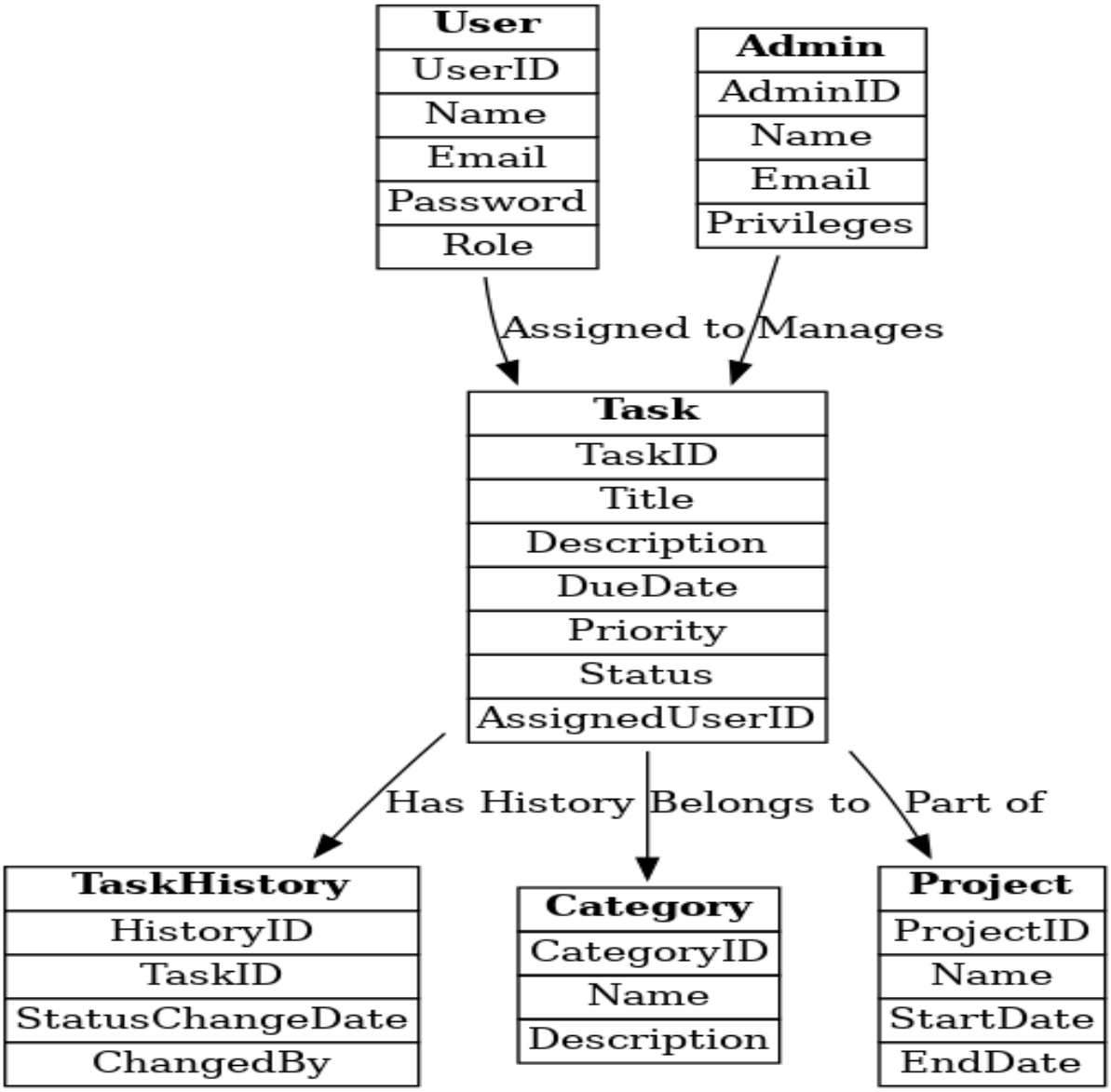
- User (/api/v1)
  - POST /signup : User Signup.
  - POST /signin : User login and JWT token issuance.
  - POST /user/metadata : To retrieve User MetaData from the Database..
- Task (/api/v1/task)
  - POST /create : Create a new task for a project.
  - DELETE /:taskId : Delete a task by its taskID.
  - GET /all : Retrieve all tasks assigned to the user or team.
  - GET /:spaceId :Get space info corresponding to spaceId.
  - POST /subtask: Create a new subtask under a specific task.
- Admin (/api/v1/admin)
  - POST /task : Create a new task within any project.
  - PUT /task/:taskId : Update an existing task by taskId.
  - GET /dashboard : Retrieve system-wide statistics and insights for admins.

Middlewares

Authentication Middleware

- Validates JWT tokens sent in the Authorization header.
- Attaches the authenticated user's information to the request object.
- Protects routes that require authentication.

Schema



Frontend Design

Technologies Used

> React.js : JavaScript library for building user interfaces.

> React Router DOM : Handling client-side routing.

> Tailwind CSS : Utility-first CSS framework for styling.

> Vite : Build tool for faster development.

> ESLint : Linting utility to maintain code quality.

Project Structure

> main.jsx : Entry point of the React application.

> App.jsx : Main application component.

> app.css : Global CSS and Tailwind directives.

Frontend Design - Routing, State Management and Key Components

Routing

Implemented using React Router:

- `/` : Landing Page
- `/login` : Login page
- `/dashboard` : dashboard page
- `/tasks` : Show all tasks
- `/team` : Show all team details
- `/trashed` : Deleted or Completed Tasks

State Management

- Data Fetching:
  - Utilizes fetch or Axios API.
  - Handles loading and error states.

Security Considerations

Authentication

- JWT Tokens:
  - Securely generated and signed with a secret key.
  - Stored in the client's localStorage.
- Password Security:
  - Passwords hashed using bcrypt.js before storing in the database.
  - Plain passwords are never stored or logged.

Authorization

- Protected Routes:
  - Backend routes require valid JWT tokens.
  - Frontend routes use higher-order components to restrict access.

Deployment Plans

Environment Setup

- Backend Environment Variables:
  - JWT_SECRET and JWT_PASSWORD: Secret key for signing JWTs.
- Frontend Environment Variables:
  - VITE_WS_URL: Base URL for the WS Server.

Deployment Steps

1. Backend Deployment:

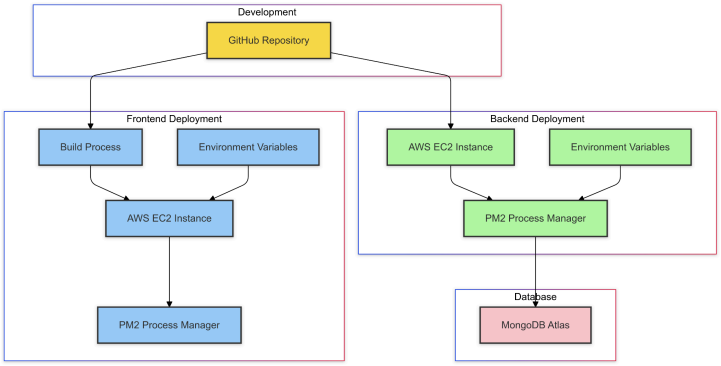
- Host on platforms like Heroku, AWS EC2, or DigitalOcean.
- Ensure environment variables are securely set.

2. Frontend Deployment:

- Build the React application using npm run build.
- Host static files on services like AWS.

3. Domain and SSL:

- Configure a custom domain.
- Set up SSL certificates for secure HTTPS communication.



Future Enhancements

Technical Improvements

- Switch to TypeScript :
    - Introduce TypeScript for type safety and better maintainability.
- State Management Library :
    - Implement Context API, Zustang for more complex state needs.
- Better UI :
    - Implement Better UI for the users to enjoy and interact.

Feature Enhancements

- Advanced Issue Tracking : Support customizable issue types, workflows, and fields for better task organization.
- Customizable Dashboards : Allow users to create personalized dashboards with real-time insights.
- Sprint Management : Implement sprint planning, backlog management, and progress tracking.
- Role-based Permissions : Provide granular access control for secure project management.
- Third-Party Integrations : Enable integration with tools like Slack, GitHub, and Trello for improved workflow.

Conclusion

The Taskify project showcases an emerging concept in Task management environments, focusing on creating an interactive, immersive experience. It integrates JavaScript for functionality, with a clear structure for future development. As the project evolves, further features and improvements could be added, expanding its potential applications in virtual spaces.