System Design Document

Overview

This document provides a detailed overview of the system architecture for the Task Management Application, which includes both frontend and backend components.

Architecture

Components

1. **Frontend**

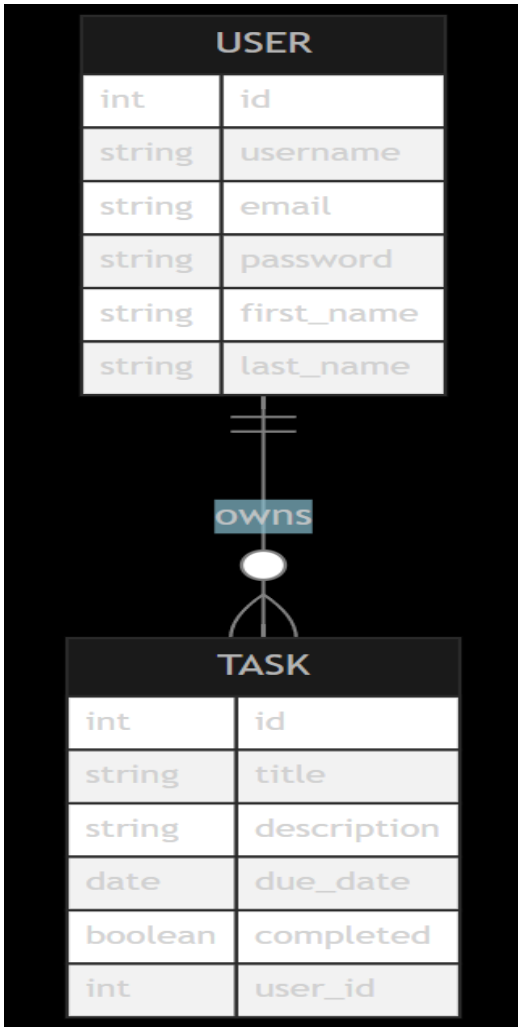
- Built with React.
- Provides a user interface for task management and user authentication.
- Communicates with the backend via RESTful API.

2. **Backend**

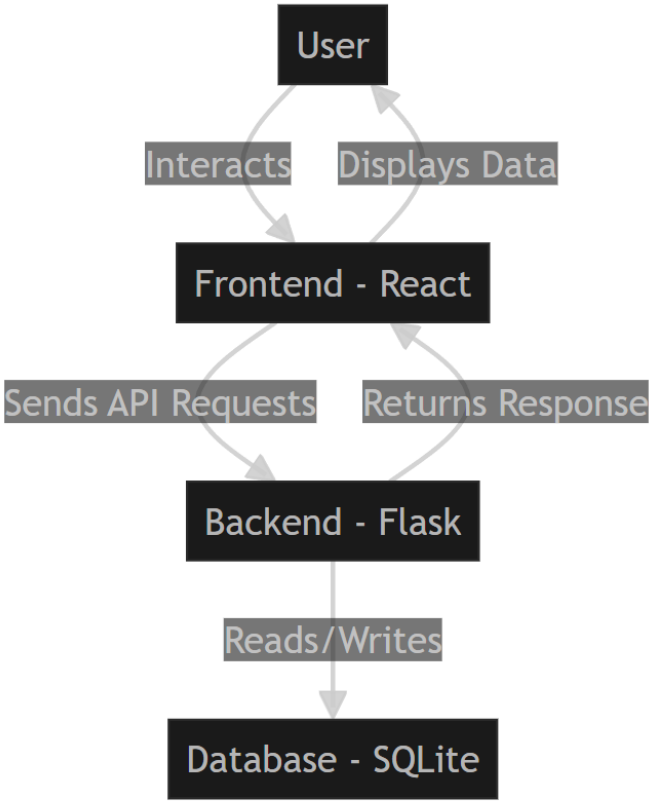
- Built with Flask.
- Provides API endpoints for user authentication and task management.
- Handles business logic and data persistence.

3. **Database**

- SQLite database for storing user and task data.
- Managed by SQLAlchemy ORM.



Architecture Diagram



Deployment

- **Docker**: The application is containerized using Docker, with a single Dockerfile handling both frontend and backend builds.
- **Docker Compose**: Used to orchestrate multi-container applications, managing both frontend and backend services.

Data Flow

1. **User Registration/Login**

- User submits registration/login form on the frontend.
- Frontend sends a POST request to the backend API.
- Backend validates the request, interacts with the database, and returns a response.

2. **Task Management**

- User interacts with tasks on the frontend.

- Frontend sends requests to the backend API for task creation, updating, or deletion.
- Backend processes the request, updates the database, and returns a response.

API Endpoints

- **Authentication**
- POST /api/auth/register: Register a new user.
- POST /api/auth/login: Authenticate a user and return a token.
- **Tasks**
- GET /api/tasks: Retrieve all tasks.
- POST /api/tasks: Create a new task.
- PUT /api/tasks/:id: Update an existing task.
- DELETE /api/tasks/:id: Delete a task.

Security

- **JWT Authentication**: Used for securing API endpoints.
- **Environment Variables**: Sensitive information such as secret keys are stored in environment variables.

Configuration

- **Config File**: Centralized configuration in config.py for easy management of environment-specific settings.

Future Enhancements

- **Scalability**: Consider migrating to a more robust database system for larger datasets.
- **Load Balancing**: Implement load balancing for handling increased traffic.
- **CI/CD**: Set up continuous integration and deployment pipelines for automated testing and deployment.