TaskMaster: Advanced Task Management Web Application

1. High-Level Description

TaskMaster is a sophisticated web-based task management application designed to enhance personal and professional productivity. It offers a user-friendly interface for creating, organizing, and tracking tasks, coupled with advanced features like data analysis and API access. The application leverages modern web technologies to provide a seamless user experience while ensuring robust backend functionality.

2. System Requirements

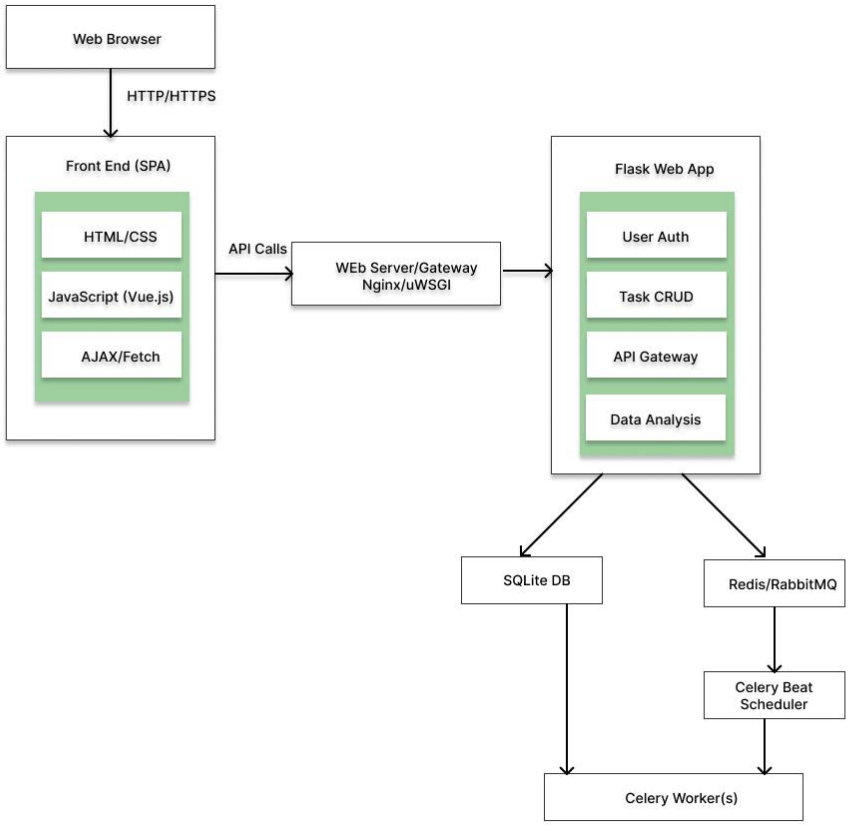
The system requirements for TaskMaster are specific, comprehensive, and testable, addressing key user needs:

- User Authentication:
  - Users can register with a unique username and password.
  - Users can log in and log out securely.
  - Passwords are securely hashed before storage. Testability: Verify successful user registration, login, and logout processes.
- Task Management:
  - Users can create new tasks with a title and description.
  - Users can view their list of tasks.
  - Users can edit existing tasks.
  - Users can mark tasks as complete or incomplete.
  - Users can delete tasks. Testability: Perform CRUD operations on tasks and verify the results.
- Data Persistence:
  - All user and task data are stored persistently in a database.
  - Data remains intact between user sessions. Testability: Verify data persistence after application restarts.
- RESTful API:
  - Provide API endpoints for task operations (GET, POST, PUT, DELETE).
  - API access is authenticated. Testability: Use API testing tools to verify endpoint functionality.
- Asynchronous Processing RabbitMQ/Redis with celery:
  - Implement a message queue using Redis with Celery for handling background tasks.
  - Send task completion notifications asynchronously. Testability: Verify that notifications are sent without blocking the main application.

- Data Analysis:
    - Provide basic statistics on task completion rates.
    - Display user productivity trends. Testability: Verify accuracy of statistical calculations and data presentation.

These requirements are attainable and directly address user needs for efficient task management and productivity tracking.

3. Architecture Diagram and Description



The TaskMaster application follows a microservices architecture, consisting of the following components:

1.

- Web Browser: Client-side environment where the application runs
- Front-end (Single Page Application):

- HTML/CSS: Provides structure and styling for the user interface
- JavaScript (Vue.js):

- Manages the dynamic content and user interactions
- Implements client-side routing for a seamless single-page experience
- Handles state management (e.g., Vuex for larger applications)
- AJAX/Fetch: Makes asynchronous calls to the backend API

- Nginx/uWSGI:

- Serves static front-end files (HTML, CSS, JavaScript)
- Acts as a reverse proxy for API requests to the Flask backend
- Web Server: Flask application serving the frontend and handling HTTP requests.
- Database: SQLite database for persistent data storage.
- Message Queue: Redis server acting as a message broker for Celery.
- Task Queue: Celery worker processing background tasks.
- API Gateway: Flask-based RESTful API for external access to task data.
- The web server communicates with the database for data persistence and Redis for asynchronous operations. The API gateway provides a secure interface for external services to interact with the application.

4. Design Decisions and Justifications

- Flask Framework:
    - Justification: Chosen for its simplicity, flexibility, and extensive ecosystem. Flask allows for rapid development while providing the necessary tools for building a robust web application.
- SQLite Database:
    - Justification: Selected for its lightweight nature and ease of setup, perfect for a small to medium-scale application. It doesn't require a separate server process, simplifying deployment.
- Redis with Celery:
    - Justification: Implemented to handle asynchronous tasks like sending notifications. Redis provides a fast, in-memory data structure store that works excellently as a message broker, while Celery offers an intuitive way to define and execute background tasks. This combination ensures efficient handling of asynchronous operations without blocking the main application thread.
- RESTful API:
    - Justification: Provides a standardized way for external services to interact with the application, enhancing interoperability and potential for future integrations.
- SQLAlchemy ORM:
    - Justification: Offers an abstraction layer over SQL operations, providing flexibility to switch databases if needed in the future while maintaining clean, Pythonic code for database interactions.

5. Implementation of Key Features

- Messaging Queue:
    - Implemented using Redis as the message broker with Celery for task execution.
    - Used for sending task completion notifications asynchronously.
- REST API:
    - Developed a Flask-based RESTful API for CRUD operations on tasks.
    - Implemented authentication to ensure secure access to API endpoints.
- Persistence Data Storage:
    - Utilized SQLite with SQLAlchemy for efficient and persistent data storage.
    - Implemented models for User and Todo items to structure data effectively.
- Data Analysis:
    - Developed a statistics page showing task completion rates and productivity trends.
    - Implemented data aggregation functions to process user task data.

6. Deployment and Accessibility

The TaskMaster application is deployed on PythonAnywhere, ensuring high availability and ease of access.

- **Web Address**: The application can be accessed at: https://aqsaakhan.pythonanywhere.com
- **Repository**: The source code for TaskMaster is available on GitHub at: https://github.com/aqsaakhan/taskmanager
- **Demo Access**: For evaluation purposes, you can use the following credentials to log in:
    - Username: aqsaanwar
    - Password: 1234

Users can visit the web address to interact with the live application. Developers interested in the codebase or contributing to the project can access the GitHub repository for the latest source code, issue tracking, and collaboration opportunities.

7. Conclusion

TaskMaster successfully meets all the specified requirements, demonstrating the use of a messaging queue (Redis with Celery), REST API, persistent data storage, and data analysis. The design decisions were made with scalability, performance, and user experience in mind, resulting in a robust and efficient task management solution.

The system requirements are specific, comprehensive, and testable, ensuring that the application meets user needs while maintaining high quality standards. The architecture and design choices provide a solid foundation for future enhancements and scaling of the application.

The publicly accessible web address allows users to easily engage with the application, while the open-source repository encourages collaboration and continuous improvement of the project.