Design Document for Real-Time Inventory Management System

Problem Statement:

In modern inventory management systems, maintaining real-time updates across multiple clients can be challenging, especially with traditional client-server architectures relying on periodic polling. This project addresses the need for instant inventory synchronization and efficient data handling by leveraging Svelte for dynamic UI updates, Flask with WebSockets for real-time communication, and DuckDB for lightweight, high-performance data storage.

The Real-Time Inventory Management System provides a seamless solution where users can add, update, and delete products, instantly reflecting changes on all connected clients. This approach ensures data consistency, reduces latency, and enhances user experience, making it ideal for businesses that require real-time inventory tracking and management.

Objectives:

- Provide a real-time solution for adding, updating, and deleting products.
- Ensure **consistent synchronization** between the front end and backend.
- Leverage **lightweight, scalable technologies** for performance efficiency.
- Create a **user-friendly interface** for seamless inventory management.

Project Overview:

Frontend (Svelte + TailwindCSS):

- The user interface uses Svelte and Flowbite Svelte for reactive updates and TailwindCSS for styling.
- Flowbite-Svelte components are used for modals, buttons, and tables.
- Real-time updates are handled through WebSockets.

Backend (Flask + Flask-SocketIO):

- The backend, built with Flask, acts as the intermediary between the front end and the database.
- Flask-SocketIO enables real-time bi-directional communication using WebSockets.
- A lightweight DuckDB database is used to store product data.

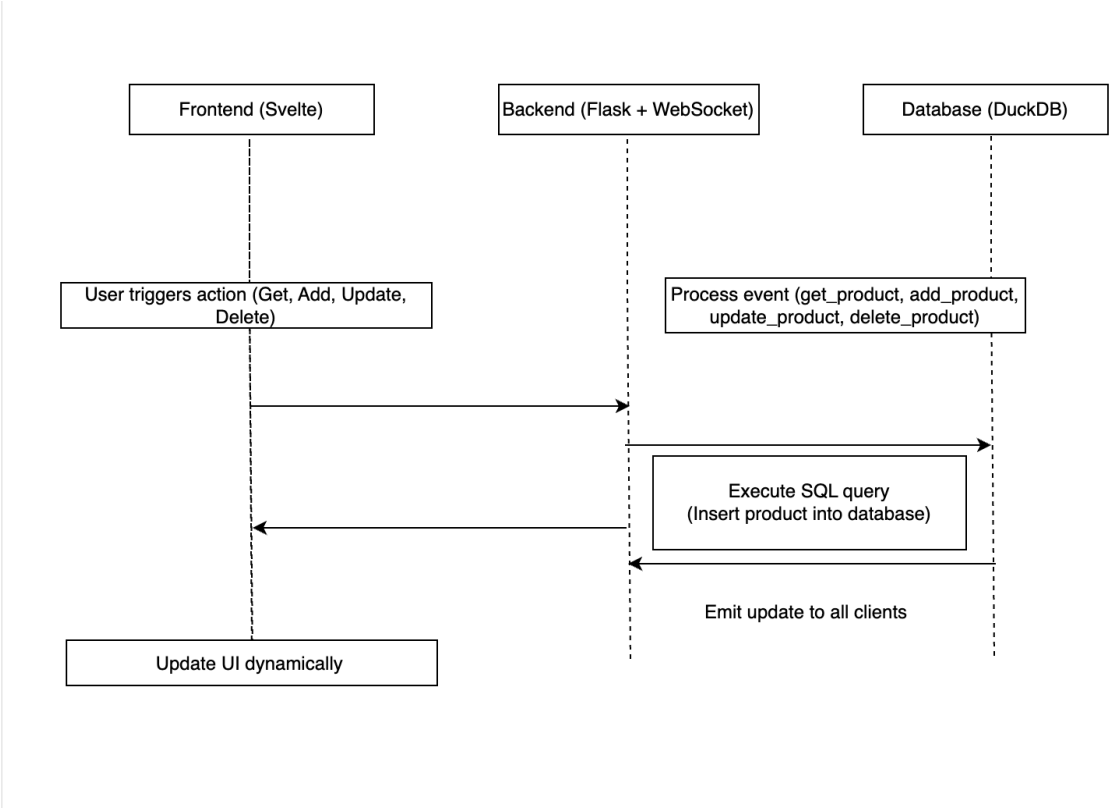
Database (DuckDB):

- DuckDB is chosen for its high performance in analytical queries and lightweight storage.

Workflow:

1. Users interact with the **front end** (Svelte app) to perform CRUD operations.
2. These operations are sent to the **backend** using **Socket.IO** events.
3. The **Flask server** processes these events and interacts with the **DuckDB** database to perform the necessary operations.
4. Upon success, updates are broadcast to all connected clients via WebSocket.

Sequence Diagram:



Contribution:

This project showcases:

- My ability to build a full-stack application from scratch.
- Proficiency in using modern web technologies like Svelte and Flask.
- Real-time data handling using WebSockets.
- Strong problem-solving skills in overcoming technical challenges.