**Solution Architecture Document**

**Project Name ISWL**

**Domain: Inventory / Supply Chain / Warehousing / Logistics**

**Version: 1.0**

**Date: 27TH August 2024**

**Author: ABDUL SHADRACH**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to describe the solution architecture for the [Inventory/Supply Chain/Warehousing/Logistics] system. This architecture is designed using GoLang and PostgreSQL for the backend and Next.js for the frontend.

**1.2 Scope**

This document covers the architectural design of the system, including technology stack, data flow, system components, integrations, and non-functional considerations such as performance, scalability, and security.

**1.3 Definitions and Acronyms**

* **API**: Application Programming Interface
* **DB**: Database
* **ORM**: Object-Relational Mapping
* **CI/CD**: Continuous Integration/Continuous Deployment
* **JWT**: JSON Web Token

**2. Solution Overview**

The solution is a web-based application that manages [Inventory/Supply Chain/Warehousing/Logistics] operations. It will provide real-time tracking of resources, order management, and visibility across different operational activities.

* **Frontend**: The client interface will be developed using Next.js, leveraging React for component-based UI development and SSR (Server-Side Rendering) to enhance SEO and performance.
* **Backend**: The backend will be developed using GoLang, which will serve as the API layer handling business logic, authentication, and interaction with the PostgreSQL database.
* **Database**: PostgreSQL will be used as the primary data store for all operational and transactional data.

**3. System Components**

**3.1 Frontend (Next.js)**

* **Technology Stack**: Next.js, React, TailwindCSS
* **Responsibilities**:
  + Rendering dynamic pages with SSR and CSR (Client-Side Rendering).
  + Interacting with backend APIs to fetch data and display it to the user.
  + Handling routing and state management across the application.
  + Authentication and authorization flows via JWT tokens.
  + Responsive design using TailwindCSS for UI/UX consistency across devices.

**3.2 Backend (GoLang)**

* **Technology Stack**: GoLang, Gorilla Mux (Router), PostgreSQL, JWT for authentication
* **Responsibilities**:
  + Serve as an API gateway for all frontend interactions.
  + Handle business logic for Inventory, Supply Chain, and Logistics workflows.
  + User authentication and authorization using JWT tokens.
  + CRUD operations for managing resources like products, orders, vendors, etc.
  + Integrating third-party services for shipping, tracking, and order processing.
  + Implementing background jobs for batch processes like reordering and reporting.

**3.3 Database (PostgreSQL)**

* **Technology Stack**: PostgreSQL, SQL
* **Responsibilities**:
  + Store all application data, including inventory records, order details, shipment data, and user information.
  + Use SQL for data manipulation and querying.
  + Indexing for optimized query performance and scalability.
  + Backup, replication, and failover mechanisms for high availability.

**4. Architectural Diagram**

Below is the high-level architecture of the system:

mermaid

Copy code

graph TD;

User-->Frontend(Next.js);

Frontend-->Backend(GoLang);

Backend-->Database(PostgreSQL);

Backend-->ThirdPartyServices(Shipping,Tracking);

Database-->BackupService;

**5. Data Flow**

1. **User Interaction**: The user interacts with the system via the Next.js frontend. The frontend communicates with the GoLang backend through RESTful APIs.
2. **API Layer**: The backend receives the user requests, processes the business logic (such as inventory updates, order placements), and interacts with the PostgreSQL database for data persistence and retrieval.
3. **Database Transactions**: All transactional data related to orders, inventory, and logistics are stored and queried from PostgreSQL.
4. **Third-Party Integrations**: For logistics operations, the backend integrates with third-party services (e.g., shipping and tracking APIs).
5. **Authentication**: The backend uses JWT tokens to handle secure authentication and authorization of users.

**6. System Design and Component Details**

**6.1 Frontend Design**

* **Pages**: The Next.js frontend will have a set of core pages:
  + Dashboard: Displaying key metrics and reports for inventory and supply chain.
  + Inventory Management: Adding, updating, and tracking inventory items.
  + Order Management: Viewing, processing, and tracking orders.
  + Logistics Management: Managing and tracking shipments.
* **UI Components**: React components will be developed for modularity and reusability, e.g., forms, tables, modals, etc.
* **State Management**: Context API or a third-party library (like Zustand or Redux) will be used to manage global state.

**6.2 Backend Design**

* **Router**: Gorilla Mux will be used to handle routing of HTTP requests.
* **Service Layer**: Each business domain (inventory, orders, logistics) will have its service handling logic and database interactions.
* **Database Access**: Go’s database/sql package will be used to interact with PostgreSQL, with proper connection pooling and transaction management.

**6.3 Database Design**

* **Tables**:
  + users: Stores user information.
  + inventory: Stores inventory item details.
  + orders: Stores order information.
  + shipments: Tracks the status of shipments.
  + vendors: Stores vendor details.
* **Relationships**:
  + orders are linked to users and inventory.
  + shipments are linked to orders.

**6.4 API Endpoints**

* **User Management**:
  + POST /api/auth/register: Register a new user.
  + POST /api/auth/login: Login and receive a JWT token.
  + GET /api/users/me: Get current user details.
* **Inventory Management**:
  + GET /api/inventory: Fetch all inventory items.
  + POST /api/inventory: Add a new inventory item.
  + PUT /api/inventory/:id: Update an inventory item.
  + DELETE /api/inventory/:id: Delete an inventory item.
* **Order Management**:
  + GET /api/orders: Fetch all orders.
  + POST /api/orders: Place a new order.
  + PUT /api/orders/:id: Update an order status.

**6.5 Authentication and Security**

* **JWT**: Secure all endpoints using JWT tokens.
* **Roles**: Implement role-based access control (RBAC) for users (Admin, Manager, Staff).
* **Security Best Practices**: Secure sensitive data with encryption, ensure secure communication with HTTPS, and implement SQL injection and XSS protection.

**7. Non-Functional Requirements**

**7.1 Performance**

* The backend must handle up to 100,000 concurrent users and up to 1,000 API requests per second with a response time of less than 200ms.

**7.2 Scalability**

* The architecture should support horizontal scaling, allowing multiple instances of the GoLang backend to run concurrently, managed through a load balancer.

**7.3 Reliability**

* PostgreSQL replication should be implemented for high availability, with backup and failover mechanisms in place.

**7.4 Security**

* OAuth2 or OpenID Connect for user authentication.
* HTTPS for all communication between frontend, backend, and third-party services.

**7.5 CI/CD**

* Automated CI/CD pipelines should be set up to handle code testing, building, and deployment using tools like GitHub Actions, Jenkins, or CircleCI.

**8. Conclusion**

This solution architecture document outlines the high-level and low-level components required to build the [Inventory / Supply Chain / Warehousing / Logistics] system using GoLang, PostgreSQL, and Next.js. The design prioritizes scalability, security, and performance to ensure the system is robust and adaptable to business needs.