**1. Database Schema**

**1.1 Database Overview**

The database schema is structured to support the main operations of the system (inventory, orders, logistics, etc.). PostgreSQL will be used for relational data storage with strict foreign key relationships, ensuring data integrity across the system.

**1.2 Tables and Fields**

1. **users**
   * **id** (UUID, PK): Unique identifier for each user.
   * **name** (VARCHAR): Full name of the user.
   * **email** (VARCHAR): User's email address.
   * **password\_hash** (TEXT): Hashed password using bcrypt or Argon2.
   * **role** (ENUM): User's role (Admin, Manager, Staff).
   * **created\_at** (TIMESTAMP): Timestamp of when the user was created.
   * **updated\_at** (TIMESTAMP): Timestamp of the last update.
2. **inventory**
   * **id** (UUID, PK): Unique identifier for each inventory item.
   * **name** (VARCHAR): Name of the inventory item.
   * **sku** (VARCHAR): Unique SKU for the item.
   * **quantity** (INT): Available quantity of the item.
   * **price** (DECIMAL): Price of the item.
   * **vendor\_id** (UUID, FK): Link to the vendor supplying the item.
   * **created\_at** (TIMESTAMP): Timestamp of when the item was added.
   * **updated\_at** (TIMESTAMP): Timestamp of the last update.
3. **orders**
   * **id** (UUID, PK): Unique identifier for each order.
   * **user\_id** (UUID, FK): Link to the user who placed the order.
   * **inventory\_id** (UUID, FK): Link to the inventory item.
   * **status** (ENUM): Order status (Pending, Shipped, Delivered, Cancelled).
   * **quantity** (INT): Quantity of the item ordered.
   * **total\_price** (DECIMAL): Total price of the order.
   * **created\_at** (TIMESTAMP): Timestamp of order creation.
   * **updated\_at** (TIMESTAMP): Timestamp of the last update.
4. **shipments**
   * **id** (UUID, PK): Unique identifier for each shipment.
   * **order\_id** (UUID, FK): Link to the associated order.
   * **tracking\_number** (VARCHAR): Unique tracking number for the shipment.
   * **shipping\_status** (ENUM): Shipping status (Pending, In Transit, Delivered).
   * **carrier** (VARCHAR): Carrier handling the shipment (FedEx, UPS, etc.).
   * **estimated\_delivery** (TIMESTAMP): Estimated delivery time.
   * **created\_at** (TIMESTAMP): Timestamp of shipment creation.
   * **updated\_at** (TIMESTAMP): Timestamp of the last update.
5. **vendors**
   * **id** (UUID, PK): Unique identifier for each vendor.
   * **name** (VARCHAR): Name of the vendor.
   * **contact\_info** (TEXT): Contact details of the vendor.
   * **created\_at** (TIMESTAMP): Timestamp of vendor creation.
   * **updated\_at** (TIMESTAMP): Timestamp of the last update.

**1.3 Relationships**

* **users** to **orders**: One-to-many (one user can place many orders).
* **inventory** to **orders**: One-to-many (one inventory item can be included in many orders).
* **orders** to **shipments**: One-to-one (each order has one shipment).
* **vendors** to **inventory**: One-to-many (one vendor can supply many inventory items).

**1.4 Constraints**

* **Foreign Keys**: Enforced between tables (e.g., user\_id in orders references id in users).
* **Unique Constraints**: email in users, sku in inventory, tracking\_number in shipments.
* **Indexes**: Placed on columns like email, sku, tracking\_number for faster lookups.

**2. API Design**

**2.1 Overview**

The API is RESTful and follows best practices for REST API design. It is built using GoLang and serves JSON over HTTP(S). JWT-based authentication is used to secure endpoints, with role-based access control (RBAC) implemented.

**2.2 API Endpoints**

Here are detailed API endpoints for core operations:

**2.2.1 Authentication Endpoints**

* POST /api/auth/register
  + **Description**: Registers a new user.
  + **Request Body**:

json

Copy code

{

"name": "John Doe",

"email": "john@example.com",

"password": "password123"

}

* + **Response**:

json

Copy code

{

"token": "JWT\_TOKEN\_HERE"

}

* POST /api/auth/login
  + **Description**: Authenticates a user and returns a JWT.
  + **Request Body**:

json

Copy code

{

"email": "john@example.com",

"password": "password123"

}

* + **Response**:

json

Copy code

{

"token": "JWT\_TOKEN\_HERE"

}

**2.2.2 Inventory Management Endpoints**

* GET /api/inventory
  + **Description**: Fetch all inventory items.
  + **Response**:

json

Copy code

[

{

"id": "1234",

"name": "Item A",

"sku": "SKU1234",

"quantity": 100,

"price": 19.99

}

]

* POST /api/inventory
  + **Description**: Add a new inventory item.
  + **Request Body**:

json

Copy code

{

"name": "Item A",

"sku": "SKU1234",

"quantity": 100,

"price": 19.99,

"vendor\_id": "5678"

}

* + **Response**:

json

Copy code

{

"id": "1234",

"message": "Inventory item created successfully"

}

**2.2.3 Order Management Endpoints**

* GET /api/orders
  + **Description**: Fetch all orders placed by the current user.
  + **Response**:

json

Copy code

[

{

"id": "12345",

"inventory\_id": "1234",

"quantity": 2,

"total\_price": 39.98,

"status": "Pending"

}

]

* POST /api/orders
  + **Description**: Place a new order.
  + **Request Body**:

json

Copy code

{

"inventory\_id": "1234",

"quantity": 2

}

* + **Response**:

json

Copy code

{

"id": "12345",

"message": "Order placed successfully"

}

**3. Security Protocols**

**3.1 Authentication and Authorization**

* **JWT**: JSON Web Tokens are used for stateless authentication. Upon login, a token is issued to the client, which is then sent with each request in the Authorization header. The backend validates the token and extracts the user information for authentication.

Example JWT payload:

json

Copy code

{

"sub": "user\_id",

"role": "Admin",

"iat": 1609459200,

"exp": 1609462800

}

* **OAuth2/OpenID Connect (Optional)**: Depending on system requirements, OAuth2 can be implemented for third-party logins.

**3.2 Role-Based Access Control (RBAC)**

* **Roles**: Different user roles will have different levels of access (Admin, Manager, Staff).
  + **Admin**: Full access to all resources (users, inventory, orders, etc.).
  + **Manager**: Access to manage inventory and orders but not user management.
  + **Staff**: Limited access to view-only operations for inventory and orders.
* **Route Guards**: Route handlers will enforce role checks based on the role claim in the JWT. Only authorized roles can access specific endpoints.

**3.3 Data Security**

* **Encryption**: All sensitive data (passwords, personal information) will be encrypted in transit (TLS/SSL) and at rest (AES encryption in PostgreSQL).
* **Password Hashing**: Passwords are stored as bcrypt/Argon2 hashes. No plaintext passwords are ever stored in the database.
* **CSRF Protection**: To prevent cross-site request forgery, the frontend will include CSRF tokens in forms, ensuring that requests come from the authorized user.
* **CORS**: The backend will be configured to allow only authorized origins, preventing unauthorized cross-origin requests.

**3.4 API Security**

* **Rate Limiting**: API requests will be rate-limited to prevent abuse or DDoS attacks.
* **Input Validation**: All input will be validated and sanitized to prevent SQL injection, XSS, and other common vulnerabilities.
* **Security Audits**: Regular security audits and code reviews will be conducted to identify and patch vulnerabilities.