Web-Based Inventory and Order Management System for Sampath Grocery Store

Design an Entity-Relationship Diagram (ERD) for a Web-Based Inventory and Order Management System for Sampath Grocery Store, supporting both retail and wholesale operations, with e-commerce website integration, home delivery, and loyalty card features. The system must handle product batch tracking, delivery routing, and automatic reorder notifications with manager approval. Ensure all entities are correctly defined with attributes, relationships, and cardinalities, and include support for credit sales and frequent delivery items (e.g., bread, string hoppers).

Entities and Attributes:

1. **User**
    a. Represents customers, staff, admins, and delivery drivers.
    b. Attributes:
        i. `user_id` (Primary Key, Integer)
        ii. `username` (String, Unique)
        iii. `password` (String, Encrypted)
        iv. `email` (String, Unique)
        v. `role_id` (Foreign Key to Role, Integer)
        vi. `created_at` (Timestamp)
2. **Role**
    a. Defines user permissions (e.g., admin, customer, delivery).
    b. Attributes:
        i. `role_id` (Primary Key, Integer)
        ii. `role_name` (String, e.g., "admin", "customer", "delivery")
        iii. `permissions` (JSON/String, e.g., "manage_inventory", "place_order")
3. **Product**
    a. Represents items for sale (e.g., rice, string hoppers).
    b. Attributes:
        i. `product_id` (Primary Key, Integer)

    ii. `name` (String)
    iii. `category_id` (Foreign Key to Category, Integer)
    iv. `barcode_number` (String, Unique)
    v. `brand` (String, Optional)
    vi. `unit` (String, e.g., "kg", "pack")
    vii. `packaging_type` (String, e.g., "bottle", "pouch")
    viii. `description` (Text, Optional)
    ix. `is_frequent_delivery_item` (Boolean, e.g., true for bread, string hoppers)
    x. `created_at` (Timestamp)
4. **ProductBatch**
    a. Tracks specific batches of products for expiry and stock management.
    b. Attributes:
        i. `batch_id` (Primary Key, Integer)
        ii. `product_id` (Foreign Key to Product, Integer)
        iii. `batch_number` (String, Unique)
        iv. `purchase_price` (Decimal)
        v. `selling_price` (Decimal)
        vi. `stock_quantity` (Integer)
        vii. `expiry_date` (Date, Optional)
        viii. `manufactured_at` (Date, Optional)
        ix. `added_at` (Timestamp)
5. **Category**
    a. Groups products (e.g., grains, vegetables).
    b. Attributes:
        i. `category_id` (Primary Key, Integer)
        ii. `category_name` (String)
        iii. `description` (Text, Optional)
6. **Cart**
    a. Temporary storage for customer-selected items.
    b. Attributes:
        i. `cart_id` (Primary Key, Integer)
        ii. `user_id` (Foreign Key to User, Integer)
        iii. `batch_id` (Foreign Key to ProductBatch, Integer)
        iv. `quantity` (Integer)
        v. `added_at` (Timestamp)
7. **Order**
    a. Represents customer purchases (retail or wholesale).

    b. Attributes:
        i. `order_id` (Primary Key, Integer)
        ii. `customer_id` (Foreign Key to Customer, Integer)
        iii. `order_date` (Timestamp)
        iv. `status_id` (Foreign Key to OrderStatus, Integer)
        v. `grand_total` (Decimal)
        vi. `payment_id` (Foreign Key to Payment, Integer)
        vii. `delivery_address` (String)
        viii. `delivery_type` (String, e.g., "standard", "route", "scheduled")
        ix. `requested_delivery_time` (Timestamp, Optional)
        x. `delivery_id` (Foreign Key to Delivery, Integer, Nullable)
8. **OrderItem**
    a. Details items in an order with line totals.
    b. Attributes:
        i. `item_id` (Primary Key, Integer)
        ii. `order_id` (Foreign Key to Order, Integer)
        iii. `batch_id` (Foreign Key to ProductBatch, Integer)
        iv. `quantity` (Integer)
        v. `unit_price` (Decimal)
        vi. `line_total` (Decimal, Calculated as `quantity * unit_price`)
9. **OrderStatus**
    a. Tracks order progress.
    b. Attributes:
        i. `status_id` (Primary Key, Integer)
        ii. `status_name` (String, e.g., "pending", "packed", "shipped", "delivered")
10. **Inventory**
    a. Manages stock levels for product batches.
    b. Attributes:
        i. `inventory_id` (Primary Key, Integer)
        ii. `batch_id` (Foreign Key to ProductBatch, Integer)
        iii. `quantity` (Integer)
        iv. `reorder_point` (Integer, e.g., 10 kg)
        v. `reorder_quantity` (Integer, e.g., 50 kg)
        vi. `last_updated` (Timestamp)
11. **StockAlert**
    a. Notifies low stock levels.
    b. Attributes:

i. `alert_id` (Primary Key, Integer)
ii. `batch_id` (Foreign Key to ProductBatch, Integer)
iii. `threshold` (Integer)
iv. `alert_date` (Timestamp)
v. `status` (String, e.g., "active", "resolved")

12. **ReorderRequest**
    a. Manages reorder requests with manager approval.
    b. Attributes:
        i. `request_id` (Primary Key, Integer)
        ii. `batch_id` (Foreign Key to ProductBatch, Integer)
        iii. `current_quantity` (Integer)
        iv. `reorder_quantity` (Integer)
        v. `request_date` (Timestamp)
        vi. `manager_id` (Foreign Key to User, Integer)
        vii. `status` (String, e.g., "pending", "approved", "rejected")
        viii. `approval_date` (Timestamp, Nullable)
        ix. `supplier_id` (Foreign Key to Supplier, Integer, Nullable)
13. **Supplier**
    a. Represents external vendors.
    b. Attributes:
        i. `supplier_id` (Primary Key, Integer)
        ii. `name` (String)
        iii. `contact_info` (String)
        iv. `address` (String)
        v. `registered_at` (Timestamp)
14. **PurchaseOrder**
    a. Orders to suppliers for restocking.
    b. Attributes:
        i. `po_id` (Primary Key, Integer)
        ii. `supplier_id` (Foreign Key to Supplier, Integer)
        iii. `batch_id` (Foreign Key to ProductBatch, Integer)
        iv. `quantity` (Integer)
        v. `order_date` (Timestamp)
        vi. `expected_delivery_date` (Timestamp)
        vii. `status` (String, e.g., "ordered", "received")
15. **Payment**
    a. Tracks transaction records.
    b. Attributes:

i. `payment_id` (Primary Key, Integer)
ii. `order_id` (Foreign Key to Order, Integer)
iii. `method_id` (Foreign Key to PaymentMethod, Integer)
iv. `amount` (Decimal)
v. `payment_date` (Timestamp)
vi. `status` (String, e.g., "completed", "pending", "credit")

16. **PaymentMethod**
    a. Defines payment types.
    b. Attributes:
        i. `method_id` (Primary Key, Integer)
        ii. `method_name` (String, e.g., "cash", "card", "online")
17. **Customer**
    a. Represents registered buyers with loyalty card details.
    b. Attributes:
        i. `customer_id` (Primary Key, Integer)
        ii. `user_id` (Foreign Key to User, Integer)
        iii. `name` (String)
        iv. `tel_number` (String)
        v. `email` (String, Optional)
        vi. `loyalty_card_number` (String, Unique, Optional)
        vii. `credit_limit` (Decimal)
        viii. `is_loyalty_member` (Boolean)
        ix. `preferred_delivery_area` (String, Optional)
18. **CustomerProfile**
    a. Stores detailed customer information.
    b. Attributes:
        i. `profile_id` (Primary Key, Integer)
        ii. `customer_id` (Foreign Key to Customer, Integer)
        iii. `name` (String)
        iv. `address` (String)
        v. `phone_number` (String)
        vi. `order_history` (JSON/Text)
19. **Invoice**
    a. Represents billing documents.
    b. Attributes:
        i. `invoice_id` (Primary Key, Integer)
        ii. `order_id` (Foreign Key to Order, Integer)
        iii. `amount` (Decimal)

iv. `issue_date` (Timestamp)
v. `status` (String, e.g., "paid", "due")

20. **Analytics**
    a. Stores data for business insights.
    b. Attributes:
        i. `analytics_id` (Primary Key, Integer)
        ii. `type` (String, e.g., "sales", "inventory", "delivery")
        iii. `data` (JSON)
        iv. `generated_at` (Timestamp)
21. **Report**
    a. Represents generated reports.
    b. Attributes:
        i. `report_id` (Primary Key, Integer)
        ii. `analytics_id` (Foreign Key to Analytics, Integer)
        iii. `title` (String)
        iv. `generated_at` (Timestamp)
22. **Notification**
    a. Sends alerts and updates (e.g., reorder requests, delivery status).
    b. Attributes:
        i. `notification_id` (Primary Key, Integer)
        ii. `user_id` (Foreign Key to User, Integer)
        iii. `message` (String, e.g., "Reorder request for 50 kg rice")
        iv. `type` (String, e.g., "order", "reorder", "delivery")
        v. `sent_at` (Timestamp)
        vi. `status` (String, e.g., "sent", "read")
23. **Feedback**
    a. Captures customer reviews.
    b. Attributes:
        i. `feedback_id` (Primary Key, Integer)
        ii. `user_id` (Foreign Key to User, Integer)
        iii. `batch_id` (Foreign Key to ProductBatch, Integer)
        iv. `rating` (Integer, 1-5)
        v. `comment` (String)
        vi. `submitted_at` (Timestamp)
24. **Delivery**
    a. Manages delivery of orders (retail or wholesale).
    b. Attributes:
        i. `delivery_id` (Primary Key, Integer)

ii. `order_id` (Foreign Key to Order, Integer)
iii. `driver_id` (Foreign Key to Driver, Integer)
iv. `vehicle_id` (Foreign Key to Vehicle, Integer)
v. `route_id` (Foreign Key to DeliveryRoute, Integer, Nullable)
vi. `dispatch_time` (Timestamp)
vii. `delivery_time` (Timestamp, Nullable)
viii. `status` (String, e.g., "pending", "out_for_delivery", "delivered")

25. **Driver**
    a. Represents delivery personnel.
    b. Attributes:
        i. `driver_id` (Primary Key, Integer)
        ii. `user_id` (Foreign Key to User, Integer)
        iii. `name` (String)
        iv. `phone_number` (String)
        v. `license_number` (String)
        vi. `status` (String, e.g., "available", "on_delivery")
26. **Vehicle**
    a. Represents delivery vehicles.
    b. Attributes:
        i. `vehicle_id` (Primary Key, Integer)
        ii. `vehicle_number` (String, Unique)
        iii. `type` (String, e.g., "bike", "van", "lorry")
        iv. `capacity` (Float, e.g., in kg or packages)
        v. `status` (String, e.g., "available", "in_use", "maintenance")
27. **DeliveryRoute**
    a. Defines daily delivery routes for frequent items (e.g., string hoppers).
    b. Attributes:
        i. `route_id` (Primary Key, Integer)
        ii. `route_name` (String, e.g., "Colombo North")
        iii. `driver_id` (Foreign Key to Driver, Integer)
        iv. `areas_covered` (JSON/String, e.g., ["Area1", "Area2"])
        v. `start_time` (Timestamp)
        vi. `end_time` (Timestamp, Nullable)
        vii. `status` (String, e.g., "active", "completed")
28. **DeliveryRequest**
    a. Stores loyalty customer requests for scheduled deliveries (e.g., 30 string hoppers).
    b. Attributes:

i. `request_id` (Primary Key, Integer)
ii. `customer_id` (Foreign Key to Customer, Integer)
iii. `batch_id` (Foreign Key to ProductBatch, Integer)
iv. `quantity` (Integer)
v. `delivery_date` (Timestamp)
vi. `submitted_at` (Timestamp)
vii. `status` (String, e.g., "pending", "confirmed", "rejected")

Relationships and Cardinalities:

- **User** to **Role**: 1:N (One user has one role, one role applies to many users).
- **User** to **Customer**: 1:N (One user can be linked to one customer, one customer has one user).
- **User** to **Driver**: 1:N (One user can be a driver).
- **Customer** to **CustomerProfile**: 1:1 (One customer has one profile).
- **Customer** to **Order**: 1:N (One customer places many orders).
- **Customer** to **DeliveryRequest**: 1:N (One customer makes many delivery requests).
- **Product** to **Category**: N:1 (Many products belong to one category).
- **Product** to **ProductBatch**: 1:N (One product has many batches).
- **ProductBatch** to **Cart**: N:1 (Many cart items reference one batch).
- **ProductBatch** to **OrderItem**: N:1 (Many order items reference one batch).
- **ProductBatch** to **Inventory**: N:1 (Many inventory records track one batch).
- **ProductBatch** to **StockAlert**: N:1 (Many alerts for one batch).
- **ProductBatch** to **PurchaseOrder**: N:1 (Many purchase orders for one batch).
- **Order** to **OrderItem**: 1:N (One order has many items).
- **Order** to **OrderStatus**: N:1 (Many orders have one status).
- **Order** to **Payment**: 1:1 (One order has one payment).
- **Order** to **Invoice**: 1:N (One order can have multiple invoices, e.g., partial payments).
- **Order** to **Delivery**: 1:1 (One order has one delivery, if applicable).
- **Payment** to **PaymentMethod**: N:1 (Many payments use one method).
- **Delivery** to **Driver**: N:1 (Many deliveries assigned to one driver).
- **Delivery** to **Vehicle**: N:1 (Many deliveries use one vehicle).
- **Delivery** to **DeliveryRoute**: N:1 (Many deliveries follow one route, if route-based).
- **DeliveryRequest** to **Customer**: N:1 (Many requests from one customer).
- **DeliveryRequest** to **ProductBatch**: N:1 (Many requests for one batch).
- **ReorderRequest** to **Supplier**: N:1 (Many requests sent to one supplier).

- **ReorderRequest** to **User**: N:1 (Many requests approved by one manager).
- **Notification** to **User**: N:1 (Many notifications sent to one user).
- **Feedback** to **User**: N:1 (Many feedbacks from one user).
- **Feedback** to **ProductBatch**: N:1 (Many feedbacks for one batch).

Business Logic:

- **Loyalty Customers**: Customers with `is_loyalty_member = true` can place `DeliveryRequest` for frequent items (e.g., 30 string hoppers for tomorrow) via the e-commerce website, specifying delivery date and time.
- **Delivery Routes**: Daily routes (`DeliveryRoute`) are defined for frequent items, covering specific areas. Deliveries are assigned to routes based on `preferred_delivery_area` in `Customer`.
- **Reorder Notifications**: When `Inventory.quantity < Inventory.reorder_point`, a `ReorderRequest` is created and sent to the manager (`User` with role "manager") via `Notification`. Upon approval, a `PurchaseOrder` is sent to the `Supplier`.
- **Credit Sales**: Track in `Payment` with `status = "credit"` and link to `Customer.credit_limit`.

ERD Requirements:

- Use crow's foot notation to show cardinalities.
- Include all attributes in entity boxes.
- Show foreign key relationships clearly.
- Ensure the diagram supports retail/wholesale orders, delivery, and loyalty features.
- Export the diagram as PNG or PDF for documentation.

Notes

- This revised prompt removes all production-related entities (29-34), relationships, and business logic references, as requested. It incorporates the corrected `Product` with `barcode_number` and `batch_number` (via `ProductBatch`), enhanced `Customer` with loyalty card details, `Order` with `OrderItem` for line totals and grand totals, `Inventory` with reorder logic, and delivery (`Delivery`, `Driver`, `Vehicle`, `DeliveryRoute`, `DeliveryRequest`) entities.

- You can use this prompt in AI tools like **Eraser.io**, **Miro AI**, or **Lucidchart** to generate the ERD.
- The relationships ensure seamless integration between e-commerce orders, delivery routing, and reorder processes.

Professional Prompt for Developing the Web-Based Inventory and Order Management System

Prompt:

Develop a standalone **Web-Based Inventory and Order Management System** for Sampath Grocery Store, supporting retail and wholesale operations, e-commerce website integration, home delivery, and loyalty card features. The system must operate **offline** (no internet dependency) for an interview demonstration, using **Bootstrap 5** for the front end, **Spring Boot** for the backend, and an embedded **H2 database** (instead of MySQL for offline compatibility) for database connectivity. The e-commerce website must function offline using **Service Workers** and **2300161**for client-side storage. Implement a responsive dashboard for admins, staff, and customers, and provide a clear file structure for the project.

System Requirements:

- **Frontend**: Use **Bootstrap 5** for a responsive UI with navigation, forms, and tables. Include an e-commerce website for customers and a dashboard for admins/staff.
- **Backend**: Use **Spring Boot** with **Spring Data JPA** for CRUD operations, **Spring Security** for role-based authentication (admin, customer, delivery), and **Spring REST** for API endpoints.
- **Database**: Use **H2** embedded database (in-memory mode) to store data locally for offline functionality, following the ERD provided (28 entities, including `User`, `Role`, `Product`, `ProductBatch`, `Order`, `OrderItem`, `Customer`, `Delivery`, etc.).
- **Offline Functionality**: Implement **Service Workers** to cache static assets (HTML, CSS, JS, Bootstrap) and **IndexedDB** to store customer orders, cart items, and product data locally for offline browsing and order placement.
- **Features**:
    - **E-commerce Website**: Allows customers to browse products, add to cart, place orders (with line totals and grand total), and request scheduled deliveries (e.g., 30 string hoppers). Supports loyalty card features (e.g., discounts, delivery preferences).

    - **Dashboard**: For admins/staff, includes:
        - Inventory management (view/edit stock, reorder requests).
        - Order management (view, update status, assign deliveries).
        - Delivery routing (assign orders to drivers/routes).
        - Analytics (sales, inventory, delivery reports).
        - Reorder notifications (alerts for low stock, manager approval).
    - **Authentication**: Role-based access (admin: full control; customer: order/delivery requests; delivery: view assigned deliveries).
    - **Credit Sales**: Track credit payments and customer credit limits.
    - **Loyalty Program**: Manage loyalty card details and scheduled delivery requests.
- **Offline Constraints**: The system must work without internet, caching all necessary assets and syncing data to the H2 database when online (for demo purposes). Use IndexedDB for client-side persistence of cart and order data.

File Structure:

```
sampath-grocery-store
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com/sampathgrocery
│   │   │       ├── config
│   │   │       │   ├── SecurityConfig.java        # Spring Security
configuration
│   │   │       │   ├── WebConfig.java            # CORS and web
configurations
│   │   │       │   └── DatabaseConfig.java       # H2 database setup
│   │   │       ├── controller
│   │   │       │   ├── AuthController.java       # Login/register
APIs
│   │   │       │   ├── ProductController.java    # Product and batch
management
│   │   │       │   ├── OrderController.java      # Order and cart
management
│   │   │       │   ├── DeliveryController.java   # Delivery and route
management
│   │   │       │   └── AnalyticsController.java  # Reports and
analytics
```

```
│   │   │       ├── entity
│   │   │       │   ├── User.java                # Entity classes per
ERD
│   │   │       │   ├── Role.java
│   │   │       │   ├── Product.java
│   │   │       │   ├── ProductBatch.java
│   │   │       │   ├── Order.java
│   │   │       │   ├── OrderItem.java
│   │   │       │   ├── Customer.java
│   │   │       │   ├── Delivery.java
│   │   │       │   └── ... (other entities per ERD, e.g.,
ReorderRequest, Notification)
│   │   │       ├── repository
│   │   │       │   ├── UserRepository.java       # JPA repositories
│   │   │       │   ├── ProductRepository.java
│   │   │       │   ├── OrderRepository.java
│   │   │       │   ├── DeliveryRepository.java
│   │   │       │   └── ... (other repositories)
│   │   │       ├── service
│   │   │       │   ├── AuthService.java          # Business logic
│   │   │       │   ├── ProductService.java
│   │   │       │   ├── OrderService.java
│   │   │       │   ├── DeliveryService.java
│   │   │       │   └── AnalyticsService.java
│   │   │       └── dto
│   │   │           ├── UserDTO.java              # Data transfer
objects
│   │   │           ├── ProductDTO.java
│   │   │           ├── OrderDTO.java
│   │   │           └── ... (other DTOs)
│   │   ├── resources
│   │       ├── static
│   │       │   ├── css
│   │       │   │   └── app.css                 # Custom styles
│   │       │   ├── js
│   │       │   │   ├── app.js                 # Main JS logic
│   │       │   │   ├── offline.js             # IndexedDB logic
│   │       │   │   └── service-worker.js      # Service Worker for
caching
```

```
│   │       │   └── images
│   │       │       └── logo.png              # Store logo
│   │       ├── templates
│   │       │   ├── index.html                # E-commerce homepage
│   │       │   ├── dashboard.html            # Admin/staff dashboard
│   │       │   ├── login.html                # Login page
│   │       │   ├── products.html             # Product browsing
│   │       │   ├── cart.html                 # Cart view
│   │       │   ├── order.html                # Order confirmation
│   │       │   └── delivery.html             # Delivery tracking
│   │       ├── application.properties        # H2 config
│   │       └── data.sql                     # Initial data for H2
├── pom.xml                                      # Maven dependencies
└── README.md                                    # Project
documentation
```

Dashboard Design:

- **Admin/Staff Dashboard** (dashboard.html):
  - **Layout**: Bootstrap navbar (top) with links to Home, Inventory, Orders, Deliveries, Analytics, and Notifications. Sidebar for quick actions (e.g., Approve Reorders).
  - **Sections**:
    - **Inventory**: Table showing `ProductBatch` (name, batch_number, stock_quantity, expiry_date, reorder_point). Button to create `ReorderRequest`.
    - **Orders**: Table listing `Order` (order_id, customer_id, grand_total, status, delivery_type). Filter by status.
    - **Deliveries**: Table of `Delivery` (order_id, driver_id, route_id, status). Assign drivers/vehicles and view routes.
    - **Analytics**: Charts (sales trends, low stock alerts) using Chart.js (cached for offline).
    - **Notifications**: List `Notification` (message, type, status) for reorder approvals and delivery updates.
  - **UI Components**: Bootstrap cards, modals for forms (e.g., add product, approve reorder), and alerts for notifications.
- **Customer E-commerce Website** (index.html, products.html, cart.html, order.html):

    - **Homepage**: Bootstrap carousel for featured products, search bar, and product categories.
    - **Product Page**: Grid of products (name, selling_price, image). Add to `Cart` button.
    - **Cart Page**: Table of `Cart` items (product, quantity, line_total, grand_total). Option to request delivery.
    - **Order Confirmation**: Summary of `Order` and `OrderItem` with delivery details.
    - **Loyalty Features**: Form to submit `DeliveryRequest` for frequent items (e.g., 30 string hoppers, delivery_date).
    - **Offline Support**: Cache product data in IndexedDB, allow cart updates and order submission to local storage, sync when online.

Implementation Details:

- **Frontend**:
    - Use Bootstrap 5 CDN (cached via Service Worker) for styling.
    - Implement Service Worker (`service-worker.js`) to cache static assets.
    - Use IndexedDB (`offline.js`) to store product catalog, cart, and orders locally.
    - Example offline flow: Cache products on first load; allow browsing/add to cart offline; store orders in IndexedDB.
    - Use Chart.js for analytics (cached for offline).
- **Backend**:
    - Configure Spring Boot with H2 in-memory database (`application.properties`).
    - Implement REST APIs for all entities (e.g., `/api/products`, `/api/orders`, `/api/delivery`).
    - Use Spring Security for JWT-based authentication with roles.
    - Implement business logic in services (e.g., `OrderService` calculates grand_total, `InventoryService` triggers `ReorderRequest` when stock < reorder_point).
- **Database**:
    - Define H2 schema based on ERD (28 entities).
    - Initialize data in `data.sql` (e.g., sample products, categories, users).
- **Offline Sync**:
    - Store API responses in IndexedDB for offline access.
    - Queue POST requests (e.g., new orders) in IndexedDB and sync when online.

- **Loyalty and Delivery**:
  - Allow loyalty customers to submit `DeliveryRequest` via form, stored in IndexedDB offline.
  - Assign `Delivery` to `DeliveryRoute` based on `Customer.preferred_delivery_area`.
- **Reorder Notifications**:
  - Trigger `ReorderRequest` when `Inventory.quantity < reorder_point`.
  - Send `Notification` to manager for approval.
  - Update `PurchaseOrder` upon approval.

Sample API Endpoints:

- `GET /api/products`: List products (cached in IndexedDB).
- `POST /api/cart`: Add to cart (store in IndexedDB offline).
- `POST /api/orders`: Place order (queue in IndexedDB offline).
- `GET /api/inventory`: View stock (admin only).
- `POST /api/reorder/approve`: Approve reorder (admin only).
- `GET /api/delivery/routes`: View routes (delivery role).

Dependencies (pom.xml):

- Spring Boot Starter Web, Data JPA, Security
- H2 Database
- JJWT for authentication
- Lombok for boilerplate reduction

Development Guidelines:

- Ensure static assets are cached for offline use.
- Test offline by disabling network in browser dev tools.
- Use Bootstrap classes for responsive design (e.g., `container`, `row`, `col-md-6`).
- Keep API responses lightweight for IndexedDB.
- Document in `README.md` (e.g., `mvn spring-boot:run`).

Deliverables:

- Fully functional standalone system on `localhost:8080`.
- Responsive e-commerce website and admin dashboard.
- Offline support for browsing, cart, and order placement.

- H2 database with preloaded data for demo.
- Clear file structure as outlined.
- Sample data for products (e.g., rice, string hoppers), customers, and orders.

Notes

- This development prompt removes all production-related features, entities, controllers, and services, focusing on inventory, orders, delivery, loyalty, and reorder processes.
- **Database Choice**: Used H2 embedded for true standalone/offline operation in an interview (no external MySQL server needed). If you prefer MySQL, configure it as embedded or local, but H2 is recommended for simplicity.
- **Dashboard and File Structure**: Decided on a user-friendly, Bootstrap-based dashboard with essential sections; file structure is modular for easy navigation.
- **Offline E-commerce**: Fully implemented via Service Workers and IndexedDB, allowing the website to work without internet (e.g., browse products, add to cart, queue orders).

Would you like me to:

- Generate the SQL schema for the 28 entities?
- Provide a sample workflow for a loyalty customer ordering 30 string hoppers?
- Suggest refinements to the development prompt for specific tools (e.g., IDE setup)?
- Create sample code for a specific file (e.g., `dashboard.html`)?

Business Flow for Inventory and Order Management System in Grocery Stores

Based on the requirements for the Web-Based Inventory and Order Management System for Sampath Grocery Store, I'll outline the typical **business flow** (also known as workflow or process flow) for such a system. This flow integrates retail/wholesale operations, e-commerce, home delivery, loyalty features, and automatic reorder notifications. The flow is designed to optimize efficiency, reduce waste, and enhance customer satisfaction in a

grocery context, where perishable items like bread or string hoppers require careful handling.

The business flow can be broken down into key stages, forming a cyclical process from procurement to sales and replenishment:

1. **Procurement and Supplier Management**:
    a. The process starts with identifying needs based on inventory levels. When stock for a product batch (e.g., rice or string hoppers) falls below the reorder point in the `Inventory` entity, the system generates a `ReorderRequest` and sends a `Notification` to the manager for approval.
    b. Upon approval, a `PurchaseOrder` is created and sent to the `Supplier`. This includes details like quantity, expected delivery date, and batch specifics.
    c. Real-world tip: Forecasting tools predict demand using historical sales data from `Analytics` to avoid overstocking perishables.
2. **Receiving and Inventory Update**:
    a. When goods arrive, staff scan barcodes (via `Product.barcode_number` and `ProductBatch.batch_number`) to update `Inventory.quantity` and `ProductBatch.stock_quantity`.
    b. Expiry dates are recorded to track perishables, triggering `StockAlert` if thresholds are met.
    c. This stage ensures accurate tracking of location and value, preventing discrepancies.
3. **Storage and Stock Monitoring**:
    a. Items are stored, with the system monitoring levels in real-time. For frequent delivery items (e.g., string hoppers marked with `is_frequent_delivery_item = true`), routes are planned via `DeliveryRoute`.
    b. Automated alerts (`StockAlert`) notify if stock is low, integrating with loyalty features for reserved stock.
4. **Customer Interaction and Order Placement (E-commerce/Retail)**:
    a. Customers (via `User` with role "customer") browse products on the e-commerce site, add to `Cart`, and place an `Order` with `OrderItem` details (quantity, unit_price, line_total, grand_total).
    b. Loyalty members (`Customer.is_loyalty_member = true`) can request scheduled deliveries via `DeliveryRequest` (e.g., 30 string hoppers for tomorrow).

    c. For credit sales, check `Customer.credit_limit` and set `Payment.status = "credit"`.
5. **Order Processing and Fulfillment**:
    a. Orders are assigned a `status_id` (e.g., "pending" from `OrderStatus`). Staff pick items, updating inventory.
    b. For deliveries, create a `Delivery` entry, assigning `Driver`, `Vehicle`, and `DeliveryRoute` based on `Customer.preferred_delivery_area`.
    c. Invoices are generated (`Invoice`), and payments processed via `Payment` and `PaymentMethod`.
6. **Delivery and Customer Feedback**:
    a. Drivers update `Delivery.status` (e.g., "out_for_delivery" to "delivered"). Customers can track via the app.
    b. Post-delivery, collect `Feedback` (rating, comment) linked to `ProductBatch`.
7. **Analytics, Reporting, and Replenishment**:
    a. Generate `Report` from `Analytics` data (e.g., sales trends, low-stock alerts) to inform decisions.
    b. The cycle loops back to procurement with automatic reorders.
    c. Additional features: Handle returns by reversing inventory updates and updating `Order.status`.

This flow ensures seamless operations, with entities like `Notification` for real-time alerts and `Analytics` for data-driven insights. In a standalone offline setup (using H2 database, Service Workers, and IndexedDB), the e-commerce site allows customers to place orders locally, syncing when online.

Real-World Solutions for Application

For real-world implementation of this system, grocery stores face challenges like perishable goods spoilage, demand fluctuations, and supply chain disruptions. Below are practical solutions, drawn from industry best practices and case studies, to make the system robust and scalable:

1. **AI-Powered Demand Forecasting and Automation**:
    a. Use AI to predict demand based on sales history, weather, or holidays, reducing waste by 15-30%. For example, in the Daily Supermarket case study, LEAFIO AI optimized inventory, cutting stockouts by 20% and overstock by 15% through automated replenishment.

    b. Integration: Add AI modules to `Analytics` entity for processing `data` JSON, forecasting reorder points dynamically.
2. **Real-Time Inventory Tracking with IoT/RFID**:
    a. Implement IoT sensors for shelf-level monitoring, ensuring 99% accuracy and reducing manual checks. A retail pharmacy study using Wiliot technology achieved unparalleled accuracy in real-time inventory.
    b. For Sampath: Link IoT data to `Inventory.last_updated` for instant updates, especially for perishables like string hoppers.
3. **Web-Based E-commerce Integration**:
    a. Cloud-based platforms like Oracle Retail or custom Spring Boot systems enable omnichannel (in-store + online) operations. A case study on e-grocery stores used SAP integration for real-time inventory in Hybris, improving order fulfillment.
    b. Offline Support: As in your system, use IndexedDB for local caching, allowing order placement during outages.
4. **Automated Reorder and Supplier Collaboration**:
    a. Systems like IT Retail or TimeForge automate reorders, integrating with suppliers for just-in-time delivery, minimizing holding costs.
    b. Solution: Enhance `ReorderRequest` with API integrations to suppliers, using manager approval workflows.
5. **Scalability and Security for Real-World Deployment**:
    a. For growing stores, use scalable cloud solutions (e.g., AWS-hosted Spring Boot) with role-based access via Spring Security.
    b. Case Study: WAVE Grocery transformed operations for multiple clients, improving efficiency through customized e-commerce and inventory tools.
    c. Address Challenges: Handle data privacy (GDPR compliance for `Customer` data) and integrate POS for in-store sales.
6. **Cost-Effective Custom Development**:
    a. Build on open-source like Bootstrap/Spring Boot, as in your prompt, for low-cost implementation. A web-based inventory system case study showed improved profitability and customer satisfaction in retail.
    b. ROI: Reduces labor by 20-30% through automation, as seen in Comosoft's real-time tracking for grocers.

These solutions make the system practical for real-world use, like in small-to-medium groceries such as Sampath, by minimizing waste, boosting sales, and ensuring reliability even offline. If needed, I can refine this with specific code examples or diagrams.