# Project Documentation

Student Name: Florea Darius-George

Group: Distributed Systems

Title: Inventory Management System with Spring Boot, Oracle, MySQL, PostgreSQL, and React

## Introduction

This documentation describes the implementation of an Inventory Management System that facilitates managing users, products, and orders. It leverages a backend built with Spring Boot and a frontend developed using React. The backend interacts with three distinct databases (Oracle, MySQL, and PostgreSQL) via Docker containers. The system demonstrates CRUD (Create, Read, Update, Delete) operations and showcases relationships between entities like users, products, and orders.

## Technologies Used

### Backend

Spring Boot: A framework for building Java-based applications, especially REST APIs, with minimal configuration. It simplifies backend development by integrating commonly used tools.

WildFly: A lightweight, fast, and modular Java application server, used for deploying Java EE applications. Wildly offers the possibility to deploy various backend applications while also providing tools for monitoring if one chooses to enable them.

Hibernate/JPA: An ORM framework for database interactions, simplifying query execution and data manipulation. JPA provides an interface implemented by Hibernate which allows for easier querying and relational object operations. As the tables were stored in separate databases, illustrating relational entities operations using JPA and Hibernate is not possible.

### Frontend

React: A JavaScript library for building dynamic, component-based user interfaces. React enables efficient updates and rendering of UI components.

### Database Management

Docker: A containerization platform used to run databases (Oracle, MySQL, PostgreSQL) in isolated environments. It ensures portability and ease of setup for the database instances.

To preserve the state of the database in between docker compose down / up volumes were attached to the docker database containers as illustrated in the screenshot below:

A screenshot of a computer

Description automatically generated

## System Architecture

1. Frontend: A React application communicates with the backend via REST APIs, enabling users to create, update, delete, and view entities.

2. Backend: Spring Boot backend exposes endpoints for managing orders, users, and products.

3. Databases:

- Oracle: Manages orders.

- MySQL: Manages products.

- PostgreSQL: Manages users.

4. Docker: Containers host the database environments and execute initialization scripts for database setup.

A screenshot of a computer

Description automatically generated

## Setup Instructions

1. Clone the Repository:

```  
git clone https://github.com/darius-f96/tpjad-db.git  
cd [repository-folder]  
```

2. Run Docker Containers:

Use the provided `docker-compose.yml` file to start the containers:

```  
docker-compose up -d  
```

3. Database Initialization: Each database is initialized with SQL scripts executed during container startup.

To setup the database tables you should do the following: Oracle:

* ```docker exec -it oracle-container resetPassword 'ora123'
* navigate to init folder and run docker cp as following: docker cp initOracle.sql oracle-container:/init.sql
* docker exec -it oracle-container bash -c "sqlplus -s system/ora123@//localhost/FREEPDB1 @/init.sql"
* More about the image here: <https://hub.docker.com/r/gvenzl/oracle-xe>

MySql:

* docker cp initMySql.sql mssql-container:/initMySql.sql
* docker exec -it mssql-container bash -c "/opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -P mSq123!! -C -i initMySql.sql"

Postgres:

* docker cp initPostgres.sql postgres-container:/init.sql
* docker exec -i postgres-container psql -U posadmin -d postgres -f /init.sql

After executing the above commands, the databases should contain the corresponding schemas and allow for a smooth experience of the spring application.

4. Run the Backend: Start the Spring Boot backend using Maven:

There is no need to start the spring application locally as running docker-compose up -d causes all necessary backend infrastructure to start and function correctly. However if one would like to start the spring app locally running the following command starts the application locally:

```  
mvn spring-boot:run  
```

5. Run the Frontend: Navigate to the `frontend` folder and start the React app:

```  
npm install  
npm start  
```

## Entities and Relationships

User: Fields include `userId` (Primary Key) and `name`. Stored in PostgreSQL.

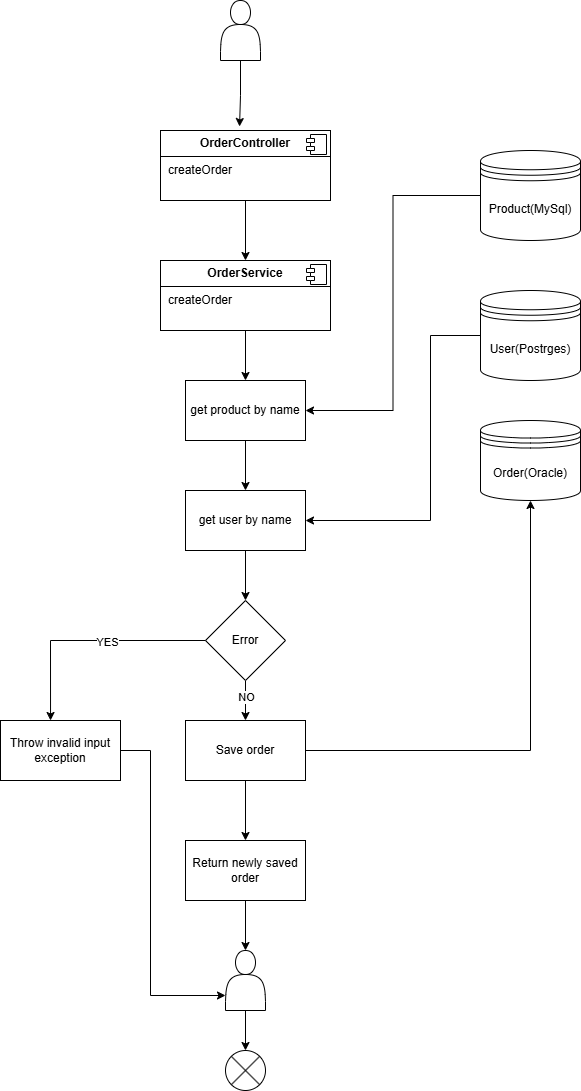
Product: Fields include `productId` (Primary Key), `name`, and `price`. Stored in MySQL.

Order: Fields include `orderId` (Primary Key), `userId` (Foreign Key), `productId` (Foreign Key), and `quantity`. Stored in Oracle.

Relationships: Orders reference users and products, enabling queries like retrieving all orders for a user or product.

No foreign keys could be defined as the tables are stored in different databases on different servers.

Below diagram illustrates the creation of an order:



## Method Prototypes

### Backend

UserController:

- `List<User> getAllUsers()`

- `User createUser(UserRequestDTO userRequestDTO)`

- `User updateUser(Long id, String name)`

- `void deleteUser(Long id)`

ProductController:

- `List<Product> getAllProducts()`

- `Product createProduct(ProductRequestDTO productRequestDTO)`

- `Product updateProduct(Long id, ProductRequestDTO productRequestDTO)`

- `void deleteProduct(Long id)`

OrderController:

- `List<Order> getAllOrders()`

- `Order createOrder(OrderRequestDTO orderRequestDTO)`

- `Order updateOrderQuantity(Long id, int quantity)`

- `void deleteOrder(Long id)`

## Usage Examples

### Create a User

Request:

```  
POST /users  
{  
 "name": "John Doe"  
}  
```

Response:

```  
{  
 "userId": 1,  
 "name": "John Doe"  
}  
```

### Create an Order

Request:

```  
POST /orders  
{  
 "userName": “John Doe”,  
 "productName": “Computer”,  
 "quantity": 3  
}  
```

Response:

```  
{"id":6,"name":null,"quantity":3,"productId":2,"userId":1}

```