Design Document

TrendTrack Design Co.

Date	Jan 5, 2024
Author	Ivet Kalcheva
Status	Completed (Version 3.0)

Version History

Version	Date	Author(s)	Changes	State
0.1	03/10/24	Ivet Kalcheva	Added introduction and architectural restriction	Draft
0.5	04/10/24	Ivet Kalcheva	Added a draft of the C4 model	Draft
0.8	05/10/24	Ivet Kalcheva	Improved C4 model and finalised document	Under Review
1.0	09/10/24	Ivet Kalcheva	Finalised C4 model	Completed (for sprint 2)
1.5	06/11/24	Ivet Kalcheva	Added Domain Model and Database Diagram	Under Review
2.0	06/11/24	Ivet Kalcheva	Added CI/CD pipeline, Docker Containers and SonarQube	Completed (for sprint 3)
3.0	05/01/25	Ivet Kalcheva	Updated CI/CD pipeline, Docker Containers and SonarQube	Completed (for sprint 5)

Distribution

Version	Date	Receivers
1.0	11/10/24	Frank Coenen and Bart Rabeling
2.0	08/11/24	Frank Coenen and Bart Rabeling
3.0	08/01/25	Frank Coenen and Bart Rabeling

Contents

Introduction

Document Purpose

This document provides an overview of the architecture and design of the TrendTrack Inventory System. The system allows clients to browse products and place orders and admins to oversee all operations. The architecture is implemented using **Java Spring Boot** for the backend, **React** for the frontend and **MySQL** as the database.

Architecture

Architecture constraints and design decisions

Spring Boot was preselected as the technical stack, which I believe is a good choice due to its modular design. It allows for the rapid development of REST APIs and smooth management of dependencies through Gradle. By enforcing clear separation between layers (controllers, services and repositories), Spring Boot adheres to the **Single Responsibility Principle**, ensuring each layer handles only its dedicated task. Additionally, its strong support for transactional operations guarantees data consistency, which is critical for maintaining inventory and order management. The **Dependency Inversion Principle** is also present in the way services interact with repositories, abstracting the data access logic to make the system more modular and maintainable.

React, similarly to Spring Boot was preselected as technical stack, due to its efficient rendering via the virtual DOM and its component-based architecture. This design naturally aligns with the **DRY (Don't Repeat Yourself) principle**, as it allows for the creation of reusable components that can be leveraged across multiple pages, reducing code duplication. React's component-based architecture also contributes to maintainability by isolating individual pieces of the UI, keeping future updates simple and efficient. The widespread community support and extensive ecosystem (including libraries for routing, forms, etc.) further enhance the development process, making React an optimal choice for the frontend.

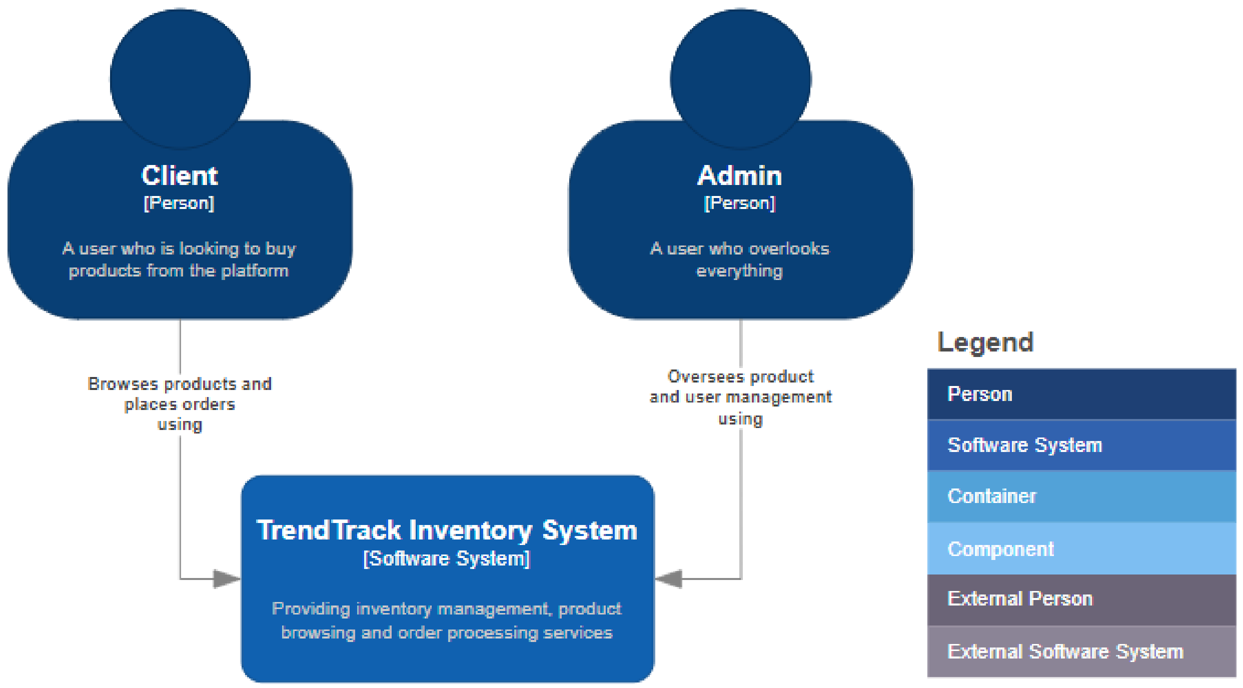
MySQL is used for its balance of cost-efficiency and scalability. Given the system's relational data structure—comprising products, users and orders—MySQL's normalized schema design ensures strong data integrity. By opting for a relational model that suits the current structured data needs, the system adheres to the **YAGNI (You Aren't Gonna Need It) principle**, avoiding the unnecessary complexity of another solution. This approach keeps the system focused on essential requirements while ensuring efficient data handling without overcomplicating the architecture.

To simplify the overall architecture and improve maintainability, I have chosen to implement **Service Classes** in my project instead of **separate Use Case Classes**. This approach allows me to combine multiple functionalities into a single cohesive unit to enhance readability and usability. This strategy mirrors the effective structure I employed last semester, facilitating a more streamlined development process.

C4 Model Overview

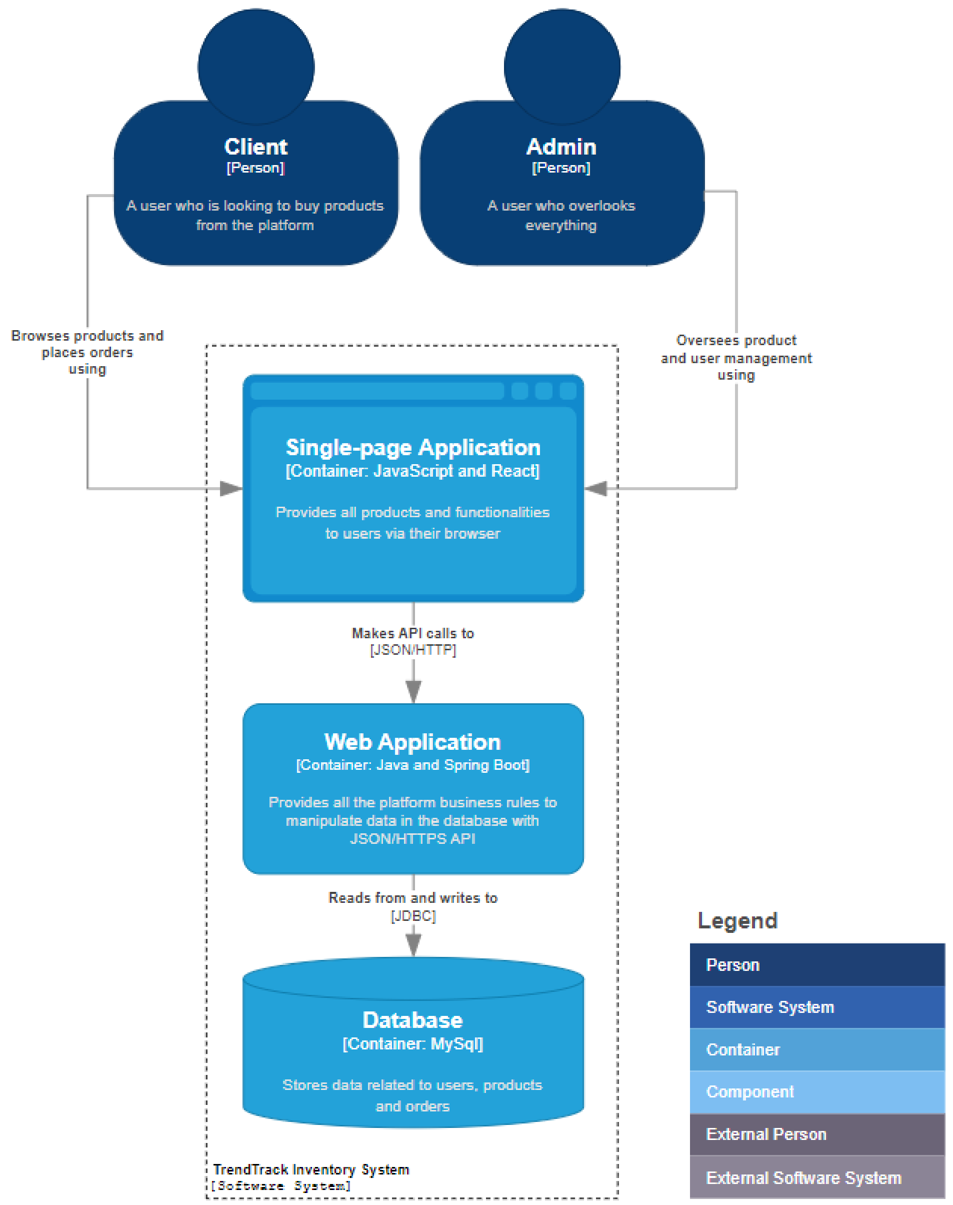
System Context (Level 1)

TrendTrack's Inventory System interacts with two main user roles: **Client** and **Admin**. Each role performs distinct tasks, from browsing and placing orders and overseeing all system (user and product-related) operations.


Client
[Person]
A user who is looking to buy products from the platform
Admin
[Person]
A user who overlooks everything
Browses products and places orders using
Oversees product and user management using
TrendTrack Inventory System
[Software System]
Providing inventory management, product browsing and order processing services
Legend
Person
Software System
Container
Component
External Person
External Software System

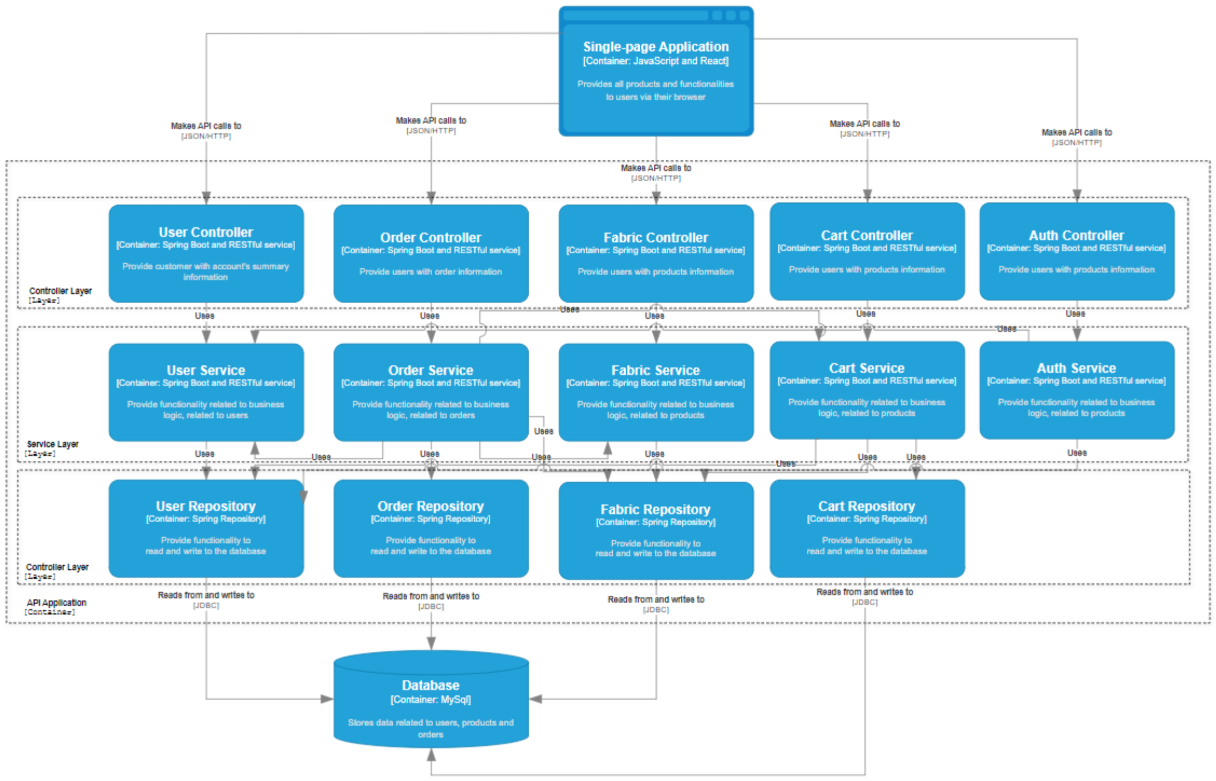
Container Diagram (Level 2)

TrendTrack's Inventory **Software System** is composed of three main containers: a **Single-page Application (SPA)** built with JavaScript and React for user interactions, a **Web Application running** on Java with Spring Boot to handle business logic and a **MySQL Database** to store product, order and user data. The SPA communicates with the Web Application through API calls, while the Web Application manages data persistence with the database.

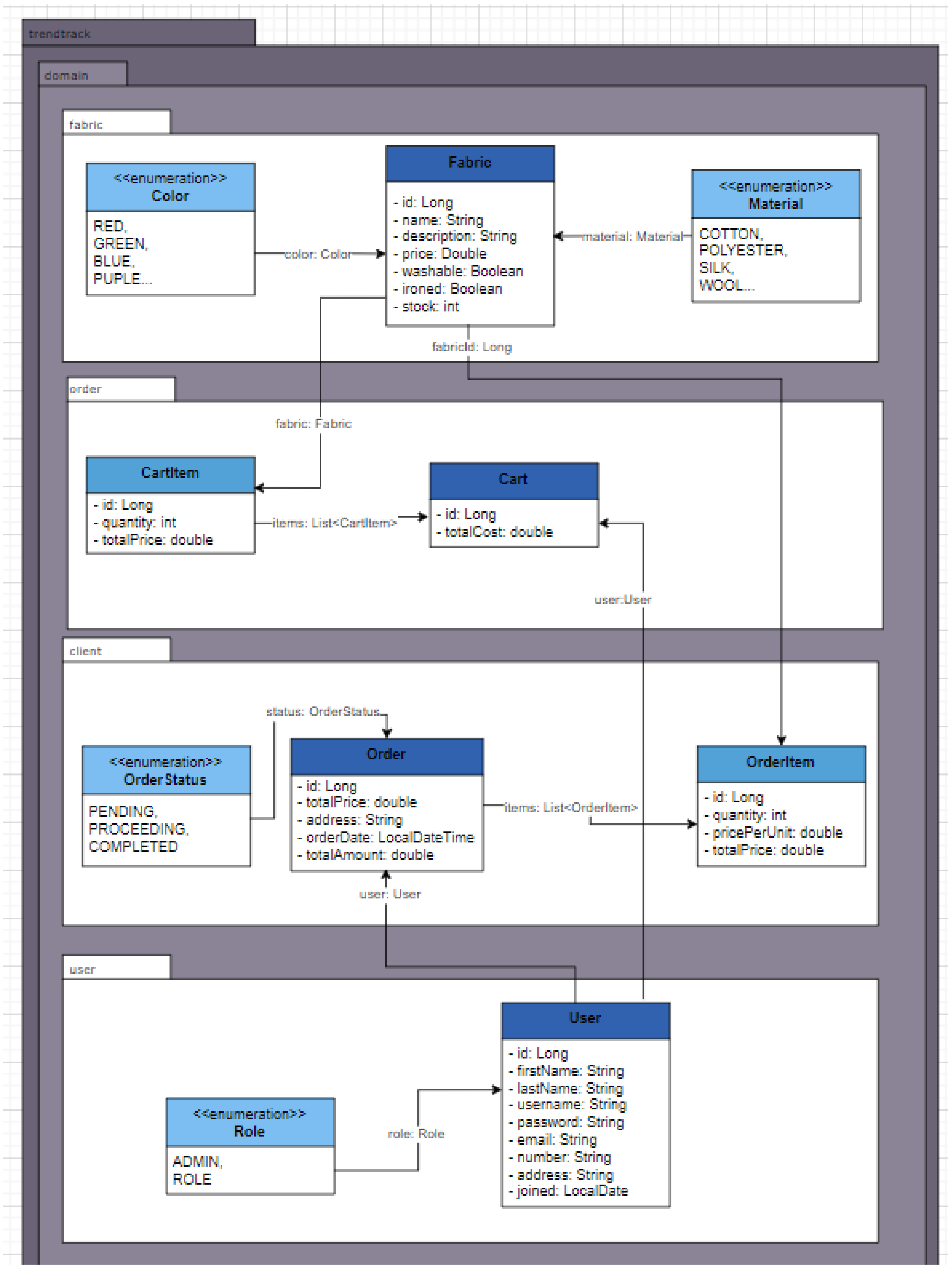

Client
[Person]
A user who is looking to buy products from the platform
Admin
[Person]
A user who overlooks everything
Browses products and places orders using
Oversees product and user management using
Single-page Application
[Container: JavaScript and React]
Provides all products and functionalities to users via their browser
Makes API calls to
[JSON/HTTP]
Web Application
[Container: Java and Spring Boot]
Provides all the platform business rules to manipulate data in the database with JSON/HTTPS API
Reads from and writes to
[JDBC]
Database
[Container: MySql]
Stores data related to users, products and orders
TrendTrack Inventory System
[Software System]
Legend
Person
Software System
Container
Component
External Person
External Software System

Component Diagram (Level 3)

The **API application** of TrendTrack's Inventory System is structured into three layers: **Controllers**, **Services** and **Repositories**. Each layer has specific responsibilities, with Controllers handling user interactions, Services executing business logic and Repositories managing database operations. This design follows the **Single Responsibility Principle** and **Dependency Inversion Principle** to ensure clear responsibilities and minimal dependencies.


Single-page Application
[Container: JavaScript and React]
Provides all products and functionalities to users via their browser
Makes API calls to
[JSON/HTTP]
User Controller
[Container: Spring Boot and RESTful service]
Provide customer with account's summary information
Order Controller
[Container: Spring Boot and RESTful service]
Provide users with order information
Fabric Controller
[Container: Spring Boot and RESTful service]
Provide users with products information
Cart Controller
[Container: Spring Boot and RESTful service]
Provide users with products information
Auth Controller
[Container: Spring Boot and RESTful service]
Provide users with products information
Controller Layer
[Layer]
Uses
User Service
[Container: Spring Boot and RESTful service]
Provide functionality related to business logic, related to users
Order Service
[Container: Spring Boot and RESTful service]
Provide functionality related to business logic, related to orders
Fabric Service
[Container: Spring Boot and RESTful service]
Provide functionality related to business logic, related to products
Cart Service
[Container: Spring Boot and RESTful service]
Provide functionality related to business logic, related to products
Auth Service
[Container: Spring Boot and RESTful service]
Provide functionality related to business logic, related to products
Service Layer
[Layer]
User Repository
[Container: Spring Repository]
Provide functionality to read and write to the database
Order Repository
[Container: Spring Repository]
Provide functionality to read and write to the database
Fabric Repository
[Container: Spring Repository]
Provide functionality to read and write to the database
Cart Repository
[Container: Spring Repository]
Provide functionality to read and write to the database
Controller Layer
[Layer]
API Application
[Container]
Reads from and writes to
[JDBC]
Database
[Container: MySql]
Stores data related to users, products and orders

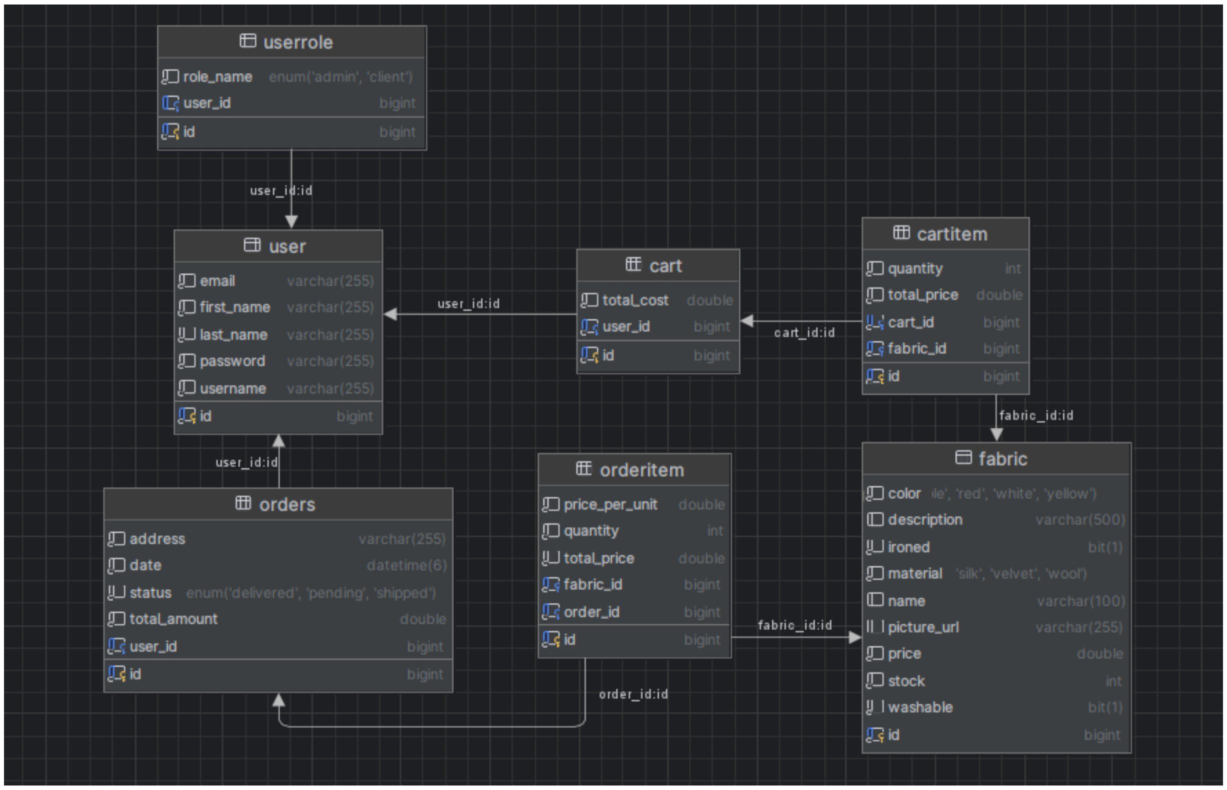
Domain Model



Database Design

The database for the TrendTrack Inventory System consists of 7 main tables, related to: **Fabrics**, **Users**, **Carts** and **Orders**.

This relational structure ensures data integrity and efficient tracking of the entire order process, from browsing fabrics to a cart to order fulfilment. The design is optimized for scalability and future growth, supporting efficient queries and seamless integration with business logic.



userrole	
role_name	enum('admin', 'client')
user_id	bigint
id	bigint

user	
email	varchar(255)
first_name	varchar(255)
last_name	varchar(255)
password	varchar(255)
username	varchar(255)
id	bigint

cart	
total_cost	double
user_id	bigint
id	bigint

cartitem	
quantity	int
total_price	double
cart_id	bigint
fabric_id	bigint
id	bigint

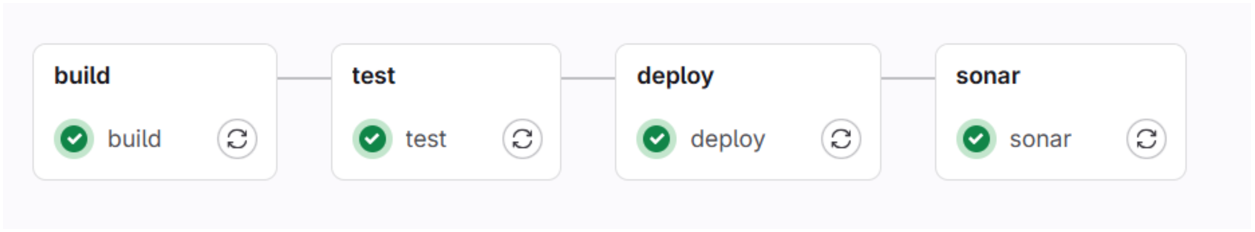
orders	
address	varchar(255)
date	datetime(6)
status	enum('delivered', 'pending', 'shipped')
total_amount	double
user_id	bigint
id	bigint

orderitem	
price_per_unit	double
quantity	int
total_price	double
fabric_id	bigint
order_id	bigint
id	bigint

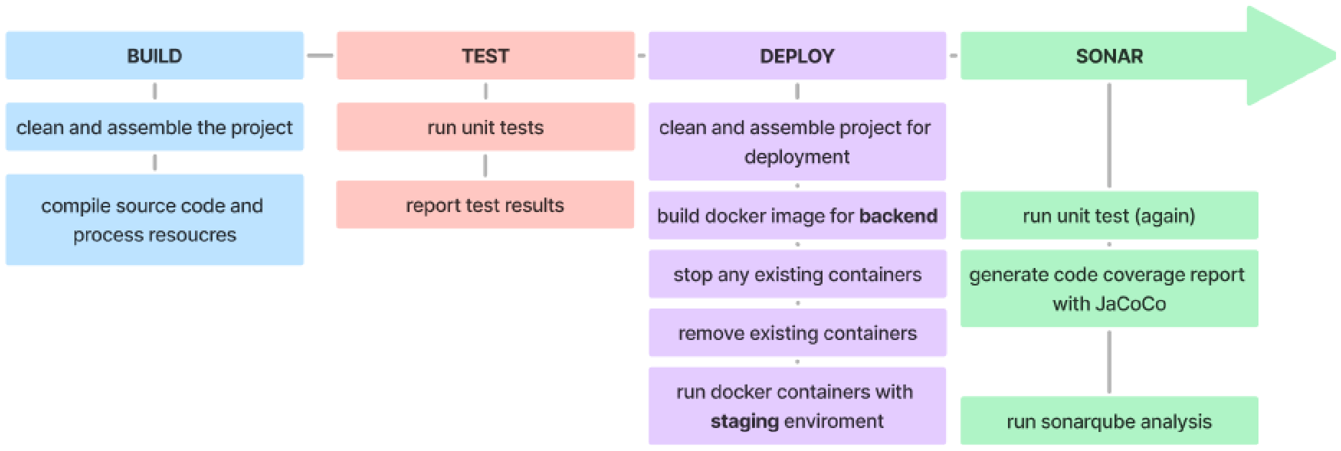
fabric	
color	le', 'red', 'white', 'yellow')
description	varchar(500)
ironed	bit(1)
material	'silk', 'velvet', 'wool')
name	varchar(100)
picture_url	varchar(255)
price	double
stock	int
washable	bit(1)
id	bigint

CI/CD pipeline

The **CI/CD pipeline** for the TrendTrack Inventory System automates the build, test, quality check and deployment processes. The pipeline consists of four stages:

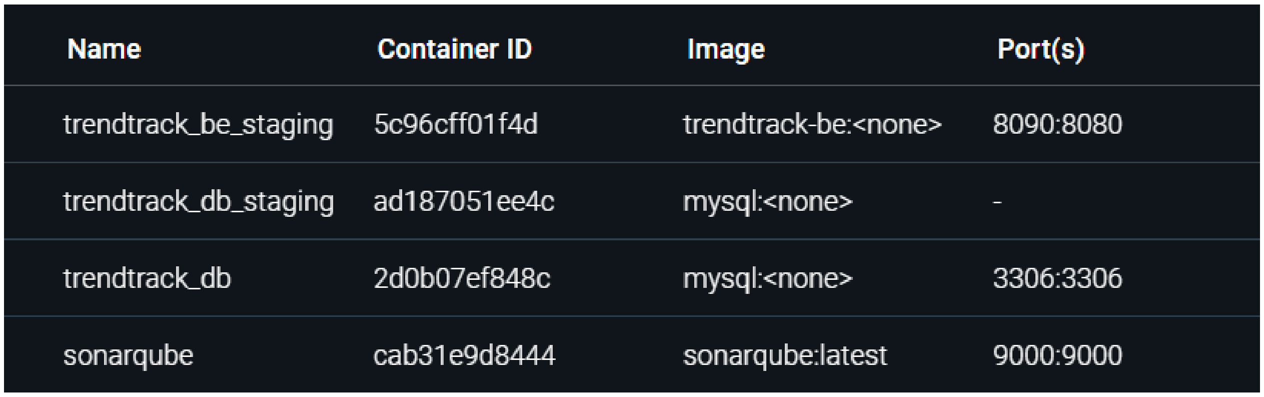


- **Build**: Compiles the application using Gradle and creates a Docker image tagged with the latest version.
- **Test**: Runs unit tests with Gradle to ensure the application functions as expected.
- **Deploy**: Deploys the application in a Docker container, making it accessible on port 8090.
- **Sonar**: Analyses the code quality using SonarQube, generating test coverage and detecting issues.


BUILD
clean and assemble the project
compile source code and process resoucres
TEST
run unit tests
report test results
DEPLOY
clean and assemble project for deployment
build docker image for **backend**
stop any existing containers
remove existing containers
run docker containers with **staging** enviroment
SONAR
run unit test (again)
generate code coverage report with JaCoCo
run sonarqube analysis

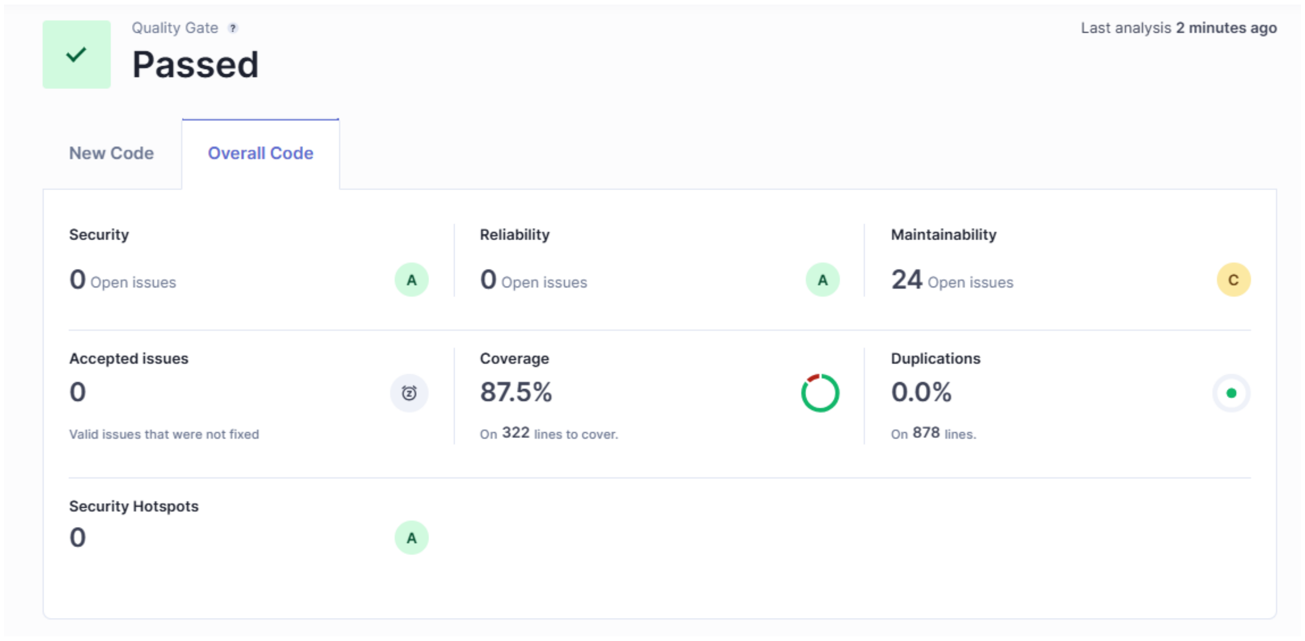
Docker Containers

The TrendTrack Inventory System currently uses 4 **Docker** containers:



Name	Container ID	Image	Port(s)
trendtrack_be_staging	5c96cff01f4d	trendtrack-be:<none>	8090:8080
trendtrack_db_staging	ad187051ee4c	mysql:<none>	-
trendtrack_db	2d0b07ef848c	mysql:<none>	3306:3306
sonarqube	cab31e9d8444	sonarqube:latest	9000:9000

SonarQube



Quality Gate

Passed

Last analysis 2 minutes ago

New Code | Overall Code

Security
0 Open issues A

Reliability
0 Open issues A

Maintainability
24 Open issues C

Accepted issues
0
Valid issues that were not fixed

Coverage
87.5%
On 322 lines to cover.

Duplications
0.0%
On 878 lines.

Security Hotspots
0 A