Veronika Valeva - 4090349

“Wear me” individual project

Software Design Document

Name (s):
Veronika
Valeva

TABLE OF CONTENTS

1.0 Versioning table *3*

2.0 INTRODUCTION *4*

2.1 Purpose *4*

2.2 Scope *4*

2.3 Overview *4*

2.4 Definitions and Acronyms *4*

4.0 SYSTEM ARCHITECTURE DESIGN *4*

4.1 Architectural Design – System context diagram *5*

5.0 COMPONENT ARCHITECTURE DESIGN *5*

5.1 Component Design – Component diagram *6*

6.0 DATA DESIGN *7*

6.1 Data Description *7*

7.0 HUMAN INTERFACE DESIGN *7*

7.1Overview of User Interface *7*

References .... Error! Bookmark not defined.

1.0 Versioning table

VERSION	DATE	AUTHOR	CHANGES	STATE
0.1	30.09.2021	Veronika Valeva	Initial state of the design document.	Sprint 2 deliverable
0.2	30.10.2021	Veronika Valeva	Refactoring of the design point, introduction, design decisions and the UML diagram.	Sprint 3 deliverable
0.3	22.11.2021	Veronika Valeva	Fully changed the structure of the design document. Updated the introduction and its components. Added more context into the system and component architectural design.	Sprint 4 deliverable
0.4	06.01.2022	Veronika Valeva	APA bibliography. Adding CI/CD diagram and context	Sprint 6 delivery

2.0 INTRODUCTION

2.1 Purpose

This software design document describes the architecture and system design of "Wear Me" clothing web shop and its functionalities.

2.2 Scope

"Wear Me" is designed to target users who favor online shopping for clothes. The software is planned to have a main page with all the clothes present on the website, a personal profile for the logged user, a filter for filtering clothes by categories and gender, a shopping cart and a list of favourites. Also, a workspace for community and sales managers. Community managers will have the possibility to provide the online shoppers with more information and answer the users' questions. The sales manager is going to be able to alter the products' data.

2.3 Overview

This document serves as a description of the functionality and software design of the "Wear Me" web shop.

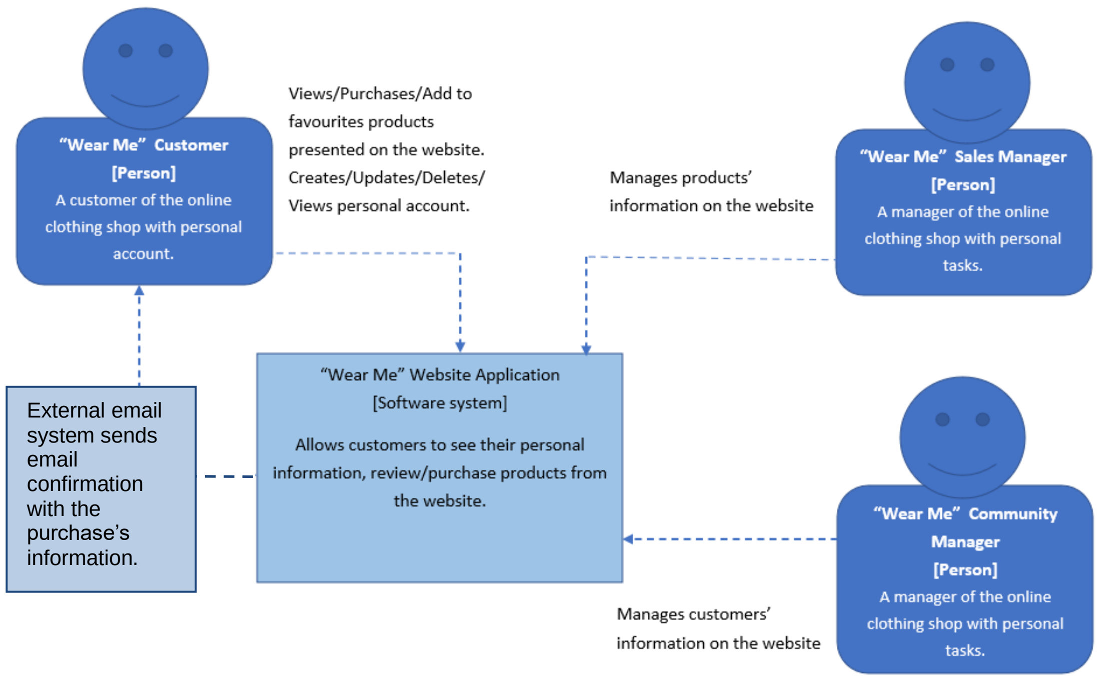
2.4 Definitions and Acronyms

SOLID principle:
S - Single-responsibility Principle. O – Open-closed Principle. L – Liskov Substitution Principle. I – Interface segregation Principle. D- Dependency Inversion Principle
Interface – A programming language keyword used to define a collection of method definitions and constant values. It can later be implemented by classes that define this interface with the "implements" keyword.
Axios - Axios is a library that helps us make http requests to external resources. Axios uses methods like get() and post() that perform http GET and POST requests for retrieving or creating resources.
REST API - A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.
JWT - JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
React - React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces or UI components.

4.0 SYSTEM ARCHITECTURE DESIGN

"Wear Me" consists of a back-end (developed using Java Spring boot) and front-end UI (React). The back end has functionalities for managing CRUD operations regarding different kind of users (customers and managers) and products. The "Wear Me" application is developed by following SOLID principles, ensuring that security is provided to the users. It uses interfaces to connect to the database layer, ensuring that there is a proper implementation. The back end of the application is connected with the front-end using HTTP requests with Axios to connect to REST endpoint, allowing the performance of CRUD operations. An SQL Database is set to run on a Docker container. It holds the information for the users, products, payments methods and orders. SQL is widely used in various applications that support transactional and analytical workloads. Docker, on the other hand, is a containerization technology using which you can bundle your applications within a container and distribute them. Docker helps users to build applications independent of the underlying operating system. The security implementation is done using the HTTP security class in Java. It consists of authentication and authorization. When registering a user his/her password is encrypted and saved, later when the user logs in, the passed password is encrypted again and used to locate the user's account. A custom authentication filter ensures a proper JWT is given to the right user, once the user is found in the database, thus creating a security layer. Once the user is logged in, each subsequent request will include the JWT, thus authorizing the user, allowing the user to access routes, services, and resources that are permitted with that token.

4.1 Architectural Design – System context diagram


"Wear Me" Customer
[Person]
A customer of the online clothing shop with personal account.
Views/Purchases/Add to favourites products presented on the website. Creates/Updates/Deletes/ Views personal account.
Manages products' information on the website
"Wear Me" Sales Manager
[Person]
A manager of the online clothing shop with personal tasks.
External email system sends email confirmation with the purchase's information.
"Wear Me" Website Application
[Software system]
Allows customers to see their personal information, review/purchase products from the website.
"Wear Me" Community Manager
[Person]
A manager of the online clothing shop with personal tasks.
Manages customers' information on the website

5.0 COMPONENT ARCHITECTURE DESIGN

- **Controllers**
  The backend has several classes annotated with @RestController, meaning they are ready for use by Spring MVC to handle web requests. Those classes take the requests sent from the React front end, they expose the application functionalities as RESTful web services. They contain methods that receive POST/GET/DELETE/PUT requests, translates them and sends them to the service classes to perform the requested action, and returns the requested data, if any.

- **Services**
  The service classes represent the service layer of the back end of the application. They are

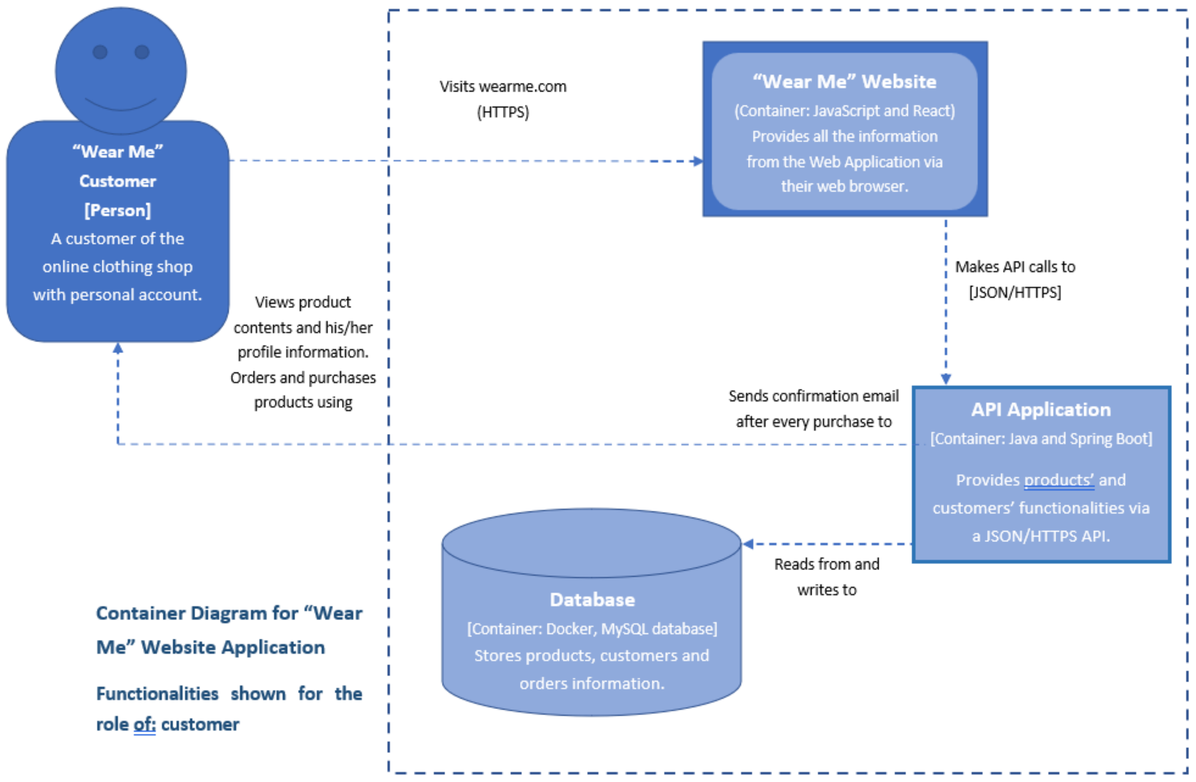
responsible for encapsulating the business logic implementation, centralization of the data access and defining where the transactions begin and end. They implement a custom service interface. Each one of the logic classes (User, Products, ShoppingCartItem and FavouriteItem) has its own service class. The service classes communicate with the repository interfaces.

- **Repositories**
  The repository (persistence layer, or adapter) abstracts persistence operations: find (by id or other criteria), save (create, update) and delete records. The backend of the application uses dependency injection to autowire relationships between the action classes and the test classes.

- **Configuration**
  The configuration class extends the abstract class WebSecurityConfigurerAdapter and extends the customization of the security of the application. The class configurates which URL paths should be secured/ authenticated and which should not. When a user successfully logs in, they are redirected to the previously requested page that required authentication. (For example, when a user tries to access his profile without being logged into his profile, he/she is redirected to the log in functionality and only after being authenticated can he/she access his/her profile).

- **Filters**
  The application uses two filter classes. The class CustomAuthenticationFilter extends the class UsernamePasswordAuthenticationFilter. The class takes two parameters to this filter: a username and password. After that it authenticates the user and return access and refresh tokens, containing the user’s username and role (customer, community or sales manager). The second filter class CustomAuthorisationFilter extends the abstract class OncePerRequestFilter that aims to guarantee a single execution per request dispatch, on any servlet container. The class receives the authentication sent from the front end (from specific url-s) and tries to decrypt the received json web token and authorize the user. The complete design of the back end can be found in the UML diagram.

5.1 Component Design – Component diagram


"Wear Me" Customer [Person] A customer of the online clothing shop with personal account.
Visits wearme.com (HTTPS)
"Wear Me" Website (Container: JavaScript and React) Provides all the information from the Web Application via their web browser.
Makes API calls to [JSON/HTTPS]
Views product contents and his/her profile information. Orders and purchases products using
Sends confirmation email after every purchase to
API Application [Container: Java and Spring Boot] Provides products' and customers' functionalities via a JSON/HTTPS API.
Reads from and writes to
Database [Container: Docker, MySQL database] Stores products, customers and orders information.
Container Diagram for "Wear Me" Website Application
Functionalities shown for the role of: customer

6.0 DATA DESIGN

6.1 Data Description

"Wear Me" web shop uses Spring Data JPA to store and retrieve data in a Docker database. All the entities save in the database are annotated in the backend as a JPA entity. This way every class annotated with @Entity is a separated table in the database and the class' unannotated variables are the database table's columns. Every entity class in the backend has a @Id variable. The id property is also annotated with @GeneratedValue to indicate that the ID should be generated automatically. Interfaces that extend the interface JPARepository with parameters – the entity class (User, Products, ShoppingCartItme or FavouriteItem) and the key value (Id), are the connection between the service classes and the database.

7.0 HUMAN INTERFACE DESIGN

7.1Overview of User Interface

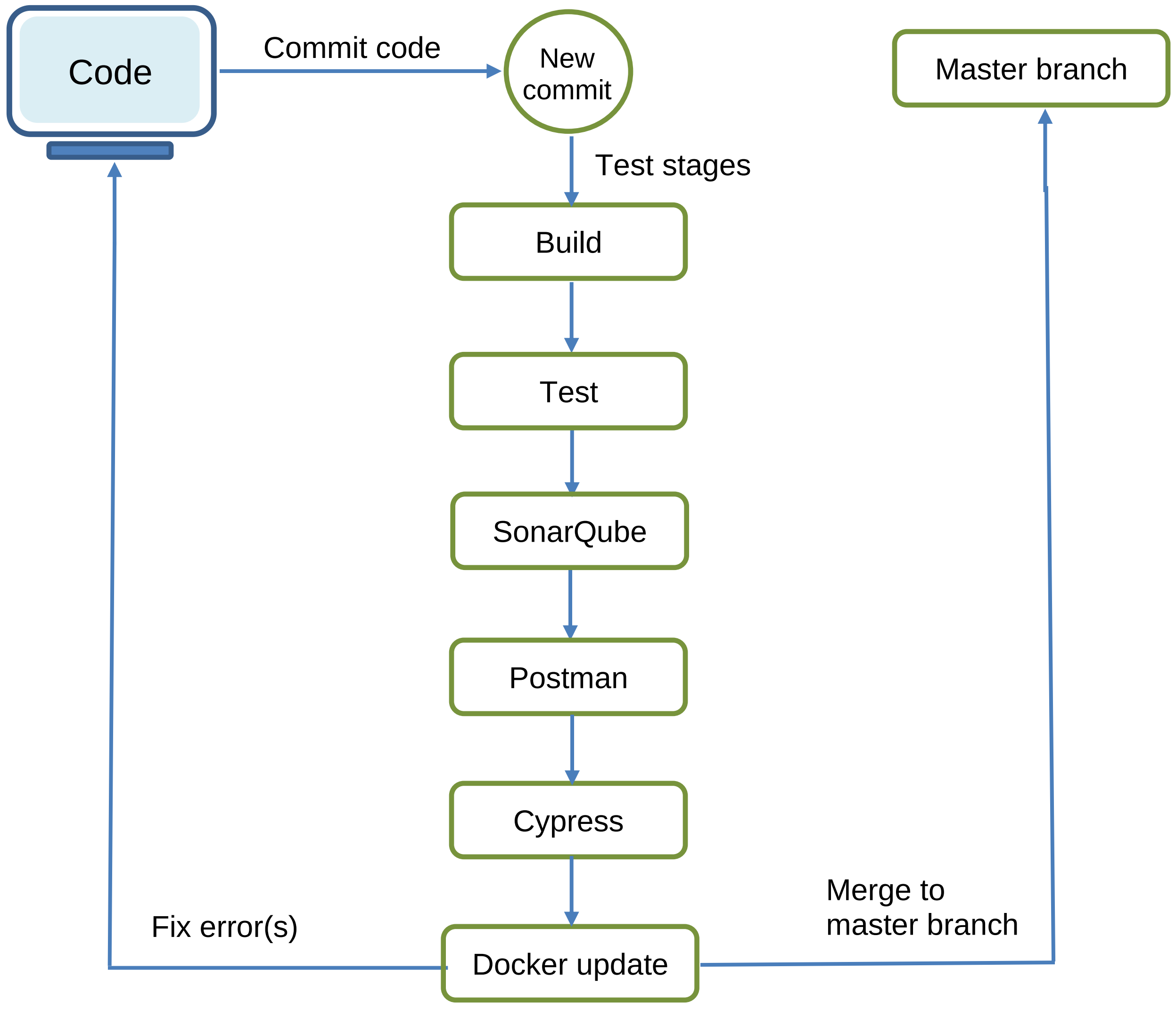
The user interface of the application if fully developed using React, using JavaScript with HTML and CSS for the styling.

The functionalities of the system from the user's perspective are described in the user requirements specification document as user stories. The URS file can be found in the

Documentation folder. And the functionalities of the website are tested with human subjects, the report from the testing is also available in folder Documentation under name User Experience Testing Document. The interface and functionalities of the website is also presented via video material and is also available in folder Documentation under name CompleteUserStories.

8.0 CICD DESIGN

8.1 CI Pipeline Diagram



8.2 Context

After committing the code 6 stages of tests are passed. The first two stages are for building and testing the Gradle project. Then the CI-CD Pipeline is integrated with SonarQube in order to have a clearer view of how testing works and manage to fix errors with greater ease. It also allows for running different statistics and tests on the application/code to find possible data/security leakage and prevent further mistakes. Furthermore, cypress has been added to the frontend part of the application to test end to end components. The entire app is also wrapped in a Docker container that finalizes the whole CI/CD process. At the end of the test stages the whole project is update into docker.

Bibliography

Brown, S. (n.d.). *The C4 model for visualising software architecture*. Retrieved from https://c4model.com/

Brown, S., & Betts, T. (2018, 7 25). *InfoQ*. Retrieved from https://www.infoq.com/articles/C4-architecture-model/

Ferrari, C. (2019, 11 27). *Axios React*. Retrieved from DEV: https://dev.to/cesareferrari/working-with-axios-in-react-540c