



# LB02 – KALYANI

Module 133

Mohammad Nuwera, Khalid Alisha

# Table of Content

---

1	Introduction.....	3
2	Corrections .....	3
3	Work Report.....	3
4	Analysis .....	5
5	Gantt.....	5
6	Design .....	6
6.1	Project Idea .....	6
6.2	Use Cases .....	6
6.2.1	Use Cases.....	6
6.2.2	Activity Diagram .....	13
6.2.3	Overview of functionalities .....	15
6.2.4	Description of functionalities .....	15
6.2.5	ERM .....	16
6.2.6	ERD .....	16
6.2.7	Sequence Diagram.....	17
6.2.8	Structure of application .....	18
6.3	Mock-up.....	18
6.4	Color Scheme .....	20
6.4.1	Color ratio.....	21
7	Implementation.....	21
7.1	Presentation-Layer .....	21
7.1.1	Atomic-Design .....	21
7.1.2	Splitting of components.....	21
7.2	Business-Layer .....	22
7.2.1	Clean-Architecture with Spring .....	22
7.3	Data-Access-Layer .....	22
7.4	Problem solving .....	22
8	Used Technologies.....	23
8.1	3-Tier Structure .....	23
8.2	Database .....	23
8.2.1	Postgre SQL.....	24

8.2.2	DBeaver.....	24
8.3	Backend.....	24
8.3.1	Spring Boot.....	24
8.3.2	Spring Data JPA.....	24
8.3.3	Spring Web.....	24
8.3.4	PostgreSQL.....	24
8.3.5	Gradle .....	24
8.4	Frontend.....	24
8.4.1	Yarn.....	24
8.4.2	Docker-Compose.....	25
8.4.3	Node .....	25
9	Tests Cases Definition .....	25
10	Testing.....	29
10.1	JUnit Testing.....	29
10.1.1	Get all clothing items.....	29
10.1.2	Register a new Customer.....	31
10.2	System Tests.....	32
10.2.1	Log in as User .....	32
11	Retrospective.....	34
11.1	Nuwera .....	34
11.2	Alisha.....	34

# 1 Introduction

This documentation is about our project for the LB02 in the module 133. In this module we learned about web-applications that's why we decided to make an online shop. It is going to be a 3-Tier-Application which consists of a Presentation, Business and Data Layer. We will be using SpringBoot for our backend and React Typescript for the frontend. Further details are explained in the project idea.

## 2 Corrections

In the following table, we will list the corrections we made during the project.

Documentation Version	Date	Correction
1.0	28.06.2022	ERD Relations and Attributes

## 3 Work Report

In the following we will be listing everything we did during our project. In the table you can read who did the task, when, where and what we did.

Who did it?	Date	What did we / I do	Where?	Time in min
Alisha Khalid	31.05.2022	Use Cases	TBZ Oase	180
Nuwera Mohammad	31.05.2022	Use Cases	TBZ Oase	180
Alisha Khalid	07.06.2022	Activity Diagram & ERD	Home	180
Nuwera Mohammad	07.06.2022	Activity Diagram & ERD	Home	180
Alisha Khalid	14.06.2022	Created MockUp	Oase	180
Nuwera Mohammad	14.06.2022	Created MockUp	Oase	180
Alisha Khalid	22.06.2022	Started with Fronted structure	Home	180
Nuwera Mohammad	22.06.2022	Started with Backend structure	Home	180
Alisha Khalid	23.06.2022	Added all compulsory dependencies	Home	100
Nuwera Mohammad	23.06.2022	Added springboot configuration	Home	100
Alisha Khalid	24.06.2022	Homepage with React	Home	100
Nuwera Mohammad	24.06.2022	Database with Hibernate	Home	100

Alisha Khalid	26.06.2022	Created Components and added functionality	Home	240
Nuwera Mohammad	26.06.2022	Endpoints and Connection to Database	Home	240
Alisha Khalid	28.06.2022	Showed 80% finished Project, Created Checkout-, Single Item-, Login Page	Oase	180
Nuwera Mohammad	28.06.2022	Showed 80% finished Project, Finished all endpoints and made minor bugfixes in DB design	Oase	180
Alisha Khalid	30.06.2022	Connected component to backend	Home	100
Nuwera Mohammad	30.06.2022	Added security layer	Home	100
Alisha Khalid	02.07.2022	Added tests, Submitted final project	Home	30
Nuwera Mohammad	02.07.2022	Added tests, Submitted final project	Home	30
Alisha Khalid	03.07.2022	Presented final project	Home	30
Nuwera Mohammad	03.07.2022	Presented final project	Home	30

## 4 Analysis

In the following pages we will talk about our project idea and show you what we prepared and planned for the implementation of the project.

## 5 Gantt

	31.05.2022	07.06.2022	14.06.2022	21.06.2022	28.06.2022	03.07.2022
Documentation						
Build basic structure	*					
Add UML Diagrams and describe them	*	*				
Explain code structure and each layer of our 3-Tier-Application					*	
Write a reflection of project						*
Design						
Create UML-Diagrams	*	*				
Describe project functionalits		*				
Create Mockup			*			
Implementation						
Create project and build a Base structure of code			*			
Homepage and single product page				*		
Cart & Checkout Page				*	*	
Error Handling and validation					*	
Testing						
Write Test cases					*	*

*\*How it really went*

## 6 Design

### 6.1 Project Idea

For this project we both decided to work according the “mvc” design pattern. The idea is to build a website where a user, logged in or just guest, can order clothing products. The products can be viewed individually or the whole collection, depending on the page you are on. By using the mvc design we can layer the application into the three tiers as well as layer the backend into another three layers.

### 6.2 Use Cases

#### 6.2.1 Use Cases

Name	User can see multiple products	
Actor	Logged in User, Guest User	
Trigger	-	
Description	As soon as the User visits the shop, he will be automatically on the home page which is also our multiple product page. There he can view all of them and scroll though the page.	
Pre-conditions	User can be logged in but doesn't have to, he can see the items even as a guest. User has to be on the Home Page.	
SW Components	Docker, HTTP	
	Application	Response
Functional Steps	Pictures of products are loaded	
		Successful / fail
Exception Cases	-	
Post-conditions	User sees multiple items from the online shop	
Span of time	<2s	
Accessibility	User	Application
	Start the application, User is on the multiple product / home page	Docker Connection
Comments, remarks	-	

Name	User can add an item to their cart
------	------------------------------------

<b>Actor</b>	Logged in User, Guest User	
<b>Trigger</b>	The product is added to the shopping cart and the users gets a snackbar message for approvment	
<b>Description</b>	The User can add products to his cart only from the single product page.	
<b>Pre-conditions</b>	User is logged in / guest	
<b>SW Components</b>	Docker, HTTP	
<b>Functional Steps</b>	<b>Application</b>	<b>Response</b>
	Shows all products	
	User clicks on a specific product	
		User gets redirected to the single product page
	User adds the item to the cart	
		Product successfully added to card / could not add because product is sold out
<b>Exception Cases</b>	The product is sold out.	
<b>Post-conditions</b>	The product has been added to the cart successfully.	
<b>Span of time</b>	<2s	
<b>Accessibility</b>	<b>User</b>	<b>Application</b>
	Start of Application, User is on the multiple product page / single product page	Docker Connection
<b>Comments, remarks</b>	-	

<b>Name</b>	<b>Register</b>
<b>Actor</b>	Guest User
<b>Trigger</b>	Account of User is created
<b>Description</b>	When the user visits the page, he has the possibility to register and create an account to profit from various discounts. The User has to fill



	a form with the required information's such as the name, country of residence, email, password etc.	
<b>Pre-conditions</b>	User has to open the form	
<b>SW Components</b>	Docker, HTTP	
	<b>Application</b>	<b>Response</b>
<b>Functional Steps</b>	User opens the registration form	
		User is directed to the registration form
	Enter personal information's	
		Validation of fields → successful or required error
<b>Exception Cases</b>	Account with the email already exists.	
<b>Post-conditions</b>	User is registered successfully; he can log in with the email and password.	
<b>Span of time</b>	<3s	
<b>Accessibility</b>	<b>User</b>	<b>Application</b>
	Start of Application, User is on the registration page	Docker Connection
<b>Comments, remarks</b>	-	

<b>Name</b>	<b>Login</b>	
<b>Actor</b>	Registered User	
<b>Trigger</b>		
<b>Description</b>	If the User is registered, he can log in, in his account. With an account the User doesn't have to reenter his information's for the checkout. It is going to be auto filled.	
<b>Pre-conditions</b>	User is already registered.	
<b>SW Components</b>	Docker, HTTP	
	<b>Application</b>	<b>Response</b>

<b>Functional Steps</b>	User enters username and password	
		Login Status is successful / failed → error message: try again
<b>Exception Cases</b>	User Account doesn't exist.	
<b>Post-conditions</b>	User is logged in.	
<b>Span of time</b>	<3s	
<b>Accessibility</b>	<b>User</b>	<b>Application</b>
	Start Application, choose to log in	Docker connection
<b>Comments, remarks</b>	-	

<b>Name</b>	<b>User can view a single item</b>	
<b>Actor</b>	User (guest or logged in)	
<b>Trigger</b>	A single page with only the selected item is directed to.	
<b>Description</b>	The user is	
<b>Pre-conditions</b>	The user has to be on the main page or know the exact path to the single page	
<b>SW Components</b>	Docker, HTTP	
	<b>Application</b>	<b>Response</b>
<b>Functional Steps</b>	The user lands on the home page and sees an item which he would like to buy and clicks on it	
		The application directs the user to a single page of the certain product
	The user checks out the detailed specification of the product	

<b>Exception Cases</b>	If the user views the single page from the path url then he has to be exact with the url and there is no redirecting to the homepage.	
<b>Post-conditions</b>	The user can view the individual pages of all items.	
<b>Span of time</b>	<2s	
<b>Accessibility</b>	<b>User</b>	<b>Application</b>
	Start the application and suggested to be on the main page	Docker connection
<b>Comments, remarks</b>	The user can only see the specific description of the product if he views the single page. There is only a possibility to add the item to the cart if he is located on a single view page	

<b>Name</b>	<b>User can remove item form cart</b>	
<b>Actor</b>	User (guest, logged in)	
<b>Trigger</b>	The amount of the cart items goes down	
<b>Description</b>	The user which can be both, a guest or logged in, can remove an item from his cart by clicking an icon which serves as a button	
<b>Pre-conditions</b>	The user must have an item already in the cart to be able to remove one.	
<b>SW Components</b>	Docker, HTTP	
<b>Functional Steps</b>	<b>Application</b>	<b>Response</b>
	User clicks remove icon button	
		Users item is removed from personal cart
	User gets notification that item was removed	
<b>Exception Cases</b>	None	
<b>Post-conditions</b>	The user's cart is now an item less	
<b>Span of time</b>	<3s	

<b>Accessibility</b>	<b>User</b>	<b>Application</b>
	Start the application, being on the page of the cart	Docker connection
<b>Comments, remarks</b>	A user can not remove an item by clicking on the single view page	

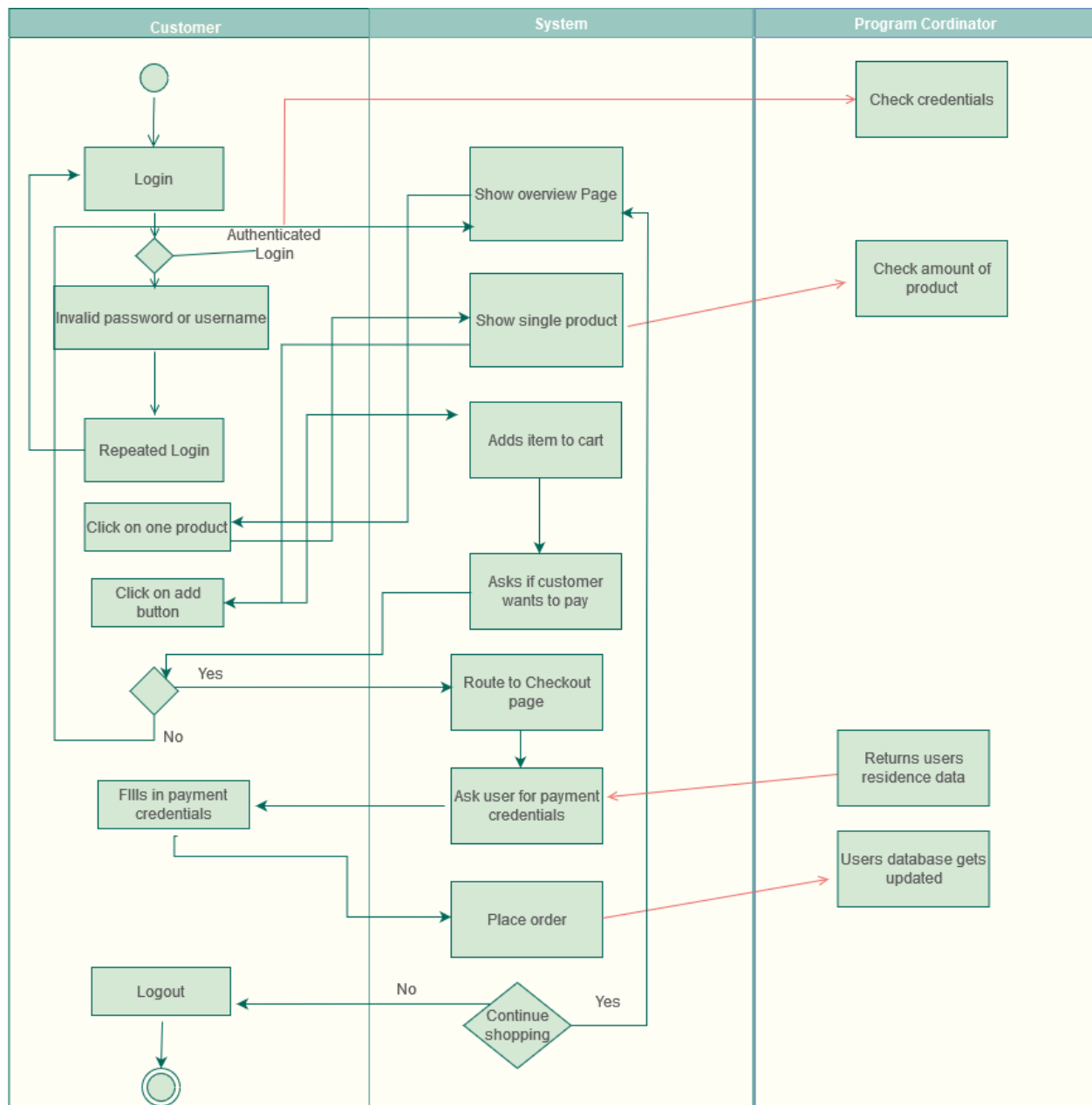
<b>Name</b>	<b>Guest-User can buy items</b>	
<b>Actor</b>	Guest-user	
<b>Trigger</b>	The (guest-)user is forwarded to a checkout page where he can buy the products from his cart	
<b>Description</b>	The user added all his items to his cart and now he wants to buy them. To buy the items he has to fill out a form where he is asked about his name, address, email and confirmation. Since the user is not logged in, he has to type in all the information. After which his confirmation pops up telling him that the items were ordered.	
<b>Pre-conditions</b>	The user needs to have items in his cart to be able to go through checkout.	
<b>SW Components</b>	Docker, HTTP	
	<b>Application</b>	<b>Response</b>
<b>Functional Steps</b>	Guest fills out form and confirms the purchase	
		Application returns a pop up telling him whether the purchase was successful or not
<b>Exception Cases</b>	If the user does not fill out all the compulsory fields the checkout is not complete	
<b>Post-conditions</b>	The checkout either gets completed or the buyer is asked again	
<b>Span of time</b>	<5s	
<b>Accessibility</b>	<b>User</b>	<b>Application</b>
	Start of application, Being on the checkout page	Docker Connection

Comments, remarks	-
-------------------	---

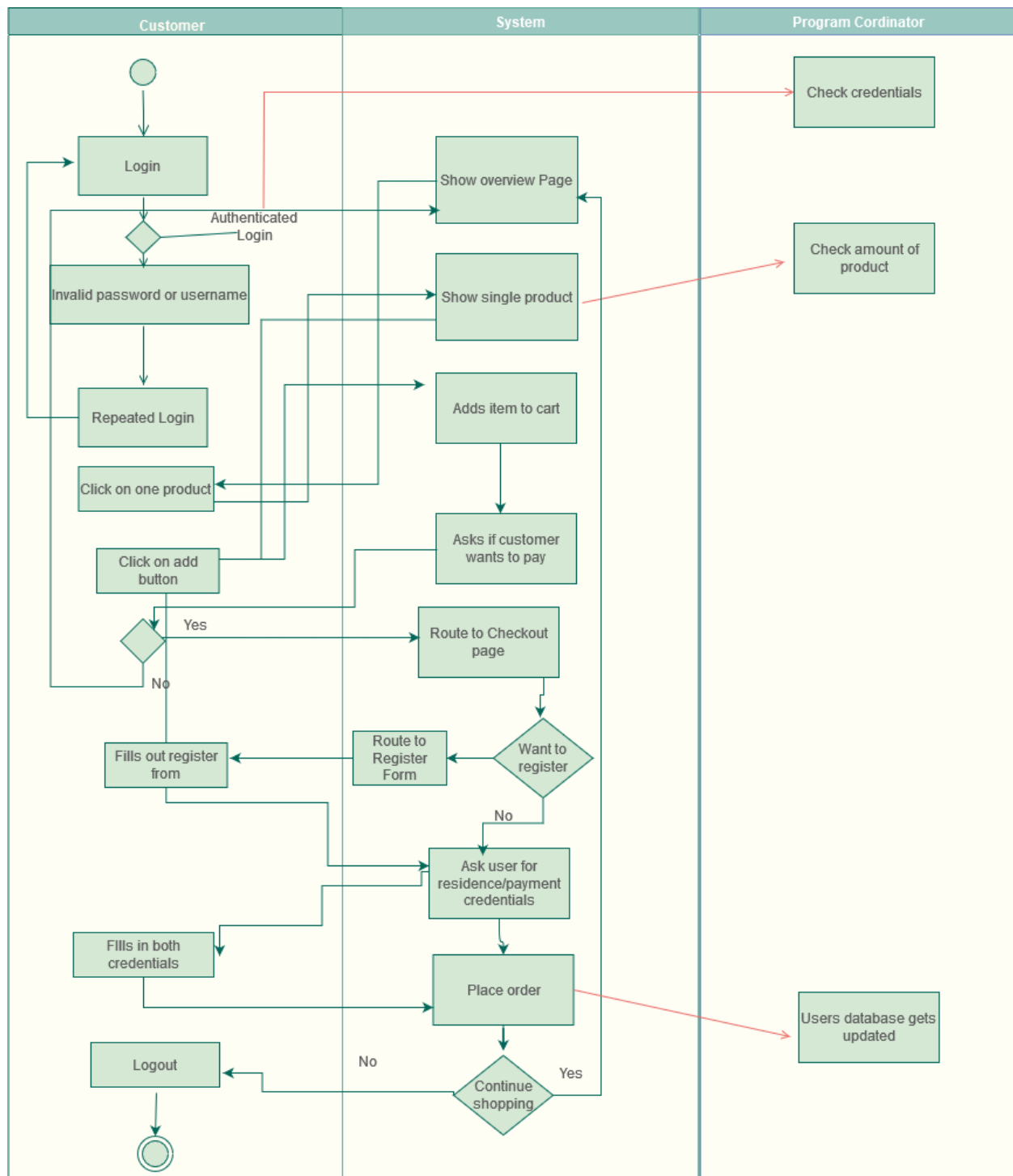
<b>Name</b>	<b>Logged-User can buy items</b>	
<b>Actor</b>	Logged-user	
<b>Trigger</b>	The (logged-)user is forwarded to a checkout page where he can buy the products from his cart	
<b>Description</b>	The user added all his items to his cart and now he wants to buy them. To buy the items he has to fill out a form where he is asked about his name, address, email and confirmation. Since the user is logged in, he does not have to type in all the information, because it is already filled out.	
<b>Pre-conditions</b>	The user needs to have items in his cart to be able to go through checkout.	
<b>SW Components</b>	Docker, HTTP	
	<b>Application</b>	<b>Response</b>
<b>Functional Steps</b>	Customer fills out form and confirms the purchase	
		Application returns a pop up telling him whether the purchase was successful or not
<b>Exception Cases</b>	If the user does not fill out all the compulsory fields the checkout is not complete	
<b>Post-conditions</b>	The checkout either gets completed or the buyer is asked again	
<b>Span of time</b>	<5s	
<b>Accessibility</b>	<b>User</b>	<b>Application</b>
	Start of application, Being on the checkout page	Docker Connection and data of the logged in user
Comments, remarks	-	

### 6.2.2 Activity Diagram

This is our activity diagram, which basically shows the case where a registered user orders a clothing piece.



This is an activity diagram of the case, where the user registers in the application to get a login.



### 6.2.3 Overview of functionalities

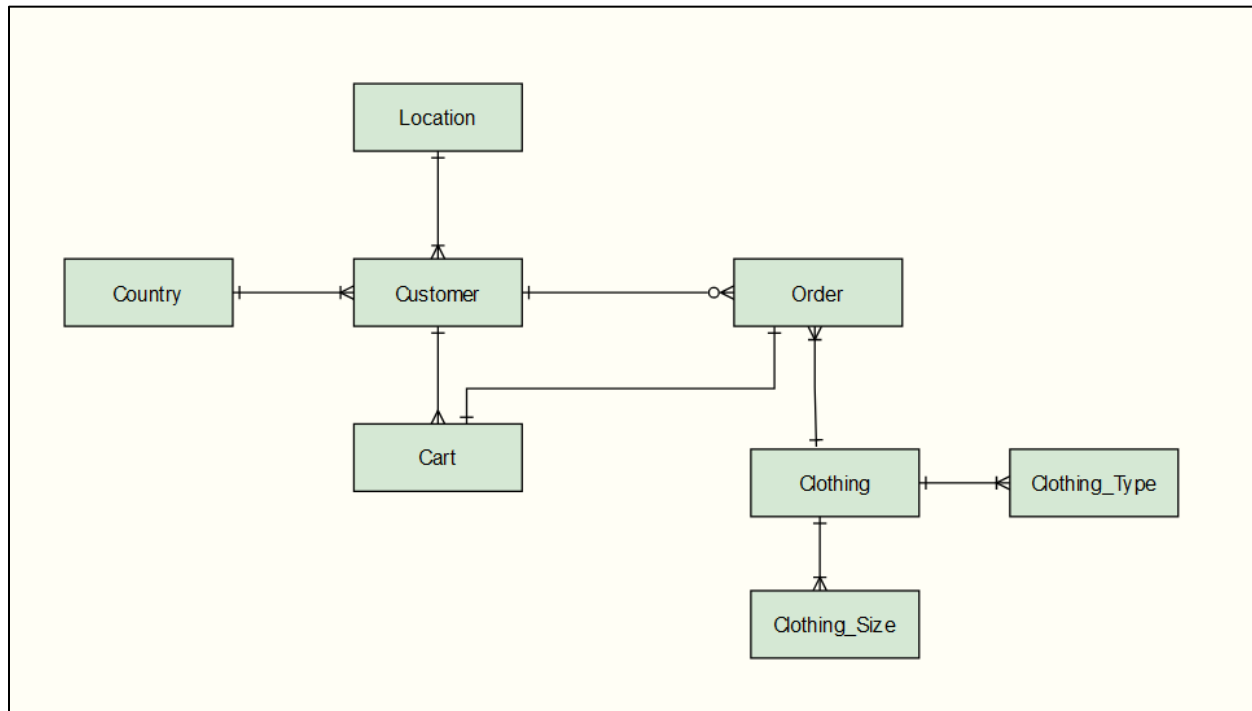
- Register / Login as user
- Buy clothes
- Look at multiple clothes simultaneously
- Have detailed overview of one clothing item
- Logout
- Add clothing item to cart

### 6.2.4 Description of functionalities

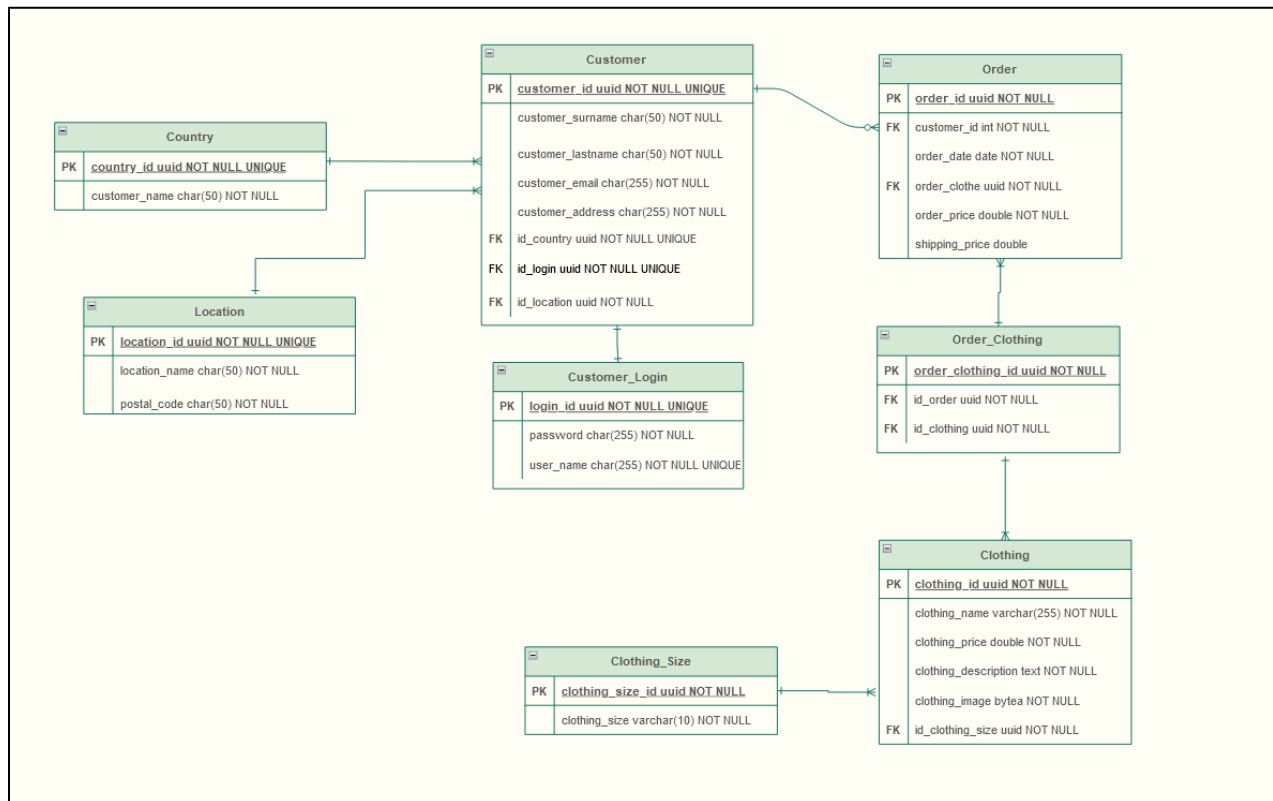
- **Register / Login as user**
  - └ As soon as the User starts the application he has two options, either register to be a client with a following login to profit from advantages or use the application as a guest which requires no information. The advantage of being a client is that the user gets discount on his first purchase, and he doesn't have to enter any data at the payment process except the card information.
- **Buy clothes**
  - └ The main thing about our clothing shop is that the user can buy clothes of his liking. To buy clothes they must be added to the cart in order to complete the payment process.
- **Look at multiple clothes simultaneously**
  - └ To be able to purchase clothes of our web shop it is compulsory to know what you are purchasing. This is why we will create an overview page to display all our clothes.
- **Have detailed overview of one clothing item**
  - └ To be able to add a clothing item into the users own cart he has to visit the single product page. The single product page does not only allow a user to add the item to his cart but also view a detailed description/ multiple images of the product.
- **Logout**
  - └ A login always has a logout option, that means the user can always log out.
- **Add clothing item to cart**
  - └ To buy clothes, they must be added to a cart. The cart is a summary of all the clothes the user would like to buy in the future. That's why every product on the single page will have a cart sign for the user to press and it will automatically be added to the cart.



## 6.2.5 ERM

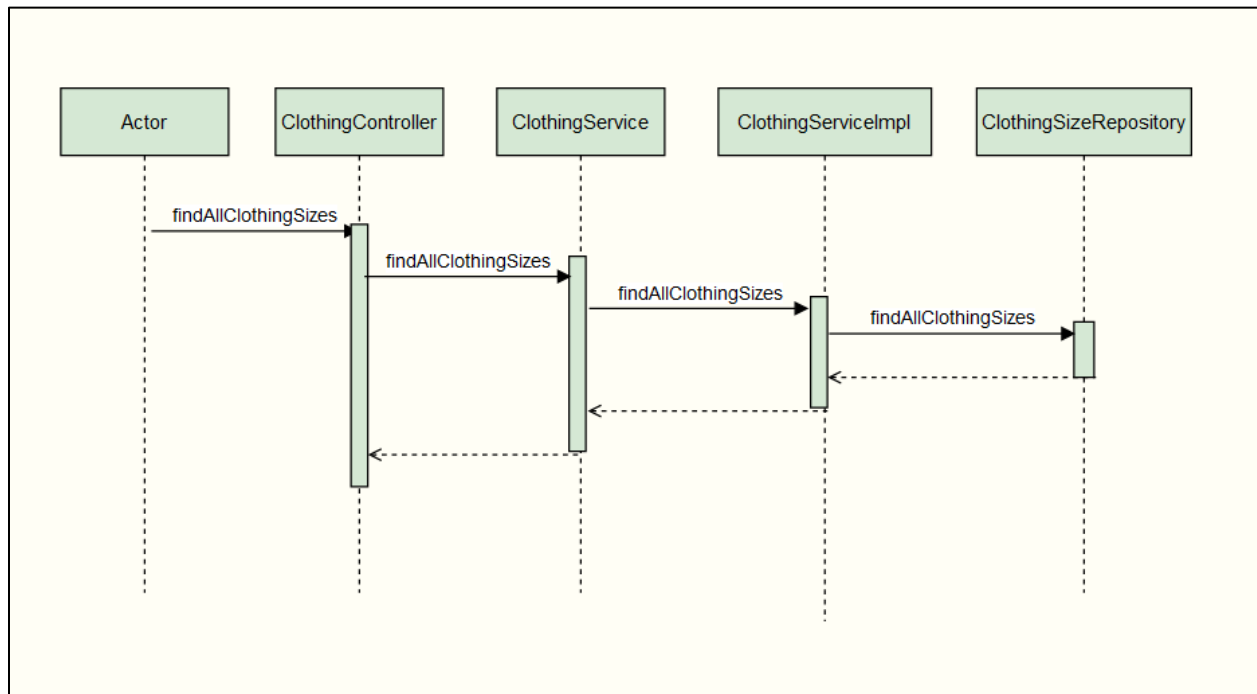


## 6.2.6 ERD

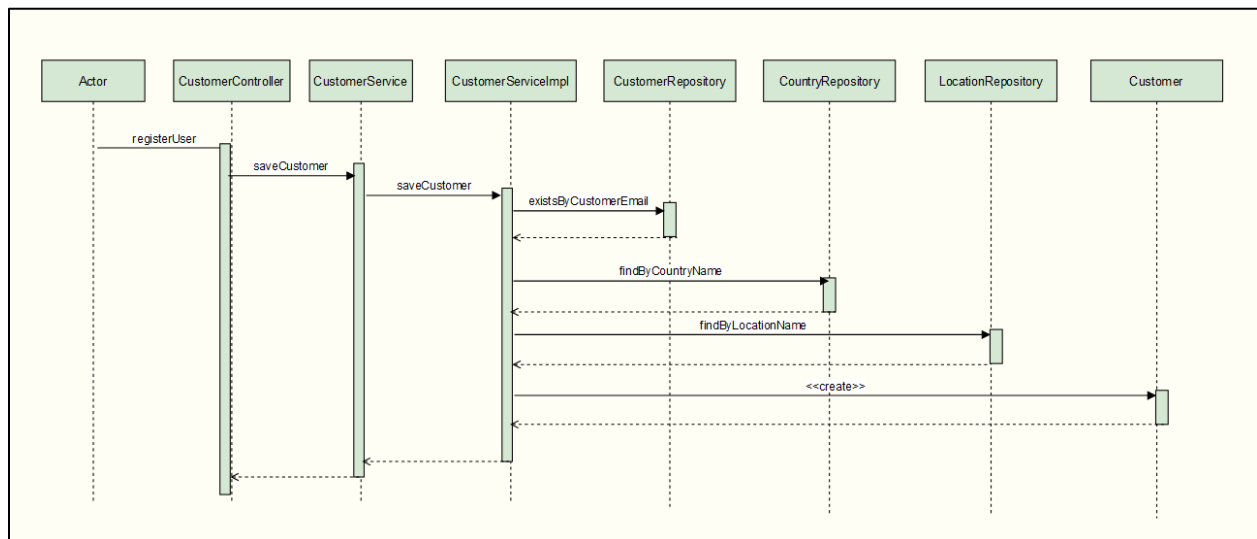


### 6.2.7 Sequence Diagram

Endpoint: Find all Clothing Sizes



Endpoint: Create Customer



### 6.2.8 Structure of application

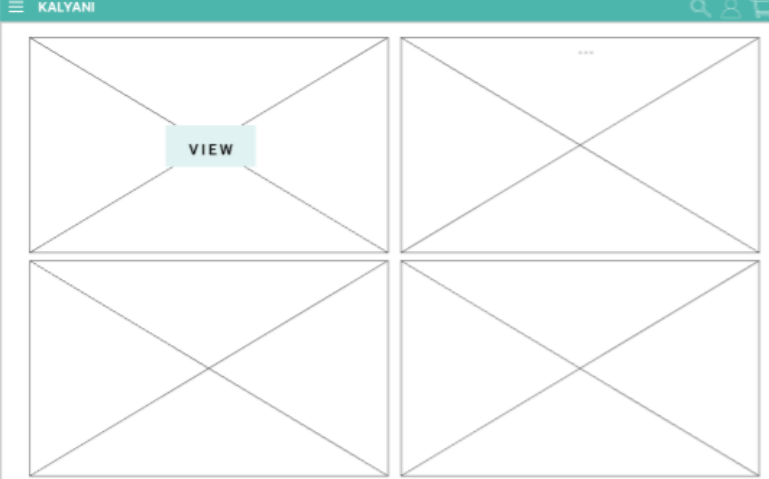
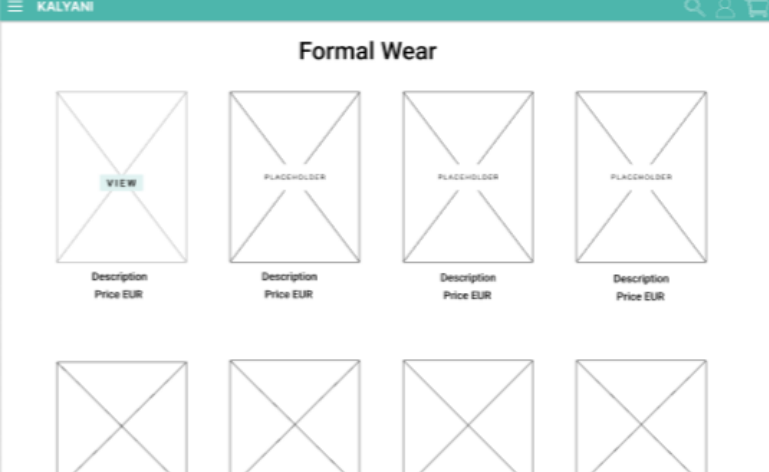
This following application is built on the 3-Tier-Layer architecture. The 3-tiers are the presentation tier, the application tier, and the data tier.


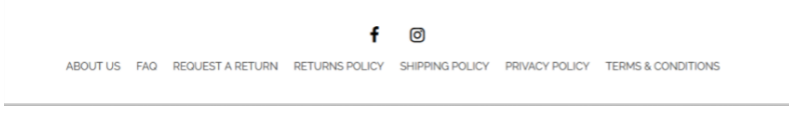
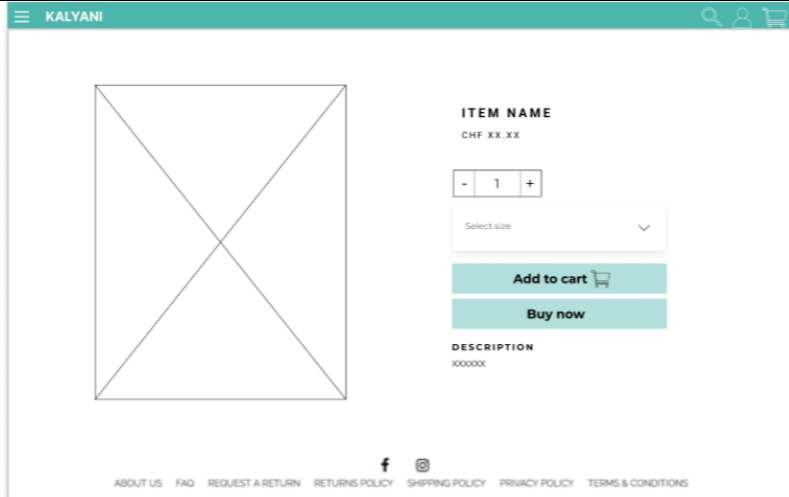
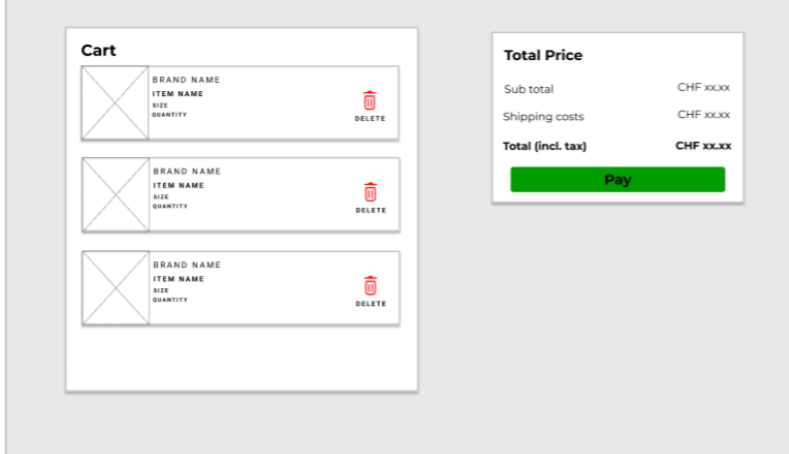
The presentation tier is, like the name says, where the user has an interface with the system. The application tier is where the data is processed and in the data tier the data is stored and effectively managed.



More information on these three layers is found [here](#).

## 6.3 Mock-up

We create a Mock-up to have a better understanding of our goal and design. Together we chose a color scheme to achieve a consistent shop. In the following we will be describing our different pages.

	<p>This is going to be our homepage. It is divided in four parts as each represents one category of our shop. The rectangle is used as placeholder for images. Each image will have a button with the label “View”. If the user clicks on it, he will be navigated to the Multitempage.</p>
	<p>This is the MultitemPage, here we have a Grid design where every clothing piece is displayed as a card with their image, name, and price. If the user clicks on the button, he will be navigated to the SingleItemPage.</p>

	<p>If the user clicks on the hamburger sign, this sidebar will open. It's here for the user to navigate through the different pages we have and see clothes of the preferred category.</p>
	<p>This is the footer of our page, which links to our social media and further sites for additional information's.</p>
	<p>This is the single product page. On the left we have a bigger picture of the clothing and on the right the details. The user can choose the quantity of how many he wants to buy and either add it to the cart or directly pay by pressing on the "buy now" button.</p>
	<p>This is the checkout Page where the user has a summary of all the products he added to the cart, and he even has the possibility to delete item if he chooses to not buy them. On the right side he will see the total price incl. shipping. By pressing on "Pay" the order will be placed.</p>

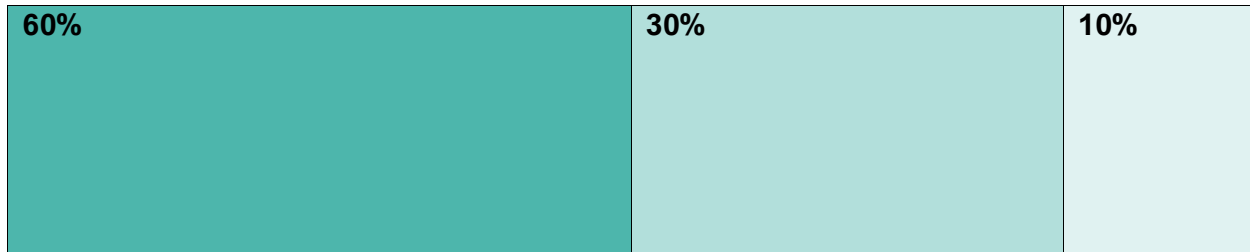
 <p>The screenshot shows a web browser window with a teal header bar containing a menu icon, the text 'KALYANI', and icons for search, user profile, and shopping cart. The main content area is white and features a 'Registration form' section. Inside this section is a teal box titled 'CREATE ACCOUNT'. The form includes input fields for 'SURNAME', 'LASTNAME', 'ADDRESS', 'POSTAL CODE', 'COUNTRY', 'EMAIL', and 'PASSWORD', followed by a 'CREATE' button.</p>	<p>This is the registration form through which the user can create an account by giving his personal Information. Such as name, email address and country where he lives.</p>
 <p>The screenshot shows a web browser window with the same teal header bar. The main content area is white and features a 'Login form' section. Inside this section is a teal box titled 'SIGN IN'. The form includes input fields for 'USERNAME' and 'PASSWORD', followed by a 'LOG IN' button. Below the login button, there is a link for 'NEW TO KALYANI?' and a 'REGISTER HERE' button.</p>	<p>This is the login page, where the user can log in his account. The username is the provided email, and the password is asked to check the authority.</p>

## 6.4 Color Scheme

#4DB6AC	#B2DFDB	#E0F2F1	#E8F3F1	#FFFFFF
---------	---------	---------	---------	---------

### 6.4.1 Color ratio

We used the 60-30-10 Rule for the shop. It describes how a palette should be used. 60% is used as the main color to catch the eye of the user. 30% represent the secondary color, which is lighter than the main color with clear distinction. Finally we have the 10% which adds a certain accent to the whole.



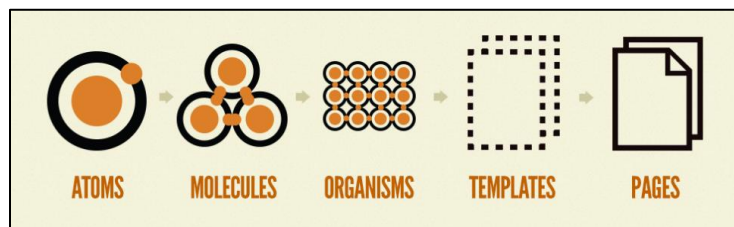
## 7 Implementation

---

### 7.1 Presentation-Layer

#### 7.1.1 Atomic-Design

The architecture we follow in the frontend is [Atomic Design](#).



Quelle: [Atomic Design](#)

In Atomic Design, the smallest components are added together more and more to create groups of these components, called molecules. The grouping is done further and further until you get to the "level" pages and the page takes shape.

#### 7.1.2 Splitting of components

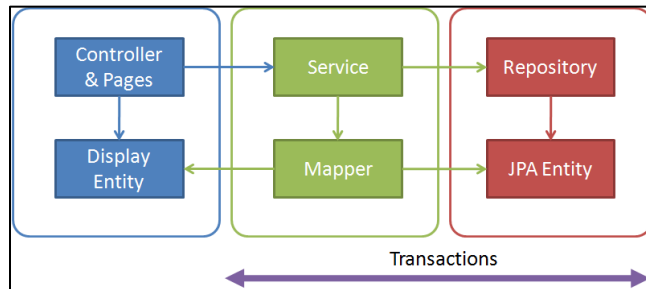
Splitting components is the foundation of Atomic Design. We make sure that a component has only one task and this component is further divided into a "dumb" or "smart" component. This division is relevant to separate functionality from appearance.

An example of a "dumb" component are for example icons, which are only displayed without having logic or state. A "smart" component, on the other hand, must pay attention to the state and change its icon depending on it.

## 7.2 Business-Layer

### 7.2.1 Clean-Architecture with Spring

The backend is built according to the "layered architecture". The three layers are: Data Access Layer, Business Logic Layer and Presentation Layer.



Quelle: [Layer Design](#)

The three layers together are the backend, with the data-access layer communicating with the database and the presentation layer communicating with the front end. This is to make it clear that queries to the database are only made through the JPA entity and the endpoints in the controller are the only access to the frontend. The layer must be separated from each other, which also means that the controller cannot access the repository directly. This has the sense that the scaling of the program is facilitated, as well as the reuse of code makes feasible.

In concrete terms, this means that the project is divided into packages, each representing a layer. These packages are model, controller, service and repository. Each class must be located in the corresponding class. For example, all models must be in the model package and all controllers in the controller package. If necessary, additional packages can be added.

## 7.3 Data-Access-Layer

A data access layer (DAL) in computer software is a layer of a computer program which provides simplified access to data stored in persistent storage of some kind, such as an entity-relational database. For our project we are using a postgresql database to store our data. To have access to various method the repositories extend from the JpaRepository.

## 7.4 Problem solving

Whenever we had a problem, we tried to approach it in different ways.

1. Define what the problem is
2. Check where the problem / error is coming from
3. Define the cause of the problem, why is it throwing this specific error
4. Try to implement a possible solution
5. Write tests to avoid such problems in the future

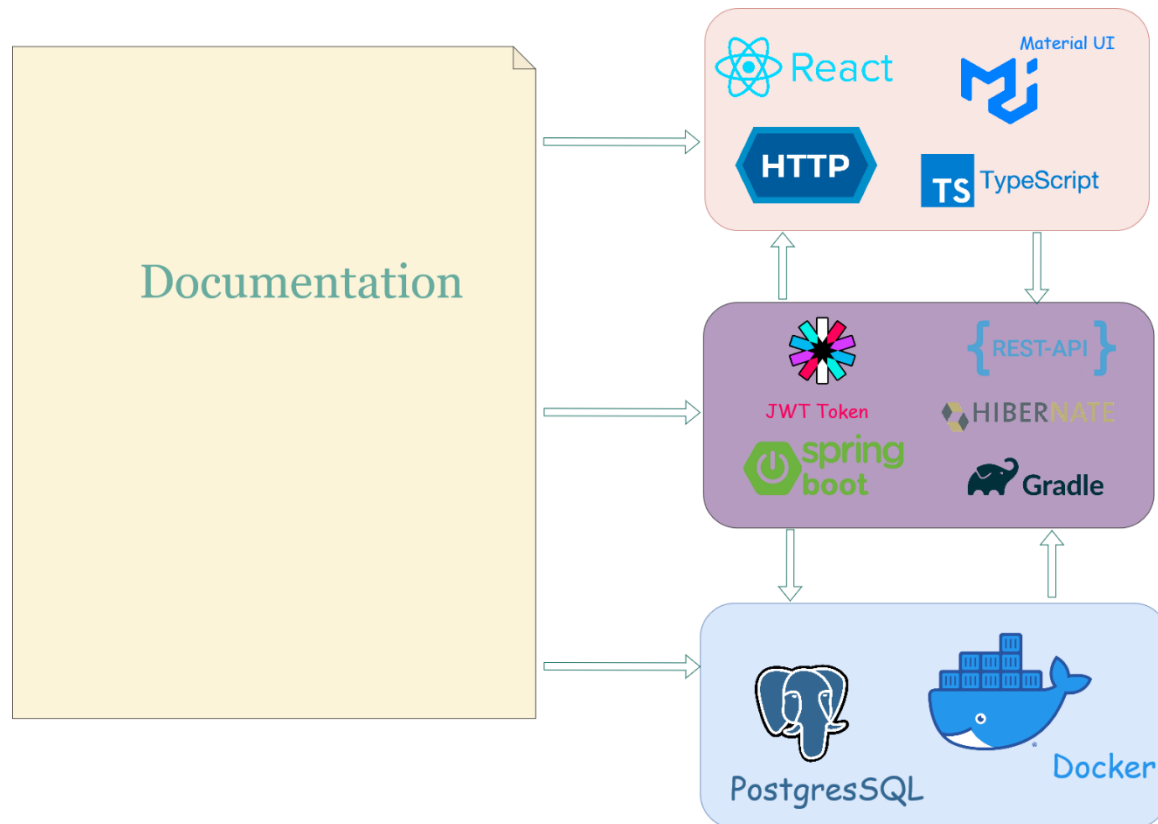
If we are stuck on the first three points, we asked each other for help and solved it together.

Tools that helped us find the solution / problem:

- Google
- Debugger
- Loggers by logging each step
- Console.log in frontend

## 8 Used Technologies

### 8.1 3-Tier Structure



This is how our application is built. The three tiers are made with the tools above. For the presentation Layer we used React Typescript and for the design Material UI Components. Material-UI is simply a library that allows us to import and use different components to create a user interface in our React application. For the Business Layer we used Java in Spring Boot. We build a gradle project using REST-API and for the authentication we implemented the JWT Token. Hibernate allows us the connection to our database with the OR-Mapper. Lastly our Data Layer, therefore we used a PostgreSQL Database with the help of Docker we set it up.

### 8.2 Database

Since we are building a whole software application, it logically needs a database to store the records.



### 8.2.1 Postgre SQL

For the application, the developers use the Postgres SQL database. This runs in the local dockers of the developers, which runs in the context of the backend.

### 8.2.2 DBeaver

To be able to insert/edit/display the data from our database, we need an environment for it. This is in our case DBeaver. We use the Community Edition, which is free of charge.

## 8.3 Backend

### 8.3.1 Spring Boot

Spring Boot makes developing web applications and microservices with Spring Framework faster and easier through three core features:

- Easy setup and management
- Standalone applications without a web server (Involved in the initialization process Tomcat/Jetty)
- Includes Spring ecosystem (Spring Data, Spring Security, ORM..)

### 8.3.2 Spring Data JPA

Spring Data JPA is one of the frameworks of Spring platform. Its goal is to simplify for the developer the persistence of data against different information repositories. With Spring Data JPA, the developers' work is made easier and partially removed. Developers do not have to write a data access layer or SQL queries.

### 8.3.3 Spring Web

Spring Web helps us build web applications quickly without a lot of "boiler code" and configurations.

### 8.3.4 PostgreSQL

PostgreSQL is an object-oriented DBMS, ORDBMS for short. PostgreSQL is a classic relational database but with more complex data types, PostgreSQL can be used for personal and business application. PostgreSQL supports SQL (relational), as well as JSON (non-relational) queries.

### 8.3.5 Gradle

Gradle is a build automation tool known for its flexibility in building software. A build automation tool is used to automate the creation of applications. The build process includes compiling, linking, and packaging the code.

## 8.4 Frontend

### 8.4.1 Yarn

Yarn is the package manager for software solutions used in the JavaScript environment with Node.js. Docker allows developers to separate the application with the infrastructure so that the software can be applied quickly and efficiently. It stores the dependency in images, which form the basis for a virtualized container that can run on any operating system.

### 8.4.2 Docker-Compose

Docker-Compose allows us to set multiple containers in one file, as well as determine their relationships with each other. After that, we can launch the containers with a single command.


### 8.4.3 Node

Node is a runtime environment that can execute JavaScript code outside of a web browser. The version that will be used in the project is Node version 16.

## 9 Tests Cases Definition

Name	Betriebsmittel in der Produktion anwenden
Test Case	1.0
Actor	User
Trigger	Images on Multi Item Page
Description	User opens the multi-item page, to see all products available in store.
Condition	User is logged in
Input	Menu: Formal-Wear
Expected Output	Images shown as card
Received Output	Images shown in card
Status	OK

Test Case	2.0
Actor	Log in
Trigger	Login Page
Description	User already has an account, he wants to log in

Condition	Already registered
Input	<pre>{   "email": "inotcreated@mail.com",   "password": "test123" }</pre>
Expected Output	Snackbar with failed statement, as the user does not exist
Received Output	Snackbar Message: 
Status	OK

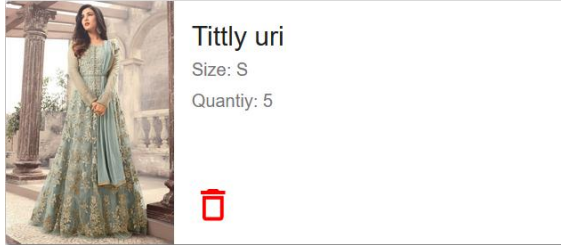
Test Case	3.0
Actor	Guest User
Trigger	Registration
Description	Guest wants to create an account
Condition	Not having an existing account
Input	<pre>{   "customerSurname": "lena",   "customerLastname": "meier",   "customerEmail": "lenameier@gmail.com",   "customerAddress": "Winterthurerstrasse 13",   "password": "123goodbye",   "country": "England",   "location": "London" }</pre>
Expected Output	User is created
Received Output	<pre>{   "customerId": "471d9891-d92e-4911-9b3f-1d037849e906",   "customerSurname": "alisha", }</pre>

	<pre> "customerLastName": "khalid", "customerEmail": "khalidalisha@gmail.com", "customerAddress": "chreuzacherstrasse 13", "password": "\$2a\$10\$As5kv5fmGWTIsLqN9LmIoeCe/DwKk1zzTU.u.53CwLaq.RzGX1W2m", "country": {   "countryId": "f5d33a92-cc45-41f0-8411-5f365a7275e5",   "countryName": "England" }, "location": {   "locationId": "8dc7f1ee-b5f6-495f-8713-b2fb042496a2",   "locationName": "Lahore",   "postalCode": "53201" } </pre>
Status	OK

Test Case	4.0
Actor	User
Trigger	HomePage
Description	Check if homepage gets rendered
Condition	User is on homepage
Input	<a href="http://localhost:3000/">http://localhost:3000/</a>
Expected Output	Homepage with its four categories should be shown
Received Output	Homepage gets all images for category
Status	OK

Test Case	5.0
Actor	User
Trigger	View Button → Multiitem Page
Description	Check if navigation of button functions to the multi-item page from the homepage
Condition	User is on homepage
Input	User clicks the view button in homepage
Expected Output	User is navigated to the multiitem page
Received Output	Multiitem page is shown
Status	OK

Test Case	6.0
Actor	User
Trigger	Searchbar
Description	User searches for clothing: "Bano"
Condition	User is on Multi-item Page
Input	Bano in searchbar
Expected Output	Clothing "Bano" is filtered from the rest
Received Output	Clothing: "Bano"
Status	OK

Test Case	7.0
Actor	User
Trigger	Ceckout Page
Description	Checkoutpage should show cart from the logged in user
Condition	User is logged in
Input	Checkout Page
Expected Output	Cart with items
Received Output	<p>Cart with items</p> 
Status	OK

## 10 Testing

### 10.1 Junit Testing

#### 10.1.1 Get all clothing items

```

@AutoConfigureMockMvc
@WebMvcTest
public class ClothingControllerTest {

    @MockBean
    private ClothingServiceImpl service;

    @Autowired
    private MockMvc mvc;

    private final String URL = "http://localhost:8080/clothing";

```

```

@Test
@Transactional
@Sql("/TestData.sql")
void findAllClothingItems() throws Exception {

    //WHEN
    MvcResult res = mvc.perform(get("/clothing/{id}", 1)).andReturn();
    //THEN
    Assert.assertEquals(200, res.getResponse().getStatus());
    Assert.assertTrue(res.getResponse().getContentAsString().contains("passing
the test"
));
}

```

The output of this test is this right here:

```

{
  "clothingId": "090f3d7c-f6ad-11ec-b939-0242ac120002",
  "clothingName": "Bahar-e-Nau",
  "clothingPrice": 120.0,
  "clothingDescription": "A 4 piece lilac bridal consisting of an elaborate ka
meez paired with a heavily embroidered farshi gharara set on a stunning organza base
complimented with a lilac dupatta.",
  "clothingImage": "https://i.pinimg.com/736x/41/86/80/418680c973c18fbf2aab67a
cd367ef19.jpg",
  "clothingSize": {
    "clothingSizeId": "b7b36b94-f641-11ec-b939-0242ac120002",
    "clothingSize": "XL"
  },
  "clothingType": {
    "clothingTypeId": "bcc8ed58-f6d6-11ec-b939-0242ac120002",
    "clothingType": "FORMAL"
  }
},
{
  "clothingId": "82c45353-c3df-489a-bc96-317b8873bc3d",
  "clothingName": "Bano",
  "clothingPrice": 130.5,
  "clothingDescription": "A 3 piece unstitched red suit",
  "clothingImage": "https://cdn.shopify.com/s/files/1/2337/7003/products/87_3a
cea76a-cfff-4f86-8f12-62d91e5c3e6c_700x.jpg?v=1647945606",
  "clothingSize": {
    "clothingSizeId": "d1cc2e67-956d-433b-a3ee-2129014acd04",
    "clothingSize": "XS"
  },
  "clothingType": {
    "clothingTypeId": "3d55bf07-b833-4aa0-9f3e-670ec6630960",
    "clothingType": "UNSTICHD"
  }
},

```

### 10.1.2 Register a new Customer

```

@Test
@DisplayName("Endpoint Create for Greater Region")
void registrateCustomer() throws Exception {
    CustomerDTO customerDTO = new CustomerDTO("Lena", "Meier",
"lena.meier@mail.com",
                                "Industriestrasse 12",
"mysuperpassword", "Switzerland", "Kloten");

    Country country =
countryRepository.findByCountryName(customerDTO.getCountry());
    Location location =
locationRepository.findByLocationName(customerDTO.getLocation());
    Customer customer = new Customer(customerDTO.getCustomerSurname(),
customerDTO.getCustomerLastname(),
customerDTO.getCustomerEmail(),
customerDTO.getCustomerAddress(),
encoder.encode(customerDTO.getPassword()),
country, location);

    JSONObject requestJson = new JSONObject();
    requestJson.put("customerSurname", "alisha").put(
        "customerLastname", "khalid").put(
        "customerEmail", "khalidalisha@gmail.com").put(
        "customerAddress", "chreuzacherstrasse 13").put(
        "password", "123goodbye").put(
        "country", "England").put(
        "location", "Lahore");

    mvc.perform(post(URL)
        .contentType(MediaType.APPLICATION_JSON)
        .content(requestJson.toString())
        .andExpect(status().isMethodNotAllowed()));
}

```

```

{
  "customerId": "3d3523d6-da3a-4894-9d29-ae1c5cb52386",
  "customerSurname": "alisha",
  "customerLastname": "khalid",
  "customerEmail": "khalidalisha@gmail.com",
  "customerAddress": "chreuzacherstrasse 13",
  "password": "$2a$10$YbG.hdmZw2y3UZ4B7zPJkezNVH3WuFO6yIWrrJKh8hK9yomFvutApO",
  "country": {
    "countryId": "f5d33a92-cc45-41f0-8411-5f365a7275e5",
    "countryName": "England"
  },
  "location": {
    "locationId": "8dc7f1ee-b5f6-495f-8713-b2fb042496a2",
    "locationName": "Lahore",
    "postalCode": "53201"
  }
}

```



```
}  
}
```

Here we are registering a new customer to our application. By adding all the attributes through our Junit Test we get the result that a new customer was added. Here you can also see it in our database:

	customer_address	customer_email	customer_lastname	customer_surname	customer_password
1	chreuzacherstrasse 13	khalidalisha@gmail.com	khalid	alisha	\$2a\$10\$1hG40IhTVX31LMJRhNhiUecHi7npC6eZeSbwc6xEsD358fvYzdePm

The password is saved encoded into our database because we worked with a JWT.

## 10.2 System Tests

### 10.2.1 Log in as User

The screenshot shows a REST client interface with a POST request to `http://localhost:8080/customer/signin`. The request body is a JSON object with email and password. The response is a JSON object containing a token, type, id, email, and password.

```
POST http://localhost:8080/customer/signin  
  
{"email": "khalidalisha@gmail.com",  
 "password": "123goodbye"}  
  
Status: 200 OK Time: 187 ms Size: 810 B  
  
{"token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJraGFsYWRRbG1zaGFZAz21haWwY29tIiwiaWF0IjoxNjU2ODgzNzkzLCJleHAiOjE2NTY4ODM3OTd9.493kaLLp9Tay00Aexwz11MEZDoreUF40xs2ryEXdHDh6H6YayNqd2Fpcny005u0_YoHw8dvKP-DBYfw0-a-8kg",  
 "type": "Bearer",  
 "id": "27839700-2ec6-4083-9313-9baf30bf07b8",  
 "email": "khalidalisha@gmail.com",  
 "password": "$2a$10$1hG40IhTVX31LMJRhNhiUecHi7npC6eZeSbwc6xEsD358fvYzdePm"}
```

The screenshot shows the Postman interface for a POST request to `http://localhost:8080/customer/signin`. The 'Tests' tab is active, displaying the following JavaScript code:

```
1 pm.test("Response time is less than 200ms", function () {  
2   pm.expect(pm.response.responseTime).to.be.below(200);  
3 });  
4 pm.test("Status code is 200", function () {  
5   pm.response.to.have.status(200);  
6 });
```

Below the code, the 'Test Results (2/2)' section shows two passed tests:

- PASS** Response time is less than 200ms
- PASS** Status code is 200

This last test is a Postman test. We successfully logged in the user which we created before and also made sure that the performance is below 200 ms and the most important that the response was positive.

## 11 Retrospective

---

### 11.1 Nuwera

This project was very successful. Alisha and I split up the work fairly and made our Mockup come true. In the start we were doing good and finished the design and database diagram really fast. In the “mid-phase” we started to get a little slower because we set such high standards for ourselves that we tried to make everything perfect. It was quite exhausting to finish everything in such a short time but we managed. And this has nothing to do with luck but with our dedication. We worked great as a team and also helped each other out if/when needed. Looking back we also were good at planning, we finished our project in sync with our GANTT-Diagram. We did not have to invest more hours than planned. If I could go back and changed one things whatsoever than I would probably divide our tasks per functionality rather than back- and frontend. Truthfully we both have our strengths and weaknesses but this still isn't an excuse to not to something. All in all I am very proud with what we were capable of delivering.

### 11.2 Alisha

This project was fun because we could choose the project idea ourselves and with Nuwera as my team partner I am quite happy with the result and our cooperation. The project was a good repetition of everything I learned the past two years because we built a Web-Shop with a front- and backend. I am glad to have the database module before this one as I could use the knowledge in this project. I think Nuwera and I worked equal in this project I was more in charge of the frontend and Nuwera in the backend. If a problem occurred, we were always there for each other and found a solution together. In the beginning we were quite good on our way to fulfill the Mockup design, although towards the end it got a little stressful. Throughout the project we were always in sync with our GANTT Diagram, and we could finish all in time because of our ambition and motivation. In retrospect, I think we planned the project very well, despite the short time frame. I would not change anything except the responsibilities so that both have worked in both areas equally.