

Simplilearn: Advanced Java Development

Project: E-commerce Website for Sporty Shoes

Karsten Hecker; Vodafone Germany

Deployed on Mac; Browser: Safari

February 2024

Project Overview:

1

HIGH LEVEL DESIGN

3

Code Project Explorer: Frontend

4

Code Project Explorer: Backend

5

Start using the Application

7

User Management Practice:

12

Product management and view practice:

17

Project Overview:

E-commerce Website for Sporty Shoes - The website displays an overview of all shoes. These are listed by ID, name, price, and image. Additionally, the products can be filtered and sorted by categories. If you are registered and logged in as an administrator, you can also see a list of all users. They decided to hire a Full Stack Developer to design and develop the website.

This project focuses on developing a backend project using JDBC, JSP, and Servlets, although a minimal frontend should be implemented. You can create frontend forms on your own or use online resources. It should include a search form to allow entries product details, list the available products with prices, and an image of the product.

Technical Components –Frontend, Backend, Database Management & Test:

- IDE: IntelliJ
- Programming Language: Java
- Java Server Pages (JSP)
- Java Servlets
- Java JDBC
- Database: MySQL
- Hibernate
- Frontend: React
- Test API functionality: Postman

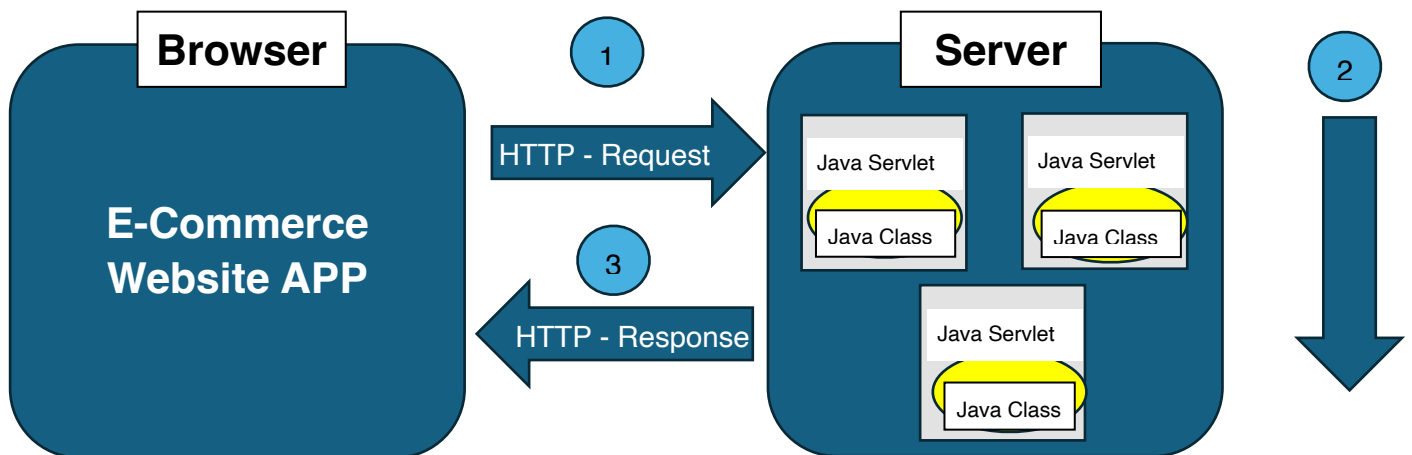
Core Concepts Demonstrated:

- JSPs, Servlets, and JDBC to implement a fully connected application to a MySQL Database
- Implement frontend Form Design
- Implement Database CRUD features using JDBC
- Populated tables via script

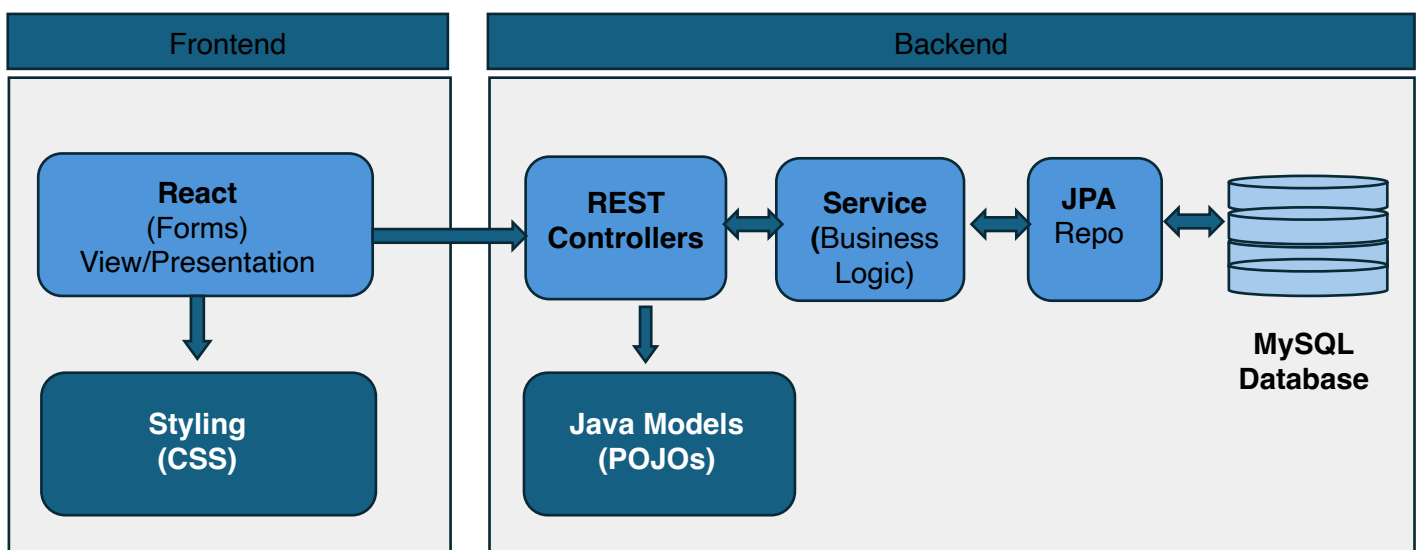
Features Roles:

All Users	Admin Users
- register new user	- Login
- Login	- Sort and filter products
- Change Password	- product maintenance (adding, updating, deleting and viewing)
- sort and filter products	- User management (adding, updating, deleting and viewing)
- click product to add to the cart	
- remove products from the cart	

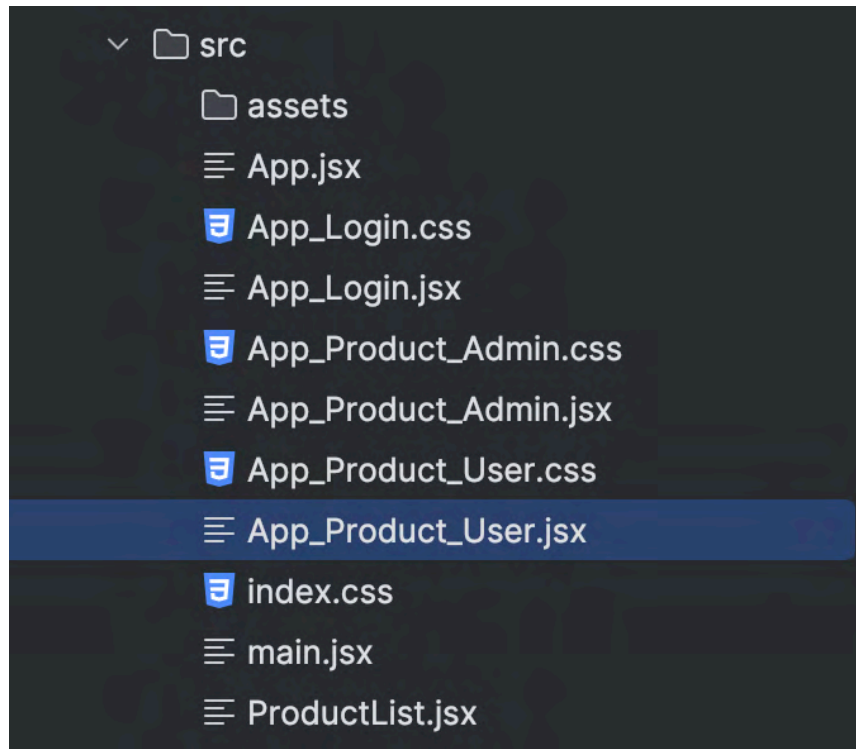
HIGH LEVEL DESIGN



1. User wants to load the E-commerce Website application. The Browser will send an HTTP request over to the server for the Website landing page.
2. Servlet will execute and generate HTML code on the fly (dynamically), and
3. Servlet will send the HTML file back to the browser as a HTTP response.



Code Project Explorer: Frontend



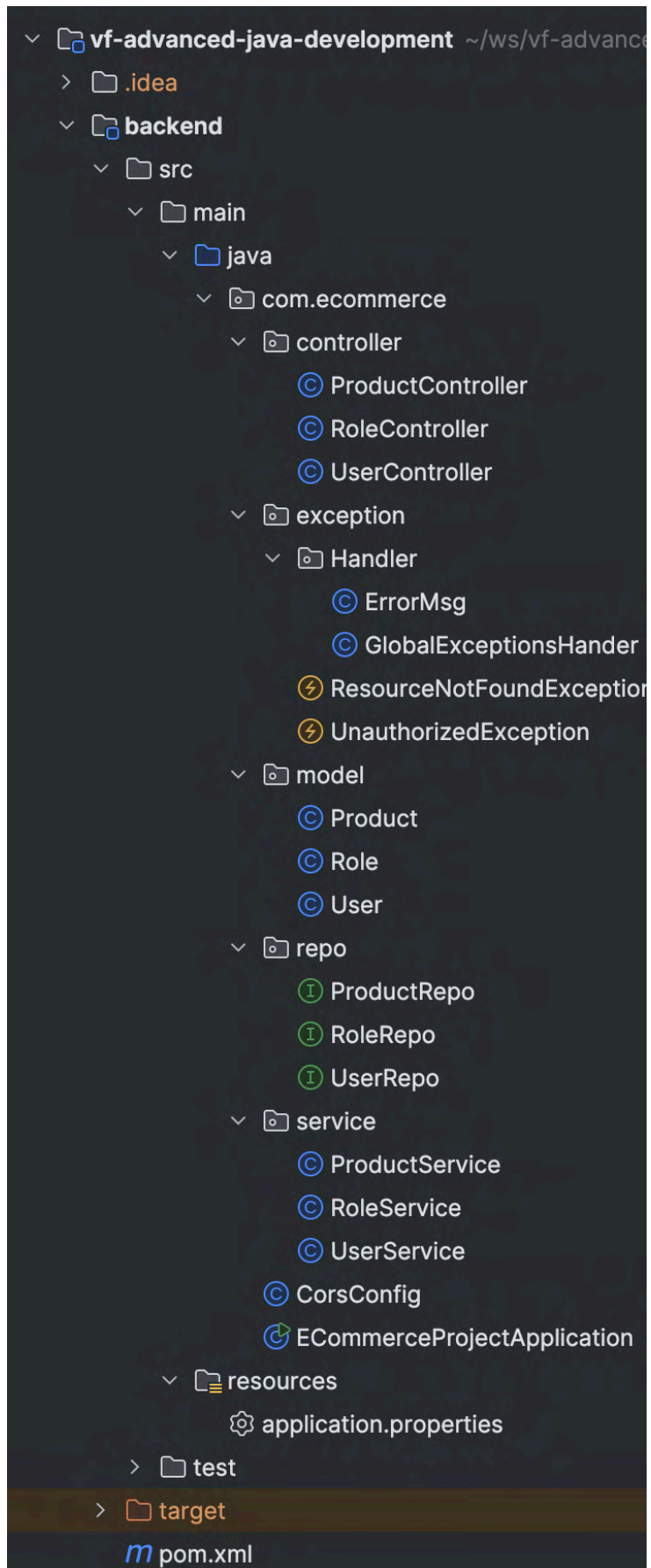
.jsx

- **App.:** Started the React Router to interact between Login, Product_Admin and Product_User.
- **App_Login.:** Includes the handle.login, handle.registration and handle.logout functionality.
- **App_Poduct_Admin.:** Started the main Website for Administrator view with a couple of administration specific functionalities.
- **App_Poduct_User.:** Started the main Website for User view with a couple of user specific functionalities.
- **Main.:** Is used to connect the React components with the HTML element on the page and render the application.

.CSS

- **All.:** Includes the styling and view configuration for each .jsx file.

Code Project Explorer: Backend



controller: Product, Role, User

The controllers are classes that receives HTTP requests and responds to them. The controllers are responsible for processing requests (e.g., GET, POST, PUT, DELETE) and returning responses to the client.

model (Pojos): Product, Role, User

POJOs are simple Java objects with no special requirements and are often used in Spring Boot to store or transfer data. They can be used as data models, **DTOs** (Data Transfer Objects), or entities. When used as JPA entities, they can interact with the database. POJOs are a fundamental concept in Spring Boot development and help in modeling data and facilitating communication between the layers of the application.

service: Product, Role, User

The services are key component that holds business logic, injected into controllers, and interacts with the data layer. They promotes a clean separation of concerns, enhances testability, and supports features like transaction management.

repo: Product, Role, User

The repository interfaces are a powerful way to interact with the database using **Spring Data JPA**. By extending `JpaRepository`, you automatically gain access to a variety of **CRUD (Create, Read, Update, Delete)** operations without the need for manual implementation. You can also define custom queries to retrieve or manipulate data based on specific criteria. This pattern significantly reduces boilerplate code and improves productivity when working with data persistence in Spring Boot.

exception:

The exceptions are used to handle errors or unexpected situations that may occur during the execution of the application. Exception handling ensures that the application can gracefully recover from errors, provide meaningful error messages to users, and maintain a clean flow of execution. It provides various ways to handle exceptions, including using global exception handlers, custom exceptions, and response status codes.

CorsConfig:

CORS (Cross-Origin Resource Sharing) is a security feature implemented by browsers that restricts web pages from making requests to a domain different from the one that served the web page. It is necessary to allow cross-origin requests, especially when the frontend and backend are hosted on different domains or ports (e.g., a React frontend interacting with a Spring Boot backend).

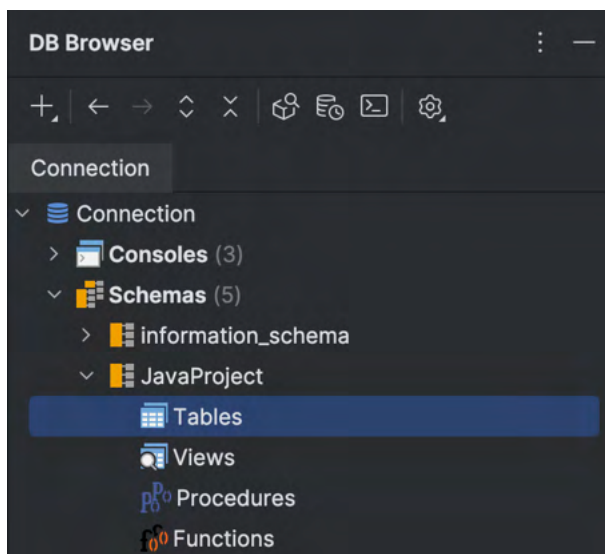
Start using the Application

Database Initialization:

To initialize the MySQL database and populate it with the required tables, the main class must be started.

The two database tables 'User' and 'Role' are related to each other. Each unique RoleId is always assigned to a user. This relationship is represented in the User table through the 'role_id' field. This is ensured in the POJO using a @ManyToOne and @JoinColumn annotation.

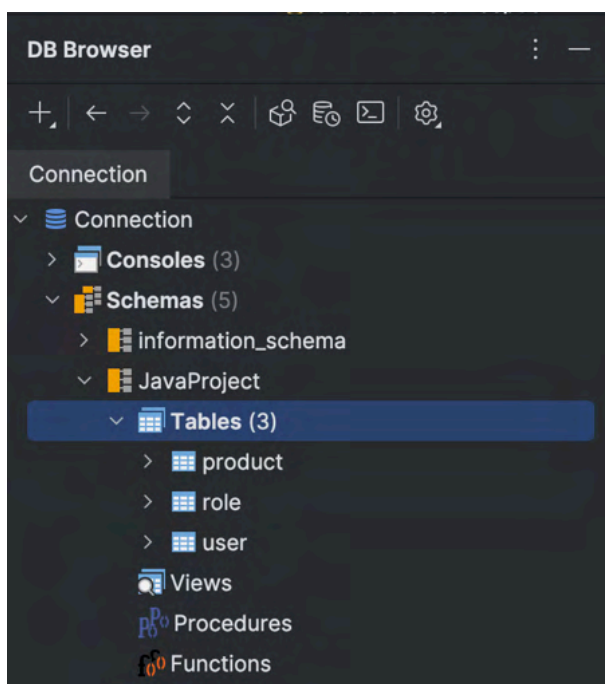
Before:



```
@Entity 25 usages karstenhecker
public class User {
    @Id 2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name; 2 usages
    private String password; 2 usages

    @ManyToOne 2 usages
    @JoinColumn(name="role_id")
    private Role role;
}
```

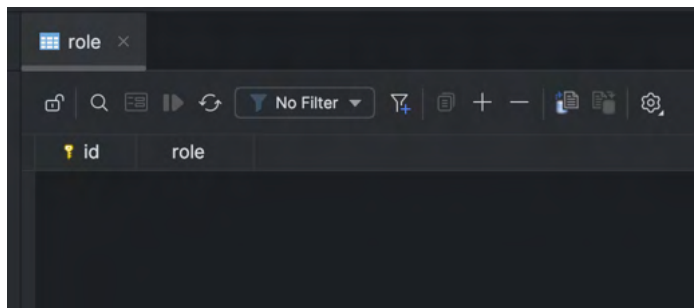
After:



Role creation:

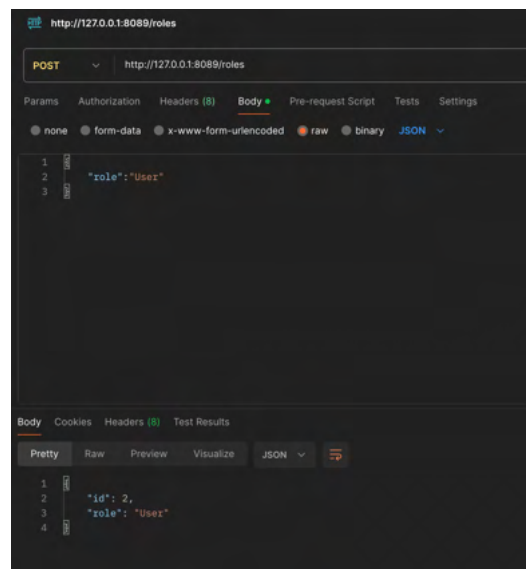
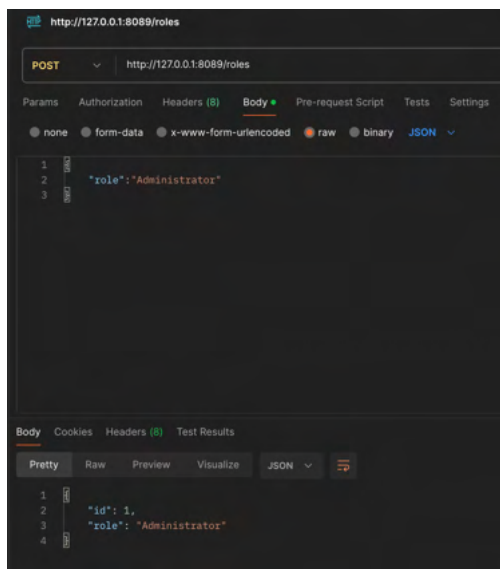
In the next step, the roles (Administrator and User) need to be created in the "Role" table. This can be easily done with the tool "Postman," as it also allows testing whether the API calls are functioning correctly.

Before:

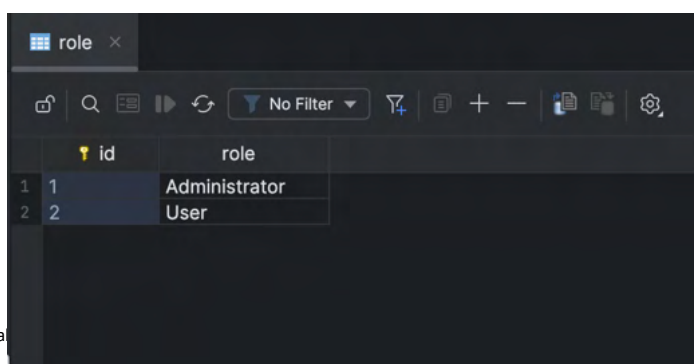


	id	role
--	----	------

Postman:



After:

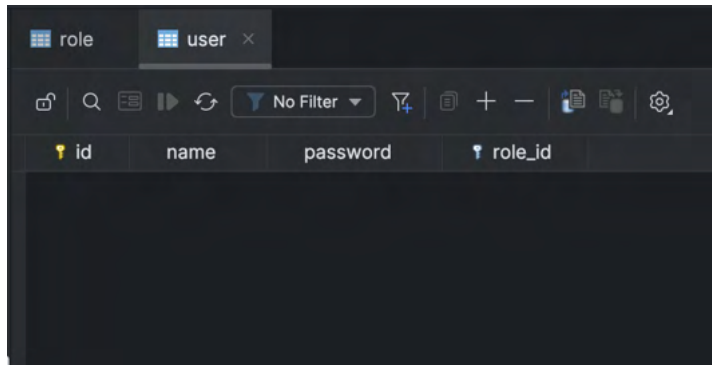


	id	role
1	1	Administrator
2	2	User

Create an administrator user:

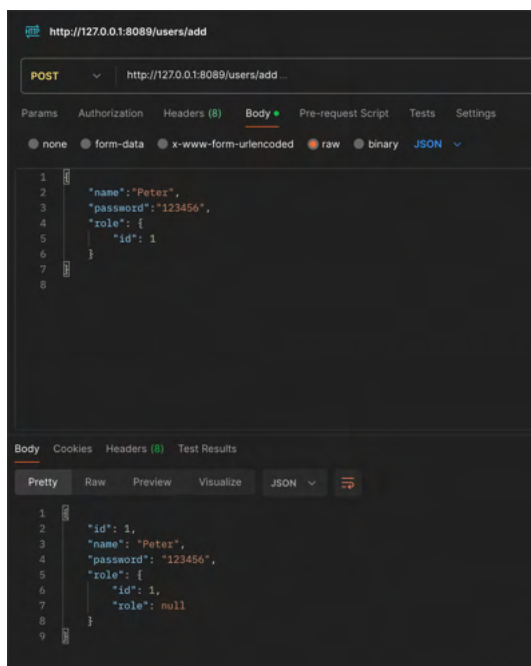
An administrator must now be created in user table to gain initial access to the UI in order to create additional users and products. For testing purposes, it is best to use the tool "Postman" to simultaneously check the API functionality.

Before:

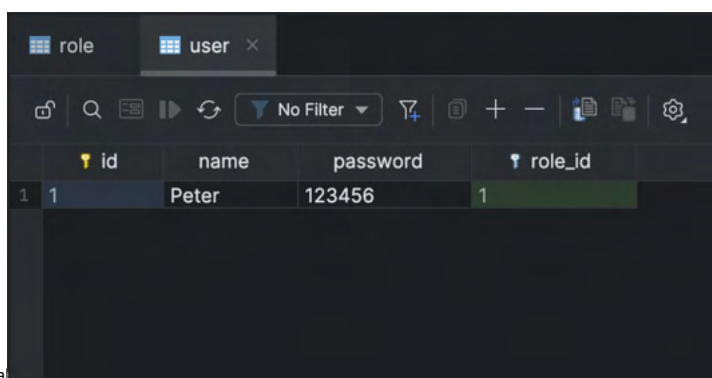


id	name	password	role_id
----	------	----------	---------

Postman:



After:



	id	name	password	role_id
1	1	Peter	123456	1

Example: how to view, add, update and delete Users

(works also for Products and Roles)

1. create a User POJO (Data Transfer Objects / Entities)
 - Represent the data exchanged between the layers (Controller, Service, Repository)
 - Are processed in the Service and returned
2. Create a Controller class
 - Receives HTTP request (e.g., GET, POST)
 - Calls methods in the Service
 - Returns a response to the client
3. Create a Service class
 - Contains the business logic
 - Calls methods in the Repository to process or retrieve data
 - Processes the POJOs and returns them to the Controller
4. Create a Repo class
 - Manages the database interaction
 - Executes database queries like `findAll()`, `save()`, `delete()`
 - Returns the data to the Service
5. Create a JSP page
 - receives the data passed to it by the controller and renders it as HTML
 - uses Java taglibs and EL (Expression Language) to access the data passed by the controller
 - Dynamic content is displayed on the JSP page based on the models provided.

User Management

Add New User

User Name

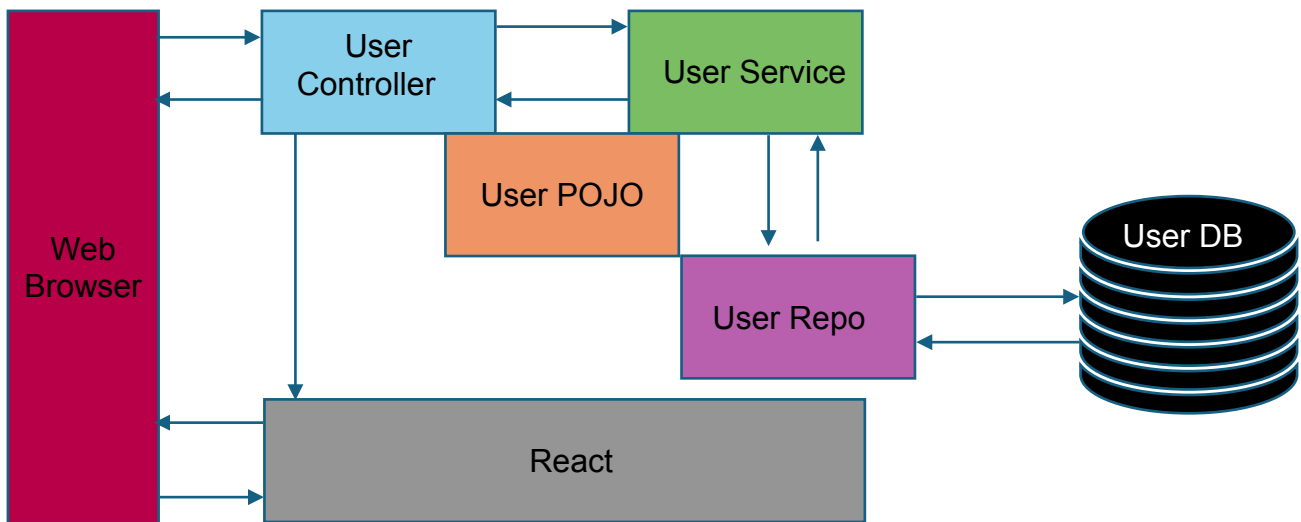
User Password

Add User

User List

ID	User Name	User Password	Actions
1	Peter	123456	<div>EditDelete</div>

Diagramm:



Visualized Interactions:

- **Browser → Controller:** The web browser sends an HTTP request to the controller.
- **Controller → Service:** The controller calls the service to process the business logic.
- **Service → Repository:** The service calls the repository to interact with the database.
- **Repository → Database:** The repository executes database queries (e.g., `findAll()`, `save()`).
- **Database → Repository:** The database returns the results to the repository.
- **Repository → Service:** The repository returns the data to the service.
- **Service → Controller:** The service returns the processed data to the controller.
- **Controller → React:** The controller passes the data to the React.
- **React → Browser:** The React renders HTML and sends it back to the browser.
- **Browser → Browser:** The web browser displays the rendered HTML page to the user.

User Management Practice:

1. Administrator add new User.

The screenshot shows the 'User Management' interface. The 'Add New User' form is highlighted with a purple oval. It contains two input fields: 'User Name' with the value 'Marta' and 'User Password' with the value 'pwd'. Below the fields is a green 'Add User' button. A success message 'User added successfully' is displayed in a white box with a 'Schließen' button. The 'User List' table below shows one user: Peter with ID 1 and password 123456.

ID	User Name	User Password	Actions
1	Peter	123456	Edit Delete

The screenshot shows the 'User Management' interface. The 'Add New User' form is highlighted with a purple oval. It contains two input fields: 'User Name' and 'User Password'. Below the fields is a green 'Add User' button. The 'User List' table below shows two users: Peter with ID 1 and password 123456, and Marta with ID 2 and password 'pwd'.

ID	User Name	User Password	Actions
1	Peter	123456	Edit Delete
2	Marta	pwd	Edit Delete

```
const handleUserSubmit = async (e) => {
  e.preventDefault();
  const user = { name: userName, password: userPassword };

  try {
    const response = await fetch("http://localhost:8089/users/add", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(user),
    });

    if (response.ok) {
      alert("User added successfully");
      fetchUsers();
      resetUserForm();
    } else {
      alert("Failed to add user");
    }
  } catch (error) {
    console.error("Error:", error);
    alert("Error adding the user");
  }
};
```

App_Product_Admin.jsx

The screenshot shows a database table with the following data:

	id	name	password	role_id
1	1	Peter	123456	1
2	2	Marta	pwd	2

IMPORTANT:

When a new user is created, they are automatically assigned the default value 'User' with the corresponding ID. This applies both during registration and in user management.

2. Administrator update a User

User Management

Add New User

User Name

User Password

Add User

User List

ID	User Name	User Password	Actions
1	Peter	123456	<button>Edit</button> <button>Delete</button>
3	Susi	6789	<button>Edit</button> <button>Delete</button>

```
const handleUpdateUser = async (e) => {
  e.preventDefault();
  const user = { name: userName, password: userPassword };

  try {
    const response = await fetch(`http://localhost:8089/users/update/${userId}`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(user),
    });

    if (response.ok) {
      alert("User updated successfully");
      fetchUsers();
      resetUserForm();
    } else {
      alert("Failed to update user");
    }
  } catch (error) {
    console.error("Error:", error);
    alert("Error updating the user");
  }
};
```

App_Product_Admin.jsx

User Management

Update User

User Name

User Password

Update User

User List

ID	User Name	User Password	Actions
1	Peter	123456	<button>Edit</button> <button>Delete</button>
3	Susi	6789	<button>Edit</button> <button>Delete</button>

User Management

Add New User

User updated successfully

User Name

User Password

Add User

User List

ID	User Name	User Password	Actions
1	Peter	123456	<button>Edit</button> <button>Delete</button>
3	Susi	9876	<button>Edit</button> <button>Delete</button>

	id	name	password	role_id
1	1	Peter	123456	1
2	3	Susi	9876	2

3. Administrator delete a User

User Management

Add New User

User Name

User Password

Add User

User List

ID	User Name	User Password	Actions
1	Peter	123456	<button>Edit</button> <button>Delete</button>
2	Marta	pwd	<button>Edit</button> <button>Delete</button>

```
const handleDeleteUser = async (userId) => {
  try {
    const response = await fetch(`http://localhost:8089/users/byUserId/${userId}`, {
      method: "DELETE",
    });

    if (response.ok) {
      alert("User deleted successfully");
      fetchUsers();
    } else {
      alert("Failed to delete user");
    }
  } catch (error) {
    console.error("Error:", error);
    alert("Error deleting the user");
  }
};
```

App_Product_Admin.jsx

User Management

Add New User

User deleted successfully

User Name

User Password

Add User

User List

ID	User Name	User Password	Actions
1	Peter	123456	<button>Edit</button> <button>Delete</button>

	id	name	password	role_id
1	1	Peter	123456	1

4. User Registration

Registration

Userername

Password

Registration

Account available? **back to Login**

Login

Userername

Password

Login

still no Account? **now register**

User View

Logout

Product Shoe Details

Filter by Name Filter by Price Filter by ID

Product List

ID	Product Name	Product Price	Product Image	Add to Cart
1	Peter	123456		
2	Susi	9876		

Shopping Cart

Product Name	Product Price	Product Image	Action

Change Password

Old Password:

New Password:

Confirm New Password:

Change Password

```
//Handle Register
nst handleRegister = async (e) => {
  e.preventDefault();

  if (!name || !password) {
    setErrorMessage('Username und Password required');
    return;
  }

  setIsLoading(true);
  setErrorMessage('');

  try {
    const response = await fetch('http://localhost:8089/users/add', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ name, password }),
    });

    if (response.ok) {
      setIsRegistering(false)
      setPassword(null)
      navigate('/'); // move to Login-page after successful registration
    } else {
      setErrorMessage('Failure with registration');
    }
  } catch (error) {
    console.error('Failure with registration:', error);
    setErrorMessage('Problem with registration');
  } finally {
    setIsLoading(false);
  }
}
```

App_Login.jsx

```
try {
  const response = await fetch('http://localhost:8089/users/authenticate', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ name, password }),
  });

  if (response.ok) {
    const data = await response.json();
    localStorage.setItem('role', data.role.role); // Save role of user.
    localStorage.setItem('userID', data.id); //save id of user
    setName('');
    setPassword('');
    navigate('/'); // navigate to main page
  } else {
    localStorage.removeItem('role')
    localStorage.removeItem('userID')
    setErrorMessage('Username or Password wrong!');
  }
} catch (error) {
  localStorage.removeItem('role')
  console.error('Failure with Authentication:', error);
  setErrorMessage('Problem with Authentication');
} finally {
  setIsLoading(false);
  window.location.reload();
}
```

App_Login.jsx

	id	name	password	role_id
1	1	Peter	123456	1
2	3	Susi	9876	2

5. User Change Password

Change Password

Old Password:
.....

New Password:
.....

Confirm New Password:
.....

Change Password

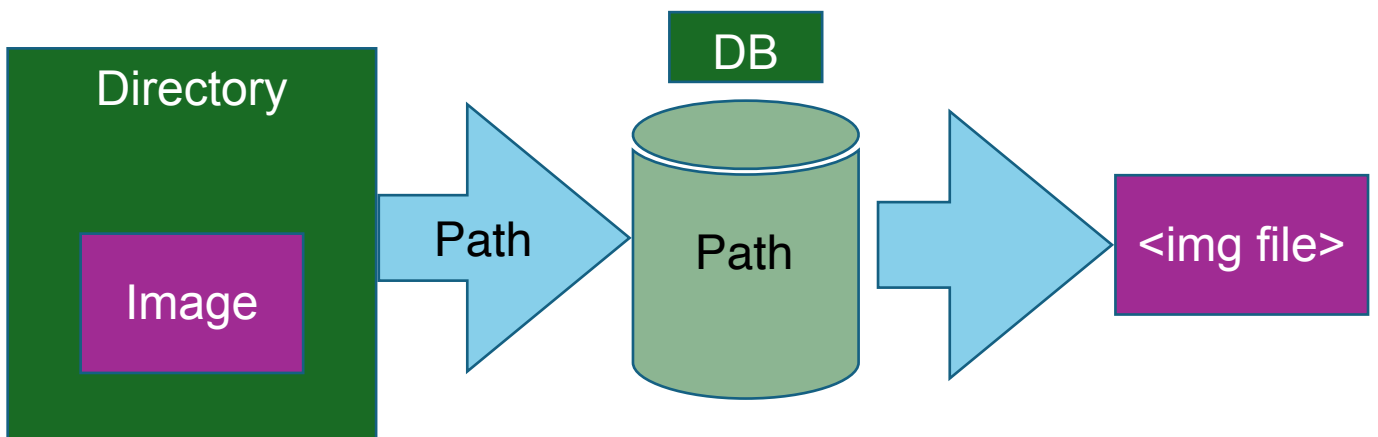
Password updated successfully.

No Filter				
	id	name	password	role_id
1	1	Peter	123456	1
2	3	Susi	6789	2

Product management and view practice:

Workflow for files (images)

- **Upload Image:** A file is uploaded via (/files/upload) and saved on the server, with the path stored in the database.
- **Retrieve Image:** The image is retrieved via /files/image/{id}, where the ID is fetched from the database and the image is sent back to the browser.
- **Display Image:** The browser will display the image from the path returned by the server.



1. Administrator add and list a new product.

Admin View

Logout

Product Shoe Details

Upload New Product

Product Name
adidas sport shoe

Product Price
150 EUR

Product Image
Datei auswählen adidas.jpeg

Upload Product

Filter by Name Filter by Price Filter by ID

Product List

ID	Product Name	Product Price	Product Image	Actions
----	--------------	---------------	---------------	---------

id	price	image_path	name

App_Product_Admin.jsx

```
const handleSubmitProduct = async (e) => {
  e.preventDefault();
  const formData = new FormData();
  formData.append("name", name);
  formData.append("price", price);
  formData.append("image", image);

  try {
    const response = await fetch("http://localhost:8089/products/add", {
      method: "POST",
      body: formData,
    });

    if (response.ok) {
      alert("Product uploaded successfully");
      fetchProducts();
      resetProductForm();
    } else {
      alert("Failed to upload product");
    }
  } catch (error) {
    console.error("Error:", error);
    alert("Error uploading the product");
  }
};
```

Admin View

Logout

Product uploaded successfully

Product Shoe Details

Upload New Product

Product Name


Product Price

Product Image
Datei auswählen adidas.jpeg

Upload Product

Filter by Name Filter by Price Filter by ID

Product List

ID	Product Name	Product Price	Product Image	Actions
1	adidas sport shoe	150 EUR		Edit Delete

id	price	image_path	name
1	150 EUR	adidas.jpeg	adidas sport shoe

App_Product_Admin.jsx

```
const fetchProducts = async () => {
  try {
    const response = await fetch("http://localhost:8089/products/all");
    if (response.ok) {
      const data = await response.json();
      setProducts(data);
    } else {
      console.error("Failed to fetch products");
    }
  } catch (error) {
    console.error("Error:", error);
  }
};
```


2. Administrator update a product.

Product Shoe Details

Upload New Product




Product Name

Product Price

Product Image
  nb.jpeg

Filter by Name Filter by Price Filter by ID

Product List


ID	Product Name	Product Price	Product Image	Actions
1	adidas sport shoe	150 EUR		<input type="button" value="Edit"/> <input type="button" value="Delete"/>
3	NB sport shoe	170 EUR		<input type="button" value="Edit"/> <input type="button" value="Delete"/>
2	nike sport shoe	150 EUR		<input type="button" value="Edit"/> <input type="button" value="Delete"/>

Update Product

Product updated successfully




Product Name

Product Price

Product Image
  nike.jpeg

Filter by Name Filter by Price Filter by ID

Product List

ID	Product Name	Product Price	Product Image	Actions
1	adidas sport shoe	150 EUR		<input type="button" value="Edit"/> <input type="button" value="Delete"/>
3	NB sport shoe	170 EUR		<input type="button" value="Edit"/> <input type="button" value="Delete"/>
2	nike sport shoe	150 EUR		<input type="button" value="Edit"/> <input type="button" value="Delete"/>

	id	price	image_path	name
1	1	150 EUR	adidas.jpeg	adidas sport shoe
2	2	150 EUR	nike.jpeg	nike sport shoe
3	3	170 EUR	nb.jpeg	NB sport shoe

```
const handleUpdateProduct = async (e) => {
  e.preventDefault();
  const formData = new FormData();
  formData.append("name", name);
  formData.append("price", price);
  formData.append("image", image);

  try {
    const response = await fetch(`http://localhost:8089/products/update/${id}`, {
      method: "POST",
      body: formData,
    });

    if (response.ok) {
      alert("Product updated successfully");
      fetchProducts();
      resetProductForm();
    } else {
      alert("Failed to update product");
    }
  } catch (error) {
    console.error("Error:", error);
    alert("Error updating the product");
  }
};
```


App_Product_Admin.jsx

	id	price	image_path	name
1	1	150 EUR	adidas.jpeg	adidas sport shoe
2	2	160 EUR	nike.jpeg	nike sport shoe
3	3	170 EUR	nb.jpeg	NB sport shoe

Upload New Product




Product Name

Product Price

Product Image
  nike.jpeg

Filter by Name Filter by Price Filter by ID

Product List

ID	Product Name	Product Price	Product Image	Actions
1	adidas sport shoe	150 EUR		<input type="button" value="Edit"/> <input type="button" value="Delete"/>
3	NB sport shoe	170 EUR		<input type="button" value="Edit"/> <input type="button" value="Delete"/>
2	nike sport shoe	160 EUR		<input type="button" value="Edit"/> <input type="button" value="Delete"/>

3. Administrator delete a product from list

Upload New Product


Product deleted successfully

Product Name

Product Price

Product Image

Datei auswählen






Upload Product

Filter by Name

Filter by Price

Filter by ID

Product List

ID	Product Name	Product Price	Product Image	Actions
1	adidas sport shoe	150 EUR		<div>Edit Delete</div>
3	NB sport shoe	170 EUR		<div>Edit Delete</div>
2	nike sport shoe	160 EUR		<div>Edit Delete</div>

	id	price	image_path	name
1	1	150 EUR	adidas.jpeg	adidas sport shoe
2	2	160 EUR	nike.jpeg	nike sport shoe
3	3	170 EUR	nb.jpeg	NB sport shoe

```
const handleDeleteProduct = async (productId) => {
  try {
    const response = await fetch(`http://localhost:8089/products/byProductId/${productId}`, {
      method: "DELETE",
    });

    if (response.ok) {
      alert("Product deleted successfully");
      fetchProducts();
    } else {
      alert("Failed to delete product");
    }
  } catch (error) {
    console.error("Error:", error);
    alert("Error deleting the product");
  }
};
```

App_Product_Admin.jsx


Upload New Product

Product Name

Product Price

Product Image

Datei auswählen





Upload Product

Filter by Name

Filter by Price

Filter by ID

Product List


ID	Product Name	Product Price	Product Image	Actions
1	adidas sport shoe	150 EUR		<div>Edit Delete</div>
3	NB sport shoe	170 EUR		<div>Edit Delete</div>

	id	price	image_path	name
1	1	150 EUR	adidas.jpeg	adidas sport shoe
2	3	170 EUR	nb.jpeg	NB sport shoe

4. Administrator filter and sort the list.


Filter by Name Filter by Price Filter by ID

Product List

ID	Product Name	Product Price	Product Image	Actions
3	NB sport shoe	170 EUR		<button>Edit</button> <button>Delete</button>


Filter by Name Filter by Price Filter by ID

Product List

ID	Product Name	Product Price	Product Image	Actions
1	adidas sport shoe	150 EUR		<button>Edit</button> <button>Delete</button>

Filter by Name Filter by Price Filter by ID



Product List

ID	Product Name	Product Price	Product Image	Actions
3	NB sport shoe	170 EUR		<button>Edit</button> <button>Delete</button>

```
// Filter- and Sort logic for products
const filteredProducts = products
  .filter((product) =>
    product.name.toLowerCase().includes(filter.name.toLowerCase())
  )
  .filter((product) =>
    product.price.toString().includes(filter.price.toString())
  )
  .filter((product) =>
    product.id.toString().includes(filter.id.toString())
  )
  .sort((a, b) => {
    if (sortField === "name") {
      return sortOrder === "asc"
        ? a.name.localeCompare(b.name)
        : b.name.localeCompare(a.name);
    } else if (sortField === "price") {
      return sortOrder === "asc" ? a.price - b.price : b.price - a.price;
    } else if (sortField === "id") {
      return sortOrder === "asc" ? a.id - b.id : b.id - a.id;
    }
    return 0;
  });
```

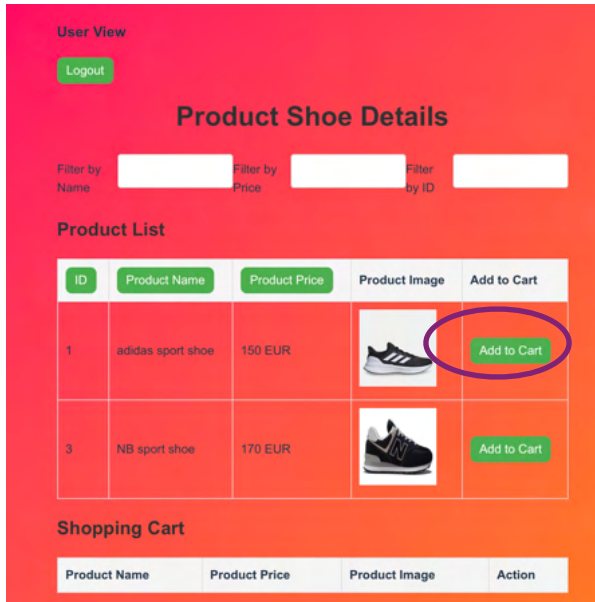
App_Product_Admin.jsx

Product List

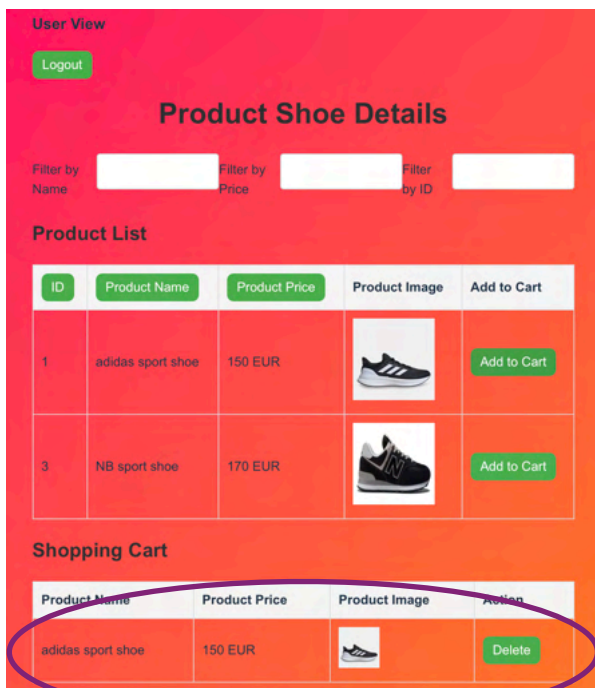
ID	Product Name	Product Price	Product Image	Actions
1	adidas sport shoe	150 EUR		<button>Edit</button> <button>Delete</button>
3	NB sport shoe	170 EUR		<button>Edit</button> <button>Delete</button>

User experience and view practice:

1. User add a product to cart



```
const handleAddToCart = (product) => {  
  setCart((prevCart) => {  
    const newCart = [...prevCart, product];  
    return newCart;  
  });  
};  
App_Product_User.jsx
```



2. User delete a product from cart



User View

Logout


Product Shoe Details

Filter by NameFilter by PriceFilter by ID

Product List

ID	Product Name	Product Price	Product Image	Add to Cart
1	adidas sport shoe	150 EUR		Add to Cart
3	NB sport shoe	170 EUR		Add to Cart

Shopping Cart

Product Name	Product Price	Product Image	Action
adidas sport shoe	150 EUR		Delete

```
const handleRemoveFromCart = (productId) => {  
  setCart((prevCart) => prevCart.filter((product, index) => index !== productId));  
};
```

App_Product_User.jsx



User View

Logout

Product Shoe Details

Filter by NameFilter by PriceFilter by ID

Product List

ID	Product Name	Product Price	Product Image	Add to Cart
1	adidas sport shoe	150 EUR		Add to Cart
3	NB sport shoe	170 EUR		Add to Cart


Shopping Cart

Product Name	Product Price	Product Image	Action
--------------	---------------	---------------	--------

3. User filter and sort the list.


Filter by Name Filter by Price Filter by ID

Product List

ID	Product Name	Product Price	Product Image	Actions
3	NB sport shoe	170 EUR		<button>Edit</button> <button>Delete</button>


Filter by Name Filter by Price Filter by ID

Product List



ID	Product Name	Product Price	Product Image	Actions
1	adidas sport shoe	150 EUR		<button>Edit</button> <button>Delete</button>

Filter by Name Filter by Price Filter by ID

Product List

ID	Product Name	Product Price	Product Image	Actions
3	NB sport shoe	170 EUR		<button>Edit</button> <button>Delete</button>

Product List

ID	Product Name	Product Price	Product Image	Actions
1	adidas sport shoe	150 EUR		<button>Edit</button> <button>Delete</button>
3	NB sport shoe	170 EUR		<button>Edit</button> <button>Delete</button>

```
const resetProductForm = () => {
  setName("");
  setPrice("");
  setImage(null);
  setId(null);
};

const filteredProducts = products
  .filter((product) =>
    product.name.toLowerCase().includes(filter.name.toLowerCase())
  )
  .filter((product) =>
    product.price.toString().includes(filter.price.toString())
  )
  .filter((product) =>
    product.id.toString().includes(filter.id.toString())
  )
  .sort((a, b) => {
    if (sortField === "name") {
      return sortOrder === "asc"
        ? a.name.localeCompare(b.name)
        : b.name.localeCompare(a.name);
    } else if (sortField === "price") {
      return sortOrder === "asc" ? a.price - b.price : b.price - a.price;
    } else if (sortField === "id") {
      return sortOrder === "asc" ? a.id - b.id : b.id - a.id;
    }
    return 0;
  });
```

App_Product_User.jsx

```
const handleSort = (field) => {
  setSortField(field);
  setSortOrder((prevOrder) => (prevOrder === "asc" ? "desc" : "asc"));
};
```

App_Product_User.jsx

4. User Logout



User View

[Logout](#)

Product Shoe Details

Filter by Name Filter by Price Filter by ID

Product List

ID	Product Name	Product Price	Product Image	Add to Cart
1	adidas sport shoe	150 EUR		Add to Cart
3	NB sport shoe	170 EUR		Add to Cart

Shopping Cart

Product Name	Product Price	Product Image	Action
--------------	---------------	---------------	--------

Change Password

Old Password:

New Password:

Confirm New Password:

[Change Password](#)

```
const handleLogout = () => {  
  localStorage.removeItem("role");  
  localStorage.removeItem('userID')  
  window.location.reload();  
};  
App_Product_User.jsx
```

Login

Username

Password

[Login](#)

still no Account? [now register](#)