ONLINE STORE (E-Commerce)

Project Developed by **P. SARAVANAN**

Content:

- Requirement Documentation.
- Software Design Documentation.
- Technical Documentation (Backend code).
- User Guide Documentation.

Project Overview:

Main Language: Java

Web Framework: Spring Boot

Tools: SpringToolSuite4

Project Purpose:

Create an e-commerce platform for customers to browse and purchase products conveniently from anywhere.

Requirement Documentation

Requirements:

- We need the correct market prices for all products.
- We require HD+ images of all products.
- Need a list of products. What products client going to sell in this web application.
- Also, we need user details for order placement, including Name, Mobile number, Complete Address, and Credit or Debit card number(for online purchases).

Luhn Algorithm:

- A mathematical algorithm utilized internally within the application to validate Credit and Debit card numbers.

Software Design Documentation

This project comprises totally 7 web pages.

HTML, CSS,Java Script,JSP are used in this project.

Login Page and Home Page of Online Store(Page 1):

HTML Functionalities:

- Created a big title called "Online Store" using `<h1>` font.
- Created an input field labeled "Name" and added a "Login" button for logging into the web application.
- Added a placeholder inside the input field saying "Enter Your Name".

CSS Functionalities:

- The title "Online Store" was created in bold, with a 3D effect using proper shadows.
- A centered container was created for logging in with a "Name" input field and a "Login" button colored in blue.
- Different parts of the layout are differentiated by minor color variations.

JS Functionalities:

- Users are restricted to entering only alphabetic characters in the input field.
- If the user does not enter anything in the input field, an error message "Please Enter Your Name" will be displayed instantly.

Products List page(Page 2):

JSP and HTML Functionalities:

- "Welcome, ${username}" is added with the username displayed in the top right corner.
- Products are listed, each with a product image, product name, product price, and description. An "Open" button has been created for all products.

CS Functionalities:

- A box type with rounded edges has been added to "Welcome, ${username}", with the username displayed in the top right corner.
- Images have been added for all products, and the price text is now displayed in blue. The "Open" button color is green.
- A 3D shadow look has been added to each product's round-edged container to differentiate it from the background.

Product Purchasing Page(Page 3):

HTML and CSS Functionalities:

- This page displays the product that the user selects using the "Open" button from the previous page (product list page).
- In this page, a Product Title is added with bold `<h1>` formatting, Product Price is displayed in orange color, added Description of the product.
- The HD+ Product Image is enclosed in a container with rounded edges and a 3D shadow effect.
- Finally, a "Buy Now" button in green color is added to the bottom right corner for the user to purchase the product.
- When the user's cursor is placed on the "Buy Now" button, it hovers in dark green.

Collecting User's Order Details Page(Page 4):

JSP , HTML ,CSS Functionalities:

- In this page, the displayed Product Name, Product Price, and Username (retrieved from the user's entry on the homepage) are added with `<h1>` bold font texts.
- Input fields for "Mobile Number" and "Address" have been added with `<h1>` bold font, along with placeholders "Enter your mobile number" and "Enter your full address".
- "Select Payment Method" has been added for choosing between payment options "Cash On Delivery" or "Online Payment", displayed in `<h1>` bold font.
- In Credit or Debit Card Number input field added placeholder “Enter 16-digit card number”.
- A blue color "Place Order" big button has been added. If a user hovers over it, the color changes to green.

JavaScript Functionalities:

- Users are restricted to entering only 10 digits numeric characters in the mobile number input field. Non-numeric characters are not permitted.
- "Please enter exactly 10 digits for the mobile number." This error message will be displayed in red color if the user attempts to input anything other than 10 digits.
- “Please Enter Your Complete Address”.This error message will be displayed in red if the user tries to submit an without address.

- If "onlinePayment" is selected on “Select Payment Method”, it displays card details. Otherwise, it hides the card details and clears the card number input field for "Cash on Delivery" selection.

- The input field “Credit or Debit Card Number” to ensure it contains precisely 16 digits. If not, it displays the error message "Please enter exactly 16 digits for the credit/debit card number" and prevents form submission; otherwise, it allows submission.
- Added a space after every 4 characters for readability.

Order Placed Page(Page 5):

HTML ,CSS Functionalities:

- **"Your Order Placed Successfully"**
  ✔ Text added in big bold green font color with a check symbol indicating completion.
- Added “Thank you for purchasing from our online store” on next line.
- Two buttons have been created:

"Cancel Order"
"Continue Shopping"

The **"Cancel Order"** button changes color to red when hovered over.

- Added Dark Blue Color on Background.

Order Cancellation and Credit or Debit card Invalid page(Page 6 and page 7):

- **"Your Order Cancelled"**
  Text added in large red font on Order Cancellation page.
- Created a **"Continue Shopping"** button at the bottom.
  When the user moves the cursor over it, the button changes to green on Order Cancellation page.

- Added a cross symbol with the text "**Your card number is incorrect. Please check your card number.**" in big red font.
- Added the text "**Thank you for purchasing from our online store.**"
- Added a "**Try Again**" button in green color on the next line.
- The background color is set to dark blue.

Technical Documentation

Created Totally 9 classes for code to "Online Store(E-Commerce)".

1. An “User_Details” class has been created to store UserName.

2. The “User_Details_CRUD” interface created for Save the UserName on Database.

3. The "Purchase_OrderDetails" class created for Store the User giving Order Details.

4. “purchase_orderDetailsCRUD” interface created for Save the User giving Order Details on Database.

5. Created class “Credit_Card_Digits” to store User Credit or Debit Card Number.

6. Developed a "StringToArray_CreditCard" class to convert string characters to arrays, then passed to "Credit_or_DebitCard_Validation" class.

7.Separated credit/debit card validation into " Credit_or_DebitCard_Validation" class.

8.This `OnlineStoreApplication` class is crucial because it contains the `main` method, which serves as the entry point for your Spring Boot application. The `@SpringBootApplication` annotation enables auto-configuration and component scanning, simplifying the setup and initialization of your application.

9.All classes are linked together within the "Store_Controller" (Controller Class).

Maven Dependencies:

- **Apache Tomcat Jasper:** Provides JSP (JavaServer Pages) and JSTL (JavaServer Pages Standard Tag Library) support.
- **Spring Boot Starter Data JPA**: Starter for utilizing Spring Data JPA, simplifying data access with JPA (Java Persistence API).
- **Spring Boot Starter Web:** Starter for developing web applications using Spring MVC (Model-View-Controller).
- **H2 Database:** Embedded relational database often used for development and testing purposes.
- **Spring Boot Starter Test:** Starter for testing Spring Boot applications, supporting unit and integration testing.

Here clear details about all the class code,

Package Name is "Store".

- This package contains classes related to the ONLINE STORE (E-Commerce) application.

`User_Details Class:`

Description

The `User_Details` class is a Spring component representing a database entity mapped to the `user_details` table. It stores user details and manages the username field as the primary key.

Annotations

- `@Component`: Marks the class as a Spring component, allowing it to be automatically detected and configured by Spring's component scanning mechanism.
- `@Entity`: Indicates that instances of this class are entities, representing rows in the corresponding database table.
- `@Table(name = "user_details")`: Specifies the name of the database table (`user_details`) to which this entity is mapped.

Fields

- `UsernamE`: Private field representing the username, which is the primary key for the `user_details` table.

Methods

- `getUsernamE()`: Getter method that retrieves the username stored in the `UsernamE` field.
- `setUsernamE(String usernamE)`: Setter method that sets the value of the `UsernamE` field to the provided username.

Conclusion

The `User_Details` class encapsulates user-related data and is integral for interacting with user details in the database. It leverages Spring annotations for component scanning and JPA annotations for entity mapping, ensuring seamless integration within the Spring Boot application.

User_Details_CRUD Interface:

Description

The `User_Details_CRUD` interface is a Spring repository interface that extends `CrudRepository`. It provides CRUD (Create, Read, Update, Delete) operations for the `User_Details` entity, allowing easy interaction with user details in the database.

Annotations

- `@Repository`: Marks the interface as a Spring repository, allowing it to be automatically detected and configured by Spring's component scanning mechanism.

Extends

- `CrudRepository<User_Details, String>`: Extends the Spring `CrudRepository` interface, which provides generic CRUD operations for the `User_Details` entity. The type parameters `<User_Details, String>` specify the entity type (`User_Details`) and the type of its primary key (`String`).

Conclusion

The `User_Details_CRUD` interface simplifies database operations for the `User_Details` entity by providing pre-defined CRUD methods inherited from `CrudRepository`. It leverages Spring's repository pattern to abstract away low-level database interactions, promoting cleaner and more maintainable code within the Spring Boot application.

Purchase_OrderDetails Class:

Description

The `Purchase_OrderDetails` class represents order details mapped to the `order_details` database table. It stores information about user orders including product details, user information, and shipping address.

Annotations

- `@Component`: Marks the class as a Spring component, allowing it to be automatically detected and configured by Spring's component scanning mechanism.

- `@Entity`: Indicates that instances of this class are entities, representing rows in the corresponding database table.
- `@Table(name = "order_details")`: Specifies the name of the database table (`order_details`) to which this entity is mapped.

Fields

- `S_no`: Private field representing the unique serial number for order details. Annotated with `@Id` to designate it as the primary key of the `order_details` table.
- `User_Name`: Private field representing the username of the user placing the order.
- `productName`: Private field representing the name of the product being ordered.
- `productPrice`: Private field representing the price of the product.
- `mobileNumber`: Private field representing the mobile number of the user placing the order.
- `address`: Private field representing the shipping address for the order.

Methods

- `getS_no()`: Getter method that retrieves the serial number (`S_no`) of the order.
- `setS_no(int s_no)`: Setter method that sets the serial number (`S_no`) of the order.
- `getUser_Name()`: Getter method that retrieves the username of the user placing the order.
- `setUser_Name(String user_Name)`: Setter method that sets the username of the user placing the order.
- `getProductName()`: Getter method that retrieves the name of the product being ordered.
- `setProductName(String productName)`: Setter method that sets the name of the product being ordered.
- `getProductPrice()`: Getter method that retrieves the price of the product being ordered.
- `setProductPrice(String productPrice)`: Setter method that sets the price of the product being ordered.
- `getMobileNumber()`: Getter method that retrieves the mobile number of the user placing the order.

- `setMobileNumber(String mobileNumber)`: Setter method that sets the mobile number of the user placing the order.
- `getAddress()`: Getter method that retrieves the shipping address for the order.
- `setAddress(String address)`: Setter method that sets the shipping address for the order.

Conclusion

The `Purchase_OrderDetails` class encapsulates order-related data and serves as a representation of order details within the application. It leverages Spring annotations for component scanning and JPA annotations for entity mapping, facilitating seamless integration with the underlying database in the Spring Boot application.

---

purchase_orderDetailsCRUD Interface:

Description

The `purchase_orderDetailsCRUD` interface is a Spring repository interface that extends `CrudRepository`. It provides CRUD (Create, Read, Update, Delete) operations for the `Purchase_OrderDetails` entity, allowing easy interaction with purchase order details in the database.

Annotations

- `@Component`: Marks the interface as a Spring component, allowing it to be automatically detected and configured by Spring's component scanning mechanism.

Extends

- `CrudRepository<Purchase_OrderDetails, Integer>`: Extends the Spring `CrudRepository` interface, which provides generic CRUD operations for the `Purchase_OrderDetails` entity. The type parameters `<Purchase_OrderDetails, Integer>` specify the entity type (`Purchase_OrderDetails`) and the type of its primary key (`Integer`).

Methods

The `purchase_orderDetailsCRUD` interface used ,

- save() method to saving the Order Details

Conclusion

The `purchase_orderDetailsCRUD` interface provides CRUD (Create, Read, Update, Delete) operations for managing `Purchase_OrderDetails` entities within the Spring Boot application. By extending `CrudRepository`, it simplifies database interactions and promotes efficient data access.

Credit_Card_Digits Class:

Description

The `Credit_Card_Digits` class represents a utility for handling credit card digits, specifically providing functionality to remove spaces from a string of digits.

Fields

- `digits`: Private field to store the credit card digits.

Methods

- `getDigits()`: Getter method that retrieves the stored credit card digits.
- `setDigits(String digits)`: Setter method that sets the credit card digits.
- `removeSpace(String strNumber)`: Method to remove spaces from a given string of digits (`strNumber`).

  Parameters:

  - `strNumber`: String containing credit card digits possibly separated by spaces.

  Returns:

  - `String`: Returns a new string containing the credit card digits without any spaces.

Implementation Details:

- Iterates through each character of the input string (`strNumber`).
- Appends non-space characters to a `StringBuilder`.
- Returns the resulting string representation of the `StringBuilder`.

Conclusion

The `Credit_Card_Digits` class provides basic functionality to handle credit card digits, allowing the removal of spaces from a string of digits. This utility can be used to preprocess credit card information before further processing or validation within the application.

StringToArray_CreditCard Class:

Description

The `StringToArray_CreditCard` class provides a utility method to convert a string of digits into an integer array representing each digit.

Annotations

- `@Component`: Marks the class as a Spring component, allowing it to be automatically detected and instantiated by the Spring framework.

Variables

- `number`: Private variable of type `String` that stores the input string of digits to be converted.

Methods

- `convertStringToIntArray(String number)`: Method that converts a string of digits (`number`) into an integer array.

  Parameters:

  - `number`: String containing digits to be converted into an integer array.

  Returns:

  - `int[]`: Returns an integer array containing the converted digits.

  Implementation Details:

- Creates an integer array (`Aray`) with a length equal to the length of the input string (`number`).
- Iterates through each character of the input string.
    - Retrieves the character at the current index (`digitChar`).
    - Converts the character to its numeric value using `Character.getNumericValue(digitChar)`.
    - Stores the numeric value in the corresponding index of the integer array (`Aray`).

Conclusion

The `StringToArray_CreditCard` class provides a useful method for converting a string of digits into an integer array, which can be used for processing credit card information or similar numerical data within the Spring Boot application. This utility enhances data handling capabilities by facilitating the conversion and manipulation of numeric strings

---

Credit_or_DebitCard_Validation Class:

Description

The `Credit_or_DebitCard_Validation` class is a Spring component that provides functionality to validate credit or debit card numbers.

Algorithm: Luhn Algorithm.

Annotations

- `@Component`: Marks the class as a Spring component, allowing it to be automatically detected and instantiated by the Spring framework.

Dependencies

- `StringToArray_CreditCard StringToArray`: Autowired dependency injection for the `StringToArray_CreditCard` bean, used for converting strings of digits into integer arrays.

Variables

- `just_a_variable`: Temporary variable used during card number validation.

Methods

- `valida(Credit_Card_Digits orderDE)`: Method that validates a credit or debit card number using the Luhn Algorithm.

  Parameters:

  - `orderDE`: An instance of `Credit_Card_Digits` containing the credit card digits to be validated.

  Returns:

  - `int`: Returns the total sum calculated during the validation process.

  Implementation Details:

  - Removes spaces from the input credit card digits using the `removeSpace` method from `Credit_Card_Digits`.
  - Converts the cleaned digits into an integer array using the `convertStringToIntArray` method from `StringToArray_CreditCard`.
  - Implements the Luhn Algorithm for card validation:
    - Doubles every second digit of the card number.
    - Splits and adds the digits if the doubled digit is greater than or equal to 10.
    - Calculates the total sum (`gsum`) of the processed card number digits.

Conclusion

The `Credit_or_DebitCard_Validation` class enhances the Spring Boot application by providing a card validation utility based on the Luhn Algorithm. It leverages dependency injection (`Autowired`) for collaborating with other components (`StringToArray_CreditCard`) to perform string manipulation and conversion tasks efficiently.

Certainly! Let's create technical documentation for the `OnlineStoreApplication` class, which serves as the main Spring Boot application class for your online store application:

OnlineStoreApplication Class:

Description

The `OnlineStoreApplication` class is the main Spring Boot application class for the online store application. It utilizes Spring Boot's `@SpringBootApplication` annotation to bootstrap the application and start the Spring framework.

Annotations

- `@SpringBootApplication`: An annotation that combines `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan` annotations, indicating that this class is the main configuration class for the Spring Boot application. It enables Spring Boot's auto-configuration mechanism and component scanning.

Methods

- `main(String[] args)`: The main method, which serves as the entry point of the application.

  Parameters:

  - `args`: Command-line arguments passed to the application (not used in this example).

  Functionality:

  - Uses `SpringApplication.run()` to start the Spring Boot application.
  - The `run()` method takes two arguments:
    - `OnlineStoreApplication.class`: Specifies the main application class.
    - `args`: Command-line arguments passed to the application.

Usage

The `OnlineStoreApplication` class initializes and starts the Spring Boot application, triggering the auto-configuration process and enabling component scanning. This class is responsible for launching the online store application and initializing the Spring context.

Conclusion

The `OnlineStoreApplication` class plays a crucial role as the entry point and configuration class for the Spring Boot-based online store application. It leverages Spring Boot's features to simplify application setup and provide a robust foundation for building and running the online store.

Store_Controller Class:

Description

The `Store_Controller` class serves as a Spring MVC controller for handling various HTTP requests related to the online store application, including user login, product display, order processing, and more.

Annotations

- `@Controller`: An annotation that identifies this class as a Spring MVC controller, allowing it to handle incoming HTTP requests.
- `@Autowired`: Used to automatically inject dependencies into the controller.

Dependencies

- `User_Details_CRUD user_detailsCrud`: Repository for managing user details.
- `purchase_orderDetailsCRUD orderDetails_Crud`: Repository for managing purchase order details.
- `Credit_or_DebitCard_Validation creditCard_Validation`: Service for validating credit/debit card numbers.
- `Purchase_OrderDetails OrderDetail`: Object for holding current user's order details.

Variables

- `serielNumber`: Integer variable to maintain the serial number for orders.

Methods:

`LoginPage()`

- **Mapping:** `@GetMapping("onlinestore")`
- **Purpose:** Displays the login page.
- **Return Type:** `String`
- **View:** Returns the login page HTML.

`productList(String username, User_Details userDetail)`

- **Mapping:** `@GetMapping("Login")`
- **Purpose:** Processes user login and displays the product list.
- **Parameters:**
  - `username`: Username from the login form.
  - `userDetail`: Instance of `User_Details` containing user details.
- **Return Type:** `ModelAndView`
- **View:** Displays the product list page (`productsList.jsp`).
- **Action:** Saves user details to the database.

Product Display Handlers (`product1`, `product2`, ..., `product9`)

- **Mapping:** `@GetMapping`
- **Purpose:** Displays specific product pages.
- **Return Type:** `String`
- **View:** Returns HTML page corresponding to the requested product.

`purchase(String productCode)`

- **Mapping:** `@GetMapping("buying")`
- **Purpose:** Processes product purchase.
- **Parameters:**
  - `productCode`: Code identifying the product to be purchased.
- **Return Type:** `ModelAndView`
- **View:** Displays the order details page (`OrderDetails.jsp`).
- **Action:** Validates credit/debit card and saves order details to the database.

`placeOrder(String productName, String productPrice, Purchase_OrderDetails orderS_detail, Credit_Card_Digits ccd, String digits)`

- **Mapping:** `@GetMapping("placeorder")`
- **Purpose:** Handles order placement.
- **Parameters:**
  - `productName`: Name of the product being purchased.
  - `productPrice`: Price of the product being purchased.
  - `orderS_detail`: Instance of `Purchase_OrderDetails` containing order details.
  - `ccd`: Instance of `Credit_Card_Digits` containing credit/debit card digits.
  - `digits`: String representing credit/debit card digits.

- **Return Type:** `String`
- **View:** Returns HTML page for successful order placement or error.
- **Action:** Validates credit/debit card using `creditCard_Validation` service and saves order details to the database.

`CONTINUE(User_Details userDetail)`

- **Mapping:** `@GetMapping("ContinueShopping")`
- **Purpose:** Continues shopping after login.
- **Parameters:**
  - `userDetail`: Instance of `User_Details` containing user details.
- **Return Type:** `ModelAndView`
- **View:** Displays the product list page (`productsList.jsp`).
- **Action:** Saves updated user details to the database.

`CancelOrder(Purchase_OrderDetails order)`

- **Mapping:** `@GetMapping("CancelOrder")`
- **Purpose:** Cancels a previously placed order.
- **Parameters:**
  - `order`: Instance of `Purchase_OrderDetails` representing the order to be canceled.
- **Return Type:** `ModelAndView`
- **View:** Displays confirmation of order deletion.
- **Action:** Deletes the specified order from the database.

Conclusion

The `Store_Controller` class plays a pivotal role in managing user interactions and business logic within the online store application. It handles user login, product display, order processing, and cancellation operations, integrating with various repositories and services to ensure smooth application flow. By leveraging Spring MVC annotations and dependency injection, this controller effectively manages the application's web layer.

User Documentation

How To Use This Web Application:

- First, run the “ONLINE STORE (E-Commerce)” application.
- Then navigate to “localhost:8080/onlinestore” in your browser; you will be directed to the Home Page of the Online Store web application.
- Login with your Name and Select any Product(Open) and then Buy the any product what u want to purchase.
- Fill the Order Details for Place Order.
- If you select cash on delivery for your purchase, the "Your Order Placed Successfully" page will be displayed.
- If you want to cancel the order, you can easily do so by clicking the "Cancel Order" button.
- If you wish to buy another product, you can continue shopping by clicking the "Continue Shopping" button.
- Suppose you are trying to purchase by online payment. Enter your 16-digit credit or debit card number.
- If your card number is valid, it will display "Your Order Placed Successfully".
- If your card number is invalid, it will display “Your card number is incorrect. Please check your card number” and you can try again by clicking the “Try Again” button.