**ShopKaro Online Shopping Application Documentation**

**Introduction**

ShopKaro is an online shopping application developed for ABC Company. The application is designed to provide a simple and convenient platform for customers to browse and purchase products, and for admins to manage the products in the store. With a user-friendly interface and a range of features, ShopKaro makes online shopping easy and accessible for everyone.

The application is built using Java Core, MySql, Maven, and SpringBoot, and follows the Model-View-Controller (MVC) architecture. It uses a relational database to store and retrieve data, and implements secure authentication and authorization for users. The codebase is structured using POJO classes and interfaces to provide separation between the business logic and the controller layer.

The application consists of several modules, including the customer, product, order, cart, and login modules. The customer module allows users to register, log in, browse and search products, add products to their cart, edit their cart, check out and purchase products, and view their order history. The product module enables admins to search for products, view product details and images, add new products to the database, update existing products, and remove products from the database. The order module allows customers to create and manage orders, view their order history and details, and receive order confirmation emails. The cart module allows customers to add products to their cart, remove products from their cart, update the quantity of products in their cart, and view a list of products in their cart. The login module provides secure user authentication and authorization, as well as password reset functionality.

This documentation provides a comprehensive guide to the ShopKaro Online Shopping Application, including detailed information about its features, architecture, modules, database, testing, and deployment. Whether you are a customer looking to shop online, or an admin looking to manage the products in the store, this documentation will help you get started with the application and make the most of its features.

**Getting Started**

This section provides a guide to getting started with the ShopKaro Online Shopping Application. Before you begin, please make sure that you have the following prerequisites installed:

1. Java Development Kit (JDK) version 8 or higher

Maven build tool version 3.2 or higher

MySQL relational database management system version 5.6 or higher

1. Installing and Running the Application
2. Clone the ShopKaro project repository from GitHub using the following command:

“git clone https://github.com/ABCCompany/ShopKaro.git”

1. Create a MySQL database and configure the database settings in the application.properties file located in the src/main/resources directory. The default database settings are as follows:

“spring.datasource.url=jdbc:mysql://localhost:3306/shopkaro

spring.datasource.username=root

spring.datasource.password=root”

Please make sure to replace the database URL, username, and password with your own database settings.

Please make sure to configure the database settings and other environment-specific settings in the application.properties file before deploying the application to a production environment.

**Architecture**

The ShopKaro Online Shopping Application is built using the Model-View-Controller (MVC) architecture pattern. The application is developed using Java Core, Spring Boot, and MySQL.

The MVC architecture pattern separates the application into three main components: the Model, the View, and the Controller. This separation of concerns allows for easier development, maintenance, and testing of the application.

Controller

The Controller acts as an intermediary between the Model and the View. It handles user requests, processes input data, and updates the Model and the View accordingly. In the ShopKaro application, the Controller is implemented using Spring MVC, which provides a robust and flexible framework for building web applications.

Database

The ShopKaro application uses MySQL as its relational database management system. The database stores and retrieves data related to Customers, Products, Orders, and Carts. The database is accessed using the Spring Data JPA framework, which provides a powerful and easy-to-use interface for working with databases.

Overall, the ShopKaro Online Shopping Application is designed to be a scalable and maintainable web application that follows industry-standard architecture patterns and best practices.

**Modules**

1. Login Module: This module allows users to register, log in, and log out of the application. It includes functionality for secure user authentication and authorization, as well as password reset functionality.
2. Customer Module: This module enables customers to browse, search for, and purchase products in the store. It includes functionality for adding products to a cart, editing cart contents, and checking out. It also includes functionality for viewing order history and details.
3. Product Module: This module enables admins to manage the products in the store. It includes functionality for searching for products, viewing product details and images, adding new products to the database, updating existing products, and removing products from the database.
4. Order Module: This module enables customers to create and manage orders. It includes functionality for viewing order history and details, as well as generating order confirmation emails.
5. Cart Module: This module allows customers to view and manage the products in their cart. It includes functionality for adding products to the cart, removing products from the cart, updating the quantity of products in the cart, and removing all products from the cart.

**Customer Module**

To add a new product to the database:

* Endpoint: POST /productSave
* Request Body: JSON object containing the product details, for example:

{

"name": "Product Name",

"description": "Product description",

"price": 9.99,

"image": "https://example.com/product-image.jpg",

"category": "Category name"

}

* Response: JSON object containing the saved product details, including the generated product ID.
* To update an existing product:
* Endpoint: PUT /productUpdate/{productId}
* Request Body: JSON object containing the updated product details, for example:

{

"name": "New Product Name",

"description": "Updated product description",

"price": 12.99,

"image": "https://example.com/new-product-image.jpg",

"category": "New Category name"

}

* Response: JSON object containing the updated product details.
* To delete a product from the database:
* Endpoint: DELETE /productDelete/{productId}
* Response: HTTP 200 OK with no response body.
* To retrieve a specific product from the database:
* Endpoint: GET /productGet/{productId}
* Response: JSON object containing the product details.
* To retrieve all products in the database:
* Endpoint: GET /products
* Response: JSON array containing all products in the database, each represented as a JSON object.
* Note: Replace {productId} with the ID of the product you want to perform the operation on.

**Product Module**

View all products

* Retrieves all products in the system.
* URL: /products
* Method: GET
* URL Params: None
* Success Response:
* Code: 200 OK <br />

Content:

[ { "productId": 1,

"productName": "Product 1",

"price": 10.0, "description": "Description 1",

"manufacturer": "Manufacturer 1",

"quantity": 5, "category": { "catId": 1, "catName": "Category 1" } },

{ "productId": 2, "productName": "Product 2",

"price": 20.0, "description": "Description 2",

"manufacturer": "Manufacturer 2",

"quantity": 10,

"category": { "catId": 2,

"catName": "Category 2" } }]

* View a specific product
* Retrieves a specific product by ID.
* URL: /products/{id}
* Method: GET
* URL Params:
* id: The ID of the product to retrieve.
* Success Response:
* Code: 200 OK <br />

Content:

{

"productId": 1,

"productName": "Product 1",

"price": 10.0,

"description": "Description 1",

"manufacturer": "Manufacturer 1",

"quantity": 5,

"category": {

"catId": 1,

"catName": "Category 1"

}

}

* Add a new product
* Adds a new product to the system.
* URL: /products
* Method: POST
* URL Params: None

Request Body:

{

"productName": "Product 3",

"price": 15.0,

"description": "Description 3",

"manufacturer": "Manufacturer 3",

"quantity": 7,

"category": {

"catId": 1

}

}

* Success Response:
* Code: 200 OK <br />

Content:

{

"productId": 3,

"productName": "Product 3",

"price": 15.0,

"description": "Description 3",

"manufacturer": "Manufacturer 3",

"quantity": 7,

"category": {

"catId": 1,

"catName": "Category 1"

}

}

* Update an existing product
* Updates an existing product in the system.
* URL: /products
* Method: PUT
* URL Params: None
* Request Body:

{

"productId": 3,

"productName": "Product 3",

"price": 20.0,

"description": "Description 3",

"manufacturer": "Manufacturer 3",

"quantity": 10,

"category": {

"catId": 2

}

}

* Success Response:
* Code: 200 OK <br />

Content:

PUT /products?sessionKey=<valid-session-key>

Content-Type: application/json

{

"productId": 1,

"productName": "Updated Product",

"price": 100.00,

"description": "This is an updated product.",

"manufacturer": "Updated Manufacturer",

"quantity": 50,

"category": {

"categoryId": 1,

"categoryName": "Updated Category"

}

}

* HTTP/1.1 200 OK
* Content-Type: application/json

{

"productId": 1,

"productName": "Updated Product",

"price": 100.00,

"description": "This is an updated product.",

"manufacturer": "Updated Manufacturer",

"quantity": 50,

"category": {

"categoryId": 1,

"categoryName": "Updated Category"

}

}

**Order Module**

This API allows you to manage orders for a shopping application.

Base URL: /orders

* HTTP Method: POST

URL: /{key}

Description: Add a new order

Request Body: JSON representation of Orders object

**{**

**"orderId": 0,**

**"orderDate": "2023-05-07",**

**"orderStatus": "string",**

**"product": [**

**{**

**"productId": 0,**

**"productName": "string",**

**"price": 0,**

**"description": "string",**

**"manufacturer": "string",**

**"quantity": 0,**

**"category": {**

**"catId": 0,**

**"catName": "GROCERY"**

**}**

**}**

**],**

**"customer": {**

**"customerId": 0,**

**"firstName": "string",**

**"lastName": "string",**

**"mobileNumber": "8836667445",**

**"email": "string",**

**"password": "string",**

**"address": {**

**"addressId": 0,**

**"streetName": "string",**

**"buildingName": "string",**

**"city": "string",**

**"state": "string",**

**"country": "string",**

**"pincode": "string"**

**}**

**},**

**"address": {**

**"addressId": 0,**

**"streetName": "string",**

**"buildingName": "string",**

**"city": "string",**

**"state": "string",**

**"country": "string",**

**"pincode": "string"**

**}**

**}**

Response Body: JSON representation of the created Orders object with the assigned order id

* HTTP Method: PUT

URL: /{key}

Description: Update an existing order

Request Body: JSON representation of Orders object

**{**

**"orderId": 0,**

**"orderDate": "2023-05-07",**

**"orderStatus": "string",**

**"product": [**

**{**

**"productId": 0,**

**"productName": "string",**

**"price": 0,**

**"description": "string",**

**"manufacturer": "string",**

**"quantity": 0,**

**"category": {**

**"catId": 0,**

**"catName": "GROCERY"**

**}**

**}**

**],**

**"customer": {**

**"customerId": 0,**

**"firstName": "string",**

**"lastName": "string",**

**"mobileNumber": "9299598379",**

**"email": "string",**

**"password": "string",**

**"address": {**

**"addressId": 0,**

**"streetName": "string",**

**"buildingName": "string",**

**"city": "string",**

**"state": "string",**

**"country": "string",**

**"pincode": "string"**

**}**

**},**

**"address": {**

**"addressId": 0,**

**"streetName": "string",**

**"buildingName": "string",**

**"city": "string",**

**"state": "string",**

**"country": "string",**

**"pincode": "string"**

**}**

**}**

Response Body: JSON representation of the updated Orders object

* HTTP Method: DELETE

URL: /{orderId}/{key}

Description: Delete an existing order by order ID

Response Body: JSON representation of the deleted Orders object

* HTTP Method: GET

URL: /orders

Description: Get a list of all orders

Response Body: JSON representation of the list of Orders object

* HTTP Method: GET

URL: /ordersId/{orderId}

Description: Get an order by order ID

Response Body: JSON representation of the Orders object with the specified order ID

* HTTP Method: GET

URL: /ordersDate/{date}

Description: Get a list of orders by order date

Response Body: JSON representation of the list of Orders object with the specified order date

* HTTP Method: GET

URL: /ordersCity/{city}

Description: Get a list of orders by order city

Response Body: JSON representation of the list of Orders object with the specified order city

Note: The {key} parameter in the URL is used for authorization purposes. It should be provided in the header of the request.

**Cart Module**

1. Add a product to the cart:

To add a product to the cart, make a POST request to the endpoint /carts/addProduct, with the following parameters:

cartId (required): the ID of the cart to which you want to add the product.

productId (required): the ID of the product you want to add to the cart.

Example:

POST /carts/addProduct?cartId=1&productId=123

1. Remove a product from the cart:

To remove a product from the cart, make a DELETE request to the endpoint /carts/removeProduct, with the following parameters:

cartId (required): the ID of the cart from which you want to remove the product.

productId (required): the ID of the product you want to remove from the cart.

Example:

DELETE /carts/removeProduct?cartId=1&productId=123

1. Update the quantity of a product in the cart:

To update the quantity of a product in the cart, make a PATCH request to the endpoint /carts/updateProduct, with the following parameters:

cartId (required): the ID of the cart in which the product is present.

productId (required): the ID of the product you want to update the quantity for.

quantity (required): the new quantity of the product in the cart.

Example:

PATCH /carts/updateProduct?cartId=1&productId=123&quantity=2

1. Remove all products from the cart:

To remove all products from the cart, make a GET request to the endpoint /carts/emptyCart, with the following parameter:

cartId (required): the ID of the cart from which you want to remove all the products.

Example:

GET /carts/emptyCart?cartId=1

* View all products in the cart:

To view all products in the cart, make a GET request to the endpoint /carts/viewProducts, with the following parameter:

cartId (required): the ID of the cart you want to view the products for.

Example:

GET /carts/viewProducts?cartId=1

* Error handling:

The API handles errors related to the cart operations. If an error occurs, the API returns an HTTP response with an appropriate error message.

Example:

json

{

"timestamp": "2023-05-07T08:21:34.219+00:00",

"status": 400,

"error": "Bad Request",

"message": "Product with ID 123 not found",

"path": "/carts/addProduct"

}

**Login Module**

**--login area----**

**---login area---**

**Database**

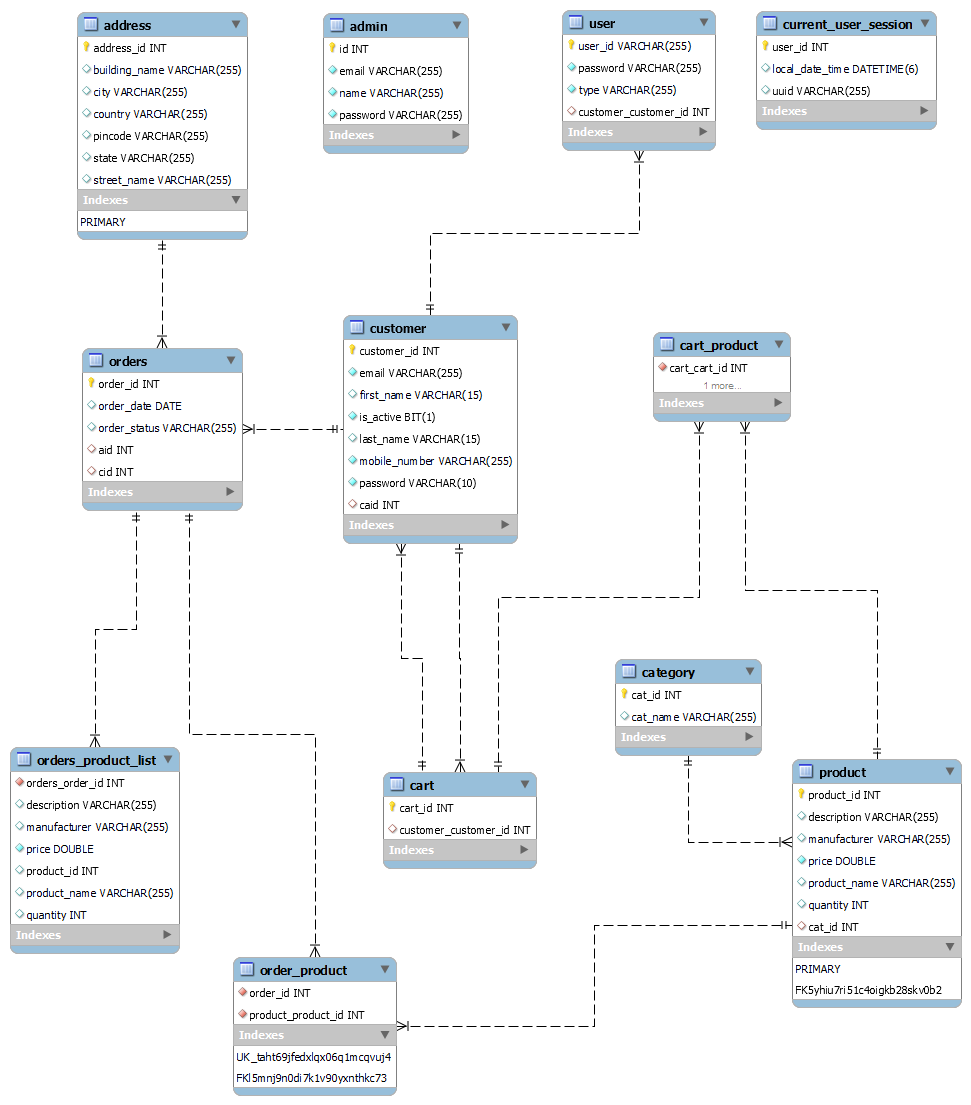
The Customer table stores information about the registered customers including their unique customer ID, name, email address, password, and shipping address. The Admin table stores information about the admins including their unique admin ID, name, email address, and password.

The Product table stores information about the products in the store including the product ID, name, description, price, and image. The Order table stores information about the orders including the order ID, customer ID, product ID, quantity, and total price. The Cart table stores information about the products in the customer's cart including the customer ID, product ID, and quantity.

The database is designed to ensure data consistency and integrity by using foreign key constraints to link the tables together. For example, the Order table has a foreign key constraint on the Customer and Product tables to ensure that only valid customer and product IDs are used in the order.

To interact with the database, the application uses Spring Data JPA which provides an easy-to-use interface for performing CRUD (Create, Read, Update, Delete) operations on the database tables. The application also uses Hibernate as the Object-Relational Mapping (ORM) framework to map the POJO classes to the database tables.

Overall, the use of a relational database in the Online Shopping Application provides a scalable and reliable solution for managing the data of customers, products, orders, and carts.



**Admin Module**

The Admin Controller is responsible for handling all the requests made by the admin for managing the products and categories in the ShopKaro Online Shopping Application. The controller communicates with the AdminService, ProductService, and CategoryService to perform the required operations.

Endpoints:

* GET "/admin/products"

This endpoint is used to fetch all the products from the database. It returns a list of products in the response body with HTTP status OK (200).

* GET "/admin/products/{id}"

This endpoint is used to fetch a specific product by its ID. It takes an ID as a path variable and returns the product in the response body with HTTP status OK (200).

* POST "/admin/products"

This endpoint is used to add a new product to the database. It takes a JSON object of the Product model as a request body and a session key as a request parameter. It returns the newly added product in the response body with HTTP status OK (200).

* PUT "/admin/products"

This endpoint is used to update an existing product in the database. It takes a JSON object of the Product model as a request body and a session key as a request parameter. It returns the updated product in the response body with HTTP status OK (200).

* DELETE "/admin/products/{id}"

This endpoint is used to delete a product from the database. It takes an ID as a path variable and a session key as a request parameter. It returns the deleted product in the response body with HTTP status OK (200).

* GET "/admin/category"

This endpoint is used to fetch all the categories from the database. It returns a list of categories in the response body with HTTP status OK (200).

* GET "/admin/category/name"

This endpoint is used to fetch all the products of a specific category. It takes a category name as a request parameter and returns a list of products in the response body with HTTP status OK (200).

* POST "/admin/category"

This endpoint is used to add a new category to the database. It takes a JSON object of the Category model as a request body and a session key as a request parameter. It returns the newly added category in the response body with HTTP status OK (200).

* DELETE "/admin/category/{id}"

This endpoint is used to delete a category from the database. It takes an ID as a path variable and a session key as a request parameter. It returns the deleted category in the response body with HTTP status OK (200).

Note: The session key is used for authentication and authorization purposes.

Dependencies:

AdminService: A service class that contains business logic for managing admin-related operations.

ProductService: A service class that contains business logic for managing product-related operations.

CategoryService: A service class that contains business logic for managing category-related operations.

Model:

Product: A model class that represents a product entity in the database.

Category: A model class that represents a category entity in the database.

Example of Model:

**[**

**{**

**"productId": 0,**

**"productName": "string",**

**"price": 0,**

**"description": "string",**

**"manufacturer": "string",**

**"quantity": 0,**

**"category": {**

**"catId": 0,**

**"catName": "GROCERY"**

**}**

**}**

**]**

**Deployment**

This section should provide instructions for deploying the application to a production environment, including any necessary configuration steps and best practices for security and performance.

**Conclusion**

I hope this documentation provides you with a clear understanding of the ShopKaro Online Shopping Application. If you have any questions or require further clarification, please do not hesitate to reach out. I am here to assist you in any way I can. Thank you for considering this documentation.