|  |
| --- |
|  |
| Technical solution description  Information system of online store  MMS  by Popov Alexander |
|  |

Content

[**Overview** 3](#_Toc39671772)

[**Used technologies and frameworks** 4](#_Toc39671773)

[**Additional features** 5](#_Toc39671774)

[**Database scheme** 6](#_Toc39671775)

[**Explanation of model implementation** 8](#_Toc39671776)

[**Application modules** 9](#_Toc39671777)

[**UI layer description** 10](#_Toc39671778)

[**Business layer description** 11](#_Toc39671779)

[**Data layer description** 12](#_Toc39671780)

[**Common interaction between application layers** 12](#_Toc39671781)

[**Application screenshots** 13](#_Toc39671782)

[**Unit tests** 18](#_Toc39671784)

[**Deployment** 19](#_Toc39671785)

[**Logging configuration** 20](#_Toc39671786)

[**Application improvements** 21](#_Toc39671787)

**Overview**

Information system of online store is a multi-user client-server application. The application provides different options depending on client’s role: user or employee. Employees have the ability to manage store’s categories and products, track the orders. Users can browse the store’s directory and purchase goods. Both users and employees are allowed to edit their accounts.

Application also has a service that provides information about top products and sends it to another application.

1. **Used technologies and frameworks**

* JDK 1.8
* Apache Tomcat
* Spring framework
* Spring security
* JPA, Hibernate framework
* MySQL
* Jackson
* Maven
* Log4j
* JUnit
* Mockito
* Bean validation API
* JSP, JSTL
* EJB
* JSF
* JMS
* Java Mail
* AS Wildfly
* Flyway DB migration
* Active MQ
* Sonar scanner

**Additional features**

In addition to the required features the following features has been implemented:

* Dynamicly added fileds in form using javascript.
* Ability to upload and save images for account, category and product view.
* Simulation of payment system.
* Ability to subscribe and get the latest top product list on email
* database migration;

**Database scheme**

The application uses the following entities:

1. User – represents users of the application (simple user or employee).
2. Address – represents the user’s address.
3. Deposit – represents the user deposit (to simulate the payment system).
4. Category – represents the store’s menu groups.
5. Parameter – represents specific properties for category.
6. Product – represents the product.
7. Product details – represents specific properties for product.
8. Order – represents order information.
9. Order details – represents information about order’s products.
10. Sales – represents information about sales.

Users table has one-to-many relationship with addresses, orders and sales tables and one-to-one with deposits table.

Categories table has one-to-many relationship with parameters and products table.

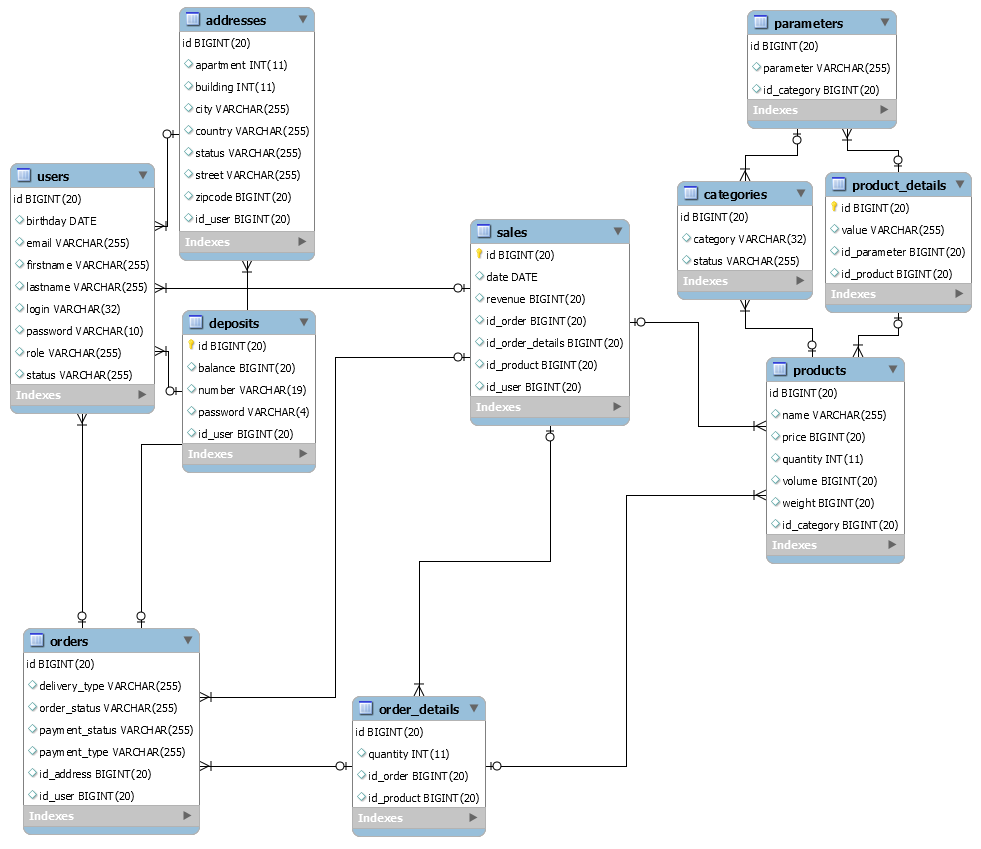
Products table has one-to-many relationship with product details table.

Product details table has many-to-many relatioship with products and parameters tables.

Order table has many-to-one relationship with users, addresses tables and one-to-many relationship with order details table.

Order details table has many-to-one relationship with products table.

Entity relationship model is shown on the picture 1.

****

Picture 1 - Entity relation model.

**Explanation of model implementation**

The application covers following business processes, divided by the user role.

For users:

1. Browse directory with ability to filter by parameters
2. Browse and edit account (information, addresses, password)
3. Make an order
4. Browse order’s history

For employees:

1. Browse orders
2. Update order’s status (shipped/not shipped, paid)
3. Browse sale’s statistics (top-10 products, uses, month/week revenue)
4. Add products
5. Manage directory’s categories

While purchasing shopping bucket should be displayed on each page both for guest and authorized user. After authorization or closing the page bucket should not clear.

**Application modules**

Application is built on three-layered architecture.

All classes are running inside Dependency Injection container, provided by Spring Framework.

The data access layer or DAO layer – provides data accessing with JPA implementation – Hibernate. Application has entity objects that are persisting to MySQL database.

The service layer – all business-logic is implemented there. Services are using DAO classes’ injections to get persistent data. All service methods are running in transactions, managed by Spring.

The representation or view layer – representation of information and handling of requests managed by controllers. Services are injected to controllers. View layer is built on Spring MVC technology. Methods of controllers handle requests and call service methods to process. Views are JSP pages which uses Apache Tiles.

In addition, there is one module that sends JSON-messages with top-product list to Active MQ broker using JMS.

**UI layer description**

UI layer consists of 10 controllers. Controllers are built, using Spring MVC technology.

Templates is a JSP with JSTL and Spring tags. Each page template use Apache tiles layouts. The default layout consists of header, navigation bar, main content and footer.

defaultHeader.jsp – basic layout, contains meta-data, styles, icons, scripts links and onlinestore’s logo.

defaultNav.jsp – contains navigation bar with search form and menu and login buttons.

defaultFooter.jsp – contains footer with copyright.

Information messages used in the UI layer such as validation errors messages are provided by messages.properties files

Application uses native javascript.

**Business layer description**

Business layer has 9 service interfaces and 9 corresponding implementations. List of services with description is listed below:

1. UserService – provides logic for CRUD operations with users.
2. DepositService – provides logic for CRUD operations with user’s deposit.
3. CategoryService – provides logic for CRUD operations with directory’s categories.
4. ProductService – provides logic for CRUD operations with products.
5. ImageService – provides logic for validating and uploading the images.
6. CartService – provides logic for stores products in shopping bucket and validating the product’s actual state (price, quantity, etc).
7. OrderService – provides logic for CRUD operations with orders and some specific actions such as validating deposit’s balance and actual product’s quantity
8. SalesService – provides CRUD operations for sales and allows to get sales statistics (month/week revenue, top-10 products, users)
9. MessageService – provides logic for creating and sending messages to broker.

All methods in service layer are transactional. Spring Framework provides transactions management.

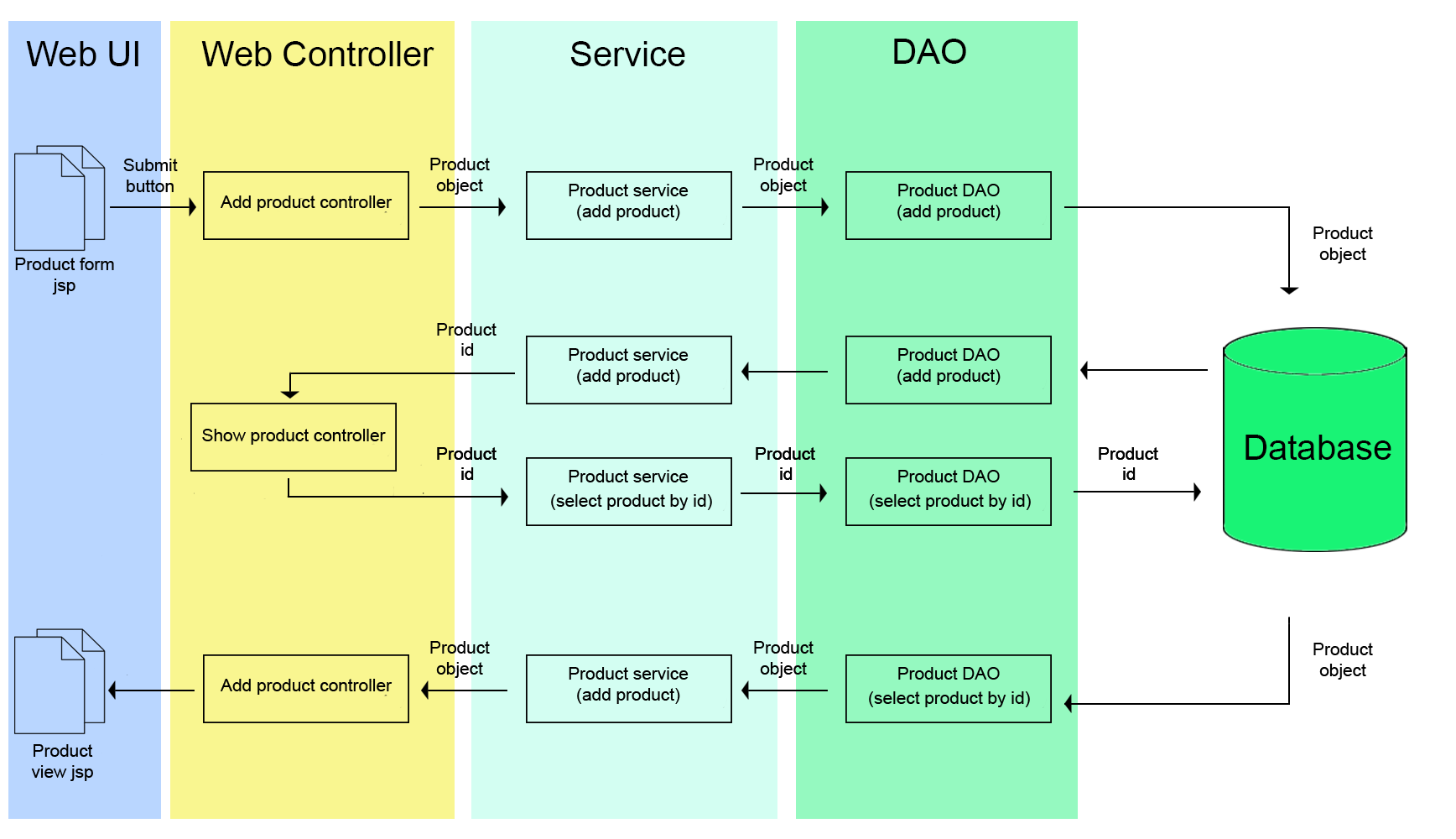
**Data layer description**

Dao layer contains 10 JPA entities, 6 DAO classes and 1 DTO classes (1 for REST service).

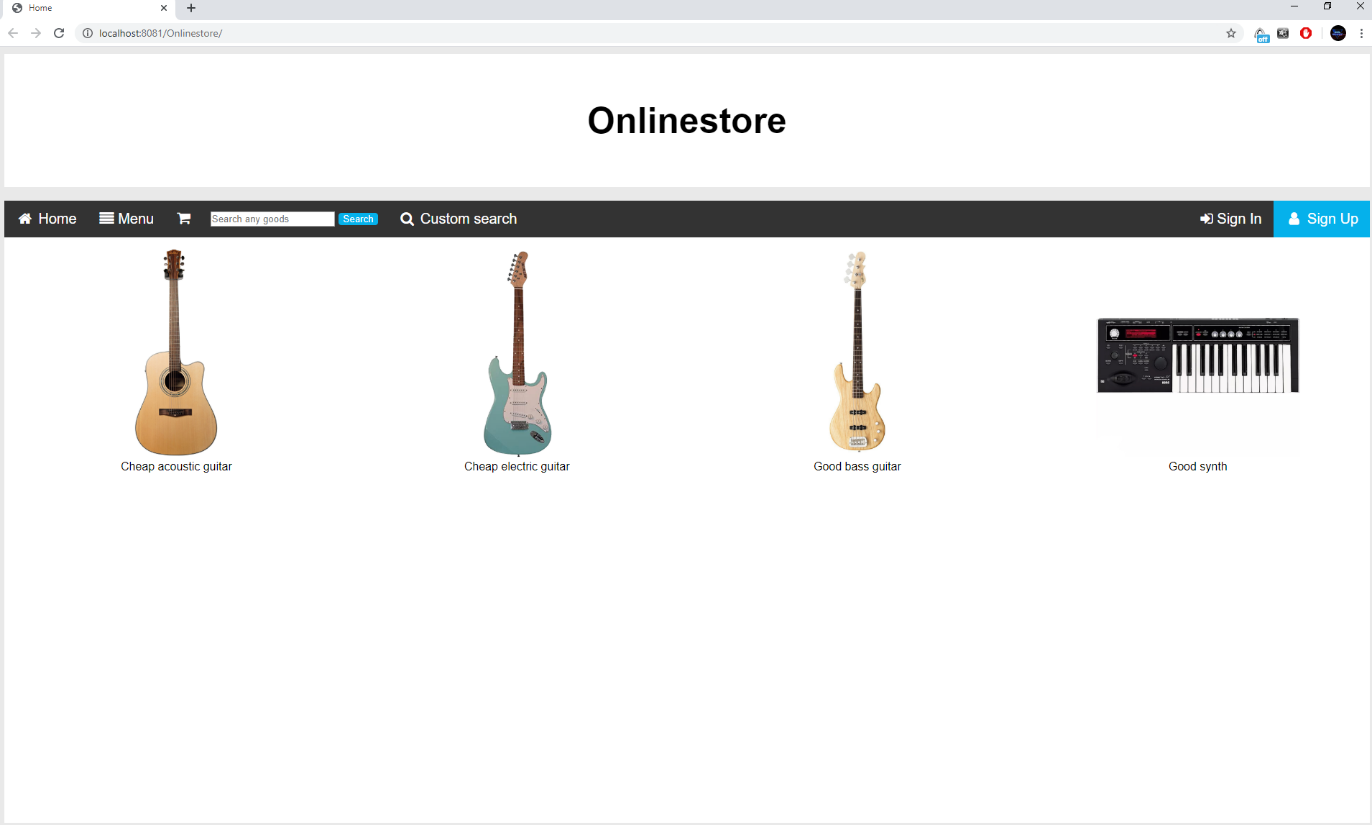
UserDao manages User and Address JPA entities. DepositDao manages only Deposit entity. CategoryDao manages Category and Parameter entities. ProductDao manages Product and ProductDetails entities. OrderDao manages Order and OrderDetails entity. SalesDao manages Sales entity.

# Common interaction between application layers

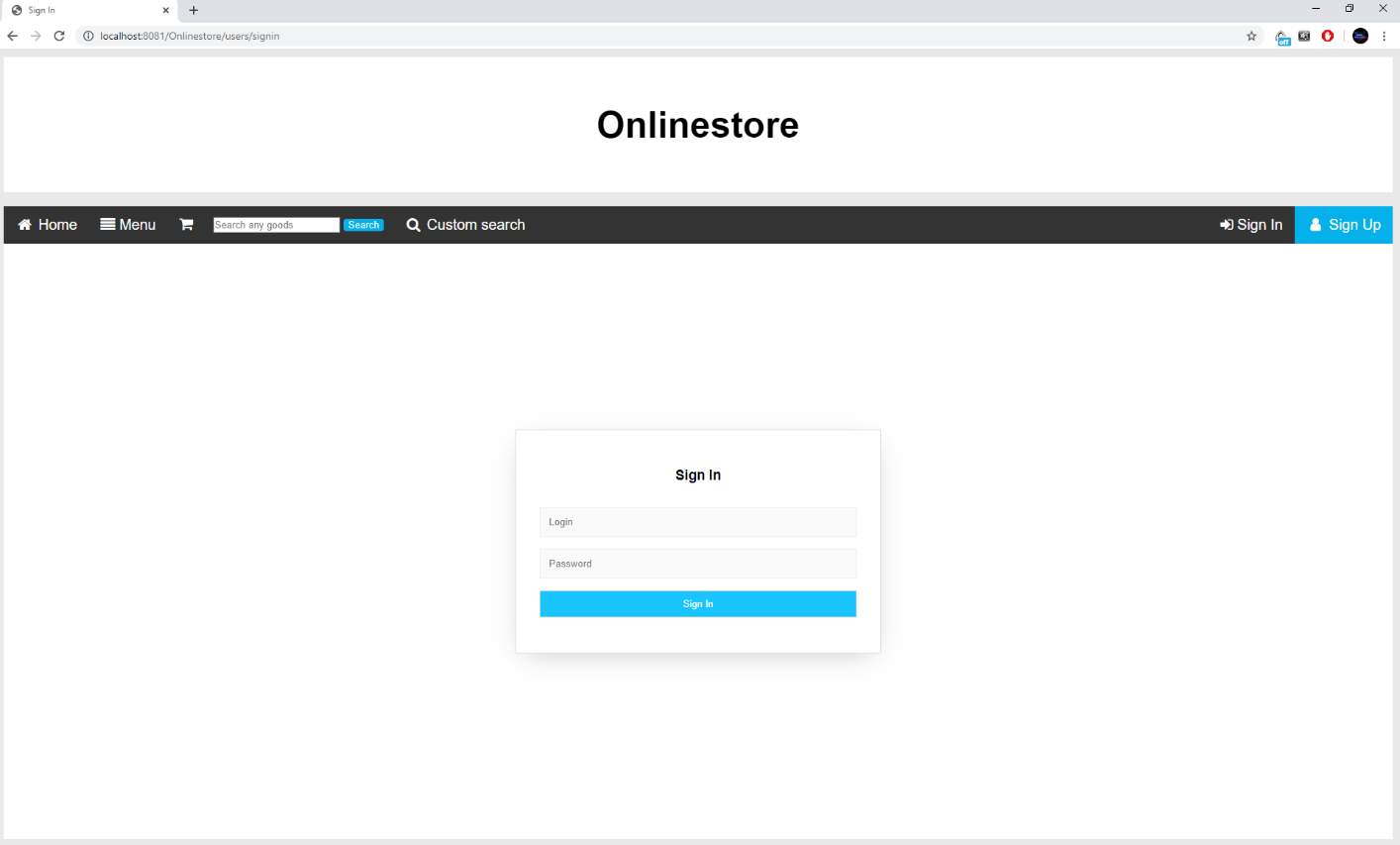
The diagram below shows a common example of interactions between layers. This example is based on add product operation.



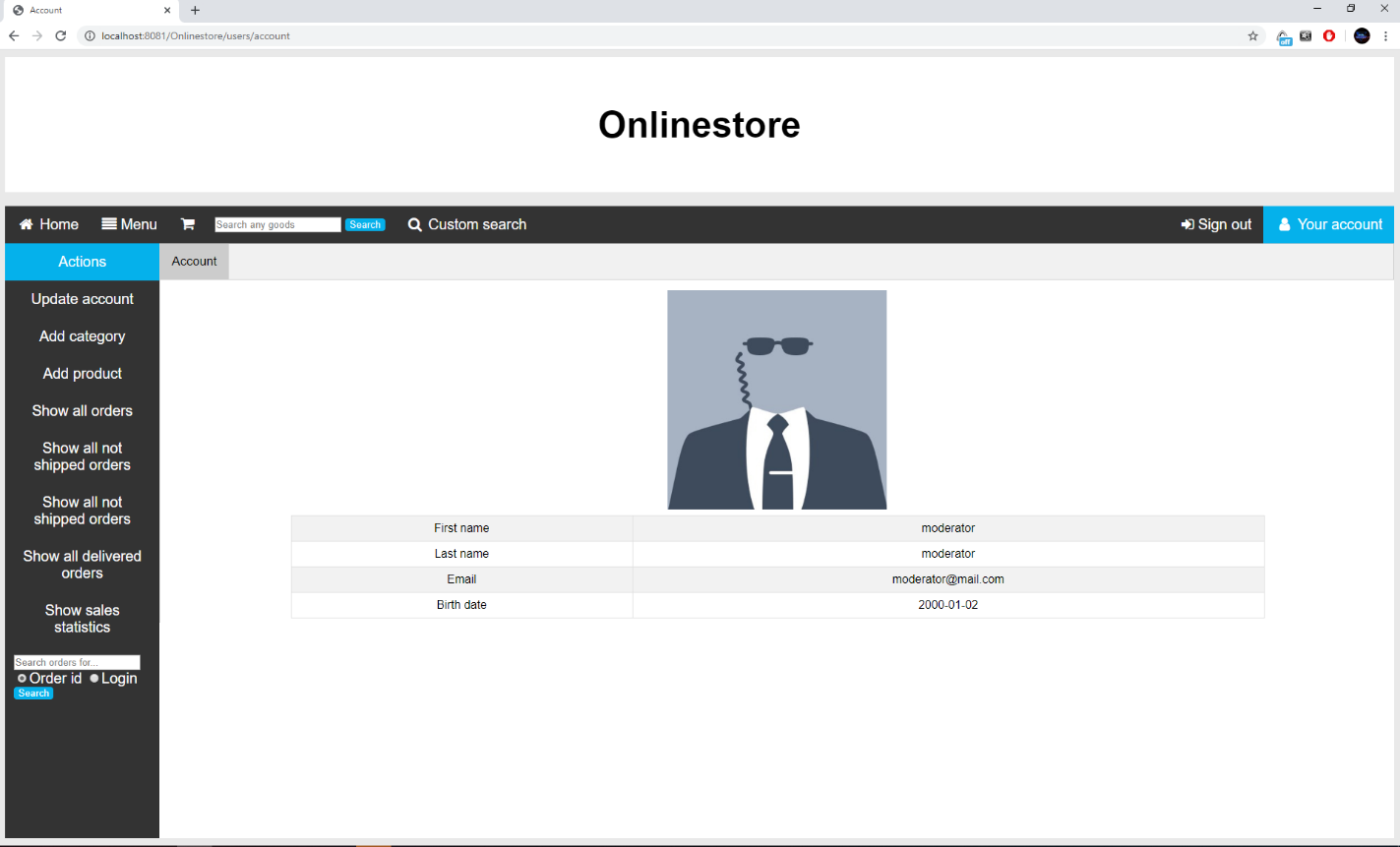
**Application screenshots**

****

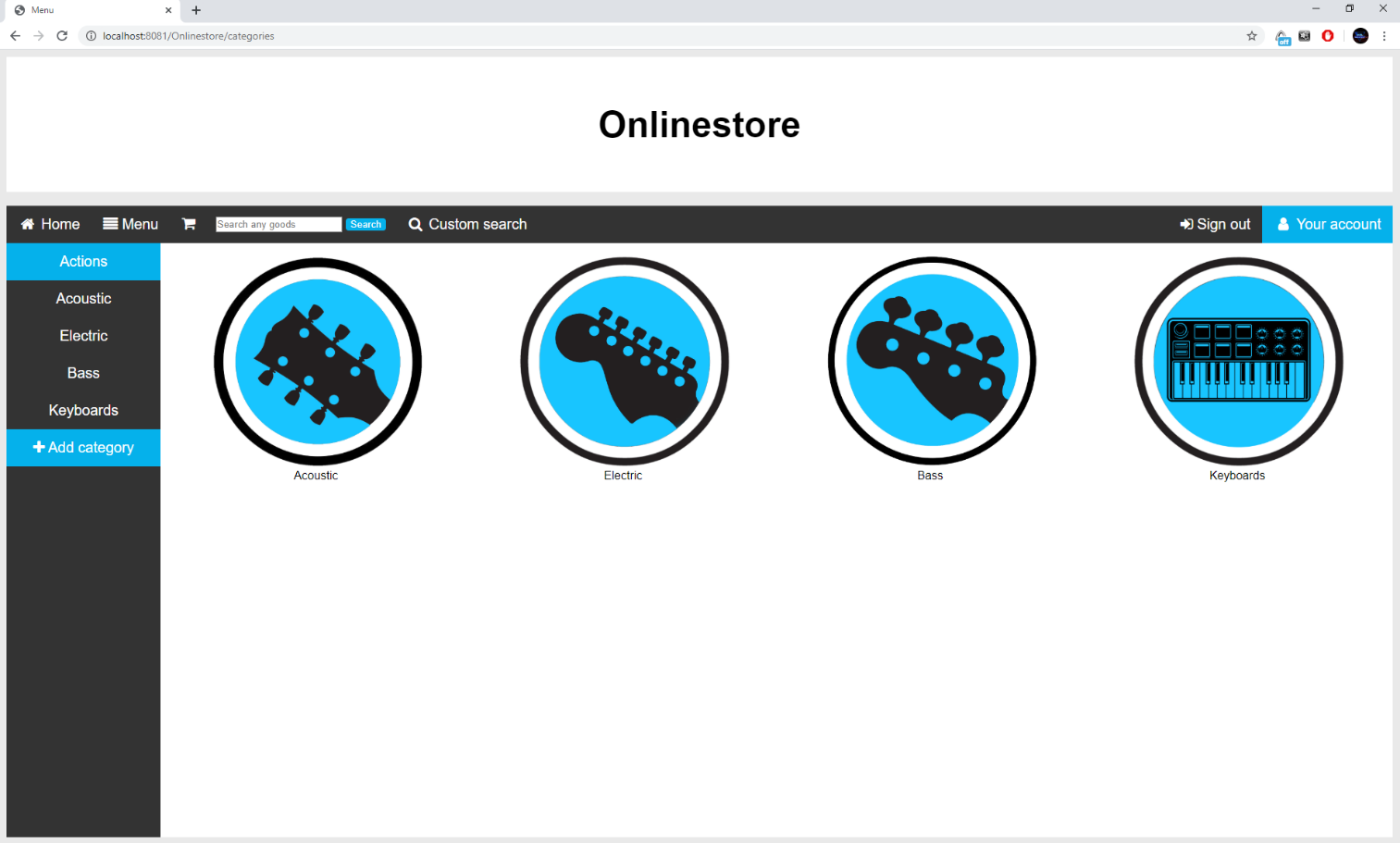
Main page



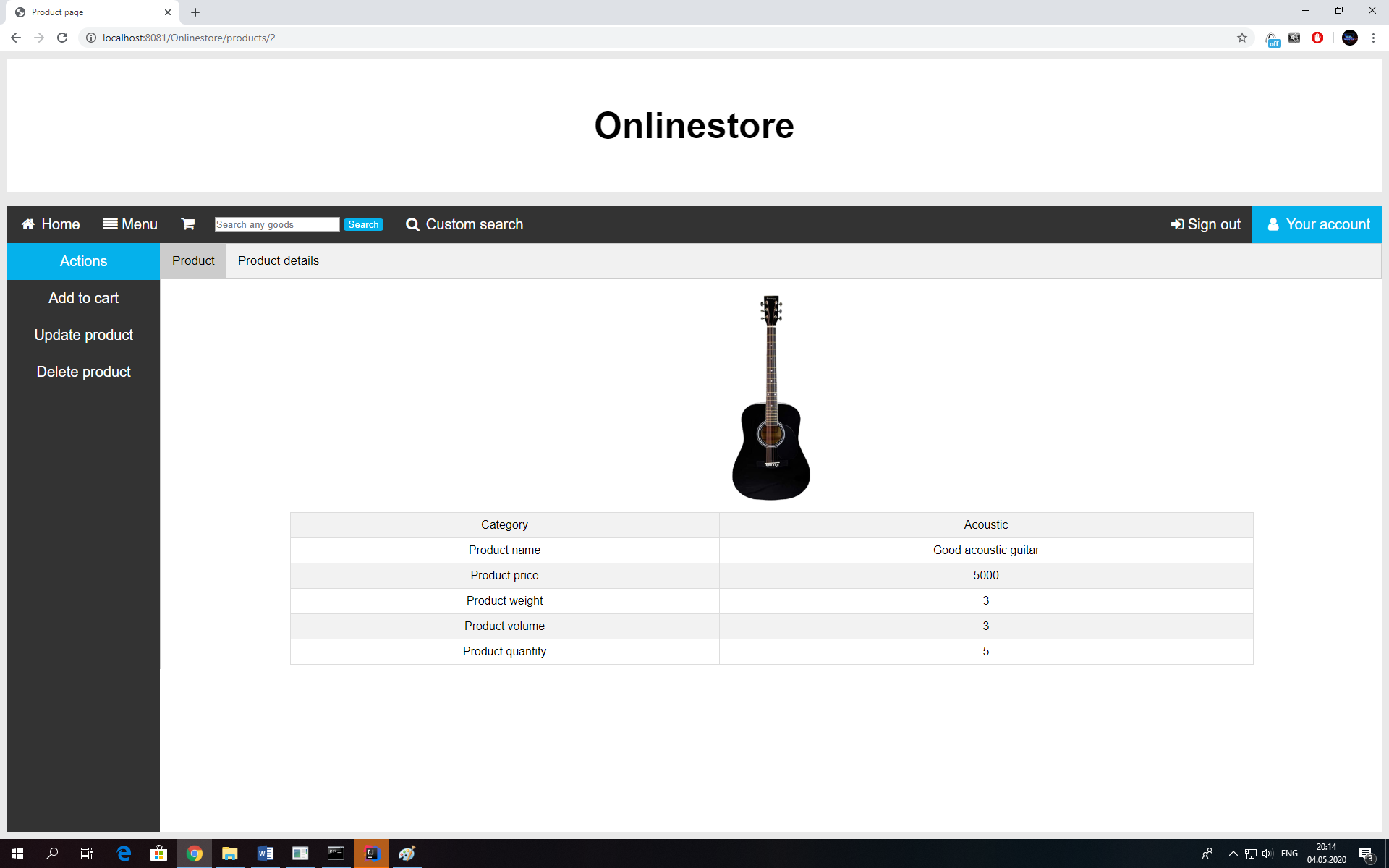
Login page



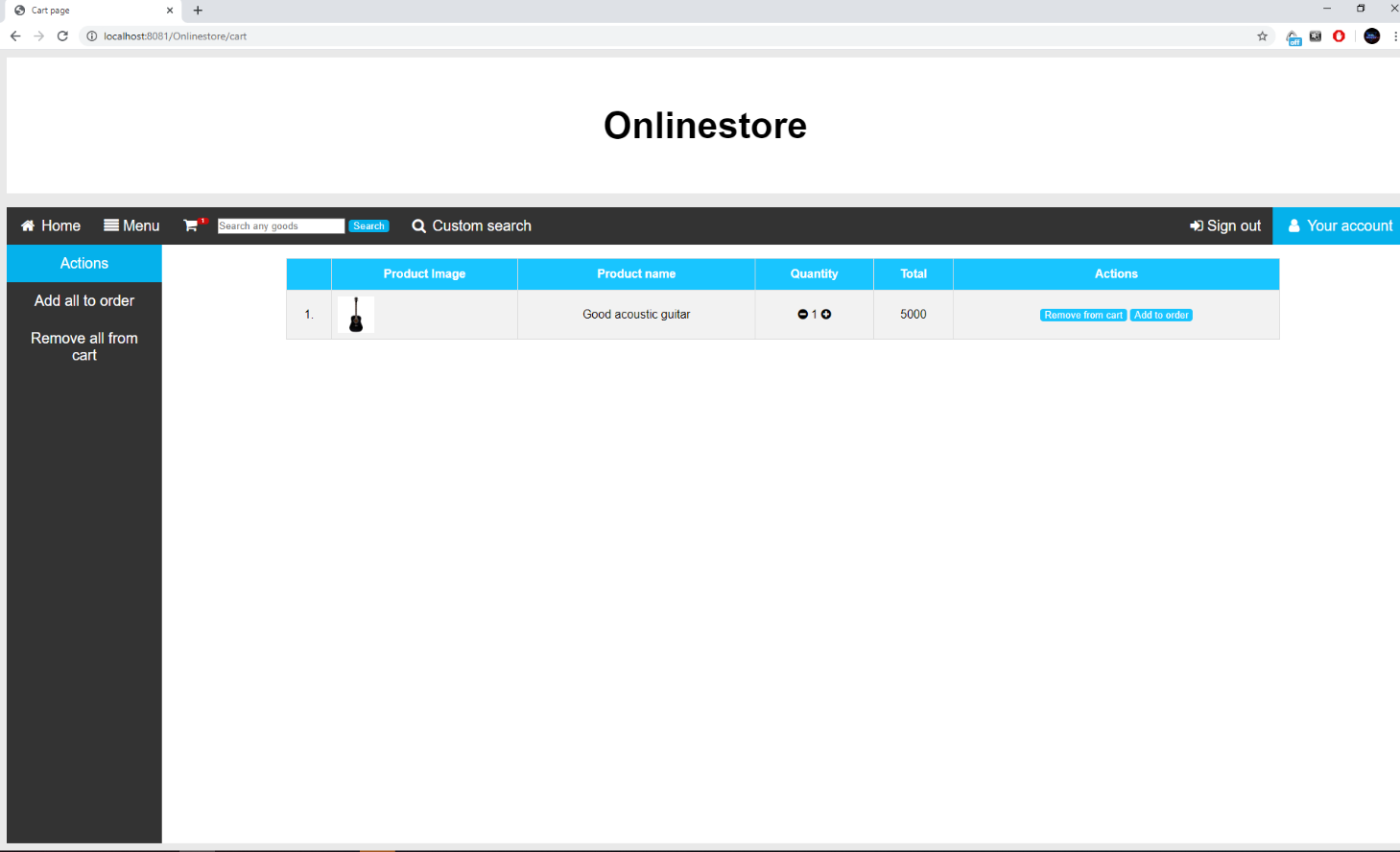
Account page



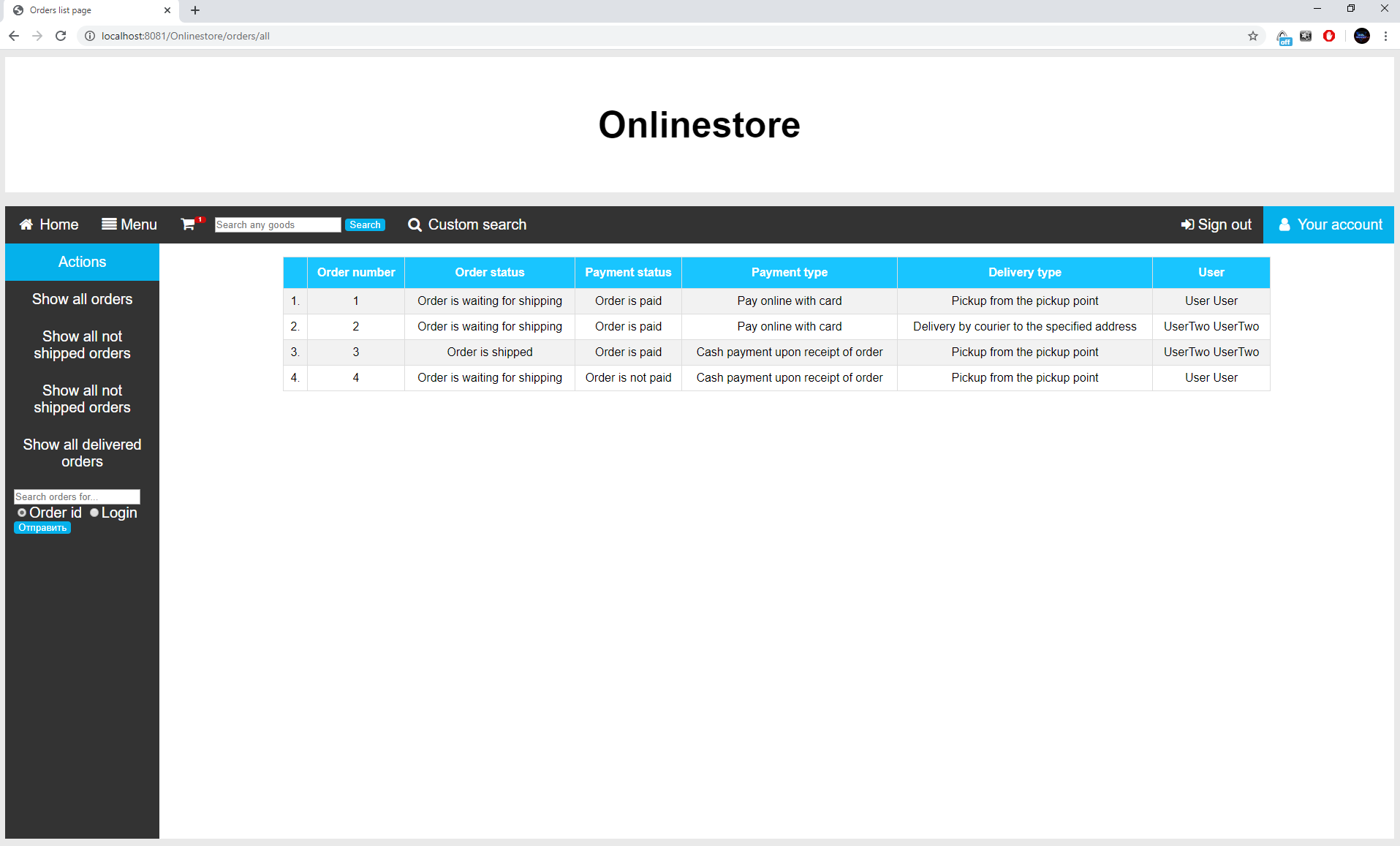
Categories page



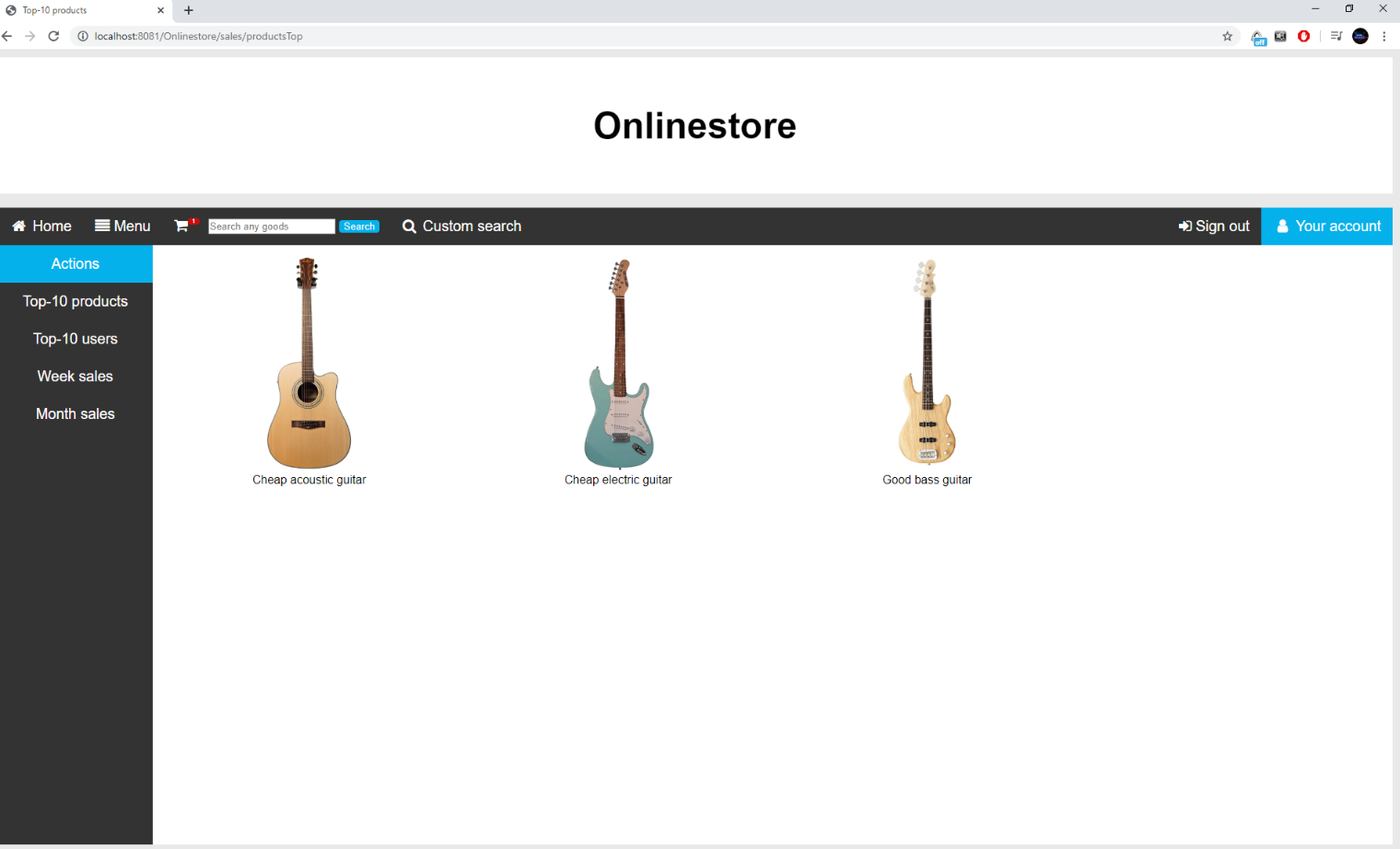
Product page

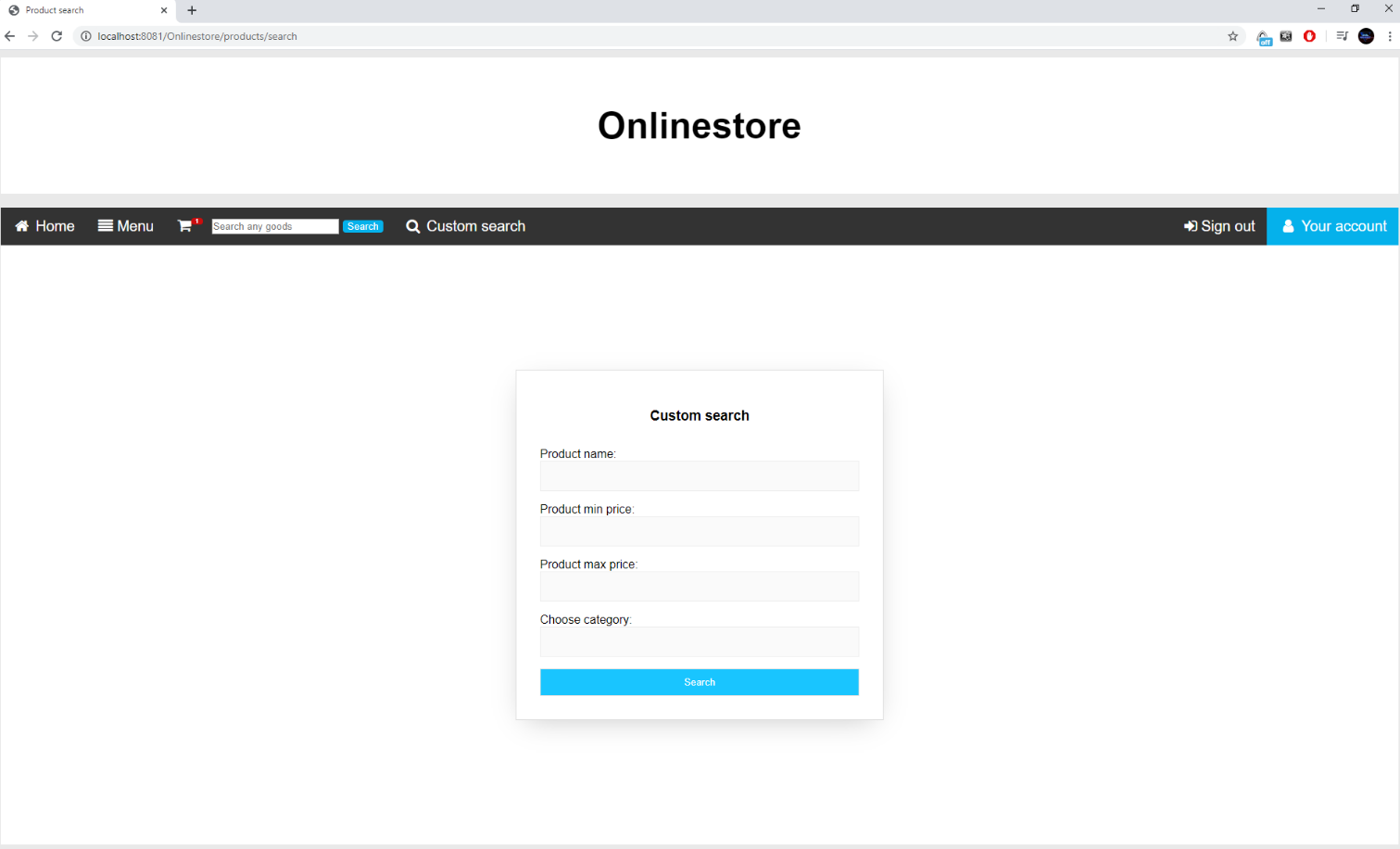


Cart page

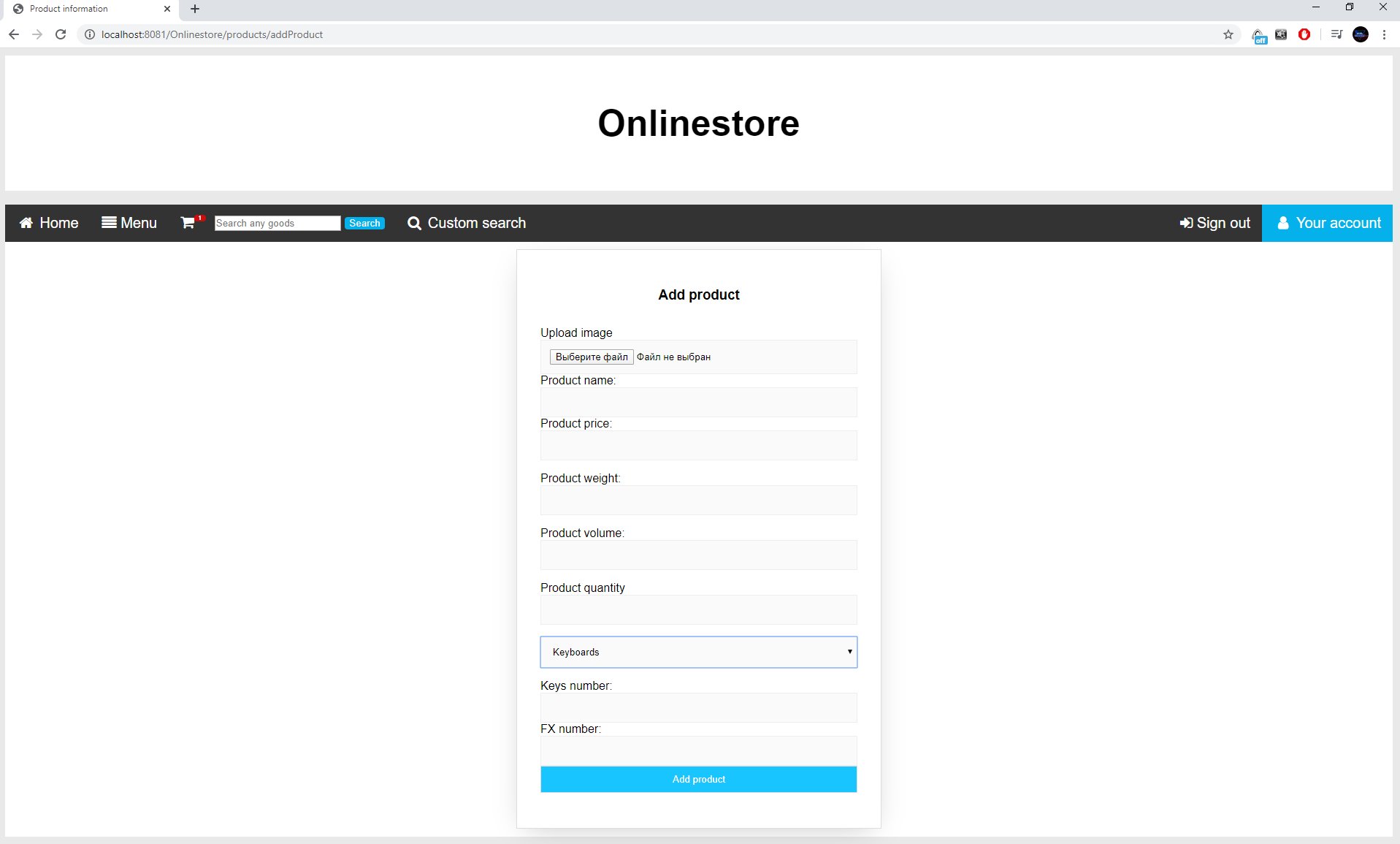


Orders page



Top-10 products page

Product search page



Product form page

**Unit tests**

Unit tests are written on all classes of service layer except imageService class, each test class tests one service class.

All test classes use mockito library to define the behavior of used dao classes in services.

List of Junit test classes for Service layer:

1. CartServiceTest
2. CategoryServiceTest
3. DepositServiceTest
4. OrderServiceTest
5. ProductServiceTest
6. UserServiceTest

Also there is Junit test for one Dao class – ProductDaoSearchTest.

**Deployment**

Applications uses IDE building process. IDE automatically builds, starts servers and deploys applications.

To deploy the applications press the «Run configuration» (green arrow) button.

This configuration skips unit tests. To run the tests use Maven command: mvn test.

**Logging configuration**

Logging is provided by Log4j library. Controller advice exception have debug level logs – for development process and debugging. Exceptional situations logged at error level.

Below is a short example of a log output:

2020-05-04 20:41:35 ERROR SqlExceptionHelper:142 - Duplicate entry 'Electric' for key 'UK\_5ky4frjmcobbiayt5jyx53mff'

2020-05-06 11:24:23 ERROR CustomControllerAdvice:106 - Nullpointer exception was thrown: For input string: "null"

2020-05-06 11:28:32 ERROR ImageServiceImpl:63 - Error while saving image: Only JPG images accepted

**Application improvements**

In further releases of the application, next improvements and features are planned:

* Split employee’s role into several subroles such as courier, moderator, heldesk
* Add a helpdesk message service
* Add discount system
* Implement Ajax techology to cart service
* Transition to micro service architecture
* Add Docker
* Add pagination