# “ENTEPRISE APPLICATION DEVELOPMENT

# OF ONLINE STORE”

# Author: Sergey Zyl

# Version 1.00

**Content**

[1. Objective. 3](#_Toc532423149)

[2. Technologies and frameworks. 3](#_Toc532423150)

[3. Database. 4](#_Toc532423151)

[3.1. Database scheme. 4](#_Toc532423152)

[3.2. Tables. 5](#_Toc532423153)

[4. Architecture. 6](#_Toc532423154)

[4.1. Patterns. 6](#_Toc532423155)

[5. Application structure of the store. 7](#_Toc532423156)

[5.1. Data access layer. 7](#_Toc532423157)

[5.1.1. Entities. 7](#_Toc532423158)

[5.1.2. DAO layer. 7](#_Toc532423159)

[5.2. Service layer. 8](#_Toc532423160)

[5.3. Controllers. 9](#_Toc532423161)

[5.4. View layer. 11](#_Toc532423162)

[6. Application structure of the stand. 17](#_Toc532423163)

[6.1. Model layer. 17](#_Toc532423164)

[6.2. Beans. 17](#_Toc532423165)

[6.2.1. Singleton bean. 17](#_Toc532423166)

[6.2.2. Managed bean. 18](#_Toc532423167)

[6.3. MessageListener. 18](#_Toc532423168)

[6.4. Interface. 18](#_Toc532423169)

[7. Interaction of the modules. 18](#_Toc532423170)

[8. Testing. 19](#_Toc532423171)

[8.1. Unit-tests. 19](#_Toc532423172)

[8.2. Selenium tests. 19](#_Toc532423173)

[9. Logging. 19](#_Toc532423174)

[10. Future tasks. 20](#_Toc532423175)

# Objective.

The aim of the project is to design architecture and to develop web application of Internet-store as a sample and its advertising stand as a separate application.

# Technologies and frameworks.

The following technologies were used for the development of the main module of the application (Internet-store):

* IDE –  IntelliJ IDEA
* Tomcat 8.5
* DB – MySQL 5.7
* Maven 3.5.4
* Hibernate 5.2.10
* Spring Framework
* JSP

While developing the second module – advertising stand - one applied the technologies below:

• WildFly AS

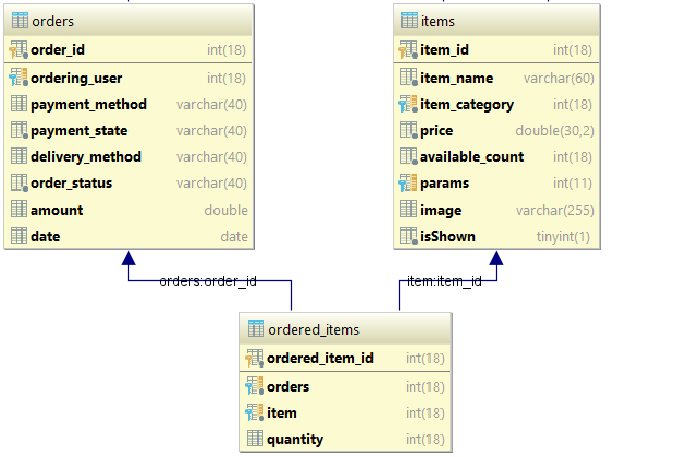
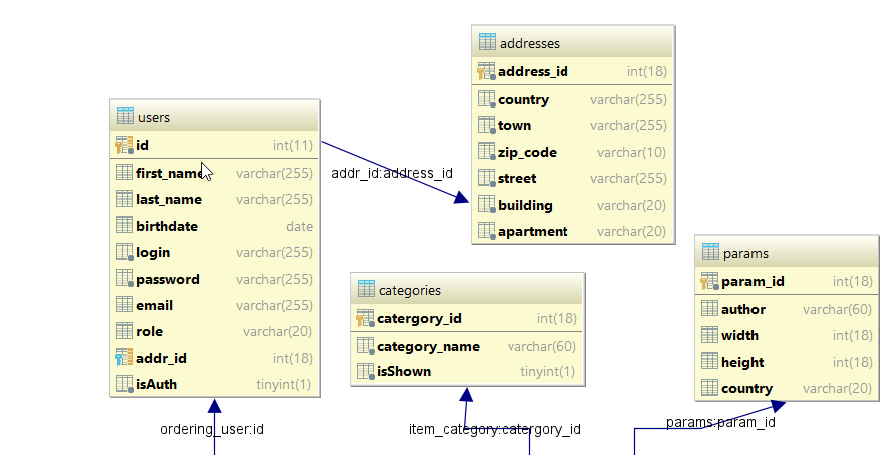
• EJB

• JSF

• ActiveMQ

# Database.

# Database scheme.



## Tables.

* **Users** – stores information about users: first name, last name, birthdate, login, password (stores as hash), email, role (USER or ADMIN), isAuth (represents whether user is authorized).

Field addr\_id is a foreign key to the table “Addresses”. Tables “Users” and “Addresses” have relation Many to One.

* **Addresses –** stores information about user’s address: country, city/town, zip code, street, building, apartment.
* **Items –** stores information about items in the shop: name, price, available count, price, isShown (represents whether item is available for users with role “USER”).

Field item\_category is a foreign key to the table “Categories”; tables “Items” and “Categories” have relation Many to One.

Field params is a foreign key to the table “Params”; tables “Items” and “Params” have relation One to One.

* **Categories -** stores information about categories in the shop: category name, isShown (represents whether category is available for users with role “USER”).
* **Params –** represents information about item’s params: author, width, height, country.
* **Orders –** represents information about user’s orders: payment method, payment status, delivery method, order status, amount, date.

Tables “Orders” and “Users” have relation Many to One.

Tables “Orders” and “Items” have relation Many to Many.

* **Ordered\_items –** table for representing relation between tables “Orders” and “Items”. It contains order\_id, item\_id and quantity fields.

# Architecture.

## Patterns.

A three-level architectural model is used in the actual application. It consists of a view level, a business logic level and a data access level.

The task of a view level is communication to the User: data conversion for visualization, provision of interaction interface and further processing of the User’s actions.

Business-level contains all functional algorithms. It plays the role of the medium between a view level and a data access level.

The last one fulfills data storage. It is a set of unified methods of data access, known as CRUD (Create, Update, Delete).

Project pattern DAO (Data Access Object) is used to accomplish data access level. DAO is an object, which makes an abstract interface work with a data base or some other data storage. This pattern gives a number of possibilities to work with data irrespective of the data storage mechanism applied.

Business logic level is implemented with the help of services – a set of interfaces, which are used in a view level for data manipulation.

Project pattern MVC (Model-View-Controller) was used to achieve a view level.

Function (View) displays model data to the User. The User interacts with the given graphic interface, thus changing the model. These changes are processed by Controller.

Function (Controller) is an interpreter of the User’s actions, which informs the model about necessary changes in the interaction interface with a business logic level.

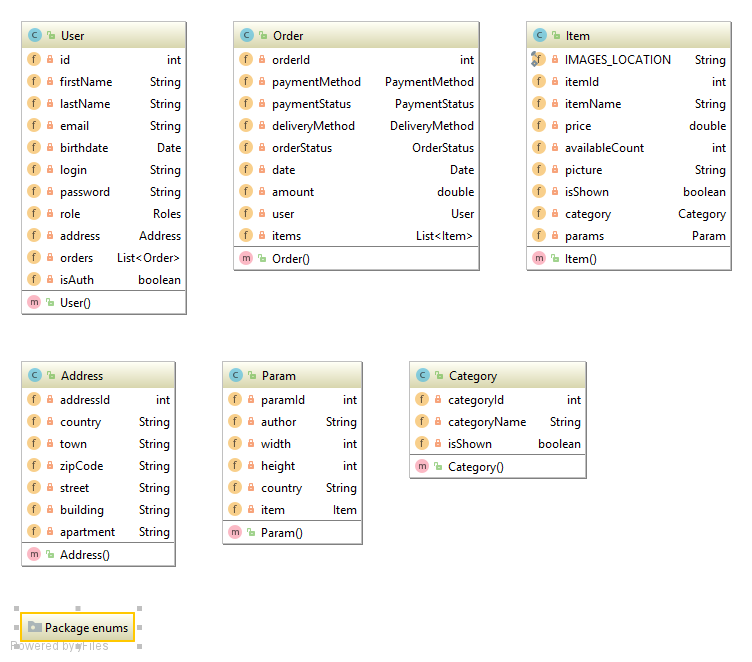
Function (Model) is a data set, which can change its type depending on the Controller commands.

# Application structure of the store.

## 5.1. Data access layer.

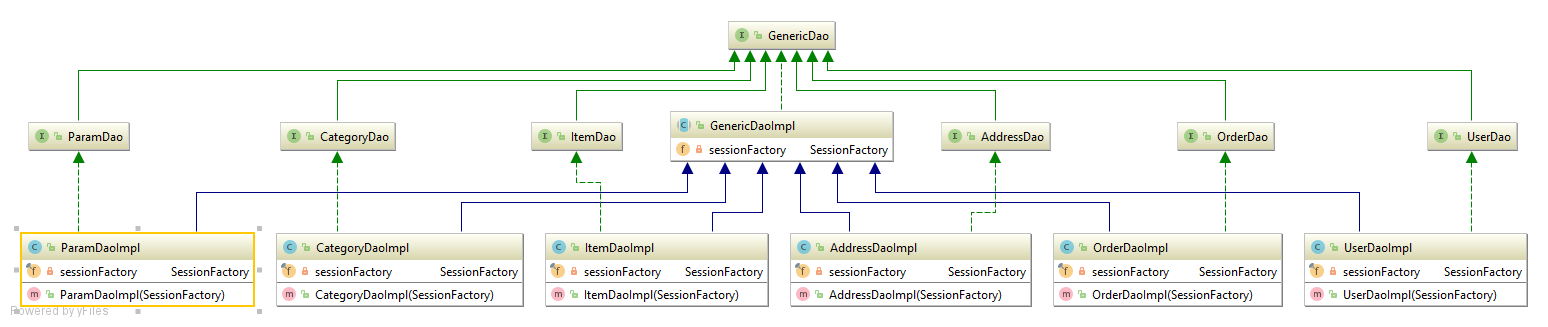
### 5.1.1. Entities.

* Address – mapped to table “addresses”. Stores information about user’s address.
* Category – mapped to table “categories”. Stores information about category of item.
* Item – mapped to table “items”. Stores information about item.
* Order – mapped to table “orders”. Stores information about orders.
* Param – mapped to table “params”. Stores information about item’s parameters.
* User – mapped to table “users”. Stores information about users.



### 5.1.2. DAO layer.

|  |  |  |
| --- | --- | --- |
| **Interface** | **Implementation** | **Description** |
| GenericDao<T> | GenericDaoImpl<T> | Generic DAO for basic CRUD operations. All other Dao implements this interface and extends abstract class. |
| AddressDao | AddressDaoImpl | Dao for working with database and addresses |
| CategoryDao | CategoryDaoImpl | Dao for working with database and categories |
| ItemDao | ItemDaoImpl | Dao for working with database and items |
| OrderDao | OrderDaoImpl | Dao for working with database and orders |
| ParamDao | ParamDaoImpl | Dao for working with database and params |
| UserDao | UserDaoImpl | Dao for working with database and users |

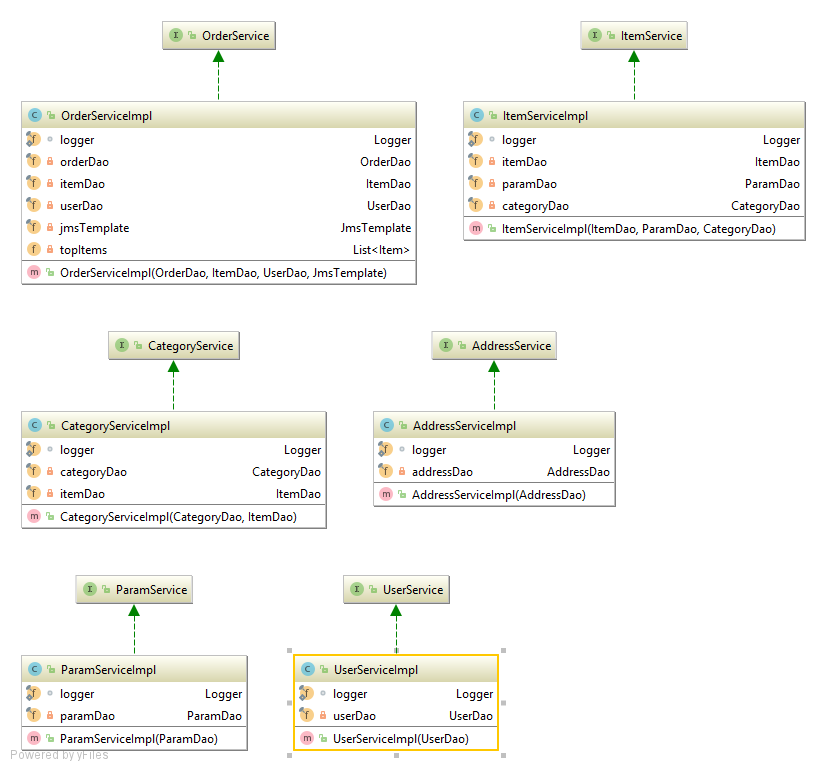


Dao objects are called in services for accessing data stored in database.

All Dao implementations has an exemplar of SessionFactory through which we can get sessions, create HQL queries and perform database operations.

## 5.2. Service layer.

|  |  |  |
| --- | --- | --- |
| **Interface** | **Implementation** | **Description** |
| AddressService | AddressServiceImpl | Service for working with addresses(add, update or delete) |
| CategoryService | CategoryServiceImpl | Service for working with categories(Add, disable, get all, etc.) |
| ItemService | ItemServiceImpl | Service for working with items(Add, edit, disable, update, etc.) |
| OrderService | OrderServiceImpl | Service for working with orderds and user basket (add item to basket, remove item from basket, add basket to orders, get all orders, etc.) |
| ParamService | ParamServiceImpl | Service for working with params(get authors, get countries, get sizes) |
| UserService | UserServiceImpl | Service for working with users(login, register, update, logout) |

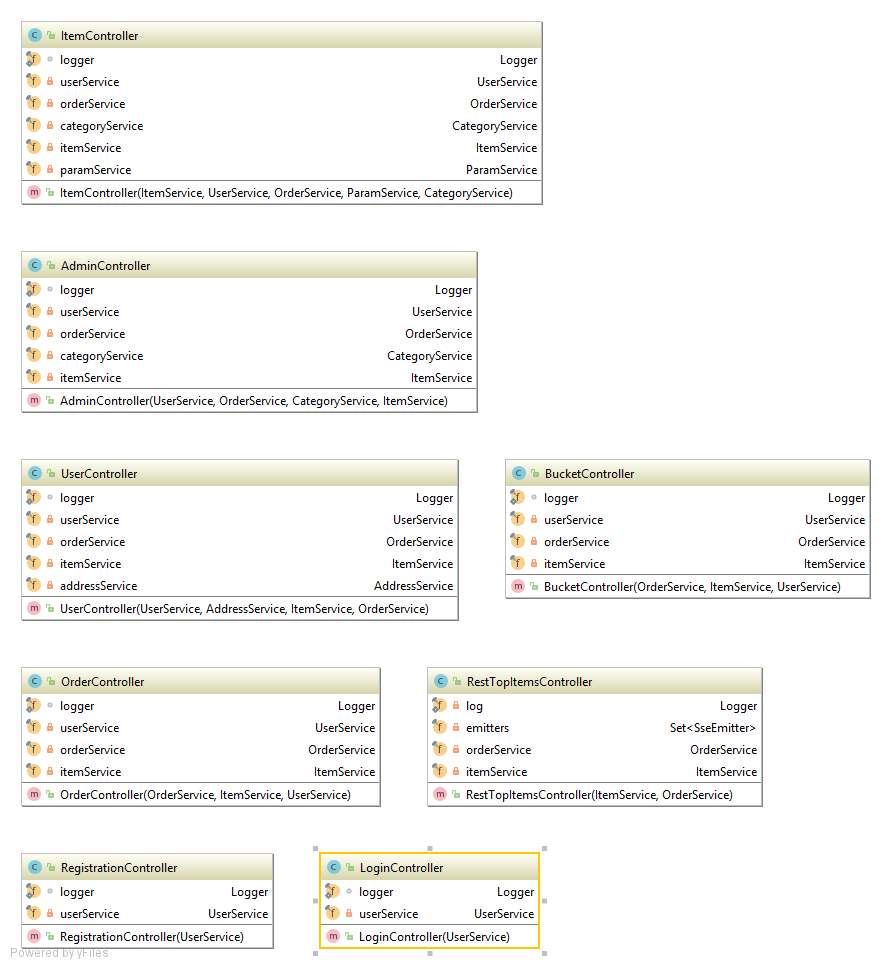


Services is a middleware between controller and model, all logic and algorithms are implemented in them. They are called in the controllers.

## 5.3. Controllers.

The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

* **AdminController –** available for authorized users with ADMIN role. By this controller admins may add/disable/edit items, edit orders made by all users, manage categories (add/disable) and see statistics of the shop.
* **UserController** – available only for authorized users. By this controller users may see/change their info, password and address.
* **RegistrationController –** controller for creating a new account.
* **LoginController** - controller for log in.
* **ItemController –** available for all users. By this controller users may see catalog, filter items by categories and params, search for the items.
* **OrderController –** available for authorized users. By this controller you may see history of orders and repeat order.
* **BucketController –** available for all users. By this controller you may add/remove item, make an order(if you are authorized), see your shopping cart.

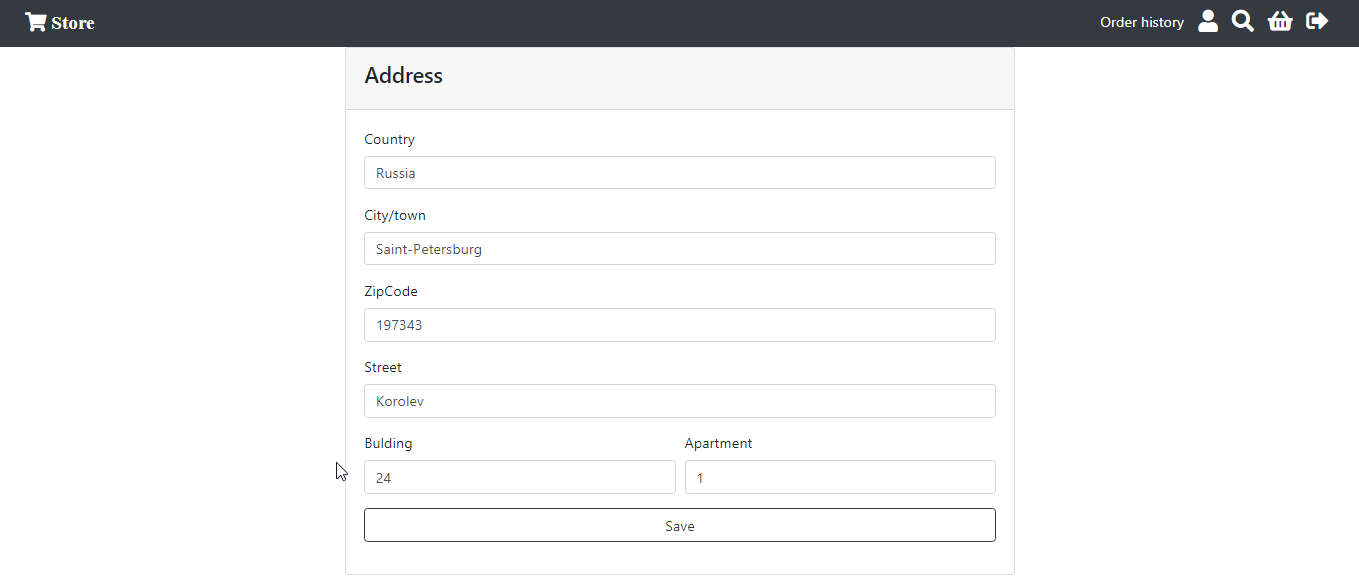
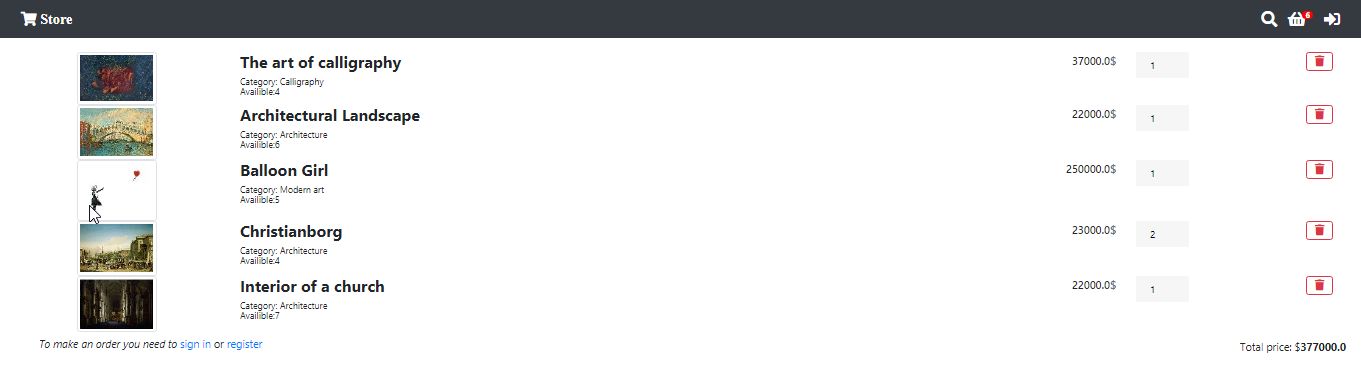


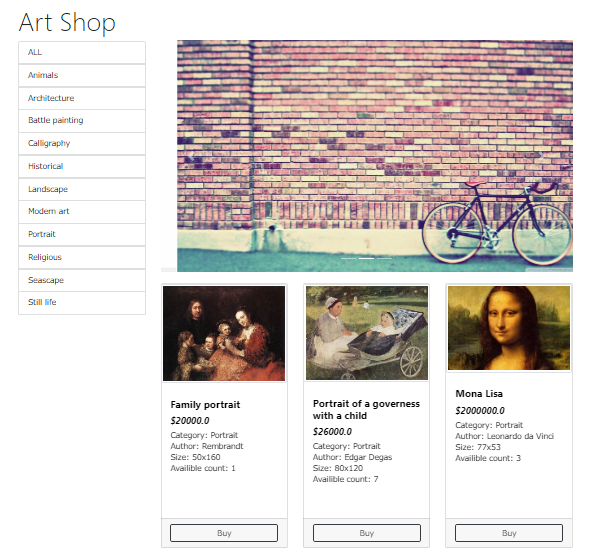
## 5.4. View layer.

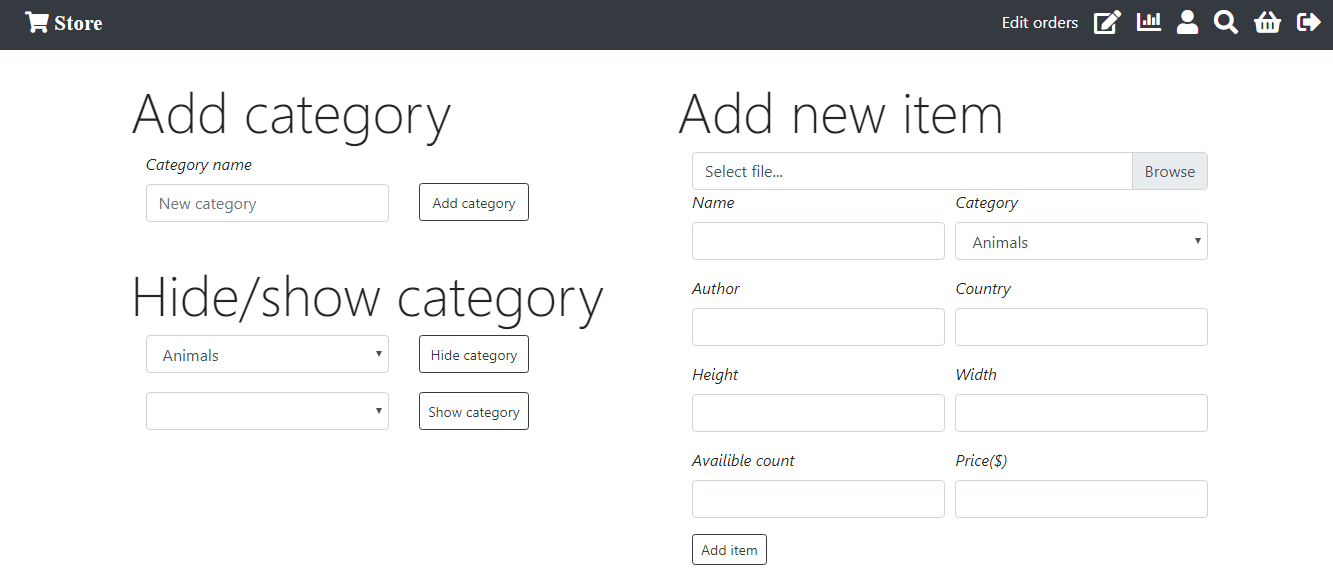
To create [dynamically generated web pages](https://en.wikipedia.org/wiki/Dynamic_web_page) based on [HTML](https://en.wikipedia.org/wiki/HTML), we used JSP (Java Server Pages). Architecturally, JSP may be viewed as a high-level [abstraction](https://en.wikipedia.org/wiki/Abstraction_(computer_science)) of [Java servlets](https://en.wikipedia.org/wiki/Java_servlet). JSPs are translated into [servlets](https://en.wikipedia.org/wiki/Java_Servlet) at runtime, therefore JSP is a Servlet; each JSP servlet is cached and re-used until the original JSP is modified. To deploy and run Java Server Pages we used a [servlet container](https://en.wikipedia.org/wiki/Servlet_container)  [Apache Tomcat](https://en.wikipedia.org/wiki/Apache_Tomcat).

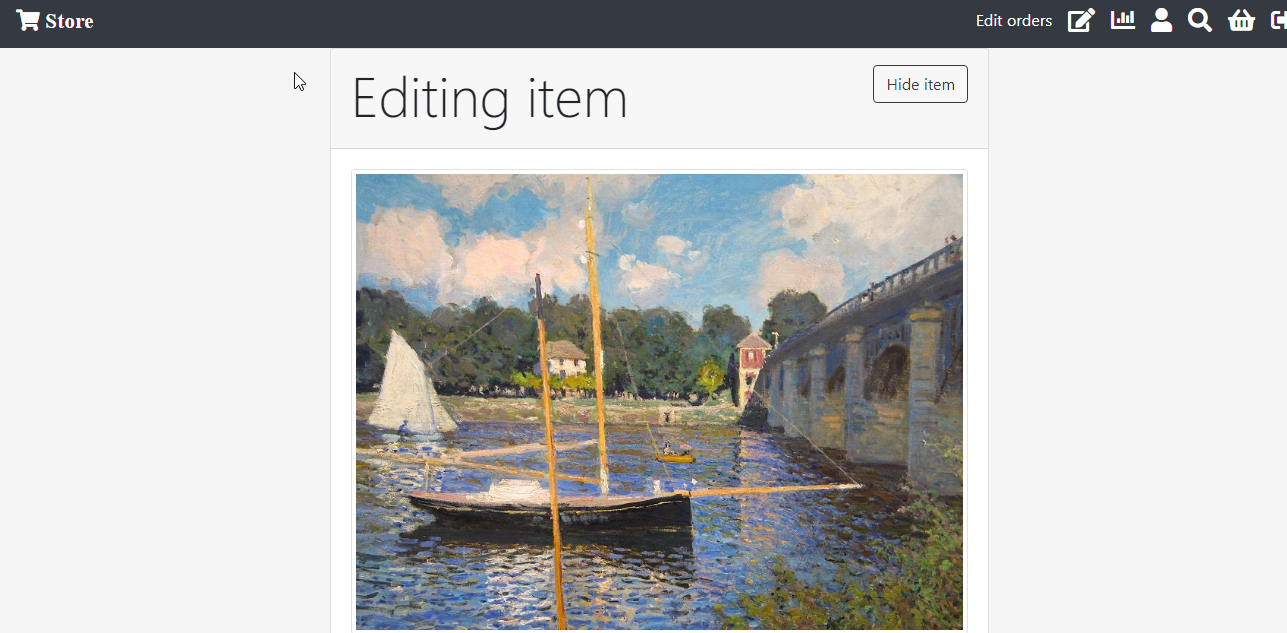
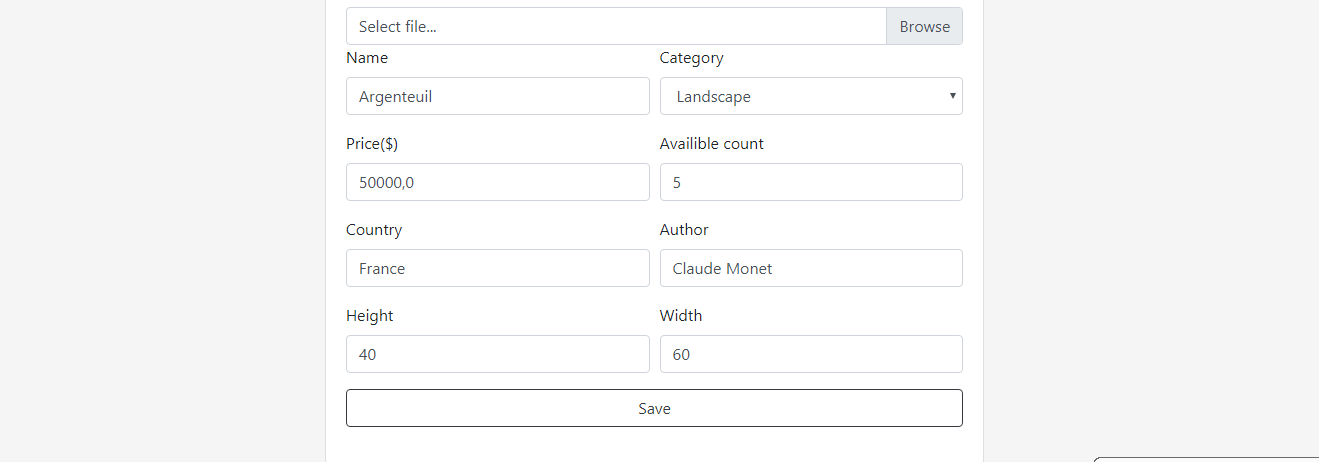
For designing pages, we used Bootstrap, a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source_software) front-end [framework](https://en.wikipedia.org/wiki/Application_framework) for designing [websites](https://en.wikipedia.org/wiki/Website) and [web applications](https://en.wikipedia.org/wiki/Web_application). For communicating on client between HTML and JavaScript we used jQuery framework.

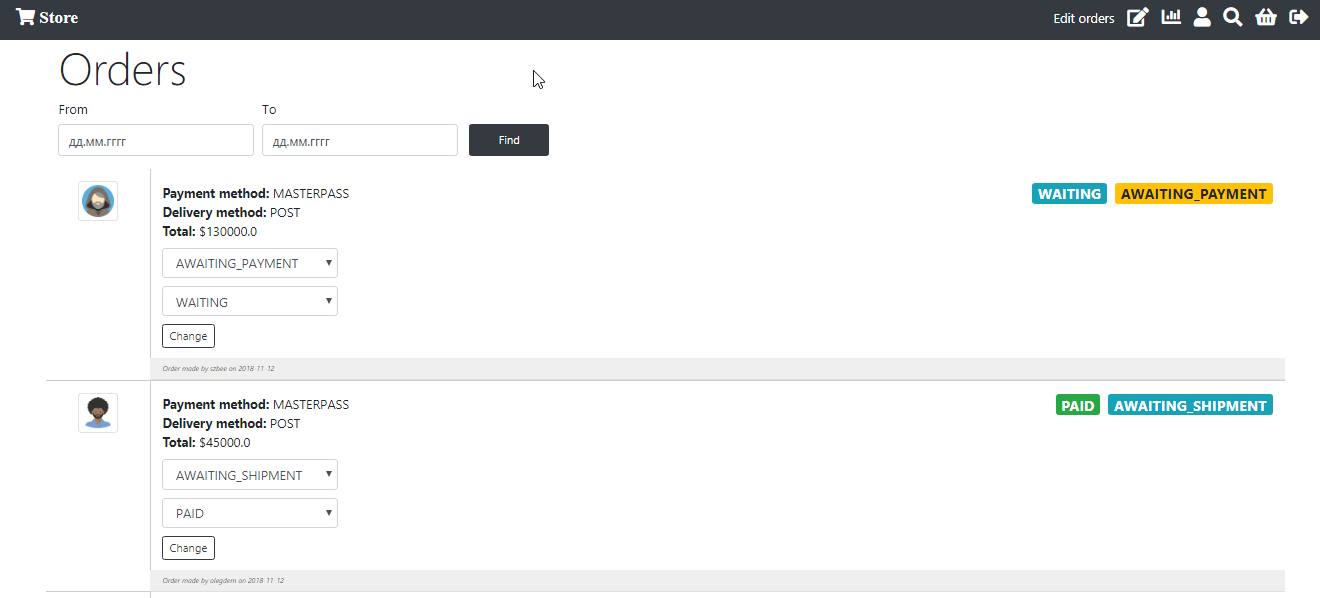
All designed pages:

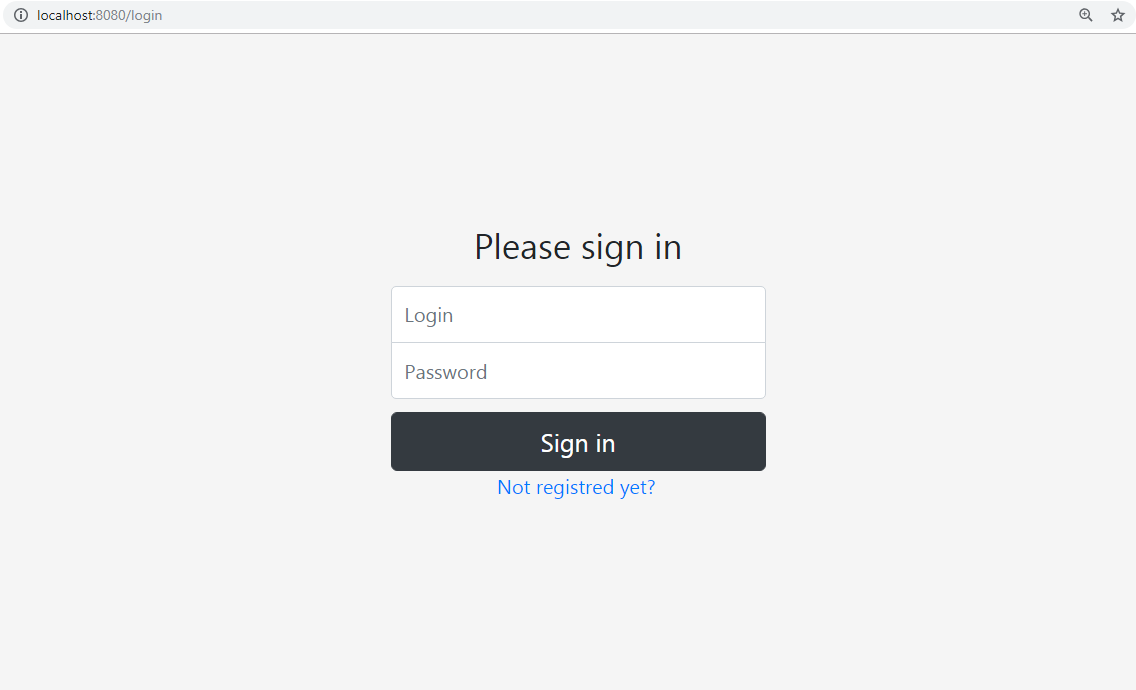
* **addres.jsp**  - page for editing address.
* **bucket.jsp –** page with a bucket.
* **catalog.jsp –** page with catalog.



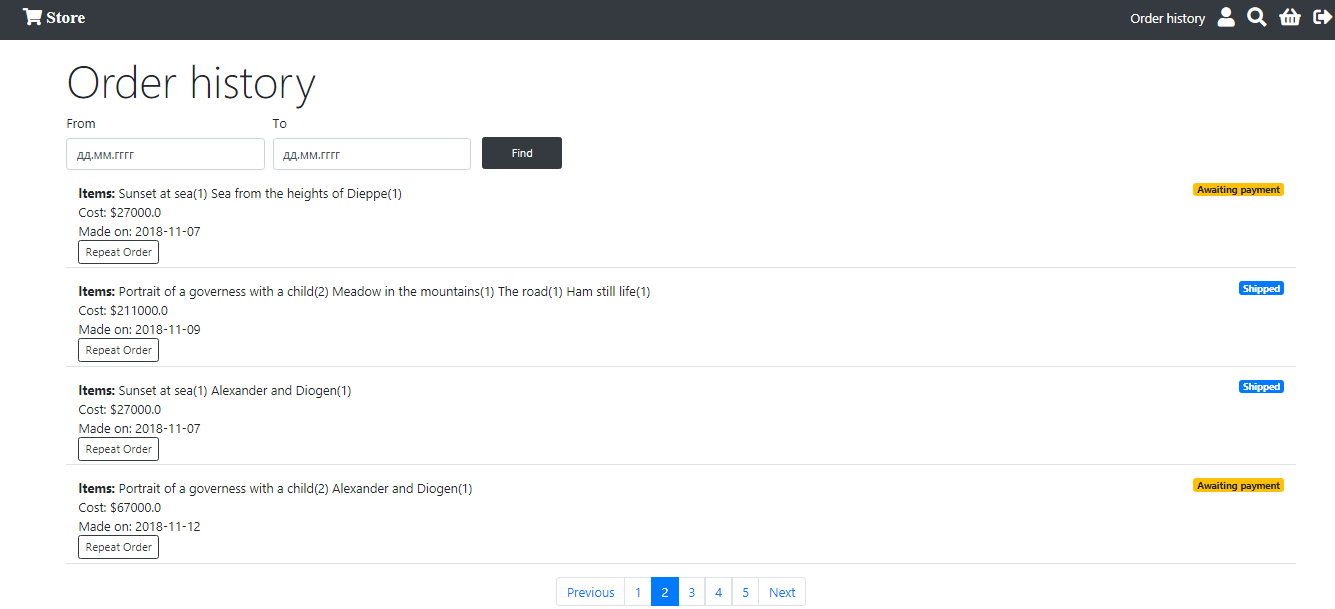
* **editCategories.jsp –** page for editing categoties and items.
* **editItem.jsp –** page for editing item.



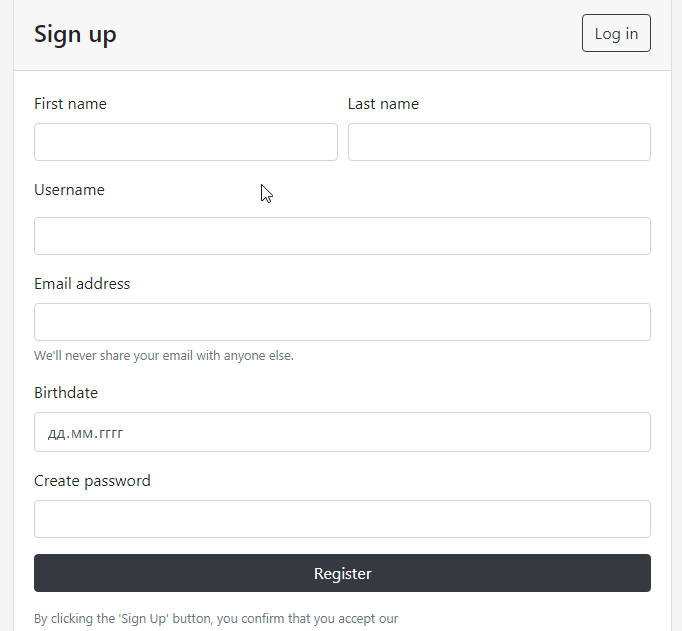
* **editOrders.jsp –** page for editing orders.
* **layout.jsp –** general dependencies(used for all pages).
* **login.jsp –** page for sign in.

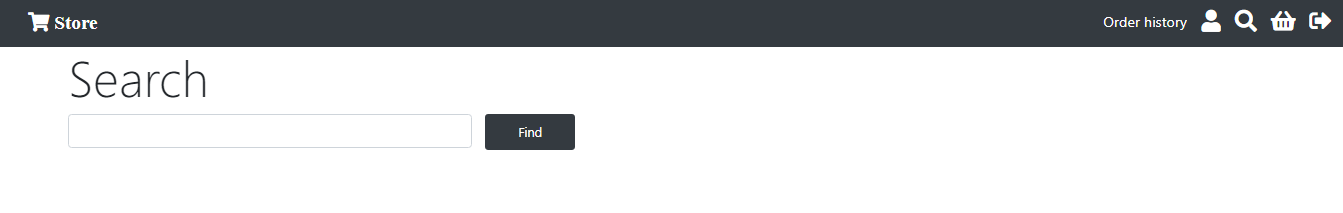
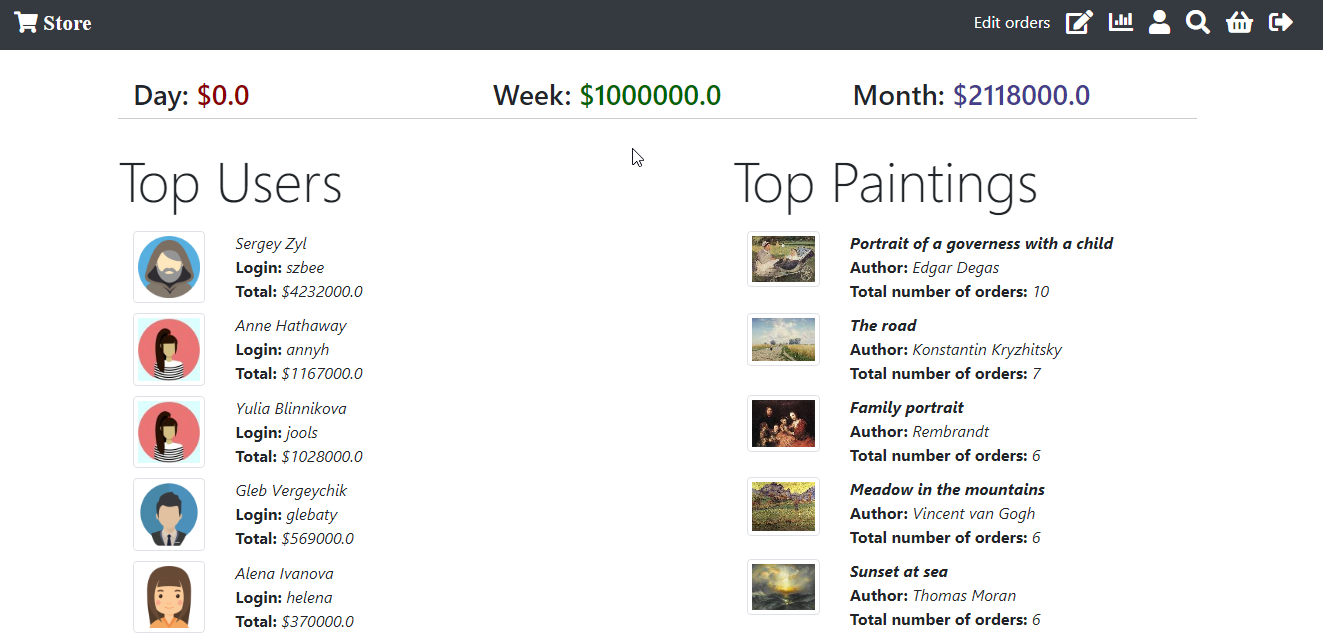
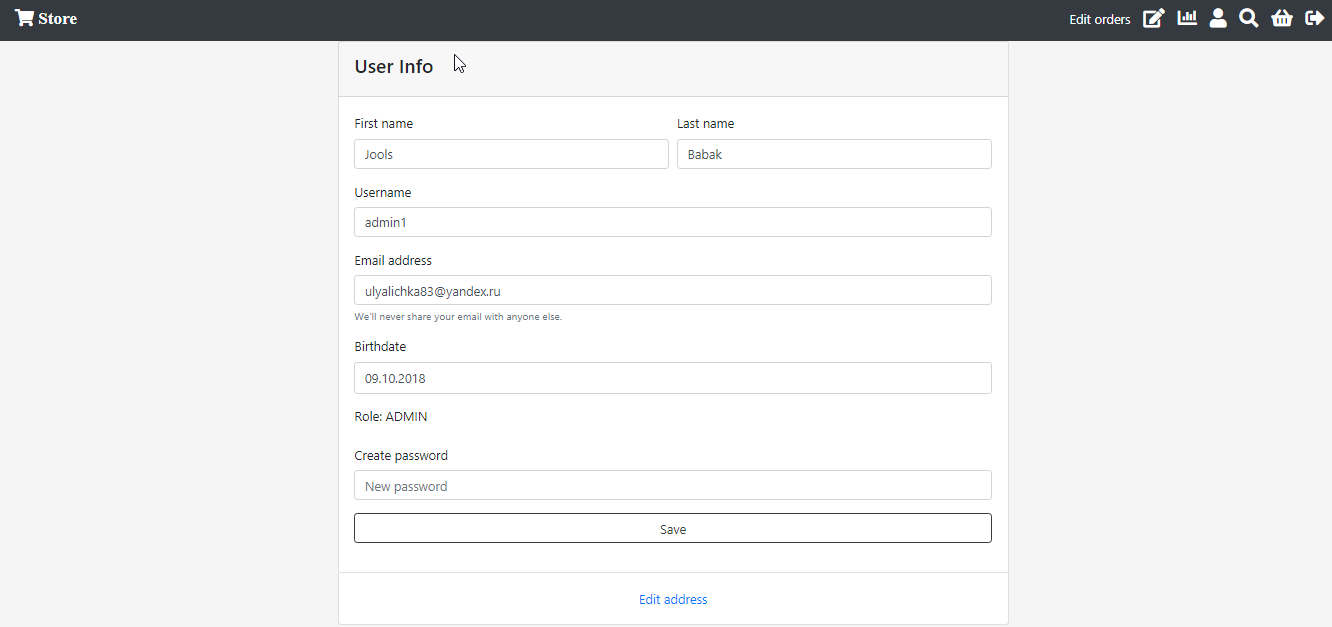


* **navbar.jsp –** navigation bar(used for all pages).
* **orderHistory.jsp –** page with a history of orders.



* **register.jsp -** page for sign up.

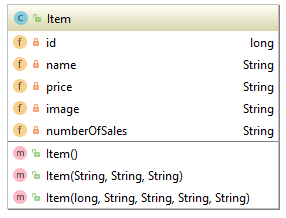


* **search.jsp -** page for searching.
* **statisticsForAdmin.jsp –** page with statistics.
* **userInfo.jsp –** page with user info

# Application structure of the stand.

## Model layer.

Model is only one class that represents information about item:



## Beans.

Enterprise JavaBeans (EJB) is one of several [Java APIs](https://en.wikipedia.org/wiki/List_of_Java_APIs) for modular construction of [enterprise software](https://en.wikipedia.org/wiki/Enterprise_software). EJB is a [server-side](https://en.wikipedia.org/wiki/Server-side) [software component](https://en.wikipedia.org/wiki/Component_(software)) that [encapsulates](https://en.wikipedia.org/wiki/Encapsulation_(object-oriented_programming)) [business logic](https://en.wikipedia.org/wiki/Business_logic) of an application. An EJB [web container](https://en.wikipedia.org/wiki/Web_container) provides a [runtime environment](https://en.wikipedia.org/wiki/Runtime_environment) for web related software components, including [computer security](https://en.wikipedia.org/wiki/Computer_security), [Java servlet lifecycle management](https://en.wikipedia.org/wiki/Java_Servlet#Life_cycle_of_a_servlet), [transaction processing](https://en.wikipedia.org/wiki/Transaction_processing), and other [web services](https://en.wikipedia.org/wiki/Web_services). The EJB specification is a subset of the [Java EE](https://en.wikipedia.org/wiki/Java_EE) specification.

### Singleton bean.

Singleton Beansare business objects having a global shared state within a JVM.

* **JmsConfig -** class to configure JMS connection(ActiveMQ server).

**Apache ActiveMQ** is an [open source](https://en.wikipedia.org/wiki/Open-source_software) [message broker](https://en.wikipedia.org/wiki/Message_broker) written in Java together with a full [Java Message Service](https://en.wikipedia.org/wiki/Java_Message_Service) (JMS) client. It provides "Enterprise Features" which in this case means fostering the communication from more than one client or server. Supported clients include Java via JMS 1.1 as well as several other "cross language" clients.

* **ItemTop** – bean that stores information about top items.

When stand-app got message from JMS server it will make a request to the main application in order to get updated top of items.

### Managed bean.

Managed Bean is a regular Java Bean class registered with JSF. In other words, Managed Beans is a Java bean managed by JSF framework. Managed bean contains the getter and setter methods, business logic, or even a backing bean (a bean contains all the HTML form value).

Managed beans works as Model for UI component. Managed Bean can be accessed from JSF page.

* **ItemList –** backing bean which stores itemTop exemplar and getter for it.

## MessageListener.

A MessageListener object is used to receive asynchronously delivered messages.

* **MessageListenerImpl –** class for listening messages from JMS server.

## Interface.

**header.xhtml –** jsf header with navigation bar.

**stand.xhtml –** jsf page the stand.

# Interaction of the modules.

With the help of ActiveMQ and an integrated into HTML5 Server-Sent Events (SSE) mechanism, at any change of top goods application store will send a message about changing in top of the goods to ActiveMQ- queue. Application-stand is signed to getting declarations from ActiveMQ-queue. So when getting such a message this application will send a request for a new list of the most popular goods. When getting a new list of top goods mechanism SSE will send a request for a new page to all the clients, who have this stand opened in a browser. After updating a client will have a page with actual list of goods displayed.

# Testing.

## Unit-tests.

These classes used mockito and junit frameworks for testing services:

* **AddressServiceTest –** class for testing AddressServiceImpl.
* **CategoryServiceTest –** class for testing CategoryServiceImpl.
* **ItemServiceTest –** class for testing ItemServiceImpl.
* **OrderServiceTest –** class for testing OrderServiceImpl.
* **ParamServiceTest –** class for testing ParamServiceImpl.
* **UserServiceTest –** class for testing UserServiceImpl.

## Selenium tests.

These classes are using for automated web application testing:

* **LoginSeleniumTest –** class for testing sign in process with correct and wrong parameters.
* **SignUpSeleniumTest –** class for testing sign up process.
* **NavbarSeleniumTest –** class for testing navigation bar.
* **EditItemSeleniumTest –** class for testing editing an item by admin.
* **Url –** class with application’s urls for assertion.

# Logging.

Apache log4j framework is used for logging in this application.

log4j.properties:



Part of log file:

2018-12-08 16:41:53 INFO UserService:62 - Getting user by id(called getById(int id))

2018-12-08 16:41:54 INFO CategoryService:59 - Getting list of categories(called listCategories())

2018-12-08 16:41:54 INFO AdminController:259 - User with id: 9 visited editCategories page.

2018-12-08 16:42:28 INFO UserService:170 - Getting authirized user id(called getAuthorizedUserId())

2018-12-08 16:42:28 INFO UserService:62 - Getting user by id(called getById(int id))

2018-12-08 16:42:29 INFO UserService:62 - Getting user by id(called getById(int id))

2018-12-08 16:42:29 INFO OrderService:143 - Getting bucket(called getBucketOrder(int userId))

# Future tasks.

* Add Spring Security support for making application secure while making [authentication](https://en.wikipedia.org/wiki/Authentication), [authorization](https://en.wikipedia.org/wiki/Authorization).
* Add Sonar to understand quality of the code.
* Create payment system.