0.Content

1) introduction

2) List of technologies and frameworks

3) My features

4) The scheme of database

5) Explication and implementation of the model from the task.

6) Modules in the application

7) UI

8) Business logic

9) Entity, DAO, Transactions

10) Screenshots of applications

11) Unit Tests

1. introduction

The application is a fast food web store. A multi-user client-server application with a network connection was created. All data is stored on the server side. Each client can download some data, after each data modification operation, the data must be synchronized with the server. The application contains admin panel, catalog with sorting by tags, registration and cart.

2. List of technologies and frameworks

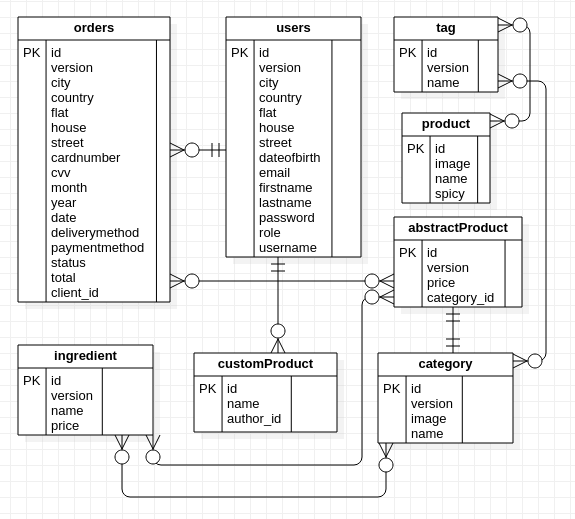
1. Spring (first application)
2. IDE IDEA
3. Wildfly
4. DB – PostgreSql
5. JSP (first application)
6. Maven
7. JPA
8. AS - WildFly
9. EJB (second application)
10. JSF (second application)
11. MQ (for notifications from the server)
12. Rest (for data exchange between the client and the server)
13. Lombok
14. Log4j
15. Hibernate
16. Vue.js

3. My features

Added ability to create and buy custom products from ingredients specified. The product cost is sum of ingredients costs.

4. The scheme of database.

Each entity except Card and Address is inherited from AbstractEntity in which there is id and version fields. Card and Address entities are embeddable. AbstractProduct has JOINED inheritance strategy and CustomProduct and CatalogProduct entities are extending AbstractProduct entity.



5. Explication and implementation of the model from the task.

In my application, according to the task, there is an admin panel where you can update product info, add and remove products,ingredients, tags or categories. All entities are converted to dto before sending to jsp pages.

Admin panel carried out using Vue.js.

Login menu created with Spring Security.

6. Modules in the application

I have 2 modules: the web store module that is responsible for displaying and service main site and the advertising stand module that is responsible for displaying top products. After order submitting or product update the top of products on advertising stand updates.

7. UI

The bootstrap was used for the graphical interface. Also I used Vue.js to refresh page content dynamically.

Here are my resources and its purpose for first module:

common.css – has common styles for whole module

admin.css – has styles for admin panel

customProduct.css – has styles for custom product page

common.js – has common functionality for whole module

admin.js = has functionality for admin panel

cart.js – has functionality for displaying cart and handling order submission

catalog.js – has functionality for displaying products in catalog filtered by tags

customProduct.js – responsible for handling custom product creation

statsAdmin.js – responsible for displaying admin stats page

Here are my resources and its purpose for second module:

main.css – main css file containing all styles

main.js – main js file containing all scripts

8. Business logic.

My services:

**FilesService**

**JavaMessageService**

**OrderService**

**ProductService**

**UserService**

In addition to CRUD requests my services have another methods

**FilesService:**

saveUploadedFiles – save files and return theirs paths

**JavaMessageService**:

sendTopProducts - send top products to active mq topic

**OrderService:**

getRecentOrderDtos — get recent orders dtos

changeStatus — change status of order by id

getOrderDtosByUser — get order dtos by user

getTotalGainDto — get dto containing total gain for week and month

getTopClientsDtos — get top clients dtos

**ProductService:**

getTopProductDtos — get top products dtos

getTagDtosByCategory — get tag dtos by category

detachOrRemoveProduct — detach and try to remove product

removeIngredientFromProduct — remove ingredient from product

removeTagFromProduct — remove tag from product

updateProduct — update existing product

**UserService:**

findUserByEmail — find user by email

changeUser — change existing user

9.Entity, DAO, Transactions

**AbstractEntity:**

Have id and version fields

**\_Order:**

Have client, paymentMethod, deliveryMethod, date, embedded card and address, items, status, total

**AbstractProduct:**

Have price, category and ingredients

**CatalogProduct:**

Have unique name, spicy, image, tags. Inherited from AbstractProduct

**CustomProduct:**

Have name and author fields

**Category:**

Have unique name, image

**Ingredient:**

Have unique name, price and categories

**Tag:**

Have unique name and category

**User:**

Have firstName, lastName, dateOfBirth, username, email, password, embedded addtess, role

**Address:**

Embeddable. Contains country, city, street, home, flat

**Card:**

Embeddable. Contains cardNumber, month, year, cvv

All entities except embeddable Card and Address extend AbstractEntity, so they can inherit id and version.

**CategoryDao:**

CRUD operations with Category

**IngredientDao:**

CRUD operations with Ingredient.

getIngredientsByCategory — get ingredients by category

**OrderDao:**

CRUD operations with Order.

changeStatus — change status of order

getOrdersByUser — get orders by user

isProductInOrder — check whether order contains product

**ProductDao:**

CRUD operations with Product.

**TagDao:**

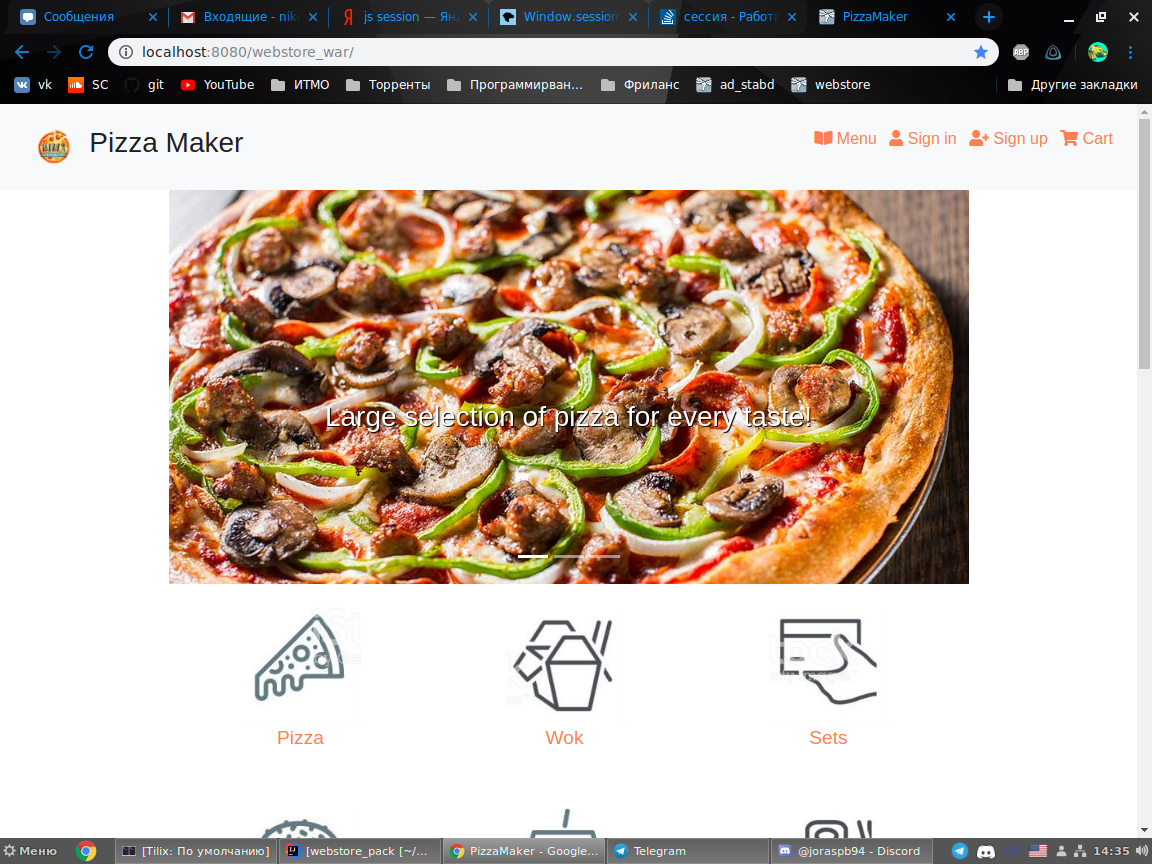
CRUD operations with Tag.

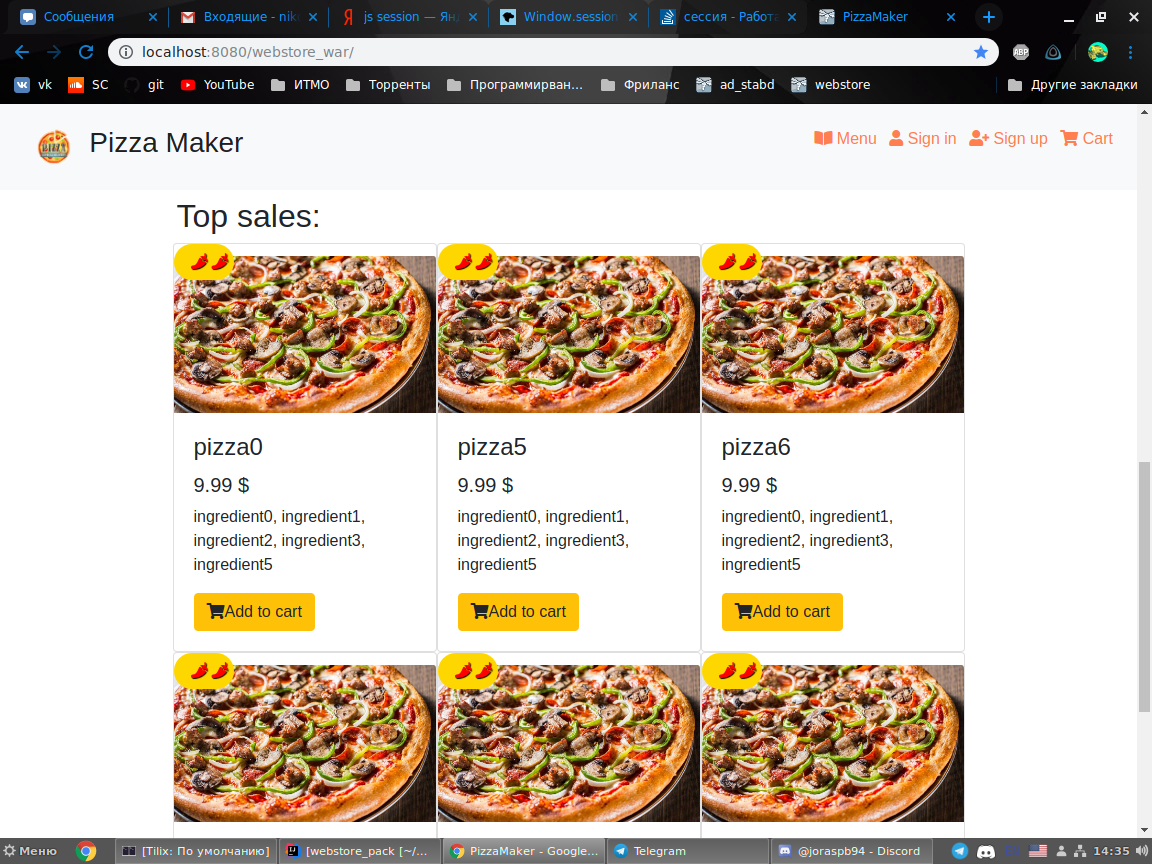
**UserDao:**

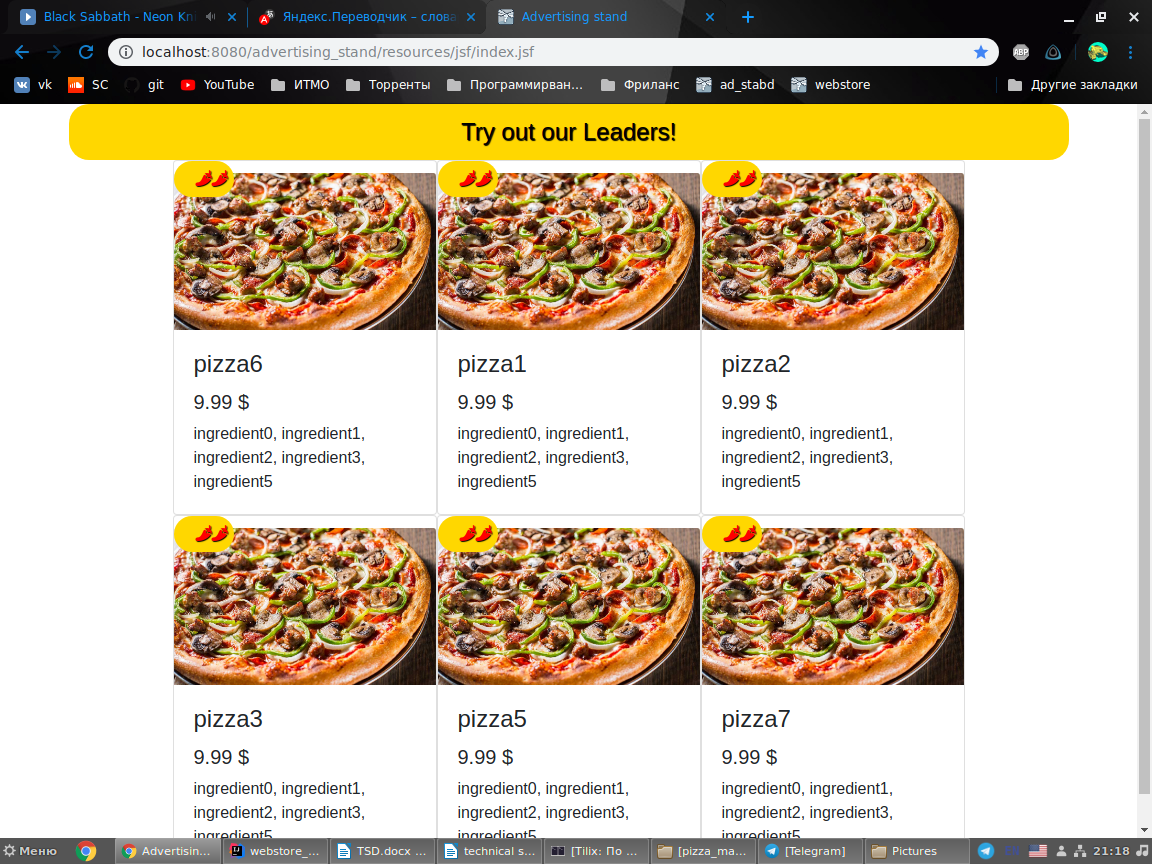
CRUD operations with User

Transactions works with hibernate managed by spring.

10. Screenshots of applications







**11. Unit Tests**

**OrderServiceTests – tests methods of OrderService**

addOrderTest() - tests that total price is correct

getTotalGainDtoTest() - tests that total gain is corect

getTopClientsTest() - tests whether top clients are correct

**ProductServiceTests – tests methods of ProductService**

getTopProductsDtosTest() - tests whether top products are correct

getProductDtosWithTagsTest() - tests that filtration by tags is working

setPriceTest() - tests that price is correct

getTopProductsDtoForAdminTest() - tests that top products with custom products are correct

**UserServiceTests – tests methods of UserService**

changeUserTest() - tests that update of user works correctly