



# FINAL PROJECT PREZENTATION

GINA AGAPESCU



## TECHNOLOGIES USED

- Java
- Spring bootstrap
- Spring Web
- Spring data
- Hibernate
- CSS
- Thymeleaf
- JavaScript

I created the project in IntelliJ IDEA: new Project -> Spring Assistant. Spring Assistant offers the ability to add the dependencies you need to build web applications.

The Initializr offers a fast way to pull in all the dependencies you need for an application and does a lot of the setup for you.

Because it is an application based on Spring, I used Maven as a construction tool.

I choose:

- Spring Boot DevTools(The aim of the module is to try and improve the development time while working with the Spring Boot application. Spring Boot DevTools pick up the changes and restart the application.)
- Spring Web(spring-web provides core HTTP integration)
- Spring Web Services(Its prime focus is to create document-driven Web Services.)
- Thymeleaf(Thymeleaf is a Java-based library used to create a web application. It provides a good support for serving a XHTML/HTML5 in web applications. )
- Spring Data JPA(It is a library/framework that adds an extra layer of abstraction on the top of our JPA provider (like Hibernate).)
- MySQL Driver
- Validation

And now the Pom.xml file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.4.RELEASE</version>
    <relativePath/> <!-- Lookup parent from repository -->
  </parent>
  <groupId>com.gina</groupId>
  <artifactId>express_menu</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>express_menu</name>
  <description>Food delivery project for Spring Boot</description>

  <properties>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
  </dependencies>
</project>
```



The Spring Initializr creates a simple application class for you.

```
package com.gina.expressmenu;

import ...

@SpringBootApplication
public class ExpressMenuApplication {

    public static void main(String[] args) { SpringApplication.run(ExpressMenuApplication.class, args); }

}
```

@SpringBootApplication is a convenience annotation that adds all of the following:

@Configuration: Tags the class as a source of bean definitions for the application context.

@EnableAutoConfiguration: Tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings

@ComponentScan: Tells Spring to look for other components, configurations, and services in the com/gina.expressMenu package, letting it find the controllers.

The main() method uses Spring Boot's SpringApplication.run() method to launch an application.

It also creates the application.properties file.

In which, because I use the MySQL database, I added the connection attributes.

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url = jdbc:mysql://localhost:3306/express_menu_db?serverTimezone=UTC
spring.datasource.username = root
spring.datasource.password = root0000
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
spring.jpa.show-sql = true
```

I create the database in MySQL Workbench

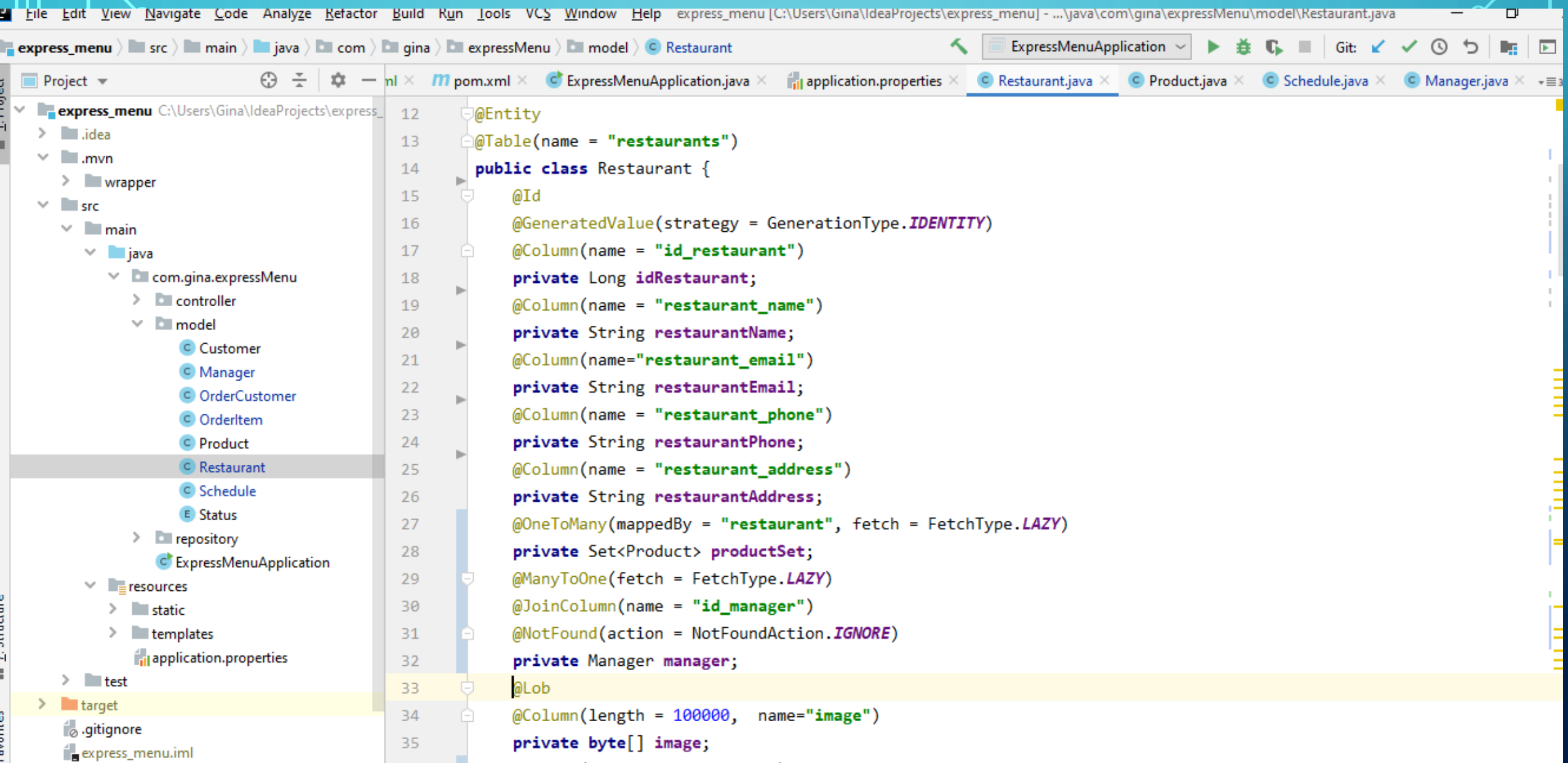


Being a food delivery application, we need to store product information. The products belong to a restaurant, and the restaurant belongs to a manager. For this I created a package. Model in which I created Entity classes: Restaurant, Product, Manager, Schedule. This would be the Provider part of the application.

All of these are defined as plain Java classes with some annotations. By adding “@Entity” before the classes, we are making their instances available to JPA. This will make it easier to store and retrieve instances from the persistent data store when needed. Additionally, the “@Id” and “@GeneratedValue” annotations allow us to indicate the unique ID field for the entity and have its value generated automatically when stored in the database.

As a restaurant you can have many products, we can define a One To Many relationship using the @OneToMany annotation on a Product Set, or a manager can own several restaurants and we use the @ManyToOne annotation on the Manager.

As seen in the next slide





Next step was to create repository package.

With JPA we can define a very useful ManagerRepository interface, RestaurantRepository interface, ProductRepository and ScheduleRepository which allow for easy CRUD operations. These interfaces will allow us to access stored manager, restaurant, product and schedule through simple method calls, such as:

- restaurantRepository.findAll():returns all restaurants
- restaurantRepository.findById(idRestaurant);: return restaurant by given Id

You can also create your own queries:

```
@Query("Select r from Restaurant r where manager.idManager=:idManager ")
List<Restaurant> findAllByManagerId(@Param("idManager")Long idManager);
```

returns all restaurant by given managerId

To create these interfaces, all we need to do is extend the JpaRepository interface.

Example: ManagerRepository

```
package com.gina.expressMenu.repository;

import com.gina.expressMenu.model.Manager;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ManagerRepository extends JpaRepository<Manager, Long> {
    Manager findByEmailAndPassword(String email, String password);
}
```

Next, we can work on the controller for this application. The controller will map request URLs to view templates and perform all necessary processing in between.

Spring Controller annotation can be applied on classes only. It's used to mark a class as a web request handler. It's mostly used with Spring MVC application.

Mapping of URLs to methods is done via simple “@GetMapping” or “@PostMapping” annotations. In this case, every method of the controller is mapped to a URL.

The model parameter of these methods allows data to be passed to the view. In essence, these are simple maps of keys to values.

Each controller method either returns the name of the Thymeleaf template to be used as view, or a URL in a specific pattern (“redirect:”) to redirect to. For example, the methods “addAdmin” and “displayRestaurantsManager” returns the name of a template.

```
@GetMapping("/manager-restaurants/{idManager}")
public String displayRestaurantsManager( @PathVariable("idManager")Long idManager, Model
model){
    Manager manager = (Manager) httpSession.getAttribute("manager");
    model.addAttribute("restaurants", restaurantRepository.findAllByManagerId(idManager));

    return "manager-restaurants";
}
```

```
@PostMapping("/admin/add")
public String addAdmin(@Valid @ModelAttribute("manager") Manager manager, BindingResult
result, Model model) {
    if (result.hasErrors()) {
        return "signUp-admin";
    }

    managerRepository.save(manager);
    HttpSession.setAttribute("manager", manager);
    model.addAttribute("manager", manager);
    model.addAttribute("idManger", manager.getIdManager());

    return "manager-operation";
}
```

Within the controller, the “@Autowired” annotations automatically assigns a valid instance of our defined repository in the corresponding field. This allows access to relevant data from within the controller without having to deal with a lot of boilerplate code.

Project ▾

express\_menu C:\Users\Gina\IdeaProjects\express\_

- .idea
- .mvn
- wrapper
- src
  - main
    - java
      - com.gina.expressMenu
        - controller
          - CustomerHomeController
          - CustomerOrderController
          - HomeController
          - ManagerController
          - ProductController
          - RestaurantController
          - ScheduleController
        - model
          - Customer
          - Manager
          - OrderCustomer
          - OrderItem
          - Product
          - Restaurant
          - Schedule
          - Status
        - repository
          - CustomerRepository
          - ManagerRepository

controller.java x ManagerController.java x ScheduleRepository.java x RestaurantRepository.java x RestaurantController.java x ProductController.java x

```
19 @Controller
20 public class ManagerController {
21     @Autowired
22     private ManagerRepository managerRepository;
23     @Autowired
24     private RestaurantRepository restaurantRepository;
25     @Autowired
26     private OrderCustomerRepository orderCustomerRepository;
27     @Autowired
28     private OrderItemRepository orderItemRepository;
29     @Autowired
30     private CustomerRepository customerRepository;
31     @Autowired
32     private HttpSession httpSession;
33     @GetMapping("/displayAdminHome")
34     public String displayAdminHome(Model model){
35         return "admin-home";
36     }
37     @GetMapping("/displayAddAdmin")
38     @ public String displayAddAdmin( Model model) {
39         model.addAttribute( attributeName: "manager", new Manager());
40         return "signUp-admin";
41     }
42
43     @PostMapping("/admin/add")
```



Finally, we need to define some templates for the views to be generated. For this we are using Thymeleaf, a simple templating engine. The model we used in controller methods is available directly within the templates, i.e. when we enter a contract into “contract” key in a model, we will be able to access the name field as “contract.name” from within the template.

Thymeleaf contains some special elements and attributes that control generation of HTML. They are very intuitive and straightforward. For example, to populate the contents of a input element with the name of a product, all you need to do is define the following attribute (assuming that the key “product” is defined in the model):

```
<input class="txt" th:field="*{productName}" type="text" id="productName" required  
name="productName"/>
```

Similarly to set the “href” attribute of an anchor element, the special attribute “th:href” can be used.

```
<p><a th:href="@{/manager-operation}">Back</a></p>
```

To send a form you must:

```
<form class="form-group" method="post" th:action="@{/products/add}" th:object="${product}"  
enctype="multipart/form-data">
```

We continue to do the same for the Customer side. Meaning , in the model package we created entity classes Customer, OrderItem, OrderCustomer, in the repository package we created CustomerRepository, OrderItemRepository, OrderCustomerRepository interfaces , Controller and Templates for them.

Further for stylization I use CSS:

```
<style>
    @import
url('https://fonts.googleapis.com/css2?family=Arimo:ital,wght@0,400;0,700;1,400;1,700&displa
y=swap');
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}
p{
font-family:Arimo;
    font-size:18px;
}
```

How does the application work?

The Home page allows the user to choose what the application uses, ie Manager or Customer.

*Food delivery application*

Home

Manager

Customer

Contact

## Customer

SignUp

Login

Add to cart

View order

Update, Delete, Save Order



## Manager

SignUp

Login

Add Restaurant

View restaurants

Update, Delete, Save Restaurant

Add Product

View products

Update, Delete, Save Product



Contact us:

☎ 0753 985 264

@gina.agapescu@gmail.com

Follow as

f [Facebook page](#)

in [Linkedin page](#)

"Humor keeps us alive. Humor and food. Don't forget food. You can go a week without laughing." "Joss Whedon"

```
@GetMapping("/displayAdminHome")
public String displayAdminHome(Model model){
    return "admin-home";
}
```

This method return admin-home template



[Home](#)

[+SignUp](#)

[LogIn](#)

[Contact](#)

[All Manager](#)

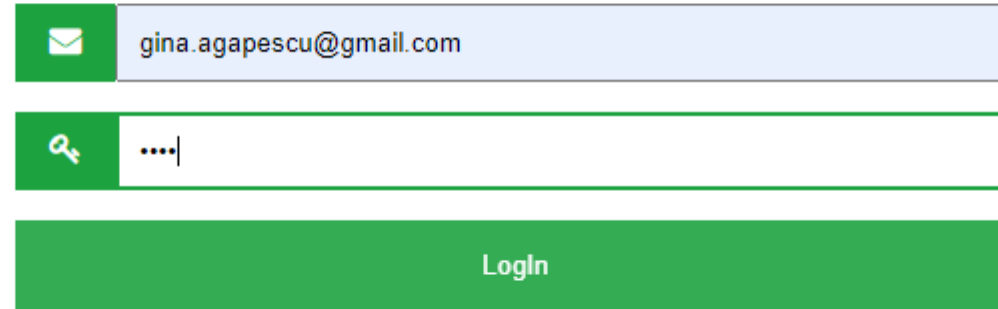
[All Managers](#)

You can add a new Manager via SignUp or you can log in via LogIn.



# LogIn manager

## Register Form



The register form consists of three stacked components. The top component is a light blue rectangular box with a green square icon on the left containing a white envelope symbol; to the right of the icon, the email address 'gina.agapescu@gmail.com' is displayed in a standard black font. The middle component is a white rectangular box with a green square icon on the left containing a white magnifying glass symbol; to the right of the icon, there are four black dots followed by a vertical cursor line, indicating a search or password field. The bottom component is a solid green rectangular button with the word 'LogIn' written in white text in the center.

Once logged in, the manager can add, edit or delete restaurants, products, schedules. He can only do these operations if he is the owner of the restaurant.

This follows from this method.

```
@GetMapping("/manager-restaurants/{idManager}")
public String displayRestaurantsManager( @PathVariable("idManager")Long idManager,
Model model){
    Manager manager = (Manager) httpSession.getAttribute("manager");
    model.addAttribute("restaurants", restaurantRepository.findAllByManagerId(idManager));
    return "manager-restaurants";
}
```

## Manager page

Welcome Gina Agapescu

+ Add restaurant





View restaurants

Edit manager

Now you can add restaurant.

Then you can add products and schedule for the restaurant, you can also edit and delete restaurants, products and schedules, but only where the manager is the owner.

### LIST OF RESTAURANTS

Id	Restaurant name	Add product	View products	Add schedule	View schedule	All orders	Orders by date	Orders by status	Orders by date and status	Update restaurant	D
1	Bad Luck Inn	<a href="#">Add product</a>	<a href="#">View products</a>	<a href="#">Add schedule</a>	<a href="#">View schedules</a>	<a href="#">All orders</a>	<input type="text" value="mm / dd / yyyy"/>  <a href="#">Orders by date</a>	<input type="text" value="AFFECTED"/>  <a href="#">Orders by status</a>	<a href="#">Orders by date and status</a>	<a href="#">Edit restaurant</a>	<a href="#">Delete restaurant</a>
2	La hanul de sub tei	<a href="#">Add product</a>	<a href="#">View products</a>	<a href="#">Add schedule</a>	<a href="#">View schedules</a>	<a href="#">All orders</a>	<input type="text" value="mm / dd / yyyy"/>  <a href="#">Orders by date</a>	<input type="text" value="AFFECTED"/>  <a href="#">Orders by status</a>	<a href="#">Orders by date and status</a>	<a href="#">Edit restaurant</a>	<a href="#">Delete restaurant</a>

[Back](#)

The customer side.

The customer home page opens where the customer can see all the restaurants from which he can order

### *Customer home page*



Bad Luck Inn  
International  
Marti Open:07:00 Close:21:00



La hanul de sub tei  
Romanesc  
Marti Open:08:30 Close:21:00



Vili  
Fast-food  
Marti Open:07:00 Close:22:00









By clicking on a restaurant you will see the available products


Penne polo		lorem ipsun dolores est	18.2
Lasagna		lorem ipsun dolores est	14.5
Tiramisu		lorem ipsun dolores est	9.5
Spaghetti Bolognese		lorem ipsun dolores est	14.5


 Place order  Back

But to make the order effective you will have to register.



enne polo		lorem ipsum dolores est	18.2
asagna		lorem ipsum dolores est	14.5
iramisu		lorem ipsum dolores est	9.5
paghetti Bolognese		lorem ipsum dolores est	14.5

 Place order

 Back

Click ok and a page appears that includes SignUp and Login

# Sign Up

Seban Vlad

 sebanvlad@gmail.com

0721457854

Oradea, Str.Milcovului, Nr.8, Ap.4

 ....|




Sign up 

Now the customer can place the order

Alege produs

Product name	Description	Price	Quantity
Penne polo	lorem ipsum dolores est	18.2	<input type="text" value="1"/>
Lasagna	lorem ipsum dolores est	14.5	<input type="text" value="0"/>
Tiramisu	lorem ipsum dolores est	9.5	<input type="text" value="2"/>
Spaghetti Bolognese	lorem ipsum dolores est	14.5	<input type="text" value="0"/>

 Adauga

 Inapoi

And the Cart look like this:

### Informatii comanda

#### Comanda

**Comanda nr:** 67

**Status**      **AFFECTED**

**Data:**      18.11.2020

**Ora:**      06:34

#### Restaurant

**Restaurant:**    Traviatta

#### Client

**Nume:**      Fitero Liviu

**Adresa:**    Oradea, Str.Soarelui, Nr.5

**Telefon:**    0745124578

#### Produse


Denumire	Pret	Cantitate	Valoare
Penne polo	18.2	1	18.20
Tiramisu	9.5	2	19.00

**Total de plata 37.20**

 **Inapoi**

 **Anuleaza**

 **Editeaza**

 **Salveaza**

From the cart page the customer can edit, delete or save the order



```
@GetMapping("/productsRestaurant/{idRestaurant}")
public String getProductsByRestaurant(@PathVariable("idRestaurant") Long idRestaurant,
                                     Model model) {

    List<OrderItem> orderItemList = new ArrayList<>();
    OrderCustomer orderCustomer = new OrderCustomer();
    Customer customer = (Customer) httpSession.getAttribute("customer");
    for(Product product:productRepository.findAllByRestaurantId(idRestaurant)){
        OrderItem orderItem = new OrderItem(product, orderCustomer);
        orderItemList.add(orderItem);
        Restaurant restaurant = orderItem.getProduct().getRestaurant();
        orderCustomer.setRestaurant(restaurant);
    }

    orderCustomer.setOrderItemList(orderItemList);
    orderCustomer.setCustomer(customer);
    orderCustomer.setDate(LocalDate.now());
    orderCustomer.setStatus(Status.AFFECTED);
    orderCustomer.setTime(LocalTime.now());
    model.addAttribute("orderCustomer", orderCustomer);
    model.addAttribute("customer", customer);

    return "order-items";
}
```

@Transactional

@PostMapping("orders/add")

```
public String saveOrder(@Valid @ModelAttribute("orderCustomer")OrderCustomer orderCustomer,
                        Model model, BindingResult result){
    if (result.hasErrors()) {
        return "orders-item";
    }
    orderCustomer.getOrderItemList().stream().forEach(orderItem -> orderItem.setOrderCustomer(orderCustomer));
    List<OrderItem> nonZeroOi = new ArrayList<>();
    for(OrderItem orderItem :orderCustomer.getOrderItemList()) {
        if(orderItem.getQuantity() > 0) {
            nonZeroOi.add(orderItem);
        }
    }
    orderCustomer.setOrderItemList(nonZeroOi);
    double total = 0;
    for(OrderItem orderItem:nonZeroOi){
        total += orderItem.getQuantity()* orderItem.getProduct().getPrice();
    }
    orderCustomerRepository.save(orderCustomer);
    model.addAttribute("orderCustomer", orderCustomer);
    orderCustomer.setStatus(Status.AFFECTED);
    model.addAttribute("restaurantName", orderCustomer.getRestaurant().getRestaurantName());
    Long id = orderCustomer.getCustomer().getIdCustomer();
    Customer customer = customerRepository.findById(id).get();
    Restaurant restaurant = restaurantRepository.findById(orderCustomer.getRestaurant().getIdRestaurant()).get();
    model.addAttribute("customer",customer);
    model.addAttribute("restaurant", restaurant);
    model.addAttribute("total", total);
    return "cart";
}
```

The background is a blue gradient with decorative white circuit-like lines in the corners. These lines consist of straight segments and small circles, resembling a stylized electronic circuit board.

Thank you for your attention