МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем

та мереж

**Звіт**  до лабораторної роботи №3

з дисципліни “Екстремальне програмування ”

**Сервіси та роутінг в додатках Angular Framework/ Models in Java**

Виконала: студентка гр. КН-311 Боштан А.О.

Прийняв:

Щербак С.С.

Львів - 2020

Теоретичні відомості

Сервіси в Angular представляють досить широкий спектр класів, які виконують деякі специфічні завдання, наприклад, логування, роботу з даними і т.д.

На відміну від компонентів і директив сервіси не працюють з уявленнями, тобто з розміткою html, не роблять на неї прямого впливу. Вони виконують строго певне і досить вузьке завдання.

Стандартні завдання сервісів:

* Надання даних з додатком. Сервіс може сам зберігати дані в пам'яті, або для отримання даних може звертатися до будь-якого джерела даних, наприклад, до сервера.
* Сервіс може представляти канал взаємодії між окремими компонентами програми
* Сервіс може інкапсулювати бізнес-логіку, різні обчислювальні завдання, завдання по логувані, які краще виносити з компонентів. Тим самим код компонентів буде зосереджений безпосередньо на роботі з поданням. Крім того, тим самим ми також можемо вирішити проблему повторення коду, якщо нам буде потрібно виконати одну і ту ж задачу в різних компонентах і класах.

[Angular Router](https://angular.io/guide/router) — потужний JavaScript роутер, створений командою розробників Angular. Його можна встановити з пакета @angular/router. Ви отримуєте бібліотеку для роутингу, з якою можна визначити декілька елементів router-outlet, декілька стратегій зіставлення маршруту з рядком запиту, з легкістю отримувати параметри маршруту та використовувати хуки для авторизованого доступу до компонентів.

Angular Router — основа платформи Angular. Роутер дозволяє створювати односторінкові застосунки з кількома представленнями та навігацією ними.

[Router-Outlet](https://angular.io/api/router/RouterOutlet) — директива з бібліотеки роутера, що визначає місце компонента. Зіставлення маршруту проходить через URL браузера. Ви можете додати декілька outlets до вашого застосунку на Angular і реалізувати складніші сценарії роутингу.

Будь-який зіставлений роутером компонент відображається як sibling-компонент для router-outlet.

Маршрути — об'єкти, що обов'язково вміщують шлях та атрибути компонента (або [redirectTo](https://angular.io/api/router/Route" \l "redirectTo" \t "_blank) шлях). Шлях належить до частини URL, що визначає унікальне представлення для відображення, а атрибути компонента визначають який компонент Angular буде пов'язаний зі шляхом. З визначенням маршруту через статичний метод RouterModule.forRoot(routes) Router може перенаправляти користувача до певного представлення.

Кожен [маршрут](https://angular.io/api/router/Route) зіставляє URL-шлях з компонентом.

Шлях може бути пустим. Це означає, що він використовується за замовчуванням і, зазвичай, при запуску.

Шлях може приймати шаблон рядка(\*\*). Роутер обере цей маршрут, якщо URL не збігається з жодним шляхом для визначених маршрутів. Такий підхід можна застосувати для «Not Found» або перенаправлення до певного представлення, якщо не було знайдено відповідності.

Виконання лабораторної роботи

Даний проект зроблений за допомогою мови програмування Java, JS, HTML, CSS.

Для дотримання MVC патерну, у даному проекті було створено такі моделі:

Hall.java

**package** model;  
  
**public enum** Hall {  
 MAIN(100), UNDERGROUND(50), OUTDOOR(70), TERRACE(120);  
  
 **private double** price;  
  
 Hall(**double** price) {  
 **this**.price = price;  
 }  
  
 **public double** getPrice() {  
 **return** price;  
 }  
}

Order.java

**public class** Order {  
 **private** User user;  
 **private** Date date;  
 **private int** personAmount;  
 **private boolean** specialOccasion;  
 **private boolean** separateRoom;  
 **private** Hall hall;  
 **private** String additionalAdvise;  
  
 **public** Order(User user, String date, **int** personAmount, String specialOccasion, String separateRoom, Hall hall, String additionalAdvise) {  
 **this**.user = user;  
 **this**.date = Date.valueOf(date);  
 **this**.personAmount = personAmount;  
 **this**.specialOccasion = (specialOccasion.equals(**"Yes"**)) ? **true** : **false**;  
 **this**.separateRoom = (separateRoom.equals(**"Yes"**)) ? **true** : **false**;  
 **this**.hall = hall;  
 **this**.additionalAdvise = additionalAdvise;  
 }  
  
 **public** User getUser() {  
 **return** user;  
 }  
  
 **public** Date getDate() {  
 **return** date;  
 }  
  
 **public int** getPersonAmount() {  
 **return** personAmount;  
 }  
  
 **public boolean** isSpecialOccasion() {  
 **return** specialOccasion;  
 }  
  
 **public boolean** isSeparateRoom() {  
 **return** separateRoom;  
 }  
  
 **public** Hall getHall() {  
 **return** hall;  
 }  
  
 **public** String getAdditionalAdvise() {  
 **return** additionalAdvise;  
 }  
  
 **public boolean** checkFreeDate() {  
 **if** (AdditionalInfo.getBusyDates().contains(date)) **return false**;  
 **else return true**;  
 }

User.java

**package** model;  
  
**public class** User {  
 **private** String name;  
 **private** String surname;  
 **private** String email;  
 **private** String phone;  
  
 **public** User(String name, String surname, String email, String phone) {  
 **this**.name = name;  
 **this**.surname = surname;  
 **this**.email = email;  
 **this**.phone = phone;  
 }  
  
 **public** String getName() {  
 **return** name;  
 }  
  
 **public** String getSurname() {  
 **return** surname;  
 }  
  
 **public** String getEmail() {  
 **return** email;  
 }  
  
 **public** String getPhone() {  
 **return** phone;  
 }  
}

Hall – enum, у які містяться варіанти залів для резервації, Order – дані про замовлення, User – дані про користувача.

Також тут було реалізовано логіку підрахунку вартості резервації столика відповідно до вхідних параметрів, тобто побажань клієнта.

**public double** calculatePrice() {  
 **boolean** isVipUser = **false**;  
 **for** (User vipUser : AdditionalInfo.getListOfVipUsers()) {  
 **if** (vipUser.getPhone().equals(user.getPhone())) {  
 isVipUser = **true**;  
 **break**;  
 }  
 }  
 **double** initialPrice = hall.getPrice();  
 **if** (separateRoom) initialPrice \*= 3; *//окрема кімната втричі дорожча* **if** (personAmount >= 5) initialPrice -= (personAmount / 5) \* 10; *//за кожного 5-го відвібувача знижка 10$* **if** (isVipUser) {  
 initialPrice \*= 0.9; *//vip клієнти мають 10% знижки* } **else** {  
 **if** (specialOccasion) initialPrice \*= 0.98; *//при святі ресторан дарує не vip клієнтам знжку 2%* }  
 **return** initialPrice;

}

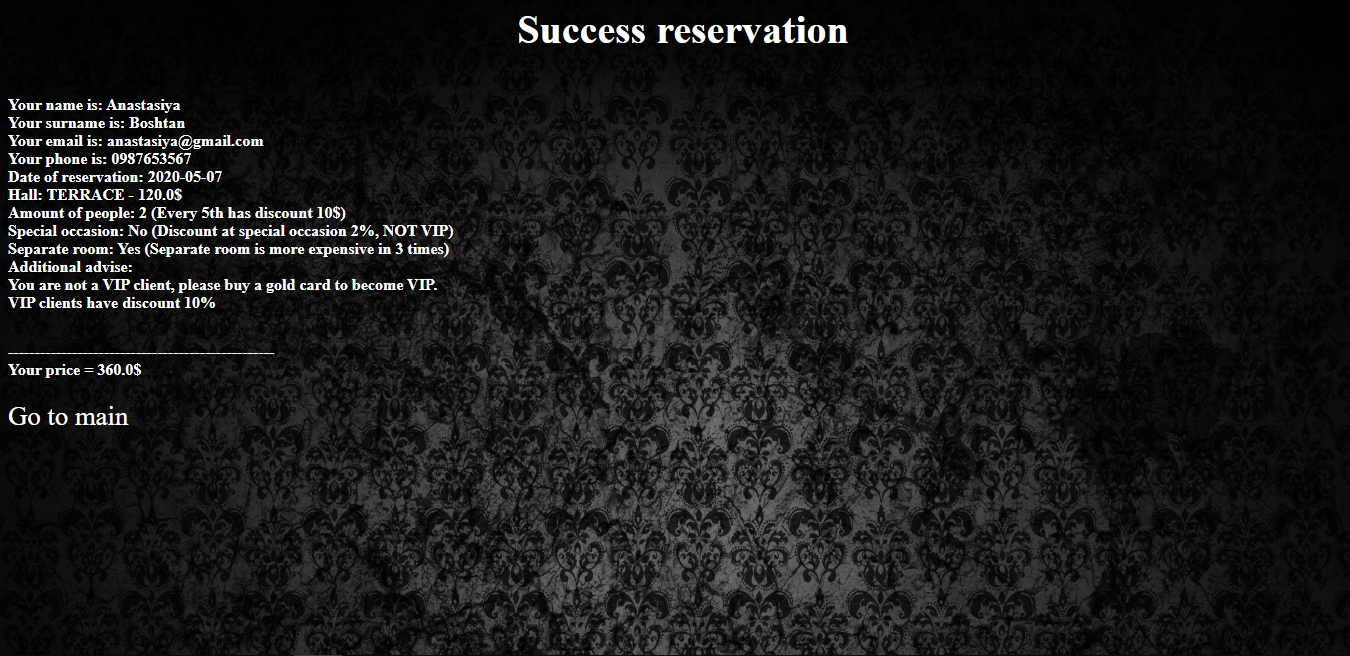


Рис.1 Підрахунок суми до сплати за резервацію

Якщо ж говорити про Angular додатки, то уся логіка програми робиться за допомгою компонент та сервісів. Компоненти вже створені. Тому залишилось створити лише сервіси.

Для того щоб створити сервіси потрібно виконати команду в терміналі

ng generate service app/data

Але для цього обов’язково потрібно в терміналі перейти в проект.

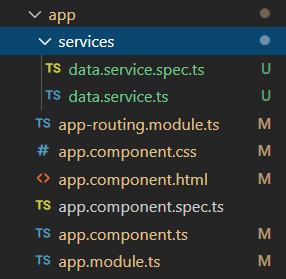


Рис.2 Структура app разом з сервісами

Отже, пишемо код в сервісі, після чого передаємо його в компонети.

data.service.ts

import { Injectable } from '@angular/core';

@Injectable({

  providedIn: 'root'

})

export class DataService {

  first="Lviv steak house";

  //second=2;

  constructor() { }

  getFirstData(){

    return this.first;

  }

}

app.component.ts

import { Component } from '@angular/core';

import { DataService } from './services/data.service';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'StakeHouseApp';

  url="";

  constructor(private data: DataService){}

  sendUrl(){

    this.url=this.data.getFirstData();

  }

}

Для здійснення роутингу в межах проекту є файл app.routing.module.ts, який призначений для цього.

Його зміст має такий вигляд:

import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }

Якщо ми хочемо здійснити якусь конфігурацію шляхів, тоді вона має бути здійснена за таким шаблоном:

const routes: Routes = [

   { path: "", pathMatch: "full", redirectTo: "main" }

  ];

Висновок: у даній лабораторній роботі було створено моделі в Java, а також ознайомлено з сервісами та роутингом в Angular.