UNIVERSITATEA ALEXANDRU IOAN CUZA IAŞI

### FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENŢĂ

Hotel Manager

propusă de

***Adrian Ștefan***

Sesiunea: *iulie, 2019*

Coordonator ştiinţific

##### Colaborator Florin Olariu

UNIVERSITATEA ALEXANDRU IOAN CUZA IAŞI FACULTATEA DE INFORMATICĂ

Hotel Manager

***Adrian Ștefan***

Sesiunea: *iulie, 2019*

Coordonator ştiinţific

###### **Colaborator Florin Olariu**

DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licenţă cu titlul “*Hotel Manager*” este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învăţământ superior din ţară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

* toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi deţin referinţa precisă a sursei;
* reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa precisă;
* codul sursă, imagini etc. preluate din proiecte *open*-*source* sau alte surse sunt utilizate cu respectarea drepturilor de autor şi deţin referinţe precise;
* rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi, *data*

Absolvent *Adrian Ștefan*

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul “*Hotel Manager*”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea Alexandru Ioan Cuza Iaşi să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi, *data*

Absolvent *Adrian Ștefan*

## Acord privind proprietatea dreptului de autor

Facultatea de Informatică este de acord ca drepturile de autor asupra programele-calculator, format executabil şi sursă, să aparţină autorului prezentei lucrări, *Prenume Nume.*

Încheierea acestui acord este necesară din următoarele motive:

*[Se explică de ce este necesar un acord, se descriu originile resurselor utilizate în realizarea produsului-program (personal, tehnologii, fonduri) şi aportul adus de fiecare resursă.]*

Iaşi, *data*

Decan *Adrian Iftene* Absolvent *Adrian Ştefan*

**Cuprins**

Momentan scriu despre tot ce consider ca trebuie si apoi o sa organizez6

Capitolul 1: Tehnologii folosite4

Descriere generala5

Type chapter title (level 3)6

Capitolul 2: Tehnologii folosite4

Descriere generala5

Type chapter title (level 3)6

**Introducere**

Tema licentei este construirea unei aplicații web care are ca și scop managemntul unui hotel. Toți utilizatori aplicației vor avea posibilitatea să vadă informații generale despre hotel și serviciile oferite de acesta. Principalele utilități ale aplicației pentru client sunt cea de rezervare a camerelor și de plasare a comenzilor de tip room-service. Accesul spre serviciile de rezervare de camere si comenzi room-service se face prin autentificarea cu un cont personal.

Aplicația oferă posibilitatea înregistrări cu trei tipuri de conturi, admin, angajat și utilizator. Ca și admin poți crea diferite categorii de camere. Odată creată o categorie poti adăuga una sau mai multe camere cu categoria respectivă acestea devenind disponibile spre rezervare. De asemenea, admini au și posibilitatea de a adauga diverse tipuri de produse pentru a putea fi comandate prin serviciul de room-service. Rezervarea camerelor poate fi facută de toți utilizatorii logați și plata se face prin PayPal pentru finalizarea rezervării. Comenzile room-service pot fi făcute de utilizatorii care sunt logați și au o rezervare de camera în momentul efectuării comenzii. Conturile de tip angajat pot vedea toate rezervările și comenzile de room-service. In plus, aceștia pot face rezervări și comenzi de room-service pentru clienți care nu au cont personal, plata pentru aceste rezervări poate fi de orice tip, nu doar PayPal.

O altă utilitate importantă aplicației este cea de generare a unui cod de bare la momentul înregistrării. Cu acest cod de bare clienții vor putea intra în camerele unde au rezervări pe parcursul rezervării. Fiecare camera a hotelului va avea un dipozitiv capabil sa scaneze codul de bare, iar în cazul în care este scanat codul de bare corect va descuia camera. Codul de bare necesar descuierii camerei este actualizat în funcție de rezervarea activă pentru acea camera.

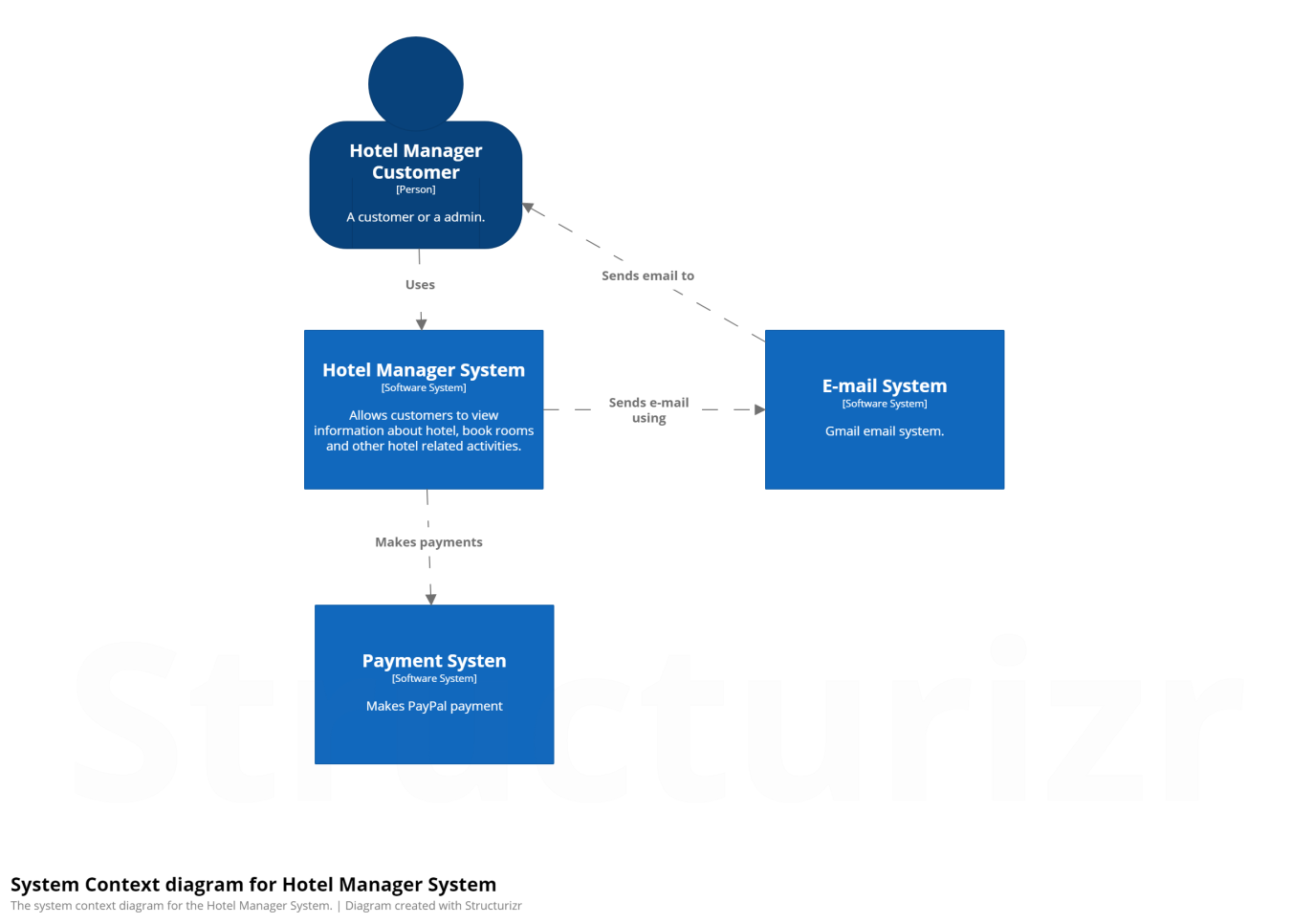
Motivul pentru care am ales această temă de licență este că, deși există multe hoteluri cu aplicații care să ofere posibilitatea rezervării unei camere, nu am găsit nici o aplicație care să faciliteze toată experiența ta la acel hotel prin intermediul aplicației. La aceste hoteluri chiar dacă îți faci rezervare la cameră online odată ajuns la hotel tot trebuie să mergi la recepție pentru a primi cheia camerei ceea ce în anumite cazuri poate duce la așteptarea la timp pierdut prin așteptarea la cozi lungi la recepție.

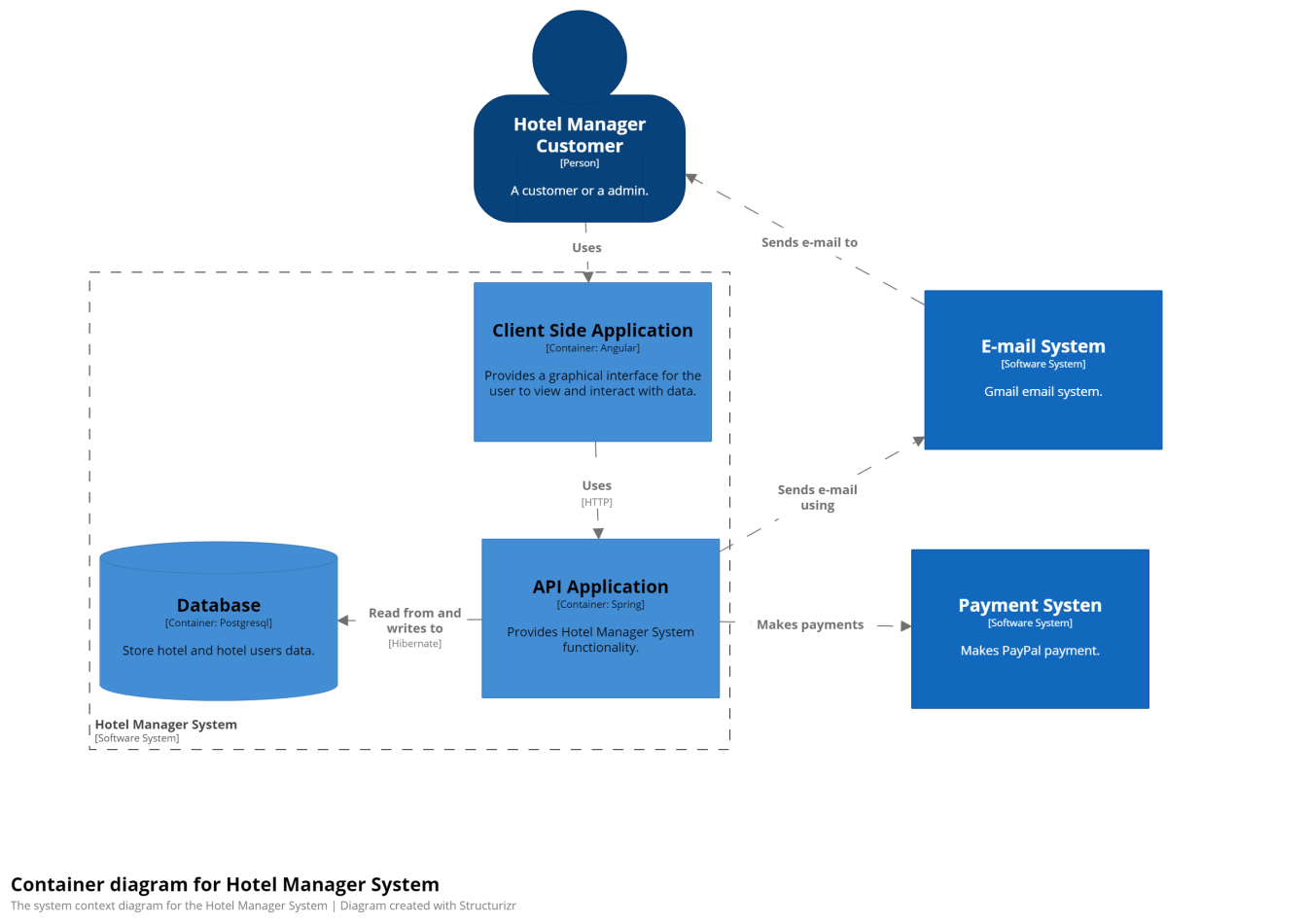
Alt element care nu l-am văzut la nici o altă aplicatie este posibilitatea plasării unei comenzi de room-service, și consider ca este mai mult pe placul lumii să plaseze o comandă prin internet și să poată vedea statusul comenzii decât prin intermediul telefonului sau în alte moduri.

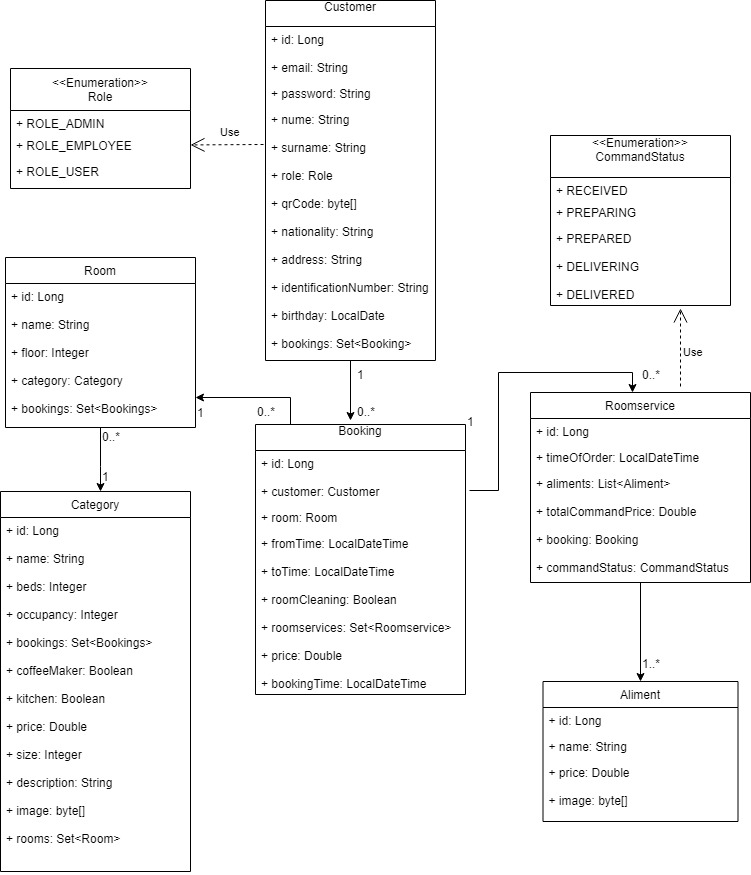
De asemenea, aplicația foloseste și algoritmi pentru a face prețul rezervării să varieze în funcție de diverse criterii, printer care și oferirea de reduceri în funcție de rezervările anterioare.

Așadar, consider că cel mai mare plus adus de aplicația mea este oferirea posibilității cazării și cererii serviciilor la un hotel doar prin intermediul aplicației, asta făcând atât cazarea cât și perioada de cazare la hotel mult mai simplă și mai plăcută pentru client.

**Diagrame arhitecturale**

****

****

****

**Tehnologii folosite**

Proiectul este împărțit în două aplicații, o aplicație este un API care se ocupă de prelucrarea informațiilor, cât și stocarea si preluarea informațiilor in baza de date, această parte este numită partea de back-end. A doua aplicație a proiectului este interfața grafică a proiectului, aceasta făcând apeluri spre partea de back-end și afisându-le în interfața grafică, această parte este numită front-end.

Tehnologiile folosite pentru back-end sunt limbajul de programare Java 8, frameworkul Hibernate pentru lucrul cu baza de date și frameworkul Spring. Baza de date folosită este PostgreSQL. Pentru aducerea de dependințe externe în proiect este folosit sistemul Maven. Spre exemplu, pentru folosirea frameworkurilor este folosit Maven. Pentru a folosi Maven se creează în proiect un fișier numit “pom.xml”. În acest fișier se pot face mai multe configurări, dar stric pentru aducerea de dependințe externe este folosit tagul <dependencies> </dependencies>, în interiorul căruia se adaugă diverse dependințe preluate de pe internet.

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<version>2.1.3.RELEASE</version>

</dependency>

</dependecies>

Hibernate este un framework care se ocupă cu maparea claselor folosite în cod la tabele în baza de date, astfel lucrând cu obiecte și clase direct pentru a prelua și adauga informații în baza de date. Pentru a folosi acest framework este folosit Java Persistence API, numit JPA, aceasta fiind specificția unei interfețe de programare a aplicațiilor care descrie managementul datelor relaționale în aplicații foloșind Hibernate pentru lucrul cu baza de date. JPA foloseste diverse adnotări și interfețe pentru lucrul cu baza de date. Unele din cele mai importante adnotări sunt @Entity folosită asupra unei clase pentru a mapa acea clasă la o tabelă în baza de date, @OneToOne, @OneToMany, @ManyToOne, @ManyToMany pentru a face legături între diferite tabele și adnotarea @Transactional pentru a specifica ca într-o anumită metodă sau clasă să se execute codul sub formă de tranzacție. De asemenea, este necesar ca această clasă să aibă metode de setare și aducere de informații (get și set).

@Entity

public class Aliment {

@Id

@GeneratedValue

private Long Id;

private String name;

private Double price;

@Lob

private byte[] image;

}

Spring este un framework cu multe utilități, cele mai importante utilități folosite în proiectul meu sunt Spring Data, Spring Boot, mecanism de inversiune de control, mecanism pentru cereri web de tipul verbelor HTTP cunoscute și Spring Security.

Spring Data oferă mai multe funcționalități peste JPA, adică multe metode pentru preluarea și stocarea infromațiilor în baza de date. În proiectul meu am folosit JpaRepository, aceasta este o interfață cu diferite implementări standard pentru lucrul cu baza de date, spre exemplu findById(id) pentru preluarea obiectului cu id-ul cerut din baza de date, findAll() pentru preluarea tuturor obiectelor din tabela mapată la clasa cerută, save(obiect) pentru adăugarea unui obiect în baza de date. De asemenea, JpaRepository oferă posibilitatea de a crea întrebări specifice la baza de date prin crearea unei metode în JpaRepository deasupra la care este pusă adnotarea @Query(“”) în interiorul căreia este scrisă întrebarea specifică dorită spre baza de date.

@Repository

public interface CategoryRepository extends JpaRepository<Category, Long> {

@Transactional

@Query("SELECT c FROM Category c LEFT JOIN FETCH c.rooms r LEFT JOIN FETCH r.bookings b ")

Set<Category> findAllCategoriesFetchingRoomsFetchingBookings();

}

Spring Boot este un mecanism folosit pentru crearea configurațiilor aplicației și pentru rularea aplicației.

Mecansimul de inversiune de control este realizat prin injectare de dependințe. Pentru a putea injecta un serviciu trebuie declarată o interfață și pusă o adnotare de tip @Service peste o clasă care implementează acea interfață dupa care această clasă trebuie declarată în constructorul clasei și deasupra constructorului trebuie pusă adnotarea @Autowired.

private AlimentService alimentService;

private AlimentMapper alimentMapper;

@Autowired

public AlimentController(AlimentService alimentService, AlimentMapper alimentMapper) {

this.alimentService = alimentService;

this.alimentMapper = alimentMapper;

}

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class);

}

}

Mecanismul pentru cereri web este realizat prin diverse adnotări pentru a zice ca o metoda este o reprezentare pentru un verb HTTP. Pentru a defini o clasă ca o clasă cu verbe HTTP este pusă adnotarea @RestController deasupra ei și adnotarea @RequestMapping(“”) pentru a zice calea la care vom găsi verbele HTTP definite în clasă. Adnotările @PostMapping, @GetMapping, @PutMapping, @PatchMapping sunt folosite deasupra unei metode din clasă pentru a marca acea metodă ca unul din verbele HTTP. Pe lânga aceste adnotări mai sunt folosite și adnotările @PathVariable, @RequestBody, @ReqeustParam pentru folosirea de varabile în URL, informații transime in corupl cererilor web sau ca parametri.

@GetMapping(value = "/{id}")

public BookingDto getById(@PathVariable Long id) {

return bookingService.getBookingDtoById(id);

}

Spring Security este folosit pentru autentificarea și securizarea aplicației de back-end prin generarea de tokeni la autentificare și folosirea acestor tokeni pentru a accesa diferite cereri web în funcție de gradul lor de securitate.

Tehnologiile folosite pentru front-end sunt frameworkul Angular 7, limbajul de programare TypeScript, limbajul de scripting HTML5 si limbajul de modelare CSS3. De asemenea, sunt folosite și serviciile Bootstrap, Angular Material și PrimeFaces pentru folosirea de elemente de interfață grafică deja definite.

Angular este un framework care folosește ca bază module, componente și servicii. Modulele sunt folosite pentru organizarea codului, acestea fiind compuse din componente, servicii și/sau pipeuri. Componentele sunt create dintr-un fișier HTML, unul CSS si unul TypeScript din care se formează o pagină sau o parte a unei pagini web. Fisierele HTML și CSS oferă interfața grafică și fisierul TypeScript face apleuri folosind servicii la aplicația de back-end pentru preluare de date din baza de date și apoi prelucrarea acestora pentru a fi afișate în interfața grafică.

**Arhitectura proiect**

Proiectul are o arhitectura pe trei nivele, prezentare, aplicație și date.

Nivelul prezentare este interfața grafică a proiectului, deci acesta este reprezentat de aplicația de front-end. Acesta face cereri la aplicația de back-end și trimite informații browserelor în forma de HTML și CSS folosind frameworkul Angular.

Nivelul aplicație este aplicația de back-end. Aceasta folosește controllere pentru a prelua cererile web de la aplicația de front-end, apoi folosește servicii penru a prelucra informațiile și a adauga sau pentru a prelua anumite informații din baza de date.

Nivelul de date este reprezentat de baza de date, care se ocupa de păstrarea și recuperarea informatiilor de către utilizator. Baza de date folosită in proiect este PostgreSQL.

Aplicația de front-end este împărțită în trei module, core, modules și shared. În modulul core se găsesc modelele pentru lucrul cu date primite de la back-end, serviciile care fac cererile web și gărzile necesare pentru securitate în funcție de gradul de acces al utilizatorului. Modulul modules conține la rândul lui diferite module organizate care conțin componente necesare pentru a crea o pagina web. În final, ultimul modul este shared care conține informații comune pentru celelalte module cum ar fi componentele cu header și footer, pipeuri serviciu pentru validări și aducere de diverse module externe.

Aplicația de back-end este împarțită la rândul ei pe trei nivele, nivelul de preluare a cererilor web de la aplicația de front-end folosind controllere și apoi folosind servicii pentru prelucararea datelor, serviciile reprezentând al doilea nivel. Ultimul nivel constă în lucrul cu bazele de informații, adică preluarea informațiilor și adăugarea acestora în baza de date, prin intermediul interfeței JpaRepository. Nivelul de lucrul cu bazele de date este folosit de nivelul de servicii. Atât folosirea serviciilor cât și a interfețelor se face prin injectare de dependințe.

Pentru lucrul cu baza de date se folosesc modele mapate la tabele din baza de date (la figura x se poate vedea diagrama de claselor de modele). Dar pentru preluarea de informații din requesturile web și pentru returnare de informații sunt folosite alte clase asemănătoare cu modelel din baza de date, numite DTO. Motivul folosirii acestor clase este dat de faptul ca modelele trebuie să aibă aceeași formă ca și tabela din baza de date iar uneori putem să avem nevoie să primim sau sa oferim informații diferite fața de cele din model. Pentru a face conversia dintre aceste clase, adică de la model la DTO și invers, este folosit câte un serviciu numit mapper care conține metodele necesare pentru a mapa la tipul care avem nevoie. Procesul de mapare este făcut prin crearea unui obiect de noul tip și instanțierea acestuia cu valorile din obiectul dinspre care este mapat.

@Service

public class AlimentMapper {

public Aliment map(AlimentDto alimentDto) {

Aliment aliment = new Aliment();

aliment.setId(alimentDto.getId());

aliment.setName(alimentDto.getName());

aliment.setPrice(alimentDto.getPrice());

aliment.setImage(alimentDto.getImage());

return aliment;

}

}

public class AlimentDto implements Serializable {

private Long id;

private String name;

private Double price;

private byte[] image;

}

Aplicația folosește excepții proprii care extind clasa RuntimeException și le este dat un cod de status HTTP. Spre exemplu, dacă se caută o camera după un id în baza de date și nu este găsită acea cameră este aruncată excepția RoomNotFoundException cu codul de status 404.

@ResponseStatus(value = HttpStatus.NOT\_FOUND, reason = "Room not found")

public class RoomNotFoundException extends RuntimeException {

public RoomNotFoundException(String message) {

super(message);

}

public RoomNotFoundException(Long id) {

this("Room with id " + id + " doesn't exist.");

}

}

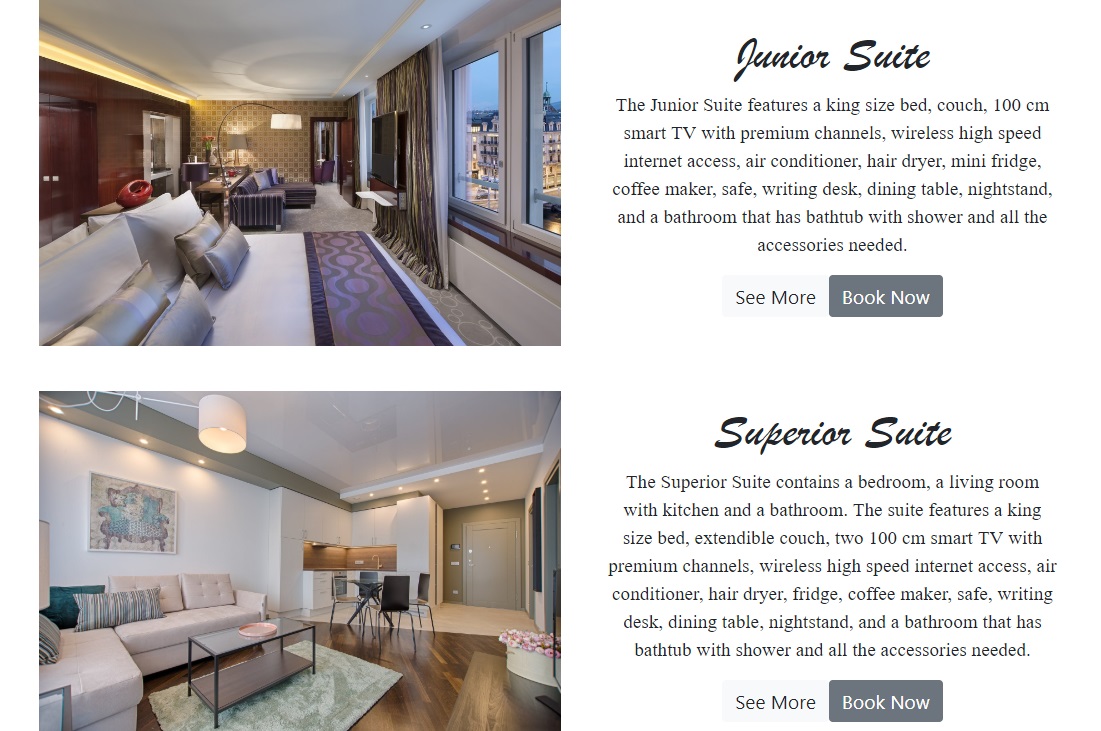
De asemnea, sunt folosite și două sisteme externe, unul pentru plata prin PayPal și unul pentru trimiterea de emailuri prin Gmail.

Pentru partea de securitate și autorizare este folosit Spring Security pentru a genera un token la autentificare în funcție de rolul utilizatorului (admin pentru utilizatori de tip admini, employee pentru utilizatori care sunt angajati sau user pentru utilizatori normali care folosesc aplicația pentru a face o rezervare de cameră sau pentru a comanda room-service). Apoi sunt configurate cererile web pentru a oferi securitatea necesară în funcție de caz, securitatea acestora fiind verificată dupa informațiile din token.

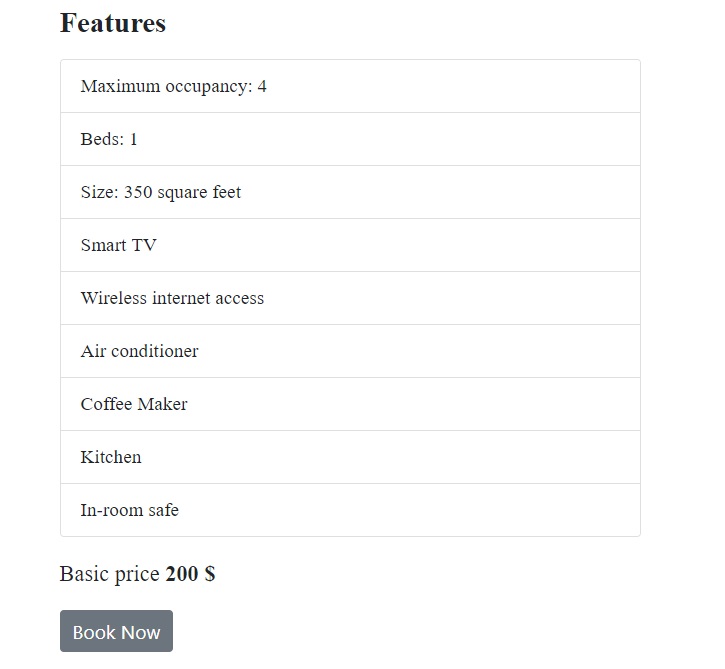
Aplicația respectă Regulamentul General Privind Protecția Datelor prin codificarea datelor introduse în baza de date și decodificare acestora la preluarea datelor din baza de date. Pentru o securitate și mai buna pentru parolă este folosită o funcție hash înainte de a fi introdusă în baza de date.

**Rezervare camera**

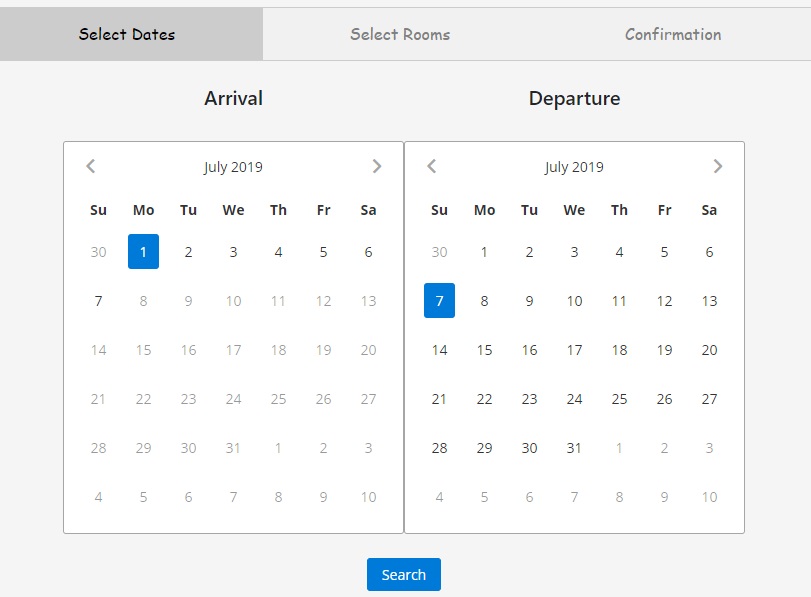
Pentru a rezerva o cameră la hotel un utilizator întai trebuie să acceseze pagina cu toate categoriile de camere disponibile spre rezervări, această pagină se poate accesa de pe pagina de acasă a aplicației unde se găsește o descriere a hotelului cat și a utilitătilor oferite de acesta sau din bara de meniu a aplicației la Accomodations.



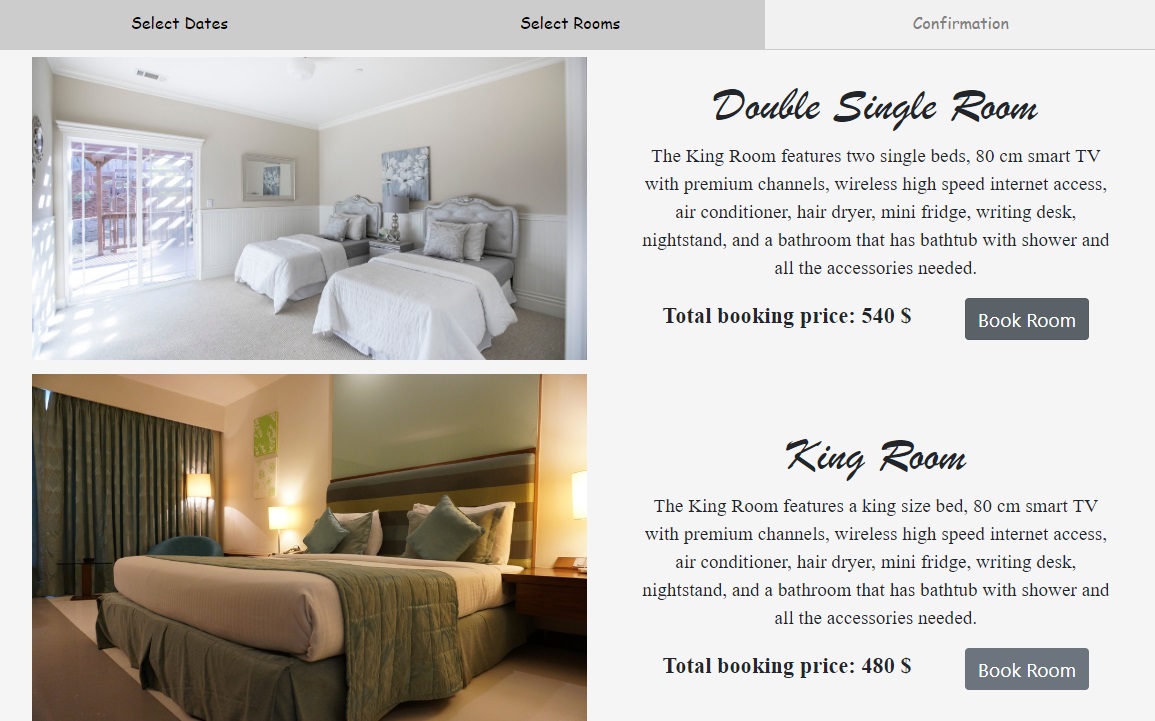
De aici utilizatorul poate alege să rezerve o categorie de cameră apăsând butonul Book Now sau sa vadă mai multe informații despre categoria de cameră dorită.



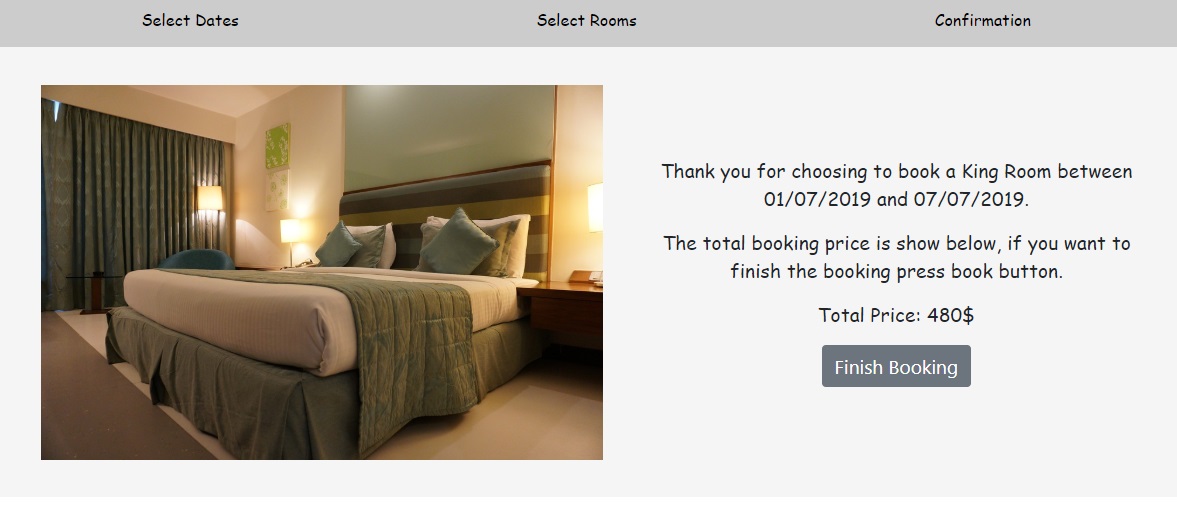
Dacă utilizatorul alege să rezerve o cameră acesta va fi dus la pagina din figura x unde va avea de ales data rezervării, adică ziua sosirii și ziua plecării.



Odata ce apasă butonul Search acesta va vedea categoriile de camere disponibile pentru data rezervării aleasă și prețul complet pentru rezervarea dorită. Pretul este calculat variabil în funcție de un algoritm.



După ce vede categoriile de camere disponibile și pretul acestora utilizatorul va putea alege o camera pentru a o rezerva în funcție de prețul și confortul dorit. La alegerea camerei dorite să fie rezervate acesta va fi dus la o pagină unde i se afisează informațiile pentru rezervarea dorită.



În final, dacă îi convin datele rezervării utilizatorul va apăsa butonul Finish Booking și va fi redirecționat către o pagină pentru realizarea plătii printr-un sistem PayPal în cazul în care utilizatorul este un client. Pentru cazul în care un client vrea să facă o rezervare când ajunge la hotel și nu are un cont personal în aplicație sau nu vrea să plătească prin PayPal, clientul poate lua legătura cu un angajat iar acesta poate face rezervarea direct fără efectuarea plății prin PayPal și poate șă încaseze banii lichizi direct de la client sau prin alt mod dorit de client.

La finalizarea comenzii se va trimite un email pentru a înștiința angajații hotelului că a fost facută o rezervare. În acest email se vor afla și informațiile necesare pentru crearea actelor închirierii camerei, acestea fiind date de utlizator la înregistrarea în aplicație.

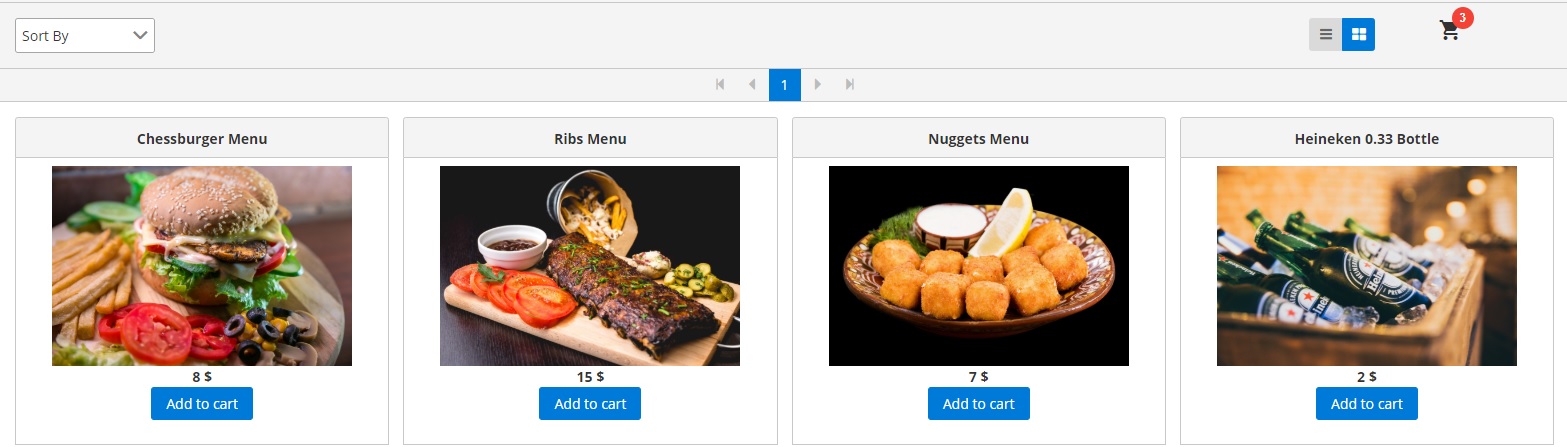
De asemenea, fiecare utilizator îşi poate vedea o listă cu toate rezervările sortate descrescător în funcţie de data cazării. Utilizatorii de tip angajat vor putea vedea o listă cu toate rezervările făcute.

Apoi, în ziua cazării, tot ce va trebui să facă clientul pentru a se caza în camera sau camerele în care are rezervare este să descarce codul de bare de pe pagina de profil și să scaneze acest cod la intrarea în cameră pentru a descuia camera. Codul este valabil pe toată durata rezervării, adică din prima zi a rezervării de la 2PM până în ultima zi a rezervării la 12PM.

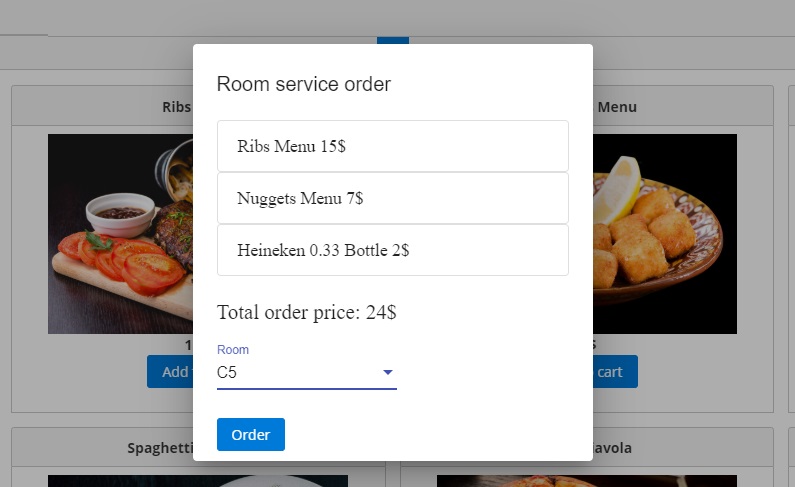
Pentru cazul în care clientul nu are acces la un telefon sau alt dispozitiv capabil să facă acest lucru, clientul poate merge și la recepție sa ceară un alt tip de cheie pentru a intra în camera.

**Comanda room-service**

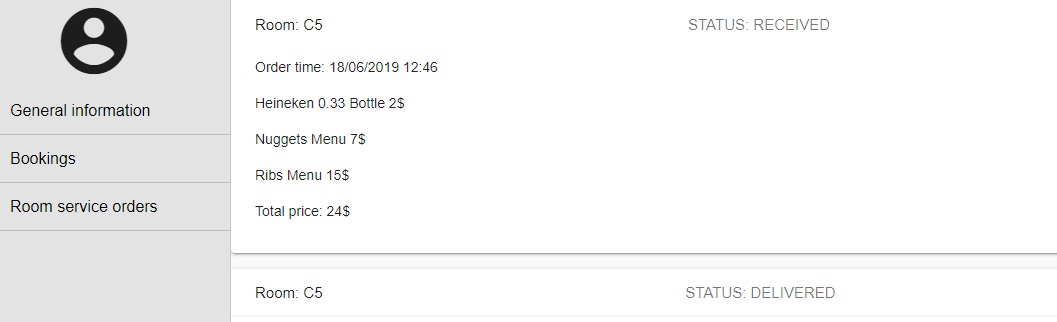
Pentru a face o comandă room-service un utilizator trebuie să acceseze pagina cu lista tuturor alimentelor posibile pentru comandă, această pagină se poate accesa de pe pagina de acasă a aplicației unde se găsește o descriere a hotelului cat și a utilitătilor oferite de acesta sau din bara de meniu a aplicației la Room-service.



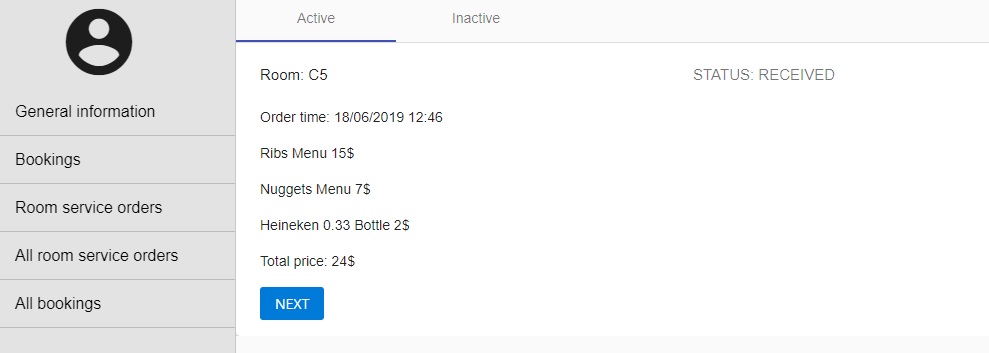
După ce ajunge aici utilizatorul poate să sorteze alimentele în funcție de preț pentru a găsi alimentele dorite de el. Apasând butonul Add to cart adaugă un aliment în coș. Pentru a finaliza comanda utilizatorul apasă pe coșul din dreapta sus, urmând să îi apară o listă cu alimentele alese și cu prețul final.



Apoi dacă îi convine prețul și alimentele alese acesta trebuie să aleagă o cameră din camerele rezervate de el a caror rezervare se desfășoară în momentul actual (o rezervare activă fiind o condiție necesară pentru plasarea unei comenzi de room-service). De asemenea, dacă un client al hotelului dorește sa facă o comandă de room-service dar nu are cont personal va putea face comanda telefonic, urmând ca acești pași să fie făcuți de un angajat al hotelului. Dupa plasarea comenzii utilizatorul va fi dus la o pagină unde ii vor apărea tot istoricul comenzilor cu detalii despre acestea, ultima fiind poziționată cel mai sus.



Un detaliu important este că în dreptul comenzii apare statusul acesteia pentru a informa utilizatorul în ce stadiu este comanda lui. La crearea comenzii acesta are statusul RECEIVED ceea ce înseamnă înregistrarea comenzii de către aplicație.



Odată ce comanda a fost trimisă utilizatorii de tip angajat pot vedea comanda în lista cu toate comenzile active în momentul de față și ei vor trece comanda în următorul pas apăsând butonul Next din figura de mai sus, astfel actualizându-se statusul acesteia.

Următorii pași ai comenzii sunt:

* PREPARING: comanda a fost preluată de angajaţi şi au început preapararea acesteia;
* PREPARED: comanda a fost preparată de bucătari şi acum se aşteaptă preluarea comenzii de către personalul hotelului care aduce comanda în cameră;
* DELIVERING: comanda a fost preluată și este în process de livrare la camera clientului;
* DELIVERED: comanda a fost livrată în cameră clientului.

**Algoritm preț variabil**

Pentru afișarea prețului la rezervarea camerei se folosește un algoritm care să facă prețul variabil în funcție de mai multe criterii.

Criteriile după care este calculat prețul sunt:

* în funcție de procentajul de camere ocupate din categoria de camere prețul poate crește în funcție de câte camere sunt libere pe durata rezervării sau poate să scadă dacă nu procentajul de rezervare al categoriei de camere respective este foarte mic;
* prețul poate crește în funcție de anumite zile ale anului în care este știut că se fac multe rezervări;
* dacă rezervarea este pe un număr de zile mare prețul poate scădea în funcție de numărul de zile pe care se face rezervarea;
* în cazul în care în ultimele 24 de ore a fost făcut un număr mare de rezervări prețul camerelor va crește;
* prețul poate scădea dacă clientul a mai avut rezervări precendente la hotel în funcție de numărul de rezervări care le-a avut și în funcție de prețul total al rezervărilor.

Algortimul este implementat într-un serviciu separate definit în interfața PriceService, această interfață are o metodă care returnează un Map (o structură de tip dicționar) în care cheile sunt nume de categorii de cameră și valoarea este prețul pentru fiecare categorie de camera, generat în funcție de categoriile pentru care se caută prețul, data sosirii, data plecării și emailul utilizatorului.

public interface PriceService {

Map<String, Double> getCategoriesPrices(Set<Category> categories, LocalDate arrivalDate, LocalDate departureDate, String email);

}

Această interfață este implementată de o clasă care suprascrie metoda din interfață și are ca membri o structură de tip map în care se vor pune categoriile cu prețul lor și serviciile pentru cameră și rervări care vor fi folosite pentru calcularea prețului categoriilor. Aceste două servicii sunt injectate în serviciul current prin procedeul de injectare de dependințe prin adnotarea @Autowired.

@Service

public class PriceServiceImpl implements PriceService {

private Map<String, Double> categoriesPrices = new HashMap<>();

private RoomService roomService;

private BookingService bookingService;

@Autowired

public PriceServiceImpl(RoomService roomService, BookingService bookingService) {

this.roomService = roomService;

this.bookingService = bookingService;

}

@Override

public Map<String, Double> getCategoriesPrices(Set<Category> categories, LocalDate arrivalDate, LocalDate departureDate, String email) {

calculateCategoriesPrices(categories, arrivalDate, departureDate, email);

return categoriesPrices;

}

Metoda suprascrisă din interfață apelează altă metodă (calculateCategoriesPrices) a cărui scop este să pună în map la chei numele categoriilor de camere primate ca parametru de aceasta și pentru valoarea fiecărei chei apelează o metodă (calculateTotalBookingPrice) a cărui scop este să calculeze și să returneze prețul rezervării pentru o categorie de cameră care o primește la parametru.

Mai departe, metoda calculateTotalBookingPrice parcurge toate zilele rezervării și adaugă într-o variabilă prețul calculate pentru fiecare zi a rezervării de metoda getBookingPriceForDayByCategory. Apoi prețului curent i se vor aplică reduceri si majorări dupa urmatoarele metodele:

* getPriceRemainingPrecentageByNumberOfBookingDays: calculeză un procetaj al prețului pe baza numărului de zile pentru care a fost făcută rezervare, dacă numărul de zile a rezervării este mare oferind o reducere a prețului;
* getPriceIncreasePercentageByNumberOfBookingInLastDay: calculează un procentaj al prețului pe baza numărului de rezervări făcute în ultimele 24 de ore, dacă numărul de rezervări este mare majorând prețul;
* getPriceDiscountPercentageByCustomerPreviousBookings: calculează un procentaj al prețului pe baza rezervărilor percedente a clientului care face cererea, oferind reduceri în funcție de numărul de rezervări precedente si de prețul total al rezervărilor (această reducere se aplică doar pentru clienții care fac rezervarea prin contul lor personal).

Mai departe voi descrie mai exact cum se calculează prețul. La apelarea metodei pentru obținere a prețului pentru o zi, acestă pune intr-o variabila prețul de bază al categoriei, după care adaugă prețului o valoare în funcție de numărul de camere ocupate apelând funcția getExtraPriceByRoomsOccupancy și înca o valoare dacă ziua respectivă se află pe lista de zile alese pentru majorare a prețului apelând funcțtia getExtraPriceBySpecialDates.

private Double getBookingPriceForDayByCategory(Category category, LocalDate date) {

Double totalDayPrice = category.getPrice();

totalDayPrice += getExtraPriceByRoomsOccupancy(category, date);

totalDayPrice += getExtraPriceBySpecialDates(category, date);

return totalDayPrice; }

}

private void calculateCategoriesPrices(Set<Category> categories, LocalDate arrivalDate, LocalDate departureDate, String email) {

categories.forEach(category ->

categoriesPrices.put(category.getName(),

calculateCategoryTotalBookingPrice(category,

arrivalDate, departureDate, email))

);

}

private Double calculateCategoryTotalBookingPrice(Category category,

LocalDate arrivalDate, LocalDate departureDate, String email) {

Double totalPrice = 0.0;

for (LocalDate date = arrivalDate; date.isBefore(departureDate);

date = date.plusDays(1)) {

totalPrice += getBookingPriceForDayByCategory(category, date);

}

totalPrice\*=getPriceRemainingPercentageByNumberOfBookingDays(arrivalDate, departureDate);

totalPrice \*= getPriceIncreasePercentageByNumberOfBookingInLastDay();

totalPrice\*=getPriceDiscountPercentageByCustomerPreviousBookings(email);

return Precision.round(totalPrice, 2);

}

Pentru calcularea adaosului de preț pe dupa procentajul de ocupare a categoriei, se va calcula procentajul de ocupare a categoriei de camere pentru ziua respective prin împărțirea numărului de camere ocupate la numărul totatl de camere din acea categorie. Pentru aflarea numărului de camere disponibile în acea zi se folosește metoda getAvailableRoomsBetweenDates din serviciul roomService. Apoi pentru calcularea adosului de preț se folosește o calsă de tip enum în care se vor găsi valorile la care dacă un procentaj de ocupare este mai mare sau egal decât valoare percent din enum atunci valoarea cu care se va multiplica pretul este valoare din campul multiplyValue. Astfel se parcurg toate valorile din enumul CategoryOcupancy iar în cazul în care procentajul de ocupare al camerelor actual este mai mare decât procentajul din enum vom returna valoarea pretului categoriei de camere înmulțită cu valoarea de multiplicare. De asemenea, avem și un caz special, adică daca mai este doar o zi până la rezervarea pentru ziua dorită si procentajul de ocupare este foarte mic vom returna o valoare negativă să se adauge la preț.

Exemplu: dacă procentajul de ocupare al unei categorii camere este mai sau egal ca 0.8 atunci se va returna 0.2 înmulțit cu prețul de bază al camerei pentru a fi adăugat la prețul camerei pentru ziua primită ca parametru de funcție.

private Double getExtraPriceByRoomsOccupancy(Category category,LocalDate date){

Integer totalRooms = category.getRooms().size();

Integer availableRooms = roomService

.getNumberOfAvailableRoomsBetweenDates(category.getRooms(), date,

date.plusDays(1));

Double occupancyPercentage =

Double.valueOf(totalRooms - availableRooms) / totalRooms;

for (CategoryOccupancy categoryOccupancy : CategoryOccupancy.values()) {

if (occupancyPercentage >= categoryOccupancy.getPercent()) {

return category.getPrice() \*

categoryOccupancy.getMultiplyValue();

}

}

if (occupancyPercentage <= 0.1 &&

DAYS.between(LocalDate.now(),date)<=1) {

return category.getPrice() \* (-0.1);

}

return 0.0;

}

public enum CategoryOccupancy {

NINETY\_PERCENT("NinetyPercent", 0.9, 0.25),

EIGHTY\_PERCENT("EightyPercent", 0.8, 0.2),

SEVENTY\_PERCENT("SeventyPercent", 0.7, 0.15),

SIXTY\_PERCENT("SixtyPercent", 0.6, 0.1),

FIFTY\_PERCENT("FiftyPercent", 0.5, 0.05);

private String label;

private Double percent;

private Double multiplyValue;

}

La calcularea adaosului de preț dupa zile special se va folosi în mod similar un enum în care se vor găsi ziua, luna și procentajul de multiplicare.

Exemplu: dacă pe în enum se gasește ziua 31, luna 12 si procentajul de multiplicare 0.3 și ziua pentru care se face rezervarea este 31/12/2019 atunci valoarea returnată de această funcție este valoarea prețului de bază înmulțită cu 0.3.

private Double getExtraPriceBySpecialDates(Category category, LocalDate date) {

for (DateRates dateRates : DateRates.values()) {

if (date.getMonthValue() == dateRates.getMonth()

&& date.getDayOfMonth() == dateRates.getDay()) {

return category.getPrice() \* dateRates.getMultiplyValue();

}

}

return 0.0;

}

Pentru aflarea noului procentaj al prețului în funcție de numărul de zile al rezervării, este folosit un nou enum a cărui câmpuri sunt un număr de zile și un procentaj de reducere.

Exemplu: dacă in enum se găsește valoarea 30 pentru numărul de zile și procentajul de reducere 0.1, în cazul în care metoda este apelată pentru o rezervare de 35 de zile metoda returnează valoarea 0.9 pentru a o înmulți cu prețul pentru rezervare calculate initial, astfel oferind o reducere de 10% la acel preț.

private Double getPriceRemainingPercentageByNumberOfBookingDays(LocalDate arrivalDate,

LocalDate departureDate) {

long bookingDays = DAYS.between(arrivalDate, departureDate);

for (NumberOfDaysDiscounts numberOfDaysDiscounts : values()) {

if (bookingDays >= numberOfDaysDiscounts.getNumberOfDays()) {

return 1.0 - numberOfDaysDiscounts.getDiscountPercentage();

}

}

return 1.0;

}

La apelul metodei getPriceIncreaseByNumberOfBookingInLastDay se va face un apel la metoda getNumberOfBookingsIn24HoursIntervalBeforeNow pentru a afla numărul de rezervări făcute în ultimele 24 de ore, iar dacă numărul de rezervări este îndeajuns de mare se va crește procentajul prețului.

private Double getPriceIncreasePercentageByNumberOfBookingInLastDay() {

Integer numberOfBookings = bookingService.

getNumberOfBookingsIn24HoursIntervalBeforeNow();

if (numberOfBookings >= 100) {

return 1.1;

}

return 1.0;

}

În final, la apelarea metodei pentru calcularea noului pocentaj în funcție de rezervarile precedente ale utilizatorului, se va face un apel la o metodă din serviciul bookingService pentru a primi o listă cu rezervările precedente ale utilizatorului în funcție de email. Mai departe, în cazul în care lista nu este goală și utilizatorul este unul cu rol ROL\_USER (adică client normal) se va returna noul procentaj al prețului în funcție de numărul total de rezervări și de prețul total al rezervărilor precedente.

private Double getPriceDiscountPercentageByCustomerPreviousBookings(String email) {

List<Booking> bookings = bookingService.getBookingsByCustomerEmail(email);

Integer numberOfBookings = bookings.size();

if (numberOfBookings != 0) {

if (bookings.get(0).getCustomer().getRole().equals(Role.ROLE\_USER)) {

Double totalBookingsPrice = bookings.stream()

.mapToDouble(Booking::getPrice).sum();

if (totalBookingsPrice >= 50000.0) {

return 0.7;

}

if (numberOfBookings >= 10) {

return 0.8;

}

if (numberOfBookings >= 5) {

return 0.9;

}

if (numberOfBookings >= 2) {

return 0.95;

}

}

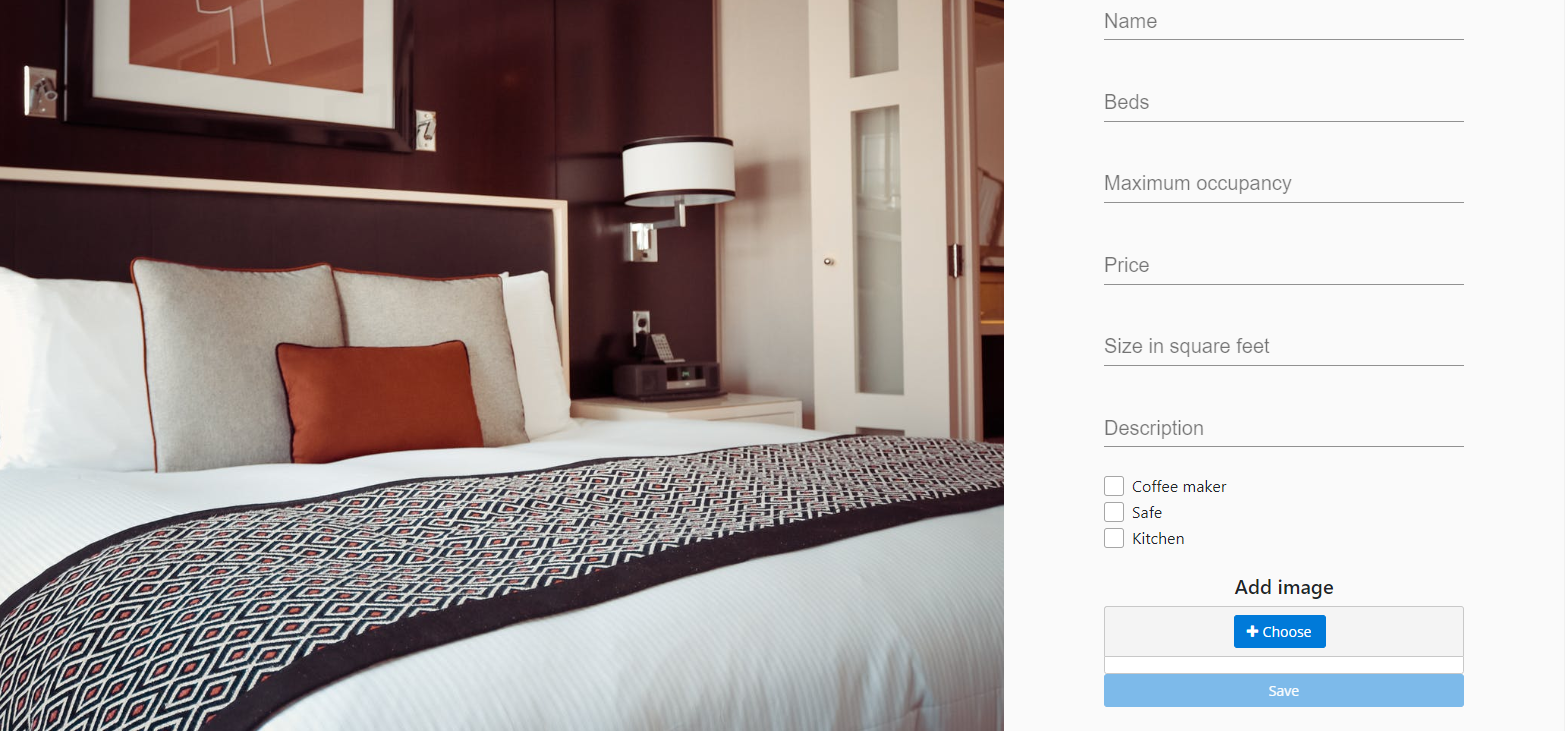
}

return 1.0;

}

**Funcţii admin**

Utilizatorul de tip este cel care are privilegiile să creeze noi categorii de camere, noi camere pe baza categoriilor deja existente și noi alimente pentru comenzile de room-service. De asemenea, utilizatorul de tip admin este cel care are drepturi să creeze utilizatori de tip angajat sau admin. Initial este introdus în aplicație un utilizator de tip admin pentru a putea crea noi utilizatori de tip admin si angajat.



**Trimitere emailuri**

La trimiterea de emailuri se folosește Gmail SMTP. Pentru a se putea folosi acest serviciu trebuie adăugată dependința spring-boot-starter-mail și trebuie făcute configurările pentru serverul care trimite emailuri (configurările se fac în fișierul application.properties).

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=hotel.mng2019@gmail.com

spring.mail.password=password

spring.mail.properties.mail.smtp.auth=true

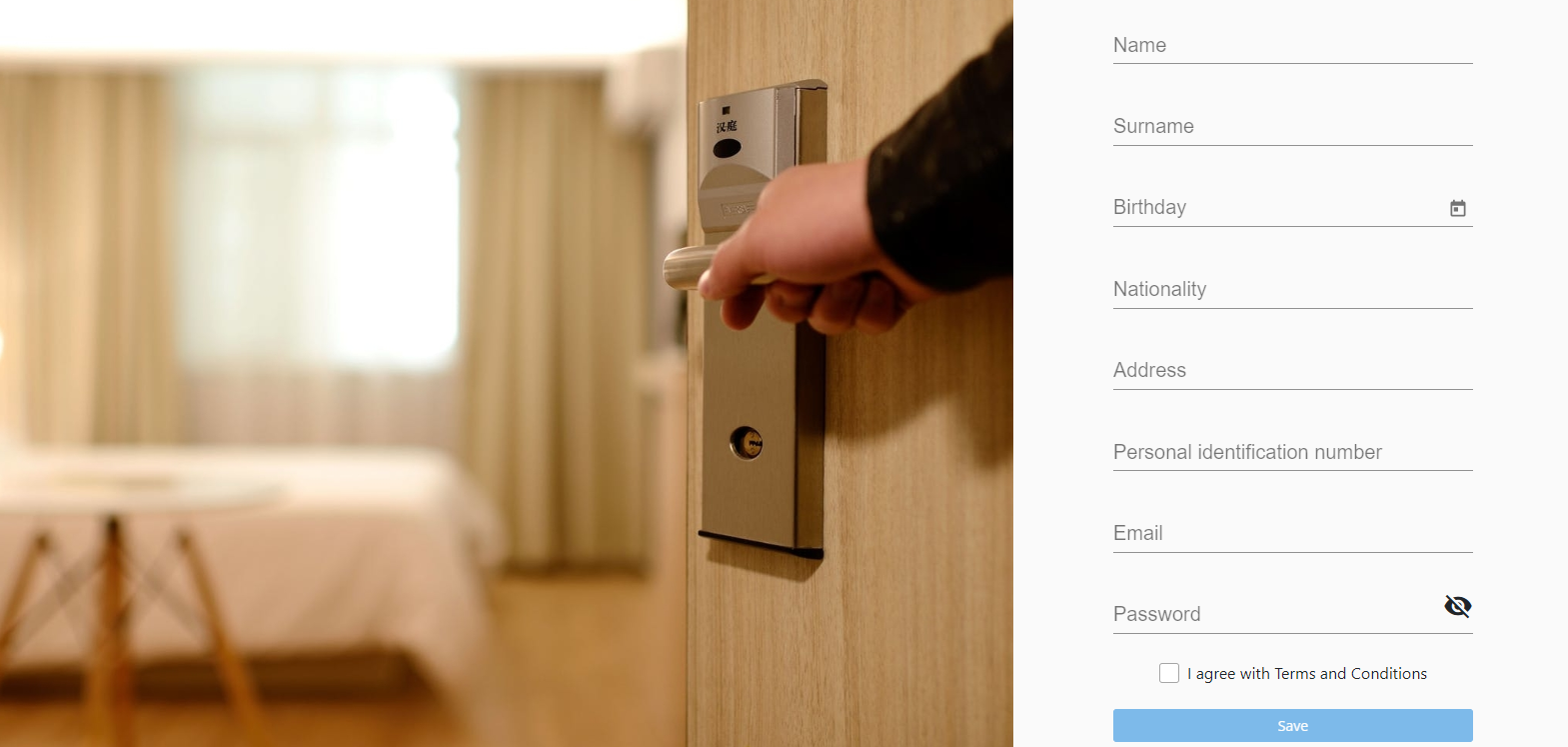
spring.mail.properties.mail.smtp.starttls.enable=true

Odată ce avem aceste configurări făcute pentru a trimite un email trebuie creat mesajul, în care punem subiectul, destinatarul și mesajul propriu. De asemenea, la trimiterea unui email vom crea un nou fir de execuție care se va ocupa cu trimiterea acestuia deoarece trimiterea unui email poate dura o perioadă de timp relative lungă (mai mult de 2-3 secunde) și nu vrem ca aplicația noastră să fie blocată în acest timp.

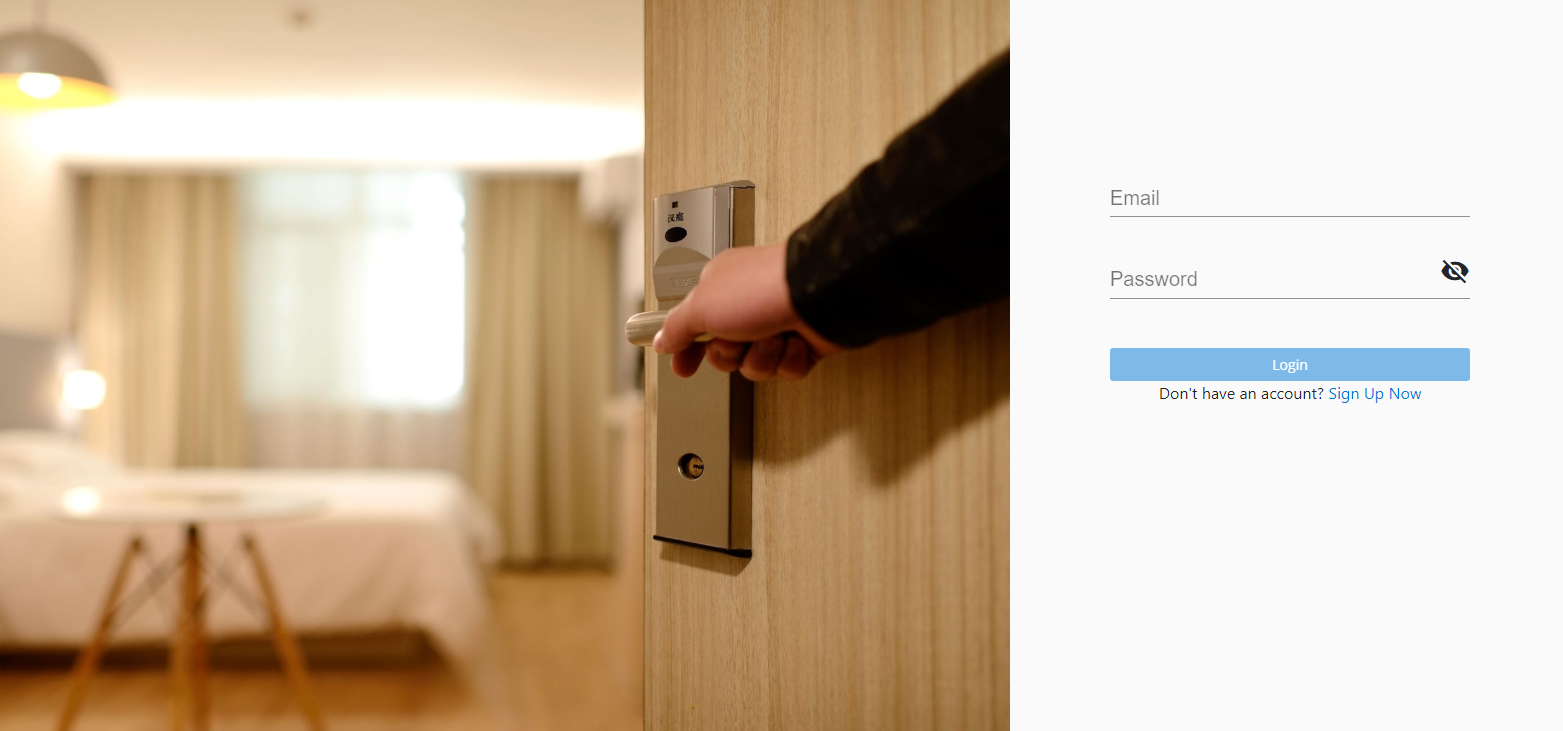
**Plată PayPal**

**Securitate**

Inițial o persoană care accesează această applicație poate vizualiza informații generale despre hotel (adică serviciile oferite de hotel) și poate vedea informații despre toate tipurile de camere ale hotelului. Pentru a face o rezervare la o cameră clientul va trebui să își creeze un cont personal în aplicție în caz că nu are deja. La crearea contului clientului i se vor cere diverse informații personale necesare pentru crearea actelor necesare de hotel la cazarea clientului. Peste parola utilizatorului se va folosi o funcție hash înainte de a se introduce în baza de date, iar peste celelalte informații sensibile despre utilizator se va utiliza un algoritm de codificare înainte de a fi introduse în baza de date. De asemenea, utilizatorul va trebui să confirme că este de acord cu termenii și condițiile hotelului. La finalul înregistrării utilizatorul va primi un email de confirmare la adresa de email introdusă.



Dupa finalizarea înregistrării utilizatorul va fi redirecționat catre pagina de autentificare, unde acesta trebuie să își introduca emailul folosit pentru crearea contului și parola.



Pentru autentificare și securizare se folosește Spring Security, adus prin introducerea dependinței spring-boot-starter-security în fișierul “pom.xml”. Apoi pentru activarea securității asupra aplicației a fost creată o clasă care să extinda WebSecurityConfigurerAdapter.

@EnableGlobalMethodSecurity(prePostEnabled = true)

@EnableWebSecurity

@EnableJpaRepositories(basePackageClasses = CustomerRepository.class)

@Configuration

public class SecurityConfiguration extends WebSecurityConfigurerAdapter

În această clasă este creat un bean AuthenticationManager care acum poate fi injectat în alte servicii. Mai departe acest bean va fi injectat în serviciul care se ocupă de autentificarea utilizatorilor și pe baza acestuia determină daca utilizatorul primește sau nu acces la aplicație.

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService)

.passwordEncoder(passwordEncoderBean());

}

@Bean

public PasswordEncoder passwordEncoderBean() {

return new BCryptPasswordEncoder();

}

@Bean

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

Beanul password encoder este cel care folosit pentru a aplica funcția hash peste parolă înainte de a fi introdusă în baza de date. De asemenea, dupa cum se vede tot acest bean este folosit și la configurarea serviciului de autentificare.

În cazul în utlizatorul este autentificat cu success se va genera un token care va fi trimis aplicației de front-end și îl va pastra în memoria acestuia locală (numită LocalStorage). Motivul păstrării tokenului în memorie este dat de faptul că odată ce este activat Spring Security asupra aplicției toate requesturile web vor fi securizate și vor avea nevoie de un token valid pentru a fi accesate. Totuși, sunt nevoie și de cereri web care să se poată face și în cazul în care utilizatorul nu este logat, cum ar fi chiar cererea care este făcuta pentru logare, așdar tot în clasa de configurare trebuie să existe o metodă configure care în care se pot zice ce adresele cererilor web care nu necesită nici un fel de autentificare.

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.cors().and()

.csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()

.authorizeRequests()

.antMatchers("/authentication/\*\*").permitAll()

.antMatchers(HttpMethod.POST,"/customers").permitAll()

.antMatchers(HttpMethod.GET,"/categories").permitAll()

.antMatchers(HttpMethod.GET,"/categories/{id}").permitAll()

.antMatchers(HttpMethod.PATCH,"/room-services/\*\*").permitAll()

.anyRequest().authenticated();

http

.addFilterBefore(authenticationTokenFilter, UsernamePasswordAuthenticationFilter.class);

}

În plus, trebuie securizat accesul la anumite requesturi web în funcție de rolul utilizatorului. Spre exemplu, doar un utilizator cu rolul admin ar trebui să poată crea o cameră. Asta se face prin punerea adnotării @PreAuthorize(“hasRole(‘role\_name’)”) deasupra metodei care reprezinta cererea web pentru care se dorește securizarea în funcție de rol.

De asemenea, metode de securitate sunt folosite și pe partea de front-end a aplicației. Asta se va face prin folosirea unei clase de tip gardă, care la accesarea unei component va verifica dacă rolul utilizatorului este cel dorit.

**Testare unitară**

**Bibliografie**

<https://www.baeldung.com/spring-email>

<https://angular.io/tutorial>

<https://coursetro.com/posts/code/171/Angular-7-Tutorial---Learn-Angular-7-by-Example>

<https://material.angular.io/>

<https://www.jetbrains.com/help/idea/running-and-debugging-typescript.html>

<https://getbootstrap.com/>

<https://www.w3schools.com/bootstrap4/default.asp>

<https://www.primefaces.org/primeng/#/>

<https://angular.io/guide/styleguide>

<https://itnext.io/choosing-a-highly-scalable-folder-structure-in-angular-d987de65ec7>

<https://github.com/mathisGarberg/angular-folder-structure>

<https://malcoded.com/posts/angular-fundamentals-modules/>

<https://www.technouz.com/4772/angular-6-app-structure-with-multiple-modules/>

<https://code-maze.com/angular-best-practices/>

<https://www.freecodecamp.org/news/how-to-center-things-with-style-in-css-dc87b7542689/>

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

<https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Ordering_Flex_Items>

<https://flexboxfroggy.com/>

<https://www.baeldung.com/security-spring>

<https://www.baeldung.com/securing-a-restful-web-service-with-spring-security>

<https://www.baeldung.com/spring-security-oauth-jwt>

<https://chariotsolutions.com/blog/post/angular-2-spring-boot-jwt-cors_part2/>

<https://mherman.org/blog/token-based-authentication-with-angular/#localstorage>

<https://codecraft.tv/courses/angular/dependency-injection-and-providers/tokens/>

<https://medium.com/engineerbabu/angular-authentication-using-jwt-d846c5ce0ac6>

<https://medium.com/@ryanchenkie_40935/angular-authentication-using-route-guards-bf7a4ca13ae3>

<https://nirajsonawane.github.io/2018/10/27/Angular-Material-Tabs-with-Router/>

<https://medium.com/@nikhildevre/creating-tabs-using-angular-material-2-and-angular-4-routing-3634c3d0f7cc>

<https://blog.angularindepth.com/angular-router-series-secondary-outlets-primer-139206595e2>

<https://www.w3schools.com/howto/howto_js_tabs.asp>

<https://css-tricks.com/left-and-right/>

<https://css-tricks.com/couple-takes-sticky-footer/>

<https://github.com/paypal/PayPal-Java-SDK>

<https://developer.paypal.com/docs/api/quickstart/payments/#additional-information>

<https://developer.paypal.com/developer/accountStatus/>

<https://c4model.com/>

<https://structurizr.com/express?src=/static/express/bigbankplc/system-context.json>

<https://www.draw.io/>

<https://hatchful.shopify.com/>

<https://www.pexels.com/search/hotel/>

<https://pixabay.com/images/search/hotel/>

<https://unsplash.com/search/photos/room-service>