

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Hotel Manager

propusă de

Adrian Ștefan

Sesiunea: *iulie, 2019*

Coordonator științific

Prof. Colaborator Florin Olariu

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Hotel Manager

Adrian Ștefan

Sesiunea: *iulie, 2019*

Coordonator științific

Prof. Colaborator Florin Olariu

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele _____

Data _____

Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemnatul(a)

domiciliul în

născut(ă) la data de, identificat prin CNP

absolvent(a) al(a) Universității „Alexandru Ioan Cuza” din Iași, Facultatea de

specializarea, promoția, declar pe

propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul

Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 si 5 referitoare la

plagiat, că lucrarea de licență cu titlul:

_____elaborată sub îndrumarea dl. / d-na

_____, pe care urmează să o susțină în fața comisiei

este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data azi,

Semnătură student

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Hotel Manager*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, 24.06.2019

Absolvent *Adrian Ștefan*

Cuprins

Introducere	5
Contribuții.....	6
Capitolul 1. Descrierea problemei	7
1.1 Tehnologii folosite.....	7
1.2 Arhitectură proiect	11
1.3 Bune practici folosite.....	14
Capitolul 2: Abordări anterioare	16
Capitolul 3: Descrierea soluției	17
3.1 Înregistrare și autentificare	17
3.2 Securitate	18
3.3 Rezervare cameră	20
3.4 Comandă room-service.....	23
3.5 Activități admin	25
3.6 Algoritm preț variabil	26
3.7 Serviciu emailuri.....	32
3.8 Plată PayPal	34
Concluzii lucrării	37
Bibliografie	38

Introducere

Tema licenței este construirea unei aplicații web care are ca și scop managementul unui hotel. Toți utilizatorii 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 și comenzi room-service se face prin autentificarea cu un cont personal.

Aplicația oferă posibilitatea înregistrării cu trei tipuri de conturi, admin, angajat și utilizator. Ca și admin poți crea diferite categorii de camere. Odată creată o categorie poți adăuga una sau mai multe camere cu categoria respectivă, acestea devenind disponibile spre rezervare. De asemenea, utilizatorii de tip admin au și posibilitatea de a adăuga diverse tipuri de produse pentru a putea fi comandate prin serviciul de room-service. Rezervarea camerelor poate fi făcută de toți utilizatorii autentificați și plata se face prin PayPal pentru finalizarea rezervării. Comenzile room-service pot fi făcute de utilizatorii care sunt autentificați și au o rezervare de cameră în momentul efectuării comenzii. Conturile de tip angajat pot vedea toate rezervările și comenzile de room-service. În 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 a aplicației este cea de generare a unui cod QR la momentul înregistrării. Cu acest cod de bare clienții vor putea intra în camerele unde au rezervări pe parcursul rezervării. Fiecare cameră a hotelului va avea un dispozitiv capabil să scaneze codul QR, iar la scanarea codului corect se va descuia camera. Codul QR 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 timp pierdut prin așteptarea la cozi lungi la recepție. Alt element lipsă la aceste aplicații este posibilitatea plasării unei comenzi de room-service, și consider că 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 folosește și algoritmi pentru a face prețul rezervării să varieze în funcție de diverse criterii, printre 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 serviciilor hotelului doar prin intermediul aplicației, asta făcând atât cazarea cât și perioada de cazare la hotel mult mai simplă și mai plăcută pentru client

Contribuții

Contribuțiile aduse de această aplicație sunt:

- Posibilitatea de a te caza la hotel doar prin intermediul aplicației, după rezervare primind un QR cod cu care poți intra în cameră, nefiind nevoie să mergi la recepția hotelului pentru a îți face check-in-ul și pentru a ridica cheia;
- Posibilitatea de a da comenzi de room-service prin intermediul aplicației, statusul comenzilor fiind actualizat constant de către angajați hotelului, astfel clienții știind exact în ce status este comanda lor;
- Camerele și alimentele disponibile sunt configurabile prin intermediul aplicației de către un utilizator de tip admin;
- Folosirea unui algoritm pentru a genera prețul unei rezervări variabil după mai multe criterii.

Ca și contribuții personale, pentru dezvoltarea acestui proiect am învățat să folosesc două frameworkuri, unul pentru dezvoltarea interfeței grafice (Angular) și unul pentru dezvoltarea back-end-ului (Spring). Pe lângă asta, am învățat și cum să folosesc sistemul extern pentru realizarea unei plăți prin PayPal și să trimit emailuri cu sistemul Gmail.

Capitolul 1. Descrierea problemei

1.1 Tehnologii folosite

Proiectul este împărțit în două aplicații, o aplicație este un API care se ocupă de prelucrarea informațiilor de la utilizator, cât și stocarea și preluarea informațiilor în 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 afișându-le în interfața grafică, această parte este numită front-end.

Back-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 dependențe 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 strict pentru aducerea de dependențe externe este folosit tag-ul <dependencies> </dependencies>, în interiorul căruia se adaugă diverse dependențe preluate de pe internet (vezi figura 1).

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.1.3.RELEASE</version>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>2.1.3.RELEASE</version>
  </dependency>
</dependencies>
```

Figura 1: Importare dependența folosind maven

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 adăuga informații în baza de date. Pentru a folosi acest framework este folosit Java Persistence API, numit JPA, aceasta fiind specificația unei interfețe de programare a aplicațiilor care descrie managementul datelor relaționale în aplicații folosind Hibernate pentru lucrul cu baza de date. JPA folosește 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 (vezi figura 2), `@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;
```

Figura 2: Mapare clasă la tabelă în baza de date folosind JPA

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 deja implementate pentru preluarea și stocarea informațiilor în baza de date. În proiectul meu am folosit `JpaRepository` (exemplu în figura 3), 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 reprezentativă pentru 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();
}
```

Figura 3: Exemplu interfață `JpaRepository` pentru interacționare cu baza de date

Spring Boot este un mecanism folosit pentru crearea configurațiilor aplicației și pentru rularea aplicației (vezi exemplu în figura 4).

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class);
    }
}
```

Figura 4: Clasa necesară pornire aplicație spring boot

Mecanismul de inversiune de control este realizat prin injectare de dependențe. Pentru a putea injecta un serviciu trebuie declarată o interfață și pusă o adnotare de tip `@Service` peste o clasă care implementează acea interfață după care această clasă trebuie declarată în constructorul clasei și deasupra constructorului trebuie pusă adnotarea `@Autowired` (vezi figura 5).

```
@Autowired
public AlimentController(AlimentService alimentService, AlimentMapper alimentMapper) {
    this.alimentService = alimentService;
    this.alimentMapper = alimentMapper;
}
```

Figura 5: Exemplu injectare servicii

Mecanismul pentru cereri web este realizat prin diverse adnotări pentru a zice ca o metodă 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 lângă aceste adnotări mai sunt folosite și adnotările `@PathVariable`, `@RequestBody`, `@RequestParam` pentru folosirea de variabile în URL, informații transmise în corpul cererilor web sau ca parametri.

```
@GetMapping(value =("/{id}")
public BookingDto getById(@PathVariable Long id) { return bookingService.getBookingDtoById(id); }
```

Figura 6: Metoda care implementează getById folosind spring

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.

De asemenea, pe partea de back-end folosesc și Junit pentru testarea unitară a anumitor părți de cod.

Front-end

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. Fișierele HTML și CSS oferă interfața grafică și fișierul TypeScript face apeluri 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. De asemenea, modulele sunt încărcate prin tehnica de lazy loading, astfel oferind o performanță mai bună aplicației.

1.2 Arhitectură 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 navigatoarelor web în forma de HTML și CSS folosind frameworkul Angular.

Nivelul aplicație este aplicația de back-end. Aceasta folosește controlere pentru a prelua cererile web de la aplicația de front-end, apoi folosește servicii pentru a prelucra informațiile și a adăuga sau pentru a prelua anumite informații din baza de date. Sunt folosite două sisteme externe, unul pentru plata prin PayPal și unul pentru trimiterea de emailuri prin Gmail (vezi figura 7).

Nivelul de date este reprezentat de baza de date, care se ocupă de păstrarea și recuperarea informațiilor de către utilizator. Baza de date folosită în proiect este PostgreSQL.

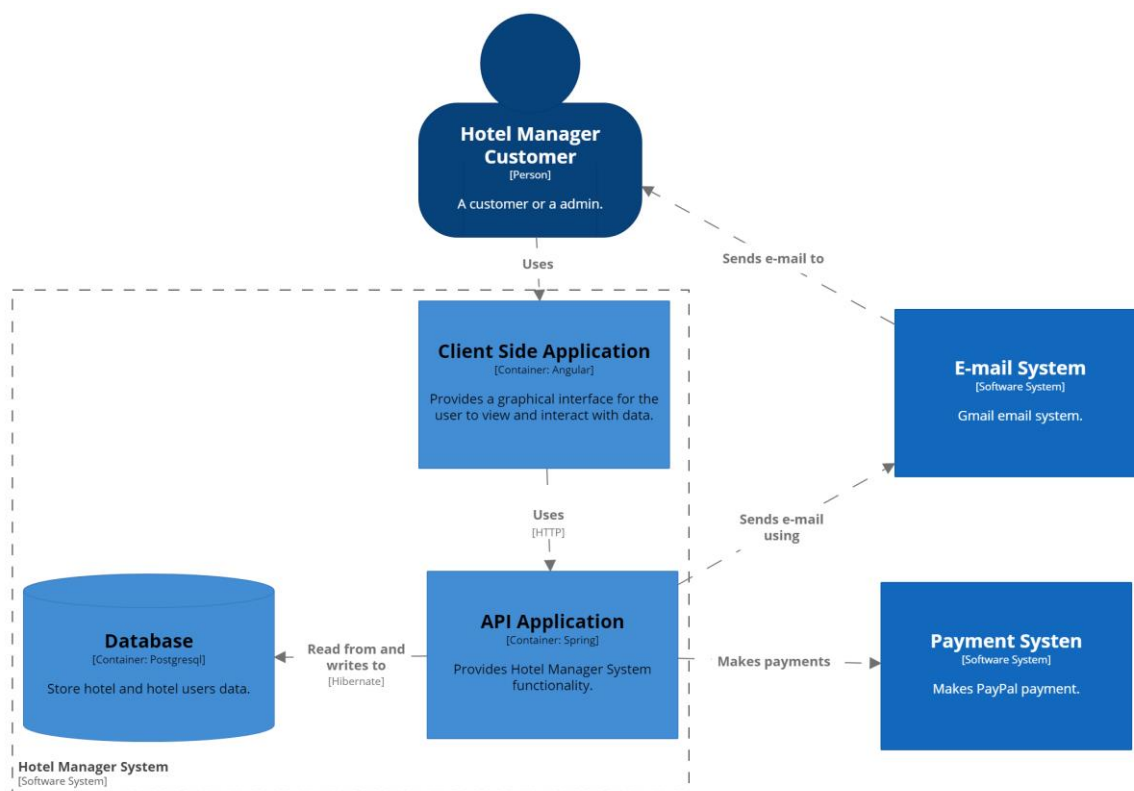


Figura 7: Diagramă de containere

Back-end

Aplicația de back-end este împărțită la rândul ei pe trei nivele, nivelul de preluare a cererilor web de la aplicația de front-end folosind controlere, mai departe folosind servicii pentru prelucrarea datelor, serviciile reprezentând al doilea nivel. Ultimul nivel constă în lucrul cu bazele de date, 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 de lucru cu baza de date se face prin injectare de dependențe.

Pentru lucrul cu baza de date se folosesc modele care reprezintă tabele din baza de date (la figura 8 se poate vedea diagrama de clase de modele a aplicației).

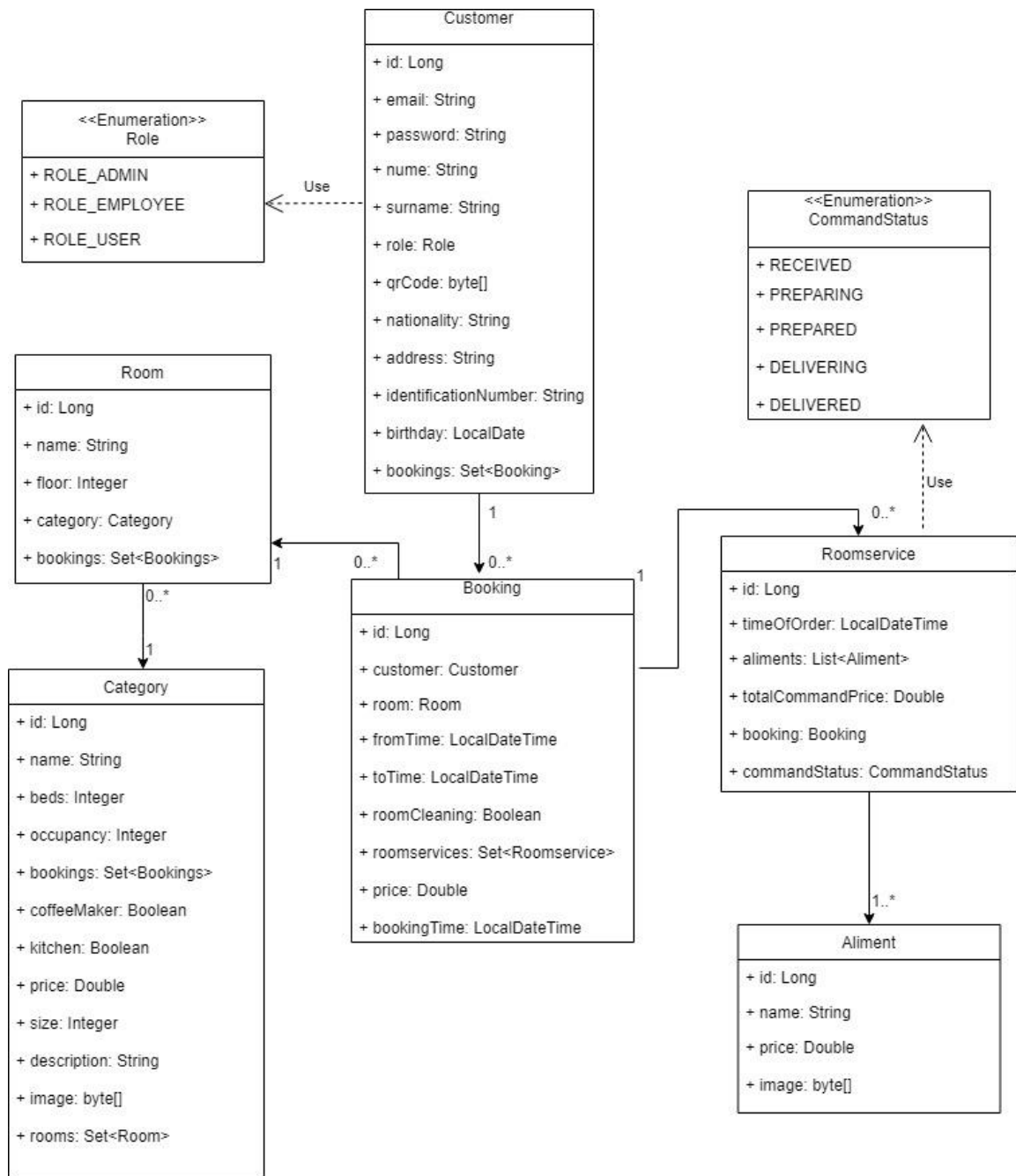


Figura 8: Diagrama claselor de modele

Front-end

Aplicația de front-end este împărțită în trei module, bază, module și comun. În modulul bază 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 module conține la rândul lui diferite module organizate care conțin componente necesare pentru a crea o pagină web. În final, ultimul modul este cel comun care conține informații comune pentru celelalte module cum ar fi componentele cu header și footer, pipe-uri, serviciu pentru validări și aducere de diverse module externe.

Securitate

Pentru partea de securitate este folosit Spring Security pentru a genera un cod (token) la autentificare în funcție de rolul utilizatorului (admin pentru utilizatori de tip admini, employee pentru utilizatori care sunt angajați 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 după informațiile din token.

1.3 Bune practici folosite

Folosirea obiecte tip DTO

Pentru preluarea de informații din cererile web și pentru returnare de informații sunt folosite alte clase asemănătoare cu modelele din baza de date, numite DTO (vezi figura 9). Motivul folosirii acestor clase este dat de faptul că modelele trebuie să aibă aceeași formă ca și tabela din baza de date iar uneori putem să avem nevoie să primim sau să oferim informații diferite față de cele din modele. 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 inițializarea acestuia cu valorile din obiectul dinspre care este transformat (vezi figura 10).

```
public class AlimentDto implements Serializable {  
  
    private Long id;  
  
    private String name;  
  
    private Double price;  
  
    private byte[] image;  
  
}
```

Figura 9: Exemplu clasă tip DTO

```
@Service  
public class AlimentMapper {  
  
    public Aliment map(AlientDto alimentDto) {  
        Aliment aliment = new Aliment();  
        aliment.setId(alimentDto.getId());  
        aliment.setName(alimentDto.getName());  
        aliment.setPrice(alimentDto.getPrice());  
        aliment.setImage(alimentDto.getImage());  
        return aliment;  
    }  
  
    public AlimentDto map(Alient aliment) {  
        AlimentDto alimentDto = new AlimentDto();  
        alimentDto.setId(aliment.getId());  
        alimentDto.setName(aliment.getName());  
        alimentDto.setPrice(aliment.getPrice());  
        alimentDto.setImage(aliment.getImage());  
        return alimentDto;  
    }  
  
}
```

Figura 10: Exemplu serviciu de transformare de la dto la model

Folosirea excepției proprii

Aplicația folosește excepții proprii care extind clasa `RuntimeException` și le este dat un cod de status HTTP. Spre exemplu, dacă se caută o cameră 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."); }
}
```

Figura 10: Excepție proprie cu un cod de status

Responsivitate aplicație

Aplicația este făcută responsivă să poată fi folosită pe orice tip de dispozitiv (spre exemplu calculator, tabletă, telefon).

Clean code

În scrierea codului sunt respectate tehnicile de clean code, adică:

- Numele funcțiilor și al variabilelor sunt descriptive pentru a fi ușor de înțeles care este rolul acestora;
- Codul este scris cât mai modular posibil pentru a fi ușor de înțeles și pentru eliminarea codului duplicat;
- În cod este păstrată consistența;
- Codul este organizat în pachete și servicii.

GDPR

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

Capitolul 2. Abordări anterioare

În practică există multe hoteluri care au aplicații făcute pentru a putea face rezervări la aceste hoteluri, dar mi se pare că cel mai mare minus la aceste aplicații este faptul că deși oferă posibilitatea de a îți rezerva o cameră prin intermediul aplicației, odată ajuns la hotel pentru a te caza în cameră este necesar să mergi la recepție pentru a face check-in-ul și pentru a lua cheia ceea ce consider că este ceva foarte greșit din punct de vedere logic și nu în ton cu evoluția tehnologiei în ziua de azi.

Un alt element care nu l-am găsit în nici o altă aplicație este posibilitatea de a face o comandă de room-service la care să poți vedea statusul comenzii în timp real.

Un plus pe care unele aplicații le au, este cel de a oferi posibilitatea de a face rezervări la anumite săli și la evenimente de tip SPA și masaj prin intermediul aplicației.

Mai jos se găsesc exemple de aplicații ale unor hoteluri existente:

- <https://www.hotelinternationaliasi.ro/>
- <https://www.portolahotel.com/>
- <https://www.thelucernehotel.com/>
- <https://www.lafondasantafe.com/>

Capitolul 3. Descrierea soluției

3.1 Înregistrare și autentificare

Inițial o persoană care accesează această aplicație poate vizualiza informații generale despre 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 aplicație, 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.

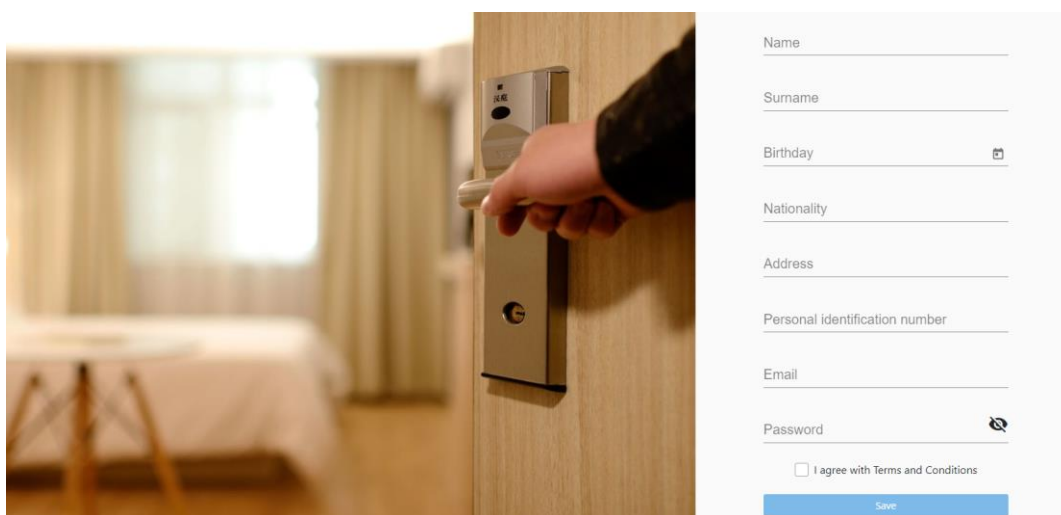


Figura 11: Pagină de înregistrare

După finalizarea înregistrării utilizatorul va fi redirecționat către pagina de autentificare, unde acesta trebuie să își introducă emailul și parola.

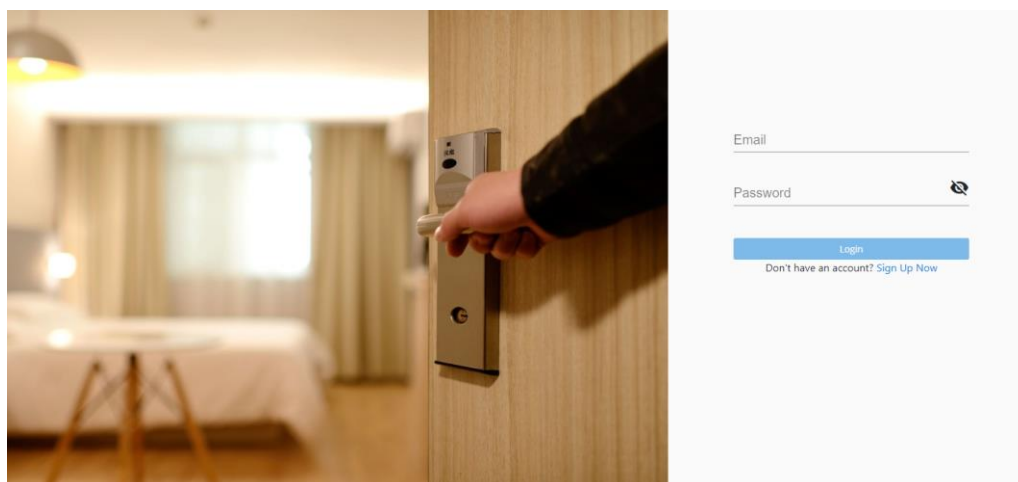


Figura 12: Pagină autentificare

3.2 Securitate

Pentru securitate se folosește Spring Security, prin introducerea dependenței `spring-boot-starter-security` în fișierul “`pom.xml`”. Apoi pentru activarea securității asupra aplicației a fost creată o clasă care să extindă `WebSecurityConfigurerAdapter` (vezi figura 13).

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
@EnableWebSecurity
@EnableJpaRepositories(basePackageClasses = CustomerRepository.class)
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter
```

Figura 13: Clasă configurări pentru securitate

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 dacă 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();
}
```

Figura 14: Configurare serviciu de autentificare

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

În cazul în care utilizatorul este autentificat cu succes se va genera un token care va fi trimis aplicației de front-end și îl va păstra în memoria acestuia locală (numită `LocalStorage`). Motivul păstrării token-ului în memorie este dat de faptul că odată ce este activat Spring Security asupra aplicației toate cererile 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 autentificat, cum ar fi chiar cererea care este făcută pentru autentificare, așadar tot în clasa de configurare trebuie să existe o metodă configurare în care se pot zice adresele cererilor web care nu necesită nici un fel de autentificare.

```

@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .cors().and() HttpSecurity
        .csrf().disable() HttpSecurity
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and() HttpSecurity
        .authorizeRequests() ExpressionInterceptUrlRegistry
        .antMatchers( ...antPatterns: "/authentication/**").permitAll() ExpressionUrlAuthorizationConfigurer<HttpSe
        .antMatchers(HttpMethod.POST, ...antPatterns: "/customers").permitAll() ExpressionUrlAuthorizationConfigu
        .antMatchers(HttpMethod.GET, ...antPatterns: "/categories").permitAll() ExpressionUrlAuthorizationConfigu
        .antMatchers(HttpMethod.GET, ...antPatterns: "/categories/{id}").permitAll() ExpressionUrlAuthorizationC
        .antMatchers(HttpMethod.PATCH, ...antPatterns: "/room-services/**").permitAll() ExpressionUrlAuthorizat
        .anyRequest().authenticated();

    http
        .addFilterBefore(authenticationTokenFilter, UsernamePasswordAuthenticationFilter.class);
}

```

Figura 15: Configurări adrese care nu necesită nici un fel de autorizare pentru a le accesa

În plus, trebuie securizat accesul la anumite cereri 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 reprezintă 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.

3.3 Rezervare cameră

Pentru a rezerva o cameră la hotel un utilizator întâi trebuie să acceseze pagina cu toate categoriile de camere (vezi figura 16) disponibile spre rezervări, această pagină se poate accesa de pe pagina de acasă a aplicației unde se găsește o descriere a hotelului cât și a serviciilor oferite de acesta sau din bara de meniu a aplicației la Accomodations.



Junior Suite

The Junior Suite features a king size bed, couch, 100 cm smart TV with premium channels, wireless high speed internet access, air conditioner, hair dryer, mini fridge, coffee maker, safe, writing desk, dining table, nightstand, and a bathroom that has bathtub with shower and all the accessories needed.

[See More](#)[Book Now](#)

Superior Suite

The Superior Suite contains a bedroom, a living room with kitchen and a bathroom. The suite features a king size bed, extendible couch, two 100 cm smart TV with premium channels, wireless high speed internet access, air conditioner, hair dryer, fridge, coffee maker, safe, writing desk, dining table, nightstand, and a bathroom that has bathtub with shower and all the accessories needed.

[See More](#)[Book Now](#)

Figura 16: Pagină listare categorii de camere

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.

Features

Maximum occupancy: 4
Beds: 1
Size: 350 square feet
Smart TV
Wireless internet access
Air conditioner
Coffee Maker
Kitchen
In-room safe

Basic price **200 \$**

[Book Now](#)

Figura 17: Informații categorie cameră

Dacă utilizatorul alege să rezerve o cameră acesta va fi dus la pagina din figura 18 unde va avea de ales data rezervării, adică ziua sosirii și ziua plecării.

The screenshot shows a web interface for selecting booking dates. It has three tabs: 'Select Dates', 'Select Rooms', and 'Confirmation'. The 'Select Dates' tab is active, displaying two calendars for July 2019. The left calendar, labeled 'Arrival', shows the 1st of July selected. The right calendar, labeled 'Departure', shows the 7th of July selected. A blue 'Search' button is positioned at the bottom center of the date selection area.

Figura 18: Pagină selectare perioadă rezervare cameră

Odată ce apasă butonul Search acesta va vedea categoriile de camere disponibile (vezi figura 19) pentru data rezervării aleasă și prețul complet pentru rezervarea dorită. Prețul este calculat variabil în funcție de un algoritm.

The screenshot displays the 'Select Rooms' section of the booking interface. It features two room categories, each with a photograph, a detailed description of amenities, the total booking price, and a 'Book Room' button.

Room Category	Description	Total booking price	Action
Double Single Room	The King Room features two single beds, 80 cm smart TV with premium channels, wireless high speed internet access, air conditioner, hair dryer, mini fridge, writing desk, nightstand, and a bathroom that has bathtub with shower and all the accessories needed.	540 S	Book Room
King Room	The King Room features a king size bed, 80 cm smart TV with premium channels, wireless high speed internet access, air conditioner, hair dryer, mini fridge, writing desk, nightstand, and a bathroom that has bathtub with shower and all the accessories needed.	480 S	Book Room

Figura 19: Pagină listare categorii disponibile în perioada de rezervare aleasă și prețul rezervării

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

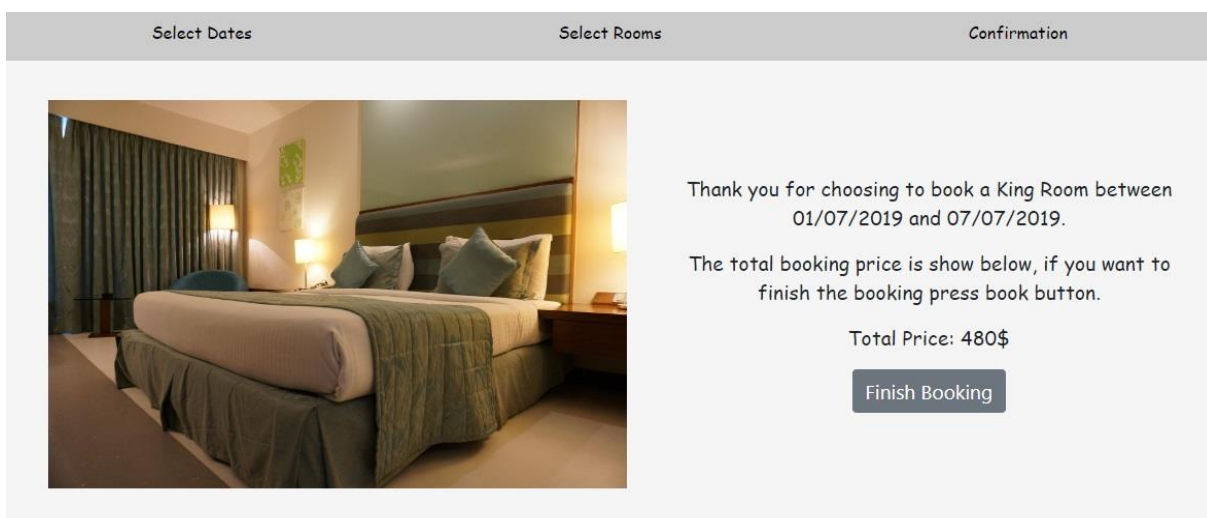


Figura 20: Pagină finalizare rezervare

În final, dacă îi convin datele rezervării utilizatorul va apăsa butonul Finish Booking și va fi redirecționat către o pagină pentru realizarea plății 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 să 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 făcută o rezervare. În acest email se vor afla și informațiile necesare pentru crearea actelor închirierii camerei, acestea fiind date de utilizator la înregistrarea în aplicație.

De asemenea, fiecare utilizator poate vedea o listă cu toate rezervările sortate descrescător după 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.

3.4 Comandă room-service

Pentru a face o comandă room-service un utilizator trebuie să acceseze pagina cu lista tuturor alimentelor posibile pentru comandă (vezi figura 21), această pagină se poate accesa de pe pagina de acasă a aplicației unde se găsește o descriere a hotelului cât și a serviciilor oferite de acesta sau din bara de meniu a aplicației la Room-service.

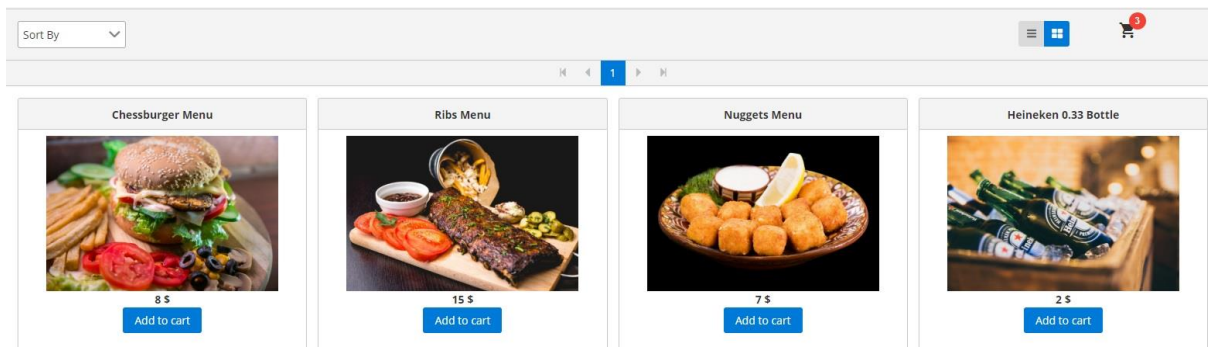


Figura 21: Pagină listare alimente posibile pentru comandă room-service

După ce ajunge aici, utilizatorul poate să sorteze alimentele în funcție de preț pentru a găsi alimentele dorite de el. Apăsând 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.

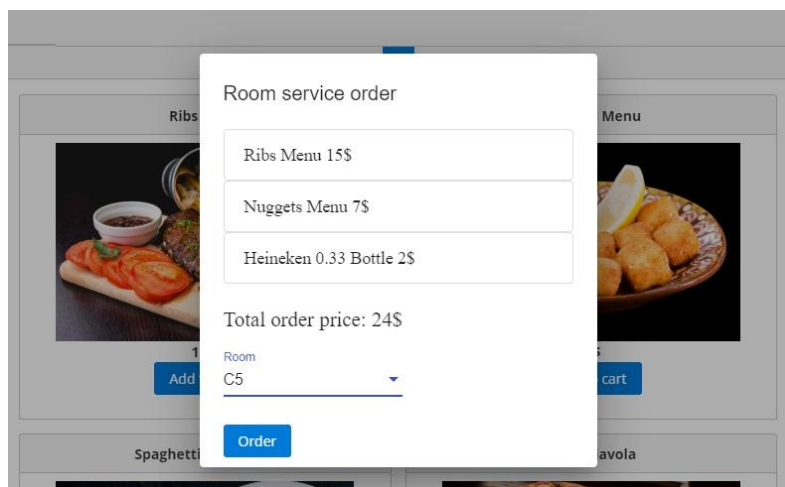


Figura 22: Modal coș care reprezintă o comandă room-service

Apoi dacă îi convine prețul și alimentele alese acesta trebuie să aleagă o cameră din camerele rezervate de el a căror rezervare se desfășoară în momentul actual după cum se vede în figura 22 (o rezervare activă fiind o condiție necesară pentru plasarea unei comenzi de room-service). De asemenea, dacă un client al hotelului dorește să 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. După plasarea comenzii utilizatorul va fi dus la o pagină unde îi vor apărea tot istoricul comenzilor cu detalii despre acestea, ultima fiind poziționată cel mai sus (vezi figura 23).


 General information Bookings Room service orders	Room: C5	STATUS: RECEIVED
	Order time: 18/06/2019 12:46	
	Heineken 0.33 Bottle 2\$	
	Nuggets Menu 7\$	
	Ribs Menu 15\$	
	Total price: 24\$	
	Room: C5	STATUS: DELIVERED

Figura 23: Pagină listare comenzi room-service ale utilizatorului

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.


 General information Bookings Room service orders All room service orders All bookings	Active	Inactive
	Room: C5	STATUS: RECEIVED
	Order time: 18/06/2019 12:46	
	Ribs Menu 15\$	
	Nuggets Menu 7\$	
	Heineken 0.33 Bottle 2\$	
	Total price: 24\$	
	NEXT	

Figura 24: Pagină listare și management al tuturor comenzilor de către angajați ai hotelului

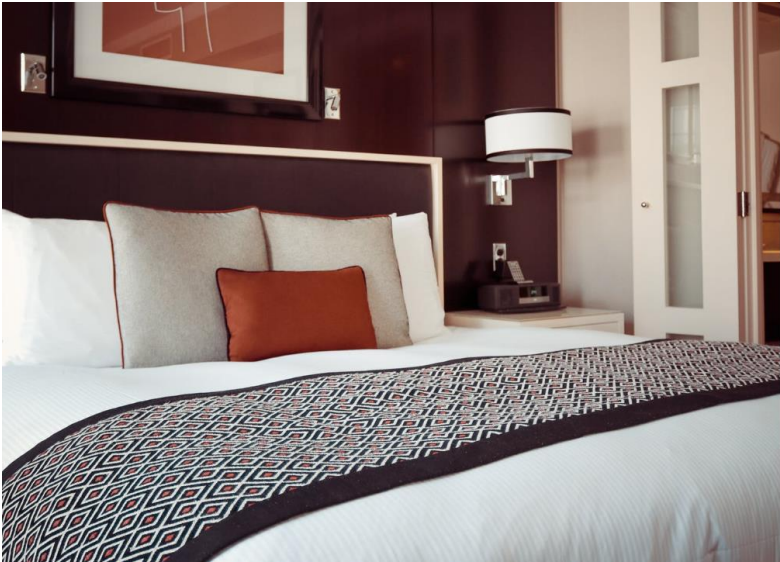
Odată ce comanda a fost trimisă utilizatorii de tip angajat pot vedea comanda în lista cu toate comenzile active în momentul de față și vor trece comanda în următorul pas apăsând butonul Next din figura de mai sus, astfel actualizându-se statusul acesteia (vezi figura 24).

Următorii pași ai comenzii sunt:

- **PREPARING**: comanda a fost preluată de angajații hotelului și au început prepararea acesteia;
- **PREPARED**: comanda a fost preparată de bucătari și acum se așteaptă preluarea comenzii de către personalul hotelului care aduce comanda în cameră;
- **DELIVERING**: comanda a fost preluată și este în proces de livrare la camera clientului;
- **DELIVERED**: comanda a fost livrată în cameră clientului.

3.5 Activități admin

Utilizatorul de tip admin 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. Inițial este introdus în aplicație un utilizator de tip admin pentru a putea crea noi utilizatori de tip admin și angajat.



Name

Beds

Maximum occupancy

Price

Size in square feet

Description

☐ Coffee maker

☐ Safe

☐ Kitchen

Add image

+ Choose

Save

Figura 25: Pagină creare categorie de cameră, accesibilă utilizatorilor de tip admin

3.6 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:

- prețul inițial al categoriei de cameră;
- în funcție de procentajul de camere ocupate din categoria de camere prețul poate crește după câte camere sunt libere pe durata rezervării sau poate să scadă dacă 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 precedente la hotel în funcție de numărul de rezervări care le-a avut și în funcție de prețul total al rezervărilor.

Algoritmul este implementat într-un serviciu separate definit în interfața PriceService (vezi figura 26), această interfață are o metodă care returnează un dicționar în care cheile sunt nume de categorii de cameră și valoarea este prețul pentru fiecare categorie de cameră, 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);  
  
}
```

Figura 26: Interfața pentru serviciul care generează prețul rezervării

Această interfață este implementată de o clasă care suprascrie metoda din interfață și are ca membri o structură de tip dicționar în care se vor pune categoriile cu prețul lor și serviciile pentru cameră și rezervări care vor fi folosite pentru calcularea prețului categoriilor (vezi figura 27). Aceste două servicii sunt injectate în serviciul curent prin procedeul de injectare de dependențe 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;
    }
}

```

Figura 27: Inițializarea serviciului care se ocupă de generarea prețului rezervării

Metoda suprascrisă din interfață apelează altă metodă (calculateCategoriesPrices găsită în figura 28) a cărei scop este să pună în dicționar la chei numele categoriilor de camere primite ca parametru de aceasta și pentru valoarea fiecărei chei apelează o metodă (calculateTotalBookingPrice găsită în figura 28) a cărei scop este să calculeze și să returneze prețul rezervării pentru o categorie de cameră primită 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 (vezi figura 29). Apoi prețului curent i se vor aplica reduceri si majorări după următoarele metode:

- getPriceRemainingPrecentageByNumberOfBookingDays (figura 33): calculează un procentaj 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 (figura 34): 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 (figura 35): calculează un procentaj al prețului pe baza rezervărilor precedente 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).

```

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, scale: 2);
}

```

Figura 28: Metoda pentru inițializare dicționar categorie-preț și metoda calculare preț total categorie

La apelarea metodei pentru obținere a prețului pentru o zi, acesta pune într-o variabilă prețul de bază al categoriei, după care adaugă prețului o valoare în funcție de numărul de camere ocupate apelând funcția `getExtraPriceByRoomsOccupancy` și o nouă valoare dacă ziua respectivă se află pe lista de zile alese pentru majorare a prețului apelând funcția `getExtraPriceBySpecialDates` după cum se vede în tabelul de mai jos.

```

private Double getBookingPriceForDayByCategory(Category category, LocalDate date) {
    Double totalDayPrice = category.getPrice();
    totalDayPrice += getExtraPriceByRoomsOccupancy(category, date);
    totalDayPrice += getExtraPriceBySpecialDates(category, date);
    return totalDayPrice;
}

```

Figura 29: Metoda care calculează prețul categoriei pentru o anumită zi

Pentru calcularea adaosului de preț după 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 total de camere din acea categorie. Pentru aflarea numărului de camere disponibile în acea zi se folosește metoda `getAvailableRoomsBetweenDates` din serviciul `roomService`. Apoi pentru calcularea adaosului de preț se folosește o structură de tip enumerare în care într-o variabilă se vor găsi valorile la care dacă un procentaj de ocupare este mai mare sau egal decât valoare percent din enumerare atunci valoarea cu care se va multiplica prețul este valoare din câmpul `multiplyValue`. Astfel se parcurg toate valorile din enumerarea `CategoryOccupancy` iar în cazul în care procentajul de ocupare al camerelor actual este mai mare decât procentajul din enumerare vom returna valoarea prețului categoriei de camere înmulțită cu valoarea de multiplicare. De asemenea, avem și un caz special, adică dacă mai este doar o zi până la rezervarea pentru ziua dorită și procentajul de ocupare este mic vom returna o valoare negativă să se adauge la preț. Vezi cele 2 tabele de mai jos pentru detalii.

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.

```

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;
}

```

Figura 30: Metoda care calculează adaosul de preț al categoriei în funcție de nivelul de ocupare al categoriei într-o anumită zi

```

public enum CategoryOccupancy {
    NINETY_PERCENT( label: "NinetyPercent", percent: 0.9, multiplyValue: 0.25),
    EIGHTY_PERCENT( label: "EightyPercent", percent: 0.8, multiplyValue: 0.2),
    SEVENTY_PERCENT( label: "SeventyPercent", percent: 0.7, multiplyValue: 0.15),
    SIXTY_PERCENT( label: "SixtyPercent", percent: 0.6, multiplyValue: 0.1),
    FIFTY_PERCENT( label: "FiftyPercent", percent: 0.5, multiplyValue: 0.05);

    private String label;
    private Double percent;
    private Double multiplyValue;
}

```

Figura 31: Enumerarea care este folosit pentru a calcula adaosul de preț pentru o zi în funcție de nivelul de ocupare

La calcularea adaosului de preț după zile special se va folosi în mod similar o enumerare în care se vor găsi ziua, luna și procentajul de multiplicare (vezi tabelul 32 pentru detalii suplimentare).

Exemplu: dacă în enumerare se găsește ziua 31, luna 12 și 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;
}

```

Figura 32: Metoda care dă un adaos de preț în caz ca ziua primită este pe o listă cu zile în care se majorează prețul

Pentru aflarea noului procentaj al prețului în funcție de numărul de zile al rezervării, este folosit altă enumerare a cărei câmpuri sunt un număr de zile și un procentaj de reducere, iar dacă numărul de zile este mai mare sau egal ca cel din enumerare vom returna valoarea 1 minus procentajul corespunzător aceluși număr de zile, astfel rezultând noul procentaj al prețului.

Exemplu: dacă în enumerare 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 calculat inițial, 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 : NumberOfDaysDiscounts.values()) {
        if (bookingDays >= numberOfDaysDiscounts.getNumberOfDays()) {
            return 1.0 - numberOfDaysDiscounts.getDiscountPercentage();
        }
    }
    return 1.0;
}
```

Figura 33: Metoda care oferă o reducere a prețului în funcție de numărul de zile pentru care este făcută rezervarea prin oferirea unui procentaj pentru a se înmulți cu valoarea actuală a prețului

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 după cum se vede în figura de mai jos.

```
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;
}
```

Figura 34: Metoda care mărește procentajul prețului actual dacă au fost făcute multe rezervări în ultimele 24 de ore

În final, la apelarea metodei pentru calcularea noului procentaj în funcție de rezervările precedente ale utilizatorului (vezi figura de mai jos), 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;
}
```

Figura 35: Metoda care oferă un procentaj scăzut al prețului în funcție de rezervările anterioare ale clientului

3.7 Serviciu emailuri

La trimiterea de emailuri se folosește Gmail SMTP. Pentru a se putea folosi acest serviciu trebuie adăugată dependența `spring-boot-starter-mail` și trebuie făcute configurările pentru serverul care trimite emailuri (configurările se fac în fișierul “`application.properties`” după cum se vede în figura de mai jos).

```
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
```

Figura 36: Configurări server pentru mail (gmail)

Odată ce avem aceste configurări făcute pentru a trimite un email trebuie creat mesajul, în care punem subiectul, destinatarul și mesajul propriu (vezi figura 37). 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 relativ lungă (mai mult de 2-3 secunde) și nu vrem ca aplicația noastră să fie blocată în acest timp.

```
@Service
public class EmailServiceImpl implements EmailService {

    private JavaMailSender javaMailSender;

    public EmailServiceImpl(JavaMailSender javaMailSender) {
        this.javaMailSender = javaMailSender;
    }

    @Override
    public void sendSimpleWelcomeMail(CustomerDto customerDto) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setTo(customerDto.getEmail());
        message.setSubject("Welcome to our community!");
        String stringMessage="Hello " + customerDto.getName()+"\n\n";
        stringMessage+="Thank you for registering to our site!";
        message.setText(stringMessage);
        createNewThreadAndSendMail(message);
    }

    private void createNewThreadAndSendMail(SimpleMailMessage simpleMailMessage){
        EmailSender emailSender=new EmailSender(simpleMailMessage,javaMailSender);
        new Thread(emailSender).start();
    }
}
```

Figura 37: Clasă creare și trimitere email

```
public class EmailSender implements Runnable {  
  
    private SimpleMailMessage message;  
    private JavaMailSender javaMailSender;  
  
    public EmailSender(SimpleMailMessage message, JavaMailSender javaMailSender) {  
        this.message = message;  
        this.javaMailSender = javaMailSender;  
    }  
  
    @Override  
    public void run() { javaMailSender.send(message); }  
}
```

Figura 38: Clasă folosită pentru trimiterea unui email într-un nou fir de execuție

3.8 Plată PayPal

La finalizarea rezervării unei camere utilizator trebuie să efectueze plata rezervării. Această plată este făcută prin PayPal în mediul de testare oferit de PayPal numit “sandbox”. Realizarea plății PayPal în mediul sandbox este similară cu plata reală, pe parte de cod schimbându-se doar un parametru al unei funcții care semnalează dacă plata să fie făcută în sistemul real sau în sistemul de test. Motivul folosirii acestui sistem de test în aplicație pentru moment este faptul că este gratis și este modalitatea recomandată pentru a dezvolta o aplicație pentru cazul în care se fac greșeli la tranzacții, la final putând trece într-un mediu real foarte ușor cu o investiție de bani pentru a îți deschide conturile necesare.

Pentru a începe dezvoltarea sistemului de plată prin PayPal inițial trebuie să se creeze un cont la adresa <https://developer.paypal.com>. Mai departe vom crea cel puțin două conturi PayPal de testare (conturi sandbox), unul din aceste conturi fiind folosit ca și contul principal pe care se vor primi bani (contul hotelului), și al doilea cont fiind folosit ca și contul care va trimite banii (contul clientului). Aceste conturi se pot configura cu orice sumă de bani, parolă și adresă de email.

Sandbox Accounts

[Create Account](#)

Questions? Check out the [Testing Guide](#). Non-US developers should read our [FAQ](#).

To link your sandbox account to your developer account, [log in with PayPal](#) and provide your sandbox account credentials.

Total records: 2

<input type="checkbox"/>	Email Address	Type	Country	Date Created	Status	Actions
<input type="checkbox"/>	▶ stefandx9726-facilitator@gmail.com	BUSINESS	US	11 Jun 2019	complete	...
<input type="checkbox"/>	▶ stefandx9726-buyer@gmail.com	PERSONAL	US	11 Jun 2019	complete	...

[Delete Accounts](#)

Figura 39: Creare conturi PayPal Sandbox

Următorul pas este introducerea sistemului extern pentru dezvoltarea unui sistem de plată PayPal. Pentru folosirea acestui sistem este folosit un SDK oferit de PayPal pentru dezvoltarea de aplicații (<https://github.com/paypal/PayPal-Java-SDK>). Apoi vom folosi acest SDK pentru a crea un obiect de tip plată (vezi figura de mai jos) care conține informațiile necesare pentru efectuarea unei plăți, adică, printre altele, metoda de plată, id-ul și secretul contului care ar urma să primească banii, suma și tipul monezii de plată.

```

private final String RECEIVER_ACCOUNT_ID = "AshmTt5td4mjBj19DvDYz1H60ErzYlQhkaK1o92jWSq9X1jyGC2spSnUkbPgnJ-30P8RHT_pDNmAkvsu";
private final String RECEIVER_ACCOUNT_SECRET = "ED7j7RfByjIguagJd0priGVxgdUPwil6xPrIlXi0mxi9hx0ju5PjLTcPuEXEpWFPnF67T9TvrF09QTeN";
private final String EXECUTION_MODE = "sandbox";
@Override
public String createPayment(PaymentCreationDto paymentCreationDto) {
    Payment payment = definePayment(paymentCreationDto);
    APIContext apiContext = new APIContext(RECEIVER_ACCOUNT_ID, RECEIVER_ACCOUNT_SECRET, EXECUTION_MODE);
    try {
        Payment createdPayment = payment.create(apiContext);
        for (Links link : createdPayment.getLinks()) {
            if (link.getRel().equalsIgnoreCase("approval_url")) {
                return link.getHref();
            }
        }
    } catch (PayPalRESTException e) {
        System.err.println(e.getDetails());
    }
    return null;
}

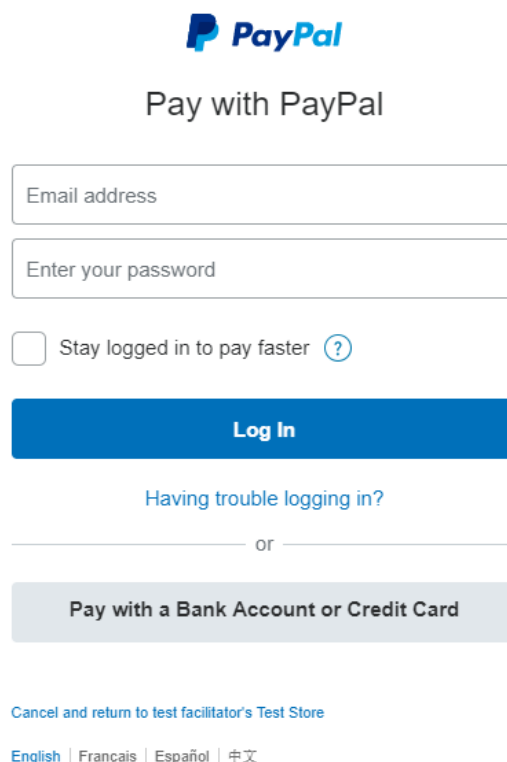
private Payment definePayment(PaymentCreationDto paymentCreationDto) {
    Payment payment = new Payment();
    payment.setIntent("sale");
    payment.setPayer(createPayer());
    payment.setRedirectUrls(createRedirectUrls());
    payment.setTransactions(createPaymentTransactions(paymentCreationDto));
    return payment;
}

```

Figura 40: Metodă creare plată

Pentru efectuarea unei plăți reale trebuie schimbată valoarea constantei EXECUTION_MODE în "live".

Odată apelată metoda createPayment din figura de mai sus, obiectul de tip plată cu informațiile trimise de noi este creat și este un returnat un link care va duce la o pagină de autentificare în contul de PayPal (în cazul nostru la contul de tip sandbox) cum se vede în figura de mai jos. Aici utilizatorul va trebui să-și introducă informațiile și după autentificare să decidă dacă acceptă sau nu plata și în ce mod să se facă plata (vezi figura 42).



The image shows the PayPal login interface. At the top is the PayPal logo. Below it is the heading "Pay with PayPal". There are two input fields: "Email address" and "Enter your password". Below these is a checkbox labeled "Stay logged in to pay faster" with a help icon. A blue "Log In" button is positioned below the checkbox. Underneath the button is a link "Having trouble logging in?". A horizontal line with the word "or" in the center separates the login section from the payment method section. The payment method section has a button labeled "Pay with a Bank Account or Credit Card". At the bottom, there is a link "Cancel and return to test facilitator's Test Store" and a row of language links: "English", "Français", "Español", and "中文".

Figura 41: Pagină autentificare PayPal

The screenshot shows the PayPal checkout interface. At the top left is the PayPal logo and a shopping cart icon. Below the logo, it says "Hi, test!". The "Ship to" section shows the address: "test buyer, 1 Main St, San Jose, CA 95131 United States", with a "Change >" link. The "Pay with" section lists three options: "Balance" (selected with a radio button), "CREDIT UNION 1 x-2652", and "Visa x-7343". There are also links to "Add a debit or credit card" and "See Offers & Apply for PayPal Credit". A "Continue" button is at the bottom. On the right, there's a graphic of shopping bags with a shield and the text "PayPal is the safer, easier way to pay".

Figura 42: Selectare metodă plată și confirmare

Dacă utilizatorul s-a autentificat și a ales să accepte plata acesta va fi redirecționat spre adresa de confirmare la care în parametri URL-ului se vor adăuga id-ul plății și id-ul utilizatorului care efectuează plata. În caz contrar, acesta va fi redirecționat către adresa folosită la anularea unei plăți. În final, dacă a fost acceptată plata se vor folosi acei parametri pentru executarea plății după cum se vede în figura de mai jos.

```
@Override
public void executePay(String paymentId, String payerId) throws PayPalRESTException {
    APIContext apiContext = new APIContext(RECEIVER_ACCOUNT_ID, RECEIVER_ACCOUNT_SECRET, EXECUTION_MODE);

    Payment payment = new Payment();
    payment.setId(paymentId);

    PaymentExecution paymentExecution = new PaymentExecution();
    paymentExecution.setPayerId(payerId);
    payment.execute(apiContext, paymentExecution);
}
```

Figura 43: Metodă care execută operația de plată

Concluziile lucrării

Așadar, această aplicație, deși pare asemănătoare cu alte aplicații de pe piața, oferă un serviciu foarte diferit clienților, adică oferă posibilitatea cazării într-o cameră și comenzilor de room-service doar prin intermediul aplicației față de restul aplicațiilor existente la care poți face rezervarea online prin intermediul aplicației dar apoi pentru a intra în camera de hotel este necesar să mergi la recepție pentru a face check-in-ul. Acest lucru consider că este un mare plus deoarece nu poate apărea riscul pierderii unei perioade lungi de timp la recepția hotelului și, de asemenea, consider că este în ton cu felul în care se dezvoltă societatea în ziua de azi, adică să faciliteze comoditatea clientului care vrea să se cazeze la hotel.

Aplicație este făcută să fie responsivă pentru orice tip de dispozitiv și respectă tehnicile de clean code fiind ușor de dezvoltat mai departe pentru a adăuga noi funcționalități.

Pe viitor, această aplicație ar putea fi extinsă prin adăugarea posibilității de a face rezervări pentru anumite săli de conferință ale hotelului și a face rezervări la evenimente de tip spa și masaj. La algoritmul folosit pentru crearea unui preț variabil s-ar putea adăuga încă un criteriu pentru calcularea prețului prin preluarea informațiilor legate de prețul camerelor de pe alte aplicații ale unor hoteluri existente din zona hotelului care ar folosi aplicația.

De asemenea, s-ar putea adăuga un mecanism prin care angajați să adauge în aplicație toate produsele consumate, de exemplu prosoape, șampon, săpun. Apoi pe baza istoricului acestora și pe baza istoricului produselor consumate în comenzile de room-service s-ar putea folosi un algoritm de învățare automată pentru a prezice stocul necesar pe viitor pentru hotel.

Bibliografie

1. <https://spring.io/guides/gs/rest-service/>
2. <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>
3. <https://www.baeldung.com/spring-controller-vs-restcontroller>
4. <https://www.baeldung.com/spring-requestmapping>
5. <https://www.baeldung.com/spring-data-jpa-query>
6. <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>
7. <https://dzone.com/articles/bounty-spring-boot-and-postgresql-database>
8. <https://www.vogella.com/tutorials/JUnit/article.html>
9. <https://www.vogella.com/tutorials/Mockito/article.html>
10. <https://www.baeldung.com/spring-email>
11. <https://angular.io/tutorial>
12. <https://coursetro.com/posts/code/171/Angular-7-Tutorial---Learn-Angular-7-by-Example>
13. <https://material.angular.io/>
14. <https://www.jetbrains.com/help/idea/running-and-debugging-typescript.html>
15. <https://getbootstrap.com/>
16. <https://www.w3schools.com/bootstrap4/default.asp>
17. <https://www.primefaces.org/primeng/#/>
18. <https://angular.io/guide/styleguide>
19. <https://itnext.io/choosing-a-highly-scalable-folder-structure-in-angular-d987de65ec7>
20. <https://github.com/mathisGarberg/angular-folder-structure>
21. <https://malcoded.com/posts/angular-fundamentals-modules/>
22. <https://www.technouz.com/4772/angular-6-app-structure-with-multiple-modules/>
23. <https://code-maze.com/angular-best-practices/>
24. <https://www.freecodecamp.org/news/how-to-center-things-with-style-in-css-dc87b7542689/>
25. <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>
26. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Ordering_Flex_Items

27. <https://flexboxfroggy.com/>
28. <https://www.baeldung.com/security-spring>
29. <https://www.baeldung.com/securing-a-restful-web-service-with-spring-security>
30. <https://www.baeldung.com/spring-security-oauth-jwt>
31. https://chariotsolutions.com/blog/post/angular-2-spring-boot-jwt-cors_part2/
32. <https://mhernan.org/blog/token-based-authentication-with-angular/#localstorage>
33. <https://codecraft.tv/courses/angular/dependency-injection-and-providers/tokens/>
34. <https://medium.com/engineerbabu/angular-authentication-using-jwt-d846c5ce0ac6>
35. https://medium.com/@ryanchenkie_40935/angular-authentication-using-route-guards-bf7a4ca13ae3
36. <https://nirajsonawane.github.io/2018/10/27/Angular-Material-Tabs-with-Router/>
37. <https://medium.com/@nikhildevre/creating-tabs-using-angular-material-2-and-angular-4-routing-3634c3d0f7cc>
38. <https://blog.angularindepth.com/angular-router-series-secondary-outlets-primer-139206595e2>
39. https://www.w3schools.com/howto/howto_js_tabs.asp
40. <https://css-tricks.com/left-and-right/>
41. <https://css-tricks.com/couple-takes-sticky-footer/>
42. <https://github.com/paypal/PayPal-Java-SDK>
43. <https://developer.paypal.com/docs/api/quickstart/payments/#additional-information>
44. <https://developer.paypal.com/developer/accountStatus/>
45. <https://c4model.com/>
46. <https://structurizr.com/express?src=/static/express/bigbankplc/system-context.json>
47. <https://www.draw.io/>
48. <https://hatchful.shopify.com/>
49. <https://www.pexels.com/search/hotel/>
50. <https://pixabay.com/images/search/hotel/>
51. <https://unsplash.com/search/photos/room-service>