

**UNIVERSITATEA BABEȘ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ**

LUCRARE DE LICENȚĂ

Dezvoltarea aplicațiilor web containerizate

**Conducător științific
Lector Dr. Camelia Chisăliță-Crețu**

*Absolvent
Alexandru Mihai Sabou*

2021

ABSTRACT

This paper contains 6 chapters. The first chapter presents an introduction to context, the motivation for choosing this topic for my bachelor degree and a short presentation of my application. Renty is a web application based on Docker containers, which facilitates a sports place reservation. It is built for 2 big categories of users: authenticated and unauthenticated users. The authenticated users are divided also in 3 categories with different roles: admin, owner, renter.

The second chapter presents a short history of the World Wide Web. This appeared at CERN in Switzerland in 1990 as a need for scientists to share the discovered data between them in a faster and easier way. Furthermore it presents the main web protocols: HTTP, SSL, SOAP and WebSocket. Also, it contains examples of requests/responses to/from the server depending on the used web protocol.

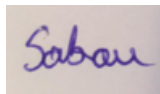
The third chapter presents REST services, their properties, HTTP methods and HTTP response codes. Furthermore it contains few advantages and comparisons over SOAP architecture.

The fourth chapter presents an analysis of tools that use containerization technology such as Docker, Kubernetes and OpenShift.

The fifth chapter presents in a detailed way the development process of Renty. It contains the defined use cases, the architecture of the application and of the project, the main technologies and frameworks used in the development process. It also contains the way I securized the application using Spring Security framework which offers a high level of security and many pre-implementations of the code. Moreover, it contains a description of Stripe payment integration which nowadays is in trend, the environments used for local development, test and production. It is also presented in a detailed way how I configured the Ubuntu OS to be the host of my web application. Besides that, it is presented the docker configuration files and the bash script for managing the application and storing backups. It is also presented the CI/CD/CA processes and how it interacts between them and Ubuntu OS.

The sixth chapter and the last one, presents some conclusions, further improvements and features for Renty application.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.



Cuprins

1	Introducere	4
1.1	Motivarea alegerii temei	4
1.2	Prezentarea pe scurt a temei abordate	5
1.3	Structura lucrării	5
2	World Wide Web	7
2.1	Istoria World Wide Web-ului	7
2.2	Principalele protocoale web	7
2.2.1	HTTP/S	7
2.2.2	WebSocket	8
2.2.3	SSL	9
2.2.4	SOAP	11
3	Servicii RESTful	13
3.1	Introducere în servicii RESTful	13
3.2	Utilizare	14
3.2.1	Proprietăți	14
3.2.2	Metode HTTP	14
3.2.3	Coduri de răspuns HTTP	15
3.3	Avantaje ale arhitecturilor RESTful	16
4	Dezvoltarea aplicațiilor bazată pe containerizare	17
4.1	Mecanismele utilizării containerelor în dezvoltarea software	17
4.2	Analiza instrumentelor folosite în containerizare	17
5	Dezvoltarea aplicației Renty	21
5.1	Etapa de analiză	21
5.1.1	Descriere funcțională	21
5.1.2	Cazuri de utilizare	21
5.2	Etapa de proiectare	41
5.2.1	Arhitectura aplicației	41

5.2.2	Structura bazei de date	41
5.2.3	Structura proiectului și diagrame de clase	42
5.3	Tehnologii și framework-uri utilizate	44
5.4	Etapa de implementare	46
5.4.1	Securitatea aplicației	46
5.4.2	Stripe API	50
5.4.3	Medii de lucru	52
5.4.4	Docker	53
5.5	Etapa de testare	54
5.6	Etapa de deploy	55
5.6.1	Găzduirea aplicației	55
5.6.2	Configurarea Linux server-ului și a aplicației web	56
5.6.3	Integrare continuă	61
6	Concluzii și posibilități de dezvoltare viitoare	65
6.1	Concluzii	65
6.2	Posibilități de dezvoltare	65
6.3	Îmbunătățiri viitoare	66
	Bibliografie	67

Capitolul 1

Introducere

Înainte, aplicațiile desktop offline erau la ordinea zilei, dar după apariția și popularizarea Internetului acestea au început să migreze încet, încet la aplicații web sau cel puțin la aplicații desktop care depind de un serviciu web. Un motiv foarte întemeiat este că la o nouă versiune a aplicației desktop (care era offline) trebuia să fie configurată adecvat pentru a putea fi utilizată. Era un pas foarte greoi și consumator de timp.

Această revoluție a aplicațiilor, a adus odată cu ea și serviciile web cum ar fi serviciile RESTful, MVC sau WebForms aflându-se în plin proces de evoluție și continuând să apară și altele.

1.1 Motivarea alegerii temei

Dezvoltarea unei aplicații web de la proiectare până la folosirea acesteia de către publicul larg a fost una dintre cele mai mari provocări ale mele. De asemenea, o soluție simplă și rapidă pentru public este apreciată, iar aplicațiile web sunt cele mai la îndemână.

Pe lângă dezvoltarea aplicațiilor web, containerizarea acestora a fost, de asemenea, cea mai mare provocare pentru mine. Înglobarea serviciilor în containere aduce câteva beneficii cum ar fi:

- **portabilitate:** sistemul de servicii este ușor de instalat pe orice sistem ce are instalat instrumentul ce se bazează pe tehnologia de containerizare;
- **izolare:** containerele sunt independente și pot fi configurate și pe alte versiuni ale tehnologiilor pe care serviciile le folosesc;
- **performanță:** folosesc direct resursele sistemului.

1.2 Prezentarea pe scurt a temei abordate

Dezvoltarea aplicațiilor web ce folosesc instrumente ce au tehnologia de containerizare aduce foarte multe beneficii și minimizează apariția erorilor la etapa de construire și a livrării sistemelor de aplicații. Deoarece aplicațiile containerizate prezintă un mediu total izolat de dezvoltare și pe lângă alte beneficii aduse de acesta, am ales astfel să dezvolt aplicația **Renty** unde serviciile sunt înglobate în containere. Scopul ei principal este de a facilita procesul de rezervare a unui loc pentru a face sport.

Este construită pentru două mari categorii de utilizatori, cei neautentificați și cei autentificați. Utilizatorii neautentificați pot doar să vizualizeze ce locuri de recreere sunt prezentate pe platformă, pe când cei autentificați, la rândul lor, se împart în 3 categorii. Utilizatorii care au rolul de “RENTER” pot să facă rezervări, utilizatorii care au rolul de “OWNER” pot de asemenea să facă rezervări, să-și modifice terenurile și activitățile sau să vadă anumite statistici și utilizatorii care au rolul de “ADMIN” unde aceștia pot face ce pot face și proprietarul și clientul, dar cu acces limitat pe anumite resurse. Acesta din urmă mai are dreptul de a face managementul utilizatorilor.

1.3 Structura lucrării

Este împărțită pe 6 capitole și o secțiune dedicată bibliografiei. Astfel:

- **Capitolul 1:** conține o introducere în context, motivarea alegerii temei de aplicații web containerizate precum și structura lucrării
- **Capitolul 2:** prezintă o scurtă istorie a World Wide Web-ului și o descriere a principalelor protocoale web (HTTP/S, WebSocket, SSL, SOAP)
- **Capitolul 3:** prezintă de asemenea o scurtă istorie a serviciilor REST, constrângerile (client-server, nu are stare, cacheable, interfață uniformă, sistem startificat, cod la cerere) aduse împreună cu apariția acestor servicii, proprietăți (sigure, idempotente, cacheable), metode HTTP, coduri de răspuns și câteva avantaje ale arhitecturii REST
- **Capitolul 4:** prezintă o analiză a instrumentelor ce folosesc tehnologia de containerizare cum ar fi Docker, Kubernetes și OpenShift
- **Capitolul 5:** prezintă dezvoltarea aplicației Renty, configurarea unui Ubuntu server pentru a îndeplini cerințele de “gazdă”, înglobarea serviciilor utilizate de aplicație în containere Docker, automatizarea procesului de livrare a

aplicației și analiză a codului (Jenkins și SonarQube), elaborarea bash script-ului pentru gestionarea resurselor aplicației, securitatea precum și plata cu cardul

- **Capitolul 6:** prezintă câteva concluzii, posibilități și îmbunătățiri de dezvoltare în viitor

Capitolul 2

World Wide Web

World Wide Web-ul sau pe scurt Web este un set de site-uri web sau pagini web care se află pe servere web conectate prin Internet. Este cel mai mare serviciu pentru obținerea de informații și oferă utilizatorilor documente care sunt legate prin hypertext sau hyperlink-uri. Acestor documente le sunt asignate o adresă numită Uniform Resource Identifier/Locator (URI/L) [2].

2.1 Istoria World Wide Web-ului

World Wide Web-ul a apărut odată cu nevoia oamenilor de știință de a partaja între ei toate resursele și cercetările mai ușor. Deoarece proiectele erau distribuite, fiind formate din mai multe echipe, englezul Tim Barners-Lee a conceput World Wide Web-ul în anul 1990 în Elveția la Organizația Europeană pentru Cercetare Nucleară (eng. *European Organization for Nuclear Research* sau pe scurt *CERN*). Până spre sfârșitul anului 1990, ca Web-ul să funcționeze, Barners-Lee a creat toate uneltele necesare: HyperText Transfer Protocol (HTTP), HyperText Markup Language (HTML), web browser-ul WorldWideWeb care a fost de asemenea și un editor web, HTTP server-ul care mai târziu a căpătat numele de CERN httpd și primul server web care și în zilele de astăzi poate fi accesat la adresa `http://info.cern.ch/` [24, 13].

2.2 Principalele protocoale web

2.2.1 HTTP/S

Apariția HTTP-ului a apărut odată cu proiectul World Wide Web a lui Tim Barners-Lee în 1990 la CERN în Elveția. Web-ul are la bază HTTP-ul fiind folosit la transferul de hypertext dintre client și server [24].

Un exemplu de cerere HTTP arată ca în Figura 2.1:

```
GET /hello.html HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; MSIE5.01; Windows NT)
Host: localhost:8080
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Figura 2.1: Exemplu de HTTP request - GET

Un exemplu de răspuns HTTP arată ca în Figura 2.2:

```
HTTP/1.1 200 OK
Date: Mon, 01 Jul 2021 22:55:34
Server: nginx/1.18.0 (Ubuntu)
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

Figura 2.2: Exemplu de HTTP response

HTTPS este o combinație dintre HTTP și SSL care prezintă câteva avantaje cum ar fi [30]:

- oferă o comunicare securizată și criptată între server și client;
- serverul permite cereri HTTPS doar dacă are un certificat valid semnat de către instituțiile care se ocupă cu eliberarea de astfel de certificate;
- de obicei cererile HTTP sunt redirectionate către HTTPS (dacă server-ul permite acest lucru).

2.2.2 WebSocket

WebSocket este un protocol bidirecțional de comunicare dintre un server și un client folosind o singură conexiune TCP. A apărut deoarece protocolul HTTP nu a fost menit să fie utilizat pentru o comunicare bidirecțională. Astfel, clientul primește date de la server fără ca acestea să fie cerute fiind o comunicare în timp real. De obicei aceste comunicări sunt făcute pe portul 80, iar dacă este integrat și SSL-ul, atunci se folosește portul 443 [18].

O cerere de la client la server folosind acest protocol este prezentată Figura 2.3, iar un răspuns de la server este prezentat în Figura 2.4:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Figura 2.3: Exemplu cerere HTTP pentru inițializare WebSocket [18]

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+x0o=
Sec-WebSocket-Protocol: chat
```

Figura 2.4: Exemplu de răspuns HTTP pentru inițializare WebSocket [18]

2.2.3 SSL

Secure Sockets Layer (SSL) este un protocol care oferă o comunicare criptată și securizată între server și client. A fost dezvoltat în 1990 de către compania Netscape, dar versiunea 1.0 nu a fost livrată către public. Versiunea 2.0 a avut probleme serioase, iar versiunea 3.0 a fost livrată în 1996 [20].

Pentru a nu permite a unei a treia părți (eng. *man-in-the-middle*) să examineze pachetele care pot conține informații confidențiale este necesară o criptare a acestor pachete. Cea mai sigură metodă de criptare se numește criptografie asimetrică (eng. *asymmetrical cryptography*) care necesită două chei, una publică și una privată. Matematica pentru obținerea acestor două chei este destul de complexă și greu de “spart”. Cheia publică se folosește pentru criptare, iar cheia privată se folosește pentru decriptare. Deoarece criptografia asimetrică implică aceste matematici complexe, este necesar de asemenea și de resurse hardware mai performante. *Transport Layer Security* (TLS) ar rezolva această problemă deoarece se folosește o criptare simetrică (eng. *symmetrical cryptography*) utilizându-se aceeași cheie pentru a cripta și decripta pachetele numită cheie de sesiune (eng. *session key*) [20].

Procesul de integrare a SSL-ului este unul foarte complex. Pașii de integrare pe o schemă largă sunt următorii [20]:

1. Clientul cere o conexiune sigură, iar server-ul răspunde cu seturi de instrumente algoritmice pentru crearea conexiunilor criptate. Clientul compară

acest lucru cu propriile lui instrumente algoritmice și “anunță” server-ul că se vor folosi.

2. Server-ul furnizează certificatul lui digital, iar clientul verifică autenticitatea acestuia.
3. Folosindu-se cheia privată și cheia publică, server-ul și clientul stabilesc o cheie de sesiune care va fi folosită pentru criptarea pachetelor. Există mai multe metode de obținere a cheii de sesiune (prin combinarea celor două chei) numite schimb de chei Diffie-Hellman.

Acești pași sunt evidențiați în Figura 2.5 de mai jos:



Figura 2.5: Integrarea SSL/TLS [22]

2.2.4 SOAP

Simple Object Access Protocol (SOAP) este un protocol de comunicare având la bază XML-ul. Constă din trei părți și anume: definirea unui cadru care descrie ce este într-un mesaj și cum ar trebui interpretat, un set de reguli de formare a datelor și a tipurilor de date și o convenție pentru exprimarea apelurilor și răspunsurilor la distanță (eng. *Remote Procedure Call* sau pe scurt RPC). Mesajele SOAP sunt frecvent întâlnite în combinație cu HTTP sau JMS, dar pot fi utilizate și alte protocoale de transport [12].

Structura unui mesaj SOAP arată ca în Figura 2.6.

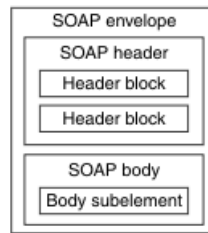


Figura 2.6: Structura unui mesaj SOAP [17]

După cum se poate observa din Figura 2.6, deducem că un mesaj SOAP are mai multe elemente: **envelope**, **header** și **body**, dar mai are și un subelement al lui body numit **fault** (care nu este prezent în figură). Elementul envelope este principalul element dintr-un mesaj SOAP și conține două subelemente: header (care este opțional) și body care este necesar. Elementul header conține informații legate de aplicație care urmează să fie procesate de noduri SOAP de-a lungul căii mesajului. Elementul body conține informații despre apel sau răspuns, iar elementul fault este folosit atunci când se raportează erori [17].

În Figura 2.7 este prezentat un exemplu de mesaj SOAP care conține în header blocurile m:reservation și n:passenger și body-ul care conține elementul p:itinerary.

```

<?xml version='1.0' Encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
      <m:reference>uuid:093a2da1-q345-739x-ba5d-pqff98fe8j7d</m:reference>
      <m:dateAndTime>2007-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
      <n:name>Fred Bloggs</n:name>
    </n:passenger>
  </env:Header>
  <env:Body>
    <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
      <p:departure>
        <p:departing>New York</p:departing>
        <p:arriving>Los Angeles</p:arriving>
        <p:departureDate>2007-12-14</p:departureDate>
        <p:departureTime>late afternoon</p:departureTime>
        <p:seatPreference>aisle</p:seatPreference>
      </p:departure>
      <p:return>
        <p:departing>Los Angeles</p:departing>
        <p:arriving>New York</p:arriving>
        <p:departureDate>2007-12-20</p:departureDate>
        <p:departureTime>mid-morning</p:departureTime>
        <p:seatPreference></p:seatPreference>
      </p:return>
    </p:itinerary>
  </env:Body>
</env:Envelope>
  
```

Figura 2.7: Exemplu de mesaj SOAP [17]

Capitolul 3

Servicii RESTful

3.1 Introducere în servicii RESTful

Representational State Transfer sau pe scurt **REST** a apărut ca o alternativă la serviciile **SOAP** sau **WSDL**, dar mult mai simplă. Trecerea la serviciile **REST** a fost marcată de furnizorii de servicii web 2.0 precum **Yahoo**, **Google** sau **Facebook** care au depreciați serviciile **SOAP** sau **WSDL** în favoarea unei alternative mai simple [34].

Serviciile **REST** se definesc printr-un set de principii arhitecturale (specifice aplicațiilor client - server) prin care se poate proiecta servicii web ce se bazează pe resursele unui sistem. **REST** a fost conceput de **Roy Fielding** în anul 2000 la **University of California** în lucrarea lui de dizertație "Architectural Styles and the Design of Network-based Software Architectures" [34].

O arhitectură web poate fi descrisă de un stil arhitectural constând dintr-un set de constrângeri a elementelor din cadrul arhitecturii. Derivând un stil arhitectural prin adăugarea de constrângeri, se poate obține un nou stil arhitectural care reflectă mai bine proprietățile dorite ale unei arhitecturi web moderne. Odată cu apariția stilului arhitectural **REST** a apărut și constrângerile specifice acestuia [19] :

- **client-server** - oferă o separare a celor două componente aducând astfel câteva beneficii cum ar fi: portabilitatea interfeței grafice pe mai multe platforme, permite lucrul în paralel pe cele două componente atâta timp cât interfața nu se modifică, iar clientul nu este interesat de stocarea datelor pe server;
- **nu are stare** (eng. *Stateless*) - fiecare cerere de la client către server trebuie să conțină toată informația necesară pentru ca să fie înțeleasă și nu poate profita de niciun context stocat pe server;
- **cacheable** - fiecare răspuns la o cerere trebuie să fie marcată explicit ca fiind cache sau non-cache; dacă un răspuns este marcat ca fiind cache, atunci acel

client are dreptul de a refolosi datele de răspuns pentru cereri viitoare echivalente;

- **interfață uniformă** (eng. *Uniform Interface*) - implementările sunt decuplate de serviciile pe care le oferă având o evoluție independentă;
- **sistem stratificat** (eng. *Layered System*) - acest stil permite unei arhitecturi să fie compusă din straturi ierarhice; fiecare componentă nu poate “vedea” dincolo de stratul cu care interacționează, de unde se promovează independența fiecărei componente;
- **cod la cerere** (eng. *Code On Demand*) - permite clientului funcționalitatea de a descărca și executa codul sub formă de applet-uri sau scripturi dar reduce vizibilitatea și astfel este o contrângere opțională.

3.2 Utilizare

3.2.1 Proprietăți

Serviciile REST trebuie să îndeplinească niște proprietăți fundamentale. Astfel, acestea trebuie să fie:

- **sigure:** dacă server-ul își păstrează starea [6]
- **idempotente:** dacă se trimite de mai multe ori aceeași cerere HTTP cu același efect, starea server-ului nu se schimbă [5]
- **cacheable:** dacă un răspuns HTTP poate fi salvat și utilizat ulterior fără a mai face aceeași cerere către server [3]

3.2.2 Metode HTTP

O cerere HTTP definește o metodă pentru a indica ce acțiune se va face pe o anumită resursă. Fiecare metodă are semantica ei, dar pot avea în comun unele proprietăți (siguranța, idempotența) [4].

În cele ce urmează, o să prezint toate metodele și rolul acestora în cererile HTTP [4]:

- **GET** se folosește atunci când este nevoie de reprezentarea unei resurse și trebuie să returneze date doar
- **HEAD** este asemănător lui GET, dar aici nu se așteaptă un corp (eng. *body*)

- **POST** se folosește atunci când se vrea o salvare a unei entități, schimbând starea server-ului.
- **PUT** se folosește atunci când se dorește schimbarea reprezentării unei resurse cu o alta
- **DELETE** se folosește atunci când se dorește ștergerea resursei specificate
- **CONNECT** se folosește atunci când se dorește să se deschidă o cale spre server către resursa specificată
- **OPTIONS** se folosește atunci când se descrie opțiunile de comunicare pentru resursa specificată
- **TRACE** efectuează un test de “loop-back” al mesajului de- a lungul căii către resursa țintă
- **PATCH** se folosește atunci când se dorește o modificare parțială asupra unei resurse specificate

3.2.3 Coduri de răspuns HTTP

Un aspect important al unei arhitecturi RESTful este codul de răspuns HTTP. O cerere HTTP s-a executat cu succes dacă codul de răspuns este 200 (“OK”), iar dacă face parte din clasele 3xx, 4xx sau 5xx atunci ceva greșit sau neașteptat s-a întâmplat. În Figura 3.1 se poate vedea un răspuns cu eroarea 404 (“Not Found”) după o cerere HTTP pentru un obiect care nu există [33].

```
404 Not Found
Content-Type: application/xml
Date: Fri, 10 Nov 2006 20:04:45 GMT
Server: AmazonS3
Transfer-Encoding: chunked
X-amz-id-2: /sBIPQxHJCsyRXJwGWNzxuL5P+K96/Wvx4FhvVACbjRfNbhbDyBH5RC511sIzOw0
X-amz-request-id: ED2168503ABB7BF4

<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>NoSuchKey</Code>
  <Message>The specified key does not exist.</Message>
  <Key>nonexistent/object</Key>
  <RequestId>ED2168503ABB7BF4</RequestId>
  <HostId>/sBIPQxHJCsyRXJwGWNzxuL5P+K96/Wvx4FhvVACbjRfNbhbDyBH5RC511sIzOw0</HostId>
</Error>
```

Figura 3.1: Cod de eroare 404 [33]

3.3 Avantaje ale arhitecturilor RESTful

O arhitectură RESTful este preferabilă deoarece, comparativ cu alte arhitecturi cum ar fi SOAP, prezintă o multitudine de avantaje, așa cum sunt prezentate de A. Altwater [8]:

- SOAP permite doar formatul XML, pe când REST mai multe formătări ale datelor;
- permite stocarea în cache a informațiilor care nu sunt modificate și care nu sunt dinamice;
- este mai rapid și folosește mai puțin banda de Internet;
- se poate integra ușor cu site-urile web deja existente, fără a reface infrastructura și permite dezvoltatorilor să adauge funcționalități noi cu ușurință.

Capitolul 4

Dezvoltarea aplicațiilor bazată pe containerizare

4.1 Mecanismele utilizării containerelor în dezvoltarea software

Configurarea unui set de aplicații cap la cap poate fi foarte dificil și consumator de timp. Deseori când se încearcă configurarea aplicației pentru un alt dezvoltator, pașii de configurare trebuie repetați. Dar, în acești pași pot apărea probleme de compatibilitate cu sistemul de operare, probleme când un serviciu depinde de un alt serviciu sau probleme când se schimbă arhitectura aplicațiilor trecându-se la tehnologii mai noi. De aceea, dezvoltatorii s-au gândit să rezolve aceste probleme prin înglobarea acestor dependențe și pachete al aplicațiilor într-un singur loc și anume container. Astfel au apărut containerele Docker unde cu o simplă comandă, mediul de dezvoltare este configurat. De asemenea au mai apărut Kubernetes și OpenShift unde primul este un sistem utilizat pentru automatizarea livrării codului, scalării și managementul aplicațiilor containerizate, iar cel de al doilea este o platformă de aplicații pentru containere bazată pe Kubernetes folosit pentru dezvoltarea aplicațiilor de întreprindere.

Containerele Docker sunt medii de dezvoltare complet izolate având fiecare propriile lor procese, rețele și configurări utilizând același OS Kernel. Sunt bazate pe imagini Docker care sunt un sistem de fișiere care conține toate dependențele și pachetele necesare aplicației pentru a putea funcționa [9].

4.2 Analiza instrumentelor folosite în containerizare

După cum se poate observa din Figura 4.1 și făcând o comparație cu Figura 4.2, deducem că mașinile virtuale folosesc Hypervisor care este pe al doilea nivel după

infrastructura sistemului. De asemenea se poate observa că fiecare mașină virtuală are propriul ei sistem de operare. Din cele prezentate mai sus se deduce că o mașină virtuală consumă mai multe resurse decât un container Docker.

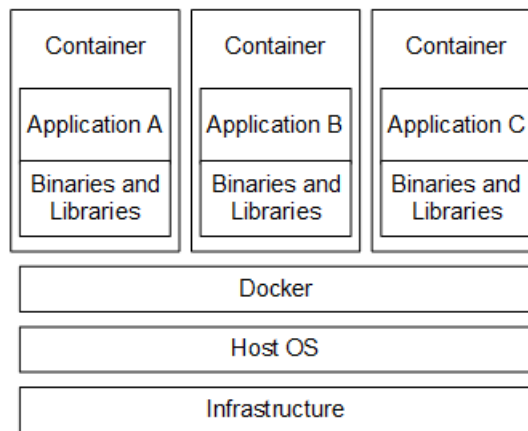


Figura 4.1: Containere Docker pe sistemul de operare [27]

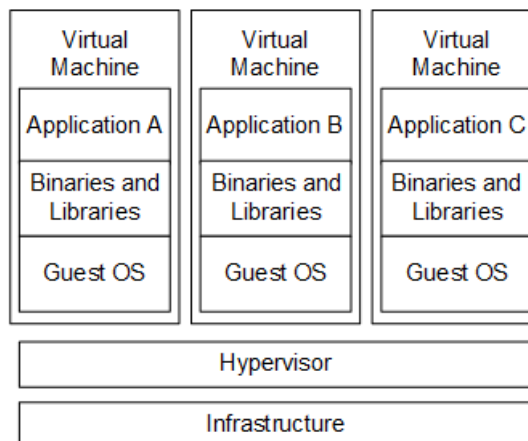


Figura 4.2: Mașini virtuale pe sistemul de operare [27]

În cele ce urmează, prezentăm un set de caracteristici și cum acestea sunt evidențiate în platformele de gestionare a containerelor Kubernetes, Docker și OpenShift [29].

	Docker	Kubernetes	OpenShift
Tipuri de sarcini de lucru	Oferă Docker EE pentru gestionarea și orchestrarea imaginilor Docker deoarece are integrat API-ul Docker	Kubernetes gestionează imaginile Docker dar și containere independente și oferă un set larg de caracteristici	OpenShift acceptă toate tipurile de sarcini de lucru deoarece sunt incluse în Kubernetes
Actualizări	Sunt lansate de Docker și Kubernetes	Sunt lansate de Docker și Kubernetes	OpenShift execută schimbările destul de încet după actualizarea lui Kubernetes, Docker și a altor platforme open source
Operațiile de integrare și livrare continuă (CI/CD)	Extensiile și instrumentele externe sunt acceptate, dar acestea trebuie instalate separat	Extensiile și instrumentele externe sunt acceptate, dar acestea trebuie instalate separat	OpenShift are deja integrat Jenkins și este mai ușor de configurat un sistem CI/CD
Activare multi-cloud	Este acceptat de cloud-uri publice: Google Cloud, EKA pe AWS și AKS pe Azure	Este acceptat de cloud-uri publice: Google Cloud, EKA pe AWS și AKS pe Azure, dar Kubernetes este o alegere prioritară pentru livrările multi-cloud	OpenShift este disponibil pe o platformă dedicată și pe Azure
Livrare și management	Livrarea și gestionarea containerelor Docker este destul de simplă	Livrarea și gestionarea containerelor Kubernetes poate fi foarte complexă	Livrarea și gestionarea containerelor OpenShift este destul de simplă

Din cele prezentate putem extrage câteva concluzii și anume [29]:

- Docker, Kubernetes și OpenShift sunt tehnologii bazate pe containere și se completează unele pe altele;
- Kubernetes poate livra și orchestra imagini Docker;
- OpenShift gestionează fără probleme clusterul Kubernetes;

- Kubernetes permite scalarea automată și are un sprijin comunitar destul de puternic;
- OpenShift poate fi privit ca și o îmbunătățire a lui Kubernetes oferind o simplitate în gestionarea sarcinilor de lucru.

Capitolul 5

Dezvoltarea aplicației Renty

5.1 Etapa de analiză

5.1.1 Descriere funcțională

Timpurile în care pentru a putea face o rezervare care necesită mult timp și multe resurse încep să devină istorie. La început pentru a putea face o rezervare, trebuia să te deplasezi până la locul respectiv și să completezi câteva documente. Odată cu apariția telefoanelor, rezervările au început să se facă cu ajutorul acestora printr-un simplu apel. Mai apoi, au apărut website-urile pentru rezervări unde în câteva secunde se poate face o rezervare fără prea mult efort și resurse.

Renty este o aplicație web care poate fi accesată la adresa **<http://renty.go.ro>**. Scopul acestei aplicații este de a facilita procesul de rezervare a unui loc de recreere pentru desfășurarea de activități sportive. Confirmarea rezervării se face doar după plata cu cardul a sumei corespunzătoare locului și activității alese.

Această aplicație este dedicată în special proprietarilor privați din toată România unde aceștia își pot face cunoscută baza sportivă, dar și oamenilor ce doresc să se recreeze practicând un anumit sport pe un teren special amenajat.

5.1.2 Cazuri de utilizare

Definirea cazurilor de utilizare este un prim pas înainte de gândirea arhitecturii proiectului. În orice dezvoltare soft, clientul vine cu problema identificată de el, iar echipa de dezvoltare va parcurge etapele necesare dezvoltării cerințelor funcționale și non-funcționale.

Pentru aplicația Renty, primele cazuri de utilizare identificate sunt prezentate în Figura 5.1 și vizează utilizatorii neautentificați (guest).

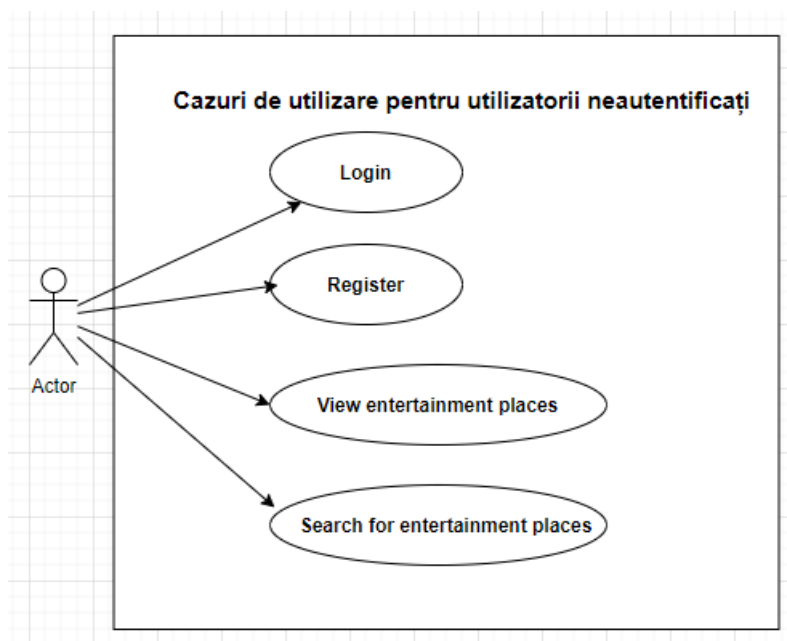


Figura 5.1: Cazuri de utilizare pentru utilizatorii neautentificați

Pentru cazul de utilizare **Login**:

- **actori:** utilizatorii cu rol de "RENTER", "OWNER", "ADMIN" sau "GUEST"
- **descriere:** utilizatorii (în funcție de roluri), primesc acces pe anumite resurse din aplicație după logare
- **precondiții:** utilizatorii navighează pe pagina de Login
- **postcondiții:** utilizatorii primesc acces la resursele oferite de aplicație
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Deschidere pagină de Login	
2.		Încărcare pagină Login
3.	Introducere credențiale	
4.		Verificare autenticitate și redirecționare

Pentru cazul de utilizare **Register**:

- **actori:** utilizatorii cu rol de "RENTER", "OWNER", "ADMIN" sau "GUEST"
- **descriere:** utilizatorii își pot crea cont de "RENTER" sau "OWNER"

- **precondiții:** utilizatorii navighează pe pagina de Login
- **postcondiții:** utilizatorii au un cont nou creat
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Deschidere pagină Login	
2.		Încărcare pagină Login
3.	Introducere credențiale și date de contact	
4.		Creeare cont sau atenționare dacă există deja unul

Pentru cazul de utilizare **View entertainment places:**

- **actori:** utilizatorii cu rol de "RENTER", "OWNER", "ADMIN" sau "GUEST"
- **descriere:** utilizatorii pot vedea bazele sportive disponibile
- **precondiții:** utilizatorii sunt pe pagina principală
- **postcondiții:** utilizatorii au văzut bazele sportive disponibile
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Navigare pe pagina principală	
2.		Încărcare baze sportive disponibile
3.	Vizualizare baze sportive	

Pentru cazul de utilizare **Search for entertainment places:**

- **actori:** utilizatorii cu rol de "RENTER", "OWNER", "ADMIN" sau "GUEST"
- **descriere:** utilizatorii pot căuta bazele sportive după preferințe
- **precondiții:** utilizatorii sunt pe pagina principală
- **postcondiții:** utilizatorii au văzut bazele sportive care îndeplinesc criteriul de căutare

• flux:

Nr.	Acțiune utilizator	Răspuns sistem
1.	Navigare pe pagina principală	
2.		Încărcare baze sportive disponibile
3.	Vizualizare baze sportive	
4.	Căutare baze sportive după criterii alese	
5.		Filtrare baze sportive
6.	Vizualizare baze sportive filtrate după criteriile alese	

Figura 5.2 prezintă cazurile de utilizare pentru utilizatorii cu rol de "RENTER".

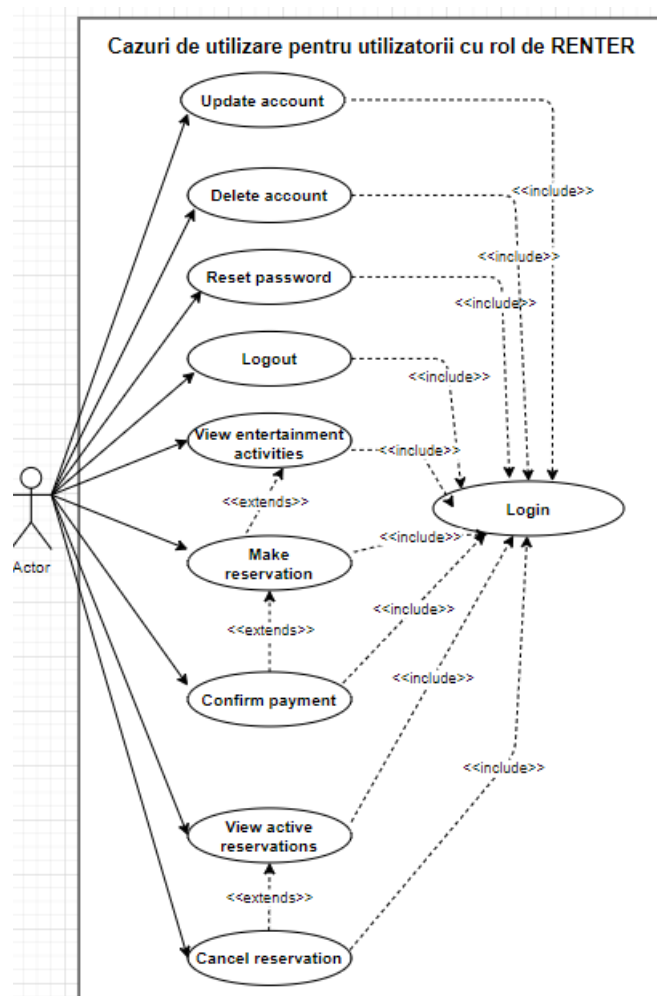


Figura 5.2: Cazuri de utilizare pentru utilizatorii cu rol de "RENTER"

Pentru cazul de utilizare **Update account**:

- **actori**: utilizatorii cu rol de "RENTER", "OWNER", "ADMIN"
- **descriere**: utilizatorii își pot modifica datele de contact
- **precondiții**: utilizatorii navighează pe pagina de Login (după ce s-au logat)
- **postcondiții**: utilizatorii au un cont modificat
- **flux**:

Nr.	Acțiune utilizator	Răspuns sistem
1.	Deschidere pagină Login	
2.		Încărcare pagină Login
3.	Navigare secțiune Update account	
4.		Încărcare pagină Update account
5.	Utilizatorul își modifică datele și salvează	
6.		Server-ul salvează modificările. Se face redirectionarea pe pagina principală.

Pentru cazul de utilizare **Reset password**:

- **actori**: utilizatorii cu rol de "RENTER", "OWNER", "ADMIN"
- **descriere**: utilizatorii își pot modifica parola
- **precondiții**: utilizatorii navighează pe pagina de Login (după ce s-au logat)
- **postcondiții**: utilizatorii au parola modificată
- **flux**:

Nr.	Acțiune utilizator	Răspuns sistem
1.	Deschidere pagină Login	
2.		Încărcare pagină Login
3.	Navigare secțiune Reset password	
4.		Încărcare pagină Reset password
5.	Utilizatorul își schimbă parola și salvează	
6.		Server-ul salvează modificările.

Pentru cazul de utilizare **Delete account**:

- **actori:** utilizatorii cu rol de "RENTER", "OWNER", "ADMIN"
- **descriere:** utilizatorii își pot șterge contul
- **precondiții:** utilizatorii logați navighează pe pagina de Login
- **postcondiții:** utilizatorii au contul șters
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Deschidere pagină Login	
2.		Încărcare pagină Login
3.	Apasă buton Delete account	
4.		Serverul șterge contul și redirecționează pe pagina principală.

Pentru cazul de utilizare **Logout**:

- **actori:** utilizatorii cu rol de "RENTER", "OWNER", "ADMIN"
- **descriere:** utilizatorii logați se pot deloga de la aplicație
- **precondiții:** -
- **postcondiții:** utilizatorii au fost delogați

- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe butonul de Logout	
2.		Ștergere token și redirectionare pe pagina principală

Pentru cazul de utilizare **View entertainment activities**:

- **actori:** utilizatorii cu rol de "RENTER", "OWNER", "ADMIN"
- **descriere:** utilizatorii pot vedea ce activități se pot desfășura la o anumită bază sportivă
- **precondiții:** utilizatorii logați sunt pe pagina principală și au dat click pe una din bazele sportive
- **postcondiții:** utilizatorii au văzut ce activități se pot desfășura la baza sportivă aleasă
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe butonul de See activities	
2.		Căutare activități corespunzătoare bazei sportive alese și navigare pe pagina corespunzătoare activităților
3.	Utilizatorii pot vedea activitățile	

Pentru cazul de utilizare **Make reservation**:

- **actori:** utilizatorii cu rol de "RENTER", "OWNER", "ADMIN"
- **descriere:** utilizatorii logați pot rezerva terenul corespunzător activității sportive alese
- **precondiții:** utilizatorii au dat click pe butonul de Book online și s-a deschis pagina cu orarul rezervărilor

- **postcondiții:** utilizatorii au văzut orarul rezervărilor

- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe butonul de Book online	
2.		Sistemul afișează orarul rezervărilor
3.	Utilizatorii pot vedea orarul rezervărilor	

Pentru cazul de utilizare **Confirm payment:**

- **actori:** utilizatorii cu rol de "RENTER", "OWNER", "ADMIN"
- **descriere:** utilizatorii pot efectua plata pentru confirmarea rezervării
- **precondiții:** utilizatorii logați au ales data și ora disponibilă și au dat click pe pătrățelul corespunzător
- **postcondiții:** utilizatorii au efectuat plata și au confirmat rezervarea
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe pătrățelul corespunzător datei și orei disponibile alese	
2.		Sistemul identifică data și ora și deschide fereastra pentru introducerea datelor de pe card
3.	Utilizatorii introduc datele cardului și apasă butonul Confirm	
4.		Sistemul verifică autenticitatea cardului și efectuează plata și redirecționează spre orarul rezervărilor
5.	Utilizatorii pot vedea rezervarea marcată cu gri pe pătrățelul corespunzător datei și orei alese	

Pentru cazul de utilizare **View active reservations**:

- **actori**: utilizatorii cu rol de "RENTER", "OWNER", "ADMIN"
- **descriere**: utilizatorii pot vedea rezervările active
- **precondiții**: utilizatorii logați au navigat pe modulul My reservations
- **postcondiții**: utilizatorii au văzut rezervările active
- **flux**:

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe modulul My reservations	
2.		Sistemul afișează rezervările active corespunzătoare utilizatorului logat
3.	Utilizatorii văd propriile rezervări active	

Pentru cazul de utilizare **Cancel reservation**:

- **actori**: utilizatorii cu rol de "RENTER", "OWNER", "ADMIN"
- **descriere**: utilizatorii pot șterge rezervările active
- **precondiții**: utilizatorii logați au navigat pe modulul My reservations
- **postcondiții**: utilizatorii au șters rezervări active
- **flux**:

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe modulul My reservations	
2.		Sistemul afișează rezervările active corespunzătoare utilizatorului logat
3.	Utilizatorii văd propriile rezervări active	
4.		Sistemul afișează un buton de ștergere în dreptul fiecărei rezervări
5.	Click pe butonul de ștergere	
6.		Sistemul șterge rezervarea și afișează rezervările active rămase
7.	Utilizatorii văd rezervările active rămase	

Pe lângă cazurile de utilizare specifice utilizatorilor (prezentate în Figura 5.3) cu rol de "OWNER", aceștia mai pot face și ceea ce fac utilizatorii doar cu rol de "RENTER". Utilizatorii cu rol de "ADMIN" văd toate bazele sportive (deoarece aceștia nu i se permite să-și creeze baze sportive) și nu are dreptul de a șterge sau modifica bazele sportive și activitățile corespunzătoare.

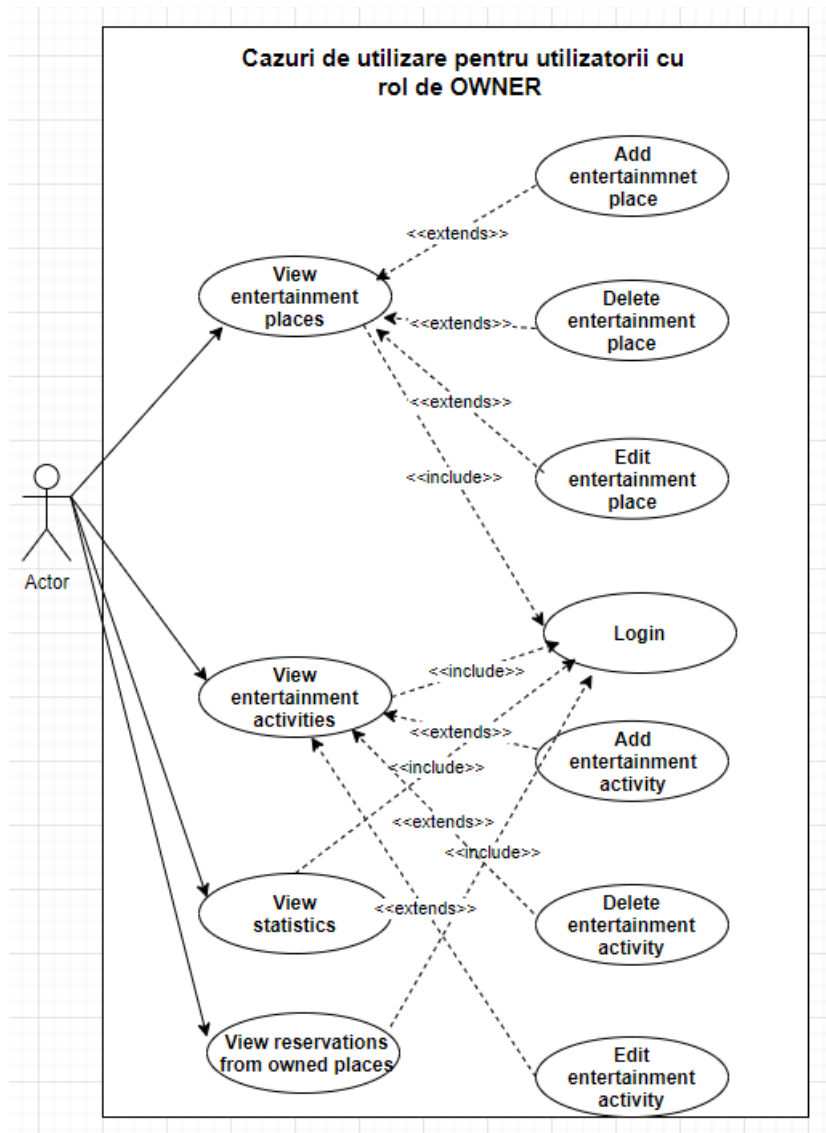


Figura 5.3: Cazuri de utilizare pentru utilizatorii cu rol de "OWNER"

Pentru cazul de utilizare **View entertainment places**:

- **actori:** utilizatorii cu rol de "OWNER", "ADMIN"
- **descriere:** utilizatorii pot vedea bazele sportive
- **precondiții:** utilizatorii logați au navigat pe modulul My places
- **postcondiții:** utilizatorii au văzut propriile baze sportive
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe modulul My places	
2.		Sistemul afișează bazele sportive corespunzătoare propriilor baze sportive
3.	Utilizatorii văd propriile baze sportive	

Pentru cazurile de utilizare **Add entertainment place**, **Delete entertainment place**, **Edit entertainment place**:

- **actori:** utilizatorii cu rol de "OWNER"
- **descriere:** utilizatorii pot vedea bazele sportive
- **precondiții:** utilizatorii logați au navigat pe modulul My places
- **postcondiții:** utilizatorii au văzut propriile baze sportive modificate
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe modulul My places	
2.		Sistemul afișează bazele sportive corespunzătoare utilizatorului logat
3.	Utilizatorii văd propriile baze sportive	
4.		Sistemul afișează o secțiune de adăugare bază sportivă și butoane de ștergere și editare a bazelor sportive curente.
5.	Utilizatorii apasă butonul de Save pentru a crea o bază sportivă, respectiv Delete forever pentru a șterge o bază sportivă și Edit pentru a edita o bază sportivă	
6.		Sistemul identifică acțiunea aleasă, face modificările și le prezintă.
7.	Utilizatorii văd schimbările efectuate.	

Pentru cazul de utilizare **View entertainment activities**:

- **actori**: utilizatorii cu rol de "OWNER", "ADMIN"
- **descriere**: utilizatorii pot vedea activitățile de la o bază sportivă
- **precondiții**: utilizatorii logați au navigat pe modulul My places și au dat click pe butonul de See activities
- **postcondiții**: utilizatorii au văzut activitățile disponibile
- **flux**:

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe modulul My places	
2.		Sistemul afișează bazele sportive corespunzătoare utilizatorului logat
3.	Utilizatorii văd propriile baze sportive și dau click pe butonul See activities	
4.		Sistemul prezintă activitățile disponibile de la baza sportivă aleasă
5.	Utilizatorii văd activitățile disponibile	

Pentru cazurile de utilizare **Add entertainment activity**, **Delete entertainment activity**, **Edit entertainment activity**:

- **actori**: utilizatorii cu rol de "OWNER"
- **descriere**: utilizatorii pot vedea activitățile disponibile
- **precondiții**: utilizatorii logați au navigat pe modulul My places și au dat click pe butonul See activities de la o bază sportivă
- **postcondiții**: utilizatorii au văzut propriile baze sportive modificate
- **flux**:

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe modulul My places	
2.		Sistemul afișează bazele sportive corespunzătoare utilizatorului logat
3.	Utilizatorii văd propriile baze sportive și dau click pe butonul See activities de la o bază sportivă	
4.		Sistemul afișează o secțiune de adăugare activitate și butoane de ștergere și editare a activităților curente.
5.	Utilizatorii apasă butonul de Save pentru a crea o bază sportivă, respectiv Delete forever pentru a șterge o activitate și Edit pentru a edita o activitate	
6.		Sistemul identifică acțiunea aleasă, face modificările și le prezintă.
7.	Utilizatorii văd schimbările efectuate.	

Pentru cazul de utilizare **View statistics**:

- **actori:** utilizatorii cu rol de "OWNER", "ADMIN"
- **descriere:** utilizatorii pot vedea statistici
- **precondiții:** utilizatorii logați au navigat pe modulul Statistics și au dat pe unul dintre butoanele de Search
- **postcondiții:** utilizatorii au văzut statisticile
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe modulul Statistics	
2.		Sistemul afișează criteriile de generare statistici
3.	Utilizatorii văd criteriile de generare și dau click pe unul dintre butoanele de Search	
4.		Sistemul prezintă statisticile pentru criteriul ales
5.	Utilizatorii văd statisticile generate	

Pentru cazul de utilizare **View reservations from owned places**:

- **actori:** utilizatorii cu rol de "OWNER", "ADMIN"
- **descriere:** utilizatorii pot vedea rezervările active făcute pentru bazele sportive proprii
- **precondiții:** utilizatorii logați au navigat pe modulul Reservations
- **postcondiții:** utilizatorii au văzut rezervările active
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe modulul Reservations	
2.		Sistemul afișează rezervările active
3.	Utilizatorii văd rezervările active	

Cazurile de utilizare specifice utilizatorilor cu rol de "ADMIN" (prezentate în Figura 5.4), aceștia mai pot face și ce fac utilizatorii cu rol de "RENTER" sau "OWNER", dar cu o singură excepție. Administratorul nu poate modifica datele bazelor sportive și a activităților corespunzătoare, precum nici să creeze altele, iar statisticile îi sunt prezentate ca un cumul peste toate bazele și activitățile sportive.

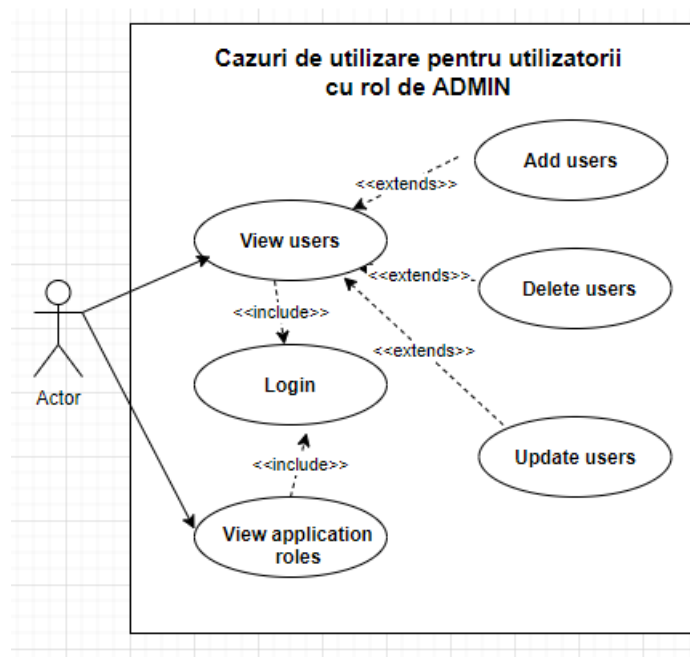


Figura 5.4: Cazuri de utilizare pentru utilizatorii cu rol de "ADMIN"

Pentru cazul de utilizare **View users**:

- **actori:** utilizatorii cu rol de "ADMIN"
- **descriere:** utilizatorii pot vedea utilizatorii din sistem
- **precondiții:** utilizatorii logați au navigat pe modulul Users
- **postcondiții:** utilizatorii au văzut utilizatorii de pe sistem
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe modulul Users	
2.		Sistemul afișează utilizatorii de pe sistem
3.	Utilizatorii văd utilizatorii de pe sistem	

Pentru cazul de utilizare **View application roles**:

- **actori:** utilizatorii cu rol de "ADMIN"
- **descriere:** utilizatorii pot vedea rolurile de pe sistem
- **precondiții:** utilizatorii logați au navigat pe modulul Roles

- **postcondiții:** utilizatorii au văzut rolurile de pe sistem
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe modulul Roles	
2.		Sistemul afișează rolurile de pe sistem
3.	Utilizatorii văd rolurile de pe sistem	

Pentru cazurile de utilizare **Add users, Delete users, Update users:**

- **actori:** utilizatorii cu rol de "ADMIN"
- **descriere:** utilizatorii pot crea, modifica și șterge useri
- **precondiții:** utilizatorii logați au navigat pe modulul Users
- **postcondiții:** utilizatorii au văzut modificările efectuate
- **flux:**

Nr.	Acțiune utilizator	Răspuns sistem
1.	Click pe modulul Users	
2.		Sistemul afișează utilizatorii de pe sistem precum și butoane de ștergere, editare sau adăugare utilizatori
3.	Utilizatorii văd utilizatorii de pe sistem și butoane de adăugare, editare și ștergere și dau click pe unul dintre aceste butoane	
4.		Sistemul identifică acțiunea și efectuează schimbările sau deschide o secțiune de adăugare utilizator
5.	Utilizatorul apasă butonul de Save	
6.		Sistemul identifică acțiunea aleasă și prezintă modificările
7.	Utilizatorii văd schimbările efectuate	

Pentru a sumariza cazurile de utilizare, le vom pune într-un tabel unde printr-o “✓” sau un “X” se va indica dacă cazul de utilizare este disponibil sau nu pentru diferitele tipuri de utilizatori.

	"GUEST"	"RENTER"	"OWNER"	"ADMIN"
Login	✓	✓	✓	✓
Register	✓	✓	✓	✓
View entertainment places	✓	✓	✓	✓
Search for entertainment places	✓	✓	✓	✓
Update account	✗	✓	✓	✓
Reset password	✗	✓	✓	✓
Delete account	✗	✓	✓	✓
Logout	✗	✓	✓	✓
Make reservation	✗	✓	✓	✓
Confirm payment	✗	✓	✓	✓
View active reservation	✗	✓	✓	✓
Cancel reservation	✗	✓	✓	✓
View entertainment places	✗	✗	✓	✓
Add/Delete/Update entertainment places	✗	✗	✓	✗
View entertainment activities	✗	✗	✓	✓
Add/Delete/Update entertainment activities	✗	✗	✓	✗
View statistics	✗	✗	✓	✓
View reservations from owned places	✗	✗	✓	✗
View users	✗	✗	✗	✓
View application roles	✗	✗	✗	✓
Add/Delete/Update users	✗	✗	✗	✓

5.2 Etapa de proiectare

În orice dezvoltare de soft, după definirea cazurilor de utilizare și înainte de a scrie codul sursă este necesar să existe o viziune de ansamblu asupra aplicației ce urmează a fi dezvoltată. Astfel, în subsecțiunile următoare va fi prezentat arhitectura aplicației.

5.2.1 Arhitectura aplicației

Aplicația Renty este dezvoltată pe modelul client - server. Astfel, clientul, care este dezvoltat în framework-ul Angular 9, trimite cereri http către REST API scris în Java folosind framework-ul Spring și interoghează baza de date. Detalii despre aceste două framework-uri vor fi prezentate în secțiunea următoare. Astfel, arhitectura aplicației Renty este ca în Figura 5.5.

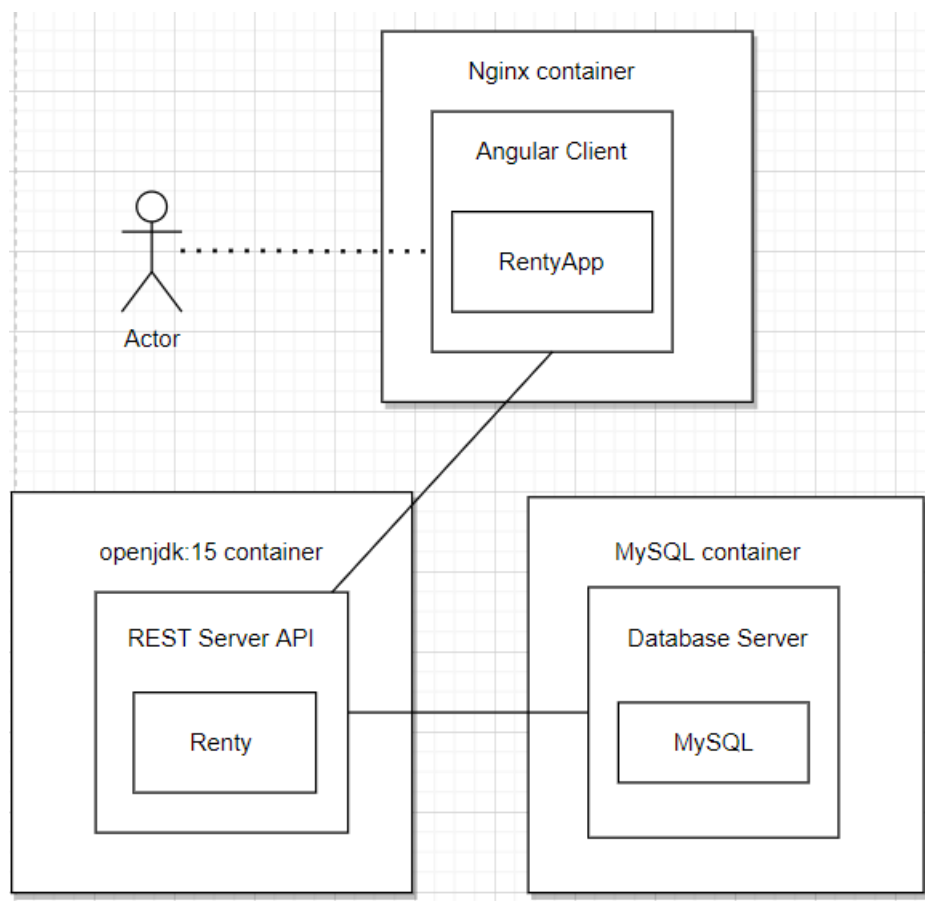


Figura 5.5: Arhitectura aplicației Renty

5.2.2 Structura bazei de date

Deoarece baza de date a fost creată cu ajutorul framework-ului Hibernate, aceasta are o structură asemănătoare cu clasele din domeniul aplicației aflat pe nivelul Data

Access Object (DAO). Hibernate este un framework Java care se ocupă cu maparea claselor din domeniul aplicației la o bază de date relațională. Așadar, structura bazei de date este ca în Figura 5.6.

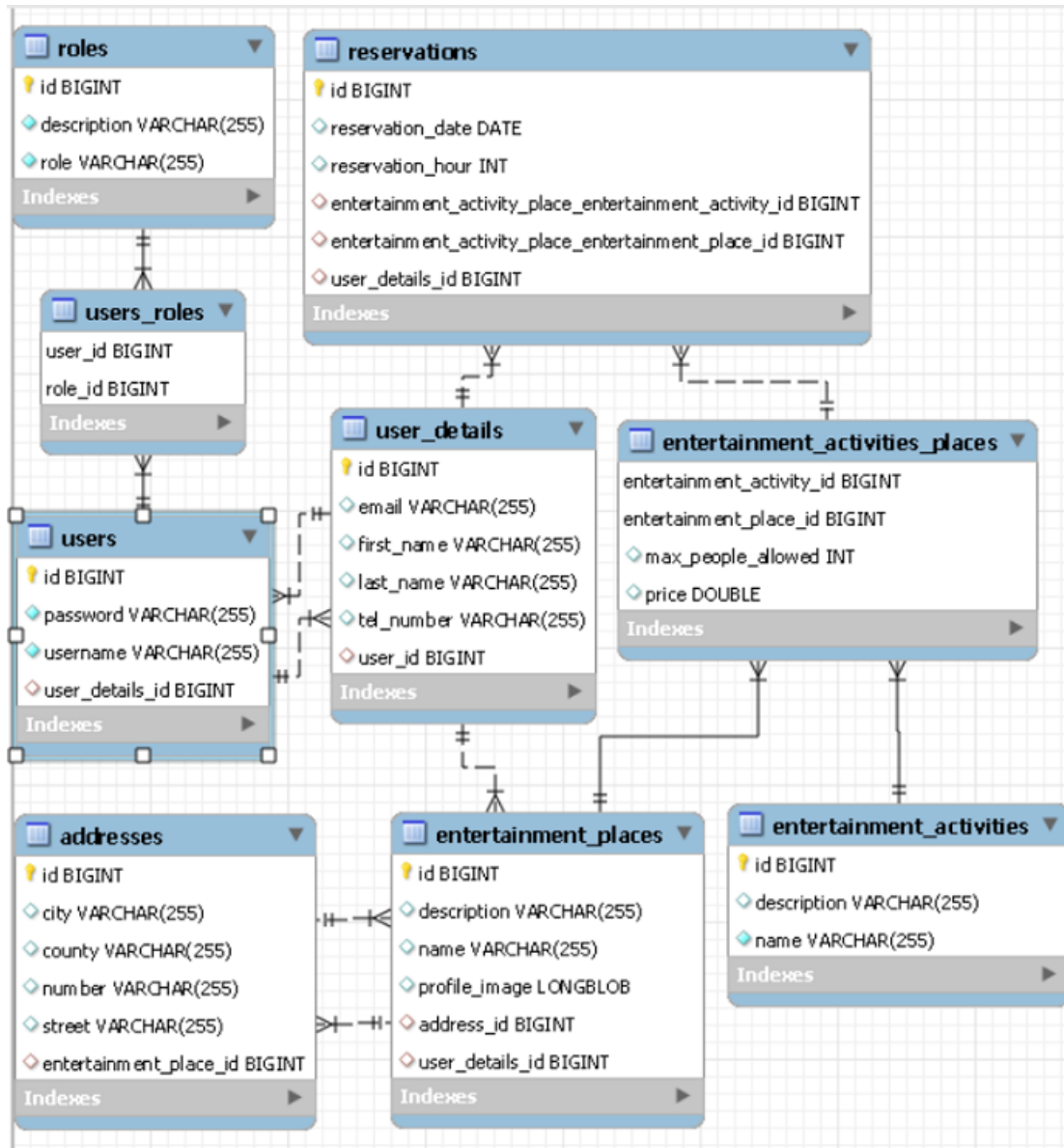


Figura 5.6: Schema bazei de date

După cum se poate observa, baza de date se află în a 3-a formă normală. Acest lucru se poate deduce din tabelul *entertainment_activities_places* deoarece toate atributele depind de o cheie în întregime, unde cheia primară este compusă din două chei străine și anume *entertainment_activity_id* și *entertainment_place_id*.

5.2.3 Structura proiectului și diagrame de clase

În orice dezvoltare soft structura codului este foarte importantă. O arhitectură straticată pe mai multe nivele este necesară și în același timp utilă din mai multe puncte

de vedere cum ar fi: păstrarea mentenanței aplicației și introducerea pe proiect și a altor programatori. Structura proiectului a unei aplicații web pe partea de back-end (care prestează servicii REST) ar trebui să respecte arhitectura descrisă în Figura 5.7.

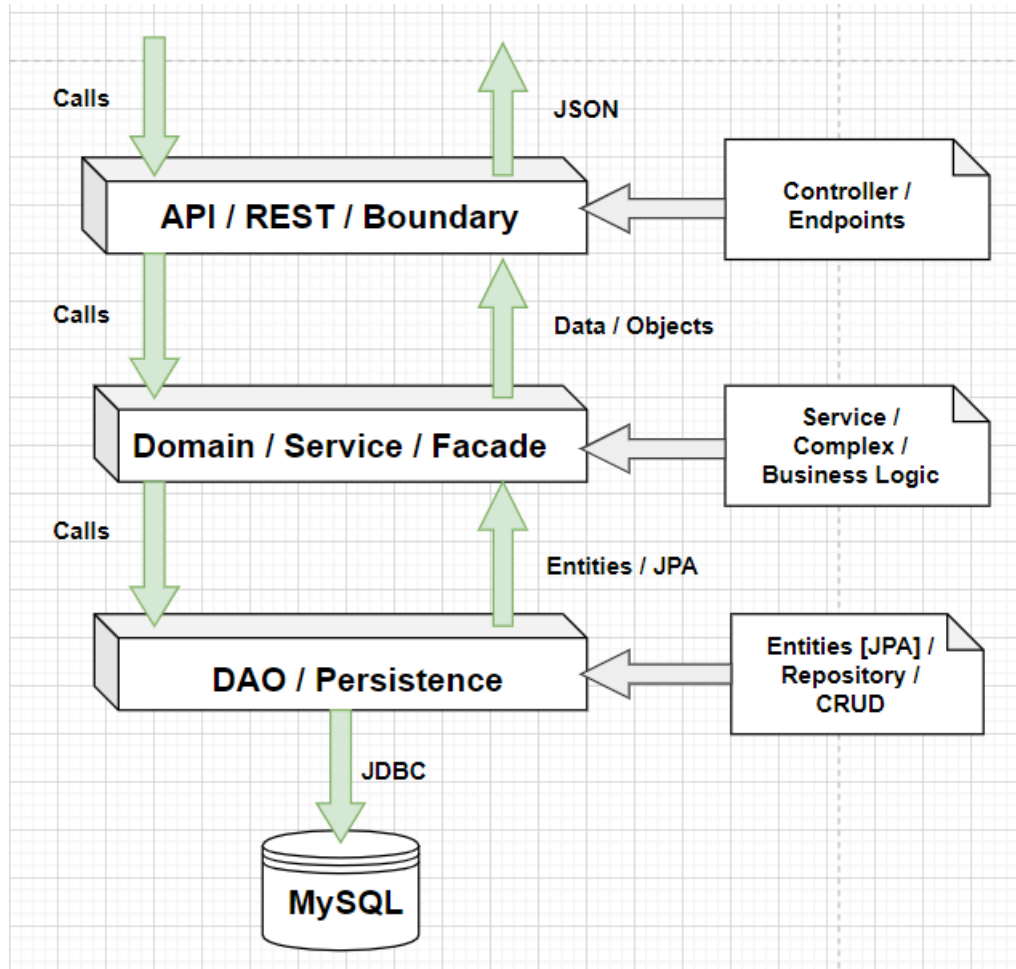


Figura 5.7: Arhitectura proiectului

Înainte de a începe efectiv implementarea funcționalităților, o definire a relațiilor dintre clasele de pe nivelul DAO care mapează tabelele din baza de date MySQL este necesară. Aceste relații trebuie stabilite chiar de la început pentru că altfel, în procesul de dezvoltare, vor apărea schimbări majore în cod. Pentru proiectul Renty, relațiile dintre clasele din domeniul aplicației sunt prezentate în Figura 5.8.

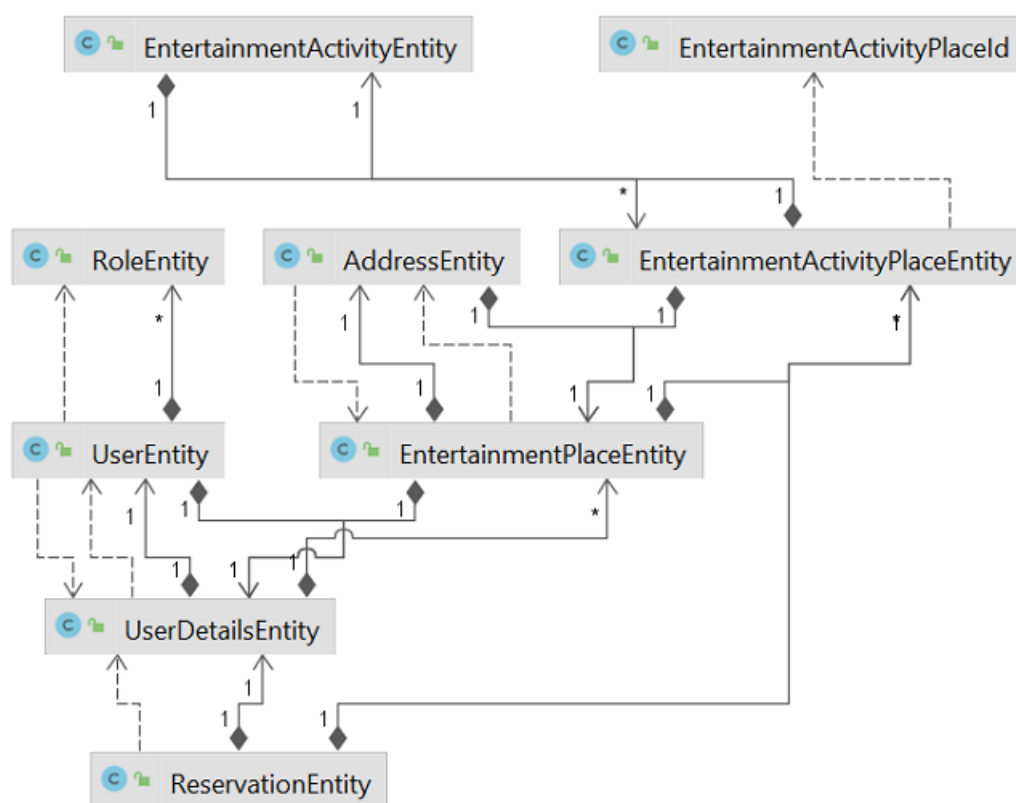


Figura 5.8: Relațiile dintre clasele din domeniul aplicației de pe nivelul DAO

5.3 Tehnologii și framework-uri utilizate

Pentru partea de back-end, s-au folosit următoarele tehnologii:

- **MySQL** este un sistem de gestionare a bazelor de date relaționale având un model client-server. MySQL, în comparație cu alte sisteme de gestionare a bazelor de date relaționale, este disponibil pe o gamă largă de sisteme de operare și totodată la multe versiuni se găsește codul sursă pe Internet care se poate modifica după bunul plac [37].
- **Java** este un limbaj de programare orientat pe obiect conceput în anul 1990 de către James Gosling și echipa lui, iar în 1992 s-a făcut public prima demonstrație funcțională. Programele scrise în Java rulează într-o mașină virtuală numită JVM [14].
- **Spring** este un framework Java ce oferă suport pentru dezvoltarea aplicațiilor Java și include preimplementări cum ar fi: *Java Database Conectivity* (JDBC), *Security*, *Test*, *Object relational mapping* (ORM), dar mai sunt și altele. **Spring Boot** este o extensie a lui Spring, dar care prezintă avantaje mari cum ar fi incorporarea unui server pe care să ruleze aplicația și multe configurări automate [10].

- **Mockito** este un framework Java care împreună cu JUnit se folosește la teste unitare. Mockito se folosește pentru a oferi o implementare fictivă a interfețelor folosite în testele unitare [23].

Pentru partea de front-end, s-au folosit următoarele tehnologii:

- **Typescript** este un limbaj de programare open-source și poate fi compilat la cod JavaScript. TypeScript este o extensie a limbajului JavaScript și are mai multe avantaje printre care și faptul că acesta poate avea tipuri de date, clase și interfețe [26].
- **Angular 9** este un framework open-source pentru aplicațiile web și mobile bazat pe TypeScript care a fost dezvoltat de Microsoft. Este bazat pe componente, iar partea de HTML și TypeScript pot comunica bidirecțional [31].

Pe partea de server (ca și “gazdă”) s-au folosit următoarele utilitare:

- **Docker** care este prezentat mai în detaliu în secțiunile următoare.
- **Nginx** instalat într-un Docker, este un server HTTP (care poate fi configurat ca proxy server și reverse proxy), recunoscut pentru simplitatea configurării și performanța acestuia [1].
- **JDK** (sau Java Development Kit) instalat de asemenea într-un Docker, este un mediu de dezvoltare a aplicațiilor java și include JRE (sau Java Runtime Environment), un interpretor (Java), un compilator (javac), un arhivator (jar), precum și alte instrumente și utilități folosite pentru dezvoltarea softului în Java [11].
- **Jenkins** (fiind instalat într-un Docker) este un utilitar (dezvoltat în Java având la bază server-ul Tomcat) folosit pentru o integrare continuă și livrare a codului pentru aproape orice limbaj, precum și a automatiza alte sarcini de dezvoltare soft. Când mai multe persoane lucrează pe un proiect, dacă folosesc Jenkins, e mult mai ușor de a integra fiecare parte de cod, fără a avea foarte multe conflicte, și de a dispune întotdeauna de ultima versiune a aplicației. Înainte, acest proces de livrare a codului era foarte greoi din cauză că apăreau foarte multe conflicte consumatoare de timp util. Prin urmare, Kohsuke Kawaguchi developer la compania Sun, a venit cu o soluție, și anume Jenkins (care înainte purta numele de Hudson). În 2008, Sun observă calitatea acestui soft, și de atunci, acesta continuă să evolueze [35].
- **SonarQube** este un utilitar folosit pentru inspecția și analiza codului automat. Acesta generează un raport care apoi este folosit ca și ghid de echipa de

dezvoltare soft pentru repararea problemelor. A apărut în 2007 (inițial cu numele de Sonar) și de atunci continuă să evolueze [16]. De precizat este că în zilele de astăzi, Sonar este un plugin pentru analiza codului aflat pe mașina de development, iar SonarQube este un server, unde pot fi analizate mai multe proiecte.

5.4 Etapa de implementare

5.4.1 Securitatea aplicației

Pentru gestionarea aspectelor legate de securitate din cadrul aplicației, a fost utilizat framework-ul Spring Security. Acesta este un framework foarte puternic care oferă aplicației o autentificare și autorizare personalizată. Pentru a securiza comunicarea dintre server și client am integrat JWT. În cele ce urmează o să prezint pașii de configurare.

Pasul 1: Definirea clasei UserEntity și a clasei RoleEntity

Clasa UserEntity trebuie să implementeze interfața UserDetails. Această interfață provine din miezul lui Spring Security și este folosită pentru a identifica utilizatorii, precum și alte atribute adiționale cum ar fi: verificarea contului activ, a contului blocat, cont expirat sau credențiale expirate. Clasa UserEntity arată ca în Figura 5.9.

```
@Table(name = "users")
public class UserEntity implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(nullable = false, unique = true, updatable = false)
    private Long id;

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    private String password;

    @ManyToMany(targetEntity = RoleEntity.class, fetch = FetchType.EAGER)
    @JoinTable(name = "users_roles",
        joinColumns = @JoinColumn(name = "user_id", referencedColumnName = "id"),
        inverseJoinColumns = @JoinColumn(name = "role_id", referencedColumnName = "id"))
    private Set<RoleEntity> authorities = new HashSet<>();

    @OneToOne(targetEntity = UserDetailsEntity.class, fetch = FetchType.EAGER)
    private UserDetailsEntity userDetails;

    @Override
    public Set<RoleEntity> getAuthorities() { return this.authorities; }
```

Figura 5.9: Clasa UserEntity

După cum se poate observa din Figura 5.9, clasa UserEntity mai are un atribut și anume un set de RoleEntity, fiind o relație de *many-to-many*. Acest lucru a fost necesar deoarece aplicația expune resurse condiționate de rolurile deținute de utilizator.

Clasa RoleEntity implementează interfața GrantedAuthorities care provine, de asemenea, din miezul lui Spring Security și este folosită pentru a identifica permisiuni în aplicație. Clasa RoleEntity arată ca în Figura 5.10.


```
@Table(name = "roles")
public class RoleEntity implements GrantedAuthority {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false, unique = true, updatable = true)
    private Long id;

    @Column(name = "role", unique = true, nullable = false)
    private String role;

    @Column(name = "description", nullable = false)
    private String description;

    @Transient
    @Override
    public String getAuthority() { return this.role; }
}
```

Figura 5.10: Clasa RoleEntity

Pasul 2: Definirea rolurilor și a permisiunilor

După definirea claselor corespunzătoare entităților utilizator și rol, urmează definirea propriu-zisă a rolurilor și permisiunilor. Permisuniile pot fi identificate în Figura 5.11, iar rolurile în Figura 5.12.

```
@Getter
public enum ApplicationUserPermission {
    OWNER_READ("place:read"),
    OWNER_WRITE("place:write"),
    ADMIN_READ("admin:read"),
    ADMIN_WRITE("admin:write"),
    RENTER_READ("renter:read"),
    RENTER_WRITE("renter:write");
}
```

Figura 5.11: Permisuniile utilizatorilor

```
@Getter
public enum ApplicationUserRole {
    RENTER(Sets.newHashSet(ApplicationUserPermission.RENTER_READ,
        ApplicationUserPermission.RENTER_WRITE)),
    OWNER(Sets.newHashSet(ApplicationUserPermission.OWNER_READ,
        ApplicationUserPermission.OWNER_WRITE,
        ApplicationUserPermission.RENTER_READ,
        ApplicationUserPermission.RENTER_WRITE)),
    ADMIN(Sets.newHashSet(ApplicationUserPermission.ADMIN_READ,
        ApplicationUserPermission.ADMIN_WRITE,
        ApplicationUserPermission.OWNER_WRITE,
        ApplicationUserPermission.OWNER_READ,
        ApplicationUserPermission.RENTER_READ,
        ApplicationUserPermission.RENTER_WRITE));
}
```

Figura 5.12: Rolurile utilizatorilor

Pasul 3: Definirea filtrelor

Pentru a accesa unele resurse de pe server, este necesar ca cererile să treacă prin anumite filtre de securitate. De exemplu, pentru logarea la aplicație este necesară verificarea în baza de date a utilizatorului, iar pentru a accesa unele resurse care necesită drepturi, trebuie să se verifice validitatea token-ului generat.

Pasul 4: Definirea clasei unde se va prelucra aspectele referitoare la securitate

Pentru ca această clasă să fie recunoscută de Spring Security ca fiind clasa principală de configurare, aceasta trebuie să extindă clasa `WebSecurityConfigurerAdapter`. Metoda care trebuie suprascrisă este cea de configurare unde se adaugă toate constrângerile pentru ca cererea să fie procesată mai departe pe server. Acest lucru se poate observa în Figura 5.13.

```

@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .cors().and() HttpSecurity
        .csrf().disable()
        .sessionManagement() SessionManagementConfigurer<HttpSecurity>
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and() HttpSecurity
        .addFilter(
            new JwtUsernameAndPasswordAuthenticationFilter
                (authenticationManager(), jwtConfig, secretKey, userService))
        .addFilterAfter(
            new JwtTokenVerifierFilter(jwtConfig, secretKey),
            JwtUsernameAndPasswordAuthenticationFilter.class)
        .authorizeRequests() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptl
        .antMatchers( ...antPatterns: "/admin/**", "/owner/**",
            "/renter/**", "/anon/**") ExpressionUrlAuthorizationConfigurer<...>.A
            .hasAnyAuthority(ApplicationUserRole.ADMIN.name()) Expres
        .antMatchers( ...antPatterns: "/owner/**", "/renter/**",
            "/anon/**") ExpressionUrlAuthorizationConfigurer<...>.AuthorizedUrl
            .hasAnyAuthority(ApplicationUserRole.OWNER.name(),
                ApplicationUserRole.ADMIN.name()) ExpressionUrl
        .antMatchers( ...antPatterns: "/renter/**", "/anon/**",
            "/payment/**") ExpressionUrlAuthorizationConfigurer<...>.AuthorizedUrl
            .hasAnyAuthority(ApplicationUserRole.RENTER.name(),
                ApplicationUserRole.ADMIN.name(),
                ApplicationUserRole.RENTER.name()) ExpressionUrlAuth
        .antMatchers( ...antPatterns: "/anon/**", "/h2-console/**").anonymous()
        .anyRequest() ExpressionUrlAuthorizationConfigurer<...>.AuthorizedUrl
        .authenticated() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegi
        .and() HttpSecurity
        .logout();

    http
        .headers().frameOptions().disable(); //for h2-console

    http
        .exceptionHandling()
        .authenticationEntryPoint(unauthorizedEntryPointHandler()) //fo
        .accessDeniedHandler(accessDeniedHandler()); //for access denie
}

```

Figura 5.13: Metodă pentru configurarea securității

5.4.2 Stripe API

Stripe este un sistem complex, complet și sigur de plată, simplu de utilizat cu posibilitatea de a opera cu 135 unități de măsură a banilor. Permite transferarea de bani dintr-un cont în altul utilizând un card de credit sau debit. Stripe se poate integra direct, cu ușurință, în platforma de business, fără ca plățile să fie făcute prin utilizarea unui intermediar [28].

Ținând cont că serviciile Stripe sunt bazate pe tokens, pentru a putea efectua o

plată trebuie ca în acel token să fie pasate datele cardului. Fiecare token poate fi folosit o singură dată, iar dacă este interceptat de un hacker, acesta nu poate extrage datele cardului. Desigur că acest token poate fi generat de către dezvoltator, dar nu este la fel de sigur ca cel creat de serviciul Stripe. Pentru a putea integra Stripe în orice aplicație este necesară deținerea unui cont pe website-ul <https://dashboard.stripe.com/> și generarea a două chei (una privată și una publică) utilizate pentru plăți [36].

În cele ce urmează prezentăm integrarea lui Stripe API în aplicația Renty. Astfel, pe partea de front-end, se preiau datele cardului scrise în input texts, se generează token-ul și se trimite la REST server printr-o metodă POST din clasa service corespunzătoare plății. Acest proces este prezentat în Figura 5.14 de mai jos.

```
card: Card;
```

```
confirmReservation() {
  (<any>window).Stripe.setPublishableKey(environment.stripePublishableKey);
  (<any>window).Stripe.card.createToken({
    number: this.card.number,
    exp_month: this.card.exp_month,
    exp_year: this.card.exp_year,
    cvc: this.card.cvc
  }, (status: number, response: any) => {
    if (status === 200) {
      let token = response.id;
      forkJoin([
        this.paymentService.chargeCard(token, this.price),
        this.renterService.createReservation(this.reservation)
      ]).subscribe(([paymentResponse, reservationResponse]) => {
        this.toastr.successToastr("Reservation created successfully!", Message);
        this.dialogRef.close();
      });
    }
  });
}
```

Figura 5.14: Creare Stripe token și trimitere la REST server

Partea de back-end, interceptează token-ul precum și suma (Figura 5.15), dar în unități reprezentând a 100-a parte corespunzătoare sumei reale, adică unitatea de măsură “bani” în cazul RON-ului. Astfel, la crearea obiectului corespunzător plății se înmulțește cu 100 pentru obținerea sumei reale (Figura 5.16). Pentru ca plata să poată fi efectuată este necesară luarea în considerare a cheii private generate.

```

@RestController
@RequestMapping("/payment")
public class PaymentController {
    @Value("${stripe.key.secret}")
    private String stripeKeySecret;

    private final StripeClient stripeClient;

    public PaymentController(StripeClient stripeClient) { this.stripeClient = stripeClient; }

    @PostMapping("/charge")
    private void chargeCard(@RequestHeader(name = "token") String token, @RequestHeader(name = "amount")
        Stripe.apiKey = stripeKeySecret;
        final Double totalAmount = ServicesUtils.convertStringToDouble(amount);
        this.stripeClient.chargeCreditCard(token, totalAmount);
    }
}

```

Figura 5.15: Interceptare Stripe token și sumă

Dacă se dorește ca în viitor să se salveze informațiile referitoare la plata efectuată, aceste informații se pot extrage ușor din obiectul Charge (Figura 5.16).

```

@Component
public class StripeClient {
    public void chargeCreditCard(final String token, final double amount)
        final Map<String, Object> params = new HashMap<>();
        params.put("amount", (int)(amount * 100));
        params.put("currency", "RON");
        params.put("source", token);
        final Charge charge = Charge.create(params);
    }
}

```

Figura 5.16: Plată folosind Stripe

5.4.3 Medii de lucru

Mediile de lucru nu sunt altceva decât niște fișiere de configurare. În orice proiect este bine să existe măcar 3 astfel de medii (dev, test, prod) pentru a nu amesteca, de exemplu, baza de date din test cu cea din prod. În aceste fișiere de configurare, în general se află datele necesare conexiunii la baza de date, dar pot fi și altele precum și proprietăți particulare definite de dezvoltator [32].

Pentru aplicația Renty am folosit 3 medii: dev, test și prod. Pentru mediile dev și test am folosit o bază de date H2, fiind foarte ușor de configurat pe local, iar pentru prod, am folosit o bază de date MySQL. Nu este recomandat a se folosi o bază de date H2 pentru prod deoarece aceasta se distruge la oprirea execuției jar-ului [21].

Un fișier de configurare pentru mediul dev pe partea de back-end este prezentat în Figura 5.17, iar pentru partea de front-end este prezentat în Figura 5.18.

```
spring:  
  datasource:  
    url: jdbc:h2:mem:devdb  
    driver-class-name: org.h2.Driver  
    username: root  
    password: root  
  jpa:  
    database-platform: org.hibernate.dialect.H2Dialect  
  h2:  
    console:  
      enabled: true  
application:  
  jwt:  
    secretKey: RentyBackend;RentyBackend;RentyBackend;RentyBackend;  
    tokenPrefix: Bearer  
    tokenExpirationAfterDays: 1  
stripe:  
  key:  
    secret: sk_test_51IMvgYJcxq1wro07xkeNp1NP0DEWPNSHSLdEh  
    public: pk_test_51IMvgYJcxq1wro07RHpy6puCkeceDfydJfMws
```

Figura 5.17: Fisier de configurare pentru mediul dev (local) pe partea de back-end

```
src > environments >  environmnet.dev.ts > ...
1 export const environment = {
2   serverUrl: "http://localhost:8080",
3   prod: false,
4   name: "dev",
5   stripePublishableKey: "pk_test_51IMvgYJcxq1wro07RHpy6puCk",
6 };

```

Figura 5.18: Fișier de configurare pentru mediul dev (local) pe partea de front-end

5.4.4 Docker

Docker poate fi privit ca o mașină virtuală portabilă, dar mult mai ușor de folosit. Se bazează pe containere, oferind un mediu izolat de dezvoltare. Containerele docker au propriile lor procese, rețele și configurări utilizând același OS Kernel și sunt construite pe imagini docker care pot fi privite ca un sistem de fișiere ce conține toate dependențele și pachetele pentru ca aplicația să funcționeze. Aceste containere nu expun în mod direct adresa și portul pe care rulează, și astfel se poate spune că Docker se poate folosi și pentru a ascunde diferite informații către exterior precum precizează A. Charles [9].

Înglobarea aplicațiilor în containere Docker prezintă câteva beneficii cum ar fi [27]:

- **portabilitate:** containerele care conțin serviciile sistemului de aplicații pot fi ușor de configurat pe orice sistem de operare care are Docker instalat;
- **performanța:** în comparație cu mașinile virtuale, containerele Docker nu conțin un sistem de operare, consumând astfel mai puține resurse;
- **proces de dezvoltare agil:** beneficiile de portabilitate și performanță oferă un proces de dezvoltare mai agil și receptiv;
- **izolare:** containerul Docker conține aplicația precum și versiunile de servicii dependente ei. Dacă se dorește să se folosească alte versiuni ale serviciilor, un alt container Docker poate fi configurat fiind independente între ele;
- **scalabilitate:** se pot crea foarte rapid alte containere dacă sistemul de aplicații are nevoie.

5.5 Etapa de testare

În orice proiect, după implementarea funcționalităților, o testare a codului va ajuta dezvoltatorul să descopere eventualele probleme. Astfel, pentru aplicația Renty, am testat nivelul de business al aplicației făcând teste unitare folosind framework-ul Mockito. Cu ajutorul acestui framework, am simulat o implementare fictivă, în special pentru obiectele ce reprezintă entitățile din domeniul aplicației de pe nivelul DAO. Un astfel de test unitar cu framework-ul Mockito arată ca în Figura 5.19.

```

@SpringBootTest
@ActiveProfiles("test")
class RoleServiceImplTest extends AbstractTest {
    @InjectMocks
    private RoleServiceImpl roleService;

    @Mock
    private IRoleRepository roleRepository;

    private Iterable<RoleEntity> mockRoles;

    @BeforeEach
    void setUp() {
        roleService = new RoleServiceImpl(roleRepository);
        final UserFactory userFactory = new UserFactory();
        mockRoles = userFactory.createMockAdminRoles();
    }

    @Test
    void whenFindAllRoles_thenReturnMockRoles() {
        when(roleRepository.findAll()).thenReturn(mockRoles);
        final List<RoleDTO> roles = roleService.findAllRoles();
        verify(roleRepository, times(wantedNumberOfInvocations: 1)).findAll();
        assertNotNull(roles);
        assertEquals(roles.size(), actual: 3);
        assertEquals(roles.get(0).getRole(), actual: "ADMIN");
    }
}

```

Figura 5.19: Clasă de testare pentru serviciul de roluri

După cum se poate observa în Figura 5.19, obiectul pentru care se oferă o implementare fictivă este *roleRepository*, iar *roleService* este obiectul ce reprezintă clasa ce se va instanția și va injecta toate obiectele marcate cu adnotarea *@Mock*.

5.6 Etapa de deploy

În ultimul rând, după implementarea și testarea aplicației este foarte important ca aceasta să poată fi folosită de publicul țintă.

5.6.1 Găzduirea aplicației

Pentru a experimenta modul în care aplicația poate fi utilizată de public, am ales să transform o unitate veche de calculator într-un Linux server pentru găzduirea de aplicații web și nu numai. Astfel, am instalat **Ubuntu 20.04**, acesta fiind foarte ușor de folosit.

5.6.2 Configurarea Linux server-ului și a aplicației web

Desigur, pentru ca server-ul să poată îndeplini cerințele de “gazdă” a trebuit să fac mai multe configurări. Deoarece sunt abonat **RCS & RDS**¹, beneficiaz de un nume de domeniu gratuit. Acest lucru m-a ajutat foarte mult din două perspective:

- deoarece IP-ul meu public este dinamic și la fiecare întrerupere de curent sau căderi de Internet mi se schimbă de fiecare dată;
- pentru a putea accesa aplicația web nu trebuie scris în browser IP-ul server-ului și portul (fiind destul de neplăcut, mai ales că IP-ul se schimbă destul de des), ci doar numele domeniului.

În cele ce urmează vom prezenta pașii de configurare a Linux server-ului și a interfeței dintre Internet și server.

Pasul 1:

La conectarea oricărui dispozitiv în rețeaua de **WI-FI**, acestuia i se asignează o adresă IPv4 care se poate schimba de fiecare dată. Astfel, în cazul Linux server-ului, pentru ca această adresă să rămână neschimbată și pentru a se pointa corect spre server, trebuie ca din interfața router-ului (accesată de pe server) să îi fie asignată o adresă IPv4 rezervată.

Configurarea unui **DMZ Host** pe router se referă defapt la configurarea unui host în rețeaua internă și pregătirea pentru port-forwarding pentru a putea accesa o resursă dintr-o rețea locală [25].

Pasul 2:

După completarea primului pas, trebuie ales numele de domeniu. Se intră în contul de DIGI și de acolo se caută secțiunea cu DNS Dinamic, iar apoi se completează numele de domeniu dorit. Dacă acesta nu este disponibil, va apărea o atenționare. După completarea domeniului, un restart al router-ului este necesar pentru ca schimbările asupra lui să fie reflectate [7].

După acești primi doi pași, putem spune că avem un server web pregătit pentru a găzdui aplicații web.

Pasul 3:

Având Docker instalat pe Linux server, am descărcat pe rând imagini de pe Docker Hub² pentru a-mi pregăti mediul de lucru. Astfel, pentru baza de date, am

¹Cunoscut și sub numele de DIGI, este o companie de televiziune și Internet din România

²Repository care conține diferite container images

luat o imagine mysql, pentru servirea site-ului am luat o imagine nginx, iar pentru execuția jar-ului, am luat o imagine a lui openjdk versiunea 15 (versiunea 15 fiind și cea folosită la scrierea codului java).

Pasul 4:

După descărcarea imaginilor Docker prezentate la pasul anterior, am creat un director corespunzător tuturor resurselor folosite pentru aplicație. Acest director are structura ca în Figura 5.20. Argumentele referitoare la utilizarea acestei structuri, precum și conținutul fiecărui fisier va fi prezentat în cele ce urmează.

```
alexandru@ubuntu:~/renty$ tree
.
├── angulararchive
│   ├── rentyapp.2021.05.03-23:22:37
│   │   ├── 3rdpartylicenses.txt
│   │   ├── assets
│   │   │   └── columns.json
│   │   ├── favicon.ico
│   │   ├── index.html
│   │   ├── main-es2015.2be40a0fe96689db836d.js
│   │   ├── main-es2015.39e69edcd905a3fa6084.js
│   │   ├── main-es2015.5d21d200f144a94dac23.js
│   │   ├── main-es5.2be40a0fe96689db836d.js
│   │   ├── main-es5.39e69edcd905a3fa6084.js
│   │   ├── main-es5.5d21d200f144a94dac23.js
│   │   ├── polyfills-es2015.f332a089ad1600448873.js
│   │   ├── polyfills-es5.177e85a9724683782539.js
│   │   ├── runtime-es2015.0dae8cbc97194c7caed4.js
│   │   ├── runtime-es5.0dae8cbc97194c7caed4.js
│   │   └── styles.5247984d6efb0fe452c1.css
│   └── rentyapp.2021.05.08-14:38:14
│       ├── 3rdpartylicenses.txt
│       ├── assets
│       │   └── columns.json
│       ├── favicon.ico
│       ├── index.html
│       ├── main-es2015.0ac80cf2cd7f183cbcd7.js
│       ├── main-es5.0ac80cf2cd7f183cbcd7.js
│       ├── polyfills-es2015.f332a089ad1600448873.js
│       ├── polyfills-es5.177e85a9724683782539.js
│       ├── runtime-es2015.0dae8cbc97194c7caed4.js
│       ├── runtime-es5.0dae8cbc97194c7caed4.js
│       └── styles.5247984d6efb0fe452c1.css
├── docker-compose.yml
├── rentyadm
├── rentyapp
│   ├── 3rdpartylicenses.txt
│   ├── assets
│   │   └── columns.json
│   ├── favicon.ico
│   ├── index.html
│   ├── main-es2015.0ac80cf2cd7f183cbcd7.js
│   ├── main-es5.0ac80cf2cd7f183cbcd7.js
│   ├── polyfills-es2015.f332a089ad1600448873.js
│   ├── polyfills-es5.177e85a9724683782539.js
│   ├── runtime-es2015.0dae8cbc97194c7caed4.js
│   ├── runtime-es5.0dae8cbc97194c7caed4.js
│   └── styles.5247984d6efb0fe452c1.css
├── renty.jar
├── renty.nginx.conf
├── restartarchive
│   └── renty.jar.2021.04.14-10:55:56
└── 8 directories, 42 files
alexandru@ubuntu:~/renty$
```

Figura 5.20: Structura directorului corespunzător tuturor resurselor folosite în aplicație

- Subdirectorul **angulararchive** este un director folosit pentru a stoca versiunile mai vechi ale părții de front-end a aplicației web. După cum se observă, la numele directorului se adugă data și ora la care a fost creat. Motivul precum și alte explicații vor fi prezentate în secțiunea următoare.
- Subdirectorul **rentyapp** este directorul în care este stocat ultima versiune a părții de front-end a aplicației Renty. Acesta conține fisierul index.html pre-

cum și alte fișiere dependente lui.

- Fișierul **renty.jar** este fișierul arhivat al părții de back-end a aplicației web.
- Subdirectorul **restarchive** este un director folosit pentru a stoca versiunile mai vechi ale părții de back-end a aplicației web.
- Fișierul **renty.nginx.conf** este fișierul de configurare folosit de Nginx la servirea site-ului. Un astfel de fișier de configurare arată ca în Figura 5.21 de mai jos.

```
alexandru@ubuntu:~/renty$ cat -n renty.nginx.conf
 1  server {
 2      listen 80;
 3      server_name renty.go.ro;
 4
 5      root /usr/share/nginx/html;
 6      index index.html;
 7
 8      location / {
 9          try_files $uri $uri/ /index.html
10      }
11  }
```

alexandru@ubuntu:~/renty\$ █

Figura 5.21: Fișier de configurare Nginx

După cum putem observa, un fișier de configurare nginx este foarte simplu. Astfel, linia 1 și linia 11 corespund începerii respectiv încheierii configurării server-ului web. În linia 2 este prezentat portul pe care nginx va servi, iar în linia 3 este prezentat domeniul ales (cel ales din contul DIGI). În linia 5 este prezentat directorul specific lui index.html și a fișierelor dependente. Pe linia 6 este prezentat fișierul index.html (principalul fișier html) iar liniile 8-10 prezintă URL-ul către fișierul solicitat [15].

- Fișierul **rentyadm** este un script bash folosit pentru a automatiza procesul de deploy și va fi prezentat în detaliu în secțiunea următoare.
- Fișierul **docker-compose.yml** este fișierul de configurare corespunzător containerelor docker. Un astfel de fișier de configurare arată ca în Figura 5.22.

```
alexandru@ubuntu:~/renty$ cat -n docker-compose.yml
 1  version: '3'
 2
 3  services:
 4    renty-database:
 5      image: mysql
 6      container_name: renty-database
 7      environment:
 8        MYSQL_USER: root
 9      ports:
10        - 8050:3306
11      networks:
12        - backendNetwork
13
14    renty-restapi:
15      image: openjdk:15
16      container_name: renty-restapi
17      ports:
18        - 8080:8080
19      depends_on:
20        - renty-database
21      volumes:
22        - ./renty.jar:/application.jar
23      command: ["java", "-jar", "application.jar"]
24      networks:
25        - backendNetwork
26        - frontendNetwork
27
28    renty-angular:
29      image: nginx
30      container_name: renty-angular
31      ports:
32        - 80:80
33      depends_on:
34        - renty-restapi
35      volumes:
36        - ./renty.nginx.conf:/etc/nginx/sites-enabled/nginx.conf
37        - ./rentyapp:/usr/share/nginx/html
38      networks:
39        - frontendNetwork
40
41  volumes:
42    rentyVolumes:
43
44  networks:
45    backendNetwork:
46    frontendNetwork:
```

alexandru@ubuntu:~/renty\$ █

Figura 5.22: Fișier de configurare docker-compose

Astfel, pentru aplicația Renty am folosit 3 servicii:

- care reprezintă baza de date MySQL și anume **renty-database**;
- un serviciu folosit pentru rularea jar-ului și anume **renty-restapi**;
- un serviciu folosit pentru furnizarea fișierelor statice reprezentat de nginx și anume **renty-angular**.

Astfel, liniile 5, 15 și 29 corespund imaginilor folosite de container, iar liniile 6, 16 și 30 sunt numele containerelor cu care pot fi identificate în sistem. Liniile 9-10, 17-18 și 31-32 corespund porturilor pe care rulează fiecare container; primul port fiind cel destinat exteriorului, iar cel de-al doilea destinat interiorului.

Liniile 11-12, 24-26 și 38-39 corespund rețelei și arată că aceste containere pot comunica între ele. Liniile 21-22 și 35-37 arată resursele care pot fi distribuite și împărțite între sistemul local și container.

Pasul 5:

După scrierea tuturor fișierelor de configurare, trebuie deschise porturile corespunzătoare fiecărei părți a aplicației web pentru a putea fi folosită și de public, nu doar local. Pentru a porni containerele docker, în terminalul deschis în directorul principal corespunzător aplicației, se scrie comanda **docker-compose up**. Această comandă va “reînvia” fiecare container sau îl va crea dacă nu există.

Astfel, după toți acești pași, aplicația poate fi folosită de public și poate fi accesată la adresa **http://renty.go.ro**.

5.6.3 Integrare continuă

Pentru a automatiza procesul de deploy, precum și cel de analiză a calității codului scris, am ales să integrez **Jenkins** și **SonarQube**. Aceste procese poartă numele de *Continuous integration/Continuous delivery/Continuous analysis (CI/CD/CA)*.

Instalarea acestor utilitare pe mașina Linux, este destul de greoaie, și astfel am ales să le integrez într-un Docker. Fișierul de configurare docker pentru aceste două utilitare arată ca în Figura 5.23.

```
alexandru@ubuntu:~/cicd$ cat -n docker-compose.yml
 1  version: '3'
 2
 3  services:
 4    sonarqube:
 5      image: sonarqube
 6      ports:
 7        - 8300:9000
 8      networks:
 9        - cicd
10
11    jenkins:
12      image: jenkins/jenkins:jdk11
13      ports:
14        - 8200:8080
15      networks:
16        - cicd
17
18  networks:
19    cicd:
```

Figura 5.23: Fișier de configurare Docker pentru Jenkins și SonarQube

Crearea conturilor pentru Jenkins și SonarQube s-a realizat, urmând logurile la crearea acestor două containere. Pentru ca aceste utilitare să poată fi accesate din exterior, este necesară deschiderea porturilor de pe mașina Linux corespunzătoare fiecărui container.

Pentru automatizarea procesului de deploy, am creat un bash script care acceptă ca argument o acțiune. Acest script se ocupă cu managementul resurselor aplicației web. Fiecărei acțiuni îi corespunde anumite funcții descrise în cele ce urmează. Acest bash script arată ca în Figurile 5.24 și 5.25 și este executat din Jenkins cu ajutorul SSH-ului.

```
1  #!/bin/bash
2
3  ARG=$1
4
5  RENTYROOT=/home/alexandru/renty
6
7  RENTYAPP=rentyapp
8  RENTYJAR=renty.jar
9
10 DOCKER=docker-compose.yml
11
12 ANGARCH=angulararchive
13 RESTARCH=restarchive
14
15 RENTYUSER=alexandru
16
17 TMPJAR=/tmp/build/libs/$RENTYJAR
18 TMPAPP=/tmp/dist/$RENTYAPP
19
20 RESTAPICONTAINER=renty-restapi
21 ANGULARCONTAINER=renty-angular
22
23 changerights() {
24     chmod -R 775 $RENTYROOT
25 }
26
27 angularcopy() {
28     changerights
29     DATE=`date '+%Y.%m.%d-%H:%M:%S'`
30     cd $RENTYROOT
31     APP=`ls | grep ${RENTYAPP} | tail -n 1`
32     if [ "$APP" != "" ]; then
33         sudo -u $RENTYUSER mv $RENTYROOT/$RENTYAPP $RENTYROOT/$ANGA
34         echo "angularcopy()"
35     fi
36 }
37
38 restapicopy() {
39     changerights
40     DATE=`date '+%Y.%m.%d-%H:%M:%S'`
41     cd $RENTYROOT
42     JAR=`ls | grep ${RENTYJAR} | tail -n 1`
43     if [ "$JAR" != "" ]; then
44         sudo -u $RENTYUSER mv $RENTYROOT/$RENTYJAR $RENTYROOT/$REST.
45         echo "restapicopy()"
46     fi
47 }
```

Figura 5.24: Script utilizat în procesul de deploy

```
48
49 preparecontainers() {
50     sudo -u $RENTYUSER docker stop $RESTAPICONTAINER
51     sudo -u $RENTYUSER docker stop $ANGULARCONTAINER
52     sudo -u $RENTYUSER docker rm $RESTAPICONTAINER
53     sudo -u $RENTYUSER docker rm $ANGULARCONTAINER
54 }
55
56 restart() {
57     changerights
58     preparecontainers
59     sudo -u $RENTYUSER chmod a+x $RENTYROOT/$RENTYJAR
60     docker-compose -f $RENTYROOT/$DOCKER up --remove-orphans -d
61     echo "restart()"
62 }
63
64 copyjarfromtmp() {
65     sudo -u $RENTYUSER cp $TMPJAR $RENTYROOT
66 }
67
68 copyappfromtmp() {
69     sudo -u $RENTYUSER cp -r $TMPAPP $RENTYROOT
70 }
71
72 case $ARG in
73     angcopy) angularcopy ;;
74     apicopy) restapicopy ;;
75     angular) copyappfromtmp ;;
76     restapi) copyjarfromtmp ;;
77     restart) restart ;;
78     *) echo "Invalid argument!"
79 esac
80
```

Figura 5.25: Script utilizat în procesul de deploy

Pentru început, după cum se poate observa, am definit câteva variabile globale pentru a-mi ușura lucrul în scrierea scriptului. Așadar:

- Funcția **changerights()** are rolul de a da toate permisiunile user-ului curent (în acest caz, user-ului Jenkins) asupra directorului corespunzător tuturor resurselor aplicației web, iar pentru ceilalți useri drepturi de citire și execuție.
- Funcțiile **angularcopy()** și **restapicopy()** fac cam același lucru, adică salvarea într-o arhivă a resurselor curente a fiecărei componente (de back-end și front-end) adăugându-se data și ora curentă. Am ales să salvez vechile versiuni, deoarece în caz de probleme majore a versiunii noi a aplicației, să se poată reveni cu ușurință la versiunea anterioară.
- Funcția **preparecontainers()** oprește containerele corespunzătoare părții de front-end și părții de back-end, după care le șterge.
- Funcția **restart()** are rolul de a “reînvia” containerele docker.
- Funcțiile **copyjarfromtmp()** și **copyappfromtmp()** fac cam același lucru, copiind din subdirectorul /tmp (unde este stocat codul livrat de Jenkins) resursele necesare aplicației.

- În cele din urmă, are loc interogarea argumentului primit și apelul funcției corespunzătoare.

Pentru ca să pot executa acest script din Jenkins a trebuit să fac foarte multe configurări și să descarc niște plugin-uri ajutătoare. În cele ce urmează o să prezint pașii cei mai importanți în configurarea Jenkins-ului.

Pentru a avea un control asupra versionării codului, am folosit GitHub. Astfel, la fiecare adăugare de cod pe git, un automatic build și deploy (din Jenkins) începea, iar mai apoi o analiză a codului era prezentată în SonarQube. Astfel în acest proces de deploy, am adăugat niște pași apelând scriptul cu parametrii corespunzători, precum și transferul resurselor arhivate (jar-ul și index.html împreună cu fișierele dependente) în directorul /tmp de pe mașina Linux. În final, am mai adăugat un pas de restartare a aplicației, constând într-o “reînviere” a containerelor docker.

Capitolul 6

Concluzii și posibilități de dezvoltare viitoare

6.1 Concluzii

Deși tehnologiile și framework-urile se schimbă și evoluează de pe o zi pe alta, aplicația trebuie să fie dezvoltată în așa fel încât să poată fi întreținută și ușor de integrat și folosit un alt framework.

Fiecare framework și tehnologie în parte are avantajele și dezavantajele lui. Utilizarea framework-ului Spring pe partea de backend a fost o alegere foarte bună deoarece acesta prezintă foarte multe facilități în dezvoltarea unei aplicații complexe client-server și nu numai. De asemenea folosirea de imagini Docker a simplificat foarte mult pregătirea mediului de lucru.

Din cele prezentate în Capitolul 5, la dezvoltarea aplicației Renty, am reușit să obțin un maxim posibil dintr-un minim disponibil ducând aplicația până în producție, putând fi folosită de public.

6.2 Posibilități de dezvoltare

Pentru ca aplicația să aibă un succes mai mare și pentru a deveni un produs cât mai complex și complet s-ar putea adăuga următoarele funcționalități:

- posibilitate de primire discount la următoarea rezervare dacă un utilizator și-a anulat-o;
- posibilitatea ca un proprietar să ofere discount la un anumit teren și anumită activitate în funcție de statistici;
- posibilitatea ca utilizatorii să comunice între ei (de exemplu: un grup de prieteni care vor să joace fotbal) pentru a stabili detaliile rezervării;

- posibilitatea plății în grup cu cardul (de exemplu: un grup de prieteni care vor să joace basket) astfel fiecare să-și poată plăti partea lui pentru a simplifica cu mult strângerea de bani de la finalul jocului și returnarea acestora către cel care a plătit rezervarea;
- adăugarea de reclame pentru ca fondatorul aplicației să aibă un minim venit din aceasta;
- posibilitatea de adăugare cont bancar pentru fiecare proprietar și evitarea folosirii unui cont general.

6.3 Îmbunătățiri viitoare

Desigur că pentru ca aplicația să poată fi întreținută pe viitor este foarte important să fie scris cod cât mai structurat, să fie o cuplare redusă și o coeziune ridicată pe cât posibil.

Pentru a minimiza riscul de a apărea erori în producție, iar apoi să nu se investească foarte mult timp în a rezolva problema, ar trebui testat toată aplicația. O posibilă îmbunătățire a codului ar fi să se facă teste unitare și pe endpoint-uri, să se facă integration testing folosind MockMVC-uri pe o bază de date H2, iar pentru partea de frontend, de asemenea teste unitare pentru fiecare componentă. Testarea automată a paginii web ar fi o abordare foarte bună atât pentru partea de client cât și server. Astfel testarea aplicației cu Selenium sau SerenityBDD ar aduce un plus semnificativ din toate punctele de vedere.

Interfața grafică, cea pe care o vede fiecare utilizator, este de asemenea foarte importantă. Aceasta trebuie să fie cât mai accesibilă din toate punctele de vedere pentru întreaga populație.

Comunicarea dintre server și client este una periculoasă. Pachetele trimise nu sunt criptate și oricând pot fi interceptate de către un hacker. Utilizarea unui certificat SSL ar reduce semnificativ riscul la care este supus în momentul de față un utilizator al aplicației Renty.

Bibliografie

- [1] ***. *Welcome to NGINX Wiki!* <https://www.nginx.com/resources/wiki/>. Ultima accesare: 03.05.2021.
- [2] ***. *What is World Wide Web?* <https://www.javatpoint.com/what-is-world-wide-web>. Ultima accesare: 02.06.2021.
- [3] MDN contributors: ***. *Cacheable*. <https://developer.mozilla.org/en-US/docs/Glossary/cacheable>. Ultima accesare: 10.05.2021.
- [4] MDN contributors: ***. *HTTP request methods*. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>. Ultima accesare: 10.05.2021.
- [5] MDN contributors: ***. *Idempotent*. <https://developer.mozilla.org/en-US/docs/Glossary/Idempotent>. Ultima accesare: 10.05.2021.
- [6] MDN contributors: ***. *Safe*. <https://developer.mozilla.org/en-US/docs/Glossary/Safe>. Ultima accesare: 10.05.2021.
- [7] Adrian. *DIGI free domain go.ro for dynamic IP, like DynDNS*. <https://en.videotutorial.ro/digi-domeniu-gratuit-go-ro-pentru-ip-dinamic-ca-dyndns>. Ultima accesare: 03.05.2021.
- [8] Alexandra Altvater. *SOAP vs. REST: The Differences and Benefits Between the Two Widely-Used Web Service Communication Protocols*. <https://stackify.com/soap-vs-rest/>. Ultima accesare: 13.05.2021.
- [9] Charles Anderson. "Docker [software engineering]". in: *Ieee Software* 32.3 (2015), pages 102–c3. DOI: 10.1109/MS.2015.62.
- [10] Baeldung. *A Comparison Between Spring and Spring Boot*. <https://www.baeldung.com/spring-vs-spring-boot>. Ultima accesare: 07.06.2021.
- [11] Krishna Bhatia. *Differences between JDK, JRE and JVM*. <https://www.geeksforgeeks.org/differences-jdk-jre-jvm/>. Ultima accesare: 03.05.2021.
- [12] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte and Dave Winer. *Simple object access protocol (SOAP) 1.1*. 2000.

- [13] David W Brooks, Diane E Nolan **and** Susan M Gallagher. *Web-teaching: A guide to designing interactive teaching for the World Wide Web*. **volume** 9. Springer Science & Business Media, 2006.
- [14] Jon Byous. "Java technology: an early history". **in:** URL: <http://java.sun.com/features/1998/05/html>, Artigo pesquisado em 07 de Junho de 2002 (1998).
- [15] nginx community. *NGINX Beginner's Guide*. http://nginx.org/en/docs/beginners_guide.html. Ultima accesare: 05.05.2021.
- [16] sonarqube community. *About SonarQube*. <https://www.sonarqube.org/about/>. Ultima accesare: 05.05.2021.
- [17] IBM Corporation. *The structure of a SOAP message*. <https://www.ibm.com/docs/en/integration-bus/10.0?topic=soap-structure-message>. Ultima accesare: 01.06.2021.
- [18] Ian Fette **and** Alexey Melnikov. *The websocket protocol*. 2011.
- [19] Roy Thomas Fielding. *CHAPTER 5 Representational State Transfer (REST)*. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Ultima accesare: 13.05.2021.
- [20] Josh Fruhlinger. *What is SSL, TLS? And how this encryption protocol works*. <https://www.csoonline.com/article/3246212/what-is-ssl-tls-and-how-this-encryption-protocol-works.html>. Ultima accesare: 02.06.2021. 2018.
- [21] *H2 Databse - Introduction*. https://www.tutorialspoint.com/h2_database/h2_database_introduction.htm. Ultima accesare: 05.05.2021.
- [22] Chris Kemmerer. *The SSL/TLS Handshake: an Overview*. <https://www.ssl.com/article/ssl-tls-handshake-overview/>. Ultima accesare: 02.06.2021. 2015.
- [23] Fabian Pfaff Lars Vogel. *Unit tests with Mockito - Tutorial*. <https://www.vogella.com/tutorials/Mockito/article.html>. Utlima accesare: 07.06.2021. 2021.
- [24] Barry M Leiner, Vinton G Cerf, David D Clark, Robert E Kahn, Leonard Kleinrock, Daniel C Lynch, Jon Postel, Larry G Roberts **and** Stephen Wolff. "A brief history of the Internet". **in:** *ACM SIGCOMM Computer Communication Review* 39.5 (2009), **pages** 22–31. DOI: <https://doi.org/10.1145/1629607.1629613>.
- [25] Ben Lutkevich. *DMZ (networking)*. <https://searchsecurity.techtarget.com/definition/DMZ>. Ultima accesare: 03.05.2021.
- [26] Dan Maharry. *TypeScript revealed*. Apress, 2013.

- [27] MicroFocus. *Using Enterprise Test Server with Docker*. <https://www.microfocus.com/documentation/enterprise-developer/ed40pu5/ETS-help/GUID-D2F52B32-3C20-40D8-9B33-8CAB49AA2E1D.html>. Ultima accesare: 08.06.2021.
- [28] Chris Motola. *How Does Stripe Work? Everything You Need To Know About Processing Payments With Stripe*. <https://www.merchantmaverick.com/how-does-stripe-work/>. Ultima accesare: 06.05.2021.
- [29] Sagar Nangare. *OpenShift, Kubernetes and Docker: A Quick Comparison*. <https://containerjournal.com/topics/container-ecosystems/openshift-kubernetes-and-docker-a-quick-comparison/>. Ultima accesare: 12.06.2021. 2019.
- [30] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki and Peter Steenkiste. "The cost of the "s" in https". **in:** *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 2014, **pages** 133–140. DOI: <https://doi.org/10.1145/2674005.2674991>.
- [31] Mr Nerd. *Angular 9: What's Angular and Angular 9 New Features*. <https://medium.com/techiediaries-com/angular-9-whats-angular-and-angular-9-new-features-ff165895915c>. Ultima accesare: 07.06.2021. 2020.
- [32] Eugen Paraschiv. *Spring Profiles*. <https://www.baeldung.com/spring-profiles>. Ultima accesare: 05.05.2021.
- [33] Leonard Richardson and Sam Ruby. *RESTful web services*. "O'Reilly Media, Inc.", 2008.
- [34] Alex Rodriguez. "Restful web services: The basics". **in:** *IBM developerWorks* 33 (2008), **page** 18.
- [35] John Ferguson Smart. *Jenkins: The Definitive Guide: Continuous Integration for the Masses*. "O'Reilly Media, Inc.", 2011.
- [36] Ihor Sokolyk. *Using Stripe payments in Spring Boot + Angular application*. <https://medium.com/oril/easy-way-to-integrate-payments-with-stripe-into-your-spring-boot-angular-application-c4d03c7fc6e>. Ultima accesare: 07.05.2021.
- [37] Steve Suehring. *MySQL Bible*. Wiley Publishing, Inc., 2001.