



**UNIVERSITATEA DIN
BUCUREȘTI**

**FACULTATEA DE
MATEMATICĂ ȘI
INFORMATICĂ**



SPECIALIZAREA INFORMATICĂ

Lucrare de licență

Pet Kindness –Aplicație web pentru adopția de animale

Absolvent

Voicu Bianca-Oana

Coordonator științific

Lect.(det.) dr. Ana Cristina Dăscălescu

București, iunie 2023

Rezumat

Pet Kindness, aplicația web care este prezentată în această lucrare de licență are ca obiectiv promovarea adopției de animale, venind atât în ajutorul persoanelor care doresc o platformă destinată exclusiv adopției, ușor de utilizat, cât și în ajutorul animalelor care necesită o casă nouă. Există și posibilitatea de a dona în sprijinul animalelor din adăposturi.

În urma analizei variantelor de adopție online existente în România, am concluzionat că nu există o platformă suficient de diversă. Există, din păcate, doar pentru cele două categorii comune, câini și pisici. După părerea mea, orice persoană care nu mai poate îngriji un animal, indiferent de specie, ar trebui să aibă posibilitatea de a-i găsi o nouă familie într-un mod facil. De asemenea, scopul meu este ca Pet Kindness să fie nu doar diversă, dar și dedicată adopției, fără varianta de vânzare, deoarece nu toată lumea este înclinată să cumpere, îngrijirea unui animal de companie fiind deja costisitoare și fără un cost inițial. Astfel, procesul de adopție poate fi mai rapid, dacă se exclud anumite costuri. În plus, dă ocazia persoanelor care și-ar dori o anumită rasă, să și-o permită, iar animaluțul primește în schimb un loc în care va fi iubit și îngrijit, atunci când actualul stăpân nu poate să îi mai ofere un trai corespunzător.

Pentru a stârni interesul, un utilizator are acces la pagina de „Adopții” fără a fi nevoie să își creeze mai întâi un cont, pentru a vedea de la început dacă există un animaluț potrivit pentru acesta. Există posibilitatea de a filtra animalele după specie și gen. În momentul în care utilizatorul se hotărăște, se poate conecta la un cont existent (folosindu-se de nume de utilizator și parolă) sau își poate crea unul nou. Apoi, pentru cererea de adopție propriu-zisă, este necesară introducerea unor date personale adiționale (date de contact și datele cărții de identitate). Astfel, documentul se completează automat și este trimis pe mail utilizatorului care a postat animalul în aplicație. De asemenea, este foarte ușor să adaugi un animaluț spre adopție. Este nevoie doar de crearea unui cont, iar pe pagina de profil se găsește un buton care redirecționează utilizatorul către un formular de adăugare. În plus, pe pagina „Centre partenere” am dorit să includ informații despre adăposturi de câini care nu au site-uri dedicate, dar care pot fi vizitate fizic.

Cu ajutorul tehnologiilor Java, Spring Boot, Angular, Bootstrap, MySQL, Maven, am dezvoltat cu succes o aplicație web care să dea o șansă pentru o nouă casă oricărui tip de animal de companie, nu doar celor mai comune specii.

Abstract

Pet Kindness, the web application described in this bachelor's thesis has as main objective the promotion of animal adoption, coming both to the aid of people who want a platform intended exclusively for adoption, which is easy to use, as well as in the help of animals which for various reasons need a new home. There is also an opportunity to donate in support of the pets in shelters.

Following the analysis of the existing online adoption options in Romania, I drew the conclusion that a platform that is sufficiently diverse, does not exist. Unfortunately, it exists only for the two most common categories, dogs and cats. In my opinion, any person who no longer can take care of a pet, regardless of species, should have the possibility to find it a new family in an easy way. Also, my goal is that Pet Kindness is not only diverse, but also dedicated to adoption, without the sales option, because not everyone is inclined to buy, caring for a pet being already expensive even without an initial cost. Therefore, the adoption process can be faster, if certain costs are excluded. In addition, it gives people that would like a specific breed, the opportunity to afford it, and the pet receives in exchange a place where it will be loved and cared for, when the current owner can no longer offer it proper living conditions.

To spark interest, a user has access to the “Adoptions” page without having to create an account first, to see from the beginning if there is a suitable pet. There is the possibility to filter the pets by species and gender. When the user decides, he can connect to an existing account (using username and password) or he can create a new one. Then, for the actual adoption application, it is necessary to enter additional personal data (contact details and identity card data). Thus, the document is filled in automatically and sent by e-mail to the user who posted the animal in the application. Also, it is very easy to add a pet for adoption. You just need to create an account, and on the profile page you can find a button that redirects the user to an add form. In addition, on the “Partner Centres” page I wanted to include information about dog shelters that do not have dedicated websites, but can be visited in person.

With the help of Java, Spring Boot, Angular, Bootstrap, MySQL, Maven technologies, I managed to develop successfully a web application that would give a chance for a new home to any type of pet, not just the most common species.

Cuprins

| | |
|---|----|
| I.Introducere | 6 |
| I.1 Scopul și motivația lucrării | 6 |
| I.2 Contribuția personală | 6 |
| I.3 Structura lucrării | 7 |
| II. Preliminarii | 8 |
| II.1 Dezvoltare Web Full Stack | 8 |
| II.2 Comunicare prin cereri HTTP | 8 |
| III. Tehnologiile folosite | 10 |
| III.1 Limbaje de programare | 10 |
| III.1.1 Java | 10 |
| III.1.2 TypeScript | 11 |
| III.2 Frameworks | 12 |
| III.2.1 Spring și Spring Boot | 12 |
| III.2.2 Angular | 13 |
| III.2.3 Hibernate | 14 |
| III.2.4 Bootstrap | 15 |
| III.3 MySQL | 16 |
| IV. Arhitectura aplicației | 17 |
| IV.1 Arhitectura MVC (model-view-controller) | 17 |
| IV.2 Baza de date | 21 |
| IV.3 Cazuri de utilizare (Use cases) | 21 |
| V. Prezentarea aplicației | 23 |
| V.1 Pagina principă („Acasă”) | 23 |
| V.2 Pagina de înregistrare („Sign up”) | 24 |
| V.3 Pagina de autentificare („Login”) | 25 |
| V.4 Pagina cu animalele spre adopție („Adopții”) | 26 |
| V.5 Pagina unui animal spre adopție | 27 |
| V.6 Pagina adăposturilor partenere („Centre partenere”) | 30 |
| V.7 Pagina de profil a utilizatorului | 31 |
| V.8 Pagina de adăugare a unui animal spre adopție | 33 |
| V.9 Pagina de statistici | 34 |
| VI. Testare și elemente de securitate | 35 |

| | |
|--|----|
| VI.1 Testare | 35 |
| VI.1.1 AuthController | 35 |
| VI.1.2 UserPersonalDetailsController | 35 |
| VI.1.3 PetCenterController | 37 |
| VI.1.4 PetController | 38 |
| VI.2 Validări | 39 |
| VI.2.1 Formularul de înregistrare | 39 |
| VI.2.2 Formularul de autentificare | 41 |
| VI.2.3 Formularul de adopție | 41 |
| VI.2.4 Formularul de adăugare a unui centru partener | 42 |
| VI.2.5 Formularul de adăugare a unui animal spre adopție | 43 |
| VI.2.6 Formularul de editare a unui animal spre adopție | 43 |
| VI.3 Elemente de securitate | 44 |
| VII. Concluzii | 45 |
| VIII. Referințe bibliografice | 46 |

I.Introducere

I.1 Scopul și motivația lucrării

Am ales să dezvolt aplicația web Pet Kindness din iubirea mea incomensurabilă față de animale și din lipsa platformelor destinate strict adopțiilor de animale pe piața din România, dar pentru specii variate, care să dea o șansă tuturor tipurilor de animăluțe de companie la o nouă familie.

După o cercetare a aplicațiilor web care asigură servicii de adopții de animale, am descoperit că aproape toate sunt dedicate adopției de câini și pisici. Cele care aveau și alte specii de animale, nu erau dedicate total adopției, ci aveau și vânzări. În urma unui sondaj efectuat de către mine pe 30 de persoane, majoritatea a fost de acord că în cazul în care o persoană este nehotărâtă, poate să se răzgândească în privința adopției, dacă tot în aceeași platformă există și vânzări. Prin urmare, am concluzionat că ar fi utilă o platformă dedicată exclusiv adopțiilor.

Scopul lucrării este de a favoriza procesul de adopție a animalelor de companie din specii diverse, printr-o platformă dedicată, ușor de folosit de utilizatorii care doresc să adopte sau să ofere spre adopție, totul dintr-un singur cont. Iar pentru persoanele care nu pot adopta un animăluț, există și posibilitatea de a dona în sprijinul celor din adăposturi.

I.2 Contribuția personală

Contribuția proprie este reprezentată de proiectarea, design-ul și implementarea aplicației web Pet Kindness. Pentru acestea am folosit cunoștințe de bază învățate în timpul anilor de studiu, pe care le-am aprofundat pentru dezvoltarea acestui proiect, împreună cu tehnologii noi, cu care nu am mai intrat în contact până acum.

La partea de interfață a utilizatorului (UI), am preferat să mențin un design simplu, coerent pe toate paginile. Am folosit în principal nuanțe calde de verde, o culoare veselă, plăcută, care duce cu gândul la relaxare și natură. Când vine vorba de experiența utilizatorului (UX), am vrut să facilitez procesul de adopție, deci să fie unul clar și simplu, bazat pe formulare aflate la un buton distanță (cu nume sugestive).

Baza de date gândită și proiectată de mine are 6 tabele (informațiile reținute fiind despre utilizatori, animalele date spre adopție și centrele de adopție partenere), menținând dorința mea de a nu complica lucrurile.

Pentru generarea automată de documente personalizate, am ales să nu folosesc Apache POI, o bibliotecă populară pentru Java, ci am folosit faptul că un document .docx este de fapt o arhivă de tip .zip (am preferat să fie o provocare, ceva ce nu am mai făcut până acum). Cererea de adopție personalizată este trimisă mai departe pe email către persoana care a publicat anunțul cu animalul de companie. Pentru a exemplifica trimiterea mesajului electronic cu atașamentul respectiv, am ales să folosesc platforma Mailtrap, care simulează un inbox real.

I.3 Structura lucrării

- **Capitolul I:** Conține scopul, motivația lucrării și descrierea pe scurt a contribuției proprii.
- **Capitolul II:** Conține noțiuni tehnologice care stau la baza temei.
- **Capitolul III:** Conține o prezentare a tehnologiilor folosite pentru a dezvolta aplicația și motivația alegerii acestora.
- **Capitolul IV:** Conține prezentarea arhitecturii și a structurii aplicației.
- **Capitolul V:** Conține prezentarea amănunțită a aplicației, cu detalii de implementare și capturi de ecran ale interfeței cu utilizatorul.
- **Capitolul VI:** Conține metodele de testare, validările efectuate și elemente de securitate.
- **Capitolul VII:** Conține concluziile finale și direcții viitoare de dezvoltare.
- **Capitolul VIII:** Conține bibliografia cu toate resursele utilizate.

II. Preliminarii

II.1 Dezvoltare Web Full Stack

Dezvoltarea web implică în mod normal partea de client care, împreună cu partea de server, formează dezvoltarea web full-stack. Partea de client se mai numește și front-end, iar cea de server este denumită back-end. După cum se poate observa în *figura 2.1*, limbajele cel mai des utilizate pe partea de client sunt HTML, CSS, JS, iar în ultimii ani, sunt preferate framework-uri ca Angular, Vue și React. Pentru partea de server, avem limbaje precum Java, C# (.NET Core), iar serverul de date poate să fie bazat pe SQL, cum ar fi MySQL, sau pe NoSQL, cum ar fi MongoDB.[1]

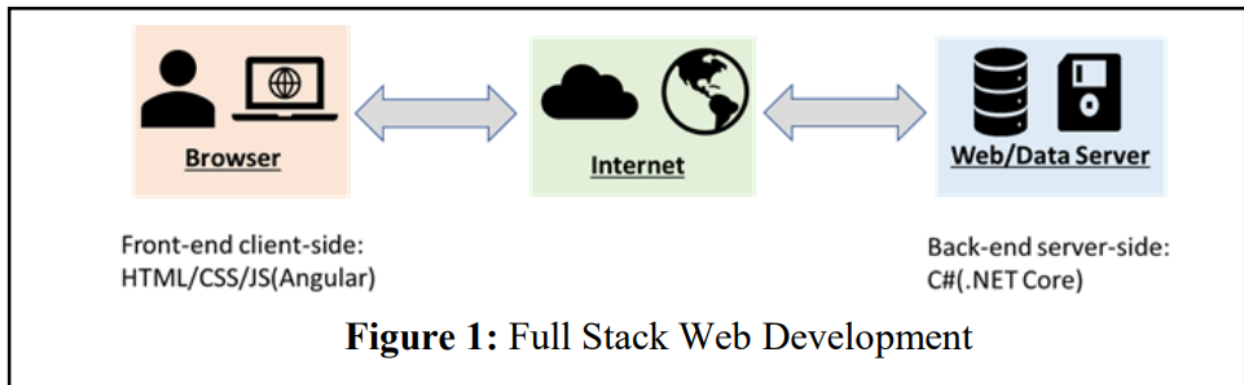


Figura 2.1. Schema dezvoltării web full-stack [1]

II.2 Comunicare prin cereri HTTP

HTTP este acronimul pentru HyperText Transfer Protocol, care este un protocol de internet foarte utilizat ce permite părții de server să comunice cu cea de client. Acest protocol funcționează în așa fel încât un client trebuie să facă o cerere pentru a efectua o anumită sarcină, iar această cerere este trimisă către back-end pentru a fi procesată. După procesare, partea de server trimite clientului un anumit răspuns cu privire la cererea făcută. Există o gamă largă de astfel de cereri pe care un client le poate face către server prin HTTP. Mai departe sunt prezentate câteva dintre acestea, care au fost folosite în procesul de dezvoltare a aplicației:

- Metoda GET – este cea mai comună și este folosită pentru a extrage date dintr-o resursă. În cazul meu, este întrebuințată pentru a afișa informații despre utilizatori și animale de companie.
- Metoda POST – foarte utilizată, aceasta se folosește atunci când clientul solicită crearea unei noi resurse. Astfel, eu creez conturi pentru utilizatori și adaug animăluțe spre adopție.
- Metoda PUT – este folosită pentru a actualiza o resursă deja existentă, prin înlocuirea acesteia cu o nouă resursă generată. Cu această metodă, un utilizator poate edita informațiile despre un animăluț pe care îl are postat pentru adopție.
- Metoda DELETE – este folosită pentru a șterge o anumită resursă printr-un URL specific. Astfel, în cazul meu, dacă un utilizator nu mai dorește să dea spre adopție un animal, acesta poate fi șters din aplicație. [2]

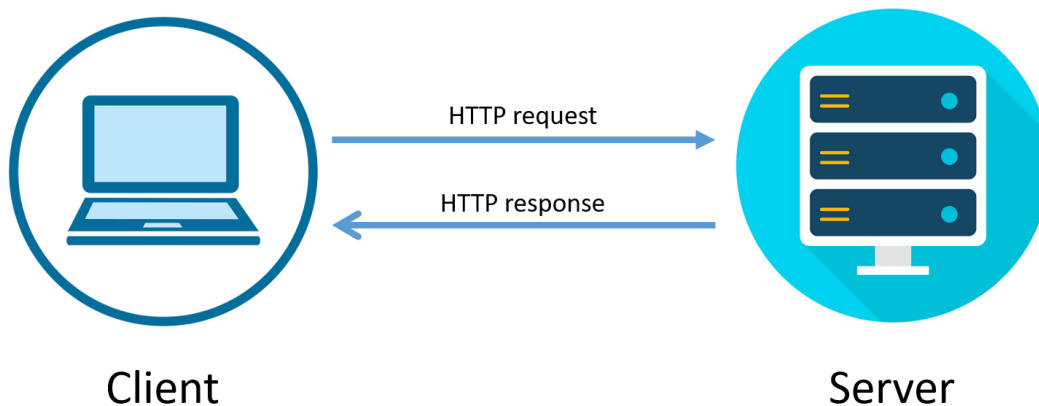


Figura 2.2 Comunicarea Client-Server [[Sursă figură](#)]

III. Tehnologiile folosite

III.1 Limbaje de programare

III.1.1 Java

Java este un limbaj de programare orientat pe obiecte, bazat pe clase, concurrent, securizat și de uz general. Este o tehnologie robustă, utilizată pe scară largă. Java a fost dezvoltat în anul 1995 de către compania Sun Microsystems, care este acum o subsidiară a Oracle. [3]

Acesta este conceput pentru a maximiza portabilitatea. Codul sursă este compilat în Java bytecode, care sunt concepute pentru a rula pe o mașină virtuală Java. Bytecodurile sunt un limbaj mașină pentru o mașină abstractă, executat de către o mașină virtuală pe fiecare sistem care suportă limbajul de programare Java. De asemenea, și alte limbaje pot fi compilate în Java Bytecodes. Codul poate fi executat cu un nivel corespunzător de protecție pentru a preveni ca autorii de clase neatenți sau rău intenționați să dăuneze sistemului. Nivelul de încredere poate fi ajustat în funcție de sursa de proveniență a bytecodurilor, de pe discul local sau din rețeaua protejată, pot fi mai de încredere decât bytecodes preluate de pe mașini arbitrare din altă parte a lumii. [4]

Working of JVM

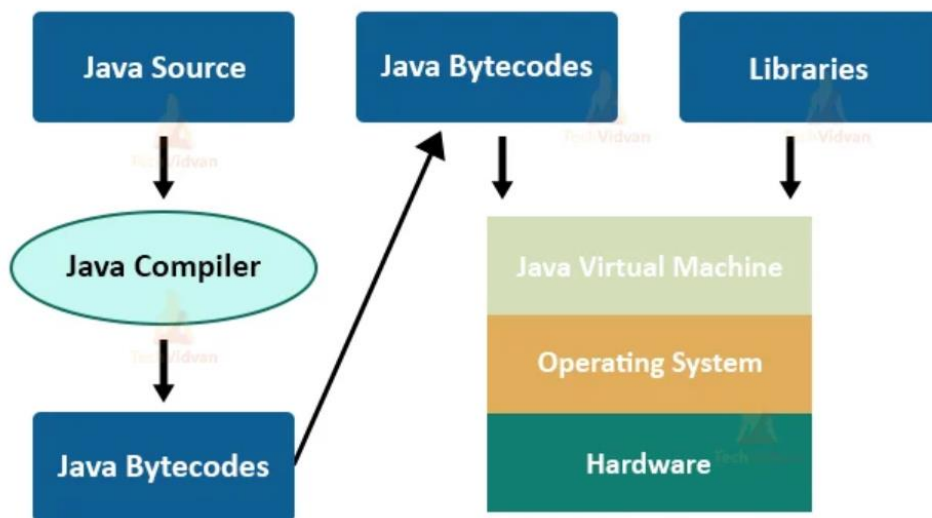


Figura 3.1.1 Funcționarea Java Virtual Machine [[Sursă figură](#)]

De ce am ales Java?

Java nu este greu de învățat, fiind proiectat pentru a fi ușor de utilizat, deci ușor de scris, de compilat și depanat. Din proprie experiență, în mediul academic am intrat în contact cu C++, C#, Python și Java, dar acesta din urmă mi-a plăcut cel mai mult. Pe parcursul liceului folosisem doar C++ și nu credeam că mi-ar fi atât de ușor să trec peste obișnuința de a îl folosi. Totuși după ce am întâlnit Java, nu m-aș mai întoarce la a programa așa mult în C++ cum făceam înainte.

Fiind orientat pe obiecte, permite crearea de programe modulare și reutilizarea codului. Un alt avantaj semnificativ este faptul că un program poate rula pe sisteme diverse, fapt efectuat pe două nivele, cel de sursă și cel binar, fiind un limbaj independent de platformă. [3]

Este open source, având un sprijin enorm din partea comunității, existând zeci de milioane de dezvoltatori. Astfel este unul dintre cele mai utilizate limbaje, având o cerere mare pe piața locurilor de muncă. De asemenea, fiind apropiat ca limbaj de C++ și C#, este ușor de trecut de la un limbaj la celălalt. [5]

III.1.2 TypeScript

TypeScript este un limbaj de programare care se bazează pe JavaScript și care oferă avantaje la scară largă. Acesta adaugă sintaxă în plus limbajului JavaScript pentru a susține o înglobare puternică a acestuia cu editorul. Permite descoperirea timpurie a erorilor în editor. Codul TypeScript se convertește în JavaScript și se execută oriunde se execută JavaScript: într-un browser, pe Node.js sau Deno și în aplicații. TypeScript înțelege JavaScript și folosește interferența tipurilor pentru a oferi instrumente excelente fără cod suplimentar. [6]

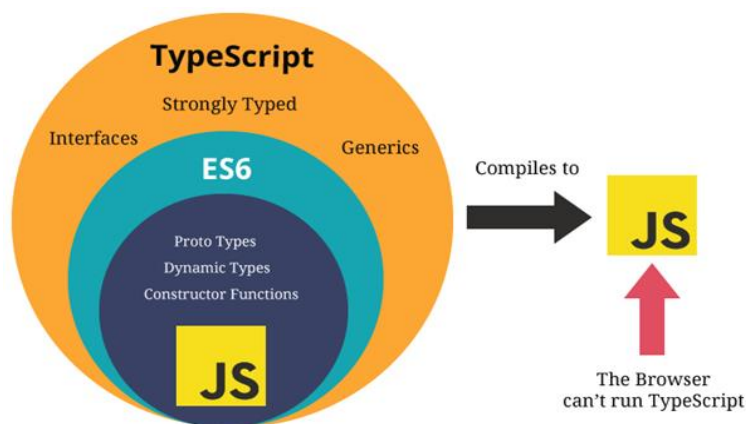


Figura 3.1.2 JavaScript inclus în TypeScript [[Sursă figură](#)]

A fost votat al doilea cel mai iubit limbaj de programare în sondajul „Stack Overflow 2020 Developer survey” și a primit premiul pentru "Cea mai adoptată tehnologie", pe baza creșterii de la an la an. TypeScript a fost utilizat de 78% dintre respondenții din „2020 State of JS”, iar 93% dintre aceștia au spus că l-ar mai utiliza din nou. [6]

De ce am ales TypeScript?

Această tehnologie a fost studiată de către mine în anii de studiu. Faptul că ajută la descoperirea rapidă a erorilor, chiar din editor, m-a ajutat să nu pierd timp ulterior cu remedierea acestora. Dar așa, fiind semnalate de la început, mi s-a părut un lucru foarte util. De asemenea, am putut să verific mereu tipurile variabilelor, parametrilor și a valorilor returnate.

III.2 Frameworks

III.2.1 Spring și Spring Boot

Spring Framework este o platformă Java care pune la dispoziție un suport de infrastructură cuprinzător pentru crearea de aplicații Java. Spring permite crearea de aplicații din “plain old Java objects” (POJOs) și aplicarea în mod neinvaziv de servicii de întreprindere. Exemple ale avantajelor folosirii Spring:

- O metodă Java se poate executa într-o tranzacție a bazei de date, fără a utiliza API-uri specifice tranzacțiilor.
- O metodă Java locală se poate transforma într-o procedură la distanță (remote), fără a fi nevoie de utilizare de remote API.
- O metodă Java locală se poate transforma într-o operațiune de gestionare fără a fi necesară utilizarea API-urilor JMX.
- O metodă Java locală se poate transforma într-un manager de mesaje fără a utiliza API-uri JMS. [7]

Spring Boot ușurează crearea de aplicații independente, bazate pe Spring, de nivel de producție, care se pot "rula pur și simplu". Caracteristici:

- Se pot crea aplicații Spring autonome.
- Tomcat, Jetty sau Undertow se pot integra direct, nu este necesar să fie implementate fișiere WAR.

- Furnizează dependențe necesare pentru început, pentru a simplifica configurația de build.
- Configurează automat atât biblioteci Spring, cât și altele de la terțe părți.
- Nu este nevoie nici de generare de cod, nici de vreo cerință de configurare XML.[8]

De ce am ales Spring și Spring Boot?

Deși nu a fost timp pentru a include aceste framework-uri în programa de la facultate, am ales să le studiez, deoarece ușurează scrisul de cod în Java. Am folosit adnotări pentru a utiliza în aplicație arhitectura MVC (Model, View, Controller). Spring conține și Java Database Connectivity (JDBC), prin care m-am putut conecta ușor la o bază de date MySQL.

De asemenea, m-am folosit de Spring Security și JWT (JSON Web Token) pentru a efectua autentificarea în aplicație, inclusiv pentru codificarea parolelor care sunt reținute mai departe în cadrul bazei de date.

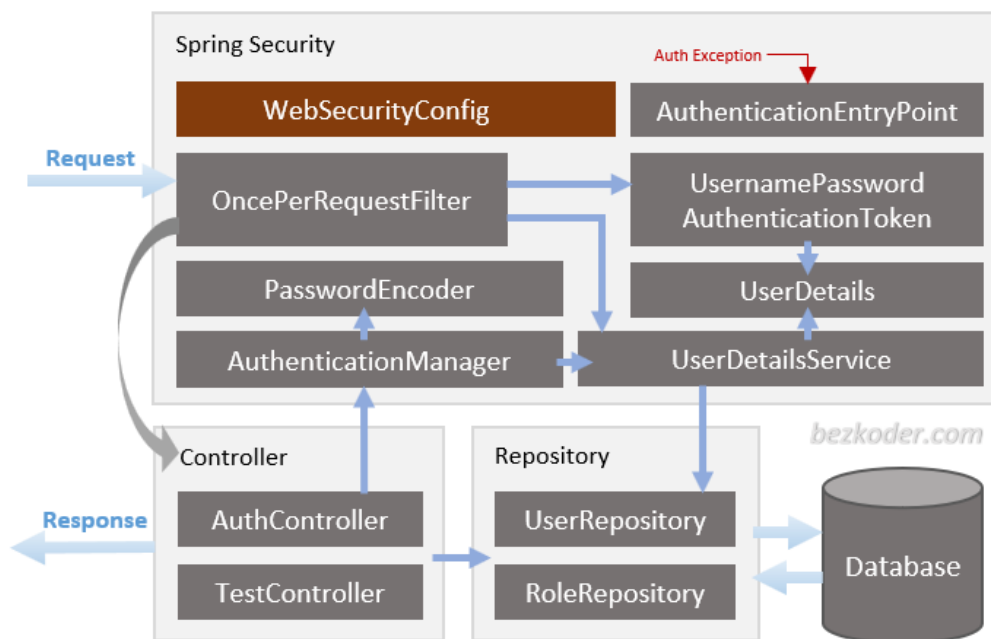


Figura 3.2.1 Schema back-end [[Sursă figură](#)]

III.2.2 Angular

Angular este o platformă și un framework de dezvoltare web pentru front-end, open-source, bazat pe TypeScript. Acesta a fost constituit pentru a aduce structură și coerență dezvoltării web, oferind în același timp o modalitate de a crea rapid aplicații web scalabile și ușor

de întreținut. Spre deosebire de începuturi, browserele au evoluat, iar unele dintre problemele pe care le rezolva Angular, nu mai sunt la fel de relevante în prezent.

Angular oferă câteva avantaje semnificative, având o structură comună, care ajută echipele de dezvoltatori. Acesta dă posibilitatea dezvoltării de aplicații mari, într-un mod ușor de întreținut. Angular permite contruirea de componente declarative proprii, care pot aduce laolaltă funcționalitatea împreună cu logica de randare, în bucăți mici și reutilizabile. De asemenea, este posibilă scrierea de servicii modulare, care mai apoi să fie injectate acolo unde este nevoie de ele. [9]

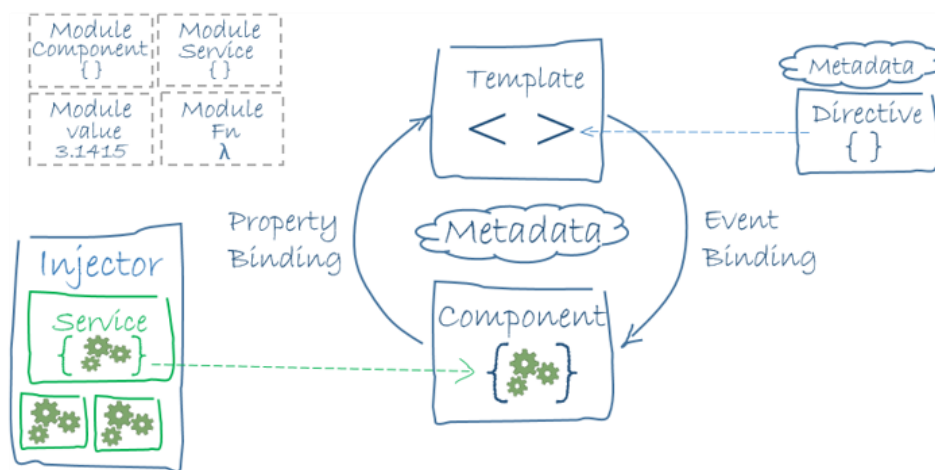


Figura 3.2.2 Conectarea pieselor în Angular [[Sursă figură](#)]

De ce am ales Angular?

Se bazează pe TypeScript (motivele pentru această alegere sunt precizate anterior) și am avut ocazia să îl folosesc practic la un opțional în cadrul facultății, astfel încât nu mi-a fost așa greu să îl utilizez acum. Mi se pare ușor de înțeles și de folosit.

III.2.3 Hibernate

Hibernate este un framework Java open-source, un instrument de corespondență obiect-relațională (ORM). Expertiza sa constă în implementarea JPA (Java Persistence API) pentru persistența datelor. Acesta se ocupă în mod expert de implementările interne; inclusiv - scrierea unei interogări pentru operațiile CRUD sau stabilirea unei conexiuni cu bazele de date, și altele asemenea. Acest framework este, în plus, popular pentru capacitatea sa de a depăși dependența de baze de date cu care se confruntă JDBC. [10]

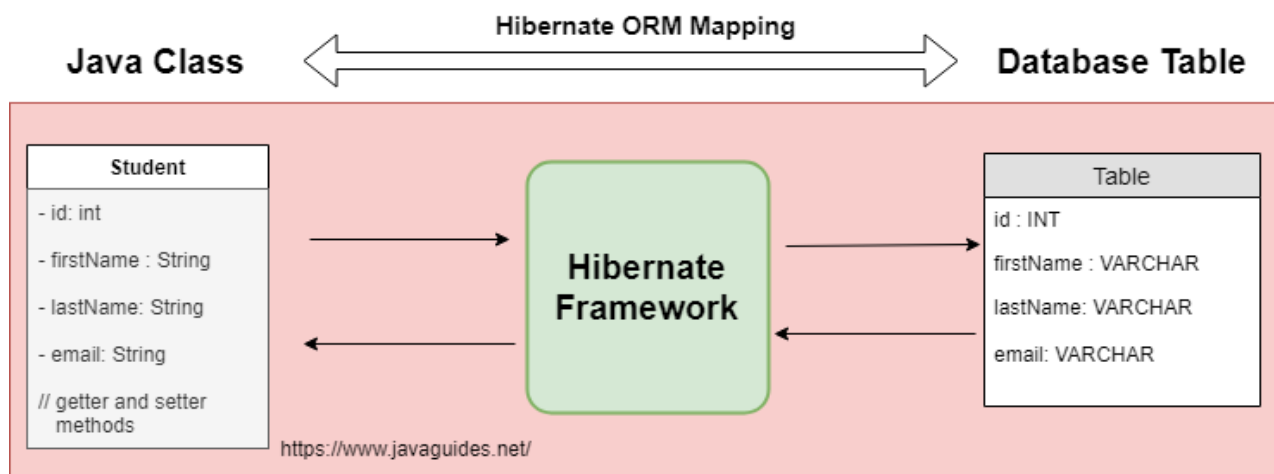


Figura 3.2.3 Arhitectura framework-ului Hibernate [[Sursă figură](#)]

De ce am ales Hibernate?

Cu ajutorul acestui instrument nu a fost nevoie să scriu cod SQL. Tabelele sunt create automat în baza de date, folosind adnotarea @Entity. De asemenea, modificările tabelor existente, cum ar fi adăugarea unei coloane, au fost realizate de Hibernate. Pentru acest lucru a fost nevoie doar să adaug următoare proprietate în aplicație: `spring.jpa.hibernate.ddl-auto=update`. În plus, acest instrument se ocupă în mod intern de legarea (binding) parametrilor și de executarea instrucțiunilor SQL, ceea ce ajută la apărarea împotriva anumitor atacuri, cum ar fi injecție SQL.

III.2.4 Bootstrap

Bootstrap este o serie de instrumente gratuite și open-source. Este cel mai utilizat framework HTML, CSS și JavaScript pentru crearea de site-uri și aplicații web adaptabile (responsive). Cu ajutorul Bootstrap, stilul oricărei pagini web poate fi modificat cu ușurință, de exemplu stilul fontului, culoarea textului, culoarea de fundal etc. Este o modalitate mai rapidă și mai ușoară de dezvoltare web, prin care se creează pagini web independente de platformă. [11]

De ce am ales Bootstrap?

Cu toate că îmi place să mă joc cu design-ul paginilor, câteodată timpul nu este de partea mea și trebuie să recurg la trucuri. Bootstrap mi s-a părut ideal pentru a aranja paginile web într-un mod cât mai plăcut fără extrem de multe linii de CSS. De asemenea, foarte util în cea ce privește responsiveness-ul elementelor din pagină. Cel mai bun exemplu este la bara de navigare, unde cu

câteva clase de Bootstrap, am rezolvat repede ceva ce m-ar fi încetinit dacă făceam acest lucru manual în CSS.

III.3 MySQL

MySQL, cel mai folosit sistem de gestionare a bazelor de date SQL, aparținând de compania Oracle Corporation. Bazele de date MySQL sunt relaționale. O astfel de bază de date stochează datele în tabele diferite, pentru a evita stocarea datelor într-o singură locație de depozitare. Structurile bazelor de date sunt constituite cu scopul de a optimiza. Modelul logic, cu obiecte precum baze de date, tabele, vizualizări, rânduri și coloane, oferă un mediu de programare flexibil. Software-ul MySQL este open-source, cu o viteză deosebită, prezintă siguranță în funcționare și este facil de utilizat. Este un sistem client/server care constă într-un server SQL multithreaded care suportă diferite back-end-uri, mai multe programe client și biblioteci diferite, instrumente administrative și o gamă largă de interfețe de programare a aplicațiilor (API). [12]

De ce am ales MySQL?

Cele două materii care vizează bazele de date urmate în cadrul facultății au fost printre preferatele mele. Acolo am lucrat foarte mult cu MySQL și era alegerea naturală pentru mine. Pentru vizualizare și verificări am folosit MySQL Workbench, un instrument ușor de utilizat.

IV. Arhitectura aplicației

IV.1 Arhitectura MVC (model-view-controller)

Acest model, conceput pentru a facilita separarea reală a conținutului de prezentarea acestuia, permite dezvoltarea de aplicații care pot adapta și personaliza prezentări în mod dinamic, bazându-se pe preferințele utilizatorului, a capacităților dispozitivului, a regulilor de afaceri și a altor constrângeri. Modelul de date nu este legat de un singur format de prezentare, care ar putea să limiteze flexibilitatea aplicației. O abordare bazată pe MVC, face posibilă combinarea flexibilității cu o divizare adecvată a responsabilităților. [13]

Pe partea de back-end, am folosit framework-ul Spring Web cu arhitectura model-view-controller (MVC), care se bazează pe DispatcherServlet, care distribuie cererile. Handler-ul implicit se folosește de anumite adnotări, cum ar fi `@Controller` și `@RequestMapping`, oferind o multitudine de metode adaptabile de manipulare. Odată cu apariția Spring 3.0, adnotarea `@Controller` permite dezvoltarea de site-uri și aplicații web RESTful, prin intermediul opțiunii `@PathVariable` și nu numai. Fluxul de lucru de procesare a cererilor din DispatcherServlet Spring Web MVC este ilustrat în următoarea diagramă. Cititorul cunoscător de modele va recunoaște că DispatcherServlet este o expresie a modelului de proiectare "Front Controller" (este un model pe care Spring Web MVC îl împarte cu alte framework-uri web de top). [14]

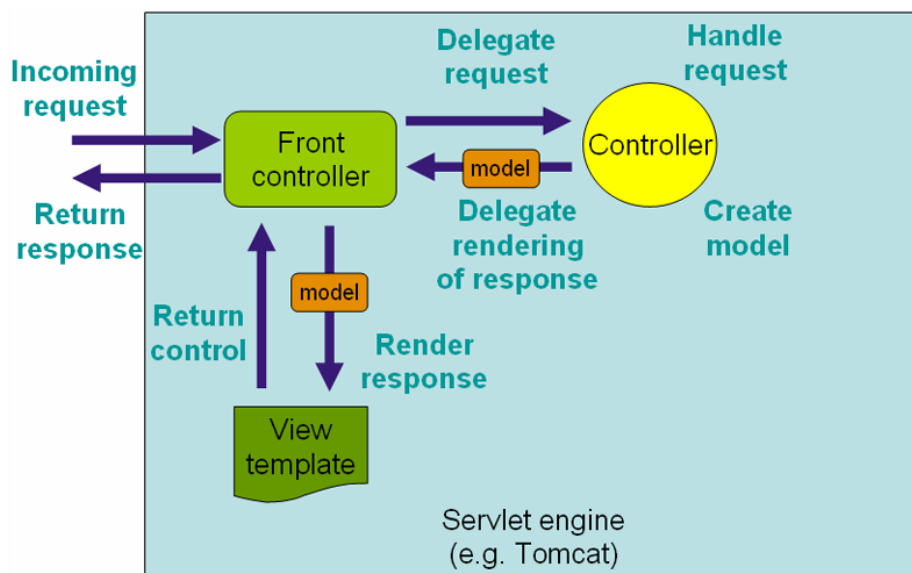


Figura 4.1 Flux procesare cereri din DispatcherServlet [14]

În cadrul aplicației Petkindess, am următoarea structură a fișierelor, dintre care am scos în evidență controllerele, modele, repository-urile și serviciile. Datorită adnotărilor specifice Spring (@Controller, @Entity, @Repository și @Service), mi-a fost ușor să le conectez și să le utilizez într-o manieră clară, împărțind responsabilitățile. (Figura 4.2)

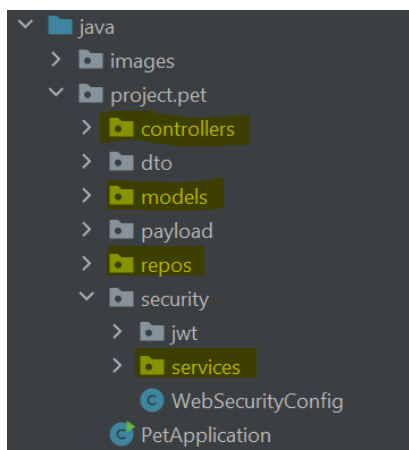


Figura 4.2

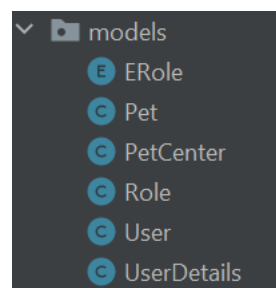


Figura 4.3

Primul pas, a fost definirea modelelor, în funcție de cum am gândit baza de date, cu tabele și relații dintre acestea (Figura 4.3). Spring conține adnotări pentru a pune toate restricțiile necesare modelării tabelelor, astfel încât acestea sunt create automat cu ajutorul framework-ului Hibernate (III.2.3). Acest lucru este un avatanj din punct de vedere al timpului.

```
@Entity
@Table(name = "users",
    uniqueConstraints = {
        @UniqueConstraint(columnNames = "username"),
        @UniqueConstraint(columnNames = "email")
    })
public class User {
    2 usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    3 usages
    @NotBlank
    @Size(max = 20)
    private String username;

    3 usages
    @NotBlank
    @Size(max = 50)
    @Email
    private String email;
```

Figura 4.4 Exemple de adnotări pentru modelarea tabelelor

Pentru crearea repository-urilor, am folosit extinderea interfeței JpaRepository (Java Persistence API), care conține API pentru operațiile CRUD clasice. De asemenea, conține și funcții specifice, de exemplu filtrarea după categorie și gen a animalelor de companie.

```
@Repository
public interface PetRepository extends JpaRepository<Pet, Long> {

    1 usage  🧑 biancavoicu
    List<Pet> findByCategory(String category);

    1 usage  🧑 biancavoicu
    Collection<? extends Pet> findByCategoryAndGender(String category, String gender);

    1 usage  🧑 biancavoicu
    Collection<? extends Pet> findByGender(String gender);

    1 usage  🧑 biancavoicu
    List<Pet> findByIdUser(String idUser);
}
```

Figura 4.5 Exemplu extindere interfeță JpaRepository

Pentru partea de servicii, care sunt apelate de către Controller, există o interfață unde am anunțat funcțiile necesare pentru a manipula datele, apoi o clasă (@Service) care implementează funcțiile din această interfață. De exemplu, în cazul PetService, poza unui animal de companie este manipulată separat, pentru a reține în baza de date doar URL-ul respectivei fotografii. În clasa de implementare, mă folosesc de repository, pentru a utiliza operațiile CRUD, care este injectat automat ca dependență cu ajutorul adnotării @Autowired. La fel sunt și serviciile pentru Mail și Documente, fiind necesare pentru conectarea funcționalităților.

```
3 usages  1 implementation  🧑 biancavoicu *
public interface PetService {

    1 usage  1 implementation  new *
    void adoptPet(Long petId, String receiverEmail,
        DocumentGenerationInputDto contract);

    2 usages  1 implementation  🧑 biancavoicu
    void addPetImage(MultipartFile file);

    1 implementation  🧑 biancavoicu
    Collection<Pet> list();

    1 implementation  🧑 biancavoicu
    Pet get(Long id);

    1 usage  1 implementation  new *
    void update(Pet pet);
}
```

Figura 4.6 interfața PetService

```
@Service
@Slf4j
public class PetServiceImpl implements PetService{

    7 usages
    @Autowired
    PetRepository petRepository;

    1 usage
    @Autowired
    DocumentGenerationService documentGenerationService;

    1 usage
    @Autowired
    MailService mailService;

    no usages  🧑 biancavoicu
    @Override
    public Pet create(Pet pet) {
        log.info("Saving new pet: {}", pet.getName());
        return petRepository.save(pet);
    }
}
```

Figura 4.7 implementarea interfeței PetService

În controllere se găsesc endpoint-urile necesare pentru apelurile din front-end, semnalizate cu adnotări specifice: `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`. Repository-urile și serviciile necesare sunt injectate automat ca dependențe (`@Autowired`). La controllere am adăugat și adnotarea `@CrossOrigin`, necesară pentru securitatea gestionării resurselor implementate în browserele web.

```
@CrossOrigin(origins = "http://localhost:4200", allowCredentials="true")
@RequestMapping("/petapp")
@RestController
public class PetController{
    14 usages
    @Autowired
    PetRepository petRepository;

    1 usage
    @Autowired
    UserRepository userRepository;

    6 usages
    @Autowired
    PetService petService;

    no usages  ⓘ biancavoicu *
    @GetMapping("/pets")
    public ResponseEntity<List<Pet>> getAllPets(
        @RequestParam(required = false) String category,
        @RequestParam(required = false) String gender
    ) {
        // ...
    }
}
```

Figura 4.8 Exemplu adnotări controller

Pe partea de front-end, am menținut același nivel de organizare, cu modele și servicii, componentele Angular (în cadrul fișierelor TypeScript) apelând funcțiile din servicii. În acestea, este realizată legătura cu back-end-ul, existând funcții corespunzătoare endpoint-urilor.

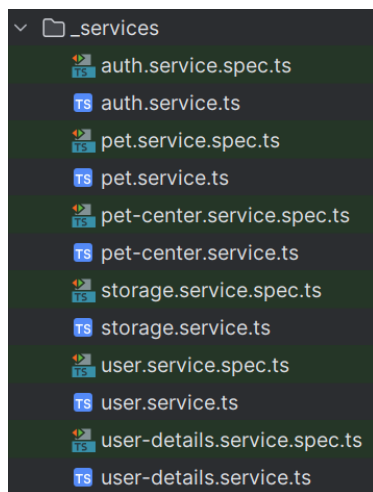


Figura 4.9 Servicii front-end

```
no usages  ⓘ biancavoicu
create(data: any, file: any): Observable<any> {
    return this.http.post(baseUrl, data, file);
}

2 usages  ⓘ biancavoicu
update(id: any, data: any): Observable<any> {
    return this.http.put( url: `${baseUrl}/${id}`, data);
}

1 usage  ⓘ biancavoicu
delete(id: any): Observable<any> {
    return this.http.delete( url: `${baseUrl}/${id}`);
}
}
```

Figura 4.10 Exemple funcții din servicii

IV.2 Baza de date

Pentru gestionarea bazei de date, am folosit instrumentul MySQL Workbench, care a fost suficient pentru necesitățile mele în cadrul aplicației. A fost util pentru a face verificări manuale ale bazei de date, pentru a mă asigura că operațiunile efectuate de framework-ul Hibernate au efectul dorit.

Structura bazei de date este compusă din următoarele entități: *Users* (datele necesare autentificării în aplicație), *Roles* (“ROLE_USER”, “ROLE_ADMIN”), *User_details* (informațiile speciale necesare adopției), *Pets* (datele complete ale animalelor postate spre adopție) și *PetCenters* (date ale adăposturilor de animale). Am dorit să rețin doar datele absolut necesare și să nu complic fără motiv aplicația, din punctul de vedere al utilizatorilor.

IV.3 Cazuri de utilizare (Use cases)

În cadrul aplicației Pet Kindness, există trei tipuri de utilizatori:

- Administrator (“ROLE_ADMIN”) (autentificat), care poate gestiona centrele de adopție partenere, pe lângă funcțiile unui utilizator obișnuit.
- Utilizator obișnuit (“ROLE_USER”) (autentificat), care poate vizualiza animăluțele, poate posta și adopta. Pentru a nu complica procesul de adopție, am considerat că este ideal ca un utilizator să poată face toate aceste acțiuni având un singur rol.
- Utilizator neautentificat/ neînregistrat, care poate vizualiza anumite pagini.

Acțiunile posibile ale unui utilizator neautentificat (guest user) :

- Poate vizualiza pagina de „Acasă”.
- Poate dona în sprijinul animăluțelor.
- Poate vizualiza pagina de „Adopții”.
- Poate vizualiza pagina de „Centre partenere”.
- Poate vizualiza pagina de „Login”.
- Poate vizualiza pagina de „Sign up”.

Acțiunile posibile ale unui utilizator obișnuit (autentificat):

- Poate vizualiza toate paginile, mai puțin cea care conține statistici.
- Poate dona în sprijinul animalelor.
- Poate posta animale de companie pentru adopție.
- Poate edita datele animalușului deja postat (din pagina de profil a utilizatorului).
- Poate șterge animalușul din lista de adopții (din pagina de profil a utilizatorului).
- Poate adopta un animal de companie, prin trimiterea unei cereri de adopție.
- Poate primi pe mail cererea de adopție pentru un animal postat de către utilizator.

Acțiunile posibile ale unui administrator (autentificat), pe lângă cele anterior menționate:

- Poate adăuga centre de adopție partenere.
- Poate edita datele centrelor de adopție partenere.
- Poate șterge un centru de adopție partener din lista existentă.
- Poate vizualiza pagina „Statistici”.

V. Prezentarea aplicației

V.1 Pagina principală („Acasă”)

În imaginile de mai jos, este afișată pagina pe care o vede un utilizator neautentificat/neînregistrat. Acesta poate vedea bara de navigare cu posibilitatea de a ajunge la paginile: *Acasă*, *Adopții*, *Centre partenere*, *Login* și *Sign up*. Pentru a fi ușor de ajuns la pagina destinată adopțiilor, am adăgat pe banner un buton care să redirecționeze utilizatorul.

Pe această pagină sunt precizați și sponsorii aplicației (fictivi, fiind momentan un proiect local). În subsolul paginii se regăsesc informațiile de contact ale echipei Pet Kindness. În plus, există pentru orice utilizator și posibilitatea de a dona pentru animăluțele din adăposturi.

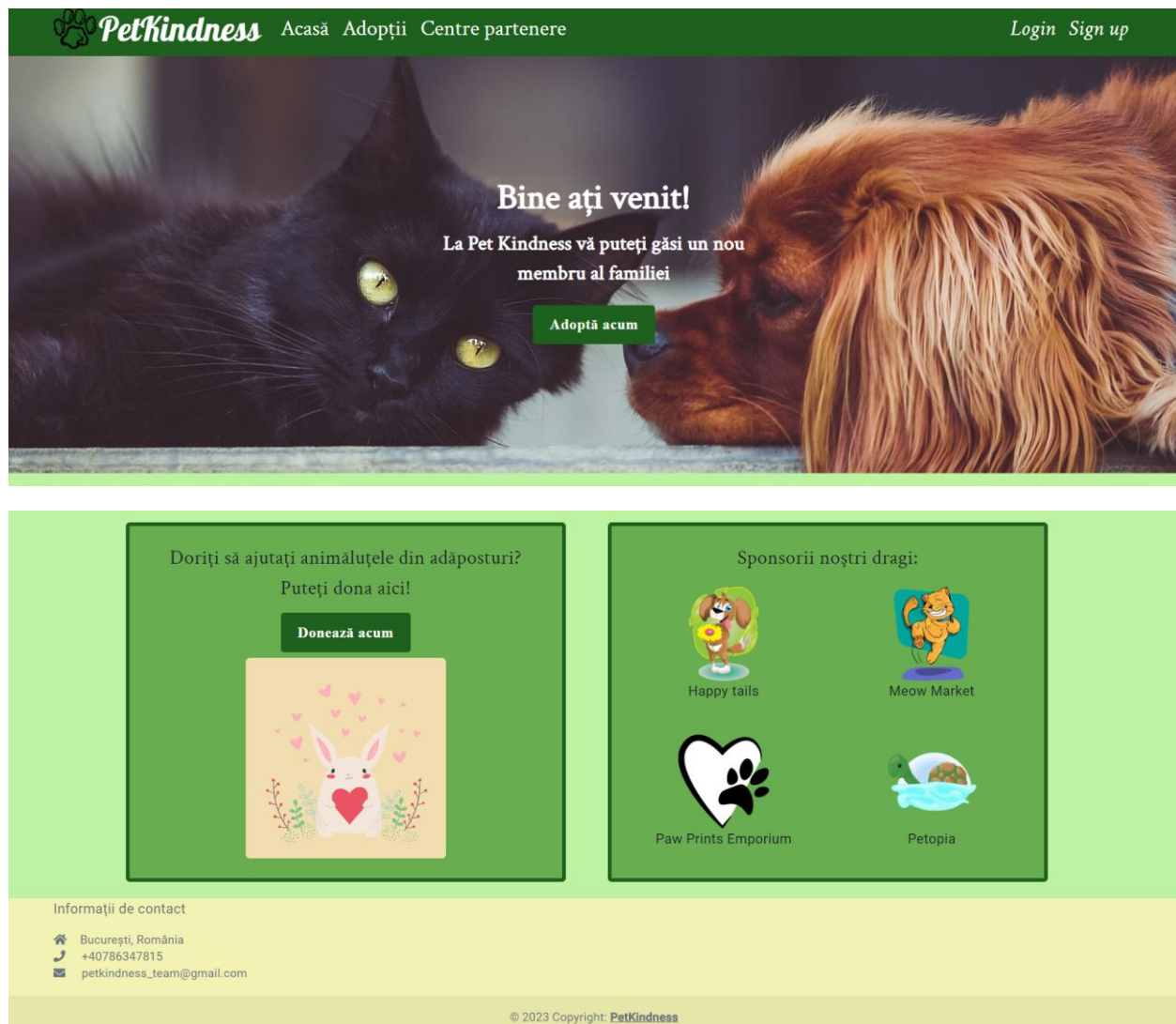


Figura 5.1.1 Pagina de start

În situația în care un utilizator apasă pe butonul „Donează acum”, se deschide o nouă pagină. Există 3 opțiuni prestabilite de valoare a donației, dar se poate alege și varianta „Other” unde se poate introduce o altă sumă. Se poate opta și pentru donație recurentă lunară. Plata se poate face prin PayPal sau prin card de credit/debit.

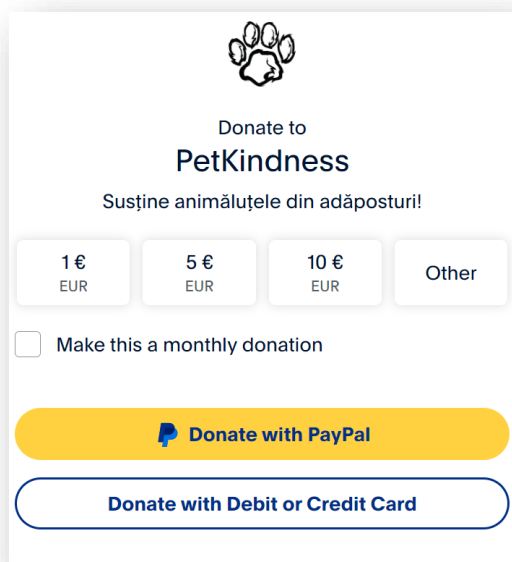


Figura 5.1.2 Pagina de donații

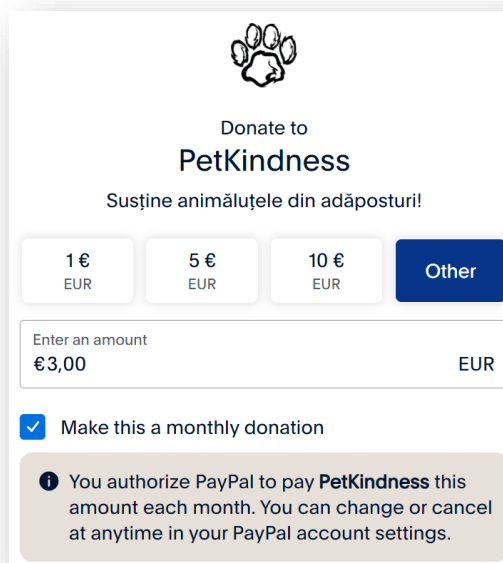


Figura 5.1.3 Donație lunară

În cazul în care un utilizator se conectează, bara de navigație își schimbă butoanele din dreapta. Apare numele de utilizator, care conduce către pagina profilului acestuia, și butonul de „Logout”.



Figura 5.1.4 Bara de navigație pentru utilizator autentificat

V.2 Pagina de înregistrare („Sign up”)

Pentru a se înregistra, un utilizator nou trebuie să își aleagă un nume de utilizator care nu este deja folosit (lungime minimă 6 caractere, lungime maximă 30), o adresă de email (validă) cu care să nu existe deja cont și o parolă (lungime minimă 6 caractere, lungime maximă 40). În cazul în care aceste constrângeri sunt încălcate, câte un mesaj corespunzător, scris cu roșu, apare sub câmpul la care există o problemă. De asemenea, dacă la apăsarea butonului de Sign up se constată că numele de utilizator sau email-ul există deja, devine vizibil câte un mesaj de eroare.

Register
Bine ați venit!

Nume de utilizator*

Email*

Password*

[Sign Up](#)

Aveți deja cont? [Login](#)

Figura 5.2.1 Pagina de înregistrare

V.3 Pagina de autentificare („Login”)

Odată ce un cont nou a fost creat, utilizatorul este redirecționat către pagina de autentificare. Atunci când utilizatorul introduce date de conectare incorecte, apare un pop-up care anunță eroarea.

Login
Bine ați revenit!

Username*

Password*

[Login](#)

Nu aveți cont? [Register](#)

Figura 5.3.1 Pagina de autentificare

V.4 Pagina cu animalele spre adopție („Adopții”)

Un utilizator autentificat sau neautentificat are acces la această pagină, în care sunt prezentate câteva dintre informațiile disponibile (*Nume, Gen, Vârstă, Rasă, Locație*), mai multe detalii găsindu-se pe pagina fiecărui animaluț, care poate fi accesată dând click pe „cartea de vizită” a acestuia. Animalele pot fi filtrate în funcție de categorie (*Câini, Pisici, Păsări, Rozătoare, Alte animale*) și/sau gen (*masculin sau feminin*) (Figura 5.4.2).



Figura 5.4.1 Pagina anunțurilor pentru adopții

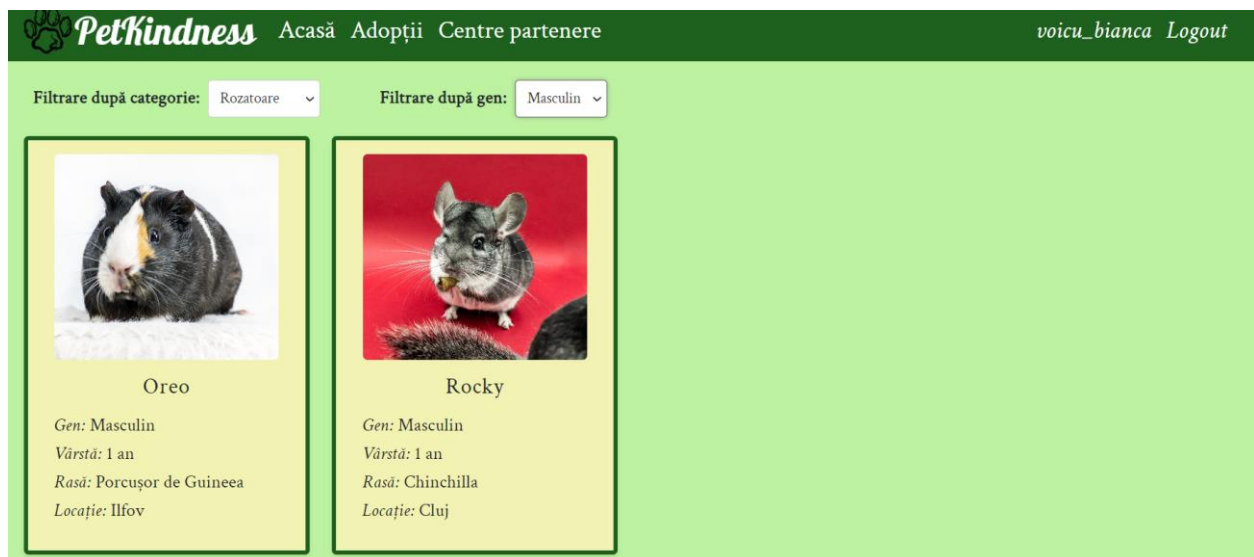


Figura 5.4.2 Filtrare după categorie și gen

V.5 Pagina unui animal spre adopție

După cum am precizat la pagina de „Adopții”, pagina unui animaluț este accesată dând click pe „cartea de vizită” a acestuia. Conține informații suplimentare, cum ar fi detalii despre sănătate sau alte detalii relevante pentru un viitor stăpân. În cazul în care utilizatorul care a adăugat animalul intră pe pagina acestuia, butonul „Adoptă” devine inactiv.



Figura 5.5.1 Pagina cu toate datele despre un animal

Apăsând butonul „Adoptă”, se dezvăluie o zonă de formular cu date personale suplimentare necesare pentru cererea de adopție (pentru un utilizator autentificat, altfel acesta este trimis către pagina de autentificare). Aceste date au validări specifice. Butonul de trimitere al cererii devine activ atunci când datele introduse sunt valide.

Pentru adopție sunt necesare următoarele detalii personale:

Nume complet *

Număr de telefon *

CNP*

Serie act identitate*

Număr act identitate*

Trimite cererea de adopție

Figura 5.5.2 Formularul cu datele pentru adopție

Dacă datele sunt corecte, este generat automat un contract cu datele din contul celui care inițiază adopția și trimis pe mail utilizatorului care a publicat anunțul. Astfel cei doi pot lua legătura, în mail-ul trimis existând datele de contact a celui ce dorește să adopte. Pentru a exemplifica trimiterea mesajului electronic cu atașamentul respectiv, am ales să folosesc platforma Mailtrap, care simulează un inbox real.

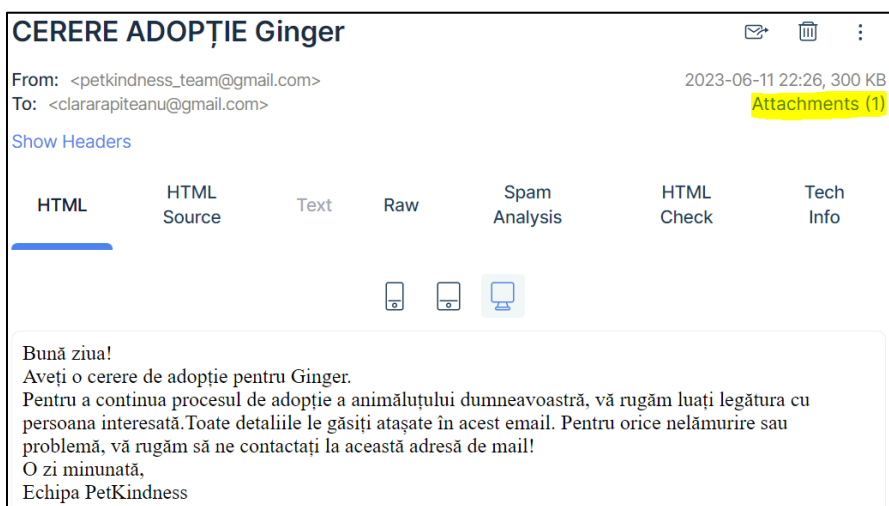


Figura 5.5.3 Exemplu mail cerere de adopție

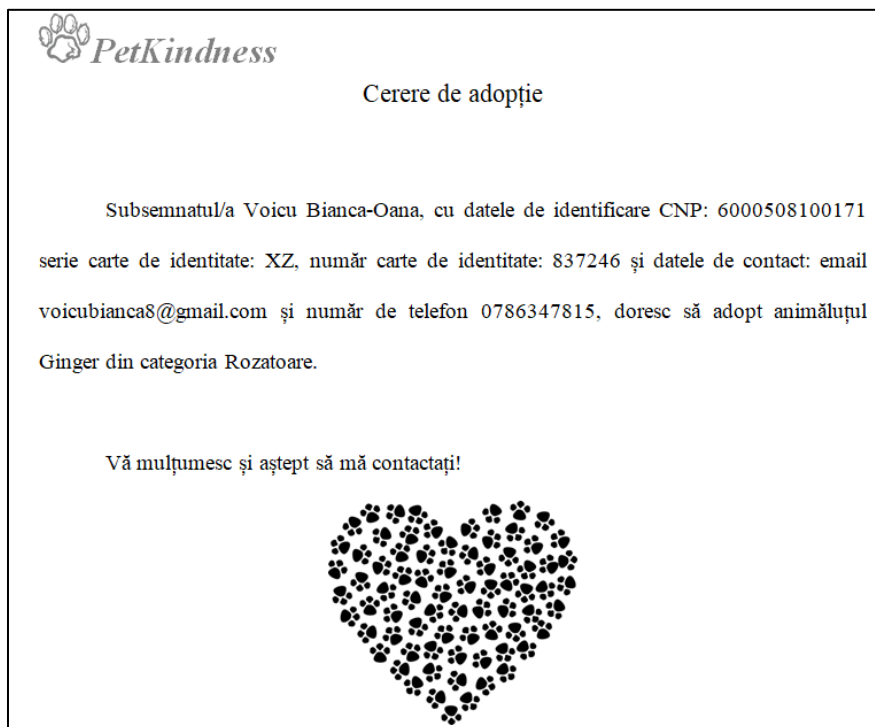


Figura 5.5.4 Exemplu mesaj din documentul generat

Pentru generarea documentului, se efectuează o copie a șablonului de cerere de adopție la care se schimbă extensia din .docx în .zip, se dezarchivează ceea ce s-a obținut și se modifică conținutul fișierului document.xml, astfel încât variabilele sunt înlocuite cu datele personale ale utilizatorului care dorește să adopte un animal. Apoi, fișierul respectiv este suprascris cu varianta modificată, se arhivează la loc și este creat un link de unde se poate descărca noul document. Codul folosit pentru realizarea generării automate a cererii de adopție personalizată:

```
public DocumentGenerationOutputDto generate(DocumentGenerationInputDto
contract) throws IOException {

    // Pasul 1 -> Fac o copie a fișierului docx original, dar o salvez cu
    // extensia .zip în loc de .docx
    File file =
    ResourceUtils.getFile("classpath:files/PET_ADOPTION_CONTRACT.docx");
    String originalTemplateFilePathName = file.getPath();
    String tempZipArchivePathName = String.format("%s%s%s",
    file.getParentFile().getPath(), File.separator,
    "PET_ADOPTION_CONTRACT_TEMP_ARCHIVE.zip");
    String tempUnzippedFolderPathName = String.format("%s%s%s",
    file.getParentFile().getPath(), File.separator,
    "PET_ADOPTION_CONTRACT_TEMP_FOLDER");
    String finalDocxFileName = String.format("%s.docx",
    UUID.randomUUID());
    String finalDocxFilePathName = String.format("%s%s%s",
    file.getParentFile().getPath(), File.separator, finalDocxFileName);

    Files.copy(Paths.get(originalTemplateFilePathName), Paths.get(tempZipArchivePat
hName), StandardCopyOption.REPLACE_EXISTING);

    // Pasul 2 -> Dezarhivăm arhiva
    this.unzipArchive(tempZipArchivePathName, tempUnzippedFolderPathName);

    // Pasul 3 -> Procesez fișierul document.xml localizat în subfolderul
    // word al arhivei extrase și înlocuiesc variabilele cu conținutul real
    String documentXmlFilePathName = String.format("%s%s%s%s%s",
    tempUnzippedFolderPathName, File.separator, "word", File.separator,
    "document.xml");

    String documentXmlContent =
    Files.readString(Paths.get(documentXmlFilePathName));

    for(Map.Entry<String, String> variableEntry :
    contract.variables.entrySet()) {
        documentXmlContent = documentXmlContent.replace("{ " +
        variableEntry.getKey() + " }", variableEntry.getValue());
    }

    // Acum pun conținutul înapoi în fișier (este suprascris cel vechi)
    Files.writeString(Paths.get(documentXmlFilePathName),
    documentXmlContent);
}
```

```
// Pasul 4 -> Arhivez la loc folderul modificat și îl salvez cu
extensia .docx, acesta fiind fișierul final
this.zipFolder(tempUnzippedFolderPathName,finalDocxFilePathName);

return new
DocumentGenerationOutputDto(String.format("http://localhost:8080/content/%s",
finalDocxFileName), new File(finalDocxFilePathName));
}
```

V.6 Pagina adăposturilor partenere („Centre partenere”)

Pentru un utilizator obișnuit (autentificat/neautentificat), pagina prezintă informații (Nume, Număr de telefon, Adresă) despre adăposturile de animale partenere:

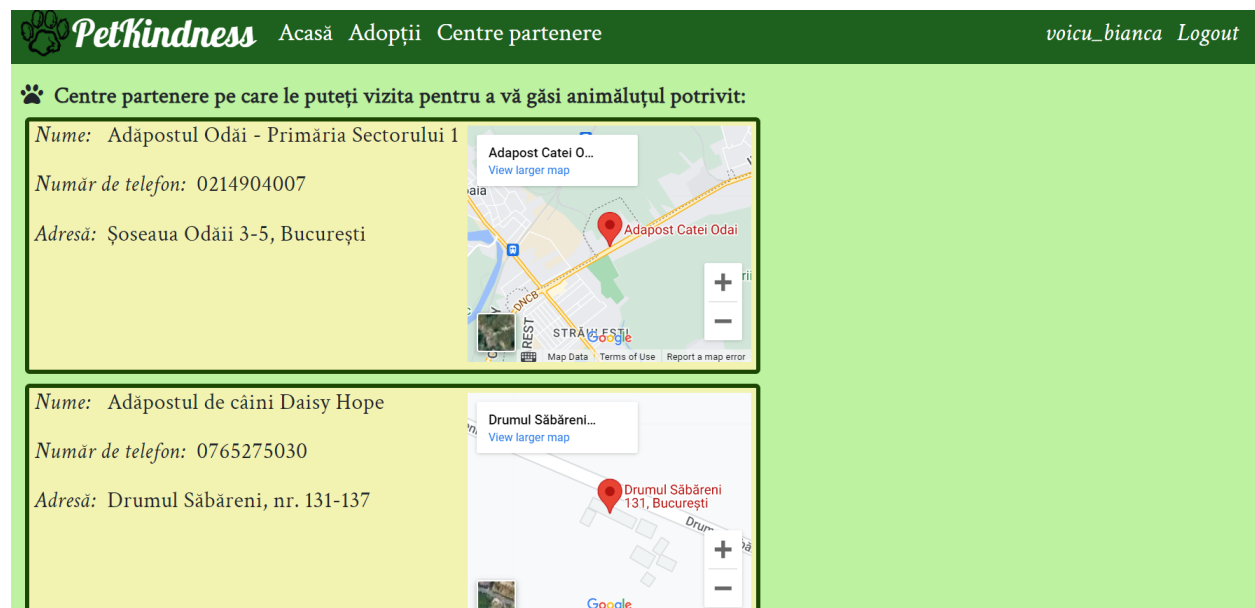


Figura 5.6.1 Pagina cu adăposturile de animale

Dar un cont de administrator, are în plus următoarele butoane: „Adaugă centru partener”, care deschide un formular de adăugare a unui nou adăpost, „Edit”, care deschide un formular pentru editarea detaliilor deja existente, și „Delete”, care șterge din listă adăpostul.

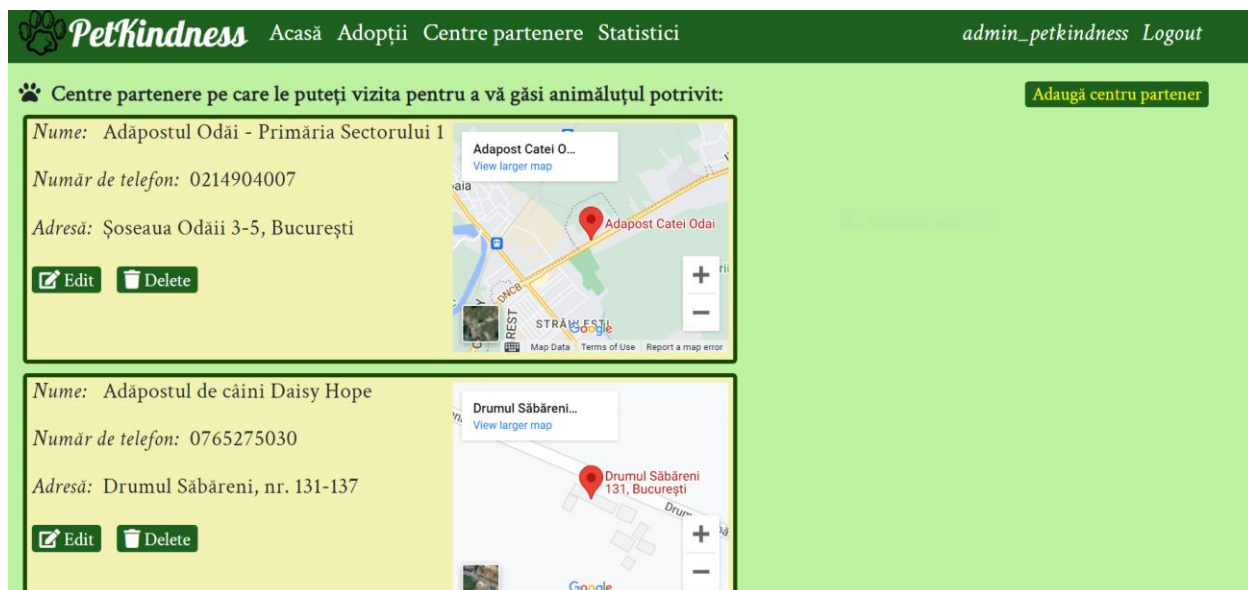


Figura 5.6.2 Pagina cu adăposturile de animale pentru administrator

Formularele menționate anterior se deschid în modale.

Adaugă centru partener

Nume:*

Număr de telefon:*

Adresă:*

Cod hartă: *

Figura 5.6.3 Formular adăugare centru

Editează informațiile

Nume:

Telefon:

Adresă:

Cod hartă:

Figura 5.6.4 Formular editare centru

V.7 Pagina de profil a utilizatorului

Această pagină este vizibilă doar pentru utilizatorii autentificați și afișează *Numele de utilizator*, *Email-ul* și link-uri către animalele date spre adopție de respectivul utilizator. Acesta are opțiunea de a edita informațiile despre animal sau de a-l șterge. În cazul în care un utilizator nu are animale date spre adopție, apare mesajul: „Nu ai adăugat încă animale spre adopție”.



Figura 5.7.1 Pagina de profil a unui utilizator

Modalul pentru editarea informațiilor despre un animaluț, permite și modificarea fotografiei acestuia.

Figura 5.7.2 Formularul de editare a datelor unui animaluț

V.8 Pagina de adăugare a unui animal spre adopție

Pe această pagină se ajunge din pagina de profil a unui utilizator, apăsând butonul „Adaugă animal pentru adopție”. Aici există un formular în care toate câmpurile sunt obligatorii, cu scopul de a furniza cât mai multe informații despre animalul de companie. Tipul de fotografii acceptate sunt *.png* și *.jpeg*.

Adăugare animaluț pentru adopție

Vă rugăm introduceți toate detaliile cerute în acest formular!

Nume*



Categorie*

Gen*

Vârstă*

Rasă* (daca nu se știe, scrieți Necunoscută)

Locație*



Locație*

Detalii despre sănătate*

Alte detalii*

Poză*

Choose File

No file chosen

Figurile 5.8.1 și 5.8.2 Formularul de adăugare a unui animal spre adopție

V.9 Pagina de statistici

Pentru administratori este disponibilă și pagina de statistici, pentru care am utilizat D3.js, care este o bibliotecă JavaScript prin care am putut manipula date extrase din baza de date (în fișiere cu format .csv). Pentru culorile graficelor am ales nuanțe de mov, fiind culoare complementară pentru verde.

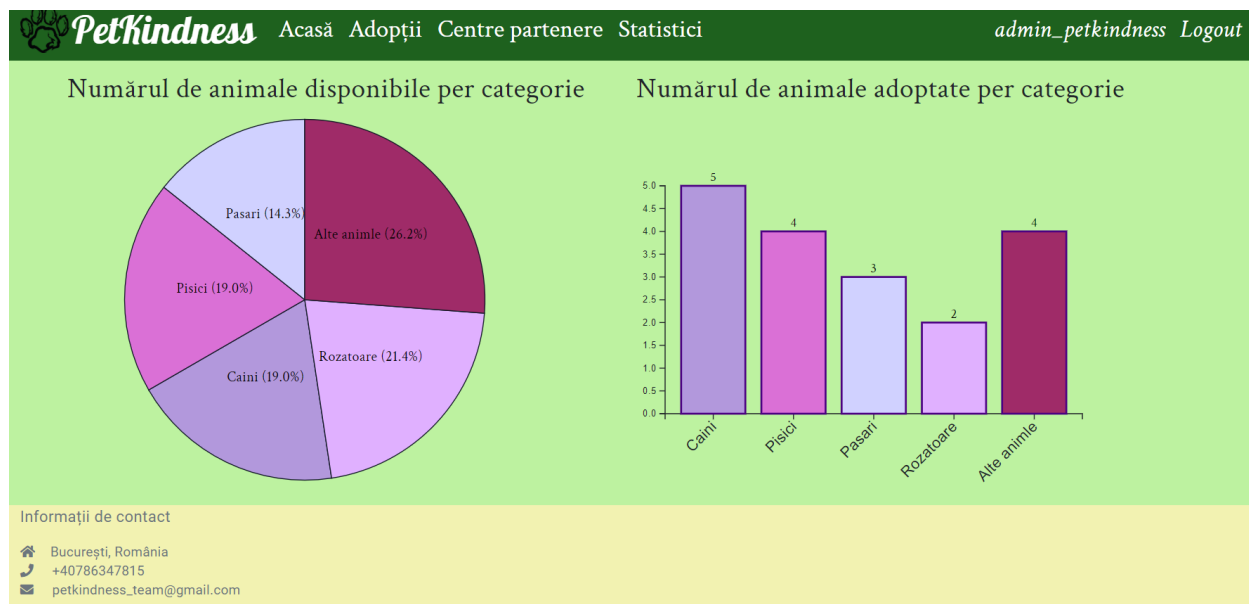


Figura 5.9.1 Pagina de statistici pentru administratori

VI. Testare și elemente de securitate

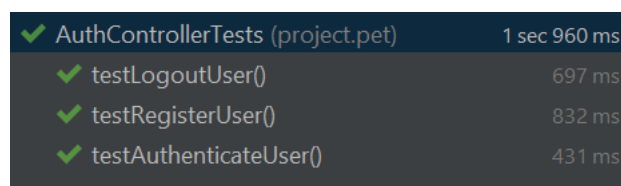
VI.1 Testare

Pentru a testa Controllere din back-end, am utilizat teste unitare, folosind framework-urile JUnit, pentru a scrie propriu-zis testele, și Mockito, pentru a simula date fictive și a nu folosi dependențe externe. Printr-un test unitar, verific dacă o anumită funcționalitate este îndeplinită în mod corespunzător.

VI.1.1 AuthController

Pentru controller-ul care se ocupă de înregistrarea și autentificarea unui utilizator al aplicației, am creat următoarele teste, la care este verificat dacă răspunsul cererilor este conform așteptărilor:

- `testAuthenticateUser()` – este simulată o cerere de autentificare (`LoginRequest`) cu credențiale existente, pe care am trimis-o la endpoint-ul corespunzător: `/petapp/auth/signin`.
- `testRegisterUser()` – este simulată o cerere de înregistrare (`SignupRequest`) cu date valide pentru un nou utilizator (numele de utilizator și email-ul trebuie să nu fie deja existente în baza de date). Această cerere am trimis-o la endpoint-ul `/petapp/auth/signup`. La fiecare rulare a acestui test, este necesar să fie modificate datele utilizatorului pentru a fi unice.
- `testLogoutUser()` – este trimisă o cerere de tip POST la endpoint-ul `/petapp/auth/signout`.



| | |
|-------------------------------------|--------------|
| ✓ AuthControllerTests (project.pet) | 1 sec 960 ms |
| ✓ testLogoutUser() | 697 ms |
| ✓ testRegisterUser() | 832 ms |
| ✓ testAuthenticateUser() | 431 ms |

Figura 6.1.1 Testele efectuate pentru AuthController

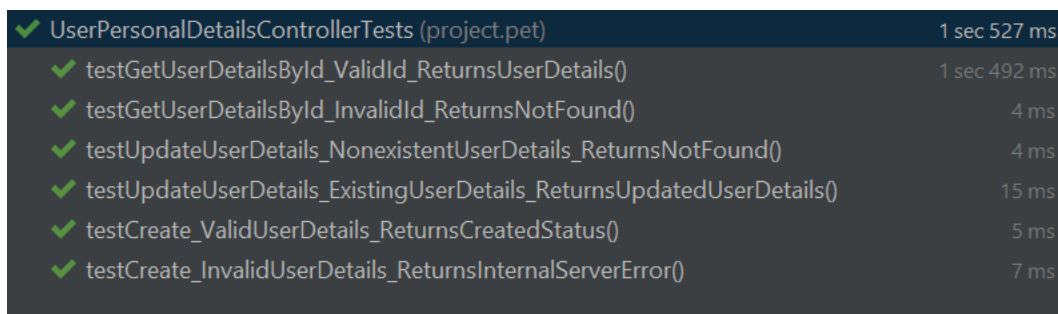
VI.1.2 UserPersonalDetailsController

Acest controller se ocupă de detaliile personale (date de contact și datele cărții de identitate) ale unui utilizator care trimite o cerere de adopție.

- `testGetUserDetailsById_ValidId_ReturnsUserDetails()` – este verificat comportamentul funcției `getUserDetailsById`, atunci când este furnizat un ID valid de utilizator. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul

HTTP OK și conține obiectul `UserDetails` așteptat.

- `testGetUserDetailsById_InvalidId_ReturnsNotFound()` – este verificat comportamentul funcției `getUserDetailsById`, atunci când este furnizat un ID invalid de utilizator. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul `HTTP NOT_FOUND`.
- `testUpdateUserDetails_NonexistentUserDetails_ReturnsNotFound()` – este verificat comportamentul funcției `updateUserDetails`, atunci când este furnizat un ID de utilizator inexistent. Testul apelează metoda și se asigură că `ResponseEntity`-ul returnat are statusul `HTTP NOT_FOUND`.
- `testUpdateUserDetails_ExistingUserDetails_ReturnsUpdatedUserDetails()` – este verificat comportamentul funcției `updateUserDetails`, atunci când este furnizat un ID valid de utilizator. Testul apelează metoda și se asigură că `ResponseEntity`-ul returnat are statusul `HTTP OK` și conține obiectul `UserDetails` actualizat.
- `testCreate_ValidUserDetails_ReturnsCreatedStatus()` – este verificat comportamentul funcției de creare din controller, care salvează datele în baza de date, atunci când sunt furnizate detalii valide ale utilizatorului. Testul apelează metoda de creare cu parametrii necesari și se asigură că `ResponseEntity`-ul returnat are statusul `HTTP CREATED` și conține obiectul `UserDetails` așteptat.
- `testCreate_InvalidUserDetails_ReturnsInternalServerError()` – este verificat comportamentul funcției de creare din controller, care salvează datele în baza de date, atunci când sunt furnizate detalii invalide ale utilizatorului. Testul simulează situația astfel încât să fie aruncată o excepție de tip `RuntimeException`. Testul apelează metoda de creare cu parametrii având valoarea „null” și se asigură că `ResponseEntity`-ul returnat are statusul `HTTP INTERNAL_SERVER_ERROR`.

A screenshot of a test runner interface showing the results of several unit tests. The tests are listed with green checkmarks, indicating they passed. The test names are in a monospaced font, and the execution times are shown in milliseconds (ms).

| | |
|---|--------------|
| ✓ UserPersonalDetailsControllerTests (project.pet) | 1 sec 527 ms |
| ✓ testGetUserDetailsById_ValidId_ReturnsUserDetails() | 1 sec 492 ms |
| ✓ testGetUserDetailsById_InvalidId_ReturnsNotFound() | 4 ms |
| ✓ testUpdateUserDetails_NonexistentUserDetails_ReturnsNotFound() | 4 ms |
| ✓ testUpdateUserDetails_ExistingUserDetails_ReturnsUpdatedUserDetails() | 15 ms |
| ✓ testCreate_ValidUserDetails_ReturnsCreatedStatus() | 5 ms |
| ✓ testCreate_InvalidUserDetails_ReturnsInternalServerError() | 7 ms |

Figura 6.1.2 Testele efectuate pentru `UserPersonalDetailsController`

VI.1.3 PetCenterController

În acest controller sunt metodele de gestionare ale centrelor partenere (adăposturi).

- `testUpdatePetCenter()` – este verificat comportamentul funcției *updatePetCenter* din controller, atunci când sunt furnizate un centru existent și un ID valid. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul HTTP OK, că obiectul `PetCenter` conținut este actualizat și că metodele *findById* și *save* au fost apelate o singură dată.
- `testGetPetCenterByIdNotFound()` – este verificat comportamentul funcției *getPetCenterById*, atunci când este furnizat un ID care nu există. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul HTTP NOT_FOUND, corpul este „null” și că metoda *findById* a fost apelată o singură dată.
- `testUpdatePetCenterNotFound()` – este verificat comportamentul funcției *updatePetCenter*, atunci când este furnizat un ID care nu există. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul HTTP NOT_FOUND, corpul este „null”, că metoda *findById* a fost apelată o singură dată și că metoda *save* nu a fost apelată.
- `testCreatePetCenter()` – este verificat comportamentul funcției de creare din controller, care salvează datele în baza de date, atunci când sunt furnizate detalii valide ale unui centru. Testul apelează metoda de creare cu parametrii necesari și se asigură că `ResponseEntity`-ul returnat are statusul HTTP CREATED, conține obiectul `PetCenter` așteptat și că metoda *save* a fost apelată o singură dată.
- `testGetPetCenterById()` – este verificat comportamentul funcției *getPetCenterById*, atunci când este furnizat un ID valid. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul HTTP OK, conține obiectul `PetCenter` așteptat și că metoda *findById* a fost apelată o singură dată.
- `testGetAllPetCentersNoContent()` – este verificat comportamentul funcției *getAllPetCenters*, atunci când nu există centre. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul HTTP NO_CONTENT, corpul este „null” și că metoda *findAll* a fost apelată o singură dată.
- `testGetAllPetCenters()` – este verificat comportamentul funcției *getAllPetCenters*, atunci

când există centre. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul HTTP OK, conține lista așteptată de obiecte `PetCenter` și că metoda `findAll` a fost apelată o singură dată.

- `testDeletePetCenter()` – este verificat comportamentul funcției `deletePetCenter`, atunci când este furnizat un ID valid. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul HTTP NO_CONTENT și că metoda `deleteById` a fost apelată o singură dată.

| | |
|--|--------------|
| ✓ PetCenterControllerTests (project.pet) | 1 sec 700 ms |
| ✓ testUpdatePetCenter() | 1 sec 666 ms |
| ✓ testGetPetCenterByIdNotFound() | 11 ms |
| ✓ testUpdatePetCenterNotFound() | 4 ms |
| ✓ testCreatePetCenter() | 4 ms |
| ✓ testGetPetCenterById() | 4 ms |
| ✓ testGetAllPetCentersNoContent() | 4 ms |
| ✓ testGetAllPetCenters() | 4 ms |
| ✓ testDeletePetCenter() | 3 ms |

Figura 6.1.3 Testele efectuate pentru `PetCenterController`

VI.1.4 PetController

Acest controller se ocupă de informațiile animalelor de companie.

- `testGetAllPetsNoContent()` – este verificat comportamentul funcției `getAllPets`, atunci când nu există animale de companie. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul HTTP NO_CONTENT, corpul este „null” și că metoda `findAll` a fost apelată o singură dată.
- `testCreatePetSuccess()` – este verificat comportamentul funcției de creare din controller, care salvează datele în baza de date, atunci când sunt furnizate detalii valide ale unui animal de companie. Testul apelează metoda de creare cu parametrii necesari și se asigură că `ResponseEntity`-ul returnat are statusul HTTP CREATED, corpul nu este „null” și că metodele `save` și `addPetImage` au fost apelate o singură dată.
- `testUploadPetPhotoNotFound()` – este verificat comportamentul funcției `uploadPetPhoto`, atunci când este furnizat un ID al unui animal de companie inexistent. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul HTTP NOT_FOUND, corpul este „null” și că metoda `findById` a fost apelată o singură dată.

- `testUploadPetPhotoSuccess()` – este verificat comportamentul funcției *uploadPetPhoto*, atunci când este furnizat un ID valid al unui animal de companie și o fotografie validă. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul HTTP OK, conține obiectul `Pet` actualizat și că metodele *findById*, *save* și *addPetImage* au fost apelate corespunzător.
- `testGetAllPetsByCategoryAndGender()` – este verificat comportamentul funcției *getAllPets*, atunci când se furnizează o categorie și un gen pentru filtrarea animalelor de companie. Testul apelează metoda din controller și se asigură că `ResponseEntity`-ul returnat are statusul HTTP OK, conține lista așteptată de obiecte `Pet` și că metoda *findByCategoryAndGender* a fost apelată o singură dată cu categoria și genul specificate.

| | |
|-------------------------------------|--------|
| ✓ PetControllerTests (project.pet) | 574 ms |
| ✓ testGetAllPetsNoContent | 514 ms |
| ✓ testCreatePetSuccess | 43 ms |
| ✓ testUploadPetPhotoNotFound | 7 ms |
| ✓ testUploadPetPhotoSuccess | 5 ms |
| ✓ testGetAllPetsByCategoryAndGender | 5 ms |

Figura 6.1.4 Testele efectuate pentru *PetController*

VI.2 Validări

În cadrul aplicației, în interfața cu utilizatorul, sunt folosite formulare, ale căror date sunt validate atunci când sunt introduse de către utilizatori.

VI.2.1 Formularul de înregistrare

Pentru procesul de înregistrare a unui utilizator există mai multe restricții. În primul rând, toate câmpurile formularului sunt obligatorii (exemplu în figura 6.2.1.2 la câmpul email). Numele de utilizator trebuie să fie unic, în caz contrar apare atenționarea din figura 6.2.1.3, să aibă cel puțin 6 caractere, și cel mult 30 de caractere (figurile 6.2.1.1 și 6.2.1.2). Adresa de email trebuie să fie unică, în caz contrar apare eroarea din figura 6.2.1.4, și să aibă un format valid (figura 6.2.1.1). Parola trebuie să aibă cel puțin 6 caractere și cel mult 40 de caractere (figurile 6.2.1.1 și 6.2.1.2).

Register

Bine ați venit!

Nume de utilizator:*

Numele de utilizator trebuie să aibă cel puțin 6 caractere.

Email*

Email-ul trebuie să fie o adresă de email validă.

Password*

Parola trebuie să aibă cel puțin 6 caractere.

Sign Up

Figura 6.2.1.1

Register

Bine ați venit!

Nume de utilizator:*

Numele de utilizator trebuie să aibă cel mult 30 de caractere.

Email*

Email-ul este necesar.

Password*

Parola trebuie să aibă cel mult 40 de caractere.

Sign Up

Figura 6.2.1.2

Register

Bine ați venit!

Nume de utilizator:*

Email*

Password*

Sign Up

Înregistrare nereușită!
Eroare! Numele de utilizator există deja.

Figura 6.2.1.3

Register

Bine ați venit!

Nume de utilizator:*

Email*

Password*

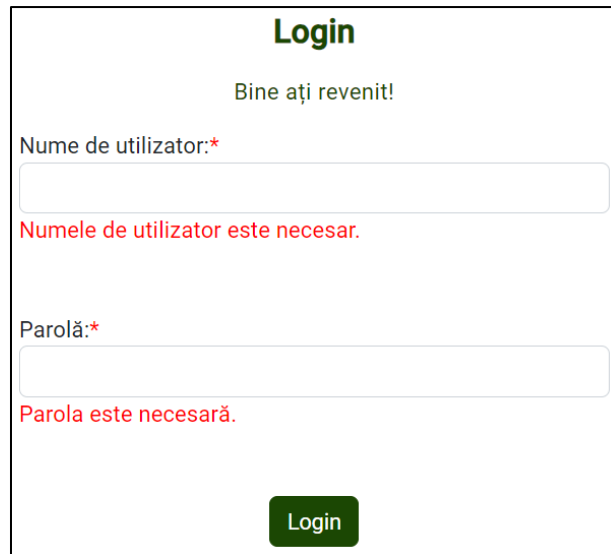
Sign Up

Înregistrare nereușită!
Eroare!

Figura 6.2.1.4

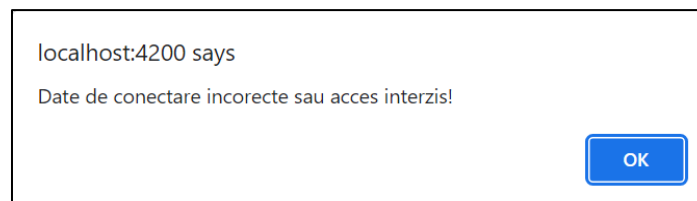
VI.2.2 Formularul de autentificare

În cazul formularului de autentificare, câmpurile sunt obligatorii și acest lucru este semnalizat prin mesaje corespunzătoare. Dacă un utilizator introduce date de conectare incorecte, atunci va apărea un pop-up cu un mesaj sugestiv (figura 6.2.2.2).



The image shows a login form titled "Login" in green. Below the title is the message "Bine ați revenit!". There are two input fields: "Nume de utilizator:*" and "Parolă:*". Below the first field is a red error message "Numele de utilizator este necesar.". Below the second field is a red error message "Parola este necesară.". At the bottom of the form is a green "Login" button.

Figura 6.2.2.1 Cerințe formular autentificare



The image shows a pop-up message box. It contains the text "localhost:4200 says" and "Date de conectare incorecte sau acces interzis!". There is a blue "OK" button at the bottom right.

Figura 6.2.2.2 Pop-up eroare autentificare

VI.2.3 Formularul de adopție

Pentru adopția unui animal de companie sunt necesare date de contact și date de pe cartea de identitate a celui care dorește să trimită cererea. În cadrul acestui formular, toate câmpurile sunt obligatorii. Pe lângă acest lucru am considerat necesare următoarele validări:

- pentru numărul de telefon este necesar un șir de 10 cifre
- CNP-ul este validat cu un algoritm, care se folosește de cifra de control
- seria actului de identitate trebuie să fie formată din două majuscule
- numărul cărții de identitate trebuie să fie format din 6 cifre

Număr de telefon *

07

Numărul de telefon trebuie să aibă 10 cifre.

CNP*

6000508100333

CNP-ul introdus este invalid.

Serie act identitate*

X

Seria actului de identitate trebuie să aibă două majuscule.

Număr act identitate*

15452

Numărul actului de identitate trebuie să aibă 6 cifre.

Figura 6.2.3 Cerințe formular adopție

VI.2.4 Formularul de adăugare a unui centru partener

Pentru utilizatorii cu rol de administrator, din pagina centrelor partenere, este disponibil formularul de adăugare a unui adăpost de animale, prin apăsarea butonului „Adaugă centru partener”. Aici toate câmpurile sunt obligatorii, iar pentru numărul de telefon este necesar un șir de 10 cifre.

Adaugă centru partener

Nume:*

Numele este obligatoriu.

Număr de telefon:*

074578564

Numărul de telefon trebuie să aibă 10 cifre.

Adresă:*

Adresa este obligatorie.

Cod hartă: *

Codul este obligatoriu.

Salvează Închide

Figura 6.2.4 Cerințe formular adăugare centru

VI.2.5 Formularul de adăugare a unui animal spre adopție

În pagina de profil a unui utilizator se găsește butonul „Adaugă animal pentru adopție”, care deschide o pagină cu un formular. În cadrul acestuia, toate câmpurile sunt obligatorii pentru a încerca să se asigure prezența a cât mai multe informații.

Adăugare animăluț pentru adopție

Vă rugăm introduceți toate detaliile cerute în acest formular!

Nume*

Numele este obligatoriu.

Categorie*

Categoria este obligatorie.

Gen*

Genul este obligatoriu.

Vârstă*

Vârsta este obligatorie.

Rasă* (daca nu se știe, scrieți Necunoscută)

Rasa este obligatorie.

Locație*

Locația este obligatorie.

Detalii despre sănătate*

Detaliile despre sănătate sunt obligatorii.

Alte detalii*

Detaliile suplimentare sunt obligatorii.

Poză*

Choose File No file chosen

Figurile 6.2.5.1 și 6.2.5.2 Cerințe formular adăugare animal pentru adopție

VI.2.6 Formularul de editare a unui animal spre adopție

În pagina de profil a unui utilizator se găsește butonul „Edit” pentru fiecare animăluț adăugat. Informațiile fiind obligatorii, câmpurile nu pot fi nule prin editare.

Editează informațiile

Nume:

Categorie: Pisici

Gen: Masculin

Vârstă: 1 an și jumătate

Rasă: Angora turcească

Locație: Iași

Detalii despre sănătate: Fără probleme de sănătate,

Alte detalii: Foarte inteligent și dragăst

Poză: Choose File No file chosen

A apărut o eroare, reîncercați

Salvează Închide

Figura 6.2.6 Eroare câmp necompletat

VI.3 Elemente de securitate

Pentru aplicația Pet Kindness, am luat în considerare cât mai multe măsuri de securitate, dar ceea ce este clar, este că pentru viitor este nevoie de îmbunătățiri la acest capitol.

- Folosesc validări ale formularelor în front-end, care sunt întărite în back-end prin folosirea unui instrument de corespondență obiect-relațională (ORM) cum este framework-ul Hibernate, care poate preveni atacuri de tip SQL Injection.
- Pentru autentificarea și înregistrarea utilizatorilor folosesc Spring Security, care oferă protecție împotriva unor atacuri, cum ar fi session fixation, clickjacking, cross site request forgery, iar parolele sunt criptate unidirecțional.
- În mesajele de eroare, evit să afișez informații sensibile sau prea specifice.
- Pentru încărcarea imaginilor, restricționez tipul acestora în front-end și dimensiunea în back-end.

VII. Concluzii

Prin această aplicație web am vrut să dau mai multe șanse animalelor din România de a fi adoptate, indiferent de specie, și să acord sprijin animalelor din adăposturi prin donații. Am dorit să fie ușor de folosit pentru utilizatori, astfel am folosit formulare clare, cu restricții specificate. Anunțurile sunt vizibile și fără crearea în prealabil a unui cont. De asemenea, am oferit posibilitatea de a adăga animale de companie spre adopție sau de a adopta, folosind un singur cont de utilizator.

După părerea mea, scopul aplicației a fost îndeplinit, fiind o aplicație intuitivă pentru utilizatori și inclusivă pentru animalele de companie. Design-ul colorat și vesel contribuie și el la atragerea utilizatorilor.

Pentru direcții viitoare de dezvoltare, m-am gândit la adăugarea unui test pentru a descoperi cu ajutorul inteligenței artificiale ce animal de companie ar fi cel mai potrivit pentru un utilizator. Din punct de vedere al securității, este necesar un certificat SSL. Un alt lucru util, ar fi și o secțiune de cabinet veterinar, cu un chat în care să poți discuta cu medici veterinari pentru a avea parte de îndrumare, sfaturi sau de mai multe păreri avizate.

VIII. Referințe bibliografice

- [1] Ziping Liu, Bidyut Gupta. *Study of Secured Full-Stack Web Development*, EPiC Series in Computing Volume 58, Pages 317–324, p. 318, 2019.
- [2] Naima Aftab. *HTTP Request Methods / Explained*, <https://linuxhint.com/http-request-methods/> [Accesat 1 iunie 2023]
- [3] *Java Tutorial*, <https://www.javatpoint.com/java-tutorial> [Accesat 1 iunie 2023]
- [4] Ken Arnold, James Gosling, David Holmes. *THE Java™ Programming Language*, Fourth Edition, p. 63, August 2005.
- [5] *Java Introduction*, https://www.w3schools.com/java/java_intro.asp [Accesat 3 iunie 2023]
- [6] <https://www.typescriptlang.org/> [Accesat 3 iunie 2023]
- [7] *Introduction to Spring Framework*, <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html> [Accesat 3 iunie 2023]
- [8] *Spring Boot*, <https://spring.io/projects/spring-boot#overview> [Accesat 4 iunie 2023]
- [9] Shyam Seshadri. *Angular: Up and Running: Learning Angular, Step by Step*, First Edition, July 2018.
- [10] *Get Familiar With Java Frameworks: Hibernate vs Spring*, <https://www.cybersuccess.biz/hibernate-vs-spring/#:~:text=Spring%20is%20useful%20for%20transaction,Security%2C%20Spring%20JDBC%20%26%20more> [Accesat 8 iunie 2023]
- [11] *Bootstrap Tutorial*, <https://www.geeksforgeeks.org/bootstrap/> [Accesat 9 iunie 2023]
- [12] *MySQL 8.0 Reference Manual*, <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html> [Accesat 12 iunie 2023]
- [13] Leon Shklar, Richard Rosen. *Web Application Architecture Principles, protocols and practices*, pp. 260-262, 2003.
- [14] *Web MVC framework*, <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html> [Accesat 14 iunie 2023]