# Web panel za postavljanje slika

**Sabo, Dominik**



**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**
**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA**

**Sveučilišni diplomski studij računarstva**

# Web panel za postavljanje slika

**Diplomski rad**

**Dominik Sabo**

**Osijek, 2021.**

# SADRŽAJ

# 1. UVOD

Ovaj diplomski rad namijenjen je ljudima koji žele što lakše naći slike, primjerice svoje najdraže vrste mačke, ili pak podijeliti slike svoje mačke kako bi ju drugi ljudi mogli vidjeti. Traženjem slika na svakojakim internetskim stranicama može dovesti do svakakvih rezultata, dok na ovoj aplikaciji, svi korisnici zajedno mogu odrediti koje su oznake na kojoj slici te tako učiniti traženje točno određenih slika lakšim. Svatko može pretraživati slike na stranici, dok je za postavljanje svojih slika i komentara, te za uređivanje oznaka na svim slikama na stranici potrebno napraviti korisnički račun. Osim oznaka na slikama, slike mogu biti postavljene u natjecanja koja kreiraju korisnici administratori. Ta natjecanja su vremenski ograničena te nakon što prođe vrijeme, slika koja ima najviše „*likeova*“ pobjeđuje i označava se kao pobjednik toga natjecanja. Slike se mogu pretraživati na razne načine; prvenstveno po oznakama na slici, primjerice ako je slika crne mačke, slika će imati oznaku „*black*“. Može se pretraživati po više oznaka, tako da će rezultat pretrage biti sve slike koje imaju na sebi obje oznake. Također, može se pretraživati po imenu korisnika koji je postavio sliku, ili po imenu natjecanja za koje je slika postavljena. Svi ti načini pretrage mogu se kombinirati, tako da ako se želi pronaći slika crne mačke koju je uploadao korisnik „x“ za natjecanje „y“, u tražilicu je potrebno upisati „*black user:x contest:y*“. Svaki korisnik također može pretražiti slike na koje je ostavio komentar ili like. Naziv ove aplikacije je „***Catbooru***“, što je skraćeni naziv za „*Cat image board*“.

Ova web aplikacija razvijena je u Java programskom jeziku u Spring Boot okruženju za logiku kojom aplikacija upravlja podacima (back end), te u HTML-u i Typescript jeziku u Angular okruženju za izgled i funkcionalnosti korisničkog sučelja (front end). Podaci se spremaju u *PostgreSQL* bazu podataka koja se nalazi u *Docker* kontejneru. Za testiranje logike aplikacije prije pisanja korisničkog sučelja koristi se aplikacija Postman.

Strukturu ovog rada tvore poglavlje u kojem se nalazi opis korištenih tehnologija, zatim poglavlje u kojem je opisan razvoj aplikacije. Nakon toga slijedi poglavlje s prikazom strukture aplikacije. Na kraju rada nalazi se zaključak.

## 1.1. Zadatak diplomskog rada

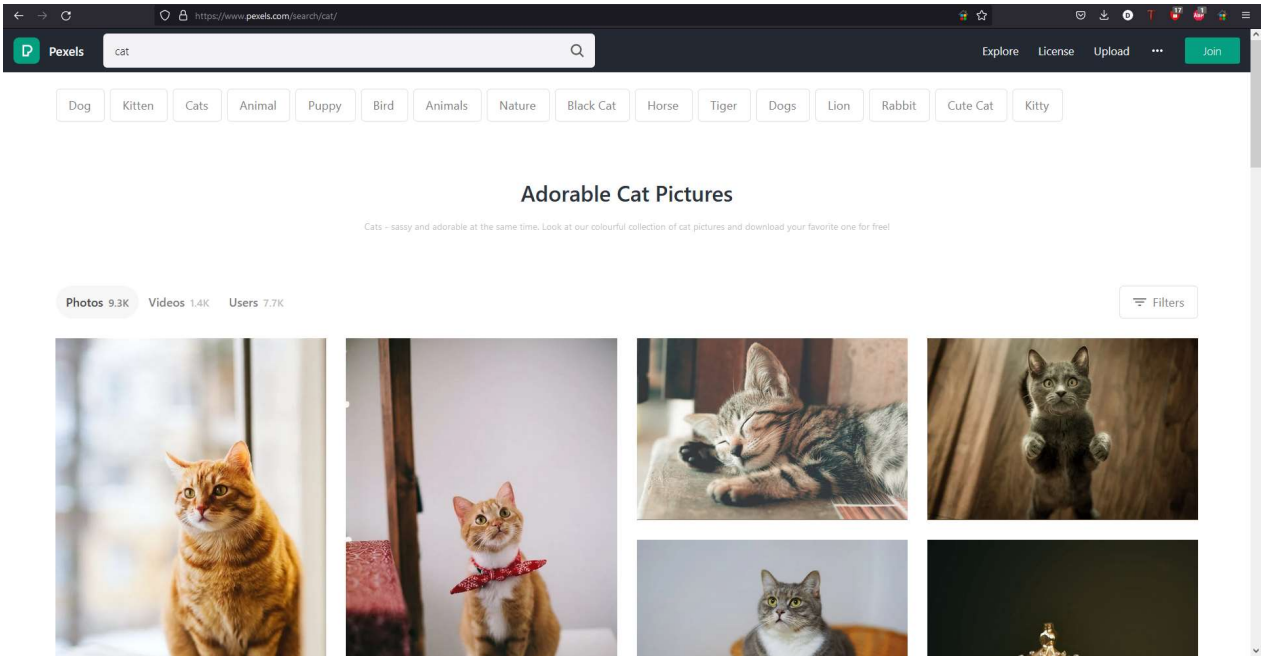
Objasniti svrhu i funkcionalnosti web panela (aplikacije) pomoću kojeg registrirani korisnici mogu objavljivati svoje slike. Izraditi web panel u kojem postoje sljedeće uloge: administrator i registrirani korisnik. Administrator treba imati mogućnost postavljanja natječaja ili izložbe slika na koje se mogu prijaviti registrirani korisnici.

# 2. PREGLED SLIČNIH RJEŠENJA

U ovom poglavlju prikazane su web stranice na kojima korisnici mogu pretraživati i postavljati slike. Svaka od stranica ima različite funkcionalnosti pretrage i korisničke interakcije, te će biti objašnjena u svojem potpoglavlju.

## 2.1. Pexels

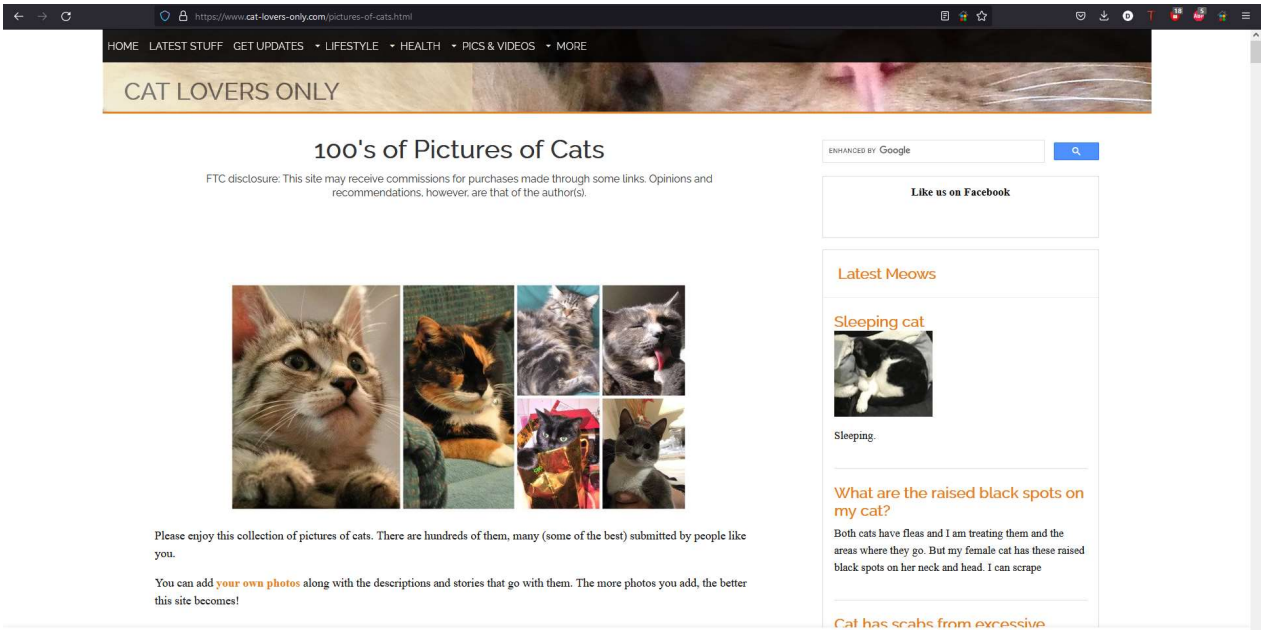
*Pexels* je web stranica na kojoj fotografi postavljaju razne slike koje drugi korisnici mogu preuzeti i besplatno koristiti na javnim mjestima, primjerice svojim vlastitim web stranicama (Slika 2.1.) [1]. O svakoj slici prikazani su podaci poput rezolucije, datuma fotografije, programske podrške kojom je fotografija uređena i slično. Stranica ima jednostavnu pretragu po nazivu fotografije koju odredi korisnik koji je fotografiju postavio.




Slika 2.1. Web stranica *Pexels.com* s pretragom „cat“.

## 2.2. Cat Lovers Only

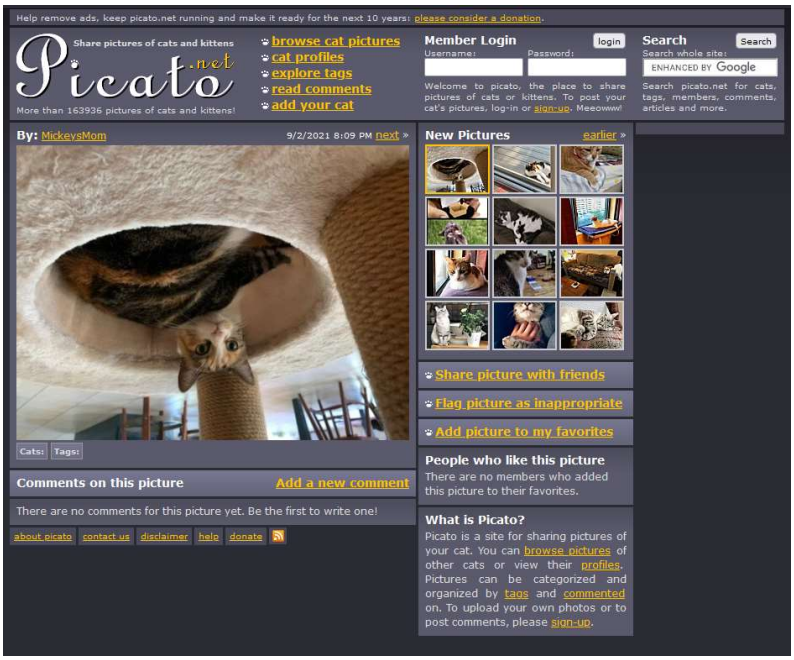
*Cat Lovers Only* je web stranica na kojoj korisnici mogu naći informacije o mačkama, uključujući i slike mačaka koje drugi korisnici postave (Slika 2.2.) [2]. Također, korisnici mogu postavljati pitanja o savjetima za odgoj svojih mačaka, te slike koje drugi korisnici mogu vidjeti. Stranica ima galeriju dostupnu svima te svaki mjesec odabiru najbolju sliku.




**Slika 2.2.** Galerija *Cat Lovers Only* web stranice.

## 2.3. Picato

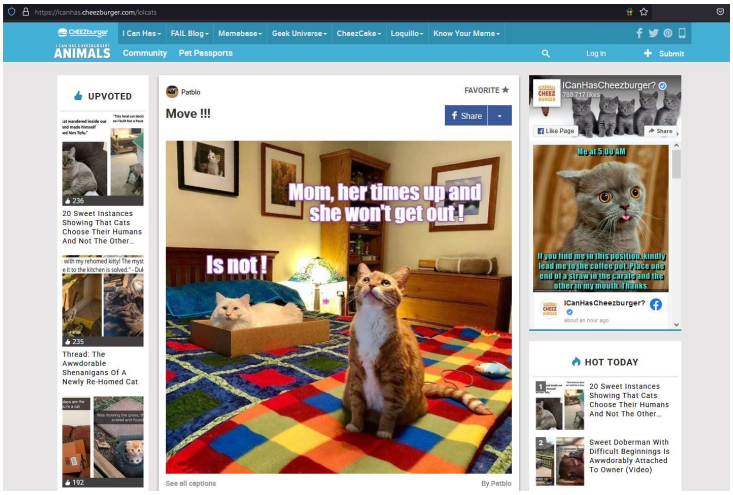
*Picato* je web stranica najsličnija zadatku ovog diplomskog rada. Na stranici se isključivo postavljaju slike mačaka, te se slike pretražuju po oznakama, korisnicima, te samim mačkama ovisno o tome ima li mačka profil koji joj je korisnik izradio (Slika 2.3.) [3]. Također, korisnici mogu komentirati slike i ostavljati like-ove.




**Slika 2.3.** Web stranica *Picato.com*

## 2.4. Cheezburger LOLcats

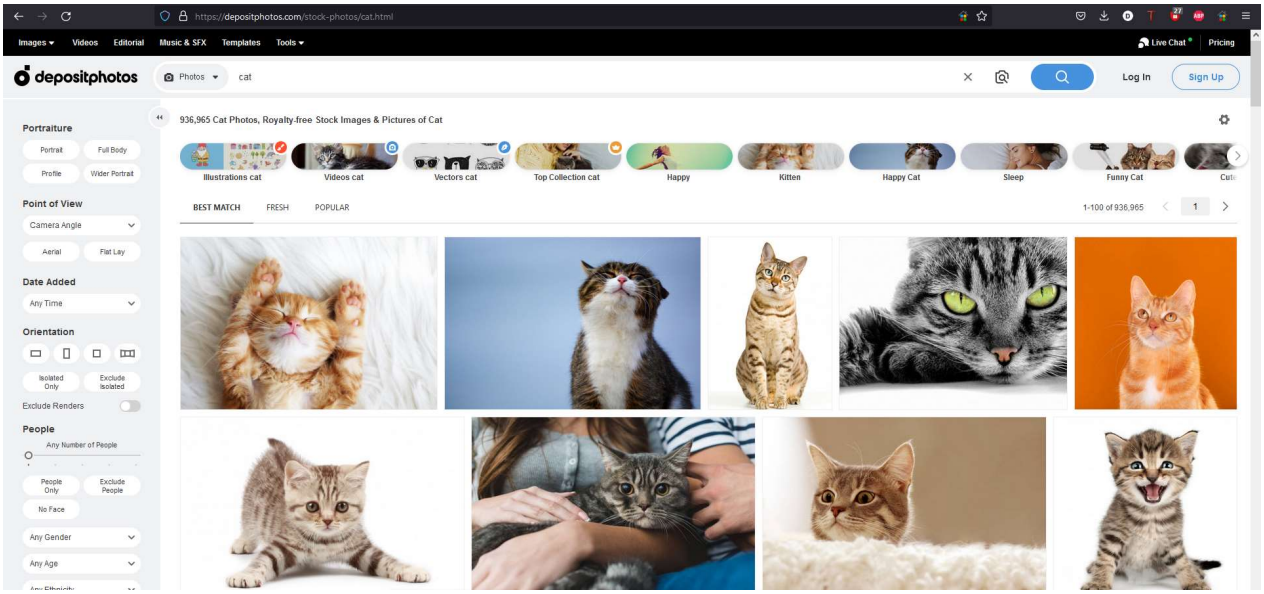
*Cheezburger LOLcats* je web stranica na kojoj korisnici postavljaju slike mačaka u komičnim situacijama (Slika 2.4.) [4]. Stranica se može pretraživati, te se na svaku od slika mogu ostavljati komentari i like-ovi.




**Slika 2.4.** Web stranica *Cheezburger LOLcats*

## 2.5. Depositphotos

*Depositphotos* je web stranica na kojoj korisnici postavljaju razne slike koje se mogu pretraživati po raznim podacima kao što su dimenzije slike, kut fotoaparata, rezolucija i slično (Slika 2.5.) [5]. Klikom na sliku, prikazuje se mogućnost preuzimanja slike, te se prikazuju slike slične po izgledu kliknutoj.




**Slika 2.5.** Web stranica *Depositphotos.com* s pretragom „*cat*“

# 3. OPIS KORIŠTENIH TEHNOLOGIJA

Tehnologije korištene za razvoj ove aplikacije opisane su u ovom poglavlju. To su programska okruženja IntelliJ IDEA i Visual Studio Code, programski okviri Spring Boot i Angular, programski jezici Java i Typescript, opisni jezici HTML i CSS, PostgreSQL baza podataka na Dockeru, te aplikacija za testiranje HTTP zahtjeva Postman.

## 3.1. IntelliJ IDEA

IntelliJ IDEA je fleksibilno programsko okruženje u kojemu se mogu pisati aplikacije u mnogim programskim jezicima. Prvenstveno je namijenjen za Java programski jezik, u kojemu je pisana ova aplikacija. IntelliJ se može konfigurirati po korisnikovoj želji te ima mnoge mogućnosti za automatsko popunjavanje koda i refaktoriranje po potrebi. Ima dinamičku detekciju grešaka i nudi korisniku automatska rješenja i optimizaciju za većinu njih. Također ima i moćni debugger koji omogućava pouzdano otkrivanje uzroka mogućih logičkih grešaka u kodu. Intellij IDEA izdala je tvrtka JetBrains u siječnju 2001 te je jedno od prvih Java programskih okruženja koje je pružalo mogućnost za refaktorirnaje i napredno pretraživanje koda. Community verziju moguće je preuzeti besplatno i dostupan je za sve najraširenije računalne operacijske sustave. Sve informacije i poveznica za preuzimanje mogu se pronaći na [6]. Back end aplikacije izrađene u ovom radu izrađen je u IntelliJ IDEA Ultimate 2021. 1. 2.

## 3.2. Visual Studio Code

Visual Studio Code je tekstualni editor koji se može koristiti za pisanje bilo kakvih dokumenata uključujući i web stranice, te uz dodatke može postati i moćno programsko okruženje. Temeljen je na Electron okviru koji se koristi za izradu Node.js web aplikacija. Pruža korisnicima mogućost da umjesto izrade projekta, kao u ostalim razvojnim okruženjima, otvore mapu, što omogućava brz pristup svim datotekama u toj mapi. Visual Studio Code podržava osnovnu podršku za programiranje te detekciju grešaka u HTML-u, CSS-u i JSON-u, te programskim jezicima

JavaScript i TypeScript, u kojem je funkcijski dio korisničkog sučelja ove aplikacije napisan. Visual Studio Code je objavila tvrtka Microsoft u studenom 2015. Korisničko sučelje ove aplikacije napisano je pomoću najnovije verzije Visual Studio Code dostupnoj na [7].

## 3.3. Spring Boot

Spring Boot okvir je dio Spring okvira koji pruža mogućnost za brzo i jednostavno pisanje Java samostojećih aplikacija koje se mogu pokrenuti i koristiti bez potrebe za ikakvom vanjskom podrškom [8]. Spring Boot pruža mogućnosti za izradu serverskih aplikacija, pakete za sigurnost, spajanje s bazama podataka i slično, te ima podršku za testiranje. Spring Boot aplikacije mogu se pisati u Kotlinu, ili u Javi, kao u ovom slučaju. Spring nudi web servis po imenu *Spring Initalizr* [9] koji omogućava generiranje Spring Boot projekta koji se može otvoriti s bilo kojim programskim okruženjem koje podržava Maven ili Gradle, ovisno o tome koja vrsta Spring Boot projekta se generira. Pri generiranju projekta, mogu se dodati paketi za funkcionalnosti potrebne u aplikaciji koja se izrađuje. Prvu verziju Spring okvira izdao je Rod Johnson u ožujku 2004. kao otvoreni kod. U ovoj aplikaciji korišten je Maven Spring Boot projekt verzije 2.5.2 s paketima za Tomcat server, Spring Security i PostgreSQL.

## 3.4. Angular

Angular web okvir je programski okvir otvorenog koda koji služi za pisanje web aplikacija temeljen na TypeScript programskom jeziku. Glavna mu je odlika generiranje komponenti web stranice pomoću svog CLI-a (engl. *Command Line Interface*). Svaka od tih komponenti sadrži funkcijski dio u TypeScript jeziku za logičku funkcionalnost, te HTML dio za izgled komponente, tako da se jednu komponentu može umetnuti u drugu i na taj način izraditi potpuno funkcionalno stablo web stranice. Svaka od komponenti ima svoju HTML oznaku te se tako umeće u druge dijelove web stranice. Na isti način mogu se generirati i servisi za slanje HTTP zahtjeva i komunikaciju s drugim web aplikacijama. Angular podržava rutiranje komponenata, tako da ovisno o URL-u, aplikacija može pokazati različite komponente u različitim stanjima, te se može izraditi intuitivni

dizajn web stranice. Angular također olakšava izrađivanje web stranica tako što kada se aplikacija podigne, svakom promjenom izvornog koda stranica se automatski osvježava. Sve o Angularu može se pronaći na [10]. Front end ove aplikacije izrađen je u Angular 12.

## 3.5. Java

Patrick Naughton, James Gosling i drugi inženjeri u tvrtki Sun Microsystems razvili su Javu, programski jezik objektno orijentiranog dizajna, u kojoj se kod nalazi unutar klasa. Godine 1991. u projektu Green započinje razvoj Jave, a objavljuje se 1995. godine. Tada je to za programere bilo nešto potpuno novo. U odnosu na tadašnje programe, programi u Javi mogli su se bez problema pokretati na svim operativni sustavima, jer svako računalo na kojemu je instaliran *Java Virtual Machine* (JVM) može izvoditi programe napisane u Javi i prevedene u Java bajt kod. C Programi su se posebno prilagođavali operacijskim sustavima na kojima su se trebali izvoditi. Java se ubraja među više programske jezike kod kojih je nedostatak da se prije izvođenja moraju prevesti, no viši programski jezici imaju više prednosti nego nedostataka. Među prednostima su mnogo jednostavnije programiranje, kraće vrijeme pisanja i kod im je lakše čitljiv. Ako na računalu Java nije instalirana, Internet stranice i aplikacije kodirane u Javi neće raditi. Java se može besplatno preuzeti i gotovo je standard da ju svako računalo ima instaliranu. [11]

## 3.6. TypeScript

TypeScript je programski jezik koji je razvio Microsoft 2012. godine [12]. Razvijen je kao postrožena inačica JavaScripta, te svaki program napisan u JavaScriptu može se koristiti kao TypeScript program. Koristi se za razvijanje klijentskih i serverskih dijelova web aplikacija, te može koristiti sve JavaScript knjižnice. Najveća razlika u odnosu na JavaScript je to što TypeScript ima tipove podataka što omogućava lakše snalaženje u kodu i pisanje dokumentacije, te smanjuje broj grešaka pri pisanju koda. TypeScriptov kompajler je i sam napisan u TypeScriptu, što znači da ga može poketati svaki web preglednik koji može prevesti Javascript. Funkcijski dio korisničkog sučelja ove aplikacije napisan je u TypeScriptu.

## 3.7. HTML

HTML (engl. *HyperText Markup Language*) je opisni jezik za definiranje izgleda web stranica, te je najosnovniji dio građe svih web stranica. HTML koristi oznake za definiranje dijelova web stranice (teksta, slika, tablica i slično) [13]. Gotovo svaki HTML element definiran je parom oznaka (otvarajuća <> i zatvarajuća </>) između kojih se nalazi sadržaj. U sadržaju elementa može biti još HTML elemenata koji se moraju zatvoriti unutar tog sadržaja. Svaki element može se dalje definirati opcijama unutar otvarajuće oznake elementa, te postoje razne opcije, primjerice za URL poveznice ili izvor slike. Opcije za izgled elemenata obično se definiraju u odvojenoj CSS datoteci koja se definira u zaglavlju HTML datoteke.

## 3.8. CSS

CSS (engl. *Cascading Style Sheets*) je opisni jezik koji služi za stiliziranje HTML ili XML dokumenata [14]. CSS omogućava mijenjanje položaja, boje, veličine i općenito izgleda svih HTML elemenata. Uz pomoć selektora klasa i identifikatora, moguće je stilizirati zasebno jedan element posebno, a moguće je i stilizirati sve elemente s istim HTML oznakama odjednom. Moguće je i stilizirati elemente koji se nalaze u specifičnim drugim elementima, te i različite pseudo klase, primjerice posebna stilizacija kada korisnik pomakne pokazivač miša preko elementa. CSS pruža veliku specifičnost i fleksibilnost pomoću kombiniranja svakakvih vrsta selektora kako bi se moglo stilizirati sve točno kako korisnik želi.

## 3.9. PostgreSQL

PostgreSQL (engl. *Postgre Structured Query Language*) ili Postgres je jedna od najrazvijenijih vrsta relacijskih baza podataka baziranih na SQL-u [15]. PostgreSQL je besplatan sistem za menadžment baza podataka te je originalno imao naziv Postgres po svom pretku Ingres, te je kasnije

dobio naziv PostgreSQL kako bi se označila podrška SQL-a, te je zadržan nadimak Postgres. Postgres podržava sve SQL relacije kao što su pogledi, okidači, procedure i slično. Dizajniran je za rukovanje visokog raspona aplikacija, te je dobar izbor baze podataka za aplikaciju koja ima mnogo aktivnih korisnika. Razvio ga je PostgreSQL Global Development Group 1996. godine. Postgres ima dobru Spring Boot podršku te je zato odabran za izradu ove aplikacije.

## 3.10. Docker

Docker je programska podrška koja omogućuje podizanje napisanih aplikacija na jednaki virtualni kontejner na bilo kojem računalnom uređaju [16]. Time se sprječava da se aplikacija ponaša drukčije kada god se pokrene na bilo kojem operativnom sustavu s bilo kojim hardverom. Docker kontejneri mogu komunicirati jedni s drugima i na taj način se na Dockeru mogu povezati mikroservisi i razne aplikacije. Kontejneri nisu zahtjevni, te se na jednom uređaju bez problema može pokretati više njih odjednom. Docker je razvila tvrtka Docker Inc. u ožujku 2013. Za aplikaciju u ovom radu, Docker se koristi za postavljanje Postgres baze podataka kako bi se jednostavno povezala s back end-om

## 3.11. Postman

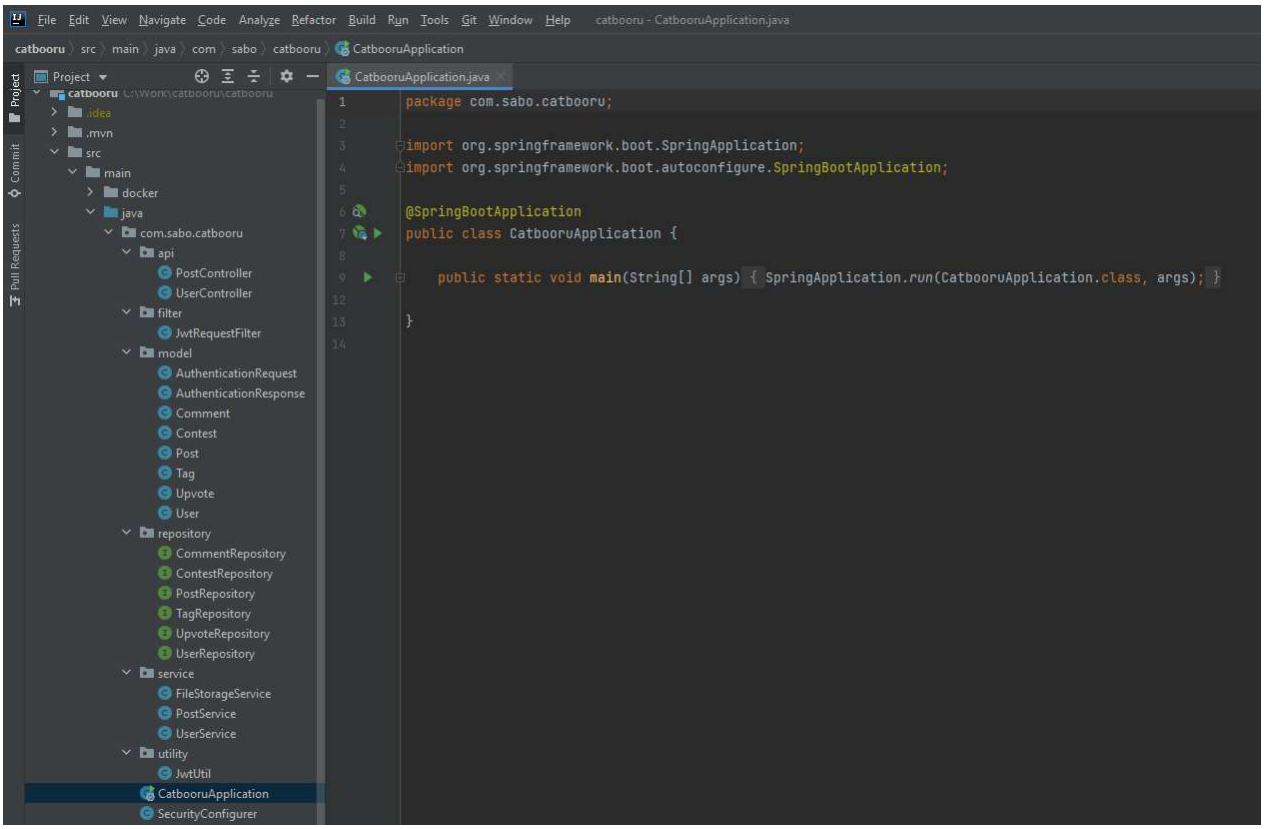
Postman je programska podrška koja služi za pojednostavljivanje izrade API-ja (engl. *Application Programming Interface*) [17]. Postmanom se mogu slati HTTP zahtjevi sa proizvoljnim parametrima, tako da je pogodan za testiranje logike web aplikacije ako još ne postoji korisničko sučelje. Za svaki napravljeni HTTP zahtjev, Postman pokaže odgovor API-ja, tako da se može prilagoditi aplikacija koja prima odgovor. U svrhe ovog rada, Postman se koristio za testiranje radi li dobro sve u Spring Boot aplikaciji koja je u principu API poveznica između korisničkog sučelja i baze podataka.

# 4. RAZVOJ APLIKACIJE

U ovom poglavlju opisan je razvoj web panela za postavljanje slika podijeljen na Spring Boot API dio, Angular korisničko sučelje, te kreacija PostgreSQL baze podataka na Docker kontejner. U Spring Boot dijelu naglasak će biti na klasnoj strukturi koda i Spring Boot specifičnim dijelovima koji aplikaciju čine web aplikacijom. Dio o korisničkom sučelju će se fokusirati na Angular komponente i servise, te komunikaciju s API-jem. Naposlijetku, Docker dio će biti kratak i pokazati kako instalirati PostgreSQL sliku i pokrenuti kontejner kako bi se aplikacije mogle spojiti na bazu.

## 4.1. Poslužiteljski dio

Na slici 4.1 prikazano je sučelje IntelliJ IDEA s otvorenim Catbooru projektom. Na lijevoj strani vidljiva je struktura projekta, svi paketi koji sadržavaju sve Java klase koje čine aplikaciju. Na desnoj strani otvorena je glavna klasa, kojom Spring Boot pokreće čitavu aplikaciju. U ovom dijelu rada, proći će se kroz sve pakete i demonstrirati na koji način se izvršava kod u Spring Boot aplikaciji. U ovom projektu koristi se *Controller -> Service -> Repository* arhitektura za protok podataka od korisničkog sučelja do baze podataka. Klasa *Controller* prima HTTP zahtjeve i podatke te ih proslijeđuje *Service* klasi, koja izvodi potrebnu logiku i pomoću *Repository* klasa komunicira s bazom podataka te sprema, izmijenjuje i dohvaća podatke po potrebi. Svako *Repository* sučelje vezano je uz svoju *Model* klasu, koja opisuje od čega se sastoji objekt s kojim repozitorij rukuje. Modeli imaju međusobne relacije te svaki od njih predstavlja tablicu u bazi podataka. Prijava korisnika u ovoj aplikaciji izvedena je pomoću JWT (engl. *JSON Web Token*) kojim rukuje *Utility* klasa te pomoću konfiguracije u *SecurityConfigurer* klasi JWT *Filter* klasa provjerava je li korisnik prijavljen ispravnim JWT-em. Konekciju na bazu podataka Spring Boot automatski obavi, no potrebno je u konfiguracijsku datoteku projekta unijeti podatke o korisničkom imenu, zaporci i adresi baze. Sve će biti detaljnije opisano u sljedećim potpoglavljima.




**Slika 4.1.** Prikaz svih klasa sadržanih u projektu zajedno s kodom Main klase.

### 4.1.1. Controller

*Controller* klase u aplikaciji zadužene su za primanje HTTP zahtjeva i odgovaranje na njih, te općenito služe za bilo kakvu komunikaciju s aplikacijom koja šalje zahtjeve. Na isječku koda 4.1. prikazan je isječak koda iz klase *PostController* u kojemu se može vidjeti konfiguracija kontrolera te metoda za dodavanje nove slike. Redovi u kodu koji započinju s „@“ znakom označavaju postavke koje koristi dio koda koji slijedi. U Spring Boot terminologiji, takve postavke nazivaju se „*Bean*“. *@RestController* označava da je klasa kontroler koji treba primati HTTP zahtjeve; u ovom slučaju to je *PostController* klasa. *@CrossOrigin* dozvoljava da zahtjevi dolaze s adrese definirane u zagradama bez posebne provjere, što je korisno jer se aplikacija korisničkog sučelja i API nalaze na istom uređaju na različitim adresama. *@RequestMapping* definira krajnju točku na koju treba slati zahtjev za taj specifični kontroler. Ta tri „*Bean-a*“ definiraju konfiguraciju *PostController* klase. Klasa u sebi sadrži

```java
@RestController
@CrossOrigin(origins = "http://localhost:4200")
@RequestMapping("/api/posts")
public class PostController {

    @Autowired
    private PostService postService;

    @PostMapping("/new")
    public ResponseEntity<byte[]> createNewPost(@RequestParam("userId") Long userId, @RequestParam(value="contestId", required = false) Long contestId,
                                                @RequestParam("imageFile") MultipartFile imageFile, @RequestParam("tags") String tags) throws Exception {
        if(contestId==null) postService.createNewPost(new Post(userId), imageFile, tags);
        else postService.createNewPost(new Post(userId, contestId), imageFile, tags);

        return ResponseEntity.ok().build();
    }
```

**Isječak koda 4.1.** Konfiguracija *PostController* klase i metoda za dodavanje slike.

instanciran objekt *PostService* klase kojemu vrijednost dodaje *@Autowired* postavka, koja automatski ubrizgava ovisnost servisa u kontroler. Metoda *createNewPost()* ima bean *@PostMapping* koji označava da prima HTTP zahtjev tipa POST na krajnju točku u zagradi. U deklaraciji metode nalaze se parametri koje metoda prima, svaki s bean-om *@RequestParam.* To znači da se ti parametri trebaju nalaziti u parametrima zahtjeva, što se sa korisničkog sučelja lako postiže slanjem iz forme. Parametar *contestId* nije potreban jer se nova slika ne mora pridružiti natjecanju, te ovisno o tome postoji li, kontroler zove metodu iz servisa ovisno o potrebi. Nakon što se ta metoda izvrši, ovisno o tome je li uspjela ili nije, kontroler vraća odgovor 200 OK ako je sve u redu, ili prikladan kod za grešku ovisno o greški koja se dogodila. Moguće greške su da je poslana datoteka krivog formata (nije slika) ili da je pak prevelika datoteka.

### 4.1.2. Service

Service klase su klase u kojima se odvija poslovna logika aplikacije s podacima koje pošalje kontroler, ili s podacima koji se dohvate iz baze podataka. Servisi su označeni s bean-om *@Service* te u sebi sadrže *@Autowired* repozitorije kojima komuniciraju s bazom podataka. U isječku koda 4.2. prikazane su metode *PostService* klase koje služe za stavljanje i uklanjanje *like*-a sa slike. Obje metode su označene s *@Transactional*, zato što se radi o izmijenjivanju podataka u bazi; ako se dogodi greška prilikom izvođenja metode, aplikacija može automatski poništiti sve promjene. Post repozitorij je zadužen za inkrementiranje i dekrementiranje broja *like*-ova u Post tablici na tom specifičnom *post-*

u. Svaki korisnik može jednu sliku *like*-ati samo jednom, stoga se stvara novi *like* u tablici, a prilikom uklanjanja, isti se briše.

```
@Transactional
public void likePost(Long userId, Long postId){
    postRepository.likePost(postId);
    upvoteRepository.save(new Upvote(userId, postId));
}

@Transactional
public void unlikePost(Long userId, Long postId){
    postRepository.unlikePost(postId);
    upvoteRepository.deleteByUserIdAndPostId(userId, postId);
}
```

**Isječak koda 4.2.** *PostService* metode za stavljanje i uklanjanje *like*-a

### 4.1.3. Repository

*Repository* sučelja nasljeđuju Spring *JpaRepository* koji služi za pravljenje veze s bazom podataka te može automatski kreirati tablice i njihove relacije koje su definirane modelima. U repozitoriju se mogu definirati metode za pretraživanje i izmjenu podataka u tablici. U isječku koda 4.3. prikazan je repozitorij za *post*-ove sa svojim definiranim metodama. *Jpa repository* ima definirane ključne riječi po kojima se može jednostavno pretraga tablice po atributima modela. Također, ako je potrebno izvesti nešto specifično, primjerice u ovom slučaju inkrementiranje jednog stupca u jednom polju tablice, to se može napraviti pomoću *@Modifying* bean-a pisanjem svojeg vlastitog SQL upita pomoću *@Query*.

```java
public interface PostRepository extends JpaRepository<Post, Long> {

    List<Post> findAllByUser_Id(Long userId);
    List<Post> findAllByContest_Id(Long contestId, Sort by);

    @Modifying
    @Query("update Post post set post.upvotes = post.upvotes + 1 where post.id = ?1")
    int likePost(Long id);

    @Modifying
    @Query("update Post post set post.upvotes = post.upvotes - 1 where post.id = ?1")
    int unlikePost(Long id);

    List<Post> findByIdIn(List<Long> ids, Sort by);
}
```

**Isječak koda 4.3.** Sučelje *PostRepository*

### 4.1.4. Model

Modeli su klase kojima se modeliraju tablice u bazi podataka. Svaki model ima repozitorij kojim se upravljaju podaci u tablici koju model predstavlja. Objekt model klase predstavlja jedan red u tablici, te se instanciranjem tih objekata vrlo jednostavno stvaraju novi redovi u tablici. Kada je objekt modela spreman, repozitorij pozove *save*() metodu s objektom kao parametrom. Ako objekt s istim identifikatorom već postoji u tablici, tada se ne stvara novi red u tablici, već se modificira postojeći. U isječku koda 4.4. prikazan je model tablice *post.* Sama klasa označena je s bean-ovima *@Entity* i *@Table* koji označavaju da se radi o entitetu u bazi podataka, specifično tablici po imenu *post.* U klasi postoje konstruktori za instanciranje objekta ovisno o tome je li slika koja se sprema u natjecanju ili ne. Svaki model treba imati svoj primarni ključ. Ovdje je to *id*, te Spring Boot-ov bean *@Id* ujedno označava da je to identifikator reda i primarni ključ u tablici. *@GeneratedValue* govori da svaki put kada se umeće novi red u tablicu, taj identifikator će biti automatski generiran, to jest, budući da se radi o podatku tipa *Long*, svaki sljedeći identifikator biti će za jedan veći od prethodnog. Nakon identifikatora, svaki parametar u klasi model predstavlja stupac u tablici. Kako korisnik postavlja sliku, svaka slika ima svog korisnika. To znači da se radi o relaciji jedan na više jer jedan

korisnik može postaviti više slika, no jedna slika ima samo jednog korisnika. Takva relacija označava se s *@ManyToOne*, te je potrebno s *@JoinColumn* definirati koji stupac u tablici korisnik je zajednički s tablicom post. Sve isto vrijedi i za natjecanja, jer natjecanje ima više slika, a jedna slika može biti samo u jednom natjecanju. Dalje u klasi nalaze se atributi, to jest stupci u tablici koje slika ima označeni s *@Column*. To su adresa slike, broj like-ova te datum i vrijeme kada je slika postavljena.

```java
@Entity
@Table(name="post")
public class Post {

    public Post(){}

    public Post(Long userId) { this.user = new User(userId); }

    public Post(Long userId, Long contestId){...}

    @Id
    @GeneratedValue
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne
    @JoinColumn(name = "contest_id")
    private Contest contest;

    @Column(name="filepath")
    private String filePath;

    @Column(name="upvotes")
    private int upvotes = 0;

    @Column(name="timestamp")
    private Timestamp timestamp = Timestamp.from(Instant.now());
```

**Isječak koda 4.4.** Model klasa *Post*

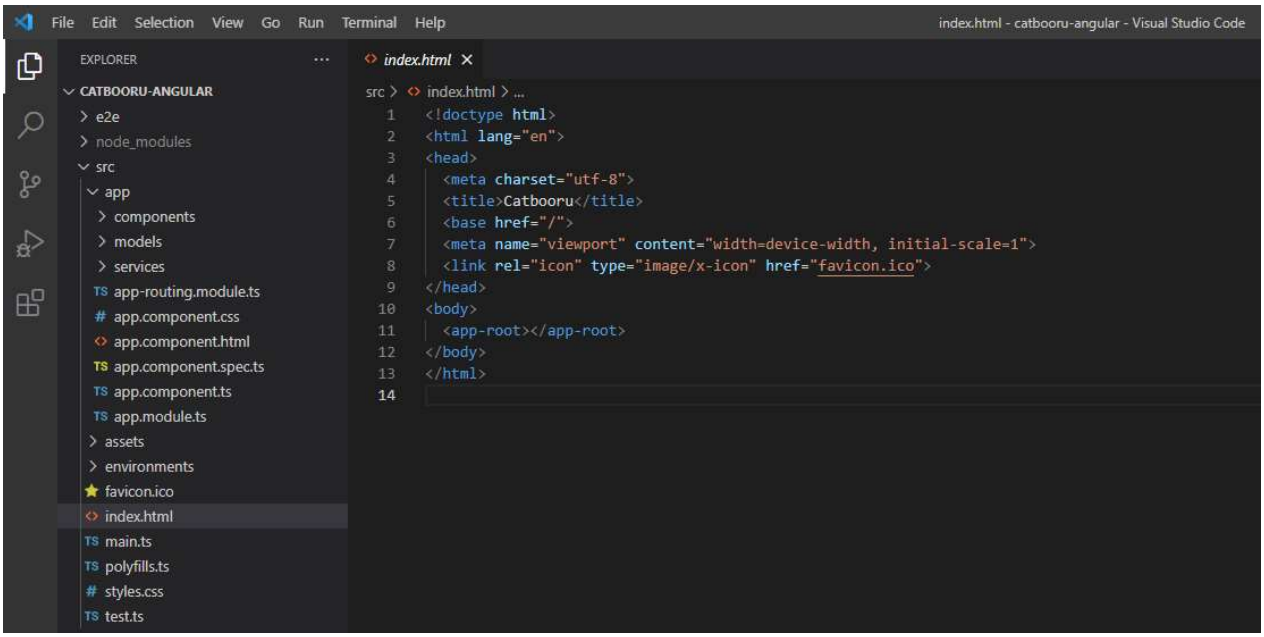
### 4.1.5. Spring Security i JWT

Kada se napravi Spring Boot aplikacija s aktiviranim Spring Security paketom, ima zadane postavke koje u ovom slučaju nisu potrebne. Zbog toga se u *SecurityConfigurer* klasi definiraju vlastite postavke o sigurnosti te se zadaje da svaki HTTP zahtjev mora proći kroz sigurnosni filter iz

*JwtRequestFilter* klase koji provjerava je li u zaglavlju zahtjeva ispravan JWT. Na osnovu toga aplikacija određuje treba li nastaviti s metodama koje određuje HTTP zahtjev, ili automatski šalje 403 Forbidden odgovor. Naravno, budući da nije potrebno da je korisnik prijavljen za svaki mogući zahtjev u aplikaciji, primjerice dohvaćanje slike za gledanje, u klasi se također definiraju zahtjeve za koje nije potrebna provjera. Također, u *SecurityConfigurer*-u je definiran enkoder za korisničke zaporke, jer ne želimo spremiti ne enkodiranu zaporku korisnika kada se prijavi u bazu podataka. Svaki put kada se registrira korisnik, generira se novi JWT pomoću *JwtUtil* klase. U toj klasi se nalazi ključ za generiranje JWT-eva, te trajanje i podaci, te metoda za provjeru je li JWT ispravan. Također, na svaku prijavu generira se novi JWT i šalje se aplikaciji korisničkog sučelja kao odgovor kako bi sučelje u budućim zahtjevima moglo u zaglavlje postaviti ispravan JWT.

## 4.2. Klijentski dio

Korisničko sučelje (front end dio) aplikacije napisano je u Angular radnom okviru u Visual Studio Code okolini. Angular se koristi s Typescript programskim jezikom te s HTML i CSS opisnim jezicima. U ovom dijelu, bit će pokazana struktura Angular aplikacije s fokusom na komponente i servise, te Angular specifične direktive koje pomnažu pri kreiranju intuitivnog web sučelja. Na slici 4.2. prikazana je struktura Angular projekta, te dio HTML koda koji sadržava zaglavlje s postavkama te tijelo koje se sastoji od komponente *app-root*, osnovne komponente u kojoj će se nalaziti sve ostale naknadno dodane komponente. U strukturi projekta može se vidjeti da se aplikacija sastoji od komponenti, modela i servisa, te routing modula i postavki. U sljedećim potpoglavljima će sve biti detaljnije opisano.




**Slika 4.2.** Angular projekt

### 4.2.1. Komponente

Komponente u Angularu sastoje se od četiri datoteke:

- HTML datoteka
- Typescript datoteka
- Typescript datoteka za testiranje
- CSS datoteka

U HTML datoteci definira se od kojih se HTML elemenata komponenta sastoji. Svaka od komponenti može se i sama upotrijebiti kao HTML element te se tako izrađuje ugniježđena struktura komponenti u Angularu. Typescript datoteka služi za dodavanje funkcionalnosti komponenti, primjerice hoće li se nešto dogoditi ako korisnik klikne na određeni element u toj komponenti. U drugoj Typescript datoteci se pišu testovi za funkcionalnosti. Naposlijetku, u CSS datoteci se pravi izgled komponente, kao što bi se to radilo za svaku uobičajenu HTML web stranicu. Na isječku koda 4.5. prikazana je HTML struktura *home* komponente aplikacije.

```
    <div class="title">
        <img (click)='home()' class='logo' src="assets/inner/logo.png" alt="">
    </div>

<div class='home'>
    <ul>
        <li (click)='onLoginClick()'>{{login}}</li>
        <li (click)='onPostsClick()'>Posts</li>
        <li (click)='onTagsClick()'>Tags</li>
        <li (click)='onContestsClick()'>Contests</li>
        <li (click)='onNewClick()'>New</li>
    </ul>


    <div class="search">
        <input (keyup)='onKey($event)' type="text">
        <select (change)='onSelect($event)' name="order" id="order">
            <option value="timestamp descending">Newest</option>
            <option value="timestamp ascending">Oldest</option>
            <option value="upvotes descending">Most upvotes</option>
        </select>
        <button (click)='onSearchClick()'>Search</button>
    </div>
</div>

<router-outlet (activate)="onActivate($event)"></router-outlet>



<h2>{{message}}</h2>
```

**Isječak koda 4.5.** HTML *home* komponente

Home komponenta sastoji se od naslovne slike koja predstavlja logo aplikacije, navigacije postignute listom te od elementa za unos teksta, padajućeg izbornika i tipke koji služe za pretraživanje slika. Na dnu se nalazi element koji prikazuje poruku ako pretragom ništa nije pronađeno. Element *router-outlet* služi za navigiranje na određene komponente ovisno o vrijednosti koja se pošalje pri aktivaciji nekom metodom ili akcijom. Sve što se nalazi unutar dvostrukih vitičastih zagrada odnosi se na Typescript varijable u istoj komponenti, primjerice, *{{message}}* je varijabla koja je inače prazna, te *h2* element ne pokazuje ništa, no ako je potrebno nešto prikazati, čim se promijeni varijabla *message*, prikaže se to u što se promijenila. Slično, postavke u HTML elementima koje se nalaze u zagradama označavaju određene akcije nad tim elementima; *(click)* znači da je kliknuto na element, *(keyup)* znači da se pritisnula tipka na tipkovnici i slično. Kada se takva akcija dogodi, okida se Typescript metoda na koju oznaka u zagradama pokazuje. Na primjer, kada se klikne na *Posts* u listi, izvršiti će se metoda *onPostsClick()*.

### 4.2.2. Servisi

Servisi su Typescript klase namijenjene za komunikaciju s vanjskim API-ima i za prihvaćanje podataka iz vanjskih izvora. U ovoj aplikaciji se koriste za komunikaciju sa Spring Boot back end dijelom. Na isječku koda 4.6. prikazan je dio servisa za slike kojim se dohvaćaju slike kada se unese upit u tražilicu.

```
@Injectable({
  providedIn: 'root'
})
export class PostService {
  query:string = '';
  order:string = 'descending';
  sort:string = 'timestamp';
  url:string = 'http://localhost:8080/api/posts/'

  constructor(private userService:UserService, private http:HttpClient) { }

  getAllPosts():Observable<Post[]>{
    return this.http.get<Post[]>(this.url + 'all', {params: new HttpParams().set('order', this.order).set('sort', this.sort) })
  }

  getPosts():Observable<Post[]>{
    if(this.query==null) this.query='';
    return this.http.get<Post[]>(this.url + 'query', {params: new HttpParams().set('query', this.query).set('order', this.order).set('sort', this.sort) })
  }
```

**Isječak koda 4.6.** Dio *PostService* klase

U parametrima klase nalaze se zadane vrijednosti za pretraživanje te URL na koji se šalju zahtijevi. Taj URL odgovara URL-u zadanom u *@RestController* postavci u back end dijelu. U konstruktoru klase nalaze se druge klase čije metode ova klasa koristi. *UserService* je za dohvaćanje podataka o prijavljenom korisniku ako je potrebno, a *HttpClient* objekt koristi se za slanje zahtjeva. Sve metode u servisu koje dohvaćaju nešto vraćaju asinkroni *Observable* objekt koji se pozove kada je dohvaćanje podataka izvršeno, kako ne bi došlo do pristupa nepostojećim podacima. Metoda za slanje zahtjeva prihvaća URL na koji se šalje zahtjev, te same parametre i zaglavlje zahtjeva. U ovim metodama zaglavlje nije potrebno eksplicitno definirati jer služe za dohvaćanje slika, što nije zaštićeno na back end-u. Ako je potrebno pristupiti zaštićenoj krajnjoj točki, potrebno je zahtjevu zadati zaglavlje s *Authorization* opcijom i točnim JWT-em (isječak koda 4.7.). U isječku koda koristi se token iz *user* klase, koja se popuni podacima kada god se korisnik prijavi.

```
addNewPost(user:User, formData:FormData){
  const headerDict = {
    'Authorization': 'Bearer '+user.token
  }
  const requestOptions = {
    headers: new HttpHeaders(headerDict),
  };
  return this.http.post(this.url + 'new', formData, requestOptions)
}
```

**Isječak koda 4.7.** Metoda za dodavanje slike

### 4.2.3. Modeli, NG direktive i routing

Modeli se u Angularu koriste za sličnu svrhu kao u Spring Boot-u. Kada neki zahtjev vraća objekt ili listu objekata, u Angularu se definira model koji sadrži parametre tog objekta, te se instancira objekt u koji se spremaju podaci. Na isječku koda 4.8. nalazi se model korisnika.

```
export class User{
    username:string;
    id:string;
    token:string;
    admin:boolean;
}
```

**Isječak koda 4.8.** Model korisnika

Taj model može se instancirati kao objekt u bilo kojoj Typescript klasi i na taj način aplikaciji je moguće pristupiti imenu korisnika, njegovom identifikatoru i JWT-u kojim je prijavljen na aplikaciju kako bi mogao pristupiti svim funkcijama aplikacije. U ovoj aplikaciji postoje još modeli za *post*-ove, komentare i natjecanja.

Angular direktive koriste se za intuitivno konfiguriranje kako se dijelovi web stranice prikazuju i na koji način. U ovoj aplikaciji koriste se dvije takve direktive: *ngIf* i *ngFor*. Obje su jednostavne, ali veoma korisne. Direktiva *ngIf* koristi se kao if granjanje u HTML kodu, to jest, ako

je zadani uvjet u *ngIf* istinit, HTML element povezan s njime se prikazuje, ako nije istinit, neće se prikazati (Isječak koda 4.9.)

```
<h2 (click)="onContestClick()">{{contest.name}}</h2>
<p>{{contest.description}}</p>
<p *ngIf='!contest.finished; else elseBlock'>Contest is underway and ends on {{date}} at {{hour}}.<br>
    <span (click)="onNewClick()">Enter new post into contest</span></p>
<ng-template #elseBlock>
    <div>
        <p>Contest finished on {{date}}. Winning post:</p>
        <img *ngIf='filepath!=null' class='image' src={{filepath}} alt="{{filepath}}" (click)="onWinnerClick()">
    </div>
</ng-template>
```

**Isječak koda 4.9.** Korištenje direktive *ngIf* u komponenti natjecanja

U ovom primjeru, potrebno je prikazati je li natjecanje završilo, te sliku koja je pobijedila ako je gotovo, vrlo jednostavno, natjecanje ima boolean varijablu je li gotovo ili nije, te *ngIf* provjeri tu varijablu i odredi što treba prikazati. Direktiva *ngFor* je slična *ngIf*, ali umjesto grananja radi se o petlji. *ngFor* prolazi kroz sve elemente u polju, te za svaku od njih u HTML-u napravi što je potrebno

```
<p class="post"><app-posts-item (click)="onClick(post)" *ngFor="let post of posts" [post]="post"></app-posts-item></p>


<h2>{{notfound}}</h2>
```

**Isječak koda 4.10.** Korištenje direktive *NgFor* u komponenti liste slika

U isječku koda 4.10. radi se o komponenti *posts-list*, koja se pojavi kada se pretražuje slike. Servis dohvati listu objekata koji se spreme u listu objekata modela. Kroz tu listu *ngFor* iterira, te za svaki objekt u listi napravi novu *posts-item* komponentu kojoj pošalje podatke o toj slici. Na taj način se po redu prikažu sve slike koje se dohvate te klikom na bilo koju od njih se slika pokaže sa svojim detaljima. Na isti način se dohvaćaju komentari za svaku pojedinačnu sliku te natjecanja.

Routing u Angular aplikaciji odnosi se na preusmjeravanja određenih URL adresa u pregledniku na njima zadane komponente. To se sve podešava u *app-routing.module* datoteci u Angular projektu (Isječak koda 4.11.).

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { PostsListComponent } from './components/posts-list/posts-list.component';
import { HomeComponent } from './components/home/home.component';
import { PostComponent } from './components/post/post.component';
import { LoginComponent } from './components/login/login.component';
import { RegisterComponent } from './components/register/register.component';
import { AccountComponent } from './components/account/account.component';
import { AllTagsComponent } from './components/all-tags/all-tags.component';
import { NewPostComponent } from './components/new-post/new-post.component';
import { ContestsComponent } from './components/contests/contests.component';

const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'posts', component: PostsListComponent},
  { path: 'posts/liked', component: PostsListComponent},
  { path: 'posts/post/:id', component: PostComponent},
  { path: 'posts/:query', component: PostsListComponent},
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'account', component: AccountComponent },
  { path: 'tags', component: AllTagsComponent },
  { path: 'contests', component: ContestsComponent },
  { path: 'new', component: NewPostComponent},
  { path: 'new/:contest', component: NewPostComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

**Isječak koda 4.11.** *Routing* modul

U routing modulu nalaze se sve komponente koje su u projektu za koje je potrebno izraditi rute. Rute se definiraju svojim URL nastavkom i komponentom u koju vode. Primjerice, ako se želi pristupiti komponenti za prijavu, nije potrebno kliknuti na element u aplikaciji koji vodi na prijavu, već joj se može pristupiti i tako da se napiše općenita adresa web aplikacije i na to doda „*./login*“. Za pravljenje routa mogu se koristiti i varijable. Na primjer, svaka slika ima svoj identifikator te ako se na rutu „*posts/post/*“ doda identifikator te slike, web stranica će odmah pokazati detaljno tu sliku.

## 4.3. Baza podataka i Docker

U ovoj aplikaciji koristi se *PostgreSQL* baza podataka dignuta u *Docker* kontejner radi lakšeg upravljanja. Samu bazu nije potrebno oblikovati zato što Spring Boot pri pokretanju aplikacije automatski kreira sve potrebne tablice i relacije te je baza odmah spremna za rad, no naravno, važno je definirati gdje se nalazi baza podataka te informacije za pristup. Te informacije se u Spring Boot projektu opisuju u *application.properties* datoteci (Isječak koda 4.12.).

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults = false
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQL9Dialect
spring.datasource.url=jdbc:postgresql://localhost:5432/catboorudb
spring.datasource.username=postgres
spring.datasource.password=password
```

**Isječak koda 4.12.** Informacije za pristup bazi podataka u *application.properties*

Baza podataka se kreira u *Docker* kontejneru nakon što se instalira virtualna *Docker* slika *postgres*. Za ovu aplikaciju koristi se slika *postgres:latest*. Baza podataka kreirana je sljedećom naredbom u komandnoj liniji:

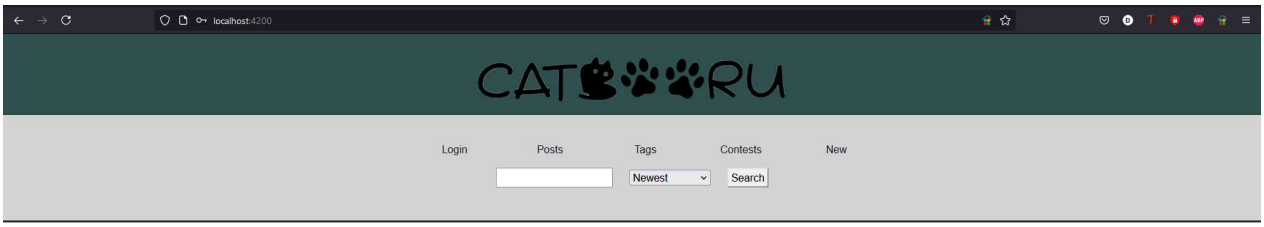
```
Docker run --name ime -p 5432:5432 -e POSTGRES_PASSWORD=zaporka postgres
```

U toj naredbi, „*Docker*“ naravno označava da se naredba pokreće programom *Docker*, „run“ služi za pokretanje kontejnera, no u ovom slučaju taj kontejner ne postoji, pa će *Docker* interpretirati da ga treba stvoriti. Postavka *–name* znači da će ime tog kontejnera biti što god se napiše nakon toga, u ovom slučaju to je „ime“. Parametar *–p* označava da se port na kontejneru koji se koristi za pokretanje te postgres baze proslijeđuje na stvarni port na fizičkom uređaju na kojemu se pokreće kontejner, tako da će toj *PostgreSQL* bazi podataka biti moguće pristupiti na portu 5432 na računalu. Parametar *–e* označava varijable okruženja, te se ovdje definira samo „POSTGRES_PASSWORD“, koji naravno označava zaporku za Postgres korisnika koji je uobičajeno samo „postgres“ što je ovdje i ostavljeno. Na kraju „postgres“ je ime slike na kojoj će se kontejner pokrenuti. Ako slika ne postoji na računalu, *Docker* će automatski preuzeti i instalirati najnoviju verziju. Nakon što se naredba izvrši i kontejner

je spreman za upotrebu, potrebno je samo kreirati bazu podataka s odgovarajućim imenom i pokrenuti Spring Boot aplikaciju.

# 5. STRUKTURA APLIKACIJE

U ovom poglavlju biti će opisan izgled aplikacije te će se detaljno proći kroz sve funkcionalnosti aplikacije. Na svakoj slici koja pokazuje web aplikaciju biti će prikazana i adresna traka preglednika kako bi se mogao vidjeti URL nastavak stranice. Opis će biti podijeljen na nekoliko glavnih funkcionalnosti. Na slici 5.1. prikazan je izgled početne web stranice aplikacije.

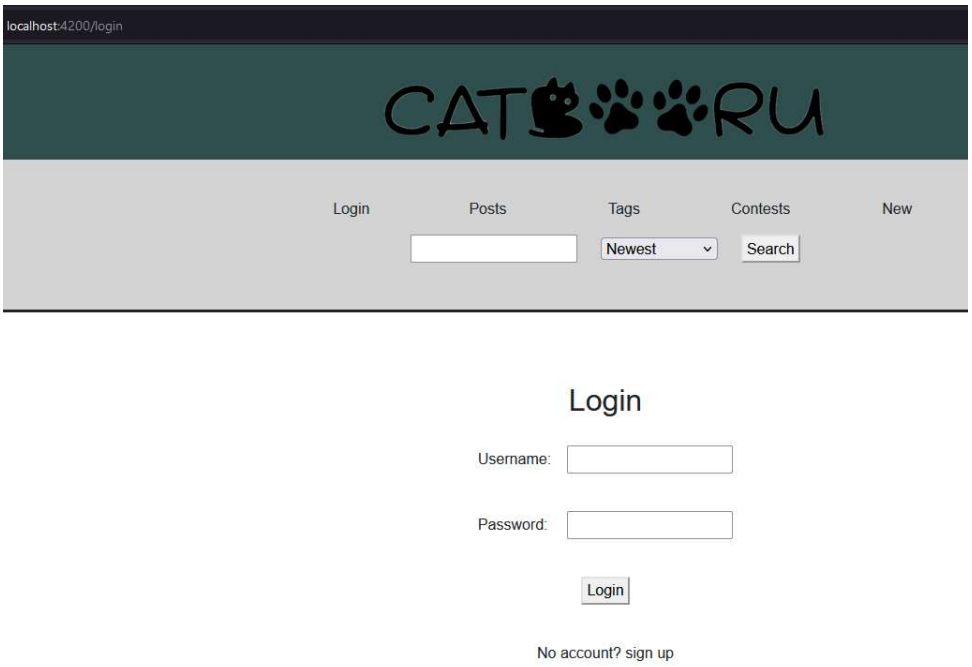



**Slika 5.1.** Početna stranica

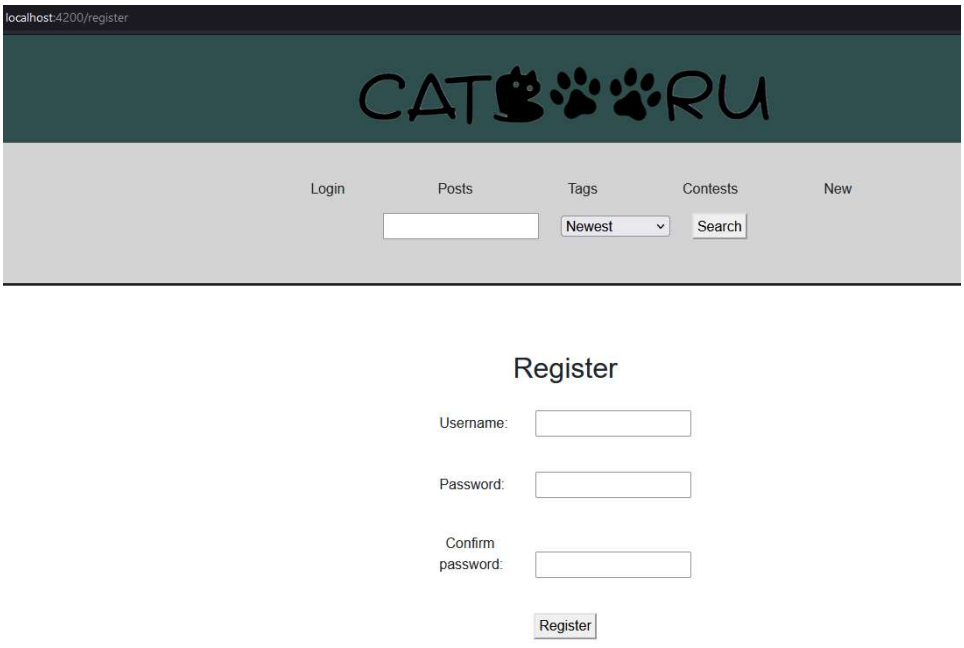
Početna se stranica sastoji od navigacije i trake za pretraživanje te je uvijek vidljiva, bez obzira na koji dio stranice korisnik navigira.

## 5.1. Prijava korisnika

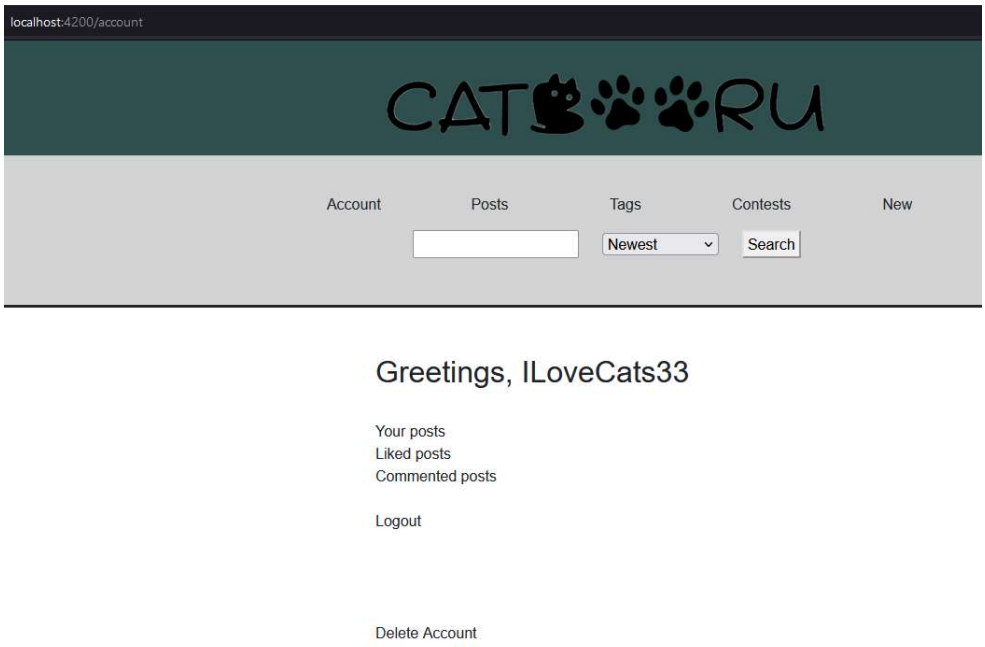
Kako bi se korisnik prijavio na aplikaciju, treba na navigacijskoj traci kliknuti na „*Login*“, što će ga odvesti na formu za prijavu prikazanu na slici 5.2. Ova forma za prijavu koristi se ako korisnik već ima korisnički račun te korisnik u nju unosi svoje korisničko ime i zaporku. Ukoliko korisnik nema račun, treba kliknuti na „*Sign up*“ ispod forme. To odvodi korisnika na formu za izradu računa prikazanu na slici 5.3. Kada se korisnik prijavi na svoj račun ili izradi novi, aplikacija ga odvodi na stranicu računa (slika 5.4.). Na toj stranici korisnik može klikom na odgovarajuću poveznicu pogledati slike koje je postavio, slike na koje je postavio komentar ili slike na koje je stavio *like.* Također, može se odjaviti i obrisati korisnički račun. Ako je korisnik administrator, na toj stranici također ima mogućnost učiniti drugog korisnika administratorom, no za to je potrebna tajna zaporka (slika 5.5.)



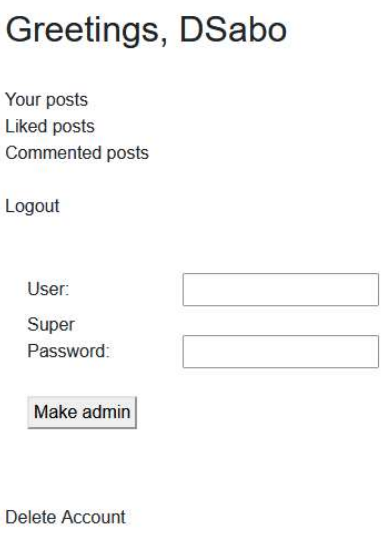

**Slika 5.2.** Forma za prijavu




**Slika 5.3.** Forma za izradu korisničkog računa



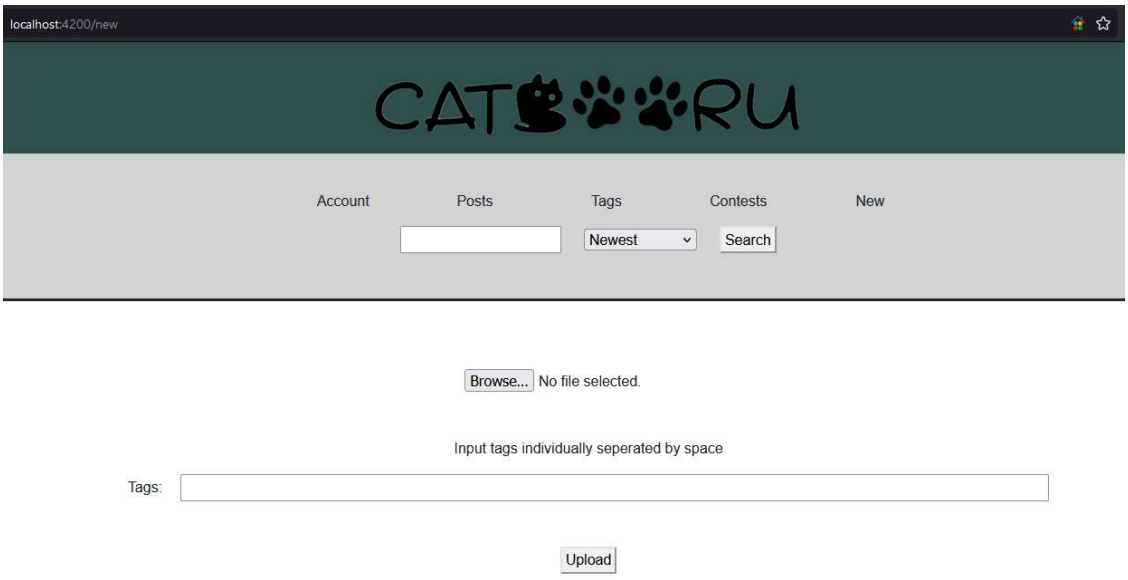

**Slika 5.4.** Stranica korisničkog računa




**Slika 5.5.** Dio stranice korisničkog računa administratora

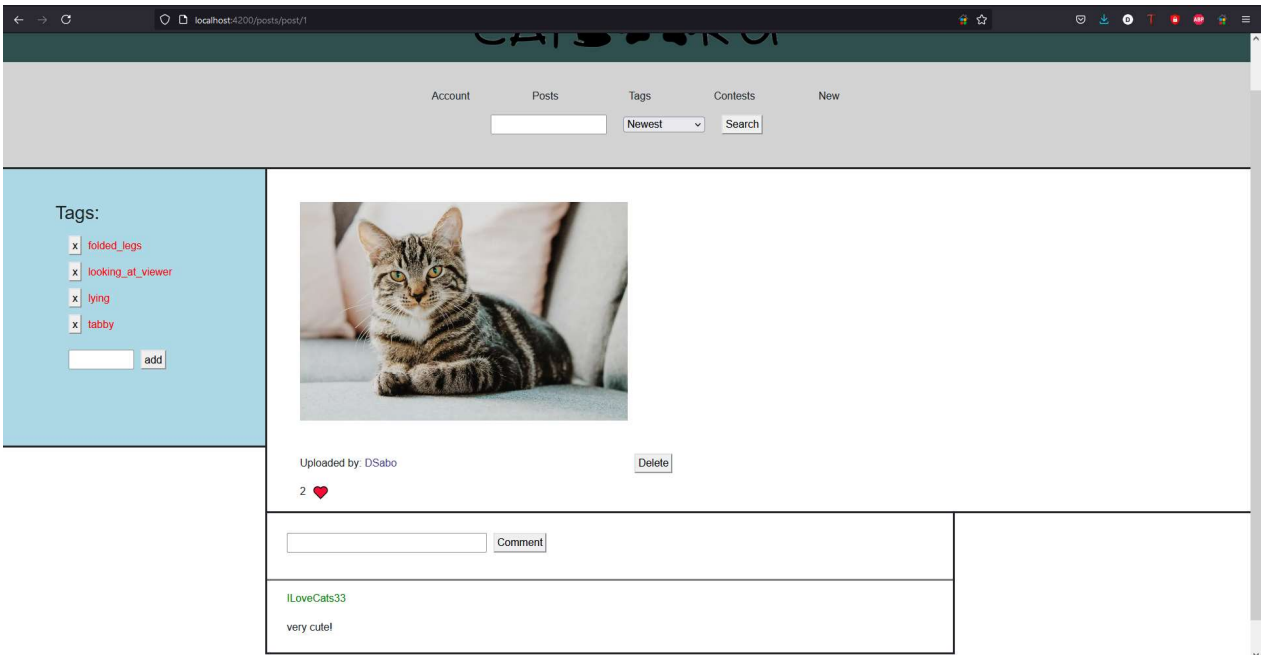
## 5.2. Postavljanje i uređivanje slika i natjecanja

Za postavljanje nove slike na aplikaciju, potrebno je na navigaciji kliknuti na „*New*“ što korisnika odvodi na formu za postavljanje slike i definiranje njenih oznaka kojima će se moći pretražiti. Ako korisnik nije prijavljen, klik na „*New*“ će ga odvesti na formu za prijavu. Stranica forme za postavljanje slike prikazana je na slici 5.6.



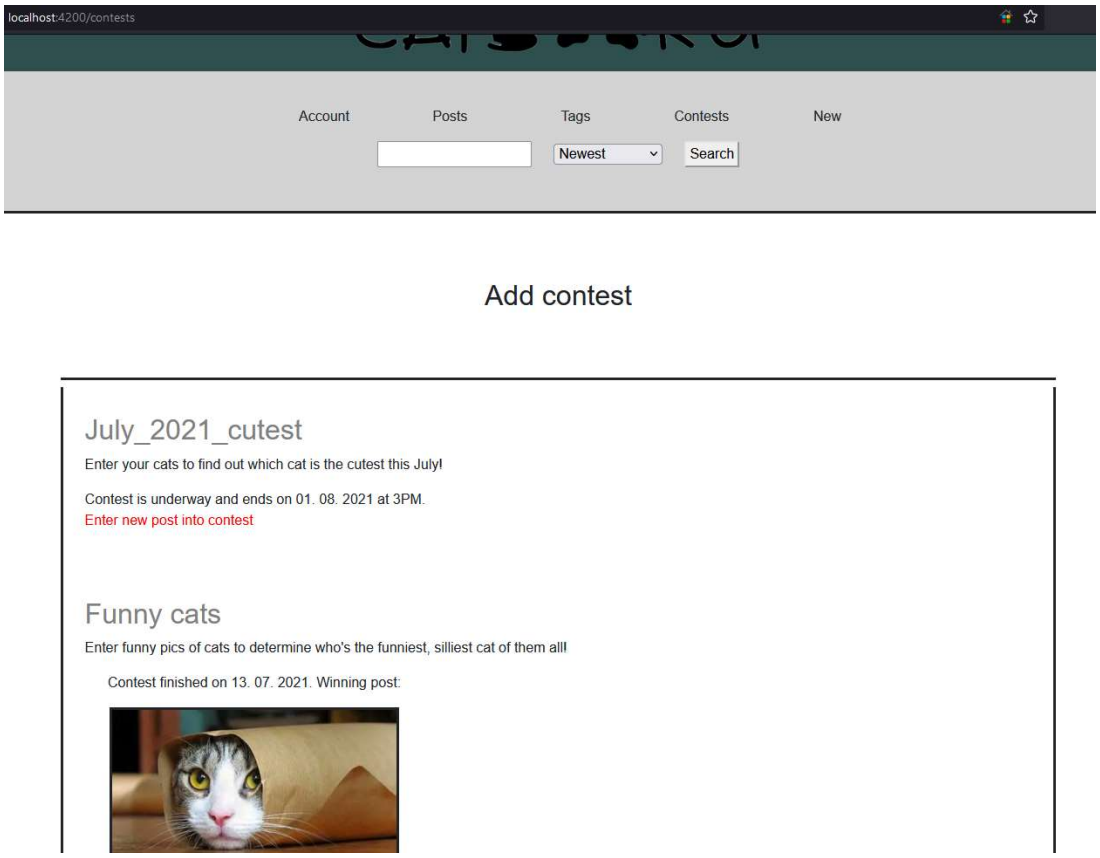

**Slika 5.6.** Forma za postavljanje slike

Klikom na tipku „*Browse...*“ otvara se pretraživanje datoteka na računalu te se može izabrati datoteka slike koju korisnik želi postaviti na aplikaciju. U polje za tekst pišu se oznake te slike kojima će se slika moći naći ako se upišu u traku za pretraživanje. To polje se može ostaviti prazno, no onda će se slika moći naći samo preko imena korisnika ili pretragom svih slika. Naposlijetku, klikom na tipku „*Upload*“ slika se postavlja na stranicu, te ju mogu vidjeti svi korisnici. Stranica slike vidi se na Slici 5.7.




**Slika 5.7.** Stranica slike

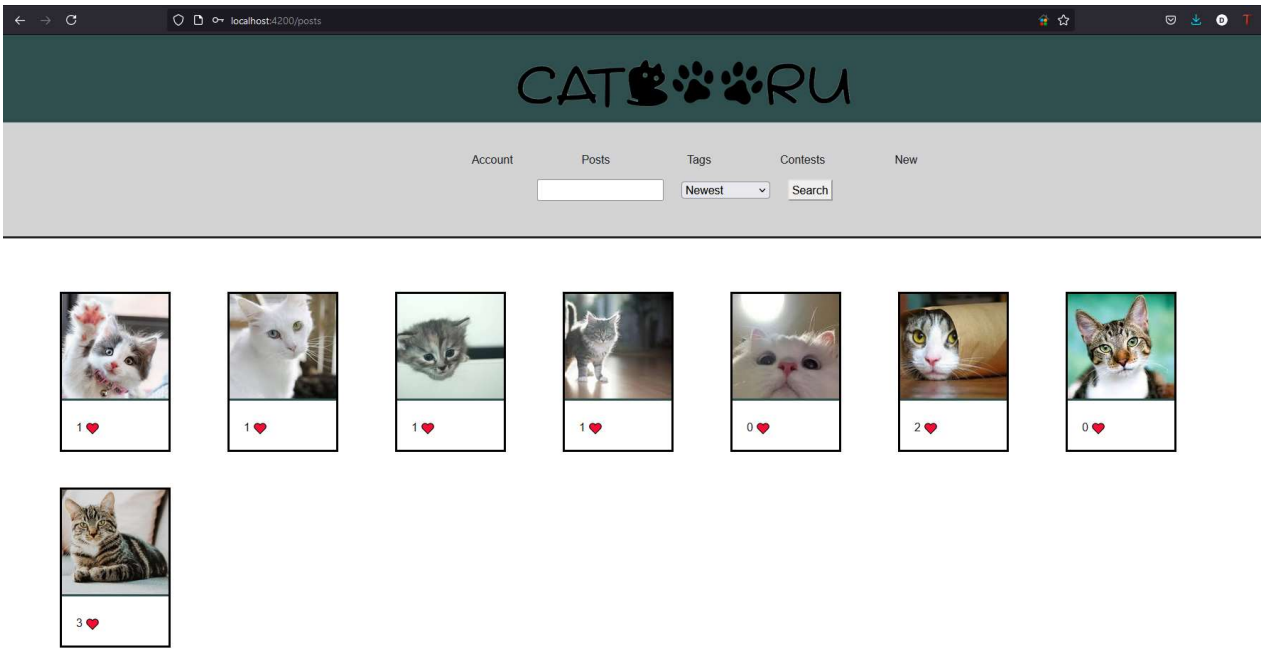
Kada se otvori pojedinačna slika, prikazana je u punoj veličini, ispod nje prikazano je ime korisnika koji ju je postavio, te u slučaju da je u natjecanju i ime natjecanja. Prikazan je i broj *like*-ova te i tipka za brisanje slike ako je prijavljen onaj korisnik koji ju je i postavio. Na lijevoj strani su oznake preko kojih se slika može pretražiti, primjerice, ako se u tražilicu upiše „*tabby*“ ova slika će se pojaviti među rezultatima. Oznake može brisati i dodavati svaki prijavljeni korisnik kako bi se slika po zajednici korisnika mogla što preciznije pretražiti. Na dnu se nalaze komentari korisnika na sliku. Ako korisnik nije prijavljen, pokazana mu je poruka da se prijavi kako bi mogao ostaviti komentar ili *like*. Za prijavu slike u natjecanje, potrebno je na stranici za natjecanja kliknuti na „*Enter new post into contest*“ ispod postojećeg natjecanja (slika 5.8.). Na stranicu natjecanja dolazi se klikom na „*Contests*“ u navigaciji. Na stranici za natjecanja prikazana su i završena natjecanja i natjecanja koja su još u tijeku. Ispod završenih natjecanja prikazana je pobjednička slika, dok je kod natjecanja u tijeku prikazan rok i poveznica za dodavanje slike u natjecanje. Korisnici administratori mogu dodati novo natjecanje klikom na „*Add contest*“ iznad popisa natjecanja. To ih odvodi na jednostavnu formu u kojoj se unosi ime, opis i trajanje natjecanja.




**Slika 5.8.** Stranica natjecanja

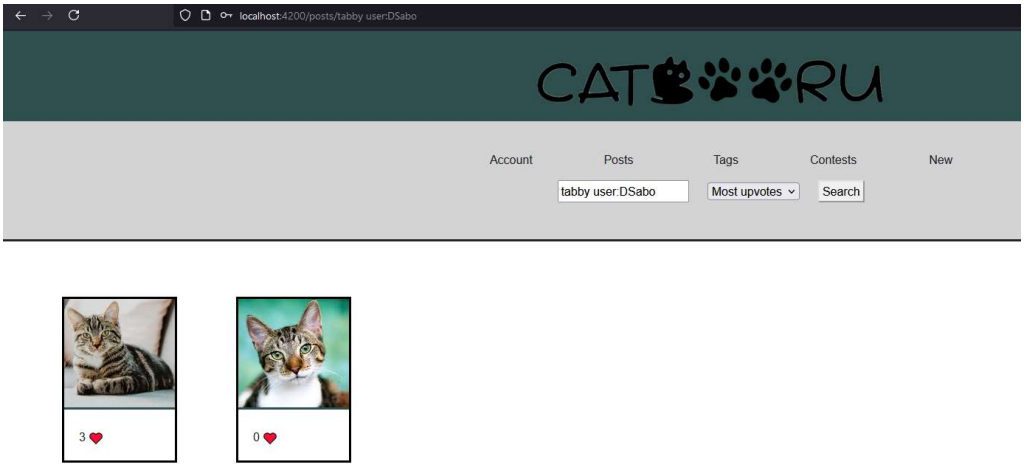
## 5.3. Pretraživanje slika

Klikom na „*Posts*“ u navigaciji prikazat će sve slike ikad postavljene na aplikaciju poslagane na način određen padajućim izbornikom. Uobičajen način pretraživanja je od najnovije slike prema najstarijoj (slika 5.9.). Moguće je pretražiti od najstarije prema najnovijoj, te po broju *like*-ova. Slike se slažu po načinu definiranim padajućim izbornikom bez obzira na koji način se slike pretražuje. Pretragom se prikazuju umanjene slike i broj *like*-ova koji ta slika ima. Klikom na „*Tags*“ u navigaciji, korisnika se odvodi na abecedni popis svih oznaka kojima su trenutno označene sve postojeće slike te se klikom na neku od njih izlistavaju sve slike s tom oznakom. Isti rezultat dobije se ako se upiše ta oznaka u tražilicu i pritisne na tipku „*Search*“ ili *enter* na tipkovnici.




**Slika 5.9.** Sve slike

Upisivanjem oznake u tražilicu prikažu se sve slike koje imaju tu oznaku. Ako se upiše više od jedne oznake, prikazuju se slike koje imaju obje oznake. Također, može se pretražiti po imenu korisnika ako se prije imena upiše „*user:*“ i po natjecanju ako se upiše „contest:“. Sve to se može kombinirati kako bi se primjerice pronašla slika s oznakom „*tabby*“ koju je postavio korisnik „*DSabo*“ (slika 5.10.).




**Slika 5.10.** Pretraga tražilicom po broju *like*-ova

# 6. ZAKLJUČAK

Web aplikacije koriste se u gotovo svim dijelovima svakodnevnog života, pogotovo za zabavu i razbibrigu. Ovaj rad zahtijevao je veliku količinu istraživanja web tehnologija i učenja kako se one koriste. Aplikacija je razvijena u dva dijela, Spring Boot API koji je u potpunosti razvijen u IntelliJ razvojnoj okolini te front end dijelu razvijenom u Visual Studio Code okolini. Ovu aplikaciju bilo je veoma poučno izrađivati jer se radilo s velikom količinom dosad nepoznatih tehnologija, od Spring Boot-a i Angular-a, pa sve do pokretanja virtualne slike na Docker-u. Aplikacija je zamišljena da bude nalik društvenoj mreži, web stranica na kojoj bilo tko može naći slike mačaka kakve želi vidjeti, bile one smiješne ili slatke. Također, korisnici mogu postaviti svoje slike kako bi mogli podijeliti sreću svog ljubimca s drugima, gdje god oni bili. Aplikacija ima razrađene sve funkcionalnosti potrebne za rad, no uvijek se može proširiti s još funkcionalnosti i podjela kako bi se mogla što jednostavnije navigirati te kako bi bila što optimiziranija. Tijekom razvoja front end dijela, bilo je potrebno staviti se na mjesto korisnika kako bi se aplikacija razvila što razumljivije i jednostavnije za korištenje.

# LITERATURA


[1] Pexels, dostupno na: https://www.pexels.com, (rujan 2021.)

[2] Cat Lovers Only, dostupno na: https://www.cat-lovers-only.com/, (rujan 2021.)

[3] Picato, dostupno na: https://www.picato.net/, (rujan 2021.)

[4] Cheezburger LOLcats, dostupno na: https://icanhas.cheezburger.com/lolcats, (rujan 2021.)

[5] Depositphotos, dostupno na: https://depositphotos.com, (rujan 2021.)

[6] JetBrains IntelliJ IDEA, dostupno na: https://www.jetbrains.com/idea/, (srpanj 2021.)

[7] Microsoft Visual Studio Code, dostupno na: https://code.visualstudio.com/, (srpanj 2021.)

[8] VMware, Inc. Spring Boot, dostupno na: https://spring.io/projects/spring-boot, (srpanj 2021.)

[9] VMware, Inc. Spring Initializr, dostupno na: https://start.spring.io/, (srpanj 2021.)

[10] Angular, dostupno na: https://angular.io/, (srpanj 2021.)

[11] Java, dostupno na: https://www.java.com/en/, (srpanj 2021.)

[12] TypeScript, dostupno na: https://www.typescriptlang.org/, (srpanj 2021.)

[13] HTML, dostupno na: https://developer.mozilla.org/en-US/docs/Web/HTML, (srpanj 2021.)

[14] CSS, dostupno na: https://developer.mozilla.org/en-US/docs/Web/CSS, (srpanj 2021.)

[15] PostgreSQL, dostupno na: https://www.postgresql.org/, (srpanj 2021.)

[16] Docker, dostupno na: https://www.docker.com/, (srpanj 2021.)

[17] Postman, dostupno na: https://www.postman.com/, (srpanj 2021.)

# SAŽETAK

U ovom diplomskom radu razvijen je web panel za postavljanje slika mačaka naziva Catbooru. Korisnici mogu pretraživati slike po oznakama ili korisnicima, komentirati ih i postavljati like-ove. Također, korisnici mogu postavljati vlastite slike te, ako žele, mogu ih staviti u natjecanja koja definiraju administratori stranice. Aplikacija je izrađena u Spring Boot radnom okviru u razvojnom okruženju IntelliJ, te u Angular radnom okviru u okruženju Visual Studio Code. Teorijski dio objašnjava tehnologije korištene za izradu aplikacije. Također, objašnjen je izgled i funkcija aplikacije popraćen slikama.

**Ključne riječi**: aplikacija, oznaka, panel, slika, web

# ABSTRACT

**Web imageboard application**

In this master's thesis, a web imageboard for pictures of cats named Catbooru was developed. Users can search posts using tags or usernames, leave comments or like them. Also, users can upload their own pictures and even enter them into contests that administrators create. The application was made using the Spring Boot framework in the IntelliJ IDE (back end) and using the Angular framework in the Visual Studio Code IDE (front end). The theoretical part of the thesis explains the technologies used to create the application. Using pictures as well as text, the design and function of the application is also accurately described.

**Keywords**: apps, imageboard, picture, tag, web