Web sustav za označavanje i pripremu medicinskih slika za primjene strojnog učenja

Bak, Luka

`SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

WEB SUSTAV ZA OZNAČAVANJE I PRIPREMU MEDICINSKIH SLIKA ZA PRIMJENE STROJNOG UČENJA

Završni rad

Luka Bak

Osijek, 2023

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Luka Bak
Studij, smjer:	Sveučilišni prijediplomski studij Programsko inženjerstvo
Mat. br. pristupnika, god.	R4172, 11.10.2021.
JMBAG:	0165076756
Mentor:	doc. dr. sc. Hrvoje Leventić
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Web sustav za označavanje i pripremu medicinskih slika za primjene strojnog učenja
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	Istražiti i opisati postojeće sustave za označavanje i pripremu podataka za primjenu strojnog učenja. Opisati neke od primjena medicinskih slika za strojno učenje, koje arhitekture se koriste za kakve medicinske slike te proces kreiranja označenih medicinskih podataka. Za praktični dio izraditi web aplikaciju koja će omogućiti medicinskim stručnjacima označavanje prikupljenih medicinskih slika i izvoz podataka za potrebe strojnog učenja. Rezervirano za: Luka Bak

Datum prijedloga ocjene završnog rada od strane mentora:	15.09.2023.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	24.09.2023.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	02.10.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 02.10.2024.

Ime i prezime Pristupnika:	Luka Bak
Studij:	Sveučilišni prijediplomski studij Programsko inženjerstvo
Mat. br. Pristupnika, godina upisa:	R4172, 11.10.2021.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Web sustav za označavanje i pripremu medicinskih slika za primjene strojnog učenja**

izrađen pod vodstvom mentora doc. dr. sc. Hrvoje Leventić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

U vremenu nezaustavljivog rasta procesorske snage modernih računala i kontinuiranog usavršavanja procesa strojnog učenja, polje medicine ističe se kao logičan izbor za primjenu i usavršavanje tih tehnologija. Mogućnost kvalitetne integracije metoda strojnog učenja i umjetne inteligencije u postojeće istraživačke procese medicinskih stručnjaka donosi značajan potencijal za revolucioniranje cijele medicinske znanosti.

Posebno se ističu područja medicinskih istraživanja u kojima postoji iznimno velika količina raznolikih uzoraka i skupova podataka koje čovjek ne može obraditi dovoljno učinkovito i precizno. Modeli strojnog učenja trenirani na velikim količinama rendgenskih slika, MRI skenova ili biopsijskih uzoraka već i danas pomažu pri detekciji tjelesnih anomalija i segmentaciji malignih tumora. Optimizacija postojećih i stvaranje novih, poboljšanih modela otvara mogućnost za drastična poboljšanja u preciznosti medicinskih dijagnoza te stvara uvjete za istraživačke pothvate u području medicine koji su prije nekoliko desetaka godina bili gotovo nezamislivi.

Ovaj rad opisuje razvoj i rad web sustava kojemu je cilj pružiti medicinskim stručnjacima efikasnu i preglednu platformu pomoću koje na jednostavan način mogu učitati, označiti i u konačnici izvesti veliku količinu podataka.

Drugo poglavlje ovog rada osvrće se na zastupljenost strojnog učenja u medicini te postojeća programska rješenja koja omogućuju integraciju računarstva i medicine. Treće poglavlje govori o tehnologijama pomoću kojih je izrađen ovaj web sustav te se ukratko opisuju korištene komponente tih tehnologija. Četvrto poglavlje prikazuje cjelokupni razvojni put web sustava, od specifikacije zahtjeva do evaluacije konačnih rezultata. Peto poglavlje demonstrira rad web sustava, prikazuje se izgled korisničkog sučelja te se govori o korisničkom iskustvu za vrijeme korištenja web sustava i njegovih komponenti. Šesto, ujedno i posljednje, poglavlje odnosi se na zaključak ovog završnog rada.

1.1 Zadatak završnog rada

Zadatak ovog završnog rada je izrada web sustava koji će omogućiti medicinskim stručnjacima da označe prikupljene medicinske slike. Nakon što su slike označene, potrebno je omogućiti

korisnicima platforme izvoz podataka u prikladnom formatu kako bi se ti podaci mogli koristiti u svrhe strojnog učenja. Korisnicima koji imaju ulogu administratora potrebno je omogućiti dodavanje novih slika i definiranje mogućih oznaka za iste. Učitane slike potrebno je predobraditi kako bi one bile u optimalnom formatu za označavanje i pohranu. Administratorima se treba omogućiti upravljanje korisnicima web sustava.

2. PREGLED STROJNOG UČENJA U MEDICINI I POSTOJEĆIHPROGRAMSKIH RJEŠENJA

Primjene strojnog učenja u medicini su brojne te svoju svrhu najčešće pronalaze u istraživačkim poljima i poljima kliničke dijagnostike. Posljednjih nekoliko godina postoji veliki interes za primjenu metoda strojnog učenja na velikim skupovima podataka kojima medicinske ustanove rukuju.[1] Ti podaci mogu biti u obliku anonimiziranih elektroničkih zapisa i bilješki o liječenju pacijenata, medicinskih slika poput rendgenskih i biopsijskih slika i sl.

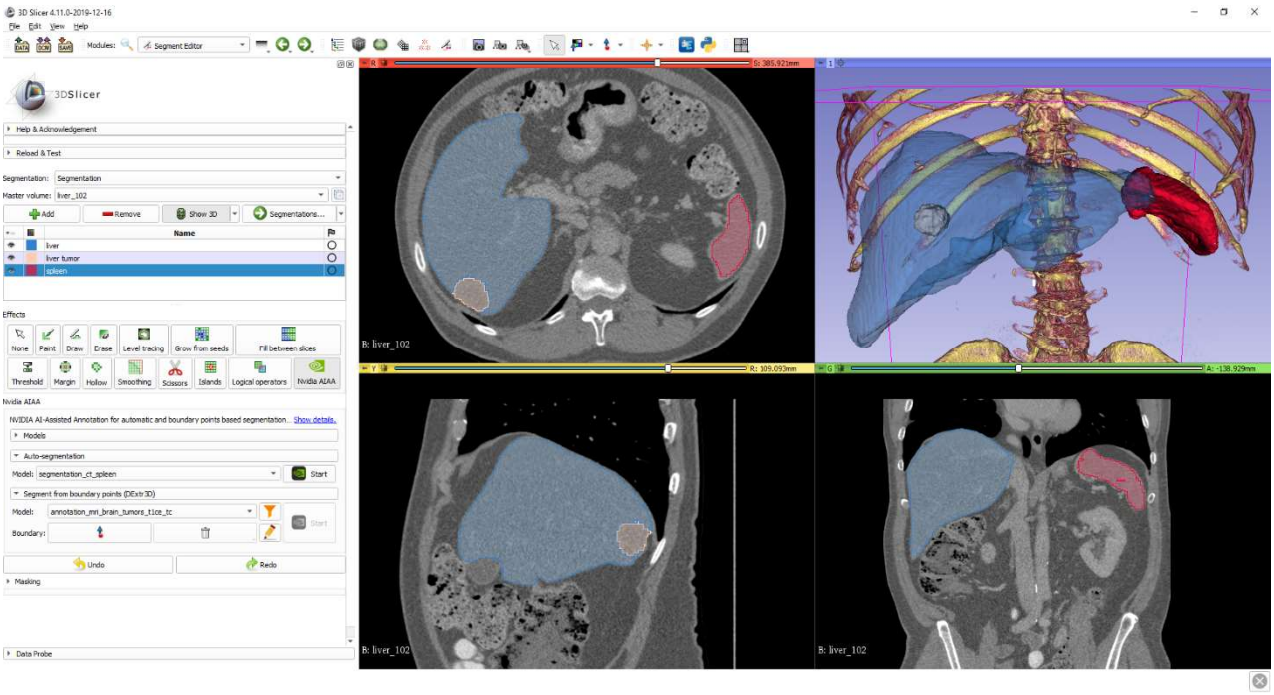
Modeli strojnog učenja generirani pomoću tih podataka imaju mogućnost analizirati velike i kompleksne podatkovne setove, uočavati uzorke i nove zakonitosti koje čovjek ne može primijetiti te pomoći medicinskom stručnjaku donijeti ispravnu i pravovremenu odluku. Neki od tih modela već i danas daju bolje rezultate nego medicinski stručnjaci.Primjerice, rad japanskih stručnjaka[2] prikazuje da je model treniran sa 73 značajke (14 ultrazvučnih slika, 54 rezultata testova krvi i 5 podatkovnih setova o pacijentima) bio bolji u detekciji recidiva reumatoidnog artritisa nego medicinski stručnjaci koji su imali pristup istim podacima.

2.1 Tipičan proces izrade označenih medicinskih podataka

[1] Proces izrade označenih medicinskih podataka započinje sa sakupljanjem relevantnih medicinskih slika iz raznih izvora kao što su istraživačke baze podataka, bolnice ili javno dostupni podatkovni skupovi. Kako bi prikupljene slike mogle biti označene potrebno ih je predobraditi. Ukoliko su prikupljeni podaci volumetrijski, potrebno je izdvojiti željene 2D presjeke ili označiti specifične interesne regije. U tu svrhu najčešće se koriste programi za segmentaciju i vizualizaciju medicinskih slika otvorenog koda (engl. „opensource“) poput 3D Slicer [3] (slika 2.1), BioImageXD [4], ImageJ [5] iitk-SNAP [6] ili njihovi komercijalni pandani poput OsiriX DICOM [7] programa.

Nakon što su slike pripremljene, sljedeći korak je proces označavanja. Tijekom procesa označavanja medicinski stručnjaci ili obučeni anotatori označavaju slike identificirajući tako specifične značajke, razne organske strukture ili anomalije. Oznake se mogu izvoditi na različitim razinama preciznosti; regionalno označavanje je brzo i efikasno, dok je označavanje na razini piksela zahtjevno ono pruža znatno veću preciznost te nerijetko i bolje rezultate[8]. Neki od

prethodno navedenih programa za vizualizaciju i segmentaciju imaju ugrađenu podršku za označavanje slika, ali zbog specifičnosti problema označavanja nerijetko se koriste programi kojima je ono primarna zadaća. Takvi programi olakšavaju proces označavanja pomoću naprednih mogućnosti kao što su interpolacija oznaka, simultani rad više anotatora i polu-automatska anotacija. Kako takvi programi nisu nužno vezani samo uz proces označavanja medicinskih slika, već se koriste u razne svrhe strojnog učenja, za problem označavanja slika postoji pregršt programskih rješenja na tržištu. U nastavku ovog poglavlja prikazuju se funkcionalnosti nekoliko popularnih programskih rješenja za označavanje slika.



Slika 2.1 – Prikaz segmentacije medicinskih slika i korisničkog sučelja aplikacije 3D Slicer [3]

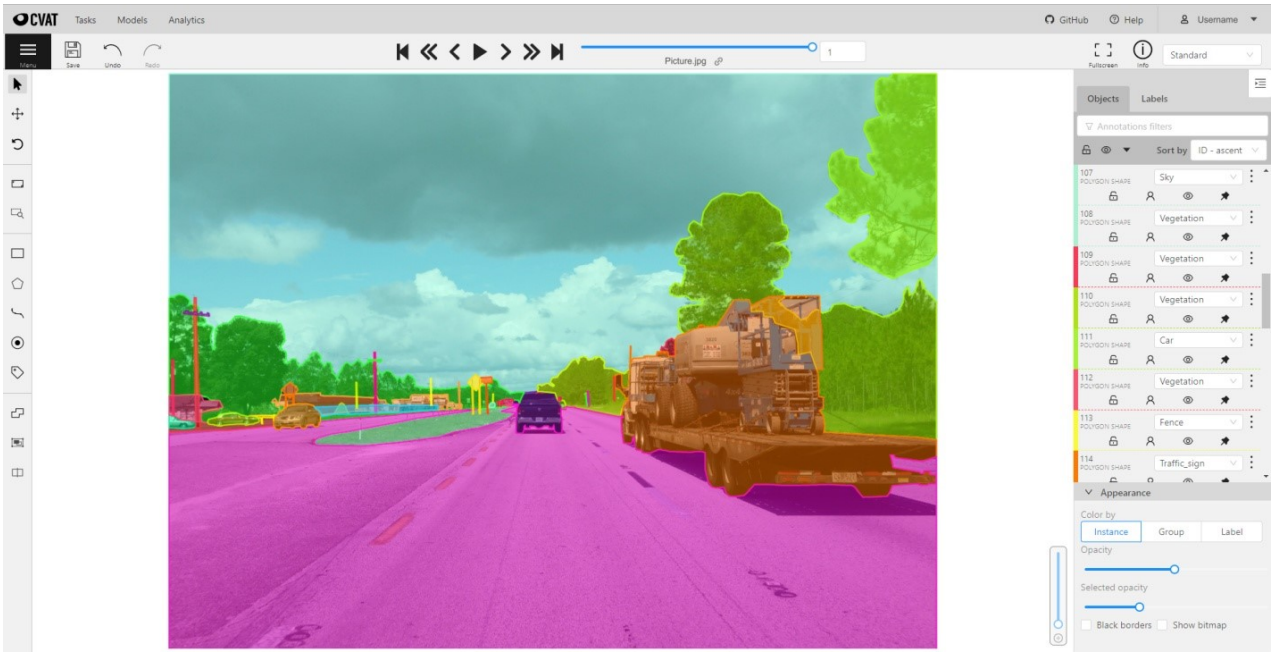
2.2 CVAT

CVAT (engl. Computer VisionAnnotation Tool) je besplatna web platforma otvorenog izvora koja se koristi za označavanje slika i videozapisa (slika 2.2). Bazirana na biblioteci OpenCV, platforma je razvijena od strane Intel korporacije 2018. godine. Nedugo nakon objavljivanja, 2019. godine Intel odlučuje ponuditi platformu kao besplatno rješenje te izvorni kod daju na korištenje prema

MIT licenci. Platforma uz instance aplikacije koje korisnik održava sam (engl. „self-hosted“) pruža i plaćenu verziju koja omogućuje korisniku da hosting i održavanje prepusti CVAT organizaciji.

CVAT platforma napisana je koristeći biblioteku React, koristeći TypeScript superset JavaScript programskog jezika. Pozadinski dio platforme (engl. „backend“) napisan je koristeći razvojni okvir Django u programskom jeziku Python.

CVAT platforma, osim osnovnih funkcionalnosti označavanja podataka, nudi svojim korisnicima i nekoliko naprednih mogućnosti. Interpolacija oblika između dva isječka videozapisa omogućuje korisniku da izbjegne označavanje svake sličice u videu te tako štedi veliku količinu vremena. Polu-automatsko anotiranje pomoću algoritama dubokog učenja omogućuje korisniku da iskoristi automatsko anotiranje nakon malog broja ručno unesenih oznaka.

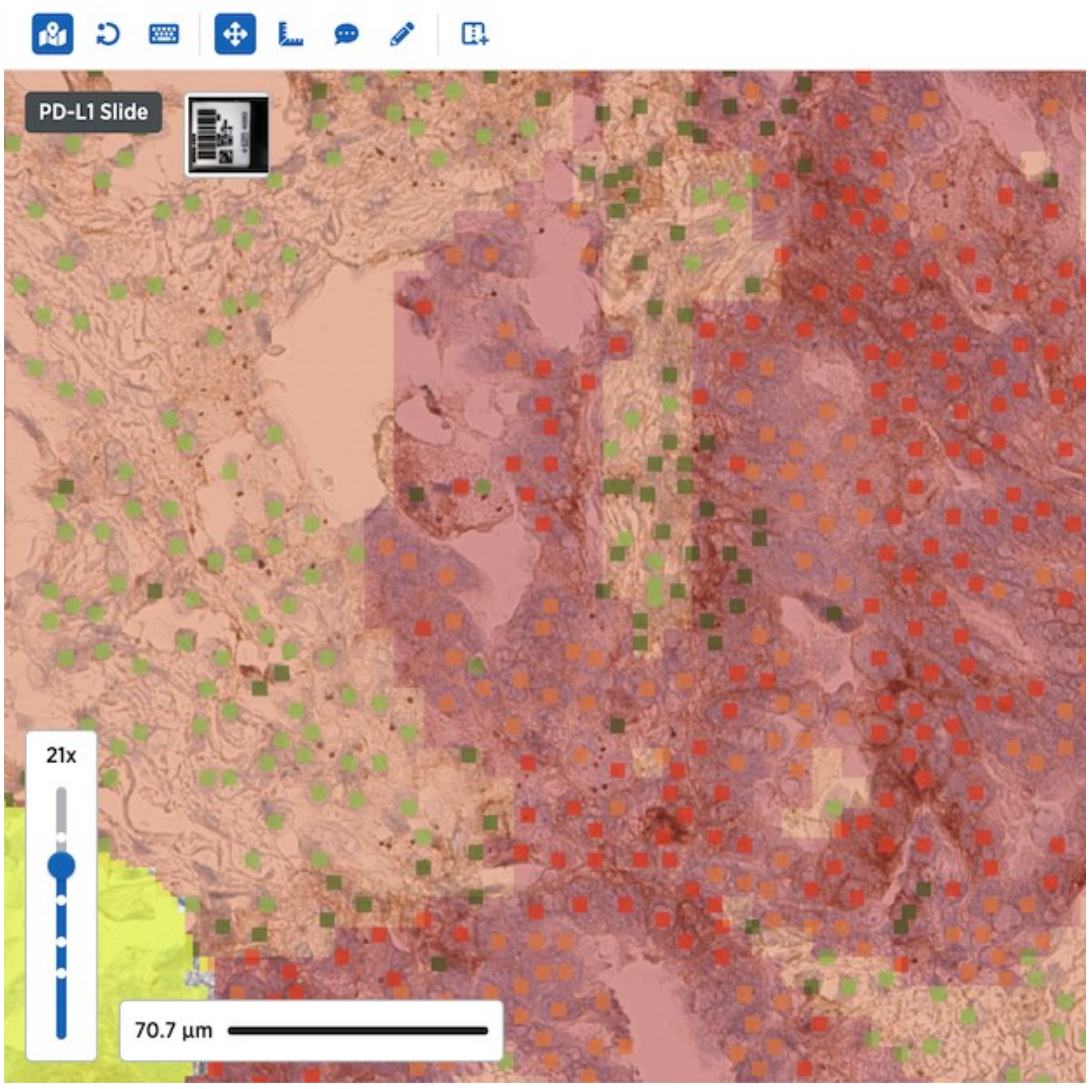


Slika 2.2 – Prikaz procesa označavanja u web platformi CVAT

2.3 PathAI

PathAI je platforma namijenjena stručnjacima patologije, potpomognuta algoritmima dubokog učenja. Platforma pruža sustave koji mogu prepoznati i istaknuti područja od potencijalnog interesa u uzorcima tkiva. Istraživačima i medicinskim stručnjacima platforma pruža više od 50 milijuna slajdova tkiva i više od 15 milijuna validiranih anotacija. Platforma nije dostupna široj publici, a cijena korištenja nije javno dostupna.

Primjer jednog od modela (AIM-PD-L1 NSCLC)[9] koje platforma pruža prikazan je na slici 2.3. Model omogućuje automatsku detekciju raka malih stanica pluća te prikladnu vizualizaciju pogođenih područja.

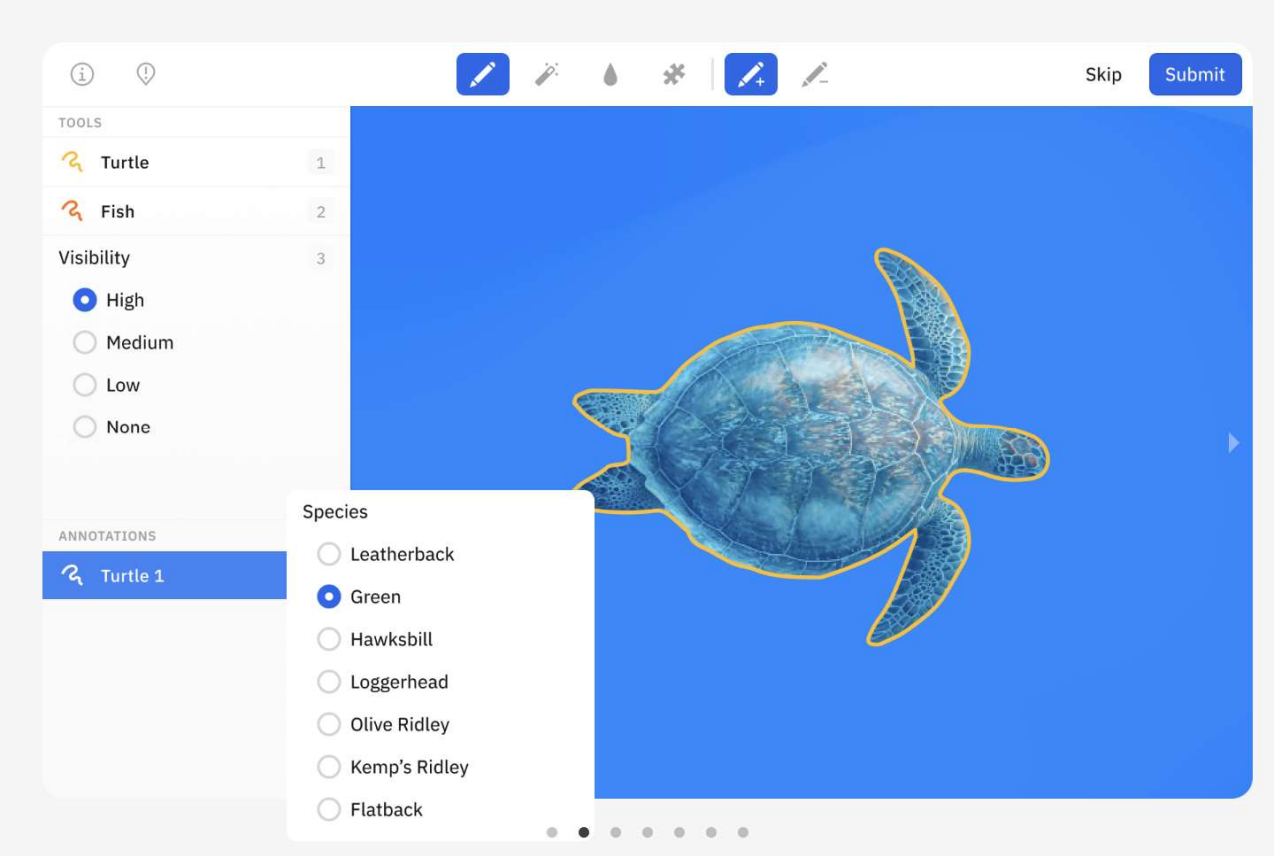


Slika 2.3 - Automatski anotirane stanice tumora u presjeku plučnog tkiva [9]

2.4 Labelbox

Labelbox je online platforma za označavanje podataka u svrhe strojnog i dubokog učenja. Platformu 2018. godineosnivaju Manu Sharma, Dan Rasmuson i Brian Rieger u San Franciscu, SAD. Uz označavanje platforma nudi svojim korisnicima simultano označavanje, automatizaciju poslova, ručnu i automatsku validaciju oznaka, automatsku segmentaciju i poluautomatsko označavanje uz ljudski nadzor. Podaci se mogu izvesti u raznim oblicima te je korisnicima ponuđeno kreiranje i validiranje modela direktno u samoj platformi. Primjer procesa označavanja korištenjem Labelbox platforme prikazan je na slici 2.4.

Platforma pruža vrhunska sučelja za označavanje svih vrsta slika te se lako konfigurira za sve potrebe, uključujući i one medicinske. Besplatna razina postoji, ali je ograničena na male setove podataka.



Slika 2.4 - anotacijsko sučelje Labelbox platforme

3. TEHNOLOGIJE KORIŠTENE ZA IZRADU RADA

3.1 Docker

Docker je platforma koja korisnicima pruža tehnologiju kontejnerizacije tijekom razvoja, implementacije i pokretanja aplikacija. Kontejneri su lagane i izolirane virtualne računalne okoline koje sadrže kreirane aplikacije te sve biblioteke, programe i konfiguracijske datoteke o kojima ta aplikacija ovisi. Docker kontejneri se mogu izvršavati na svim Windows, Linux i MacOS uređajima. Takav pristup razvoju aplikacija omogućuje aplikacijama da se konzistentno i točno izvršavaju na različitim računalnim okruženjima i infrastrukturama.

Za razliku od virtualnih strojeva, koji također pružaju virtualnu okolinu, Docker kontejneri ne virtualiziraju računalno sklopovlje već samo traženi operacijski sustav, tj. Docker kontejneri dijele jezgru operacijskog sustava sa računalom na kojim se pokreću te koriste zajedničke biblioteke i resurse. Takav princip rada omogućuje jedno fizičko računalo da istovremeno pokreće znatno veći broj Docker kontejnera nego što bi bilo moguće s virtualnim strojevima.

Docker kontejneri se pokreću iz generiranih Docker slika. Korisnik može učitati ili preuzeti generirane Docker slike iz raznih javnih ili privatnih Docker registara. Docker slike se generiraju pomoću Dockerfile datoteke (isječak koda 3.1) u kojoj su opisani traženi operacijski sustav, vanjske ovisnosti, konfiguracijske datoteke i varijable te proces pokretanja željene aplikacije.

```
FROM ruby:3.2.1
RUN apt-get update -qq && apt-get install -y nodejs postgresql-client postgresql-server-dev-
all
RUN mkdir /myapp
WORKDIR /myapp
COPY Gemfile /myapp/Gemfile
COPY Gemfile.lock /myapp/Gemfile.lock
RUN bundle install --deployment --without development test
COPY . /myapp
ENV RAILS_ENV production
COPY entrypoint.sh /usr/bin/
RUN chmod +x /usr/bin/entrypoint.sh
ENTRYPOINT ["entrypoint.sh"]
EXPOSE 80
CMD ["bundle", "exec", "rails", "server", "-b", "0.0.0.0", "-p", "80"]
```

Isječak koda 3.1 – primjer Dockerfile datoteke

3.2 JavaScript i TypeScript

3.2.1 JavaScript

JavaScript je interpretirani (ponekad i JIT kompajlirani), dinamično pisani, slabo tipizirani programski jezik visoke razine. Svoju primjenu pronalazi ponajviše kao skriptni jezik web stranica, no u prethodnom desetljeću postiže popularnost i u okruženjima izvan web preglednika, poglavito u serverskom okruženju Node.js.

JavaScript podržava brojne programerske paradigme kao što su: dinamičko pisanje, funkcije prve klase, programiranje vođeno događajima, funkcionalno, imperativno i objektno orijentirano programiranje i sl.

Nastaje 1995. godine suradnjom između Netscape Navigatora, najpopularnijeg Internet preglednika u to vrijeme, i Sun Microsystems, kreatora Java programskog jezika.[10] Nastaje zbog potrebe za standardiziranim načinom dodavanja interaktivnosti na klijentskoj strani web stranica. Iako Sun Microsystems (danas Oracle) drži patent za ime JavaScript koje ovaj jezik dijeli sa Java programskim jezikom, njihova sličnost je zanemariva te je to zbunjujuće imenovanje plod marketinga iz Netscape korporacije.Dizajn JavaScripta prati ECMAScript specifikaciju jezika, ECMA-262[11]

3.2.2 TypeScript

TypeScript je superset JavaScripta, kreiran od strane Microsofta 2012. godine. TypeScript u svijet JavaScripta donosi statično tipiziranje varijabli te opcionalno anotiranje varijabli i povratnih vrijednosti funkcija. Svoju ulogu nalazi ponajprije u velikim bazama koda pisanim u JavaScriptu, gdje prije spomenute funkcionalnosti donose veću sigurnost koda, otkrivanje grešaka prije izvršavanja koda kroz statičku analizu i bolju podršku za razvojne okoline (linteri, jezični serveri i sl.). TypeScript kod se ne izvodi direktno već se on prvotno transpajlira u JavaScript kod.

```
function add(a: number, b: number): number {
  return a + b
}
```

Isječak koda 3.2 – statički tipizirana funkcija u TypeScriptu

Primjer koda dan je isječkom koda 3.2, gdje primjećujemo kako su u funkciji koja zbraja dva broja striktno definirani tip i broj parametara. Funkcija također ima i striktno definirani povratni tip podatka te će u slučaju nepodudarajućih tipova parametara ili povratnog podataka biti registrirana greška. Važno je napomenuti da se sve tipizirane anotacije, sučelja i aliasi tipova uklanjaju tijekom transpajliranja TypeScripta u JavaScript (engl. „typeerasure“)[12], tj. TypeScript služi svrsi samo tijekom razvoja aplikacije te je kod koji se u konačnici pokreće na korisničkim računalima i dalje JavaScript.

3.3 React

React je besplatna biblioteka otvorenog izvora pisana u JavaScriptu namijenjena za izradu modernih korisničkih sučelja za web stranice. Biblioteka je originalno razvijena u Facebooku 2013. godine za njihove interne potrebe, no ubrzo je objavljena javno uz MIT licencu.

React aplikacije grade se od manjih, izoliranih dijelova koda, takozvanih komponenti. Iako je u Reactu omogućeno pisanje čistog HTML-a, etablirana je praksa korištenja JSX-a. JSX je ekstenzija jezične sintakse JavaScripta koja omogućuje React programerima da direktno miješaju JavaScript kod sa regularnim HTML-om, naspram indirektne manipulacije DOM-om kroz posebne JavaScript funkcije. Na taj način programer ne mora odvajati logiku označnih jezika i programsku logiku u posebne sekcije ili datoteke. Na isječku koda 3.3 prikazana je jednostavna komponenta koja uz korištenje uvjetnog prikazivanja demonstrira integraciju HTML-a i JavaScript koda.

```
const App = () => {
    const i = 1;

    return (
      <div>
        <h1>{ i === 1 ? 'true' : 'false' }</h1>
      </div>
    );
 }
```

Isječak koda 3.3 – prikaz jednostavne React komponente koja kao povratnu vrijednost vraća JSX

Kako se React smatra bibliotekom bez čvrstog stava (engl. „unopinionated“), to znači da ne nameće određene arhitekture ili dizajnerske smjernice. Programeri će sve osim osnovnih funkcionalnosti morati implementirati sami ili pomoć potražiti u nekom od brojnih programskih okvira koji su temeljeni na React biblioteci, kao primjerice Next, Remix ili Astro.

3.3.1 React hooks

React kukice (engl. „hooks“) su dio React biblioteke koje omogućuju upotrebu reaktivnog stanja („useState“ hook), upravljanja nuspojavama („useEffect“ hook), i konteksta („useContext“ hook) u funkcionalnim komponentama umjesto prethodno popularnih metoda životnog ciklusa unutar klasnih komponenti.

Kako programer ne mora brinuti o inicijalizaciji i životnom vijeku funkcionalnih komponenti, omogućeno mu je da kreira kompleksne prilagođene hookove bazirane na onim ugrađenima. Te prilagođene hookove programer može iskoristiti za enkapsulaciju logike koja se koristi kroz više komponenti. Takav pristup razvoju React komponenata pridonosi pojednostavljenju razvoja, poboljšava čitljivost koda te omogućava programeru da lakše poopći svoje komponente te ih iskoristi više puta na više mjesta.

3.4 Ruby

Ruby je objektno orijentirani, dinamično pisani, interpretirani (u novijim verzijama, JIT kompajlirani), snažno tipiziran, programski jezik visoke razine. Svoju primjenu pronalazi kao skriptni jezik CLI alata (Brew - upravitelj paketa, Chef - automatizacija infrastrukture i sl.) i u svijetu programskih okvira za izradu web aplikacija (Ruby on Rails, Sinatra, Jekyll i sl.).

Ruby je karakteriziran potpunom primjenom objektno orijentiranih načela, svaka vrijednost predstavlja objekt, uključujući klase i tipove koje su u mnogim drugim jezicima primitivni tipovi (brojevi, Booleove vrijednosti, „nil“ vrijednost). [13]

Ruby podržava brojne programerske paradigme poput funkcionalnog programiranja (anonimne i lambda funkcije), proceduralnog programiranja (metode definirane izvan klase pripadaju korijenskoj *Object* klasi) i metaprogramiranja.

Kreirao ga je 1995. godine YukihiroMatsumoto zbog želje da napravi ozbiljan objektno orijentirani jezik kojim bi zamijenio Perl, s kojim nije bio zadovoljan[13].

3.5 Ruby on Rails

Ruby on Rails, često poznat jednostavno kao Rails, je popularni okvir za razvoj web aplikacija otvorenog koda napisan u Ruby programskom jeziku. Napisao ga je i predstavio 2004. godine David HeinemeierHansson, poznatiji kao DHH u popularnoj programerskoj kulturi. Postaje popularan zbog pregršt ugrađenih funkcionalnosti i snažnom naglasku na konvenciju umjesto konfiguracije,[14] što programerima omogućuje brz razvoj aplikacija s velikim brojem mogućnosti.

Ruby on Rails baziran je na *model-view-controller* (MVC) paradigmi koja odvaja prezentacijski sloj (*view*) od poslovne logike (*model*) koji su u konačnici povezani kontrolerima(*controller*).

Ruby on Rails sadržava razne posrednike (engl. „middleware“) i module koji omogućuju rapidni razvoj web aplikacija. Neki od njih su:

- *ActiveRecord* - ugrađeni ORM koji podržava sve relevantne baze podataka
- *ActiveView*- uz pomoć ERB predložaka generira dinamične web stranice
- *ActiveMailer* - uz pomoć ERB predložaka generira i šalje email poruke
- *ActiveStorage*- lokalna ili udaljena pohrana datoteka
- *Hotwire, ActiveCable*- izrada dinamičkih web stranica pomoću WebSocket tehnologije
- *ActiveJob*- sustav izvođenja pozadinskih poslova

Također ugrađen u razvojni okvir je i moćni CLI alat koji omogućuje razne akcije, od generiranja novih aplikacija, modela i kontrolera, reverzibilnih migracija baze podataka pa sve do automatskog generiranja cjelina poslovne logike pomoću*scaffold*naredbe.

3.6 Fabric.js

Fabric.js je popularna JavaScript biblioteka otvorenog kod za rad sa HTML5 platnima (engl. „canvas“). HTML5 platna omogućuju razvoj alata za crtanje, uređivanje slika i raznih drugih multimedijskih aplikacija. Kako je aplikacijsko sučelje HTML5 platna vrlo jednostavno, svaka imalo zahtjevnija funkcionalnost iziskuje znatnu količinu programskog koda i veliki ulog vremena od strane programera. Iz togje razloga JuriyZaytsev 2010. godine predstavio Fabric.js.

Fabric.js predstavlja nadogradnju na postojeći sustav HTML5 sustav platna. Biblioteka sadrži veliki skup raznih funkcija i metoda za rad s elementima platna, uključujući kreiranje oblika, tekstualnih elemenata, slika, grupiranje elemenata, skaliranje, rotaciju i mnoge druge. U isječku koda 3.4 prikazan je kod koji na platno postavlja obojani krug.

```
const circle = new fabric.Circle({
    left: pointer.x,
    top: pointer.y,
    originX: 'center',
    originY: 'center',
    radius: 10,
    fill: toRGBA(color, 0.3),
    strokeWidth: 2,
    stroke: color,
    selectable: true,
    hasControls: true,
    lockScalingX: false,
    lockScalingY: false,
    hasRotatingPoint: true,
    strokeUniform: true,
});
canvas.add(circle);
```

Isječak koda 3.4 -dodavanje kruga na Fabric.js platno

Jedna od najvažnijih mogućnosti ove biblioteke je mogućnost rukovanja s događajima, tj. korisničkim interakcijama s platnom. Tako je moguće definirati funkcije povratnog poziva (engl. „callbackfunction“) koje obavljaju određene poslove u skladu s korisničkim akcijama. Podržani su razni događaji poput dodavanja, promjene veličine i uklanjanja objekata, kao i akcije s mišem i tipkovnicom poput klika, dvoklika i pritiskanja tipki na tipkovnici. Tako se, primjerice, pomoću koda u isječku koda 3.5 omogućuje korisniku da okretanjem kotačića na mišu zumira i odzumira Fabic.js platno.

```
canvas?.on("mouse:wheel", function (opt) {
  var delta = opt.e.deltaY;
  var zoom = canvas!.getZoom();
  zoom *= 0.999 ** delta;
  if (zoom > 20) zoom = 20;
  if (zoom < 0.01) zoom = 0.01;
  canvas!.zoomToPoint({ x: opt.e.offsetX, y: opt.e.offsetY }, zoom);
  opt.e.preventDefault();
  opt.e.stopPropagation();
});
```

Isječak koda 3.5 - funkcija povratnog poziva za zumiranje Fabric.js platna

3.7 PostgreSQL

PostgreSQL, često poznat jednostavno kao Postgres ili pgsql, besplatni je sustav za upravljanje relacijskim bazama podataka otvorenog koda. Nastaje 1996. godine na Berkeley sveučilištu u Kaliforniji kao nadogradnja na prethodni sustav, Ingres, po čemu i dobiva ime[15].

PostgreSQL baze podataka podržavaju reverzibilne transakcije i imaju punu implementaciju ACID svojstava. Kreirane tablice imaju podršku za poglede (engl. „views“), okidače (engl. „triggers“), strane ključeve i pohranjene procedure.

Sustav upravlja istodobnim zahtjevima pomoću *Multiversionconcurrencycontrol*(MVCC) protokola koji svakoj transakciji daje pristup tzv. snimci baze podataka, što dopušta istodobnim transakcijama da mijenjaju bazu u isto vrijeme bez zaključavanja određenih tabličnih unosa.

PostgreSQL podržava razne tipove podataka, uključujući brojeve različite preciznost, nizove, tekst i slike. Od velike važnosti za ovaj web sustav su također i JSON i JSONb polja koje Postgres pruža. JSON polja donose mogućnost spremanja nestrukturiranih podataka uz podršku za indeksiranje i pisanje upita bez potrebe za uključivanje dodatnih, nerelacijskih baza podataka.

3.8 CSS i SCSS

3.8.1 CSS

CSS je kaskadni stilski jezik koji se koristi za definiranje izgleda XML, HTML i SVG elemenata u internetskim pretraživačima. Prve verzije web stranica sadržavale su sav svoj (limitirani) stil u HTML elementima čineći te datoteke teško čitljivima. Uz to, programeri su željeli održavati stil stranica odvojeno od podatkovne strukture HTML-a[16]. Iz tog razloga nastaje CSS. Prva verzija izdana je 1996. godine te se od tada CSS održava pod okriljem W3 konzorcija (W3C)

3.8.2 SCSS

SCSS je superset CSS-a koji uvodi napredne značajke poput varijabli, funkcija i ugniježđena stilska pravila. SCSS je jedna od dvijusintaksnih vrsta SASSCSS predprocesora. SCSS se koristi uglavnom kako bi programeru olakšao pisanje stilova koji se mogu upotrijebiti na više mjesta i kako bi se olakšalo pisanje stilova za duboko ugniježđene elemente.

Iako je i dalje popularan, SCSS zahtjeva korak interpretacije kako bi se pretvorio u CSS kojega pretraživač razumije te mu popularnost lagano opada kako standardna CSS specifikacija polako dobiva sve mogućnosti koje SCSS i SASS nude.

3.9 Bulma

Bulma je je besplatni okvir za stiliziranje web stranica otvorenog koda. Bulma se sastoji od gotovih stilskih komponenti te omogućuje programeru da stilizira stranicu te je prilagodi svim uređajima i veličinama ekrana bez pisanja velike količine CSS-a. Bulma je bazirana na modernim CSS tehnologijama te za razliku od sličnih rješenja ne zahtijeva uključivanje dodatnih ovisnosti u projekt.

Uz veliki broj gotovih komponenti kao što su sekcije, tipke, kontejneri i naslovi,Bulma donosi i veliki broj utilitarnih pomagača poput margina, razmaka između paragrafa ili slova i boja koje programer može slobodno primjenjivati na postojećim HTML elementima kako bi ih prilagodio svojoj specifičnoj potrebi.

4. REALIZACIJA WEB SUSTAVA

Cilj ovog rada je izraditi web sustav koja će omogućiti medicinskim stručnjacima označavanje prikupljenih medicinskih slika i izvoz podataka za potrebe strojnog učenja.

4.1 Specifikacija zahtjeva

U ovom poglavlju obrađuju se i razlažu zahtjevi koje ovaj web sustav mora ispunjavati.

Postavljeni zahtjevi:

1. Mogućnost prijave korisnika
2. Podjela korisnika na administratore i regularne korisnike
3. Upravljanje korisnicima od strane administratora (dodavanje, brisanje, uređivanje)
4. Kako otvorena registracija nije omogućena, slanje elektroničke pošte sa detaljima računa nakon što ga administrator kreira
5. Dinamično definiranje više kategorija slika koje trebaju biti anotirane
6. Dinamično definiranje više vrsta anotacija za svaku kategoriju slika
7. Učitavanje slika za pojedinu kategoriju
8. Učitavanje većeg broja slika od jednom
9. Omogućiti prikaz i filtriranje slika administratorima:
    a. Filtriranje po korisniku i statusu anotiranja (prazno, započeto, dovršeno)
    b. Prikaz informacija o pojedinoj slici, uključujući i trenutni status anotacija
10. Omogućiti anotiranje slika:
    a. Prijavljeni korisnici jednostavno mogu započeti proces anotacije
    b. Korisnici trebaju moći postaviti grafičke anotacije na učitanu sliku u obliku kruga/elipse ili slobodnog oblika sa proizvoljnim brojem stranica
    c. Korisnik mora moći spremiti napredak te ponovo pokrenuti proces anotiranja iste slike u drugo vrijeme i drugom uređaju bez da izgubi podatke
    d. Korisnik može u isto vrijeme imati samo dvije aktivne slike, prije nego što učita novu mora spremiti i završiti jednu od prethodno aktivnih
    e. Slika se pri procesu anotiranja mora moći zumirati i pomicati bez gubitka pozicijske preciznosti anotacija

11. Omogućiti učinkovito anotiranje slika:
    a. Kako slike koje će se anotirati sadrže značajan broj elemenata koje je potrebno označiti potrebno je omogućiti anotiranje sa tipkama miša (lijevi klik, dvostruki lijevi klik, desni klik) bez otvaranja izbornika za svaku anotaciju
    b. Omogućiti korisniku da samostalno promijeni odabrane tipke miša za željeni tip anotacije
    c. Omogućiti dinamično mijenjanje kontrasta i svjetline slike koje se anotira
    d. Svaka akcija anotiranja treba se moći poništiti i ponoviti (engl. „undo/redo“)
    e. Korisnik mora moći jednostavno obrisati dodane anotacije - pojedinačno i grupno
    f. Napredak u anotiranju slike treba se automatski spremati svake dvije minute kako bi korisnik mogao neometano raditi.
    g. Korisniku se treba omogućiti niz prečaca na mišu i tipkovnici za brži rad:
        i. Spremanje napretka (CTRL-S)
        ii. Odznačavanje trenutno označene anotacije (ESC ili desni klik)
        iii. Brisanje anotacija (dvostruki srednji klik)
        iv. Undo/redo (CTRL-Z, CRTL-Y)
12. Pružiti korisnicima informacije o procesu anotiranja i svim funkcionalnostima web sustava unutar aplikacije
13. Omogućiti izvoz podataka za anotirane slike

4.2 Kreiranje Ruby on Rails aplikacije

Nakon uspješne instalacije Ruby on Rails gema korisniku je dostupno već spomenuto CLI sučelje za upravljanje i razvoj Ruby on Rails aplikacije. Pokretanjem naredbe prikazanom u isječku koda 4.1 unutar naredbenog retka kreira novu aplikaciju.

```
rails new Cyto --database=postgresql --api
```

Isječak koda 4.1 – naredba za kreiranje nove Ruby on Rails aplikacije

Prvi parametar *new*metode rails generatora je putanja do željene lokacije nove aplikacije. U ovom slučaju predan je samo naziv projekta „Cyto“ te je generator napravio novi direktorij s tim imenom u trenutnom direktoriju. Parametar *--database=postgresql*govori generatoru da je željena

baza podataka PostgreSQL. Generator automatski postavlja u Gemfileodgovarajući adapter za povezivanje PostgreSQL-a i ActiveRecord ORM-a.

Parametar *--api* govori generatoru da će Rails aplikacija raditi u API modu, tj. da se neće koristiti ActiveView posrednik (engl. „middleware“) za generiranje HTML odgovora na zahtjeve. Ukoliko je aplikacija postavljena u API modus rada, automatski se generira konfiguracijska datoteka za CORS zaglavlje.To zaglavlje potrebno je zato što se poslužiteljska i klijentska aplikacija neće nalaziti na istoj domeni. Isječak koda 4.2 prikazuje CORS datoteku koja pomoću CORS HTTP zaglavlja dozvoljava poslužiteljskojReact aplikaciji da pristupa resursima i putanjama Rails poslužiteljske aplikacije. Prikazani isječak koda dozvoljava pristup resursima samo ako zahtjev iz pretraživača dolazi sa adrese *cyto.website* - domene poslužiteljske aplikacije ovog web sustava.

```
Rails.application.config.middleware.insert_before 0, Rack::Cors do
  allow do
    if Rails.env.production?
      origins 'https://cyto.website', 'https://www.cyto.website'
    else
      origins '*'
    end
    resource '*',
             headers: :any,
             methods: [:get, :post, :put, :patch, :delete, :options, :head],
             expose: ['Authorization']
  end
end
```

Isječak koda 4.2 - CORS.rb datoteka

Generator nakon kreiranja aplikacije inicijalizira prazni git repozitorij te postavlja konfiguracijske vrijednosti za razvojne okoline i konekcije s bazom podataka na početne vrijednosti. Konfiguracija povezivosti sa bazom podataka obavlja se kroz datoteku *database.yml* koja se nalazi u *config* direktoriju. Konfigurirana *database.yml*datoteka prikazana je u isječku koda 4.2. Zadana su lokalna imena baze podataka i detalji lokalnog PostgreSQL korisnika koji će se koristiti u development i test razvojnim okruženjima. Detalji konekcije za produkcijski način rada nisu specificirani, već se oni učitavaju iz ENV varijabli prilikom implementacije (engl. „deployment“) na produkcijskom serveru.

```
default: &default
  adapter: postgresql
  encoding: utf8
  host: localhost
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>

development:
  <<: *default
  database: cyto_development
  username: myapp
  password: ----------------
test:
  <<: *default
  database: cyto_test
  username: myapp
  password: ----------------

production:
  adapter: postgresql
  pool: 5
  encoding: utf8
  host: <%= ENV["PROD_DATABASE_HOST"] %>
  database: cyto_production
  username: <%= ENV["PROD_DATABASE_USERNAME"] %>
  password: <%= ENV["PROD_DATABASE_PASSWORD"] %>
```

Isječak koda 4.3 - konfiguracija povezivanja sa PostgreSQL bazom podataka

Nakon konfiguracije baze podataka istu je potrebno i kreirati na lokalnom, razvojnom računalu. Pri tome nam pomaže ugrađeni *rake* zadatak*rake db:create* koji se pokreće iz naredbenog retka.

Nakon što je aplikacija generirana i baza podataka postavljena poslužitelj možemo pokrenuti sa *railsserve* naredbom. Tako pokrenuta aplikacija dostupna je na lokalnom poslužitelju na portu 3000.

4.3 Kreiranje React aplikacije

Za kreiranje React aplikacije odabran je alat Create React App pokrenut preko Yarn upravitelja paketima. Za korištenje Yarn upravitelja paketima i Create React App alata potrebno je instalirati Node.js JavaScript serversko okruženje.

```
yarn create react-app Cyto --template typescript
```

Isječak koda 4.4 - naredba za kreiranje nove React aplikacije

Na isječku koda 4.3 prikazana je naredba za kreiranje React aplikacije naziva Cyto uz korištenje predefiniranih alata za upotrebu TypeScript jezika.

Create React App alat omogućuje kreiranje React projekta bez kreiranja složene konfiguracije popratnih alata za razvoj JavaScript aplikacija kao što su primjerice Babel za transpajliranje TypeScripta u JavaScript, Webpack za kreiranje modularnog koda i ESLint za statičku analizu koda.

4.4 Modeliranje Docker kontejnera

4.4.1 Backend

Isječak koda 3.1 u poglavlju 3.1 prkazuje Dockerfile datoteku korištenu za kreiranje Docker slike za Ruby on Rails aplikacije. Pomoću naredbe *FROM ruby:3.2.1*odabrana je početna konfiguracija kontejnera - u ovom slučaju Ubuntu operacijski sustav sa predinstaliranom Ruby verzijom 3.2.1.

Naredbom *RUN* pokrećemo alate i aplikacije dostupne u naredbenom retku kako bi se kreiralo okruženje koje ima sve potrebne biblioteke i aplikacije za ispravan radkontajnerizirane aplikacije. Definirane *RUN* naredbe u već spomenutoj Dockerfile datoteci instaliraju razne biblioteke za povezivanje sa bazom podataka, obrađivanje slika i sl.

Nakon što je operacijski sustav podešen potrebno je postaviti Ruby on Rails aplikaciju u produkcijski modus rada pomoću *ENV* naredbe te instalirati vanjske Ruby biblioteke o kojima aplikacija ovisi pomoću *Bundler* upravitelja paketima.

Nakon što su te naredbe izvršene okruženje je postavljeno i aplikacija je spremna za rad. Potrebno je specificirati radni direktorij aplikacije sa naredbom *WORKDIR*, definirati port na koji želimo omogućiti spajanje iz vanjskih izvora pomoću naredbe *EXPOSE.*

Za kraj je potrebno definirati i izvršnu naredbu koja će se pokretati svaki puta kada se pokrene Docker kontejner pomoću koje se pokreće kontajnerizirana aplikacija. Za to se koristi naredba CMD koja je u slučaju ove aplikacije definirana kao *CMD ["bundle", "exec", "rails", "server", "-b", "0.0.0.0"]* što nam govori da se pri pokretanju Docker kontejnera pokreće Ruby on Rails aplikacija u produkcijskom načinu rada te da se ona veže na sva dostupna mrežna sučelja kako bi bila dostupna svim uređajima i aplikacijama na istoj mreži.

4.4.2 Frontend

Iako se React aplikacije tipično pokreću samo u okruženju korisničkih Internet pretraživača te kao takve ne zahtijevaju dedicirane *backend* poslužitelje, korištenje poslužitelja donosi nekoliko prednosti. Uz kombinaciju sa Docker kontejnerima, posluživanje React aplikacije pomoću nekog od brojnih poslužitelja (u ovom slučaju odabran je Node.js) omogućuje korištenje ENV varijabli u React kodu, lako ugrađivanje aplikacije u CI/CD cjevovode (engl. „pipelines“) te lako korištenje optimizacija u produkcijskom modu rada koje pruža Yarn biblioteka. Kreirana Dockerfile datoteka dana je isječkom koda 4.4

```
FROM node:18
WORKDIR /app
COPY package*.json yarn.lock ./
RUN yarn install --production --silent
COPY . .
RUN yarn build
RUN yarn global add serve
CMD ["serve", "-s", "build", "-p", "80"]
```

Isječak koda 4.5 - Dockerfile datoteka za React aplikaciju u produkcijskom modu rada

4.5 Autentifikacija i autorizacija

Za autentifikaciju korisnika koristi se JWT i Devise Ruby on Rails proširenje sa JWT dodatkom. JSON Web Token (JWT) je kompaktan, URL kompatibilan način predstavljanja potraživanja koja

se prenose između dvije strane tijekom komuniciranja preko Interneta[17]. JWT predstavlja moderno i skalabilno rješenje za autentifikaciju i autorizaciju jer omogućuje autentifikaciju bez stanja (engl. „stateless“), što znači da nije potrebno pohranjivati stanje korisničke sesije na poslužitelju povećavajući tako skalabilnost i performanse cijelog sustava. Za razliku od kolačića (engl. „Cookies“) programer ima eksplicitnu kontrolu nad procesom slanja JWT sigurnosnih (engl. „Bearer“) tokena serveru, eliminirajući tako veliki broj kibernetičkih napada. JW tokeni također nisu ograničeni na komunikaciju između korisničkih klijenata i poslužitelja, već se često koriste i za komunikaciju između poslužiteljskih servisa. Svaki JW token sastoji se od tri dijela:

- Zaglavlje
- Korisni sadržaj tokena (engl. „payload“)
- Potpis

Zaglavlje i potpis su podaci tehničke prirode, daju informacije o vrsti enkripcije (ukoliko ona postoji), vrsti tokena i digitalnom potpisu nad sadržajem tokena. Payload tokena sadrži sva potraživanja, najčešće *iss*(izdavač tokena), *exp* (vrijeme trajanja tokena) i *sub* (subjekt potraživanja)

4.5.1 Autentifikacija

Uz pomoć Devise Ruby on Rails proširenja otvorenog koda modelu korisnika (detaljnije opisanog u slijedećem potpoglavlju) dodajemo set ekstenzija koje omogućuju prijavu i autentifikaciju korisnika.

```
class User < ApplicationRecord
  devise :database_authenticatable, :jwt_authenticatable,
         :recoverable, :rememberable, :validatable,
         jwt_revocation_strategy: JwtDenylist
end
```

Isječak koda 4.6 - isječak klase User sa Devise ekstenzijama

U isječku koda 4.5 prikazana je klasa User sa nekoliko Devise ekstenzija, od kojih su najvažnije:

- *database_authenticatable*- spremanje korisničke lozinke u bazu podataka, koristeći BCrypt algoritam za širfiranje, kroz 12 rundi
- *jwt_authenticatable*- omogućuje prijavu sa JW tokenima

Iako je jedna od glavnih prednosti JWT-a komunikacija bez pamćenja stanja, ukoliko želimo na siguran način korisnicima pružiti funkcionalnost odjave, moramo na neki način implementirati zabranu prijave sa izdanim tokenom kada se korisnik odjavi prije nego što token istekne (*exp*zaglavlje u payloadu tokena).

Za ovaj web sustav koristi se strategija crne liste koja sprema u bazu podataka sve JW tokene koji nisu istekli ali se više ne smiju koristiti za prijavu. Ta strategija je implementirana na isječku koda 4.5 kao *JwtDenylist* i dodana kao *jwt_revocation_strategy* korisničkom modelu.

4.5.2 Autorizacija

Pošto su svi resursi i stranice ovog web sustava dostupne samo prijavljenim korisnicima, sve stranice i funkcionalnosti nalaze se u korisničkom imenskom prostoru (engl. „namespace“). U isječku koda 4.6 prikazan je osnovni kontroler korisničkog imenskog prostora kojega nasljeđuju svi drugi kontroleri koji se nalaze u istom imenskom prostoru. Taj kontroler definira *before_action*funkciju povratnog poziva koja pri svakom pokušaju pristupa nekog resursa u korisničkom imenskom prostoru poziva metodu *authenticate_user!* definiranu u Devise proširenju. Ta će metoda provjeriti ispravnost JW tokena i u slučaju da je on ispravan dopustiti pristup resursu, a u suprotnom će pristup biti zabranjen i korisnik obaviješten o tome da resurs zahtjeva prijavu.

```ruby
class Users::BaseController < ApplicationController
  before_action :authenticate_user!

  protected

  def authenticate_admin!
    return if current_user.is_admin?

    render json: { message: 'You are not authorized to perform this action.' }, status: 401
  end
end
```

Isječak koda 4.7 - Osnovni kontroler korisničkog imenskog prostora

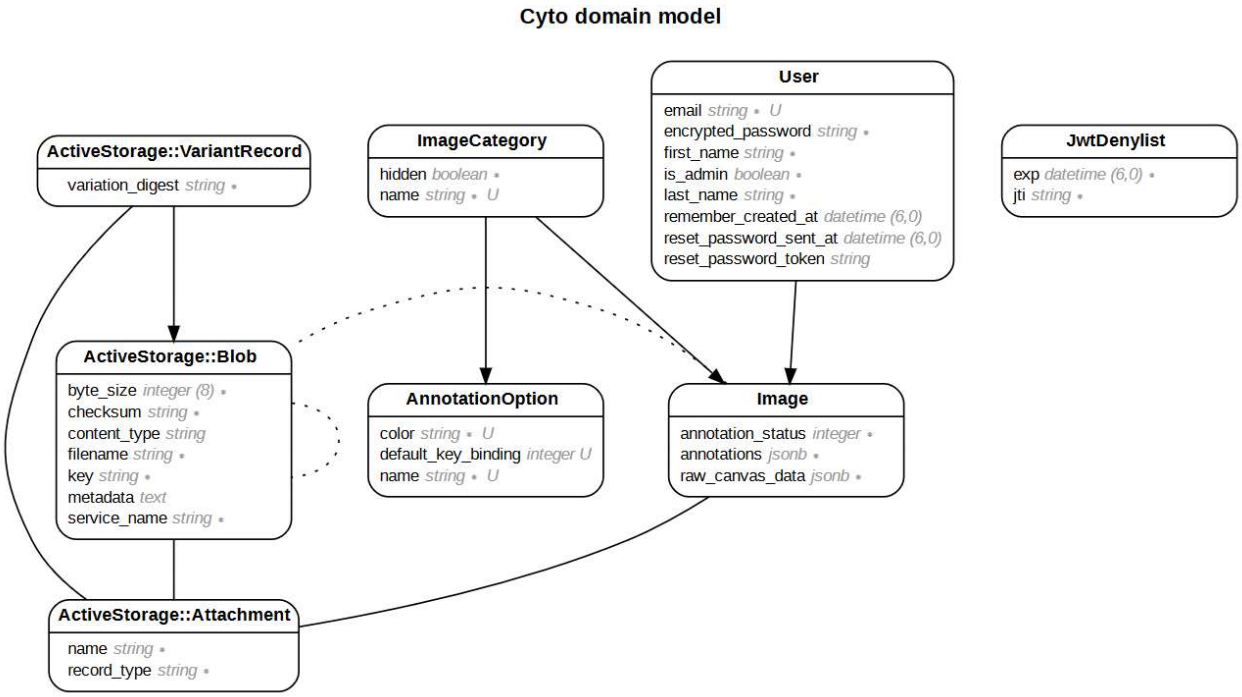
Za potrebe ovog web sustava potrebne su samo dvije uloge: administrator i regularni korisnik.

Zbog malog broja uloga i malog broja resursa koji su dostupni samo administratorima ovaj sustav vrši autorizaciju preko *is_admin*polja Booleovog tipa koji je dodan svakom korisniku. Ukoliko regularni korisnik pokuša pristupiti resursu koji je dostupan samo administratorima biti će preusmjeren na početnu stranicu uz prikladnu poruku.Stranice i resursi koji su dostupni samo administratorima autoriziraju se pomoću metode *authenticate_admin!* koja je definirana u isječku koda 4.6 i dostupna svim kontrolerima koji su definirani u korisničkom imenskom prostoru.

Resursi koji pripadaju pojedinačnim korisnicima (slike, anotacije i sl.) iz baze podataka dohvaćaju se sa upitom koji uvijek nadoda jedinstveni identifikator korisnika (polje *id*) koji je zatražio resurs te ukoliko traženi resurs ne pripada tome korisniku on neće biti poslužen.

4.6 Kreiranje modela

U sklopu ovog potpoglavlja detaljno se opisuju Ruby on Rails modeli koji definiraju podatkovnu strukturu web sustava. Razmatra se način podjele podataka na logičke cjeline i njihova međusobna povezanost i interakcija. Konačna podjela modela i njihove međusobne relacije iskazane su slikom 4.1.



Slika 4.1 - ER dijagram web sustava

4.6.1 Korisnik

Model korisnik (*User*) predstavlja svakog registriranog korisnika aplikacije. Email adresa se koristi za prijavu korisnika te uz ime i prezime jedinstveno opisuje korisnika. Svaki korisnik ima već spomenuto polje *is_admin.* Polje *email* mora biti prisutno i jedinstveno, dok se na poljima *first_name*i *last_name*vrši samo validacija prisutnosti.

Svaki korisnik može imati jednu ili više slika koje mu pripadaju, tj. slika koje je korisnik anotira - na isječku koda 4.7 definirana je *has_many*(hr. „ima mnogo“) relacija između korisnika i slika koja to omogućuje. Na toj relaciji također definiramo zavisnost pri brisanju korisnika sa atributom *dependent: :nullify*koja će na slikama od obrisanog korisnika strane ključeve postaviti na prazne vrijednosti (null).

```
class User < ApplicationRecord
  devise :database_authenticatable, :jwt_authenticatable,
         :recoverable, :rememberable, :validatable,
         jwt_revocation_strategy: JwtDenylist

  validates :email, presence: true, uniqueness: true
  validates :first_name, :last_name, presence: true

  has_many :images, dependent: :nullify
end
```

Isječak koda 4.8 - klasa User

4.6.2 Slika

Model slike (*Image)*predstavlja jednu sliku zajedno sa svojim anotacijama. Svaka slika pripada dvama korisnicima; jednom administratoru koji ju je učitao i jednom korisniku(može biti i administrator) koji ju je anotirao. Svaka slika također ima i poveznicu na jednu kategoriju kojoj pripada. Te relacije prikazane su isječkom koda 4.8. Tim relacijama dano je posebno ime (*uploaded_by*i *annotated_by)* pošto se iz *admin_id*i *user_id*teško može zaključiti da se govori od onome tko je sliku učitao ili anotirao.

```
belongs_to :uploaded_by, class_name: 'User', foreign_key: :admin_id, inverse_of: :images
belongs_to :annotated_by, class_name: 'User', foreign_key:  :user_id, inverse_of: :images,
optional: true
belongs_to :image_category
```

Isječak koda 4.9 - relacije *Image*modela sa ostalim elementima sustava

Status anotiranje svake pojedine slike prati se sa atributom *annotation_status*koji može poprimiti brojčanu vrijednost 0, 1 ili 2. Kako bi status bio razumljiviji definiran je *enum* koji dodaje ime pojedinim statusima:

```
enumannotation_status: { empty:0, in_progress:1, completed:2}
```

Sama slika spaja se na model kroz *ActiveStorage* posrednika koji pruža metodu *has_one_attached*čija je upotreba prikazana na isječku koda 4.9. Korištenjem te metode postavili smo učitanu sliku na *image* atribut modela te smo u predanom bloku koda definirali jednu dodatnu

thumb varijantu slike manjih dimenzija koju koristimo na skupnom prikazu svih slika kako se taj prikaz ne bi znatno usporio sa učitavanjem slika u originalnoj veličini. Pomoću *preprocessed: true*atributa varijanta se generira odmah pri kreiranju slike.

```
has_one_attached :image do |attachable|
  attachable.variant :thumb, resize_to_limit: [400, 400], preprocessed: true
end
```

Isječak koda 4.10 - *has_one_attached* metoda

Definirane su i dvije klasne metode, prikazane na isječku koda 4.10, koje pomažu pri odabiru slike kada korisnik započne proces anotiranja.Metoda *random_image* služi kako bi korisnik dobio novu sliku kada je započeo proces anotacije ili kada je završio već započetu sliku (opcionalni parametar metode *current_image_id*). Metoda pretražuje sve slike u bazi podataka koje su u anotacijskom statusu *empty*te od svih njih nasumično odabire jednu.

Metoda *unfinished_images*koristi se kako bi se napravila provjera postoje li već slike koje je korisnik krenuo označavati a nije dovršio. Prikazana implementacija ograničava korisnika na dvije slike koje istovremeno može anotirati.

```
  class << self
    def random_image(current_image_id = nil, image_category_id)
      images = Image
                .includes(:uploaded_by, image_category: :annotation_options, image_attachment:
:blob)
                .where(annotation_status: :empty)
                .where.not(id: current_image_id)
      images = images.where(image_category_id: image_category_id) if image_category_id
      images.sample(1).first
    end

    def unfinished_image(current_user, current_image_id = nil)
      Image
        .includes(:uploaded_by, image_category: :annotation_options, image_attachment: :blob)
        .where.not(id: current_image_id)
        .find_by(annotation_status: :in_progress, annotated_by: current_user)
    end
  end
```

Isječak koda 4.11 - klasne metode za odabir slika za proces anotiranja

Model slika sadrži dva atributa za spremanje anotacija, oba podatkovnog tipa JSONb. Prvi je *raw_canvas_data*u kojemu se spremaju sirovi, neobrađeni JSON podaci koji predstavljaju cijelo FabricJS platno.Taj se atribut koristi za vrijeme anotiranja za spremanja napretka. U slučaju da korisnik izađe iz web sustava prije nego li je završio anotiranje iz tog atributa će se rekreirati platno kako bi korisnik nastavio sa radom gdje je stao. Nakon što je korisnik anotirao i završio sliku, iz *raw_canvas_data* atributa se parsiranjem JSON-a dobiva konačno, strukturirano stanje anotirane slike. Ti se podaci onda spremaju u *annotations* atribut koji se koristi za konačni izvoz podataka.

JSONb polja odabrana su za spremanje platna i anotacija zbog nestrukturiranosti tih podataka. Oba polja mogu sadržavati veliki broj različitih objekata (oznake, kategorije, slike i sl.) te bi izvod takvih struktura u klasičnom, relacijskom načinu bio nespretan. Također, ne postoji potreba za kreiranje upita za podskup podataka iz tih polja - platno i oznake koriste se isključivo kao cjeloviti podaci. Nikada se u poslužiteljskoj aplikaciji ne vrše operacije na jednoj oznaci ili na samo

jednom dijelu platna. Izmjene tih podataka rade se također isključivo na razini cijelog polja. Korisnička aplikacija pri spremanju napretka šalje cijelo platno te se pri spremanju novog *raw_canvas_data*podatka u potpunosti prepisuju stare vrijednosti.

Integritet JSON podataka osigurava se korištenjem validacijske JSON sheme. Predefinirane sheme definiraju osnovnu strukturu tih podataka koja mora biti ispoštovana kako bi podaci bili pohranjeni. Korištenjem JSON sheme sprječava se spremanje neispravnog stanja koje može nastati zbog greške u klijentskoj aplikaciji ili zbog malicioznog korisnika. Integritet podataka validiranih JSON shemom nije potpun kao integritet standardnih tipova podataka u relacijskim bazama podataka, no za potrebe ovog web sustava JSON polja predstavljaju korektno rješenje. Isječkom koda 4.11 prikazan je proces validacije JSON polja *annotations*i *raw_canvas_data.*

```
validate :json_fields_with_schema

def json_fields_with_schema
  require "json-schema"

  if (annotations != {})
    schema_file = File.open("app/assets/json_schemas/annotations_schema.json")
    schema = JSON.load(schema_file)
    schema_file.close
    if(!JSON::Validator.validate(schema, annotations))
      errors.add(:annotations, "Annotations not valid!")
    end
  end

  if (raw_canvas_data != {})
    schema_file = File.open("app/assets/json_schemas/raw_canvas_data_schema.json")
    schema = JSON.load(schema_file)
    schema_file.close
    if(!JSON::Validator.validate(schema, raw_canvas_data))
      errors.add(:raw_canvas_data, "Canvas data is not valid!")
    end
  end
end
```

Isječak koda 4.12 -validacija JSON podataka

4.6.3 Kategorija slike

Kategorija slike (*ImageCategory*) omogućuje administratorima da definiraju kategorije kojima učitana slika pripada. Svaka kategorija može imati više slika i više anotacijskih tipova - na isječku koda 4.11 vidljivo je da je zavisnost pri brisanju kategorije slika postavljena tako da se povezane anotacijski tipovii slike kaskadno također uklanjaju.

Kako je brisanje kategorija destruktivno i povlači sa sobom brisanje mnoštva drugih objekata, brisanje kategorija, iako administratorima predočeno kao takvo, je zapravo proces skrivanja kategorija. Tada podaci postaju nedostupni u klijentskoj aplikaciji kao da su obrisani, ali ostaju postojani u bazi podataka ako budu ikad opet potrebni. Skrivanje se obavlja kroz metodu *hide*i *default_scope*(hr. „zadani opesg“) prikazano na isječku koda 4.11

```
class ImageCategory < ApplicationRecord
  has_many :annotation_options, dependent: :destroy
  has_many :images, dependent: :destroy
  accepts_nested_attributes_for :annotation_options, allow_destroy: true

  validates :name, presence: true, uniqueness:
            { scope: :hidden }, length: { maximum: 50 }

  default_scope -> { where(hidden: false).order(created_at: :desc) }

  def hide
    update(hidden: true, name: "#{name}-#{SecureRandom.hex(6)} (deleted)")
  end
end
```

Isječak koda 4.13 -klasa ImageCategory

4.6.4 Tip anotacija

Tip anotacije (*AnnotationOption*) predstavlja jednu od mogućih semantičkih vrsta anotacija koju korisnik može napraviti na slici, odnosno ovim modelom administratori definiraju tražene elemente koje žele da budu anotirani na slikama.

Svaki tip anotacije pripada jednoj kategoriji slika. Na svakom tipu anotacije definiraju se ime anotacije, boja anotacije te vrste, i početno zadana tipka miša s kojom će se postavljati anotacije te vrste. Zadane su validacije jedinstvenosti za ime i boju na razini kategorije slike te maksimalna dužina od 50 znakova za naziv tipa anotacije (prikazano na isječku koda 4.12)

```
class AnnotationOption < ApplicationRecord
  enum key_binding: { left_mouse_click: 0, double_left_mouse_click: 1,
                      right_mouse_click: 2 }

  belongs_to :image_category

  validates :name, presence: true, length: { maximum: 50 },
              uniqueness: { scope: :image_category_id }
  validates :color, presence: true, uniqueness: { scope: :image_category_id }
end
```

Isječak koda 4.14 - klasa AnnotationOption

4.7 Upravljanje rutama

Rute se u Ruby on Rails aplikacijama definiraju u *routes.rb*datoteci koja se nalazi u mapi *config*. Ta datoteka predstavlja ulazni vektor u aplikaciju, naime preko sadržaja te datoteke se korisnički zahtjevi usmjeravaju prema definiranim kontrolerima koji na te zahtjeve tada odgovaraju. Sadržaj *routes.rb*datoteke za ovaj web sustav prikazan je isječkom koda 4.13.

Root(početna) ruta postavljena je na kontroler za stvaranje nove korisničke sesije pošto je to jedini dio web sustava koji je dostupan neprijavljenim korisnicima. Pomoćna metoda *devise_for* koristi se za generiranje svih potrebnih ruta za autentifikaciju korisnika.

Definiran je jedan imenski prostor, *users*, u kojemu se nalazi sva funkcionalnost aplikacije. Zadan je parametar *defaults*sa formatom JSON što postavlja zadanu vrstu zahtjeva i tip odgovora na JSON. Na taj način omogućen je pristup resursima iz korisničke aplikacije bez specificiranje vrste odgovora, primjerice u React aplikaciji možemo tražiti resurs *images* sa URL-om*api.cyto.hr/users/images*umjesto *api.cyto.hr/users/images.json.*

```
devise_for :users,
           controllers: { sessions: 'users/sessions' }
root 'devise/sessions#new'
get 'validate_token/:reset_token', to: "users#validate_token", defaults: { format: :json}
patch 'set_password/', to: "users#set_password", defaults: { format: :json}

namespace :users, defaults: { format: :json } do
  resources :users, except: [:new, :edit]
  resources :image_categories, except: [:new, :edit]
  resources :annotation_options, except: [:new, :edit]
  resources :images, except: [:new, :edit]

  get 'images/next_image', to: 'images#next_image'
  patch 'images/:id/save_progress', to: 'images#save_progress'
  patch 'images/:id/complete_annotation', to: 'images#complete_annotation'
end
```

Isječak koda 4.15 - sadržaj routes.rb datoteke

4.8 Upravljanje rutama na klijentskoj strani

Iako su aplikacije kreirane sa React bibliotekom tipično SPA aplikacije, zbog boljeg korisničkog iskustva za potrebe ovog web sustava koristi se React DOM Router koji dodaje mogućnost korištenja navigacijskih tipki u korisničkim pretraživačima interneta i definiranje trajnih URL-ova na resurse u web sustavu.

Rute su definirane u datoteci *main.tsx*koja se nalazi u mapi *routes*. U isječku koda 4.14 prikazana je funkcija *router*koja definira i vraća ruter klijentske strane. Definiran je početni element *PrivateRoute* na početnoj putanji „/“ čija je implementacija prikazana u isječku koda 4.15. *PrivateRoute*komponenta provjerava status korisničke prijave i ukoliko je ona ispravna prikazuje *App* komponentu koja je polazna točka ovog web sustava. U suprotnom, ukoliko korisnik nije prijavljen ili je JW token neispravan, korisnik se preusmjerava na stranicu za prijavu.

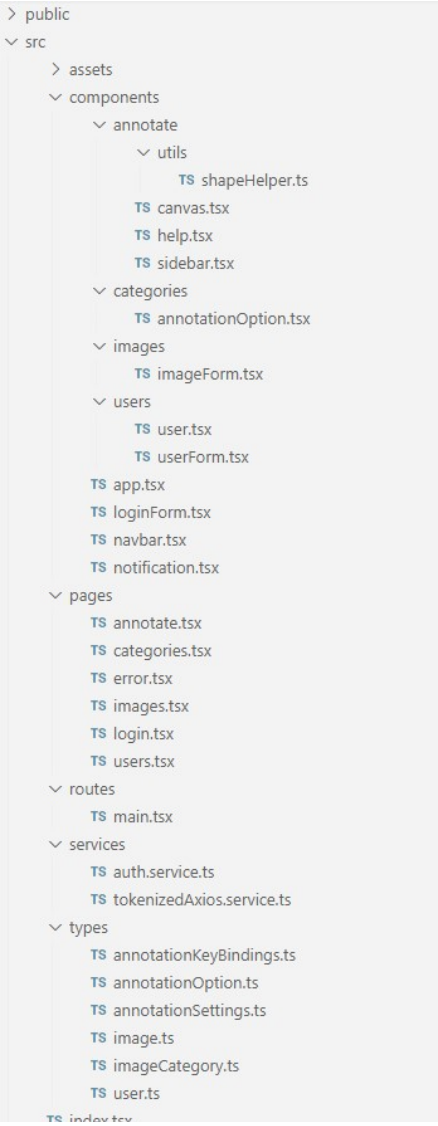
Ostale komponente ugniježđene su unutar početne *App* komponente te su dostupne samo prijavljenim korisnicima. Nakon uspješne autentifikacije korisnici se preusmjeravaju na *Annotate*komponentu koja sadrži sustav za anotiranje slika.

```
export default function router() {
  return createBrowserRouter([
    {
      path: "/",
      element: <PrivateRoute element={<App />} />,
      errorElement: <ErrorPage />,
      children: [
        {
          path: "/",
          element: <Annotate />,
        },
        {
          path: "/users",
          element: <Users />,
        },
        {
          path: "/images",
          element: <Images />,
        },
        {
          path: "/images/:id/edit",
          element: <ImageForm />,
        },
        {
          path: "/categories",
          element: <Categories />,
        }
      ],
    },
    {
      path: "/login",
      element: <LoginPage />,
      errorElement: <ErrorPage />,
    }
  ])
};
```

Isječak koda 4.16 - funkcija za kreiranje ruta na klijentskoj strani

```
function PrivateRoute({ element, ...rest }: any) {
  const isAuthenticated = userSignedIn();
  return isAuthenticated ? (
    element
  ) : (
    <Navigate to="/login" replace />
  );
}
```

Isječak koda 4.17 - TSX komponenta PrivateRoute



Slika 4.2 - struktura klijentskog dijela projekta

Klijentski dio web sustava raspoređen je u stranice i njihove pripadajuće komponente. Svaka stranica ima svoju pripadajuću rutu te jednu ili više pripadajućih komponenti. Uz stranice i njihove komponente dodatno su prisutni:

- Mapa *assets*- slike, fontovi, CSS datoteke
- Mapa*services* -funkcije za autorizaciju/autentifikaciju
- Mapa *types* - TypeScript podatkovni tipovi

Raspored, broj i vrsta datoteka u klijentskom dijelu web sustava prikazan je slikom 4.2

4.9 Pristup autoriziranim resursima

Nakon uspješne prijave na klijentskom se uređaju JW token sprema u lokalnu pohranu (engl. „localstorage“) te se on kasnije koristi za pristup svim elementima web sustava. JWT se mora pridružiti svakom zahtjevu za autoriziranim resursom, a kako bi to bilo nepraktično izvoditi u svakom dijelu koda koji šalji neki zahtjev poslužiteljskoj aplikaciji definirana je pomoćna uslužna funkcija *tokenizedAxios.service,* prikazana u isječku koda 4.16.

```
import axios from 'axios';
import { getToken } from './auth.service';


const instance = axios.create({
    baseURL: process.env.REACT_APP_API_URL + "users/",
    headers: {
        Accept: "application/json",
        "Content-Type": "application/json",
    },
});

instance.interceptors.request.use((config) => {
    const token: string | null = getToken();
    if (token) {
        config.headers.Authorization = token;
    } else {
        localStorage.removeItem("user");
        localStorage.removeItem("jwt");
        window.location.href = "/login";
    }
    return config;
});

export default instance;
```

Isječak koda 4.18 - tokenizedAxios pomoćna funkcija

Prikazana funkcija generira instancu *axios* HTTP klijenta koja usmjerava početni URL svakog zahtjeva na URL Ruby on Rails poslužitelja koji se definira pomoću ENV varijabli. Također se postavlja i zadana vrijednost tražene vrste odgovora kao „application/json“ što znači da klijentska aplikacija očekuje odgovor u obliku JSON za svaki zahtjev.

Na generiranu instancu *axios*klijenta postavljen je presretač (engl. „interceptor“) koji prije svakog zahtjeva prema poslužitelju dohvaća JW token te ga postavlja na autorizacijsko zaglavlje. U slučaju da je JW token nepostojeći ili neispravan korisnik se preusmjerava na stranicu za prijavu.

Nakon što se ta tokenizirana verzija *axios* instance učita u komponentu pomoću import naredbe, može se koristiti s istim sučeljem kao i obična *axios*instanca. Primjer korištenja može se vidjeti u isječku koda 4.17 u kojemu se šalje zahtjev za popisom kategorija slika na autorizirani URL /users/image_categories/ kojemu je automatski pridodan JW token u *Authorization*zaglavlju.

```
tokenizedAxios.get("/image_categories").then((response) => {
    if (response.data.length === 0) {
            notify("You have to create at least one image category before you can add images",
        "info");
            navigate("/categories", { replace: true });
            return;
    }
    setCategories(response.data);
    setSelectedCategory(response.data[0].id);
});
```

Isječak koda 4.19 - primjer korištenja tokeniziraneaxios instance

4.10 Upravljanje korisnicima

Kako je po specifikaciji zahtjeva onemogućena samostalna registracija korisnika, administratorima je omogućeno dodavanje novih korisnika kroz sučelje aplikacije. Nakon što je administrator uspješno dodao novog korisnika poslužiteljska Ruby on Rails aplikacija šalje mail sa poveznicom i uputstvima kako postaviti lozinku kako bi korisnik mogao započeti korištenje web sustava. Na isječku koda 4.18 prikazana je *create*metoda kontrolera *Users*koja se poziva pri kreiranju novog korisnika.

```
def create
  @user = User.new(user_params)
  @user.password = SecureRandom.base64
  raw_token, enc_token = Devise.token_generator.generate(User, :reset_password_token)
  @user.reset_password_token = enc_token
  @user.reset_password_sent_at = Time.now
  if @user.save
    render json: @user, status: 200
    UserCredentialsMailer.with(admin: current_user, user: @user, token:
enc_token).update_password.deliver_later
  else
    render json: { message: @user.errors.full_messages }, status: 422
  end
end
```

Isječak koda 4.20 - create metoda Users kontrolera

Pri kreiranju novog korisnika generira se privremena, nasumična lozinka te se odmah generira i token koji se koristi u slučaju kada korisnik zaboravi lozinku.U ovom slučaju token služi da se korisniku omogući prvotno postavljanje lozinke. Ukoliko su podaci o novom korisniku ispravni te se on uspješno spremi u bazu podataka novokreiranom korisniku šalje se email sa uputama o postavljanju nove lozinke kako bi mogao započeti korištenje sustava.

4.11 Označavanje slika

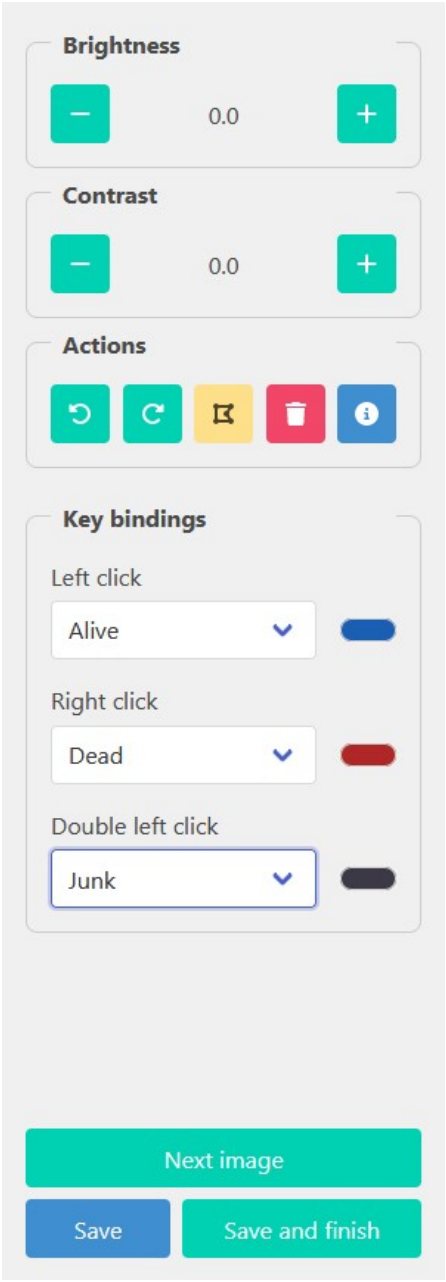
Proces označavanja slika korisnik započinje na početnoj stranici aplikacije. Kada korisnik započne proces označavanja poslužiteljska aplikacija zaprimi zahtjev na *next_image*metodu prikazanu isječkom koda 4.21 koja prvo provjerava postoji li slika na kojoj je korisnik već započeo označavanje, a nije završio. Metoda *unfinished_image,*uz trenutnog korisnika, kao parametar prima i ID trenutno aktivne slike (korisnik trenutno već označava neku sliku) te se na taj način korisnika ograničava na dvije aktivne slike. Ukoliko metoda *unfinished_images*nije pronašla niti jednu aktivnu sliku kod trenutnog korisnika varijabla *@image* postavlja se na nasumično odabranu sliku.

U slučaju da je odgovarajuća slika pronađena i postavljena u varijablu *@image*, odabrana slika vraća se kao odgovor na zahtjev u korisničku aplikaciju i proces označavanja započinje. U slučaju da je varijabla *@image* prazna to znači da ne postoji slika koja je u statusu *empty* te se korisniku prikazuje poruka da trenutno nema više slika za označiti.

```
def next_image
  @image = Image.unfinished_image(current_user, params[:current_image_id])
  @image = Image.random_image(params[:current_image_id]) if @image.blank?
  if @image.present?
    render 'users/images/next_image', status: 200
    @image.update(annotation_status: :in_progress, annotated_by: current_user) if
@image.annotation_status == 'empty'
  else
    render json: [], status: 200
  end
end
```

Isječak koda 4.21 - metoda next_image

Kada je slika uspješno pronađena i zaprimljena na klijentskoj aplikaciji kreira se novi *canvas*element biblioteke Fabric.js. Na isječku koda 4.22 prikazan je isječak komponente *Annotate*koja predstavlja cijelu stranicu sustava za označavanje. Sustav za označavanje sastoji se od dvije komponente, pomoćne trake sa akcijama (*Sidebar)* i platna koje sadrži *canvas*element Fabric.js biblioteke (*Canvas).* Tim komponentama pružena su tri React konteksta, *ImageContext*koji sadržava trenutno aktivnu sliku, *AnnotationSettingsContext* koji sadržava trenutne postavke anotacije i *CanvasContext*koji sadrži trenutno stanje platna za označavanje.




```
<ImageContext.Provider value={{ currentImage: currentImage, setCurrentImage: s
}}>
<AnnotationSettingsContext.Provider value={{ annotationSettings: annotationSet
setAnnotationSettings: setAnnotationSettings }}>
<CanvasContext.Provider value={{ currentCanvas: currentCanvas, setCurrentCanva
setCurrentCanvas }}>
        <div className="wrapper">
            <Sidebar />
            <main className="main">
                {currentImage && <Canvas/> }
            </main>
        </div>
</CanvasContext.Provider>
</AnnotationSettingsContext.Provider>
</ImageContext.Provider>
```

Isječak koda 4.22 - komponenta Annotate

4.11.1 Pomoćna traka

Pomoćna traka prikazana je slikom 4.4 i pomoću nje korisnik upravlja cjelokupnim procesom označavanja.

Korisniku je omogućeno mijenjanje svjetline i kontrasta slike pomoću standardnih filtera dostupnih u Fabric.js biblioteci, čija se implementacija nalazi u funkciji *applyFilters*prikazanoj isječkom koda 4.23

Od akcija, korisniku je omogućeno:

- *undo/redo*funkcionalnost
- odabir oblika (elipsa ili slobodan oblik)
- brisanje označenih anotacija
- prikaz uputa za korištenje

Tipovi anotacija prvotno su postavljeni na zadane vrijednosti koje je definirao administrator za kategoriju slike koja se označava. Korisnik je slobodan rasporediti si dostupne tipove na akcije mišem koje mu odgovaraju.

Slika 4.3 - pomoćna traka

Klikom na *Save* tipku sprema se trenutni napredak, a *Save andfinish*tipka označava kraj označavanja.

Odabirom tipke *Nextimage* korisnik učitava slijedeću sliku za označavanje, ukoliko ona postoji.

```
function applyFilters() {
  if (currentCanvas && currentImage && annotationSettings) {
    const img = currentCanvas.backgroundImage as Image;
    if (img !== null) {
      const { brightness, contrast } = annotationSettings!;
      const filters: fabric.IBaseFilter[] = [];
      filters.push(
        new fabric.Image.filters.Brightness({
          brightness: brightness.toNumber(),
        })
      );
      filters.push(
        new fabric.Image.filters.Contrast({ contrast: contrast.toNumber() })
      );
      img.applyFilters(filters);
      currentCanvas.setBackgroundImage(
        img,
        currentCanvas.renderAll.bind(currentCanvas)
      );
    }
  }
}
```

Isječak koda 4.23 - funckijaapplyFitlers

4.11.2 Element platna

Element platna (*Canvas*komponenta*)* sadrži HTML5 *canvas*element na kojemu se odvija proces označavanja. Komponenta vraća samo već spomenuti element koji se definira kao `<canvasref={canvasRef}id="annotaion-canvas"/>`te se pomoću *ref*atributa taj element povezuje na Fabric.js biblioteku. Kreiranje Fabric.js platna prikazano je isječkom koda 4.24

```
const { width, height } = currentImage.metadata;

canvas = new fabric.Canvas(canvasRef.current!, {
  width,
  height,
  fireMiddleClick: true,
  fireRightClick: true,
  stopContextMenu: true,
});
```

Isječak koda 4.24 - inicijalizacija Fabric.js platna

Dimenzije platna postavljaju na dimenzije slike koja se označava kako bi koordinate oznaka uvijek bile točne u odnosu na sliku. Kako su slike koje se označavaju gotovo uvijek znatno veće nego rezolucija korisničkog zaslona, nakon početne inicijalizacije poziva se funkcija *resizeCanvas,* prikazana isječkom koda 4.25,koja prilagođava zoom razinu kako bi se korisniku prvotno prikazala slika koja kraćom dimenzijom zauzima 100% dostupnog prostora platna.

```
const resizeCanvas = () => {
  if (canvas && currentImage) {
    const { width, height } = currentImage.metadata;
    const parentElement = canvasRef.current?.parentNode as HTMLElement;
    const parentWidth = parentElement.clientWidth;
    const parentHeight = parentElement.clientHeight;

    const scaleX = parentWidth / width;
    const scaleY = parentHeight / height;
    const zoomRatio = Math.min(scaleX, scaleY);

    let imageTextureSize = width > height ? width : height;
    if (imageTextureSize > fabric.textureSize) {
      fabric.textureSize = imageTextureSize;
      //The default texture limit of 2048 is not enough for this usecase
    }

    canvas.setZoom(zoomRatio);
    canvas.renderAll();
  }
};
```

Isječak koda 4.25 - funkcija resizeCanvas

Funkcionalnost zumiranja pomoću kotačića na mišu implementirana je pomoću funkcije povratnog poziva na događaj okretanja kotačića, prikazana isječkom koda 4.26

```
canvas?.on("mouse:wheel", function (opt) {
  var delta = opt.e.deltaY;
  var zoom = canvas!.getZoom();
  zoom *= 0.999 ** delta;
  if (zoom > 20) zoom = 20;
  if (zoom < 0.01) zoom = 0.01;
  canvas!.zoomToPoint({ x: opt.e.offsetX, y: opt.e.offsetY }, zoom);
  opt.e.preventDefault();
  opt.e.stopPropagation();
});
```

Isječak koda 4.26 - implementacija zumiranja platna

Zumiranje je ograničeno intervalom [0.01, 20], manje i veće vrijednosti nemaju smisla te bi potencijalno mogle zbuniti korisnika. Pomoću metode *zoomToPoint*prilikom zumiranja prikazani dio platna automatski se pomjera prema poziciji korisnikovog pokazivača.

Ostatak komponente definira razne funkcionalnosti, poput pomicanja platna srednjim klikom miša, postavljana raznih tipkovničkih prečaca te, najvažnije, postavljanje anotacijskih elemenata. Isječak koda 4.27 prikazuje dio funkcije povratnog poziva koja se poziva na klik miša. Na isječku je prikazan dio koda koji postavlja anotacijski element definiran za lijevi klik miša.

```
if (evt.button === 0) {
  // Check for left click (button value of 0)
  if (annotationSettings.keyBindings.leftClick !== undefined) {
    let params = {
      annotationOption: annotationSettings.keyBindings.leftClick,
      shape: annotationSettings.shape
      canvas: currentCanvas,
      opt: opt,
    };
    addShape(params);
    return;
  }
}
```

Isječak koda 4.27 - postavljanje oznaka na platno

Funkcija *addShape*dodaje novi element na platno, u ovisnosti o trenutnim anotacijskim postavkama. Ako je, primjerice, korisnik odabrao vrstu oznake elipsa tada će funkcija *addShape* pozvati funkciju *addCircle,* prikazanu isječkom koda 4.28.

Ta funkcija će uz predane parametre x,y lokacije pokazivača, definirane boje, reference na trenutno platno i jedinstvene oznake tipa anotacije kreirati novi element na platnu. Element se postavlja na trenutnu poziciju korisnikovog pokazivača. Elementu se nadodaje novi *annotation_option_id* atribut koji će pri izvozu podataka pomoći u povezivanju oznaka sa tipovima oznaka u bazi podataka.

```
function addCircle({ pointer, color, canvas, annotationOptionId }:
    { pointer: { x: number, y: number }; color: string; canvas: fabric.Canvas;
annotationOptionId: number }) {
    const circle = new fabric.Circle({
        left: pointer.x,
        top: pointer.y,
        originX: 'center',
        originY: 'center',
        radius: 10,
        fill: toRGBA(color, 0.3),
        strokeWidth: 2,
        stroke: color,
        selectable: true,
        hasControls: true,
        lockScalingX: false,
        lockScalingY: false,
        hasRotatingPoint: true,
        strokeUniform: true,
    });
    circle.set("annotation_option_id", annotationOptionId);
    canvas.add(circle);
    canvas.setActiveObject(circle);
    canvas.renderAll();
}
```

Isječak koda 4.28 - kreiranje nove eliptične oznake

4.12 Izvoz podataka

Nakon što je korisnik označio sliku kao anotiranu poslužiteljska aplikacija zaprima posljednju verziju sirovih, neobrađenih podataka iz anotirajućeg platna. Nakon toga poziva se uslužna klasa CanvasParserService, čiji je isječak prikazan isječkom koda 4.29

```ruby
def initialize(image_id, raw_canvas_data)
  super()
  @raw_canvas_data = raw_canvas_data
  @image = Image.includes(image_category: :annotation_options).find(image_id)
end

def call
  canvas_data = snakecase_keys(raw_canvas_data)
  {
    background_image: extract_background_image(canvas_data),
    circle_objects: extract_circle_objects(canvas_data),
    freeshape_objects: extract_freeshape_objects(canvas_data),
  }
end
```

Isječak koda 4.29 - dio koda CanvasParserSerivce klase

Nakon što je klasa inicijalizirana sa sirovim podacima sa anotacijskog platna i označenom slikom, pozivom na metodu *call* kreira se objekt koji predstavlja sve anotacije pripadajuće slike, zajedno sa podacima o samoj slici. Tako kreirani objekt sprema se tada u polje annotations modela *Image* te se korisniku u klijentskoj aplikaciji prikazuje poruka da je slika uspješno označena.

U svakom trenutku administrator ima mogućnost pozvati izvoz podataka svih označenih slika iz željene kategorije. Kada se pokrene izvoz podataka klijentskoj se aplikaciji šalje JSON datoteka koja sadrži sve označene slike iz zatražene kategorije. Implementacija metode za izvoz podataka prikazana je isječkom koda 4.30 a kratki isječak izvezene datoteke isječkom koda 4.31

```ruby
def export_data
  finished_images = images.where(annotation_status: :completed)
  {
    category_name: name,
    images_count: finished_images.count,
    images: finished_images.map(&:annotations),
    annotation_options: annotation_options.map do |annotation_option|
      {
        id: annotation_option.id,
        name: annotation_option.name,
        color: annotation_option.color,
      }
    end,
  }.to_json
end
```

Isječak koda 4.30 - metoda za izvoz podataka, export_data

```
{
  "category_name": "Alive-Dead",
  "images_count": 1,
  "images": [
    {
      "circle_objects": [
        {
          "top": 1809.83,
          "left": 603.3,
          "angle": 0,
          "flip_x": false,
          "flip_y": false,
          "radius": 10,
          "scale_x": 1.19,
          "scale_y": 1.19,
          "origin_x": "center",
          "origin_y": "center",
          "end_angle": 360,
          "start_angle": 0,
          "annotation_option": {
            "id": 1,
            "name": "Alive",
            "color": "#1a5fb4",
            "image_category_id": 1,
          },
          "annotation_option_id": 1
        },
       },
  ]
}
```


Isječak koda 4.31 - isječak izvezene JSON datoteke

5. PREGLED FUNKCIONALNOSTI I IZGLEDA WEB SUSTAVA

Korisnik svoje korištenje web sustava započinje kada zaprimi email poruku koja sadrži poziv za pridruženje platformi. Klikom na poveznicu, prikazanu na slici 4.3, korisnik se upućuje na stranicu na kojoj postavlja svoju lozinku, prikazanu na slici 5.1.

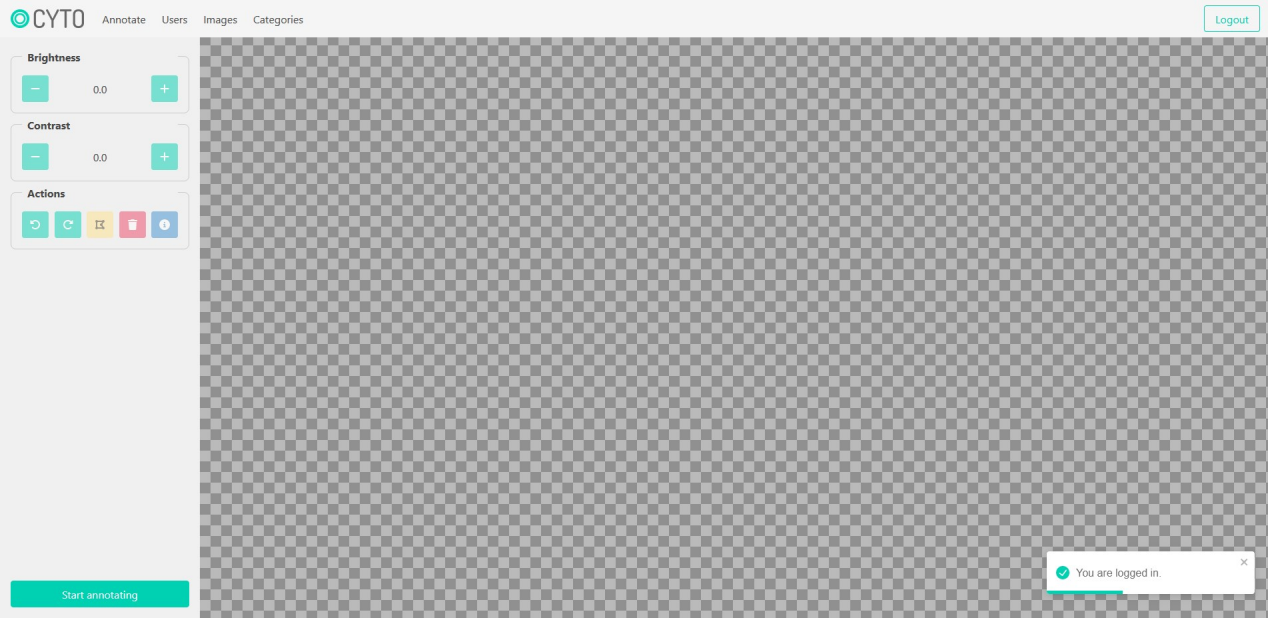
Slika 5.1 - stranica za postavljanje lozinke

Nakon što je unio te spremio ispravnu lozinku, korisnik se preusmjerava na stranicu za prijavu, uz

odgovarajuću poruku.

Slika 5.2 - stranica za prijavu

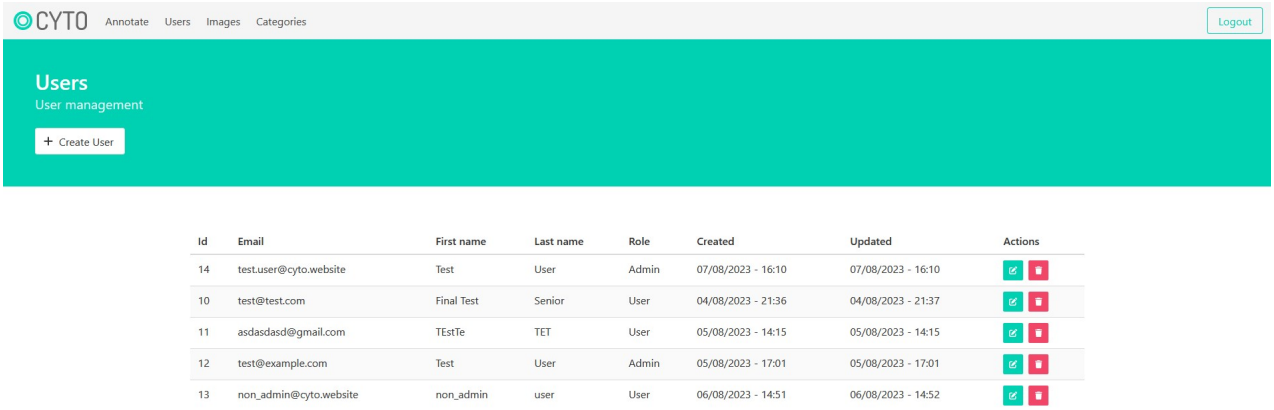
Nakon uspješne prijave korisnik se preusmjerava na početnu stranicu web sustava, stranicu za anotiranje slika, prikazanu slikom 5.3



Slika 5.3 - stranica za označavanje slika

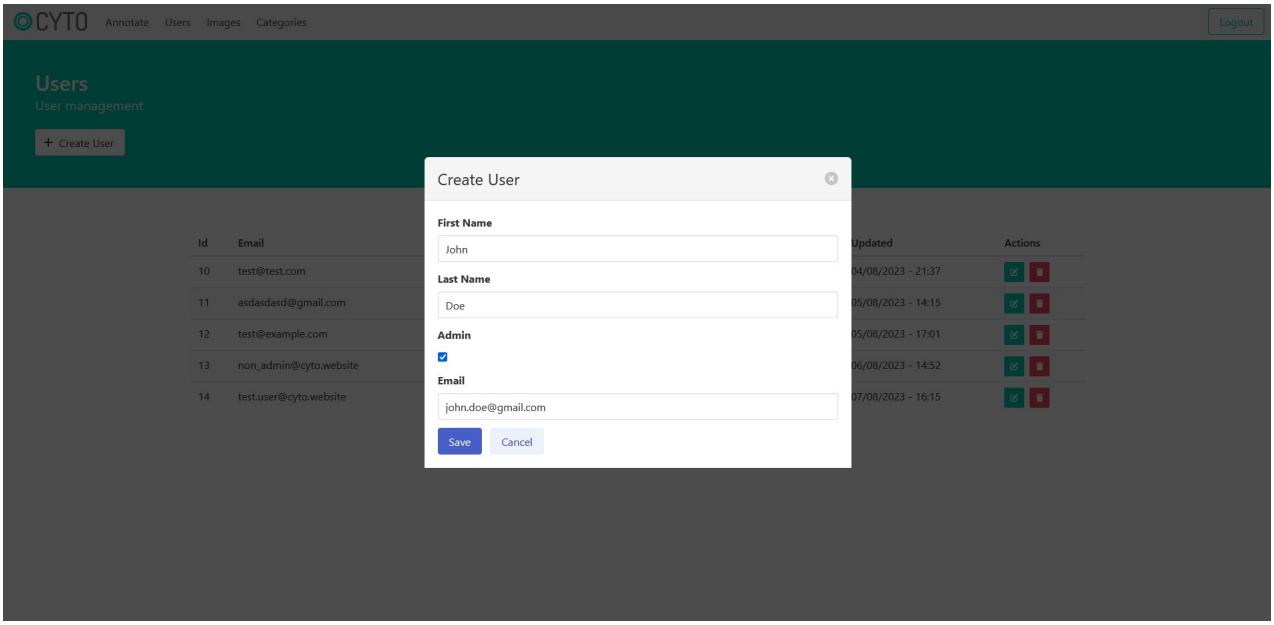
Korisnik odmah može početi sa označavanjem slika ili, kako je postavljen za administratora, koristiti ostale stranice namijenjene za upravljanje sustavom.

Stranica *Users* omogućava administratorima da upravljaju sa korisnicima. Omogućeno je brisanje,



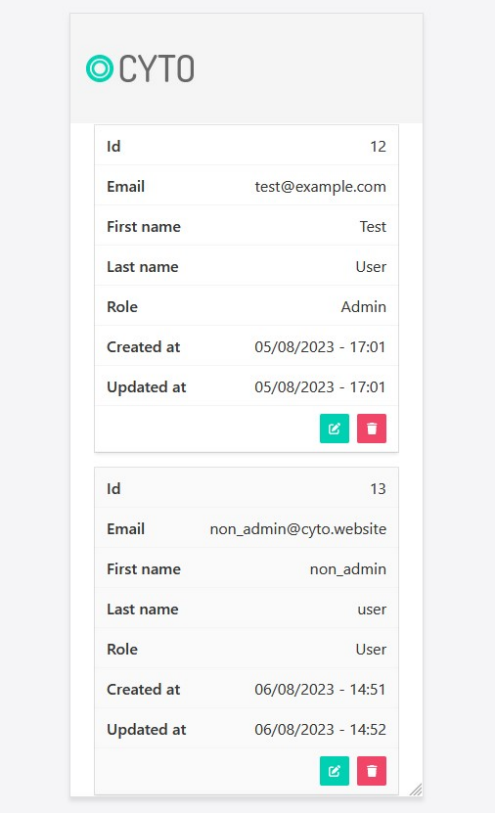
Slika 5.4 - tablični prikaz korisnika

dodavanje i uređivanje korisnika. Funkcionalnost stranice prikazana je slikama 5.4 i 5.5



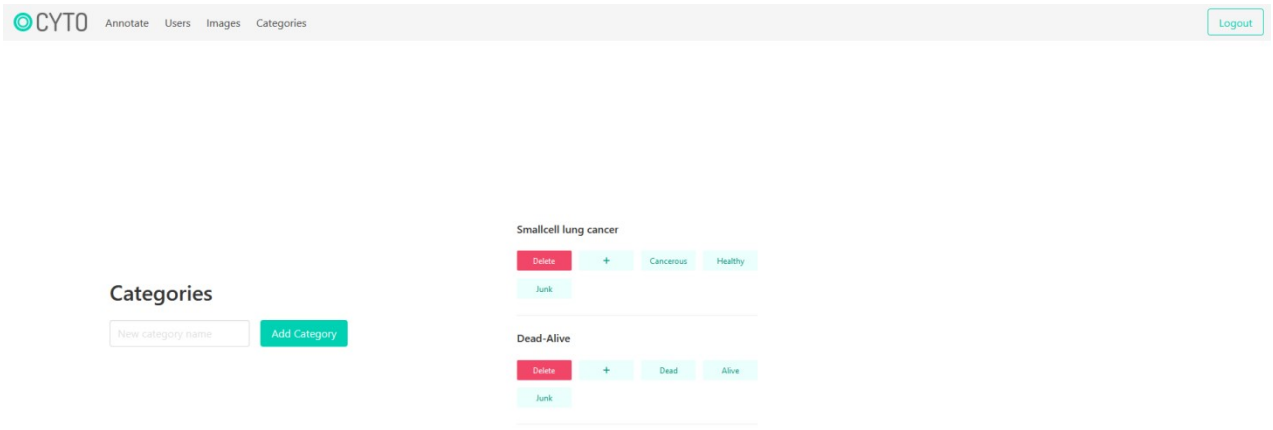
Slika 5.5 - forma za uređivanje i stvaranje korisnika

Iako je proces označavanja podržan samo na računalima, administracijske stranice prilagođene su za korištenje na mobilnim uređajima, slika 5.6 prikazuje mobilnu verziju tabličnog prikaza korisnika

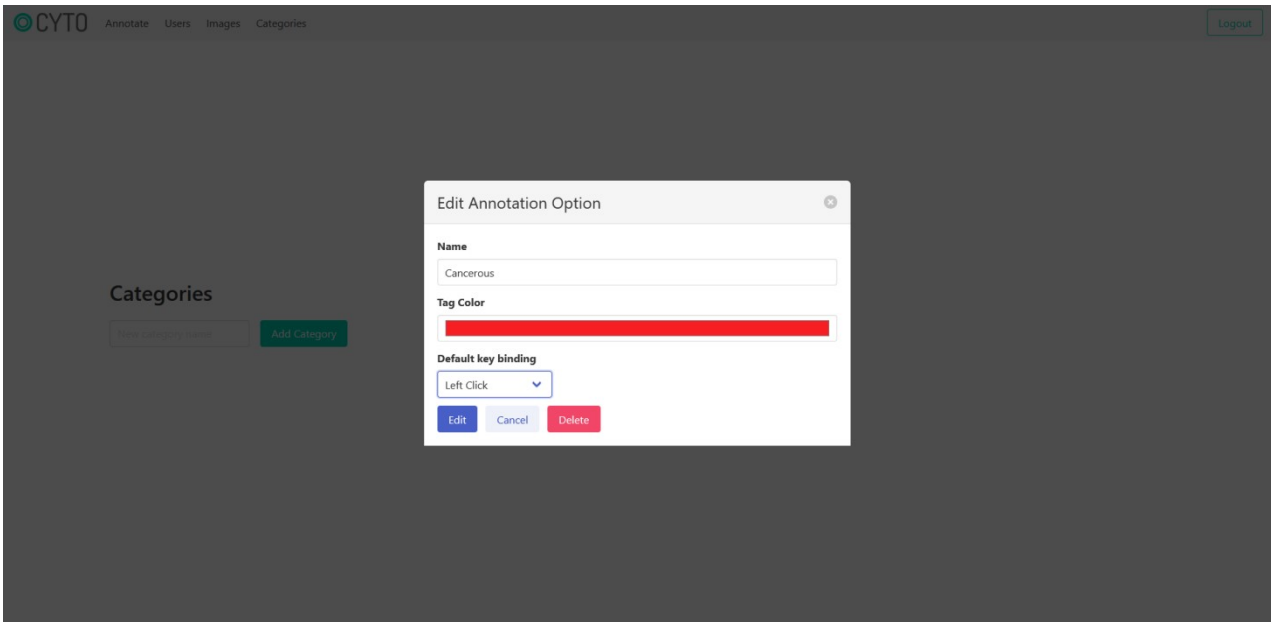


Slika 5.6 - prikaz korisnika na mobilnom uređaju

Stranica *Categories* omogućuje administratorima da definiraju kategorije po kojima će se učitavati i označavati slike te njihove prikladne tipove anotacija. Slika 5.7 prikazuje stranicu sa izlistajem svih kategorija dok slika 5.8 prikazuje formu pomoću koje se dodaju i uređuju tipovi anotacija.

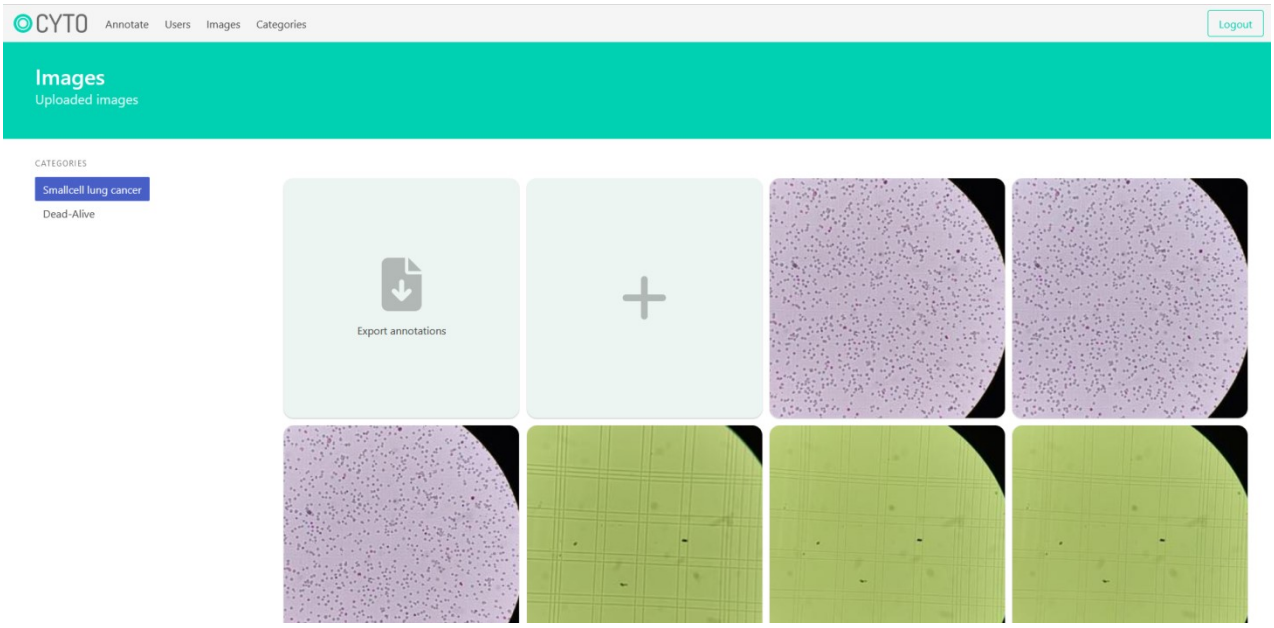


Slika 5.7 - prikaz kategorija slika

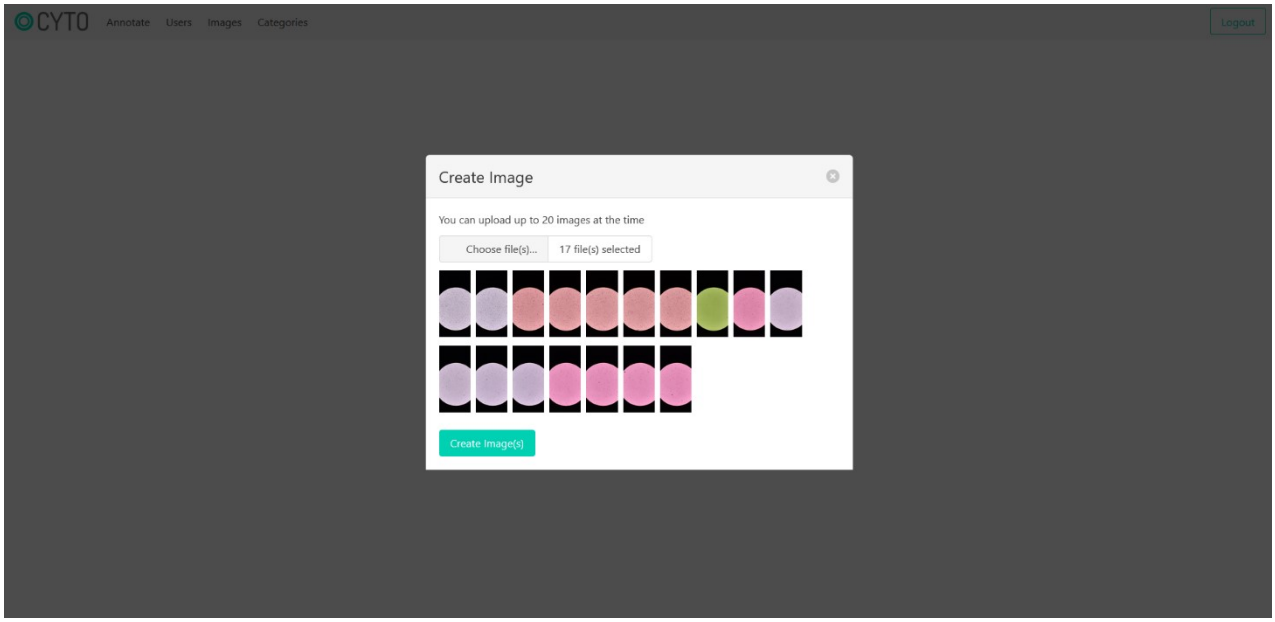


Slika 5.8 - forma za dodavanje i uređivanje tipova anotacija

Nakon što je definirana barem jedna kategorija, administrator može učitati slike. Nakon što je otvorio stranicu *Images,* prikazanu slikom 5.9,administrator odabire kategoriju iz lijeve trake te za odabranu kategoriju ima mogućnost dodati nove slike ili napraviti izvoz podataka za sve označene slike iz odabrane kategorije. Pri učitavanju slika administrator može učitati do 20 slika istovremeno. Forma za učitavanje slika prikazana je slikom 5.10.

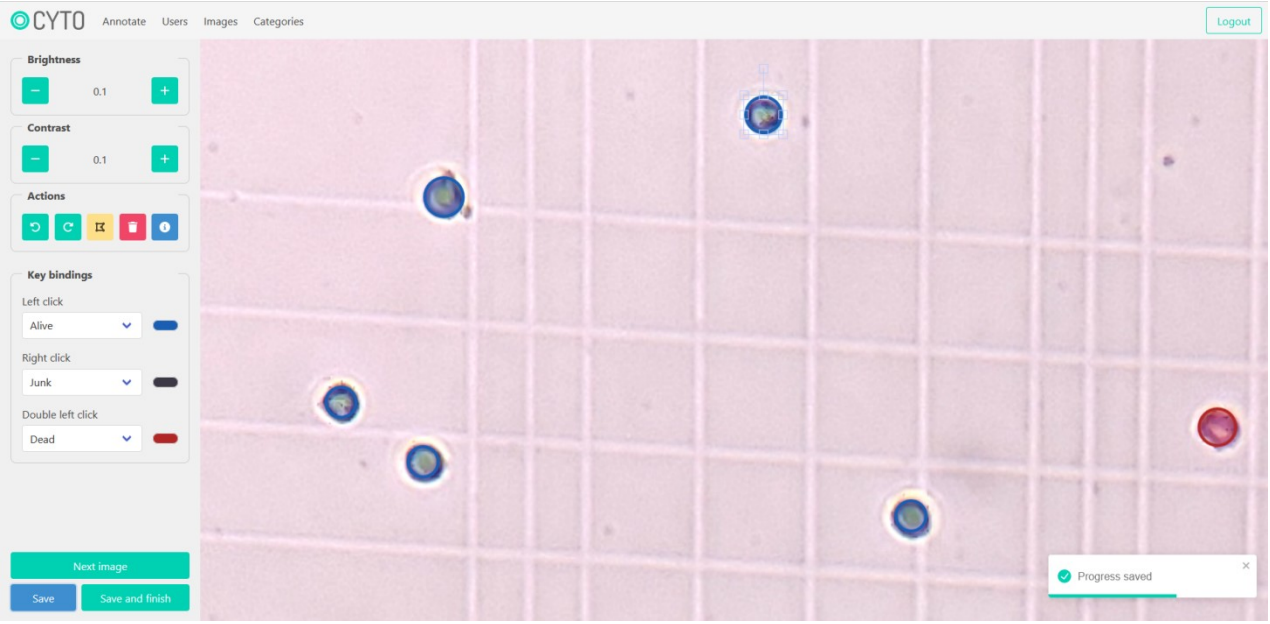


Slika 5.9 - stranica Images



Slika 5.10 - forma za učitavanje slika

Nakon što su kategorije postavljene i slike učitane, proces označavanja može početi. Slikom 5.11 prikazan je izgled *Annotate*stranice tijekom procesa označavanja.



Slika 5.11 - proces označavanja slika

6. ZAKLJUČAK

Kako je korištenje modela strojnog učenja u medicinske svrhe doživjelo streloviti rast u posljednjem desetljeću, znatan se naglasak stavlja na optimizaciju cjelokupnog procesa. Upravo proces označavanja, koji je tema ovog rada, nerijetko predstavlja vremenski najzahtjevniji dio kreiranja modela strojnog učenja. Zadatak rada bio je razviti web sustav koji će omogućiti medicinskim stručnjacima brzo, učinkovito i kvalitetno označavanje velikog broja medicinskih slika.

U uvodnim poglavljima rada ukratko je opisan zadatak te je napravljen kratki osvrt na trenutnu zastupljenost strojnog učenja u medicinskoj praksi. Prikazana su slična programska rješenja koja suveć dostupna na tržištu. Iako su sva postojeća rješenja znatno naprednija i sadrže veliki broj mogućnosti, problem nastaje zbog njihove opće namjene. Naime, već spomenuti veliki broj mogućnosti utječe na brzinu anotiranja slika. Za potrebe označavanja velikog broja slika s velikim brojem anotacija po slici,ovaj web sustav predstavlja korektno rješenje. Plaćena rješenja u većini slučajeva su nepristupačnih cijena.

U sljedećim su poglavljima obrađeneglavne tehnologije za izradu ovog rada: JavaScript biblioteka React, biblioteka Fabric.js iRuby on Rails razvojni okvir te njihova proširenja i pripadajuće tehnologije. Spomenute su također i pomoćne tehnologije poput Docker-a i PostgreSQL-a. Rad zatim obrađuje ključne dijelove web sustava te njihovu individualnu implementaciju. Poglavlje započinje specifikacijom zahtjeva te se nadalje opisuje cjelokupni razvojni proces - od kreiranja klijentske i poslužiteljske aplikacije pa sve do samog procesa označavanja i izvoza anotacijskih podataka, s jasnim naglaskom na osiguravanje brzog i učinkovitog rada pri označavanju slika.

Rezultat ovog rada je web sustav koji svojim korisnicima i administratorima pruža jednostavno sučelje i kvalitetne alate za provođenje cjelokupnog procesa označavanja medicinskih slika. Moguće nadogradnje, poput sustava koji bi omogućio korisnicima da međusobno validiraju anotacijeili sustava poluautomatskog anotiranja uz pomoć modela dubokog učenja, značajno bi nadogradile preciznost i efikasnost procesa označavanja.

LITERATURA

[1] Heang-Ping C., Samala R., Hadjiiski L., Zhou C., DeepLearninginMedicalImageAnalysis, Department ofRadiology, University of Michigan, Siječanj 2021. [Online]. Dostupno na: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7442218/pdf/nihms-1617552.pdf [Pristupljeno: 11-09-2023]

[2] Matsuo H, Kamada M, Imamura A, Shimizu M, Inagaki M, Tsuji Y, Hashimoto M, Tanaka M, Ito H, Fujii Y., Machinelearning-basedpredictionofrelapseinrheumatoidarthritispatientsusing data on ultrasoundexaminationandblood test, ScientificReports, Svibanj 2022. [Online]. Dostupno na: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9068780/pdf/41598_2022_Article_11361.pdf [Pristupljeno: 24-06-2023]

[3] 3D Slicer, https://www.slicer.org/ [Online]. [Pristupljeno: 11-09-2023]

[4] BioImageXD, https://www.bioimagexd.net/ [Online]. [Pristupljeno: 11-09-2023]

[5] ImageJ, https://imagej.net/ij/index.html/ [Online]. [Pristupljeno: 11-09-2023]

[6] itk-SNAP, http://www.itksnap.org/pmwiki/pmwiki.php [Online]. [Pristupljeno: 11-09-2023]

[7] OsiriX DICOM Viewer, https://www.osirix-viewer.com/ [Online]. [Pristupljeno: 11-09-2023]

[8] Du J, Zhang X, Liu P, Wang T., Coarse-RefinedConsistencyLearningUsingPixel-LevelFeatures for Semi-SupervisedMedicalImageSegmentation. IEEE Journal ofBiomedicaland Health Informatic,. Svibanj 2023. [Online]. Dostupno na: https://ieeexplore.ieee.org/document/10131969 [Pristupljeno: 24-06-2023]

[9] AIM-PD-L1, PathAI, [Online]. Dostupno na: https://www.pathai.com/aim-pd-l1/ [Pristupljeno: 28-06-2023]

[10] „NETSCAPE AND SUN ANNOUNCE JAVASCRIPT, THE OPEN, CROSS-PLATFORM OBJECT SCRIPTING LANGUAGE FOR ENTERPRISE NETWORKS AND THE INTERNET", Netscape Company Press Relations,Prosinac 1995. [Online]. Dostupno na: https://web.archive.org/web/20070916144913/https://wp.netscape.com/newsref/pr/newsrelease67.html [Pristupljeno: 25-06-2023]

[11] ECMAScript® 2024 LanguageSpecification,, Srpanj 2023. [Online]. Dosupno na: https://tc39.es/ecma262/[Pristupljeno: 25-06-2023]

[12] Whatistypeerasure?. [Online]. Dostupno na : https://github.com/microsoft/TypeScript/wiki/FAQ#what-is-type-erasure [Pristupljeno: 26-06-2023]

[13] ThePhilosophyof Ruby -A ConversationwithYukihiroMatsumoto, Bill Venners, Rujan 2003. [Online]. Dostupno na: https://www.artima.com/articles/the-philosophy-of-ruby [Pristupljeno: 05-08-2023]

[14] The Rails Doctrine., David HeinemeierHansson, Dostupno na: https://rubyonrails.org/doctrine#convention-over-configuration [Pristupljeno: 04-08-2023]

[15] Michael Stonebraker, Lawrence A. Rowe. „THE DESIGN OF POSTGRES“. University of California Berkeley, 1986.

[16] „A briefhistoryof CSS unitl 2016“, Bert Bos, w3.org, Prosinac 2016. [Online]. Dostupno na: https://www.w3.org/Style/CSS20/history.html [Pristupljeno: 24-07-2023]

[17] Jones M, Bradley K, Sakimura N, JSON Web Token(JWT), Svibanj 2015. [Online]. Dostupno na: https://datatracker.ietf.org/doc/html/rfc7519 [Pristupljeno: 03-08-2023]

POPIS KRATICA

Kratica	Engleski naziv	Hrvatski naziv
ACID	atomicity, consistency, isolation, durability	atomičnost, konzistentnost, izolacija, trajnost
API	ApplicationProgramming Interface	Aplikacijsko programsko sučelje
CLI	Command Line Interface	Sučelje naredbenog retka
CSS	CascadingStyleSheets	Kaskadni stilski obrasci
CORS	Cross-OriginResourceSharing	Dijeljenje resursa preko različitih izvora
DOM	DocumentObject Model	Model objekata dokumenta
ENV	Environment	Okruženje
ERB	Embedded Ruby	Ugrađeni Ruby
HTML	HyperTextMarkupLanguage	Hipertekstualni označni jezik
HTTP	HyperText Transfer Protocol	Protokol za prijenos hiperteksta
JIT	Justin Time	Upravo na vrijeme
JSON	JavaScript objectnotation	JavaScript objektna notacija
JSONb	JSON binary	Binarni JSON
JSX	JavaScript XML	JavaScript XML
JWT	JSON Web Token	JSON internetski token
ORM	Object-RelationalMapper	Maper objekata i relacija
SASS	SyntacticallyAwesomeStyleSheets	Sintaktički izvrsni stilski obrasci
SCSS	Sassy CSS	Sassy CSS
SPA	Single pageapplication	Aplikacija na jednoj stranici
SQL	StructuredQueryLanguage	Strukturirani upitni jezik
URL	Universalresourcelocator	Univerzalni lokator resursa

SAŽETAK

Ovaj rad opisuje razvoj web sustava za označavanje medicinskih slika u svrhe strojnog učenja. Sustav omogućuje učitavanje i anotiranje slika te izvoz anotiranih slika u prikladnom formatu. Za izradu web sustava korištene su razne tehnologije za razvoj web aplikacija; React JavaScript biblioteka za izradu klijentske aplikacije, Fabric.js biblioteka za izradu platna za označavanje te Ruby on Rails za izradu poslužiteljske aplikacije. Dana je detaljna specifikacija zahtjeva i prikazan je cijeli proces razvoja web sustava. Uz prikazane isječke kodova, prikazan je i izgled web sustava uz detaljan pregled korisničkog iskustva. Rezultat ovog rada je web sustav koji svojim korisnicima i administratorima pruža jednostavno sučelje i kvalitetne alate za provođenje cjelokupnog procesa označivanja medicinskih slika.

Ključne riječi: označavanje medicinskih slika, web sustav, Ruby on Rails, React, strojno učenje

ABSTRACT

Web system for the annotation and preparation of medical images for machine learning purposes

This paper presents the development of a web system for annotating medical images for machine learning purposes. The system enables image loading, annotation, and export of annotated images in a suitable format. Various web technologies were utilized in the system's development, including the React JavaScript library for the frontend application, the Fabric.js library for developing the annotating canvas, and Ruby on Rails for the backend application. A detailed requirement specification is given, and the entire development process is shown. Alongside the displayed code snippets, the visual design of the web system is presented, providing a detailed overview of the user experience. The result of this paper is a web system that provides its users with a simple interface and valuable tools for conducting the medical image annotation process.

Keywords: medical image annotation, web system, Ruby on Rails, React, machine learning

ŽIVOTOPIS

Luka Bak rođen je 6. travnja 1999. godine u Vinkovcima. Pohađao je Osnovnu školu Ivan Goran Kovačić Vinkovci, nakon koje upisuje Tehničku školu Ruđer Bošković Vinkovci, smjer tehničke gimnazije. Tijekom školskog obrazovanja sudjeluje u nekoliko natjecanja, uključujući i državno natjecanje iz informatike. Nakon završene srednje škole upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Od 2020. godine zaposlen je kao student u Bamboo Lab na poziciji backend developera.

Luka Bak

PRILOZI

Na priloženom disku nalazi se:

- Programski kod klijentske aplikacije web sustava
- Programski kod poslužiteljske aplikacije web sustava
- Završni rad u .docx formatu
- Završni rad u .pdf formatu