Sustav za dojavu problematike, provedbe mjera dezinfekcije, dezinsekcije i deratizacije (DDD) nadležnoj jedinici lokalne samouprave

Sinožić, Sebastian

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike u Puli

SEBASTIAN SINOŽIĆ

SUSTAV ZA DOJAVU PROBLEMATIKE, PROVEDBE MJERA DEZINFEKCIJE, DEZINSEKCIJE I DERATIZACIJE (DDD) NADLEŽNOJ JEDINICI LOKALNE SAMOUPRAVE

Diplomski rad

Pula, lipanj, 2020 godine

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike u Puli

SEBASTIAN SINOŽIĆ

SUSTAV ZA DOJAVU PROBLEMATIKE, PROVEDBE MJERA DEZINFEKCIJE, DEZINSEKCIJE I DERATIZACIJE (DDD) NADLEŽNOJ JEDINICI LOKALNE SAMOUPRAVE

Diplomski rad

JMBAG: 0303046213, redoviti student

Studijski smjer: Informatika

Kolegij: Izrada informatičkih projekata

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijsko komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i Informatologija

Komentor: doc. dr. sc. Nikola Tanković

Mentor: doc. dr. sc. Siniša Sovilj

Pula, lipanj, 2020 godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisan Sebastian Sinožić, kandidat za magistra informatike, ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Sebastian Sinožić

U Puli, lipanj, 2020. godine

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, Sebastian Sinožić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Sustav za dojavu problematike, provedbe mjera dezinfekcije, dezinsekcije i deratizacije (DDD) nadležnoj jedinici lokalne samouprave“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

Student

Sebastian Sinožić

Sadržaj

UVOD

Na temelju sve učestalijih apela raznih neovisnih udruga, Zavoda za javno zdravstvo naše županije, izvodi se zajednički nazivnik koji ima naglasak na očuvanju našeg planeta Zemlje. Djelujući kroz razne aspekte u cilju smanjenja stakleničkih plinova, potrošnje fosilnih goriva, smanjenja otrova koji se kroz razne proizvodne i druge procese odlažu u prirodu.

Sukladno važećoj zakonskoj regulativi, jedinice lokalne i područne uprave, odnosno, samouprave, dužne su provoditi unutar svojeg djelokruga poslova, preventivne mjere sukladno Programu mjera preventivne dezinfekcije, dezinsekcije i deratizacije (DDD). Sva djelovanja unutar predmetne tematike, vrše se uz koordinaciju putem Zavoda za javno zdravstvo Istarske županije (ZZJZ IŽ). Kroz tekstove i radijske emisije, stalan je naglasak struke na prevenciji, odnosno, identifikaciji problematike kako bi blagovremeno djelovali na sprečavanju razvoja raznih nametnika koje struka unutar DDD mjera provodi na terenu. Primjerice, potrebno je što ranije uočiti razna legla komaraca, iz kojih se razvijaju ličinke, odnosno, u kasnijem stadiju razvoja, komarci raznih tipova koji predstavljaju prave molestante te nam time uvelike zagorčavaju život.

Pravovremenim djelovanjem, dok su komarci još u razvoju (larvicidna faza), u prirodu se polaže daleko manji broj tvari koje ličinke komaraca sprečavaju u svojem daljnjem razvoju te ih drže na biološkom minimumu. Kasnijim djelovanjem, kroz razne tretmane i špricanja odraslih komaraca (adulticidna faza), uz vrlo mali učinak (od svega 10 do 12 %), u prirodu se nehotice apliciraju razna sredstva i otrovi. Tom prilikom, uzgredno i nepotrebno tretiramo voćnjake, vrtove, pčele i ostale izuzetno važne kukce, a naravno, dobar dio tih tvari završava i u našim domovima.

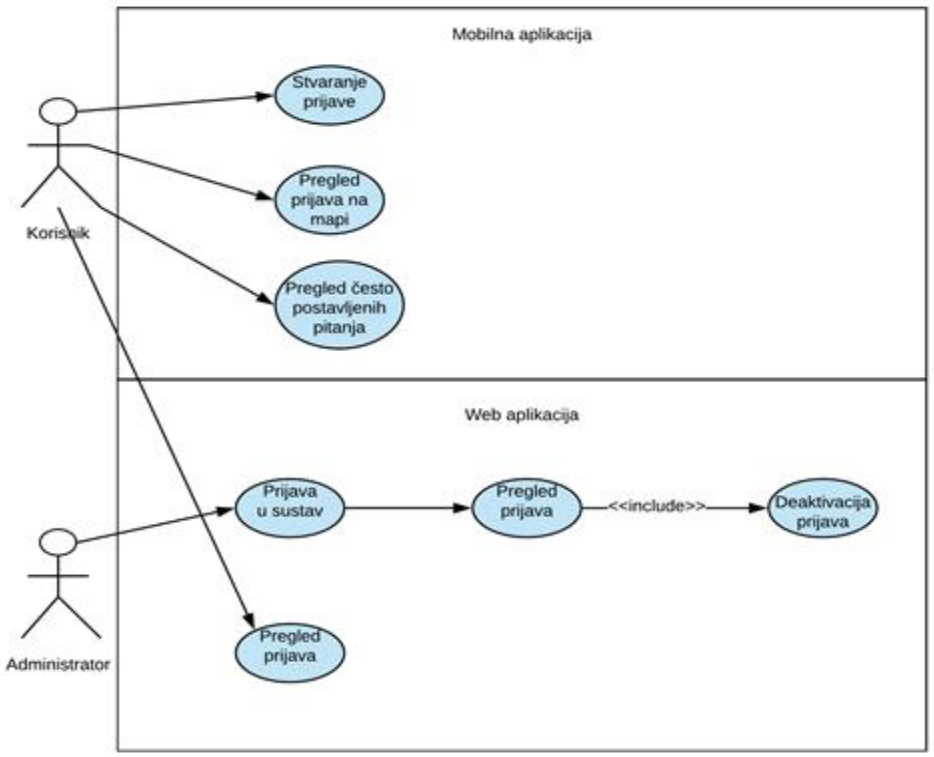
Kako bi se spriječila ustaljena praksa zakašnjelog djelovanja, potrebno je probuditi i educirati svijet naših stanovnika. Jedan od načina je svakako i tema ovog diplomskog rada, odnosno, razvoj sustava za dojavu problematike, provedbe mjera dezinfekcije, dezinsekcije i deratizacije (DDD) nadležnoj jedinici lokalne samouprave.

1. Prijedlog rješenja

U sljedećem poglavlju prikazan je prijedlog rješenja putem raznih modela. Interakciju korisnika s web i mobilnom aplikacijom prikazani su s Use Case modelom, a kasnije prošireni s use case sequence modelima. Za prikaz arhitekture kompletne aplikacije korišten je komponentni model.

1.1 Use Case model

U sljedećem Use Case modelu prikazana je interakcija korisnika s web i mobilnom aplikacijom, te administratora isključivo s web aplikacijom. Korisnik u mobilnoj aplikaciji može stvoriti prijavu, što je ujedno i glavna funkcionalnost mobilne aplikacije. Također može pregledati vlastite prijave te prijave ostalih korisnika na mapi, gdje se vidi u koju kategoriju prijava pripada, naslov i lokaciju prijave. Zadnja funkcionalnost mobilne aplikacije jest mogućnost pregleda često postavljenih pitanja gdje zapravo korisnik može steći nova znanja te čak i pojedine odgovore na namjeravana pitanja, a koje su ostali korisnici već prijavili.



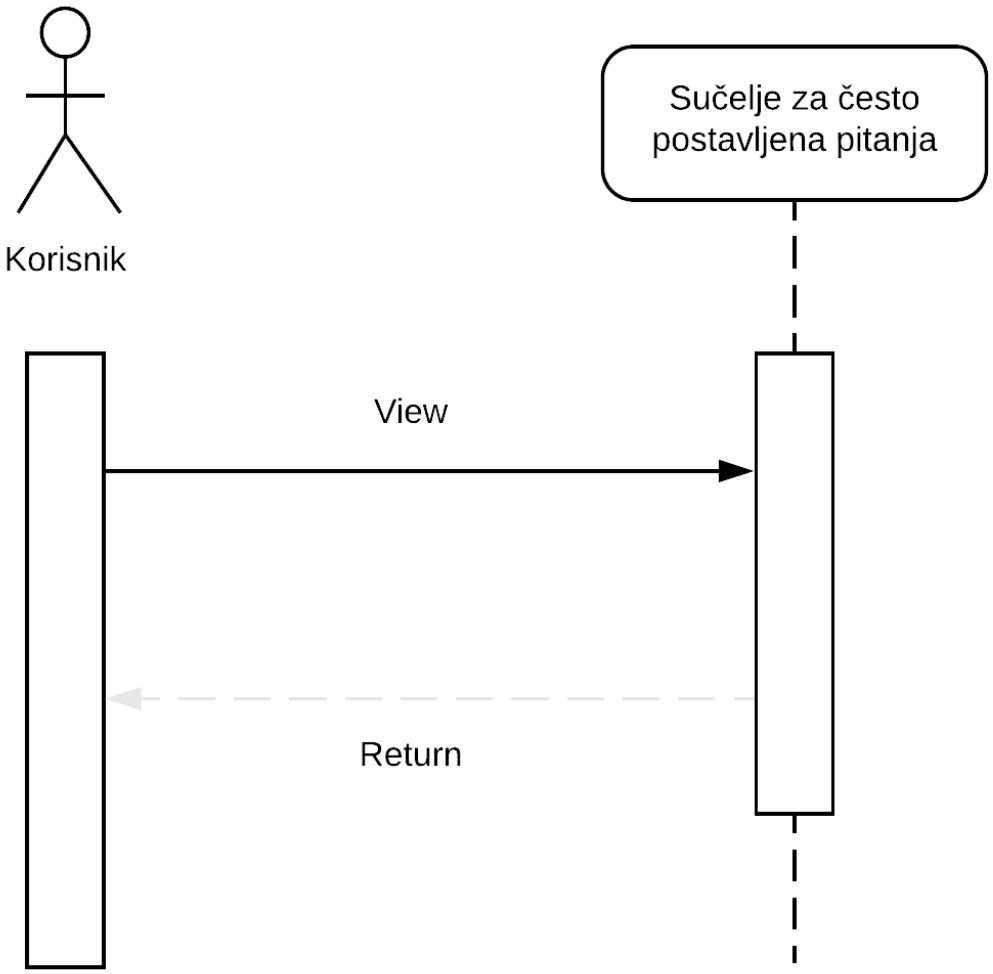
Slika 1. Use Case diagram

Izvor: Autor

1.2 Use case Sequence model

U sljedećim use case sequence modelima prikazati ćemo interakciju korisnika i administratora sa svim sučeljima mobilne i web aplikacije, a sve u svrhu boljeg razumijevanja samih interakcija.

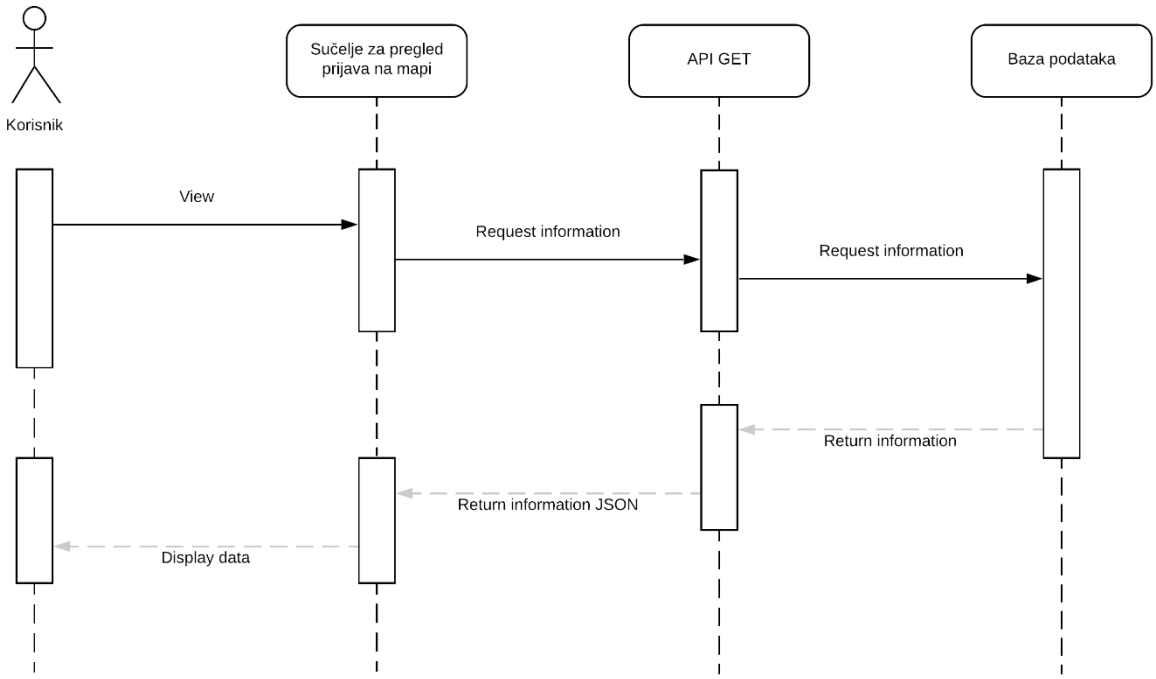
1.2.1 Mobilna aplikacija



Slika 2. Interface for frequently asked questions

Izvor: Autor

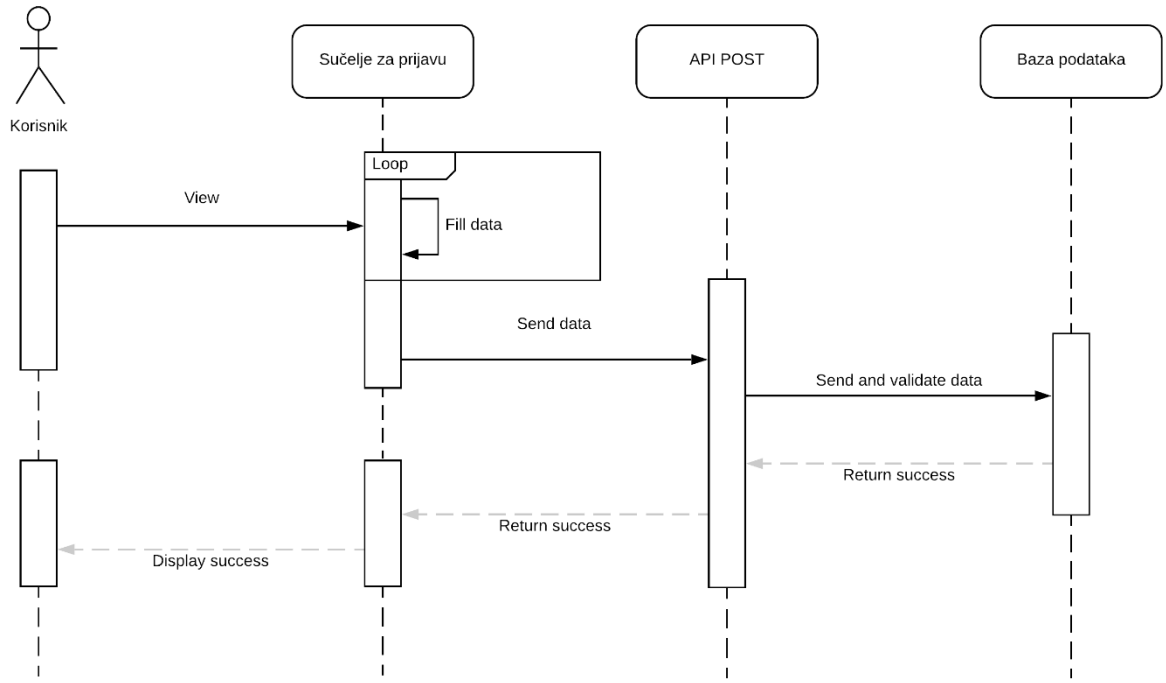
U sučelju za često postavljena pitanja imamo korisnika koji pristupa samom sučelju. Sučelje uspješno vraća informacije korisniku.



Slika 3. Map overview interface

Izvor: Autor

U sučelju za pregled prijava na mapi korisnik pristupa samom sučelju. Sučelje dalje traži informacije od API-a (kategoriju, naslov, lokaciju). U sljedećem koraku API traži informacije od baze podataka te nakon što ih dobije, odradi samu validaciju podataka i pošalje ih natrag sučelju u odgovarajućem formatu. Na kraju sučelje uspješno prikazuje korisniku podatke.

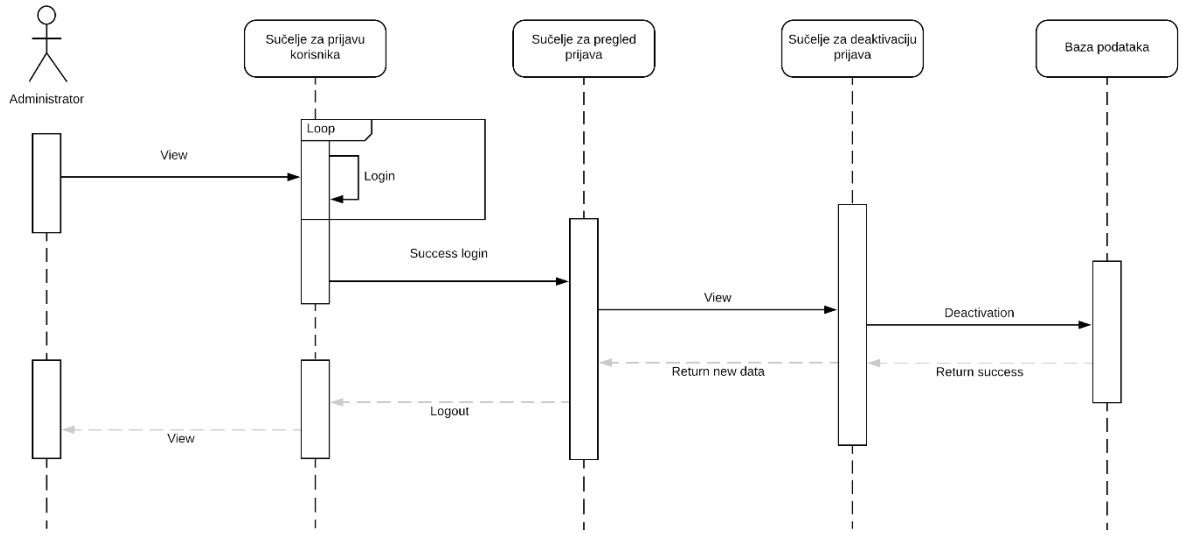


Slika 4. Login view interface

Izvor: Autor

U sučelju za prijavu, korisnik pristupa samom sučelju. Korisnik ispunjava potrebne podatke na sučelju za prijavu, sve dok nisu uspješno validirani. Sučelje dalje šalje informaciju na API, koji se brine da su svi podaci validni i uspješno došli. Nakon validacije podataka API šalje podatke u bazu podataka. Po uspješnom slanju podataka u bazu podataka, ista vraća povratnu informaciju sučelju o uspješnom zapisu. Na kraju sučelje prikazuje korisniku informaciju o uspješnoj prijavi.

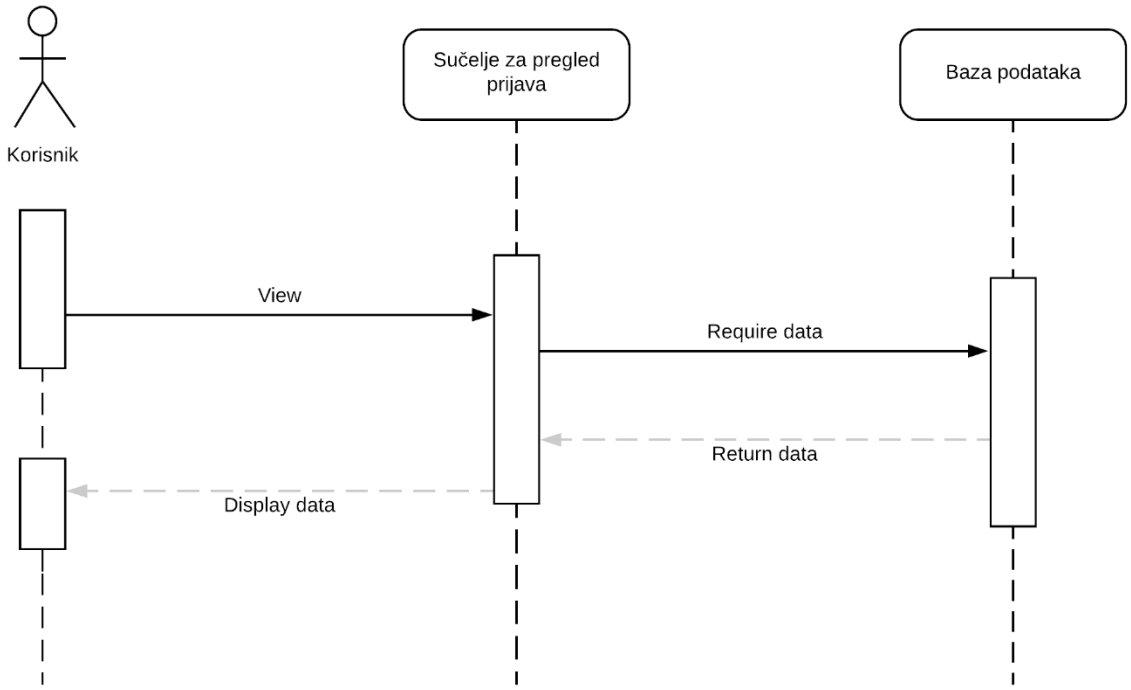
1.2.2 Web aplikacija



Slika 5. Application deactivation interface

Izvor: Autor

U sučelju za deaktivaciju prijava, administrator prvo mora pristupiti sučelju za prijavu korisnika. U sučelju za prijavu, administrator mora uspješno unijeti podatke sve dok nisu ispravni kako bi mogao pristupiti sljedećem sučelju. Nakon uspješne prijave administrator dolazi do sučelja za pregled prijava koje u sebi sadrži sučelje za deaktivaciju. Administrator odabere željenu prijavu te je šalje u postupak same deaktivacije. Sučelje za deaktivaciju prijava brine se o komunikaciji s bazom podataka te o samoj deaktivaciji prijave. Nakon uspješne deaktivacije, baza podataka vraća uspješnu informaciju sučelju za deaktivaciju prijava, koje izravno komunicira s sučeljem za pregled prijava. Administrator sada može vidjeti svježe informacije tj. uspješno deaktiviranu prijavu. Nakon završetka rada, administrator se odjavljuje te vraća na sučelje za prijavu.



Slika 6. Application review interface

Izvor: Autor

U sučelju za pregled prijava, korisnik pristupa samom sučelju. Zatim sučelje direktno komunicira s bazom podataka te potražuje podatke. Baza podataka vraća podatke sučelju te ih prikazuje korisniku.

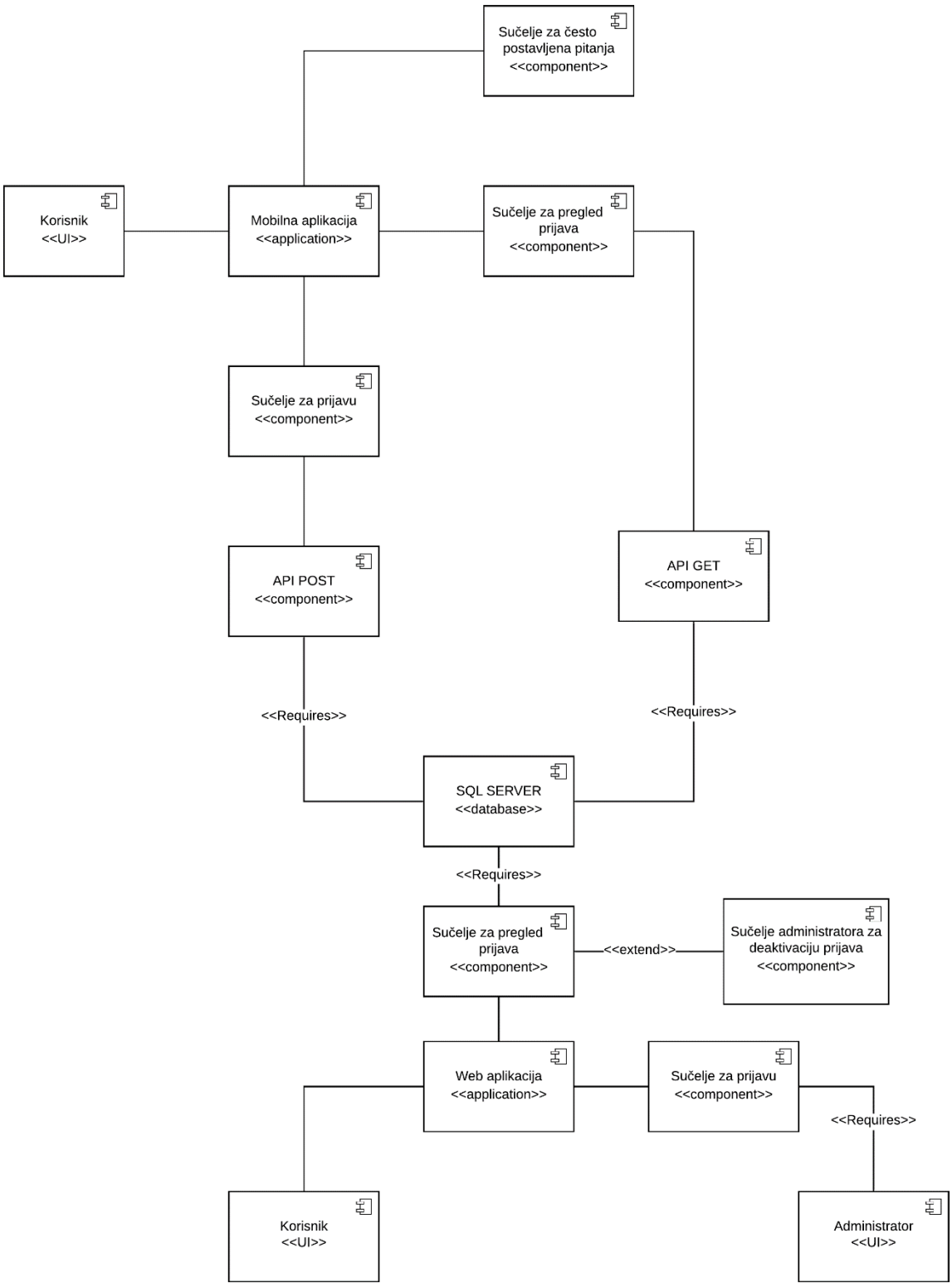
1.3 Komponentni model

U sljedećem dijagramu (Slika 7.) prikazana je kompletna arhitektura mobilne i web aplikacije i na koji način međusobno komuniciraju. Arhitektura je odabrana na način da su mobilna i web aplikacija zasebne cjeline te komuniciraju preko API-a za ispravan rad kompletnog sustava. Mobilna aplikacije se sastoji od 3 glavnih komponenti:

1. Sučelje za prijavu,
2. Sučelje za pregled prijava,
3. Sučelje za često postavljena pitanja.

Sučelje za često postavljena pitanja je najjednostavnija komponenta koja komunicira isključivo s mobilnom aplikacijom. Sljedeća komponenta jest sučelje za prijavu koja direktno komunicira s POST API-em koji se brine za validaciju dostavljenih podataka, spremanja proslijeđenih slika direktno na tvrdi disk, a potom prosljeđivanjem podataka u samu bazu podataka. Sljedeća komponenta koju ćemo opisati je sučelje za pregled prijava, koja komunicira s GET API-em, s razlikom da u konkretnom slučaju API dostavlja podatke komponenti. Kako bi API mogao generirati podatke za sučelje, potrebno je da preuzme podatke iz baze podataka, formatira ih na pogodan način te ih potom dostavi sučelju za pregled prijava.

Web aplikaciji pristupaju dva tipa korisnika tj. običan korisnik i administrator. U slučaju običnog korisnika imamo samo jednu komponentu odnosno sučelje za pregled prijava, dok je kod administratora isti slučaj samo što je komponenta proširena s funkcionalnošću za deaktivaciju. Komponenta funkcionira na način da jednostavno čita informacije iz baze podataka te ih prikazuje na webu, a zapravo to su iste informacije koje su poslane iz mobilne komponente za prijavu. U slučaju administratora postoji dodatna komponenta odnosno sučelje za prijavu, koje služi kako bi se omogućila funkcionalnost deaktiviranja prijava.



Slika 7. Component model of the entire application

Izvor: Autor

2. Implementacija rješenja sustava

Implementacija rješenja sustava u nastavku je prikazana kroz četiri osnovne cjeline; Korištene tehnologije, prikaz i opis baze podataka te implementacija i prikaz mobilne i web aplikacije s njihovim grafičkim sučeljima.

2.1 Korištene tehnologije

2.1.1 React native

React native je javascript okvir (eng. Framework) koji služi za pisanje stvarnih, izvornih mobilnih aplikacija koje podržavaju platforme IOS i Android. Temelji se na Facebook-ovoj Javascript biblioteci za izgradnju korisničkih sučelja koje su pogodne za mobilne platforme. Slično kao React za razvoj Web-a, React Native aplikacije pisane su u kombinaciji Javascripta i XML-a (Extensible Markup Language). Potom, u pozadini, React Native poziva izvorne API-e (Application Programming Interface) za prikaz sučelja u Objective-C (za IOS) ili Jave (za Android). Iz tog razloga, aplikacije će se prikazivati iz stvarnih mobilnih komponenti, a ne iz web preglednika, te izgledati će kao i svaka druga mobilna aplikacija. Neki od razloga zbog kojih je React Native postao široko rasprostranjeni programski jezik su sljedeći: [1]

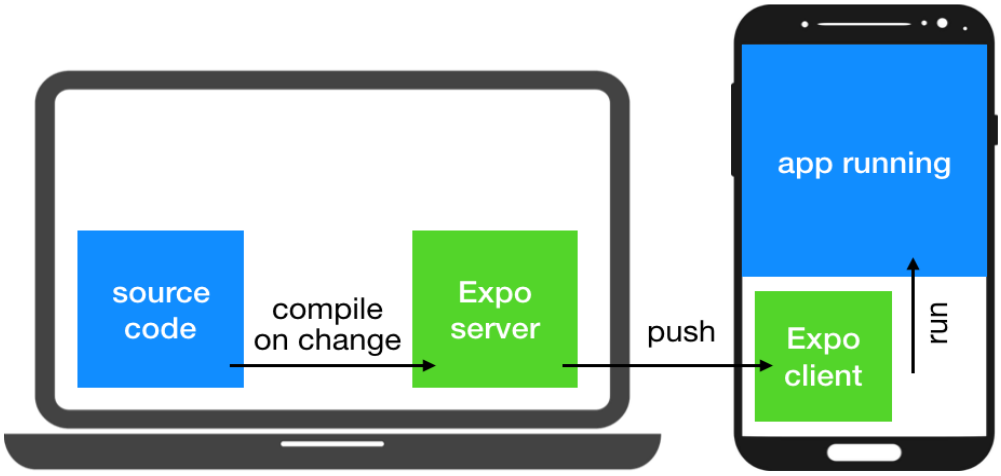
- Brže kompiliranje koda,
- Jedan jezik, više platformi,
- Izravno vidljive promjene bez potrebe kompiliranja koda,
- Manji timovi,
- Brze aplikacije,
- Pojednostavljeno korisničko sučelje.

[1] React Native Pros and Cons - Facebook's Framework in 2019 - https://www.netguru.com/blog/react-native-pros-and-cons

2.1.2 .Expo

Expo je framework i platforma za univerzalne React Native aplikacije. To je zapravo skup alata i usluga izgrađenih oko React Native koji pomažu u razvoju, izgradnji, implementaciji mobilnih aplikacija za Androiod i IOS platforme. Expo ima mnoge prednosti u odnosu na „klasičnu“ React Native razvoju okolinu, kao što su: [2]

- Postavljanje projekta je jednostavno i može se odraditi u nekoliko minuta,
- Istovremeno otvaranje projekta za jednu ili više osoba,
- Dijeljenje aplikacije je jednostavno (putem QR koda ili veze), ne mora se slati cijela .apk ili .ipa datoteka,
- Nema potrebe za kompiliranjem kako bi se pokrenula aplikacija,
- Integrira pojedine osnovne biblioteke iz React Native-a,
- Expo može izgraditi datoteke eksttenzija .apk i .ipa koje se kasnije mogu komercijalizirati u trgovine za platforme IOS i Android (App store i Google play).



Slika 8. Expo development environment architecture

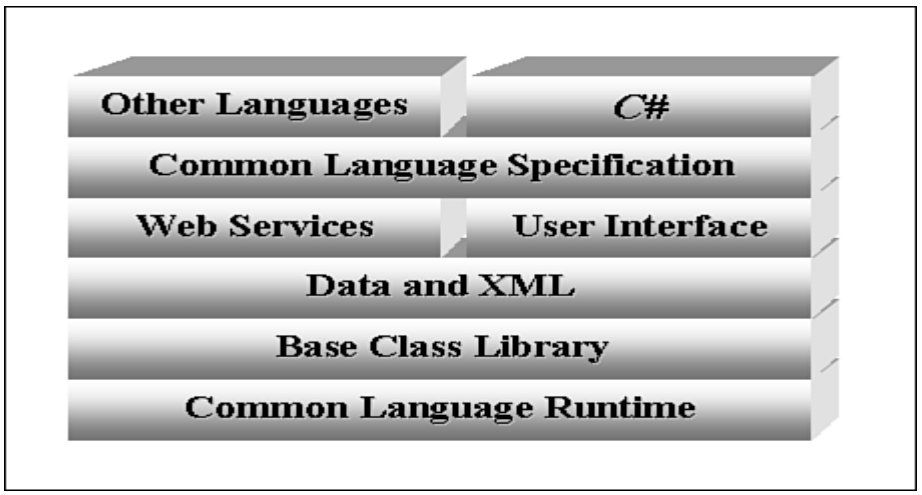
Izvor: Developing Mau King – https://medium.com/developing-mau-king/developing-mau-king-react-native-in-expo-vs-ejected-mode-and-how-to-get-both-at-the-same-time-f36e5af607dc

[2] Understanding Expo for React Native, 2018. Hackernoon - https://hackernoon.com/understanding-expo-for-react-native-7bf23054bbcd

2.1.3 C#

C# je objektno orijentirani programski jezik općih namjena, izvorno razvijen 2001. godine od strane Microsofta. Hibrid je C i C++ programskih jezika. C# je jednostavan, moderan, fleksibilan te, što je jedan od važnijih razloga, besplatan. Koristi se XML (Extensible Markup Language) web uslugama na .NET (dot net) platformi. Koristi se za izradu gotovo svega, ali posebno je pogodan u izgradnji Windows desktop aplikacija i igara. Također može se koristiti za razvoj web aplikacija, a sve je popularniji i za mobilni razvoj. U nastavku su navedeni razlozi zašto je C# postao jezik širokih namjena i korištenja:[3]

- Moderan i lagan za koristiti,
- Brz te ima kod otvorenog izvora,
- Namijenjen za više platformi,
- Siguran,
- Namijenjen za sve vrste upotreba,
- I dalje u razvoju.



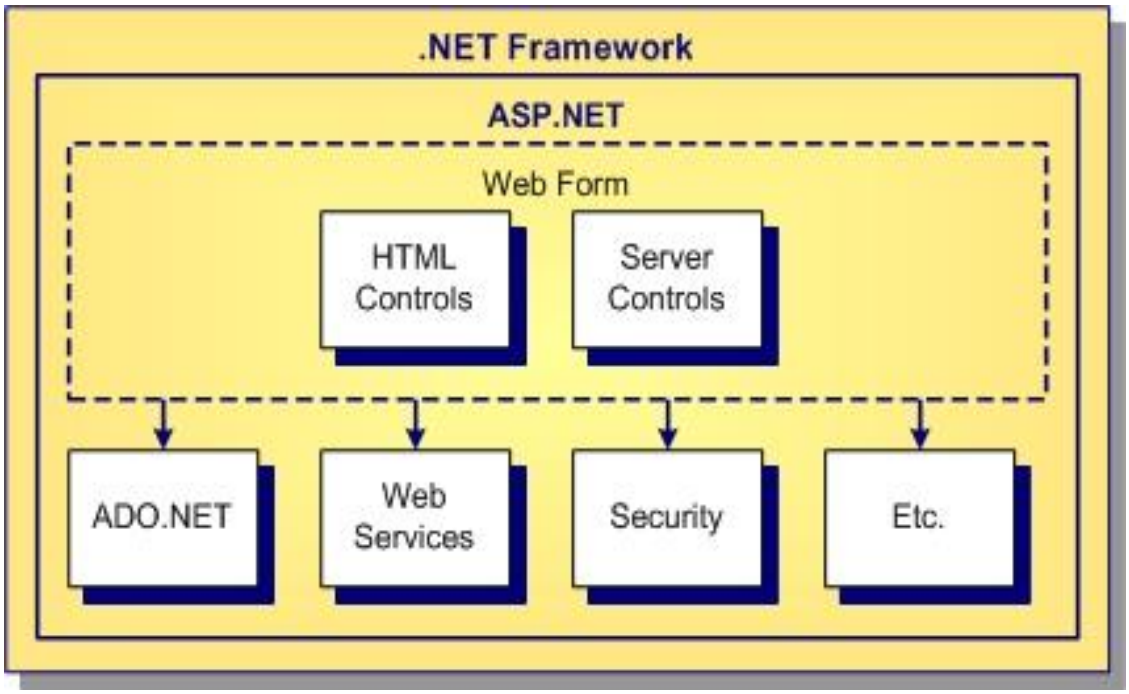
Slika 9. C# programming language architecture

Izvor: O'REILLY – C# Unleashed by Jseph Mayo, https://www.oreilly.com/library/view/c-unleashed/067232122X/067232122X_ch01lev1sec3.html

[3] Ben, A., Joseph, A. (2017), C# 7.0 in a Nutshell, Sebastopol: O'Reilly

2.1.4 ASP .NET

ASP .NET je softverski okvir otvorenog tipa na strani poslužitelja, stvoren od Microsofta početkom 2000-ih. Prva verzija ASP .NET-a bila je 1.0, a danas je već u uporabi 4.6. inačica. ASP .NET izgrađen je na .NET okviru (framework) koji pruža sučelje aplikacijskog programa (API) za softverske programere. Alati za razvoj .NET mogu se koristiti za stvaranje standardnih Windows aplikacija, ali i za web aplikacije. Kako bi web aplikacije ispravno funkcionirale, moraju biti objavljene na poslužiteljima koji podržavaju ASP .NET programe. Microsoftov web-poslužitelj internetskih informacija (IIS) daleko je najčešća platforma koja se koristi za ASP .NET aplikacije.



Slika 10. ASP .NET framework architecture

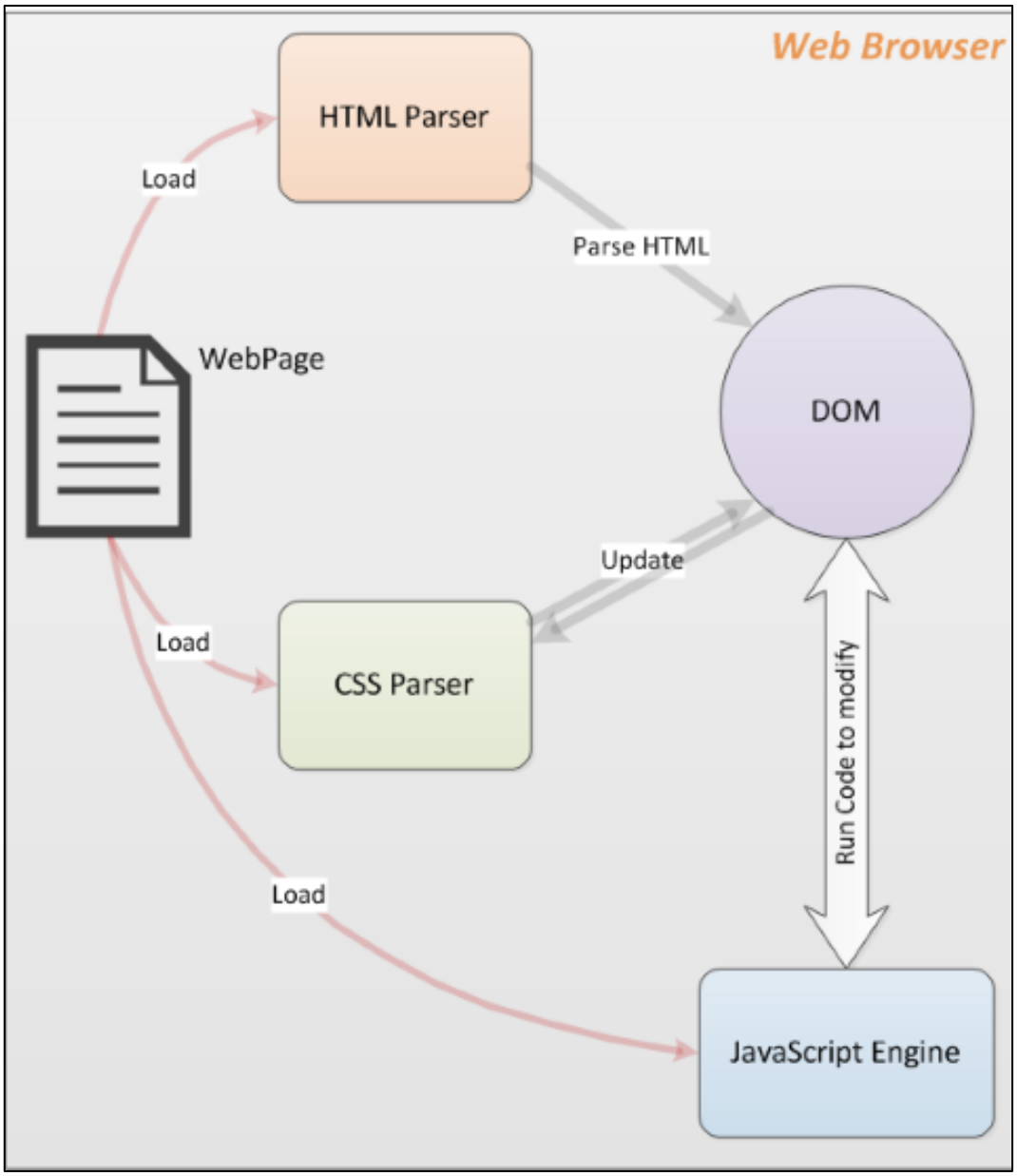
Izvor: ASP.NET Overview, http://docs.embarcadero.com/products/rad_studio/radstudio2007/RS2007_helpupdates/HUpdate4/EN/html/devnet/aspnetov_xml.html

2.1.5 Javascript

Javascript je skriptni programski jezik; lagan je, objektno orijentiran i najčešće se koristi kao dio web stranica, čija implementacija omogućuje skripti da se izvršava na strani klijenta. Prvobitno, javascript bio je poznat pod nazivom LiveScript, a pojavio se u NetScape 2.0 verziji 1995. godine. Javascript se koristi za stvaranje interaktivnih elemenata za web stranice, poboljšavajući korisničko iskustvo, a danas je implementiran u gotovo sve preglednike. Stvari poput izbornika, animacija, videoplayera, pa čak i jednostavnih igara u pregledniku, mogu se s javascriptom razviti brzo i jednostavno. Javascript danas je jedan od najpopularnijih skriptnih jezika, a u nastavku su navedeni razlozi njegovog korištenja: [4]

1. Brzina,
2. Jednostavnost,
3. Popularnost,
4. Kompatibilnost s ostalim jezicima,
5. Ažuriranja dijelova stranica bez korištenja serverske strane.

[4] Marijn, H. (2018) Eloquent Javascript – 3rd edition

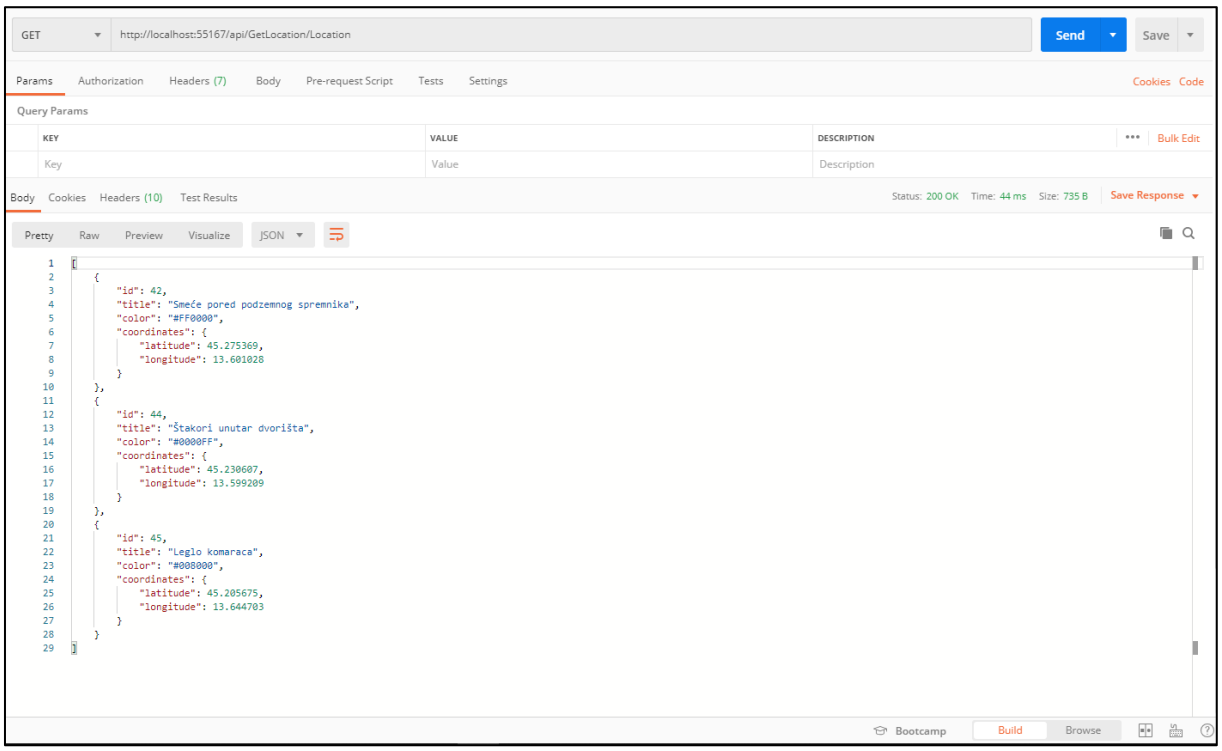


Slika 11. Javascript architecture on web pages

Izvor: MakeUseOf – What is JavaScript and How Does It Work?, https://www.makeuseof.com/tag/what-is-javascript/

2.1.6 Postman

Postman je razvojni alat za API (application programming interface) koji pomaže u izgradnji, testiranju i izmjeni samih API-ja, a koristi ga aktivno više od 5 milijuna programera svakog mjeseca. Ima mogućnost upućivanja različitih vrsta HTTP zahtjeva (GET, POST, PUT, PATCH), stvaranja i spremanja okruženja za kasniju upotrebu, pretvaranja API koda u različite jezike (Python, Javascript) itd.

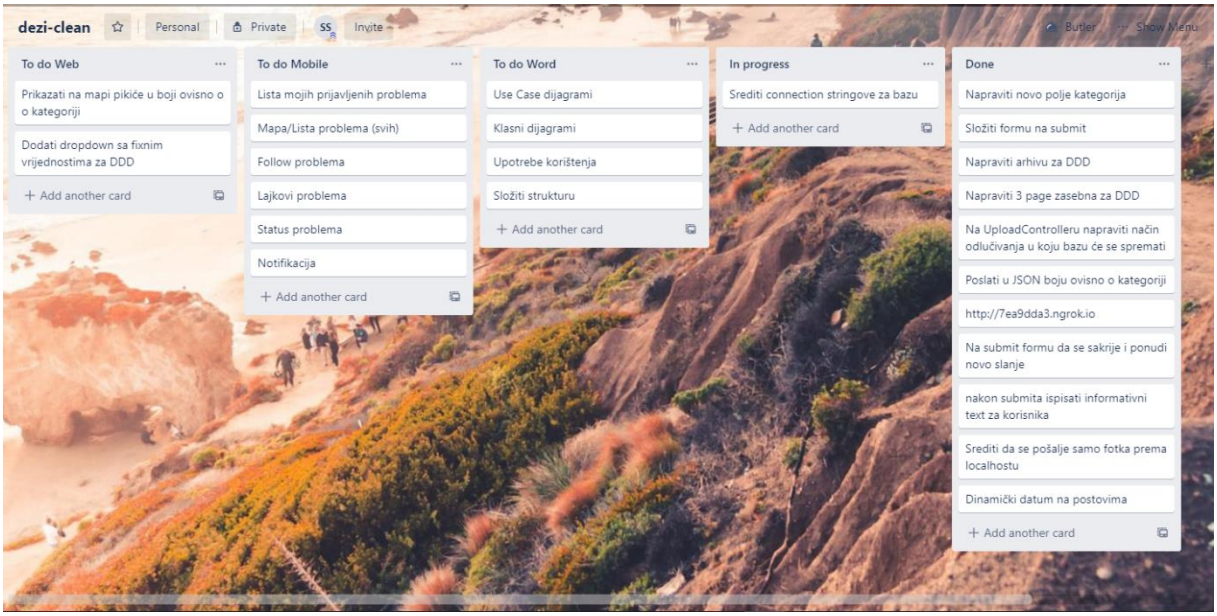


Slika 12.Simple postman query

Izvor: Autor

2.1.7 Trello

Trello je alat za suradnju koji pruža vizualni pregled onoga na čemu se radi, tko radi na tome te koliko je posla obavljeno. Alat projekte organizira u „ploče“, a zatim u kartice i popise. Platforma je nadahnuta sustavom KanBan koji je Toyota razvila kao način održavanja fleksibilnosti uz visoku razinu proizvodnje. Trello je svojevrsna unija alata, i to alata za popis obveza, alata za vizualno praćenje ili pak za upravljanje projektima koji omogućuje dodjeljivanje zadataka timu, dodavanje spiskova zadataka, rasprava, dokumenata, slika i praćenje njihovog napretka.



Slika 13. Project view through Trello tool

Izvor: Autor

2.2 Baza podataka

Za izradu baze podataka korišten je Microsoft SQL Server, što je u biti sustav za upravljanje relacijskim bazama podataka (RDBMS). Prvenstveno je osmišljen i razvijen kao konkurencija MySQL i Oracle bazama podataka. Baziran je na standardnom SQL programskom jeziku (Structured Query Language). SQL je poseban programski jezik dizajniran za obradu podataka u relacijskim bazama podataka. Međutim, SQL Server dolazi sa vlastitom implementacijom T-SQL jezika (Transact-SQL). Transact-SQL omogućuje deklariranje varijabli, postupanje s iznimkama, pohranjene procedure i sl. Korišteni alat za pristupanje SQL Serveru je Microsoft SQL Server Managment Studio (SSMS).

U SQL Serveru definirane su dvije tablice koje su potrebne kako bi aplikacija funkcionirala prema očekivanju:

- Tablica dbo.Data: sadrži sve podatke koje se prikupljaju iz mobilne aplikacije, a koje se kasnije koriste za web aplikaciju,

- Tablica dbo. Users: sadrži podatke o korisnicima koji pristupaju web aplikaciji.

2.2.1 Tablica dbo.Data

Tablica dbo.Data sastoji se od 12 atributa. Prvi atribut tablice je „Id“ koji je postavljen na automatsko povećanje (eng. Autoincrement) odnosno primarni ključ tablice koji se automatizmom postavlja kod dodavanje novog reda u tablici. Atribut „Title“ je tipa varchar(50) odnosno znakovnog niza duljine pedeset znakova u kojem se sprema naslov same prijave. Atributi „name“ i „lastname“ su također tipa varchar(50) u kojima se spremaju ime i prezime prijavitelja. Atribut „problemdescription“ je tipa varchar(max) u kojem se sprema opis prijave, a zbog same veličine opisa koji može dosegnuti poveći broj znakova, atribut je postavljen na „max“. Sljedeći atributi su „latitude“ i „longitude“ koji su tipa varchar(50) koji označavaju lokaciju latitude i longitude iz mobilne aplikacije. Atribut „Imagepath“ je tipa varchar(max) koji označava putanju slike do mjesta gdje je pohranjena slika na disku. Zbog potencijalno velikih imena slika, atribut je postavljen na „max“. Atribut „date“ je tipa varchar(50) u kojem se pohranjuje datum stvaranja prijave. Atribut „category“ je tipa varchar(20) odnosno

duljine 20 znakova, a označava kategoriju prijave koja se šalje iz mobilne aplikacije. Atribut „color“ je tipa varchar(20), a zapravo označava vrstu kategorije po boji koja kasnije služi za prikaz karte na mobilne aplikaciji. Posljednji atribut „aktivan“ je tipa varchar(20) u kojem se pohranjuju 2 vrijednosti, odnosno „true“ koji označava da je prijava još uvijek aktivna ili „false“ ako je prijava obrađena.

Tablica 1. Popis atributa u tablici dbo.Data

Naziv atributa	Tip podatka	Specifičnosti	Opis
Id	Int	Primarni ključ, obavezan unos	Jedinstveni identifikacijski broj
Title	Varchar(50)	Obavezno polje	Naziv prijave
name	Varchar(50)	Obavezno polje	Ime prijavitelja
lastname	Varchar(50)	Obavezno polje	Prezime prijavitelja
problemdescription	Varchar(max)	Obavezno polje	Opis prijave
latitude	Varchar(50)	Obavezno polje	Latituda lokacije
longitude	Varchar(50)	Obavezno polje	Longituda lokacije
Imagepath	Varchar(max)	Obavezno polje	Putanja slike do mjesta gdje je pohranjena slika
date	Varchar(50)	Obavezno polje	Datum stvaranja prijave
category	Varchar(20)	Obavezno polje	Kategorija prijave
color	Varchar(20)	Obavezno polje	Boja prijave koja služi za mobilnu aplikaciju
aktivan	Varchar(20)	Obavezno polje	Status prijave

Izvor: Autor

2.2.2 Tablica dbo.Users

Tablica dbo.Users je manja tablica koja se sastoji od 3 atributa. Kao i u prvoj tablici prvi atribut je „id“ koji je postavljen na automatsko povećanje (eng. Autoincrement) odnosno primarni ključ tablice koji se automatizmom postavlja kod dodavanje novog reda u tablici. Sljedeća 2 atributa su „username“ i „password“ koji su tipa varchar(20), a služe kao autentifikacija korisnika za web aplikaciju.

Tablica 2. Popis atributa u tablici dbo.Users

Naziv atributa	Tip podatka	Specifičnosti	Opis
Id	Int	Primarni ključ, obavezan unos	Jedinstveni identifikacijski broj
username	Varchar(50)	Obavezno polje	Korisničko ime
password	Varchar(50)	Obavezno polje	Lozinka

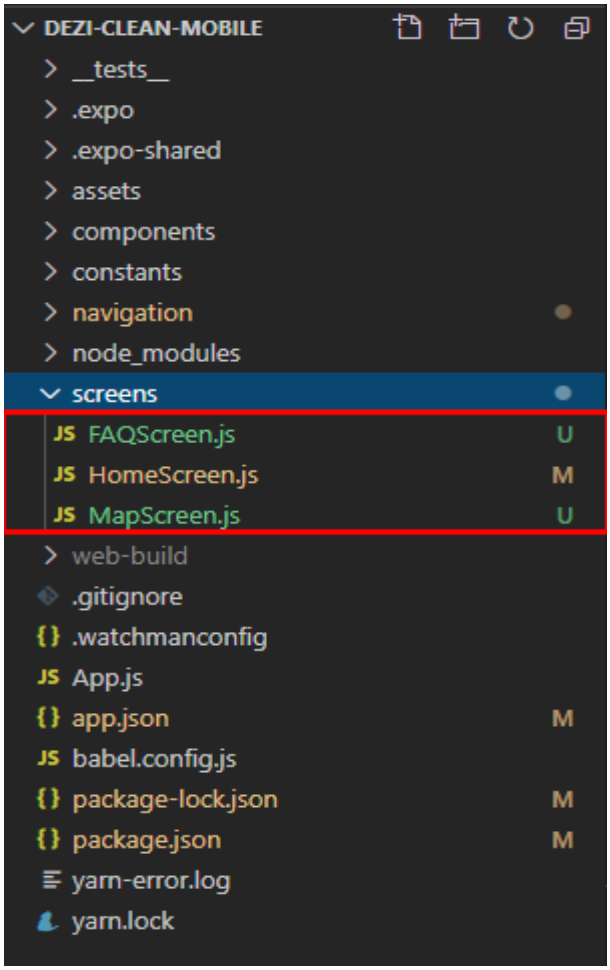
Izvor: Autor

2.3 Mobilna aplikacija

Mobilni dio aplikacije izrađen je u tehnologiji React Native u kombinaciji Expo platforme koja pomaže u razvoju, izgradnji i implementaciji mobilnih aplikacija. Spomenute tehnologije detaljnije su objašnjene na početku ovoga rada.

2.3.1 Prikaz datotečnog i programskog dijela web aplikacije

Mobilna aplikacija sustava za dojavu problematike, provedbe mjera dezinfekcije, dezinsekcije i deratizacije (DDD) nadležnoj jedinici lokalne samouprave sastoji se od tri glavne datoteke tj. zaslona koji će biti detaljnije opisani u sljedećim poglavljima, a koje su smještene u mapi „*screens*“. Datoteke su prikazane na slici 14., a sve završavaju s ekstenzijom .js.



Slika 14. Main files - screens

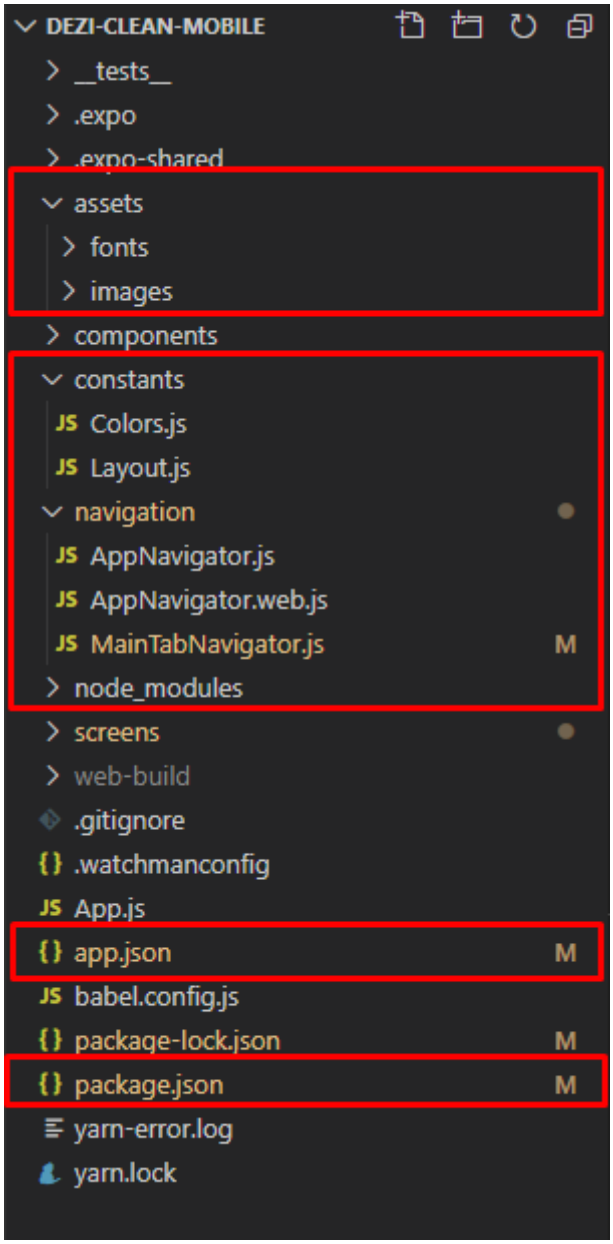
Izvor: Autor

Osim glavnih datoteka imamo sporedne mape s pripadajućim datotekama koje služe kako bi aplikacija ispravno funkcionirala. U mapi „*assets*“ nalaze se dodatne dvije podmape „*fonts*“ i „*images*“ u kojima se nalaze sve datoteke za fontove i slike koje upotrebljava aplikacija. U sljedećoj mapi „*constats*“ nalazimo datoteke „*colors.js*“ i „*layout.js*“ koje definiraju konstante vrijednosti aplikacije za boje i dimenzije. Mapa „*navigation*“ sastoji se od tri datoteke, a to su:

- AppNavigator.js
- AppNavigator.web.js
- MainTabNavigator.js

Navedene datoteke služe za kompletnu navigaciju mobilne aplikacije, te međusobno prebacivanje između zaslona (eng. Screens). Sljedeća jako važna mapa je „*node_modules*“ koja sadrži sve vanjske biblioteke koje pomažu u razvoju same aplikacije. Biblioteke se mogu preuzeti putem NPM5 (Node Package Manager-a). Za kraj treba spomenuti konfiguracijske datoteke „*app.json*“ i „*package.json*“ koje su vrlo važne, jer se u njima određuju prava pristupa komponentama mobilnih uređaja te verzije specifičnih paketa.

[5] What is npm?, 2011. Node.js - https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/



Slika 15. Other important files for the proper execution of the mobile application

Izvor: Autor

2.3.1.1 Implementacija i opis datoteke HomeScreen.js

U ovom poglavlju opisati ćemo prvi zaslon tj. „*HomeScreen.js*“ što je ujedno prvi zaslon pri pokretanju mobilne aplikacije, a služi za prijavu problematike. U ovoj datoteci nalazi se forma za prijavu te pomoćne funkcije za ispravan rad same forme. Forma se sastoji od padajućeg izbornika u kojem se definira kategorija problema kojeg želimo prijaviti. Odmah ispod padajućeg izbornika nalaze se polja u kojima

korisnik upisuje podatke. Na kraju forme nalazi se dva gumba, a to su gumb za odabir slike iz mobilnog uređaja i gumb za slanje ispunjene forme same prijave. Prikaz programskog koda prikazan je na slici 16.

```
<Formik
  initialValues={initialValues}
  onSubmit={onSubmit.bind(this)}
>
{({ handleChange, handleSubmit, values, resetForm, validateForm }) => (
  <ScrollView
  keyboardShouldPersistTaps={"always"}
  >
    <SearchableDropdown
      onTextChange={text => console.log(text)} ···
    //To remove the underline from the android input
    />
    <TextInput
      style={[styles.textinputnaslov]}
      onChangeText={handleChange('naslov')}
      value={values.naslov}
      label="Unesite naslov"
      placeholder="npr. ležište komaraca"
    />
    <TextInput
      style={[styles.textinputopis]}
      multiline={true}
      //numberOfLines={4}
      onChangeText={handleChange('opis')}
      value={values.opis}
      label="Opišite problem"
      placeholder="npr. Ovo je mjesto zagađenja"
    />
    <TextInput
      style={[styles.textinputime]}
      onChangeText={handleChange('ime')}
      value={values.ime}
      label="Unesite ime"
      placeholder="npr. Pero"
    />
    <TextInput
      style={[styles.textinputprezime]}
      onChangeText={handleChange('prezime')}
      value={values.prezime}
      label="Unesite prezime"
      placeholder="npr. Perić"
    />
    <Button
      icon="add-a-photo" mode="contained" style={styles.button}
      onPress={() => { _pickImage(handleChange('image')), findCoordinates(handleChange('location')) }}
    >Odaberi sliku</Button>
    {values.image && values.image.length > 0 ?
      <Image source={{ uri: values.image }} style={styles.imagesubmit} /> : null}
    <Button onPress={handleSubmit} style={styles.button}><Text style={style = { color: '#fff' }}>Pošalji</Text></Button>
  </ScrollView>
)}
</Formik>
</ImageBackground>
</View>
```

Slika 16. Program code for the form on the home screen

Izvor: Autor

Prilikom pritiska na gumb za odabir sliku izvršavaju sve dvije funkcije, a to su:

- Funkcija za odabir slike iz mobilnog uređaja („*_pickImage*“)
- Funkcija za dohvaćanje lokacije iz mobilnog uređaja („*findCoordinates*“)

Funkcija za odabir slike izvršava se asinkrono[6] s ključnom riječi „*await“* te na početku traži odobrenje korisnika za pristup galeriji slika mobilnog uređaja. Nakon toga izvršava se također asinkrona funkcija „*launchImageLibraryAsync*“ koja omogućava korisniku pristup galeriji te odabir slike iz iste.

```
83
84  async function _pickImage(handleChange) {
85    await this.askPermissionsAsyncCamera();
86    let result = await ImagePicker.launchImageLibraryAsync({
87      allowsEditing: true,
88      aspect: [4, 3]
89    })
90    console.log(result)
91    if (!result.cancelled) {
92      handleChange(result.uri)
93    }
94  }
95
```

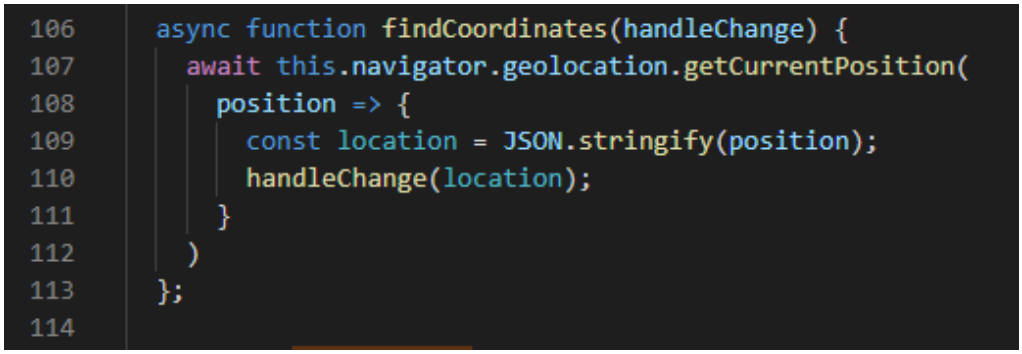
Slika 17. Function view "*_pickImage*"

Izvor: Autor

Funkcija za dohvaćanje lokacije mobilnog uređaja također se izvršava asinkrono te kao prvi upit traži pravo pristupa korisnika kako bi se omogućio dohvat lokacije. Sljedeće, izvršava se asinkrona funkcija za dohvat lokacije „*getCurrentPosition*“ koja uzima koordinate mobilnog uređaja te ih vraća u JSON[7] (JavaScript Object Notation) oblikovnom formatu.

[6] Understaing async-await in Javascript, 2018. Hackernoon - https://hackernoon.com/understanding-async-await-in-javascript-1d81bb079b2c

[7] What is JSON? A better format for dana exchange, 2019. InfoWorld - https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html



Slika 18. Function view "findCoordinates"

Izvor: Autor

Prilikom pritiska na zadnji gumb forme tj. gumb za slanje podataka, izvršava se glavna funkcija „*onSubmit*“, u koju se proslijede svi podaci iz forme. Na početku funkcije izvršava se dodatna funkcija „*validateForm*“ koja provjerava da li su sva polja ispunjena u formi. U slučaju da jedno od polja nije ispunjeno, program izađe iz funkcije te komunicira korisniku koje polje nije ispunio. Ukoliko validacija prođe uspješno, glavna funkcija se nastavlja te mapira podatke u format koji odgovara API-u prema kojem će biti proslijeđeni podaci. Ukoliko API iz nekog razloga nije dostupan, funkcija će komunicirati korisniku poruku s greškom, a u suprotnom komunicirati će se uspješna obavijest korisniku.

```
function onSubmit(values, { resetForm }) {
  let dataError = validateForm(values)
  if (dataError != 0) return;
  var index = [];

  // build the index
  for (var x in values) {
    index.push(x);
  }
  let localUri = values[index[0]];
  let lokacijajson = values[index[1]];
  let kategorija = values[index[2]];

  let filename = localUri.split('/').pop();
  let lokacija = JSON.parse(lokacijajson);

  var latitude = lokacija.coords.latitude;
  var longitude = lokacija.coords.longitude;
  // Infer the type of the image
  let match = /\.(\w+)$/.exec(filename);
  let type = match ? `image/${match[1]}` : `image`;

  // Upload the image using the fetch and FormData APIs
  let formData = new FormData();
  // Assume "photo" is the name of the form field the server expects
  formData.append('photo', { uri: localUri, name: filename, type: type });
  formData.append('naslov', values.naslov);
  formData.append('ime', values.ime);
  formData.append('latitude', latitude);
  formData.append('longitude', longitude);
  formData.append('prezime', values.prezime);
  formData.append('opis', values.opis);
  formData.append('kategorija', kategorija.name);
  fetch("http://4f9ff3f0bd29.ngrok.io", {
    method: 'POST',
    body: formData,
    header: {
      'content-type': 'multipart/form-data',
    },
  }).then(res => res.json())
    .catch(error => alert("Dogodila se greška"))
    .then(res => alert(res))
    .then(resetForm)
}
```

Slika 19. Function view "*onSubmit*"

Izvor: Autor

2.3.1.2 Implementacija i opis datoteke *MapScreen.js*

U ovom poglavlju opisati ćemo drugi zaslon tj. „*MapScreen.js*“, koji služi za pregled prijava na mapi. U ovoj datoteci nalazi se mapa, te pomoćne funkcije koje služe kako bi mapi dostavili potrebne podatke za prikaz prijava. Mapa se sastoji od obilježja (eng. marker) u raznim bojama s pripadajućim naslovom i lokacijom, a u desnom donjem kutu prikazana je legenda radi lakšeg razumijevanja značenja obilježja.

```
render() {
  return (
    <View style={styles.container}>
      <Header
        statusBarProps={{ barStyle: 'light-content' }}
        barStyle="light-content" // or directly
        centerComponent={{ text: 'Mapa prijava', style: { color: '#fff', fontStyle: 'normal', fontSize: 20 } }}
        containerStyle={{
          backgroundColor: '#32CD32',
          justifyContent: 'space-around',
        }}
      />
      <MapView style={styles.container}
        showsUserLocation={true}
        showsMyLocationButton={true}
        initialRegion={{
          latitude: 45.218818,
          longitude: 13.928048,
          latitudeDelta: 1,
          longitudeDelta: 1
        }}
      >
        {this.state.markers.map(marker => (
          <React.Fragment key={marker.id}>
            <MapView.Marker
              coordinate={marker.coordinates}
              title={marker.title}
              pinColor={marker.color}
            >
            </MapView.Marker>
          </React.Fragment>
        ))}
      </MapView>
        <Image source={require('../assets/images/legenda2.png')} style={styles.legend} />
    </View>
  );
  }
}
```

Slika 20. Program code for the map on the second screen

Izvor: Autor

Kako bi se podaci ispravno proslijedili u mapu postoje dvije funkcije:

- *componentDidMount i*
- *findCoordinates*

Funkcija „*componentDidMount*“ je funkcija koja dolazi s React Native jezikom. Ona služi tome da se izvrši tek nakon što se dogodilo potpuno učitavanje datoteke (eng. render), a izvršava se isključivo jednom. Funkcija „*findCoordinates*“ služi tome da dohvaća podatke s API-a s web aplikacije koji su u JSON formatu, a kasnije ih prosljeđuje mapi kako bi se ispravno ispunila obilježja. Funkcija za dohvat podataka nalazi se unutar funkcije „*componentDiDMount*“, iz razloga što želimo da se poziv prema API-u izvrši samo jednom, a da se podaci dostave mapi tek kada se mapa

učitala. Kako bi se podaci učitavali u realnom vremenu, na funkciji za dohvat podataka postavljen je interval od 100 000 milisekundi, nakon kojeg se funkcija u pozadini izvrši te preuzme nove podatke.

```
19  componentDidMount() {
20    this.findCoordinates().then(setInterval(() => {
21      this.findCoordinates();
22    }, 100000))
23  }
24
25  findCoordinates = async () => {
26    try {
27      let response = await fetch('http://4f9ff3f0bd29.ngrok.io/api/GetLocation/Location');
28      let responseJson = await response.json();
29      this.state.markers = responseJson;
30      console.log(this.state.markers);
31      this.setState({ markers: this.state.markers })
32    } catch (error) {
33      console.error(error);
34    }
35  }
36
```

Slika 21. View "componentDidMount" and "findCoordinates" functions

Izvor: Autor

2.3.2 Prikaz grafičkog dijela mobilne aplikacije

U ovom poglavlju prikazani su svi zasloni mobilne aplikacije sustava za dojavu problematike, provedbe mjera dezinfekcije, dezinsekcije i deratizacije (DDD) nadležnoj jedinici lokalne samouprave.

Slika 22. Form for reporting problems in the mobile application

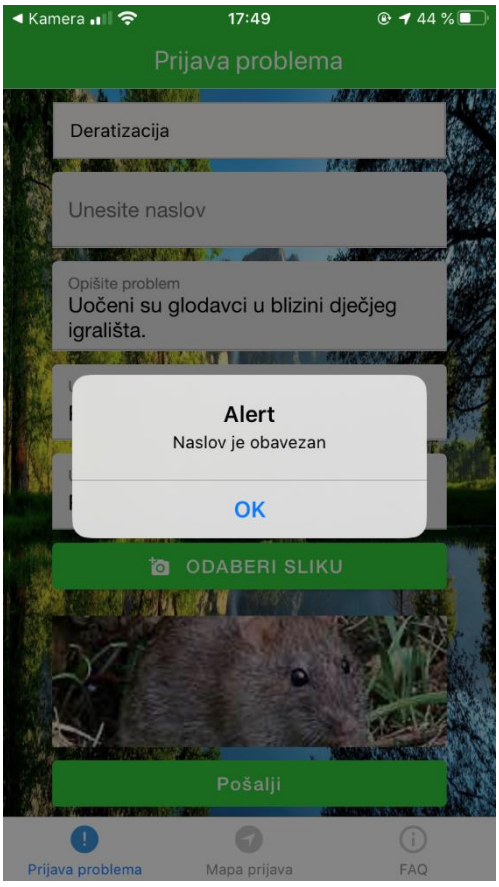
Izvor: Autor

Forma sadrži jedan padajući izbornik, niz polja za upisivanje podataka te dva gumba. U padajućem izborniku korisnik bira kategoriju problematike što želi prijaviti, a

trenutno su ponuđene mogućnosti prijave problematike koja se odnosi na: dezinfekciju, dezinsekciju i deratizaciju. U ostalim poljima korisnik ručno upisuje s tipkovnice tražene podatke, a to su:

- Naslov prijave,
- Opširniji opis problema,
- Ime i
- Prezime.

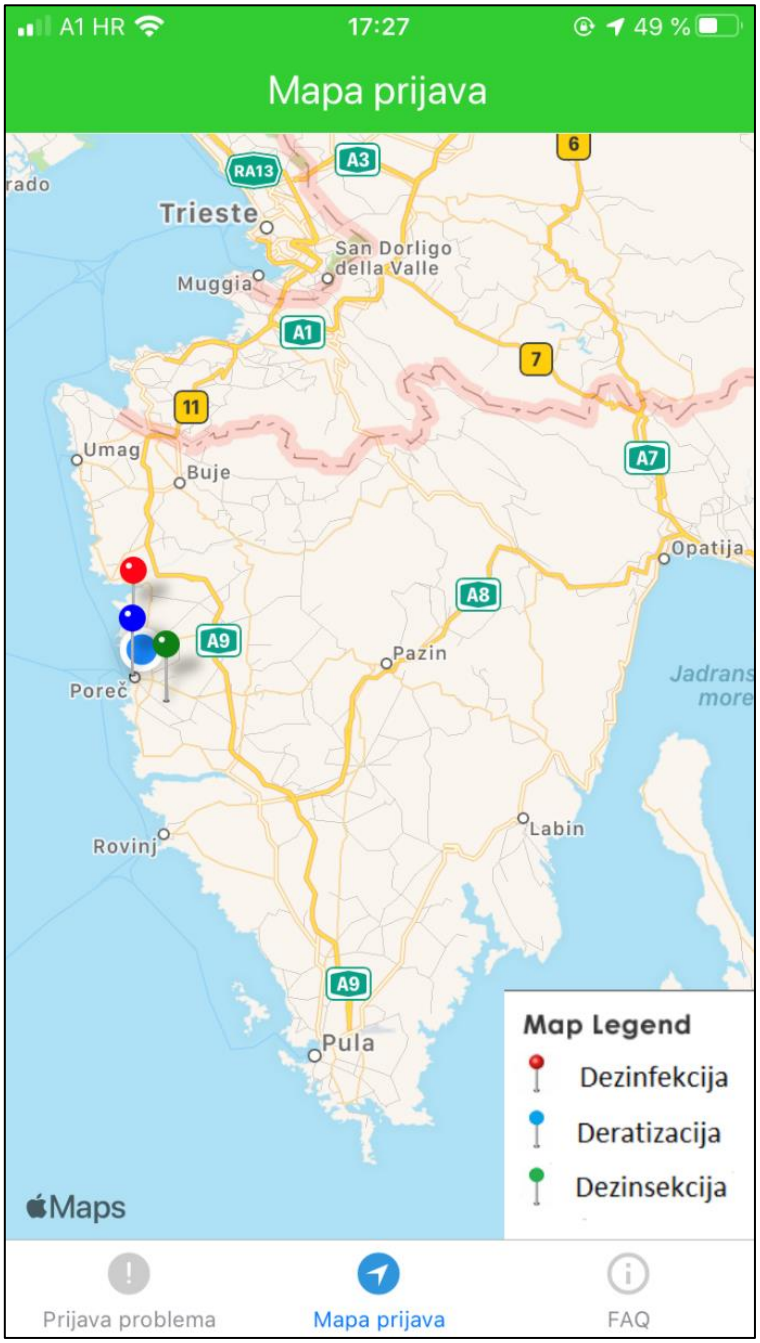
Nakon što je ispunio podatke, pritiskom na gumb „odaberi sliku“, korisnik iz vlastite galerije mobitela bira fotografiju koju želi poslati uz prijavu. Nakon što je odabrao fotografiju, korisnik pregledava podatke te gumbom „pošalji“ izvršava prijavu. U slučaju neispravnog unosa podataka korisniku se prikazuje povratna informacija kao što je prikazana na slici 23.



Slika 23. Error display in case of incorrect form filling

Izvor: Autor

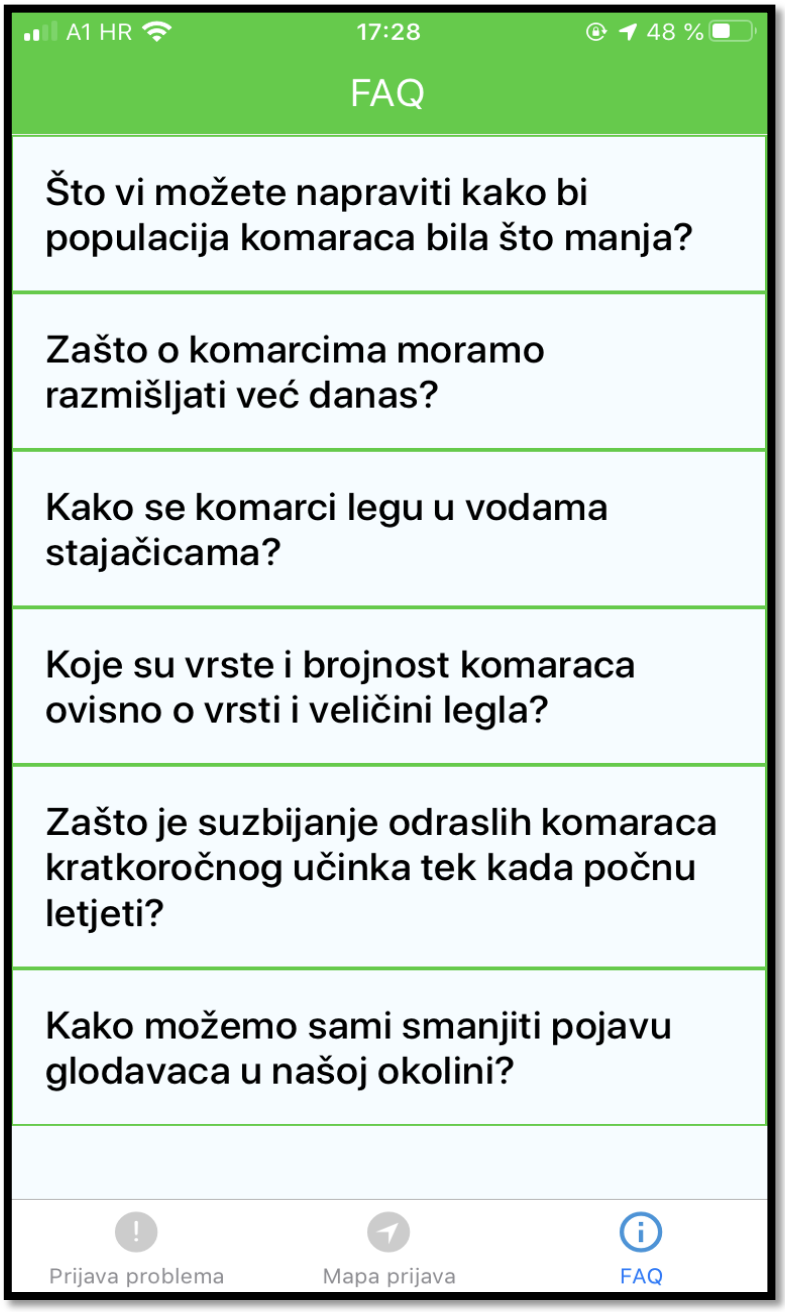
Na drugom zaslonu nalazimo mapu prijava. Mapa sadrži razna obilježja u bojama koja označavaju kategoriju prijave. Također, korisniku je vidljiva vlastita lokacija, što mu pomaže odrediti udaljenost od prijava. U desnom donjem kutu nalazi se legenda koja objašnjava pojedino obilježje. Ukoliko korisnik pritisne na jedno od obilježja, prikazati će se naslov prijave.



Slika 24. Screen for viewing reported problems on the map in the mobile application

Izvor: Autor

Na zadnjem zaslonu nalazimo sučelje za često postavljena pitanja. Pritiskom na jednu od sekcija imamo mogućnost proširenja (eng. expand) kako bi doznali odgovor na pitanje. Ukoliko pritisnemo na proširenu sekciju, ona će se jednostavno smanjiti (eng. collapse), odnosno, vratiti u početno stanje.



Slika 25. Screen for reviewing frequently asked questions

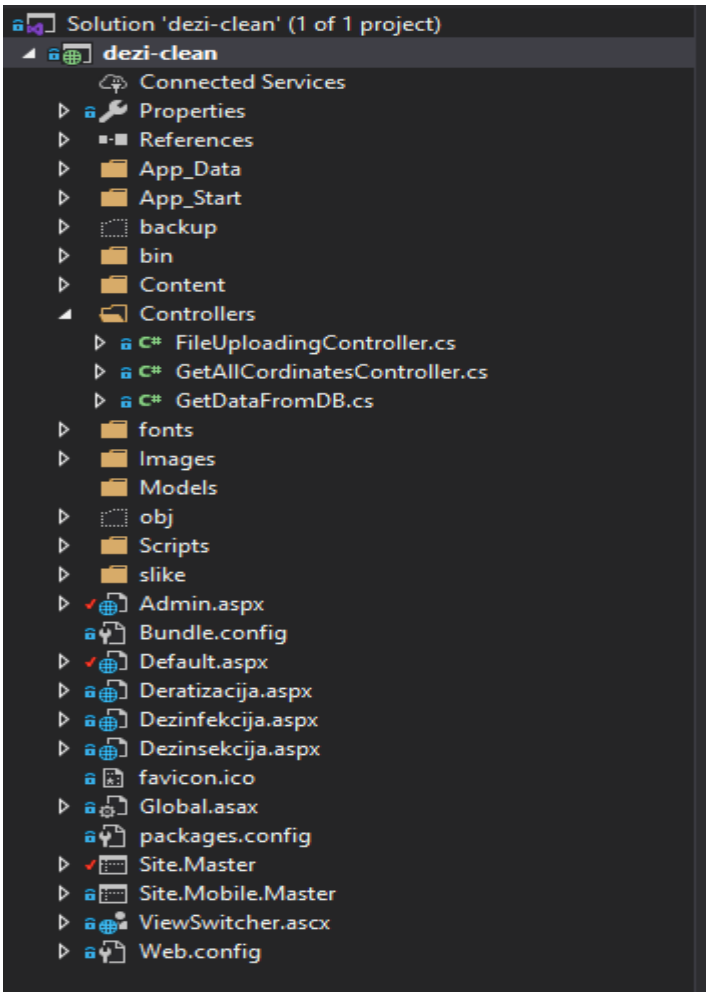
Izvor: Autor

2.4 Web aplikacija

Web aplikacija sustava za dojavu problematike, provedbe mjera dezinfekcije, dezinsekcije i deratizacije (DDD) nadležnoj jedinici lokalne samouprave izrađena je kroz Microsoftove tehnologije C# i ASP.NET Web forms koji su detaljnije objašnjeni na početku ovoga rada. Za spremanje i obradu podataka korišten je SQL Server koji je također detaljnije prikazan na početku rada.

2.4.1 Prikaz datotečnog i programskog dijela web aplikacije

Web aplikacija se sastoji od jednog projekta u kojem se nalazi niz mapa i datoteka koje osiguravaju ispravan rad aplikacije.



Slika 26. List of folders and files in a web application
Izvor: Autor

U projektu dezi-clean nalazi se mapa Controllers koja sadrži tri kontrolera8, od kojih dvoje služe za API pozive za mobilnu aplikaciju (*FileUploadingController* i *GetAllCordinatesController*), a preostali za dohvat informacija iz baze podataka. Sljedeće treba spomenuti ASP .NET stranice koje se nalaze u projektu, a to su:

- Default.aspx – početna stranica prilikom pristupanja web aplikaciji,
- Admin.aspx – stranica namijenjena isključivo za administratore,
- Deratizacija.aspx – stranica koja prikazuje podatke iz kategorije deratizacija,
- Dezinfekcija.aspx – stranica koja prikazuje podatke iz kategorije dezinfekcija,
- Dezinsekcija.aspx – stranica koja prikazuje podatke iz kategorije dezinsekcija.

Sve stranice iz projekta za dohvat podataka koriste kontroler „*GetDataFromDB*“, a koji će biti detaljnije prikazan u sljedećim poglavljima.

```
1   using dezi_clean.Controllers;
2   using System;
3   using System.Collections;
4   using System.Collections.Generic;
5   using System.Linq;
6   using System.Web;
7   using System.Web.UI;
8   using System.Web.UI.WebControls;
9
10  namespace dezi_clean
11  {
        2 references
12      public partial class Deratizacija : System.Web.UI.Page
13      {
            0 references
14          protected void Page_Load(object sender, EventArgs e)
15          {
16              ArrayList values = new ArrayList();
17              values = GetDataFromDB.GetDataForBind("='deratizacija'");
18
19
20              myCustomRepeater.DataSource = values;
21              myCustomRepeater.DataBind();
22          }
23      }
24  }
```

Slika 27. Example of backend ASP .NET data display page
Izvor: Autor

[8] Planning an ASP.NET Web Forms Application: Controllers and Testing, 2017. Telerik - https://www.telerik.com/blogs/planning-aspnet-web-forms-application-controllers-and-testing

Na slici 28 prikazan „backend“ dio ASP .NET stranica koji dohvaća i priprema podatke koje će kasnije proslijediti „frontend“ dijelu. Na početku funkcije „*Page_Load*“ stvara se prazna lista koju će popuniti informacijama kontroler za dohvat podataka. Nakon što uspješno napuni podatke u listu, podaci se moraju povezati (eng. bind9) za „frontend“ dio stranice, a konkretno se vežu za ponavljač (eng. repeater10). Repetaer je komponenta ASP .NET okvira koja iterira po listi podataka.

```
<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <!-- CONTENT -->
    <h1 class="recent-post-title1">Recent Posts</h1>
    <asp:Repeater runat="server" ID="myCustomRepeater">
        <ItemTemplate>
            <div class="row">
                <div class="content clearfix">
                    <div class="main-content">



                        <div class="post">
                            <img src="<%# DataBinder.Eval(Container.DataItem, "ImagePath") %>" alt="" class="post-image" />
                            <div class="post-preview">
                                <h2><%# DataBinder.Eval(Container.DataItem, "Title") %></h2>
                              <i class="fa fa-user" style="color: black">
                                  <%# DataBinder.Eval(Container.DataItem,"Name")%>
                                  <%#DataBinder.Eval(Container.DataItem, "Lastname") %></i>
                                 
                        <i class="fa fa-calendar"><%# DataBinder.Eval(Container.DataItem, "Date") %></i>
                                <p class="preview-text"><%# DataBinder.Eval(Container.DataItem, "Problemdescription") %></p>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </ItemTemplate>
    </asp:Repeater>
    <!-- CONTENTEND -->
</asp:Content>
```

Slika 28. Example of "frontend" ASP .NET data display page
Izvor: Autor

[9] Data binding overview in WPF, 2019. Microsoft - https://docs.microsoft.com/en-us/dotnet/desktop-wpf/data/data-binding-overview

[10] ASP.Net Repeater Control Using C#, 2015. C# Corner - https://www.c-sharpcorner.com/UploadFile/ca2535/Asp-Net-repeater-control-using-C-Sharp/

Od važnih datoteka treba također spomenuti glavne stranice tj. Site.Master i Site.Mobile.Master koje služe za stvaranje izgleda i ponašanje za sve stranice (ili grupu stranica) u kompletnoj aplikaciji. Glavna stranica pruža predložak za ostale stranice, s zajedničkim izgledom i funkcijama, a rezultat je kombinacija glavne stranice i stranica s sadržajem (sve stranice koje smo naveli na početku poglavlja). Zadnja datoteka koju treba spomenu je konfiguracijska datoteka web.config koja služi za upravljanje različitim postavkama kompletne aplikacije. Na taj način možemo konfigurirati postavke nezavisno od ostatka koda aplikacije. Specifično u ovoj aplikaciji web.config datoteka je vrlo bitna jer u nju spremamo postavke za spajanje na bazu podataka.

2.4.1.1 FileUploadingController

FileUploadingController je zapravo Web API Controller koji obrađuje dolazne HTTP zahtjeve te šalje natrag odgovor pozivatelju. Na početku funkcije kontroler „čeka“ dolazni HTTP zahtjev, a što u našem slučaju dolazi iz mobilnog djela aplikacije. Nakon što kontroler primi podatak, treba ga razdvojiti i prilagoditi formatu za upis u bazu podataka. Prvi korak jest spremiti relativnu putanju slike u određenu varijablu (eng. relative path[11]), a zatim pohraniti sliku na tvrdi disk računala. Ostali podaci, također se spremaju u zasebne varijable, a to su:

- Naslov
- Ime
- Prezime
- Opis problema
- Longituda
- Latituda
- Datum
- Kategorija
- Boja

[11] Absolute vs Relative Path - Which Should You Be Using?, 2017. Keycdn - https://www.keycdn.com/blog/relative-path

- Aktivan

Na kraju funkcije otvara se konekcija prema bazi podataka te se ubacuju podaci koje smo prethodno spremili u varijable.

```
[HttpPost]
[Route("api/FileUploading/UploadFile")]
0 references
public async Task<string> UploadFile()
{
    var ctx = HttpContext.Current;
    var root = ctx.Server.MapPath("~/slike");
    var relativePath = root.Substring(root.Length - 6);
    var provider = new MultipartFormDataStreamProvider(root);
    try
    {
        await Request.Content.ReadAsMultipartAsync(provider);
        string connStr = ConfigurationManager.ConnectionStrings["myConnectionString"].ConnectionString;
        SqlConnection con = new SqlConnection(connStr);
        con.Open();
        {
            foreach (var file in provider.FileData)
            {
                var name = file.Headers.ContentDisposition.FileName;
                //Remove dobule quotes from string
                name = name.Trim('"');
                var localFileName = file.LocalFileName;
                var filePath = Path.Combine(root, name);
                var relativePathInsert = Path.Combine(relativePath, name);
                File.Move(localFileName, filePath);

                var naslov = provider.FormData.GetValues("naslov").FirstOrDefault();
                var namee = provider.FormData.GetValues("ime").FirstOrDefault();
                var lastname = provider.FormData.GetValues("prezime").FirstOrDefault();
                var opisproblema = provider.FormData.GetValues("opis").FirstOrDefault();
                var latitude = provider.FormData.GetValues("latitude").FirstOrDefault();
                var longitude = provider.FormData.GetValues("longitude").FirstOrDefault();
                var datum = DateTime.Now.ToString("d/M/yyyy");
                var kategorija = provider.FormData.GetValues("kategorija").FirstOrDefault().ToLower();
                var boja = defineColor(kategorija);
                var aktivan = "true";

                SqlCommand cmd = con.CreateCommand();
                cmd.CommandText = "INSERT INTO [dezi-me].dbo.Data VALUES( @title,@name, @lastname,@problemdescription,@latitude, @longitude, @imagepath,@date,@category,@color,@aktivan)";
                cmd.Parameters.AddWithValue("@title",naslov);
                cmd.Parameters.AddWithValue("@name", namee);
                cmd.Parameters.AddWithValue("@lastname", lastname);
                cmd.Parameters.AddWithValue("@problemdescription", opisproblema);
                cmd.Parameters.AddWithValue("@latitude", latitude);
                cmd.Parameters.AddWithValue("@longitude", longitude);
                cmd.Parameters.AddWithValue("@imagepath", relativePathInsert);
                cmd.Parameters.AddWithValue("@date", datum);
                cmd.Parameters.AddWithValue("@category", kategorija);
                cmd.Parameters.AddWithValue("@color", boja);
                cmd.Parameters.AddWithValue("@aktivan", aktivan);
                cmd.ExecuteNonQuery();
                con.Close();
            }

        }
    }
    catch (Exception e)
    {
        return $"Error: {e.Message}";
    }
    return "Uspješno ste prijavili problem!";
```

Slika 29. Example implementation of Web API Controller - *FileUploadingController.cs*
Izvor: Autor

2.4.1.2 GetAllCordinatesController

GetAllCordinatesController je zapravo Web API Controller koji eksponira podatke kroz HTTP protokol, a sve u svrhu kako bi ostale aplikacije mogle pristupiti navedenim podacima. Metoda „*CreateJSONData*“ na početku stvara praznu listu koju kasnije popuni funkcija „*FillData*“, a koja će biti u nastavku opisana. Nakon što funkcija vrati podatke, isti se pretvaraju u JSON oblik te eksponiraju prema ostalim aplikacijama.

```
        [HttpGet]
        [Route("api/GetLocation/Location")]
        0 references
        public HttpResponseMessage CreateJSONData()
        {
            List<Marker> data = new List<Marker>();
            data = FillData();
            var jsonmsg = JsonConvert.SerializeObject(data);
            var res = Request.CreateResponse(HttpStatusCode.OK);
            res.Content = new StringContent(jsonmsg, System.Text.Encoding.UTF8, "application/json");
            return res;
        }
    }
}
```

Slika 30. Example of the CreateJSONData function for data exposure

Izvor: Autor

Funkcija „*FillData*“ na početku stvara praznu listu. Sljedeći korak jest otvaranje konekcije prema bazi podataka te mapiranje dohvaćenih podataka u zasebne varijable. Na kraju funkcije podaci se dodaju u listu koja je stvorena na samom početku funkcije te potom vraćaju kao izlazni rezultat.

```
public class GetAllCordinatesController : ApiController
{
    //Student[] students = new Student[]
    List<Marker> AllMarkers = new List<Marker>();
    1 reference
    public List<Marker> FillData()
    {
        string connStr = ConfigurationManager.ConnectionStrings["myConnectionString"].ConnectionString + "MultipleActiveResultSets=true";
        using (SqlConnection connection = new SqlConnection(connStr))
        using (SqlCommand command = new SqlCommand("SELECT TOP (1000) [id],[title],[latitude],[longitude],[aktivan], [color] FROM [dezi-me].[dbo].[Data] where aktivan = 'true'", connection))
        {
            connection.Open();
            using (SqlDataReader reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    int id;
                    var title = "";
                    string longitude = "";
                    string latitude = "";
                    string color = "";
                    //Marker seba = new Marker();
                    id = (int)reader["id"];
                    title = reader["title"].ToString();
                    longitude = reader["longitude"].ToString();
                    latitude = reader["latitude"].ToString();
                    color = reader["color"].ToString();
                    var c = System.Threading.Thread.CurrentThread.CurrentCulture;
                    var s = c.NumberFormat.CurrencyDecimalSeparator;

                    longitude = longitude.Replace(",", s);
                    longitude = longitude.Replace(".", s);
                    latitude = latitude.Replace(",", s);
                    latitude = latitude.Replace(".", s);

                    decimal longitude_pravi = Convert.ToDecimal(longitude);
                    decimal latitude_pravi = Convert.ToDecimal(latitude);

                    AllMarkers.Add(new Marker { id = id, title = title, color = color, coordinates = new Coordinates() { longitude = longitude_pravi, latitude = latitude_pravi } });

                }
            }
            connection.Close();
        }
        return AllMarkers;
    }
```

Slika 31. Example FillData function for retrieving and processing data

Izvor: Autor

2.4.1.3 GetDataFromDB

Kontroler *GetDataFromDB* služi za dohvat podataka pojedinih stranica u projektu kao što smo spomenuli na početku poglavlja. Kontroler sadrži funkciju *GetDataForBind* koja vraća listu, a prima kategoriju kao ulaznu varijablu. Ulazna varijabla kategorija dinamički se zamjenjuje s varijablom koja izvršava upit prema bazi. Varijabla se zamjenjuje iz razloga što želimo zatražiti samo podatke koji su npr. iz kategorije za dezinfekciju. Podatke koji dolaze iz baze podataka zamjenjujemo varijablama te na kraju vraćamo listu podataka kao što je prikazano na slici 32.

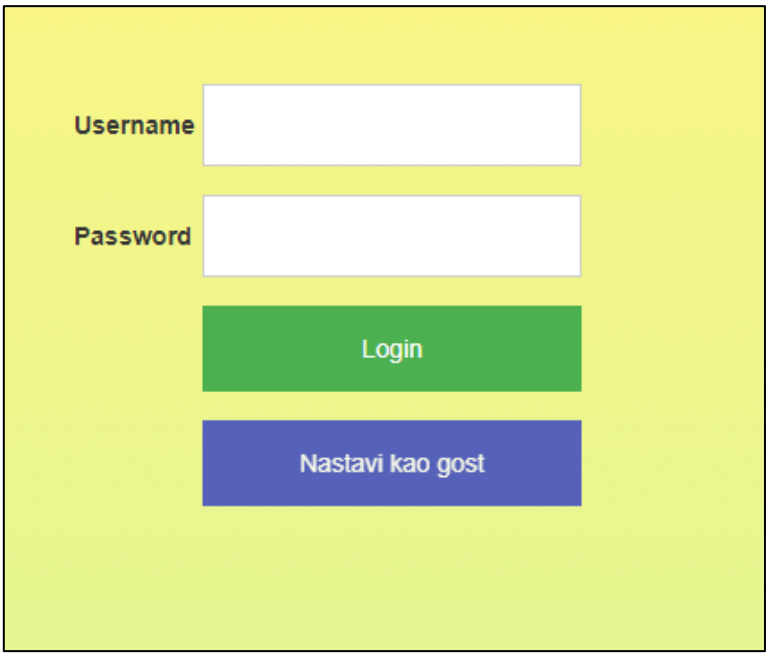
```
38      public static ArrayList GetDataForBind(string kategorija)
39      {
40          ArrayList values = new ArrayList();
41          string sqlquery = String.Format("SELECT TOP (1000) [id],[title],[name],[lastname],[problemdescription],[latitude],[longitude],[imagepath],[date],[category],[color],[aktivan] " +
42              "FROM [dezi-me].[dbo].[Data]" +
43              " where aktivan = 'true' and category {0} order by id desc", kategorija);
44          string connStr = ConfigurationManager.ConnectionStrings["myConnectionString"].ConnectionString + "MultipleActiveResultSets=true";
45          using (SqlConnection connection = new SqlConnection(connStr))
46          using (SqlCommand command = new SqlCommand(sqlquery, connection))
47          {
48              connection.Open();
49              using (SqlDataReader reader = command.ExecuteReader())
50              {
51
52                  while (reader.Read())
53                  {
54                      var id = reader["id"].ToString();
55                      var title = reader["title"].ToString();
56                      var name = reader["name"].ToString();
57                      var lastname = reader["lastname"].ToString();
58                      var problemdescription = reader["problemdescription"].ToString();
59                      var latitude = reader["latitude"].ToString();
60                      var longitude = reader["longitude"].ToString();
61                      var imagepath = reader["imagepath"].ToString();
62                      var date = reader["date"].ToString();
63                      values.Add(new Data(id, title, name, lastname, problemdescription, longitude, latitude, imagepath, date));
64                  }
65              }
66              connection.Close();
67          }
68
69          return values;
70      }
```

Slika 32. Example *GetDataForBind* function for retrieving information from database

Izvor: Autor

2.4.2 Prikaz grafičkog dijela desktop aplikacije

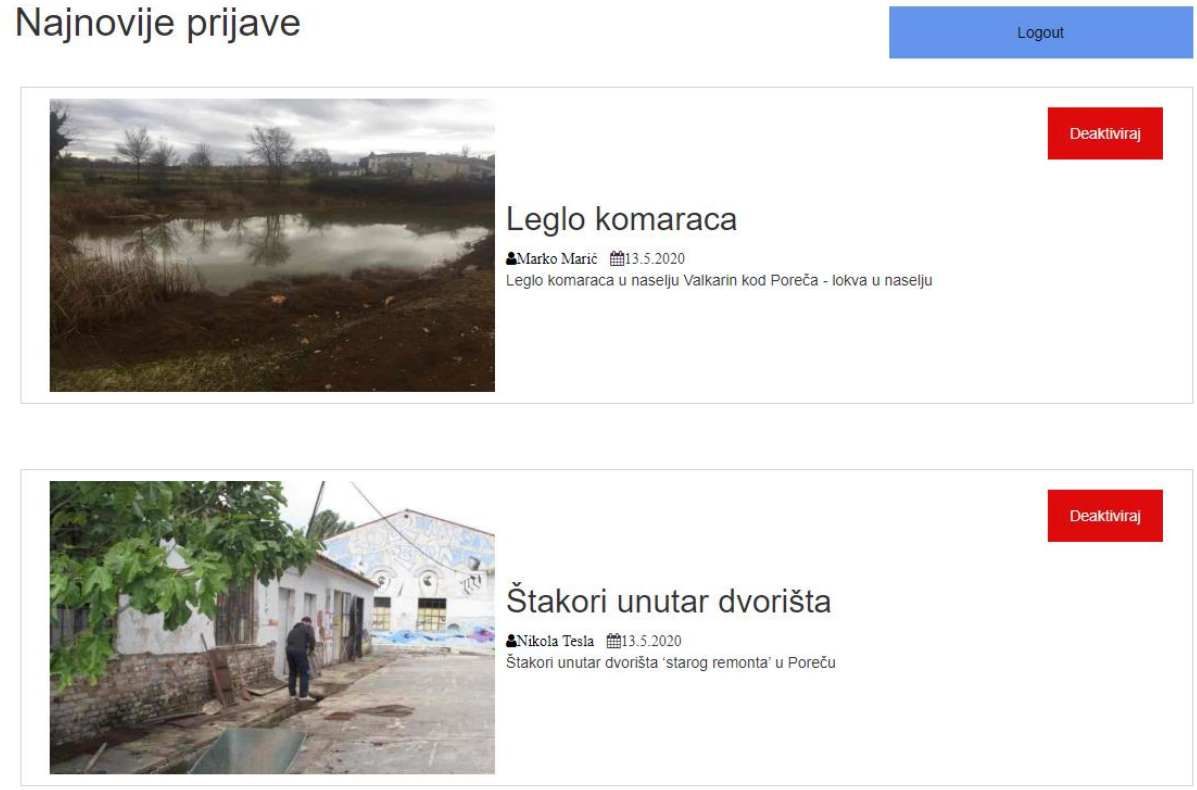
U ovom pod poglavlju prikazano je sučelje web aplikacije sustava za dojavu problematike, provedbe mjera dezinfekcije, dezinsekcije i deratizacije nadležnoj jedinici lokalne samouprave. Na slici 33 prikazana je forma za prijavu u web aplikaciju sustava, a također možemo se prijaviti kao običan gost.



Slika 33. View problem report into the web application

Izvor: Autor

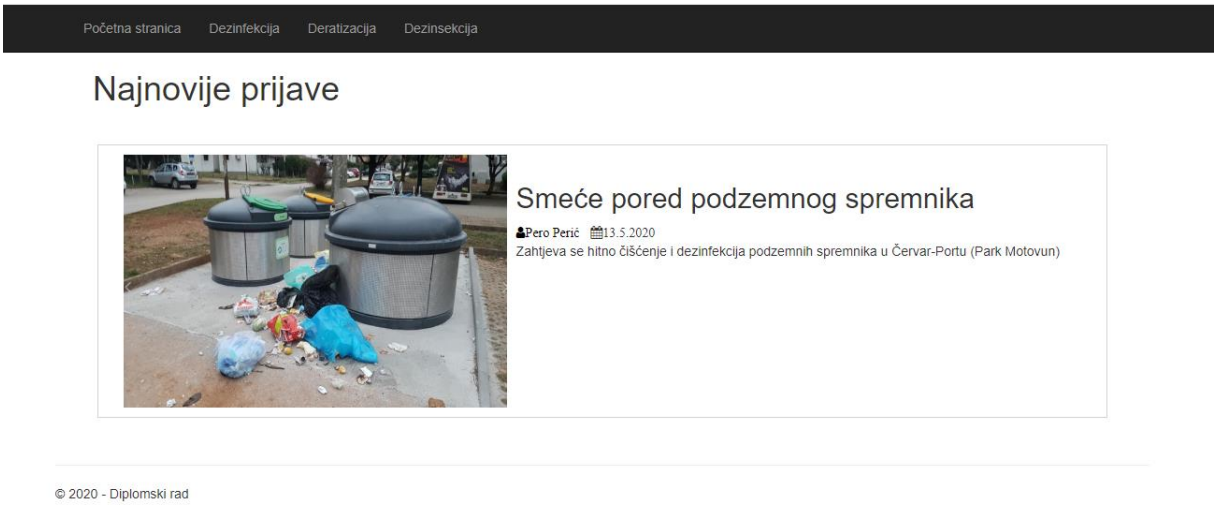
U slučaju da se želimo prijaviti kao administrator, potrebno je unijeti ispravne podatke te stisnuti na zeleni gumb „Login“. Nakon uspješne prijave dolazimo na sučelje za administratora gdje se prikazuju sve prijave korisnika. Prijave administrator može deaktivirati pritiskom gumb „deaktiviraj“, a po završetku zahvata može se odjaviti pritiskom na gumb „logout“ koji ga vraća na sučelje za prijavu.



Slika 34. Interface display for administrator

Izvor: Autor

Ukoliko želimo pristupiti aplikaciji kao običan korisnik, moramo odabrati gumb „Nastavi kao gost“. Gumb nas vodi na prvu kategoriju u izborniku tj. stranicu za pregled prijava dezinfekcije. Kod običnih korisnika svaka od kategorija ima svoju stranicu, što olakšava pregled samih prijava, a njihov međusobni pristup odvija se kroz navigaciju. Također postoji poveznica koja vraća korisnika na početnu stranicu prijave.



Slika 35. Regular user interface display

Izvor: Autor

ZAKLJUČAK

Cilj ovog diplomskog rada je izrada sustava za dojavu problematike, prilikom provedbe mjera preventivne dezinfekcije, dezinsekcije i deratizacije (DDD) nadležnoj jedinici lokalno samouprave. Ideja same aplikacije je zapravo digitalizirati proces koji se događa u lokalnim jedinicama samouprave, a također podignuti svijest lokalnih građana o očuvanju okoliša.

Kompletan sustav sastoji se od mobilne i web aplikacije. Većina mobilne aplikacije izrađena je u React Native programskom jeziku, dok je web aplikacija izrađena u Microsoftovim tehnologijama C# i ASP.NET Web Forms.

Funkcionalnosti mobilnog djela aplikacije sustava za dojavu problematike sastoje se od forme za prijavu uočene nepravilnosti, pregled svih prijava na mapi te pregled sučelja za često postavljena pitanja.

Mobilni dio aplikacije ovog sustava ima puno prostora za poboljšanje i dodavanje novih funkcionalnosti kao što su lista prijavljenih problema po pojedinom korisniku, uz mapu prijava složiti listu svih prijavljenih problema, mogućnost praćenja problema, kako bi korisnik znao u kojoj je fazi rješavanja, općeniti status prijave te povratna notifikacija korisniku kada se prijava obradi.

Funkcionalnosti web aplikacije jesu pregled svih prijava po kategorijama (DDD) za običnog korisnika, dok za administratora postoji posebno sučelje uz prethodnu prijavu gdje je omogućena funkcija deaktiviranja samih prijava.

Trenutno na tržištu ne postoji ovakva vrsta aplikacije, ali postoje jako slične aplikacije po strukturi, ali različitoj namjeni. Kako smo već spomenuli da sustav ima puno prostora za poboljšanje, neophodno je dalje razvijati nove funkcionalnosti i poboljšanja, a sve u svrhu opstanka na tržištu, naravno, sve u dogovoru s naručiteljem.

LITERATURA

Knjige i radovi:

Ben, A., Joseph, A. (2017), C# 7.0 in a Nutshell, Sebastopol: O'Reilly

Marijn, H. (2018) Eloquent Javascript – 3rd edition

Internet stranice:

React Native Pros and Cons - Facebook's Framework in 2019 - https://www.netguru.com/blog/react-native-pros-and-cons (Pristupljeno: 15. svibnja 2020)

Understanding Expo for React Native, 2018. Hackernoon - https://hackernoon.com/understanding-expo-for-react-native-7bf23054bbcd (Pristupljeno: 16. svibnja 2020)

What is npm?, 2011. Node.js - https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/ (Pristupljeno: 26. svibnja 2020)

Understaing async-await in Javascript, 2018. Hackernoon - https://hackernoon.com/understanding-async-await-in-javascript-1d81bb079b2c (Pristupljeno: 29. svibnja 2020)

What is JSON? A better format for dana exchange, 2019. InfoWorld - https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html (Pristupljeno: 01. lipnja 2020)

Planning an ASP.NET Web Forms Application: Controllers and Testing, 2017. Telerik - https://www.telerik.com/blogs/planning-aspnet-web-forms-application-controllers-and-testing (Pristupljeno: 04. lipnja 2020)

Data binding overview in WPF, 2019. Microsoft - https://docs.microsoft.com/en-us/dotnet/desktop-wpf/data/data-binding-overview (Pristupljeno: 10. lipnja 2020)

ASP.Net Repeater Control Using C#, 2015. C# Corner - https://www.c-sharpcorner.com/UploadFile/ca2535/Asp-Net-repeater-control-using-C-Sharp/ (Pristupljeno: 14. lipnja 2020)

Absolute vs Relative Path - Which Should You Be Using?, 2017. Keycdn - https://www.keycdn.com/blog/relative-path (Pristupljeno: 18. lipnja 2020)

Popis slika

Popis tablica

Sažetak

Cilj ovog rada je prikazati te zorno objasniti na koji način je moguće koristiti sustav za dojavu problematike, provedbe mjera preventivne dezinfekcije, dezinsekcije i deratizacije (DDD) nadležnoj jedinici lokalno samouprave. Kroz rad su predočene same tehnologije u kojima je isti izrađen. Kroz razna poglavlja objašnjena je kompletna arhitektura aplikacije, a također raznim dijagramima prikazana je sama interakcija korisnika i same aplikacije. Na kraju samog rada prikazani su dijelovi programskog koda te sučelja mobilne i web aplikacije.

Ključne riječi: mobilna aplikacija, web aplikacija, React Native, Expo, ASP.NET Web Forms, Visual Studio Code, Microsoft Visual Studio, Microsoft SQL, Microsoft Managment Studio 2019

Abstract

The aim of this paper is to present and clearly explain how it is possible to use problem-processing system, proven measures for disinfection, disinsection and deratization (DDD) of the competent local self-government units. Through the paper, the technologies in which it is made are presented. Through various chapters, the complete architecture of the application is explained, and also various diagrams show the interaction between the user and the application itself. At the end of the paper, parts of the program code and the interfaces of the mobile and web application are shown.

Keywords: mobile application, web application, React Native, Expo, ASP .NET Web Forms, Visual Studio Code, Microsoft Visual Studio, Microsoft SQL, Microsoft Managment Studio 2019