# Uloga paradigme REST u razvoju mrežnih i mobilnih aplikacija

**Pereša, Marino**

**Master's thesis / Diplomski rad**

**2019**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Pula / Sveučilište Jurja Dobrile u Puli**

*Permanent link / Trajna poveznica:* https://urn.nsk.hr/urn:nbn:hr:137:676672

*Rights / Prava:* In copyright / Zaštićeno autorskim pravom.

*Download date / Datum preuzimanja:* **2024-05-16**

*Repository / Repozitorij:*

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

**MARINO PEREŠA**

# ULOGA PARADIGME REST U RAZVOJU MREŽNIH I MOBILNIH APLIKACIJA

Diplomski rad

Pula, 2019. godine.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

**MARINO PEREŠA**

# ULOGA PARADIGME REST U RAZVOJU MREŽNIH I MOBILNIH APLIKACIJA

Diplomski rad

**JMBAG: 0303038155, redoviti student**
**Studijski smjer: Diplomski sveučilišni studij informatike**

**Predmet:** **Izrada informatičkih projekata**
**Znanstveno područje:** **Društvene znanosti**
**Znanstveno polje:** **Informacijske i komunikacijske znanosti**

**Mentor: doc. dr. sc. Siniša Sovilj**
**Komentor: dr. sc. Nikola Tanković**

Pula, 04. rujna 2019. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Marino Pereša, 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

_____

U Puli, 04. rujna 2019. godine

IZJAVA
o korištenju autorskog djela

Ja, Marino Pereša dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Uloga paradigme REST u razvoju mrežnih i mobilnih aplikacija“ 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 sa 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.

U Puli, 04. rujna 2019. godine

Potpis

_____

SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE U PULI

Pula, 1. ožujka 2017.

# DIPLOMSKI ZADATAK

| | |
|---|---|
| Pristupnik: | **Marino Pereša (0303038155)** |
| Studij: | Sveučilišni diplomski studij Informatike |
| Naslov (hrv.): | **Uloga paradigme REST u razvoju mrežnih i mobilnih aplikacija** |
| Naslov (eng.): | The role of REST paradigm in the development of network and mobile applications |
| Opis zadatka: | Zadatak je realizirati sustav koji će ubrzati postupak prijama robe u skladištima. Proučiti REST, njegove prednosti i ograničenja. Napraviti web i mobilnu aplikaciju i povezati sustav REST-om. |

Zadatak uručen pristupniku: 1. ožujka 2017.
Rok za predaju rada: 1. rujna 2019.

Mentor:

_____
doc.dr.sc. Siniša Sovilj

# Sadržaj


UVOD ..... 1
1. PREGLED TEHNOLOGIJA ..... 2
1.1. HTML ..... 2
1.2. PHP ..... 3
1.3. JavaScript ..... 4
1.4. MySQL ..... 5
1.5. React ..... 6
1.6. Laravel ..... 8
1.7. Swift ..... 11
2. API ..... 13
2.1. Operacijski sustavi ..... 14
2.2. Remote API ..... 14
2.3. Web API ..... 15
2.4. Primjeri popularnih API-ja ..... 17
3. REST API ..... 19
3.1. Način rada RESTful API-ja ..... 20
3.1.1. REST vs SOAP ..... 21
3.1.2. Nepostojanje stanja ..... 22
3.2. Prednosti RESTful servisa ..... 22
3.3. RESTful API Arhitektonska ograničenja ..... 23
3.3.1. Uniformno sučelje ..... 23
3.3.2. Nepostojanje stanja ..... 24
3.3.3. Mogućnost predmemoriranja ..... 25
3.3.4. Arhitektura klijent – poslužitelj ..... 25
3.3.5. Slojni sustav ..... 25
3.3.6. Kod na zahtjev ..... 26
4. REST API smjernice ..... 27
4.1. Microsoft API smjernice ..... 27
4.1.1. Smjernice za klijenta ..... 27
4.1.2. Osnovne dosljednosti ..... 27
4.1.3. CORS ..... 28
4.1.4. Kolekcije ..... 30
4.1.5. Verzije ..... 30
4.1.6. Operacije s dugim trajanjem ..... 31

4.1.7. Provlačenje, kvote i limiti ..... 31
4.1.8. Nepodržani zahtjevi ..... 32
4.1.9. Smjernice za imenovanje ..... 32
5. RICHARDSONOV MODEL ZRELOSTI ..... 33
5.1. Razina 0 - The Swamp of POX ..... 33
5.2. Razina 1 - Resursi ..... 34
5.3. Razina 2 – HTTP metode ..... 34
5.4. Razina 3 – Hypermedia ..... 35
6. HATEOAS ..... 36
6.1. HATEOAS Implementacija ..... 37
6.1.1. RFC 5988 (web povezivanje) ..... 37
6.1.2. JSON Hypermedia API Language (HAL) ..... 38
7. OPĆENITO O RAPiD SUSTAVU ..... 39
8. FUNKCIONALNOSTI ..... 40
8.1. Poduzeće ..... 41
8.2. Zaposlenik ..... 44
9. IMPLEMENTACIJA ..... 48
9.1. Baza podataka ..... 48
9.2. Web aplikacija - Backend ..... 49
9.2.1. OAuth 1.0 vs OAuth 2.0 ..... 51
9.2.2. Dodatni paketi ..... 52
9.3. Web aplikacija - Frontend ..... 52
9.3.1. Dodatni paketi ..... 57
9.4. Mobilna aplikacija ..... 58
9.5. REST API ..... 59
10. KORISNIČKE UPUTE ..... 65
10.1. Web aplikacija ..... 65
10.2. Mobilna aplikacija ..... 74
ZAKLJUČAK ..... 81
LITERATURA ..... 82
POPIS SLIKA ..... 84
SAŽETAK ..... 84
SUMMARY ..... 87

# UVOD

Proizvodna i trgovačka poduzeća kao sastavni dio svog poslovanja moraju imati svoje skladište, gdje će skladištiti primljenu robu. U skladištima se roba čuva do trenutka distribucije robe.

Od prijema robe, skladištenja te konačno distribucije robe javljaju se razne situacije koje treba biti u mogućnosti riješiti. Na primjer, roba koja se skladišti može u vremenu koje provede u skladištu izgubiti na težini, oštetiti se ili pokvariti. Takve je situacije potrebno optimizirati, tj. ubrzati proces skladištenja kako bi se izbjegli problemi.

U ovom diplomskom radu pozornost je posvećena prijemu robe i načinu optimizacije postupka prijema robe, te kontroli skladištenja. Fokus je ubrzati postupak prijema robe, te nakon prijema brzo i jednostavno imati pristup količinama robe na zalihi u realnom vremenu.

Jedan od načina kako ubrzati sam proces zaprimanja robe je upotrebom ručnih skenera, dok će drugi način biti opisan kroz ovaj rad. Kako su u današnje vrijeme sve popularnije mobilne aplikacije i rješenja koja koriste mobilne aplikacije, sustav koji će biti opisan u ovom diplomskom radu koristi upravo mobilnu aplikaciju kao jedan od bitnijih značajki koja zamjenjuje ručne skenere te nudi dodatne mogućnosti.

Kroz ovaj rad opisat će se na koji način radi sustav, što se koristilo pri izradi sustava, te kako korisnici mogu koristiti taj sustav i sve mogućnosti koje on pruža.

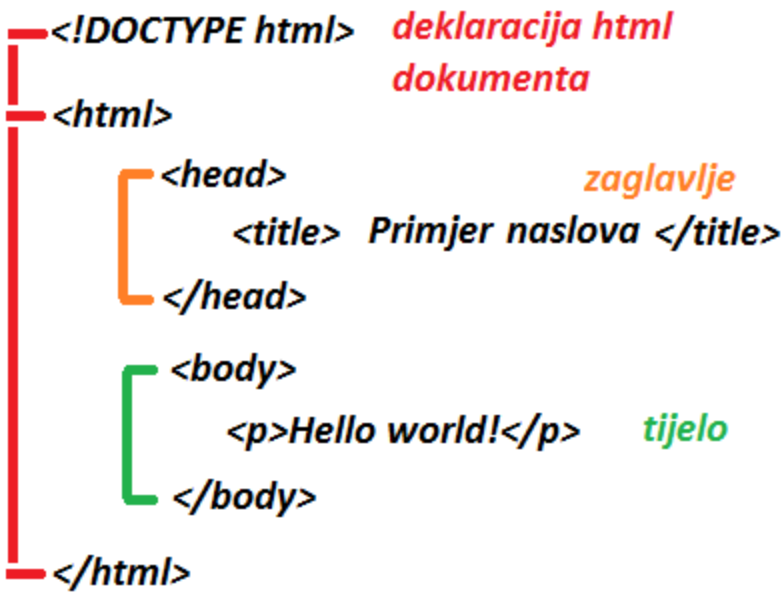
# 1. PREGLED TEHNOLOGIJA

Prilikom izrade projekta korišteno je više različitih razvojnih okvira, servisa, te programskih jezika. Većina jezika i razvojnih okvira imaju mnogo sličnih obilježja, zbog čega se i neki nepoznati jezik ili razvojni okvir koji nije prethodno korišten, može lakše savladati. Kroz sljedeće cjeline detaljnije će se opisati tehnologije korištene u razvoju projekta.

## 1.1. HTML

HTML[1] pruža način za stvaranje strukturiranih dokumenata označavanjem strukturne semantike za tekst kao što su naslovi, odlomci, popisi, linkovi i slično za web stranice i web aplikacije. Međutim HTML nije programski jezik, stoga nema mogućnost da kreira dinamičke funkcionalnosti. Umjesto toga s njim se može organizirati i formatirati dokumente. HTML je standardni *markup* jezik za dokumente dizajnirane za prikaz u web-pregledniku.

*Slika 1. Primjer HTML dokumenta*




Izvor: http://www.webtech.com.hr/html.php

[1] HTML – HyperText Markup Language

HTML elementi su sastavni dijelovi HTML stranica. S HTML konstrukcijama, slike i drugi objekti, poput interaktivnih oblika, mogu biti ugrađeni u prikazanu stranicu.

HTML je jezik za opisivanje strukture dokumenta, a ne njegovo stvarno predstavljanje. Ideja je da većina dokumenata ima zajedničke elemente - na primjer, naslove, odlomke i popise. Prije nego što se počne pisati, potrebno je prepoznati i definirati skup elemenata u tom dokumentu i na odgovarajući ih način imenovati. (Kyrnin, 2015)

Kako bi se HTML elementi stilizirali, koristi se CSS[2]. CSS opisuje kako bi se elementi trebali prikazati na ekranu, papiru ili drugim medijima. Dosljednom upotrebom CSS-a postiže se potpuna separacija prezentacije web dokumenta od njegovog sadržaja. (Holjevac, 2010)

## 1.2. PHP

PHP[3] je je *open-source* skriptni jezik koji se koristi kod razvoja dinamičkih interaktivnih web stranica ili web aplikacija. PHP skripte se mogu koristiti samo na serverima koji imaju instaliran PHP, š

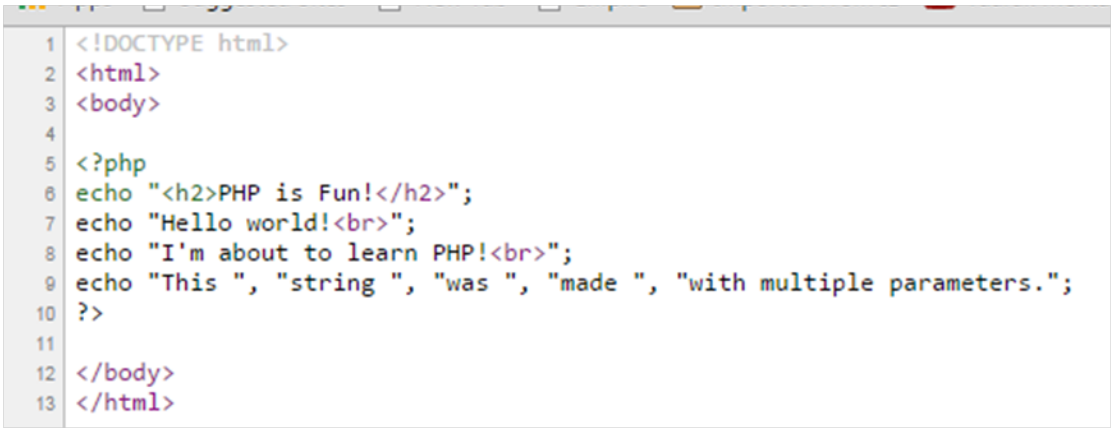
to nije veliki problem s obzirom da većina web hosting-a već ima unaprijed instaliran PHP. Računala klijenata koji pristupaju PHP skriptama moraju imati samo web preglednik da bi mogli pokrenuti skripte.

PHP je jedan od najzastupljenijih programskih jezika za programiranje web aplikacija. Njegova zajednica je velika i raznolika. Sastoji se od brojnih biblioteka, okvira i komponenti. Programeri često kombiniraju razne PHP biblioteke i komponente većinom sa nekim okvirom kako bi olakšali način programiranja.

Kada korisnik zatraži web stranicu koja sadrži PHP kod, taj kod se procesuira na udaljenom serveru koji generira HTML dokument. Nakon što se generira HTML dokument, prikazuje se na web pregledniku korisnika.

[2] CSS (Cascading Style Sheets) – koristi se za oblikovanje HTML elemenata
[3] PHP - Hypertext Pre-processor

*Slika 2. Primjer PHP koda ugrađen u HTML*



```
<!DOCTYPE html>
<html>
<body>

<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>

</body>
</html>
```

Izvor: https://i.stack.imgur.com/wUVPN.png

PHP se može koristiti na svim glavnim operativnim sustavima uključujući Linux, razne Unix varijante, Microsoft Windows, macOS i druge. Jedna od najznačajnijih značajki PHP-a je što podržava veliki raspon baza podataka. Stoga je izrada web stranica koje sadrže bazu podataka  vrlo jednostavna. Da bi se napravila interakcija između klijenta i servera, koriste se HTTP metode poput GET ili POST.

HTTP[4] je protokol dizajniran da omogući komunikaciju između klijenta i servera. Radi na način da klijent(web preglednik) pošalje HTTP zahtjev serveru, nakon čega server vraća odgovor klijentu. Odgovor sadrži informacije o statusu zahtjeva i može sadržavati zatraženi sadržaj. To može biti na primjer dohvaćanje informacija o korisniku iz baze podataka.

## 1.3. JavaScript

JavaScript je objektno orijentirani skriptni jezik kojemu je glavna svrha da poveća interakciju korisnika sa web stranicom. JavaScript se također koristi pri razvoju igara i mobilnih aplikacija.

[4] HTTP – Hypertext Transfer Protocol

Postoje mnogi koji misle da su JavaScript i Java ista stvar. Međutim JavaScript i Java su jako različiti. Sličnost je samo u imenu. JavaScript je nastao u vrijeme kada je Java bio popularni jezik, a izumitelji JavaScripta iskoristili su tu popularnost koristeći ime Java. Oba jezika posuđuju neku sintaksu iz programskih jezika poput C, ali osim toga, oni su prilično različiti. (Robson, 2014.)

Java je kompliciran programski jezik, dok je JavaScript samo skriptni jezik. Zbog toga što je skriptni jezik, ne može se samostalno izvoditi. Web preglednik je odgovoran za izvođenje JavaScript koda. Kada korisnik zatraži HTML stranicu koja sadrži JavaScript, skripta se šalje pregledniku i o njemu ovisi hoće li se skripta pokrenuti. Glavna prednost JavaScripta je ta što ga podržavaju svi moderni web preglednici. Glavne prednosti JavaScripta su:

- manje interakcije sa serverom – može se procijeniti input korisnika prije slanja na server. Ovo smanjuje promet na serveru, što znači manje učitavanja na serveru
- trenutne povratne informacije posjetitelju web stranice – ne treba se čekati da se stranica osvježi da se vidi je li se zaboravilo nešto unijeti
- povećana interaktivnost – može se kreirati sučelje koje reagira kada korisnik prođe mišem preko ili aktivira nešto s tipkovnicom
- bogatije sučelje – može se koristiti JavaScript koji uključuje slajdere ili „drag-and-drop“ komponente kako bi se napravilo bogatije sučelje za posjetitelje web stranice

JavaScript jezik razlikuje velika i mala slova. To znači da jezične ključne riječi, varijable, imena funkcija i drugi identifikatori moraju uvijek biti upisani s dosljednim velikim slovima. Na primjer, ključna riječ mora biti upisana while, a ne While ili WHILE. Slično tome, online, Online, OnLine i ONLINE su četiri različita imena varijabli. (Flanagan, 2011.)

## 1.4. MySQL

MySQL je *open-source* relacijski sustav za upravljanje bazom podataka. Baziran je na klijent-poslužitelj modelu. Jezgra MySQL-a je MySQL poslužitelj, koji obrađuje

sve upute (ili naredbe) baze podataka. MySQL poslužitelj je dostupan kao zaseban program za korištenje u umreženom klijent-poslužitelj okruženju ili kao biblioteka koja se može ugraditi (ili povezati) u zasebne aplikacije.

MySQL je izvorno razvijen za brzo upravljanje velikim bazama podataka. Iako je obično instaliran na samo jednom računalu, ima mogućnost slanja baze podataka na više različitih lokacija, jer korisnici mogu pristupiti putem različitih MySQL klijentskih sučelja. MySQL omogućuje pohranjivanje podataka i pristupanje kodovima na više sustava za pohranu, uključujući InnoDB, CVS i NDB.

MySQL je napisan u C i C++ te je dostupan na preko dvadeset platformi, uključujući Mac, Windows, Linux i Unix. RDBMS[5] podržava velike baze podataka s milijunskim zapisima i podržava mnoge tipove podataka. Za naučiti MySQL sustav nije potrebno učiti nove naredbe, zbog toga što se podacima može pristupiti uz pomoć standardnih SQL naredbi.

Zbog sigurnosti MySQL koristi privilegiju pristupa i šifrirani sustav zaporki. MySQL klijenti mogu se povezati na MySQL server koristeći nekoliko protokola, uključujući TCP/IP *socket* na bilo kojoj platformi. Također podržava klijentske i uslužne programe, *command-line* programe i administrativne alate poput MySQL Workbench.

## 1.5. React

React je deklarativna, efikasna i fleksibilna JavaScript biblioteka koja se koristi za izradu korisničkih sučelja. React omogućuje rastavljanje kompleksnih korisničkih sučelja na male i izolirane dijelove koda koji se nazivaju komponente. Komponente su samostalni moduli koji daju nekakav output. Elemente sučelja poput gumba ili polja za unos može se napisati kao React komponenta. Komponenta može sadržavati jednu ili više različitih komponenti kao svoj output.

[5] RDBMS - relational database management system

*Slika 3. Primjer React komponente*

Izvor: https://levelup.gitconnected.com/react-component-patterns-ab1f09be2c82

U srcu bilo koje osnovne web stranice nalaze se HTML dokumenti. Web preglednici čitaju te dokumente i prikazuju ih na računalu, tabletu ili pametnom telefonu kao web stranicu. Kroz trajanje tog procesa, preglednici kreiraju DOM[6], reprezentacijsko stablo koje prikazuje kako je web stranica strukturirana.

Budući da je logika komponenata napisana u JavaScriptu umjesto templateu, lako se može proslijediti mnogo podataka kroz aplikaciju i držati *state* izvan DOM-a. (React)

Za razliku od svojih prethodnika, React ne djeluje izravno na DOM-u preglednika, već na virtualnom DOM-u. To jest, umjesto da manipulira dokumentom u pregledniku nakon promjena naših podataka (što može biti prilično sporo), on rješava promjene na DOM-u koji je ugrađen i radi u potpunosti u memoriji. Nakon ažuriranja virtualnog DOM-a, React inteligentno određuje koje će se promjene unijeti u stvarni DOM preglednika. Virtualni React DOM u potpunosti postoji u memoriji i predstavlja web stranicu DOM web preglednika. Zbog toga, kada pišemo React komponentu, ne pišemo izravno DOM-u, već pišemo virtualnu komponentu koja će se React pretvoriti u DOM.

JSX[7] je React ekstenzija koja web programerima omogućuje jednostavno modificiranje DOM-a kroz korištenje jednostavnog koda, sa HTML stilom. Svi moderni

[6] DOM – Document Object Model
[7] JSX – JavaScript eXtension

web preglednici podržavaju React JS, stoga JSX je kompatibilan sa bilo kojom web platformom. To nije samo stvar praktičnosti, već korištenje JSX-a za ažuriranje DOM-a dovodi do značajnih poboljšanja performansi web stranica i učinkovitosti u razvoju.

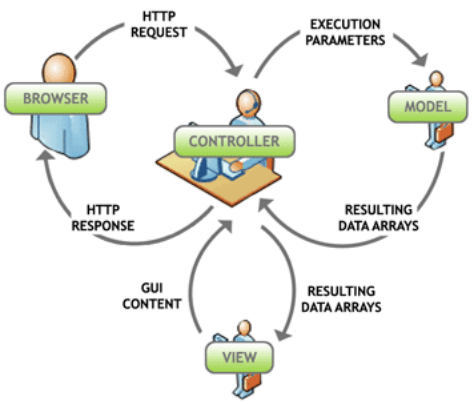
## 1.6. Laravel

PHP je jedan od najraširenijih web programskih jezika. Postoji mnogo PHP okvira, ali samo nekolicina koristi puni potencijal PHP jezika. PHP je evoluirao kroz zadnjih nekoliko godina, gdje dodaje moderne funkcionalnosti da bi ispunio zahtjeve web programera.

Svaki PHP okvir ima svoju implementaciju, funkcionalnosti i mogućnosti. Jedan od PHP okvira je Laravel, besplatan i *open-source* PHP okvir namijenjen za razvoj web aplikacija koje se baziraju na MVC[8] arhitekturi. Laravel se bazira na Symfonyu, drugom najpopularnijem PHP okviru. MVC arhitektura dijeli aplikacije na tri komponente:

- *model*
- *view*
- *controller*

*Slika 4. MVC Arhitektura*




Izvor: https://psychopathya.wordpress.com/2010/02/03/mvcc-model-view-controller-confusion/

[8] MVC - model-view-controller

*Model* je središnja komponenta. On predstavlja dinamičku strukturu podataka aplikacije neovisno o korisničkom sučelju. Izravno upravlja podacima, logikom i pravilima aplikacije. U osnovi je to baza podataka aplikacije.

*View* je ono što se prezentira korisniku. *View* koristi *model* i prezentira podatke u obliku u kojem korisnik to želi. Korisniku se također može omogućiti izmjena podataka predstavljenih korisniku. Sastoje se od statičkih i dinamičkih stranica koje se pružaju ili šalju korisniku kada ih on zatraži.

*Controller* kontrolira zahtjeve korisnika i zatim stvara odgovarajući odgovor koji upućuje posjetitelju. Korisnik obično komunicira s *Viewom*, što zauzvrat generira odgovarajući zahtjev, kojim će upravljati *Controller*. *Controller* daje odgovarajući *View* s podacima modela kao odgovor.

Ukratko model predstavlja podatke, *View* je korisničko sučelje, a *Controller* obrađuje zahtjeve.

Laravel posjeduje veliki ekosustav koji uključuje značajke poput trenutne implementacije, *routinga*, ORM-a, DB upita, predložaka i sl. Neke od ključnih značajki Laravel okvira su:

- **Autorizacija i logička tehnika kodiranja** - ključni dio svake web aplikacije je autentičnost i znanje o tome koliko je vremena programerima potrebno za razvoj koda za provjeru autentičnosti. Laravel ima logičku tehniku koja pomaže u organiziranju logike autorizacije i kontrolira pristup resursima. Uz novije verzije Laravela uključen je kod za validaciju, čime se smanjilo vrijeme pisanja koda. Obrazloženje aplikacija glavna je karakteristika koja ga čini popularnijim jer nudi fleksibilnost i slobodu programerima za izradu svega u rasponu od male web stranice do ogromnog softvera.
- **Inovativni Blade Engine** - Laravel je također poznat po ugrađenim inovativnim i laganim predlošcima. To omogućuje programerima da stvore nešto dinamično i primamljivo. Različiti dostupni *widgeti* također pomažu u stvaranju čvrstih struktura za aplikaciju. Ovaj okvir je zamišljen tako da stvori nešto jedinstveno s karakterističnim odjeljcima.

- **Učinkovit ORM**[9] okvir opremljen je najinteligentnijim ORM-om s PHP Active Record implementacijom. To omogućava programerima da pokreću greške ili upite prema bazi koristeći jednostavnu PHP sintaksu izbjegavajući pisanje SQL koda. Programerima nudi jednostavnu interakciju s bazom podataka, nudeći odgovarajući model za svaku od tablica. To čini ovaj ORM mnogo bržim od ostalih PHP okvira.
- **Biblioteke i moduli** dodaju vrijednost ovom okviru. Najpopularnija je biblioteka za provjeru autentičnosti s mnoštvom ažuriranih i korisnih značajki poput resetiranje zaporke, praćenja aktivnih korisnika, Bycript kriptiranje, CSRF zaštite i slično.
- **Artisan** je povezan s ugrađenim uređajem za naredbe koja se također naziva Artisan. Ova naredba omogućuje programerima da izvršavaju nekoliko ponavljajućih i zamornih zadataka koje često izbjegavaju obavljati ručno. Pomaže programerima izraditi strukturu baze podataka i njihove migracije. Na taj način upravljanje bazom podataka u sustavu postaje jednostavno.
- **Sigurnosni sustav migracija** - Migracijski proces Laravela je vrlo siguran što omogućava proširenje baze podataka web aplikacije olakšavajući programerima uvođenje promjena. Proces migracije postaje cjelovit, siguran i nepropustan. U svom ovom procesu koristi se PHP kod umjesto SQL-a. Također pomaže u izradi tablica baze podataka, uključuje stupce i indekse bez mnogo muke.
- **Jedinstveni Unit-Testing** – Ovaj okvir pokreće nekoliko testova kako bi se osiguralo da promjene neće naštetiti ili napraviti neželjene prekide u web aplikaciji. Može se smatrati pažljivim okvirom koji upozorava na poznate propuste. Uz sve to, programeri mogu lako napisati jedinične testove u vlastitom kodu.
- **Netaknuta sigurnost** - Naravno, sigurnost aplikacije glavna je briga pri izradi svake web aplikacije. Bez sumnje, svaki programer koristi neke učinkovite procese koji održavaju aplikaciju sigurnom. Ovaj je okvir dobro opremljen sigurnosnim sustavom u vlastitom okviru. Napunjena je kriptiranom zaporkom, što znači da se nikakva zaporka nikada ne može spremiti kao uobičajeni tekst u bazu podataka. Stvara se šifrirani prikaz zaporke pomoću

[9] ORM – Object Relational Mapping

Bcrypt Hashing Algorithm, što je siguran način generiranja zaporke. Pored toga, on primjenjuje i SQL izraze, čime štiti od svih injekcijskih napada.

- **Odlični tutorijali (Laracasts)** - Svojim tutorijalima Laracasts je već postao bogat i originalan izvor učenja za početnike i napredne programere. To je postao sjajan alat za učenje Laravel web razvoja. Bez obzira koriste li se besplatni videi ili usluga plaćenih vodiča lako se može naučiti kako koristiti Laravel. Ponuđene upute su precizne, jasne i jednostavne za razumjeti, što može stvoriti vrhunski sadržaj i materijale za trening.

## 1.7. Swift

Swift je intuitivni programski jezik za macOS, iOS, watchOS, tvOS i Linux. Swift pruža kompajlirani opće namjenski jezik za Apple uređaje i računala, te sustave bazirane na Linux-u.

Apple je započeo razvijanje Swifta 2010. godine. Jezik je razvijen tako da je jednostavan za korištenje i siguran, eliminirajući čitave klase uobičajenih pogrešaka kod kodiranja. Također je napravljen da bude siguran bez da utječe na performanse. Spajanje objektno orijentiranog programiranja sa sličnostima s C jezikom pomoglo je njegovoj popularnosti kod programera. Osmišljen je na način da ga se lako nauči, primamljiv je za studente kao i za već postojeće programere.

Swift održava kompatibilnost s postojećim programima koji su napisani kao Objective-C i radi s Cocoa i Cocoa Touch API okvirima za macOS i iOS uređaje. Swift je uključen u Appleov razvojni softver Xcode (SDK). Apple nudi besplatnu dokumentaciju Swifta u iBooks Store-u s priručnikom The Swift Programing Language.

Siguran je, brz i interaktivan jezik koji kombinira najbolje iz modernih jezika u kombinaciji s već stečenim znanjem šireg Apple-ovog inženjerskog društva i ostalih doprinosa iz njegove zajednice. Kompajler je optimiziran za performanse i jezik je optimiziran za programere, bez kompromisa na ijednom.

Swift definira velike klase uobičajenih programskih pogrešaka usvajanjem suvremenih programskih obrazaca:

- Varijable se uvijek inicijaliziraju prije upotrebe
- Indeksi niza provjeravaju se zbog pogrešaka izvan granica
- Optionals[10] osiguravaju da se nulte vrijednosti pravilno kontroliraju
- Memorija se automatski upravlja
- Rukovanje pogreškama omogućuje kontrolirani oporavak od neočekivanih kvarova

[10] Optionals u Swiftu je tip koji sadrži vrijednost ili ne sadrži ništa, pišu se tako da se dodaje na bilo koji tip podatka.

# 2. API

U računalnom programiranju, API[11] je skup definicija potprograma, komunikacijskih protokola i alata za izradu softvera. Općenito govoreći, to je skup jasno definiranih metoda komunikacije među različitim komponentama. Dobar API olakšava razvoj računalnog programa pružanjem svih blokova koje nakon toga programer sastavlja. API može biti za internetski sustav, operativni sustav, sustav baze podataka, računalni hardver ili softversku biblioteku.

Specifikacija API-ja može imati mnoge oblike, ali često uključuje specifikacije za rutine, strukture podataka, klase objekata, varijable ili udaljene pozive. POSIX, Windows API i ASPI primjeri su različitih oblika API-ja. Dokumentacija za API obično se pruža kako bi se olakšala uporaba i implementacija. U izradi aplikacija, API pojednostavljuje programiranje apstrahirajući temeljnu implementaciju i izlažući samo objekte ili radnje koje programeru trebaju. Iako grafičko sučelje za klijenta e-pošte može korisniku pružiti gumb koji izvodi sve korake za dohvaćanje i isticanje novih poruka emaila, API za unos / izlaz datoteke može programeru dati funkciju koja kopira datoteku s jedne lokacije na drugu bez potrebe da programer razumije funkcije datotečnog sustava koje se događaju iza scene.

API je obično povezan sa softverskom bibliotekom. API opisuje i propisuje "očekivano ponašanje" (specifikaciju), dok je biblioteka "stvarna implementacija" ovog skupa pravila. Jedan API može imati više implementacija (ili nijednu, apstraktnu) u obliku različitih biblioteka koje dijele isto programsko sučelje. Odvajanje API-ja od njegove implementacije može omogućiti programima napisanim na jednom jeziku da koriste biblioteku napisanu na drugom.

Jezična povezivanja također su API. Mapiranjem značajki i mogućnosti jednog jezika u sučelju implementiranom na drugom jeziku, jezično povezivanje omogućava korištenje biblioteke ili usluge napisanih na jednom jeziku da se koriste u razvijanju na drugom jeziku. Alati poput SWIG i F2PY, generatora sučelja Fortran-to-Python, olakšavaju stvaranje takvih sučelja. API se također može povezati sa softverskim okvirom. Okvir se može temeljiti na nekoliko biblioteka koje implementiraju nekoliko

[11] API – Application Programming Interface

API-ja, ali za razliku od uobičajene uporabe API-ja, pristup ponašanju ugrađenom u biblioteku posreduje se proširivanjem njegovog sadržaja s novim klasama uključen u sam okvir.

## 2.1. Operacijski sustavi

API može odrediti sučelje između aplikacije i operacijskog sustava. POSIX, na primjer, određuje skup zajedničkih API-ja koji imaju cilj omogućiti prijavu napisanu za POSIX operativni sustav da bi se kompajlirao za drugi POSIX operativni sustav. Linux i Berkeley Software Distribution su primjeri operativnih sustava koji implementiraju POSIX API-je.

Microsoft je pokazao snažnu predanost *backward-compatible* API-ju, posebno unutar svoje Windows API (Win32) biblioteke, tako da se starije aplikacije mogu izvoditi na novijim verzijama sustava Windows koristeći postavku specifičnu za izvršenje nazvanu Compatibility Mode. API se razlikuje od ABI[12] po tome što se API temelji na izvornom kodu, dok se ABI temelji na binarnom obliku. Na primjer, POSIX pruža API, dok Linux Standard Base pruža ABI.

## 2.2. Remote API

*Remote* API-ji omogućuju programerima da manipuliraju udaljenim resursima putem protokola, posebnih standarda za komunikaciju koji omogućuju da različite tehnologije rade zajedno, bez obzira na jezik ili platformu. Na primjer, Java Database Connectivity API omogućava programerima da traže mnogo različitih vrsta baza podataka s istim skupom funkcija, dok API za udaljenu metodu Java koristi protokol Java Remote Method Protocol kako bi omogućio pozivanje funkcija koje djeluju na daljinu, ali izgledaju kao da djeluju lokalno programeru.

[12] ABI - application binary interface

Stoga su udaljeni API-ji korisni u održavanju apstrakcije objekta u objektno orijentiranom programiranju; poziv metode, izvodi se lokalno na *proxy* objektu, poziva odgovarajuću metodu na udaljenom objektu, koristeći *remote* protokol i dobiva rezultat koji će se koristiti lokalno kao povratna vrijednost. Promjena *proxy* objekta također će rezultirati odgovarajućom izmjenom udaljenog objekta.

## 2.3. Web API

Web API su definirana sučelja kroz koja se događaju interakcije između poduzeća i aplikacija koje koriste njegovu imovinu, a koji je SLA[13] za određivanje funkcionalnog pružatelja usluga i otkrivanje puta usluge ili URL-a za svoje API korisnike. API pristup je arhitektonski pristup koji se vrti oko pružanja programskog sučelja skupu usluga različitim aplikacijama koje služe različitim vrstama potrošača. Postoje četiri glavna tipa API-ja:

- ***Open* API** - Također poznat kao javni API, nema ograničenja za pristup tim vrstama API-ja, jer su javno dostupni
- ***Partner* API** - za pristup ovoj vrsti API-ja potrebna su posebna prava ili licence, jer oni nisu javno dostupni
- ***Internal* API** - Poznati i kao privatni API-i, samo interni sustavi koriste ovu vrstu API-ja, koji je, prema tome, manje poznat i često namijenjen upotrebi unutar tvrtke. Tvrtka koristi ovu vrstu API-ja među različitim internim timovima kako bi mogla poboljšati svoje proizvode i usluge
- ***Composite* API** - Ova vrsta API-ja kombinira različite API-je za podatke i usluge. To je niz zadataka koji se pokreću sinkrono kao rezultat izvršenja, a ne na zahtjev zadatka. Njegove glavne uporabe su ubrzavanje procesa izvršenja i poboljšanje performansi slušatelja na web sučeljima.

Kad se koristi u kontekstu web razvoja, API se obično definira kao skup specifikacija, poput HTTP poruke zahtjeva, zajedno s definicijom strukture odgovora,

[13] SLA - Service Level Agreement

obično u XML[14] ili JSON[15] formatu. Primjer može biti API brodarske tvrtke koji se može dodati na web stranicu usmjerenu na e-trgovinu kako bi se olakšalo naručivanje usluga isporuke i automatski uključilo trenutne cijene otpreme, a da programer web stranice ne mora unositi tablicu cijene otprema u web bazu podataka. Iako je web API povijesno praktički bio sinonim za web uslugu, nedavni trend (tzv. Web 2.0) odmiče se od web servisa zasnovanih na SOAP-u[16] i SOA-u[17] prema direktnijem REST stilu web resursa i arhitektura orijentiranoj na resurse (ROA[18]). Dio ovog trenda povezan je sa semantičkim kretanjem weba prema RDF-u[19], konceptu za promoviranje mrežne onologije inženjerskih tehnologija. Web API-ji omogućavaju kombiniranje više API-ja u nove aplikacije.

U skupu društvenih medija, web API-ji su omogućili web zajednicama da olakšavaju razmjenu sadržaja i podataka između zajednica i aplikacija. Na taj se način sadržaj koji se dinamički stvara na jednom mjestu može objavljivati i ažurirati na više lokacija na webu. Na primjer, Twitter-ov REST API omogućuje programerima pristup temeljnim podacima na Twitteru, a Search API pruža metode za programere za interakciju s Twitter pretraživanjem i podacima o trendovima. Osim glavnih web API-ja, postoje API-ji web servisi:

- **SOAP** je protokol koji koristi XML kao format za prijenos podataka. Njegova je glavna funkcija definiranje strukture poruka i načina komunikacije. Također koristi WSDL[20], u strojno čitljivom dokumentu da objavi definiciju svog sučelja.
- **XML-RPC** je protokol koji koristi poseban XML format za prijenos podataka u odnosu na SOAP koji koristi vlastiti XML format. Također je stariji od SOAP-a. XML-RPC koristi minimalnu propusnost i puno je jednostavniji od SOAP-a.
- **JSON-RPC** je protokol sličan XML-RPC, ali umjesto da koristi XML format za prijenos podataka, koristi JSON.
- **REST** nije protokol kao ostali web servisi, već je to skup arhitektonskih načela. REST usluga mora imati određene karakteristike, uključujući jednostavna

[14] XML - Extensible Markup Language
[15] JSON - JavaScript Object Notation
[16] SOAP - Simple Object Access Protocol
[17] SOA - service-oriented architecture
[18] ROA - resource-oriented architecture
[19] RDF - Resource Description Framework
[20] WSDL - Web Services Definition Language

sučelja, gdje se resursi lako identificiraju unutar zahtjeva i manipulira se resursima pomoću sučelja.

## 2.4. Primjeri popularnih API-ja

API-ji se koriste i iz svakakvih razloga. Na primjer, ako se koristi Google karte objekt ugrađen u web stranicu, ta web stranica koristi Google karte API za ugradnju te karte. Google izlaže API-je poput ovog web programerima, koji tada mogu pomoću API-ja crtati složene objekte izravno na svojoj web stranici. Ako takvi API-ji ne postoje, programeri će možda morati kreirati vlastite karte i pružiti vlastite podatke karata samo kako bi postavili malu interaktivnu kartu na web stranicu.

Budući da je riječ o API-ju, Google može kontrolirati pristup Google kartama na drugim web lokacijama, osiguravajući da ih koriste na dosljedan način, a ne da pokušavaju neuredno ugraditi okvir koji prikazuje Google karte. To se odnosi na mnoge različite internetske usluge. Postoje API-ji za traženje prijevoda teksta s Google prevoditelja ili za umetanje Facebook komentara ili objave s Twittera na web stranicu. Neki od popularnijih API-ja su:

- **Google Maps API** - omogućuju programerima da ugrade Google karte na web stranice pomoću JavaScript ili Flash sučelja. Ovaj API dizajniran je za rad na mobilnim uređajima i preglednicima.
- **YouTube API** - omogućuju programerima da integriraju YouTube videozapise i funkcionalnosti u web stranice ili programe. YouTube API-ji uključuju YouTube Analytics API, YouTube Data API, YouTube Live Streaming API, YouTube Player API i druge.
- **Flickr API** – koriste ga programeri za pristup podijeljenim fotografijama zajednice Flickr. Flickr API se sastoji od skupa metoda koje se može nazvati i nekih krajnjih API točaka.
- **Twitter API** - Twitter nudi dva API-ja. API REST omogućuje programerima pristup osnovnim podacima Twittera, a API za pretraživanje pruža metode za razvojne programere da imaju interakciju s Twitter pretraživanjem i podacima o trendovima.

- **Amazon Product Advertising API** - Amazonov API za oglašavanje proizvoda pruža programerima pristup funkcijama Amazon-ovom odabiru i otkrivanju proizvoda za oglašavanje Amazon-ovih proizvoda radi unovčavanja web stranice.

# 3. REST API

REST je arhitektonski stil definiran da pomogne u stvaranju i organiziranju distribuiranih sustava. Ključna riječ iz te definicije trebao bi biti stil, jer je važan aspekt REST-a taj da je to arhitektonski stil, ne smjernica, niti standard, niti išta što bi podrazumijevalo da postoji niz teških pravila koja treba slijediti da bi na kraju postala RESTful arhitektura. (Doglio, 2018)

Representational State Transfer (REST) uobičajena je kategorija API-ja koja ne ovisi o konkretnom protokolu. Nudi mogućnost fleksibilne integracije koja omogućava programerima da koriste standardizirani skup procesa za postizanje svojih ciljeva. Kada se poziva RESTful API, poslužitelj će klijentu prenijeti prikaz stanja traženog resursa. (Avraham, 2017)

REST se smatra lako razumljivim, stoga mnogi programeri imaju iskustva s ovom tehnologijom. REST je arhitektonski stil koji definira skup ograničenja koja će se koristiti za stvaranje web usluga. Sustavi koji odgovaraju REST-u, često se nazivaju RESTful sustavi, omogućuju interoperabilnost računalnih sustava na Internetu. RESTful Web usluge omogućuju sustavima koji traže zahtjev za pristup i manipuliranje tekstualnim prikazima web resursa pomoću jedinstvenog i unaprijed definiranog skupa operacija bez stanja. Ostale vrste web usluga, poput SOAP Web usluga, izlažu vlastitim proizvoljnim skupovima operacija.

Web resursi prvi su put definirani na World Wide Webu kao dokumenti ili datoteke identificirane njihovim URL-ovima. Međutim, oni danas imaju mnogo općenitiju i apstraktniju definiciju koja obuhvaća svaku stvar ili entitet koji se na bilo koji način može identificirati, imenovati, adresirati ili rukovati na bilo koji način. U RESTful web usluzi zahtjevi upućeni na URI resursa dobit će odgovor formatiran u HTML, XML, JSON ili nekom drugom obliku.

JavaScript Object Notation (JSON) lagan je, temeljen na tekstu, format za razmjenu podataka neovisan o jeziku. (Bray, 2017)

XML (eXtensible Markup Language) je metajezik (jezik koji se koristi za opisivanje drugih jezika) za definiranje vokabulara (prilagođeni označni jezici), što je

ključ za važnost i popularnost XML-a. Rječnici utemeljeni na XML-u (kao što je XHTML) omogućuju opisivanje dokumenata na smisleni način. (Friesen, 2019)

Odgovor može potvrditi da je izvršena određena izmjena na pohranjenom resursu, a odgovor može pružiti hipertekst veze do ostalih povezanih resursa ili zbirki resursa. Kada se koristi HTTP, kao što je najčešće, dostupne su operacije (HTTP metode) GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS i TRACE.

Korištenjem protokola bez stanja i standardnim operacijama, RESTful sustavi ciljaju na brze performanse, pouzdanost i sposobnost rasta pomoću ponovne upotrebe komponenata kojima se može upravljati i ažurirati bez utjecaja na sustav u cjelini, čak i dok se pokreće.

## 3.1. Način rada RESTful API-ja

RESTful API raščlanjuje transakciju kako bi stvorio niz malih modula. Svaki modul adresira određeni temeljni dio transakcije. Ova modularnost pruža programerima veliku fleksibilnost, ali programerima može biti teško da dizajniraju ispočetka. Trenutno su najpopularniji modeli koje pružaju Amazon S3, Cloud Data Interface (CDMI) i OpenStack Swift. (Rouse, 2019)

RESTful API izričito koristi HTTP metodologije definirane RFC 2616 protokolom. Oni koriste:

- GET za pronalaženje resursa
- PUT za promjenu stanja ili ažuriranje resursa, koji može biti objekt, datoteka ili blok
- POST za stvaranje tog resursa
- DELETE za ukloniti resurs

Uz REST umrežene komponente su resurs kojem trebate pristupiti - crni okvir čiji detalji implementacije nisu jasni. Pretpostavka je da su svi pozivi *stateless*, RESTful servis ništa ne može zadržati između izvođenja.

Budući da su pozivi *stateless*, REST je koristan u *cloud* aplikacijama. *Stateless* komponente mogu se slobodno preusmjeriti ako nešto ne uspije i mogu se prilagoditi promjenama opterećenja. To je zato što se svaki zahtjev može usmjeriti na bilo koju instancu komponente, ne može se sačuvati ništa što se mora zapamtiti sljedećom transakcijom. To REST daje prednost web upotrebi, ali RESTful model također je koristan u *cloud* servisima jer je vezanje za uslugu putem API-ja stvar kontrole načina dekodiranja URL-a. *Cloud computing* i mikroservisi gotovo su sigurni da će RESTful API dizajn postati pravilo u budućnosti.

U REST arhitektonskom stilu, implementacija klijenta i implementacija poslužitelja mogu se izvršiti samostalno. To znači da se kod na strani klijenta može u bilo kojem trenutku promijeniti bez utjecaja na rad poslužitelja, a kod na strani poslužitelja se može promijeniti bez utjecaja na rad klijenta.

Sve dok svaka strana zna koji format poruka treba poslati drugoj, može ih se zadržati modularno i odvojeno. Odvajajući brige o korisničkom sučelju od brige o pohrani podataka, poboljšava se fleksibilnost sučelja na svim platformama i poboljšava se skalabilnost pojednostavljujući komponente poslužitelja. Uz to, razdvajanje omogućuje svakoj komponenti sposobnost da se samostalno razvija.

### 3.1.1. REST vs SOAP

REST koristi pristup temeljen na resursima za internetske interakcije. Pomoću REST-a locira se resurs na poslužitelju i odluči ga se ažurirati, izbrisati ili dobiti neke informacije o njemu. (Rouse, 2019)

Pomoću SOAP-a klijent ne želi izravno komunicirati s resursom, već poziva uslugu i ta usluga ublažava pristup raznim objektima i resursima iza scene. (Rouse, 2019)

### 3.1.2. Nepostojanje stanja

Sustavi koji slijede REST paradigmu nemaju status, što znači da poslužitelj ne mora znati ništa o stanju klijenta i obrnuto. Na taj način i poslužitelj i klijent mogu razumjeti bilo koju primljenu poruku, čak i bez gledanja prethodnih poruka. Ovo ograničenje statusa ostvaruje se uporabom resursa, a ne naredbi. Resursi su imenice Weba i opisuju bilo koji predmet, dokument ili stvar koji će se možda trebati spremiti ili poslati drugim uslugama.

Budući da REST sustavi komuniciraju standardnim operacijama na resursima, ne oslanjaju se na implementaciju sučelja. Ta ograničenja pomažu RESTful aplikacijama da postignu pouzdanost, brze performanse i skalabilnost, kao komponente kojima se može upravljati, ažurirati i ponovo koristiti bez utjecaja na sustav u cjelini, čak i za vrijeme rada sustava.

## 3.2. Prednosti RESTful servisa

RESTfull servisi su jednostavni, koriste postojeće mrežne infrastrukture, te imaju fleksibilnost formata kod vraćenih podataka. Glavne prednosti su:

- **Razdvajanje klijenta i poslužitelja** - REST protokol potpuno razdvaja korisničko sučelje od poslužitelja i pohrane podataka što ima određene prednosti prilikom izrade projekata. Na primjer, poboljšava prenosivost sučelja na druge vrste platformi, povećava skalabilnost projekata i omogućava da se različite komponente razvoja događaju neovisno.
- **Vidljivost, pouzdanost i skalabilnost** - Razdvajanje između klijenta i poslužitelja ima jednu očitu prednost i to je da svaki razvojni tim može skalirati proizvod bez previše problema. Oni se mogu migrirati na druge poslužitelje ili unositi sve vrste promjena u bazu podataka, pod uvjetom da su podaci iz svakog zahtjeva ispravno poslani. Odvajanje olakšava da se sučelje i pozadina nalaze na različitim poslužiteljima, a to aplikacije čini fleksibilnijim za rad.

- **REST API je uvijek neovisan o vrsti platforme ili jezicima** - REST API uvijek se prilagođava vrsti sintakse ili platformi koja se koristi, što daje značajnu slobodu pri promjeni ili testiranju novih okruženja u razvoju. Pomoću REST API-ja može se imati PHP, Java, Python ili Node.js poslužitelje. Jedino je neophodno da se odgovori na zahtjeve uvijek odvijaju na jeziku koji se koristi za razmjenu informacija, obično XML-u ili JSON-u.

## 3.3. RESTful API Arhitektonska ograničenja

RESTful sustav sastoji se od klijenta koji traži resurse i poslužitelja koji ima resurse. Važno je kreirati REST API u skladu s industrijskim standardima, što rezultira jednostavnim razvojem i povećanjem broja klijenata. Postoji šest arhitektonskih ograničenja koja sačinjavaju bilo koju web uslugu, a to su:

- Uniformno sučelje
- Nepostojanje stanja
- Mogućnost predmemoriranja (Cacheable)
- Arhitektura Klijent - Poslužitelj
- Slojni sustav
- Kod na zahtjev

Jedino izborno ograničenje REST arhitekture je kod na zahtjev. Ako neka usluga krši neko drugo ograničenje, ne može se nazvati RESTful.

### 3.3.1. Uniformno sučelje

Resursi bi se trebali jedinstveno identificirati putem jednog URL-a, i to korištenjem temeljnih metoda mrežnog protokola, metoda poput DELETE, PUT i GET s HTTP-om, koko bi se omogućilo manipuliranje resursom. Ravnomjerno ograničenje sučelja je temeljno za dizajn bilo kojeg RESTful sustava. Pojednostavljuje i odvaja arhitekturu,

što omogućava da se svaki dio samostalno razvija. Četiri ograničenja za uniformno sučelje su:

- **Identifikacija resursa u zahtjevima** - Pojedinačni resursi identificiraju se u zahtjevima, primjerice pomoću URI-ova u RESTful web uslugama. Sami resursi konceptualno su odvojeni od predstavljanja koja se vraćaju klijentu. Na primjer, poslužitelj može slati podatke iz svoje baze podataka u obliku HTML, XML ili kao JSON - od kojih nijedan nije predstavnik poslužitelja.
- **Manipulacija resursima putem reprezentacija** - Kad klijent ima predstavu resursa, uključujući priložene meta podatke, ima dovoljno podataka za izmjenu ili brisanje resursa.
- **Poruke koje se same opisuju** - Svaka poruka sadrži dovoljno informacija za opisivanje kako je obraditi.
- **Hypermedia as the engine of application state (HATEOAS)** - Nakon pristupa početnom URI-u za REST aplikaciju - analognom ljudskom korisniku weba koji pristupa početnoj stranici web stranice - REST klijent bi tada trebao biti u mogućnosti dinamički koristiti veze na poslužitelju kako bi otkrio sve dostupne radnje i resurse koji su mu potrebni. Kako se pristup nastavlja, poslužitelj odgovara tekstom koji uključuje hiperveze na ostale radnje koje su trenutno dostupne. Nema potrebe da klijent bude strogo kodiran s informacijama o strukturi ili dinamici aplikacije.

### 3.3.2. Nepostojanje stanja

To znači da je potrebno stanje za obradu zahtjeva sadržano u samom zahtjevu i poslužitelj ne pohranjuje ništa povezano sa sesijom. U REST-u klijent mora uključiti sve podatke za poslužitelja kako bi ispunio zahtjev, bilo kao dio upita parametra, zaglavlja ili URI-ja. Nepostojanje stanja omogućuje veću dostupnost jer poslužitelj ne mora održavati, ažurirati ili komunicirati to stanje sesije. Postoji nedostatak kada klijent treba poslati previše podataka poslužitelju, tako da smanjuje opseg optimizacije mreže i zahtijeva veću propusnost.

### 3.3.3. Mogućnost predmemoriranja

Za svaki odgovor treba znati može li se spremiti ili ne, te koliko dugo se odgovor može spremiti na strani klijenta. Klijent vraća podatke iz svoje predmemorije za bilo koji sljedeći zahtjev, te više nije potrebno slanje zahtjeva poslužitelju. Dobro upravljano predmemoriranje djelomično ili u potpunosti eliminira neke interakcije klijenta i poslužitelja, dodatno poboljšavajući dostupnost i performanse. No, ponekad postoje mogućnosti da korisnik može primiti zaostale podatke, tj. podatke koji nisu ažurirani.

### 3.3.4. Arhitektura klijent – poslužitelj

REST aplikacija treba imati arhitekturu klijent-poslužitelj. Klijent je netko tko traži resurse i nije zabrinut za pohranu podataka, koja ostaje interna za svaki poslužitelj, a poslužitelj je netko tko drži resurse i nije zabrinut za korisničko sučelje ili stanje korisnika. Mogu se samostalno razvijati. Klijent ne mora znati ništa o poslovnoj logici, a poslužitelj ne mora znati ništa o  korisničkom sučelju.

Odvajanje briga o korisničkom sučelju od zabrinutosti za pohranu podataka poboljšava prenosivost korisničkih sučelja na više platformi. Također poboljšava skalabilnost pojednostavljivanjem komponenti poslužitelja. Možda je najvažnije za Internet da razdvajanje omogućuje komponentama da se samostalno razvijaju, podržavajući tako višestruke organizacijske domene na Internetu.

### 3.3.5. Slojni sustav

Arhitektura aplikacije mora biti sastavljena od više slojeva. Svaki sloj ne zna ništa o bilo kojem sloju osim onog neposrednog sloja i može biti mnogo posredničkih poslužitelja između klijenta i krajnjeg poslužitelja. Posrednički poslužitelji mogu poboljšati dostupnost sustava omogućavanjem uravnoteženja opterećenja i pružanjem zajedničkih predmemorija.

### 3.3.6. Kod na zahtjev

REST omogućava proširenje funkcionalnosti klijenta za preuzimanje i izvršavanje koda u obliku umetka ili skripti. To pojednostavljuje klijente smanjenjem broja značajki koje je potrebno prethodno implementirati. Dopuštanje značajki za preuzimanje nakon implementacije poboljšava proširivost sustava. Također se smanjuje vidljivost, a samim tim je samo neobavezno ograničenje unutar REST-a. (Fielding, 2000)

To je opcionalna značajka. Većinu vremena poslužitelj će poslati statičke prikaze resursa u obliku XML-a ili JSON-a. Međutim, prema potrebi, poslužitelji mogu poslati izvršni kod klijentu.

# 4. REST API smjernice

Ne postoje određena pravila kojih se svi REST API moraju držati, svatko može napraviti svoju verziju API-ja. Međutim nije loše imati neke smjernice po kojima bi se napravio API, stoga će u ovom poglavlju biti objašnjene Microsoft API smjernice. Smjernice će biti ukratko objašnjene, a dodatne informacije dostupne su na poveznici https://github.com/Microsoft/api-guidelines/blob/vNext/Guidelines.md.

## 4.1. Microsoft API smjernice

### 4.1.1. Smjernice za klijenta

Za slabo povezane klijente kod kojih točan oblik podataka nije poznat prije poziva, ako poslužitelj vrati nešto što klijent nije očekivao, klijent MORA to sigurno ignorirati. Klijenti se NE smiju pouzdati u redoslijed po kojem se podaci pojavljuju u JSON servisnim odgovorima. Klijenti koji traže OPCIJSKU funkciju poslužitelja (poput opcionalnih zaglavlja) MORAJU biti spremni da poslužitelj zanemari tu određenu funkcionalnost. (Microsoft)

### 4.1.2. Osnovne dosljednosti

Ljudi bi trebali biti u stanju lako čitati i stvarati URL-ove. Usluge koje mogu generirati URL-ove duže od 2083 znaka MORAJU se prilagoditi za klijente koje ih žele podržavati. Pored prijateljskih URL-ova, resursi koji se mogu premjestiti ili preimenovati TREBA izložiti URL koji sadrži jedinstveni stabilni identifikator. (Microsoft)

Sve vrijednosti zaglavlja MORAJU slijediti pravila sintakse navedena u specifikaciji u kojoj je definirano polje zaglavlja. Mnogo HTTP zaglavlja definirano je u RFC7231, no potpuni popis odobrenih zaglavlja može se naći u IANA registru

zaglavlja. Prilagođena zaglavlja NE MORAJU biti potrebna za osnovni rad danog API-ja. (Microsoft)

Neka zaglavlja predstavljaju izazove za neke scenarije, poput AJAX klijenata, posebno pri upućivanju poziva na više domena gdje dodavanje zaglavlja MOŽE biti podržano. Kao takva, neka se zaglavlja mogu prihvatiti kao parametri upita pored zaglavlja, s istim imenovanjem kao i zaglavlje. Kriteriji za razmatranje kada prihvatiti zaglavlja kao parametre su:

- Svako prilagođeno zaglavlje MORA biti prihvaćeno i kao parametar

- Potrebna standardna zaglavlja MOGU se prihvatiti kao parametri

- Potrebna zaglavlja sa sigurnosnom osjetljivošću (npr. Zaglavlje autorizacije) NE MOGU biti odgovarajuća kao parametri, vlasnik usluge TREBA ih ocijeniti od slučaja do slučaja

U skladu s pravilima o privatnosti svoje organizacije, klijenti NE MORAJU prenijeti parametre podataka (PII) koji se mogu osobno identificirati u URL-u (kao dio putanje ili niza upita) jer se ti podaci mogu nenamjerno izložiti putem dnevnika klijenta, mreže i poslužitelja i drugih mehanizama. (Microsoft)

Imena svojstava JSON TREBALA bi biti *camelCase*. Usluga TREBA pružiti JSON kao zadano kodiranje. Na HTTP-u bi klijent trebao zatražiti format odgovora pomoću zaglavlja Accept. To je savjet i poslužitelj ga može ignorirati ako odluči, čak i ako to nije tipično za dobro poslužene poslužitelje. Klijenti MOGU poslati više zaglavlja Accept, a usluga MOŽE odabrati jedno od njih. (Microsoft)

### 4.1.3. CORS

Usluge sukladne Microsoft REST API smjernicama MORAJU podržati CORS (Cross Origin Resource Sharing). Usluge TREBA podržati dopušteno podrijetlo CORS * i nametnuti autorizaciju putem valjanih OAuth tokena. Usluge NE TREBAJU podržavati korisničke podatke s provjerom podrijetla. MOŽE biti izuzetaka za posebne slučajeve. (Microsoft)

#### 4.1.3.1. *Smjernice za klijenta*

Web programeri obično ne trebaju raditi ništa posebno da bi iskoristili prednost CORS-a. Svi se koraci rukovanja događaju nevidljivo kao dio standardnih XMLHttpRequest poziva koje upućuju. (Microsoft)

#### 4.1.3.2. *Smjernice za usluge*

Minimalno što usluga MORA je:

- Shvatiti zaglavlje zahtjeva za porijeklo koje preglednici šalju na zahtjevima za više domena i zaglavlje zahtjeva Access-Control-Request-Method, koje oni šalju na zahtjev za predbilježbom OPTIONS koji pregledavaju pristup.
- Ako je zaglavlje Origin prisutno u zahtjevu:
    - Ako zahtjev koristi metodu OPTIONS i sadrži zaglavlje Access-Control-Request-Method, tada je to zahtjev za namijenjen ispitivanju pristupa prije stvarnog zahtjeva. Inače je stvarni zahtjev
    - U odgovor dodati zaglavlje Access-Control-Allow-Origin koji sadrži istu vrijednost kao i zaglavlje zahtjeva Origin. Imati na umu da to zahtijeva da usluge dinamički generiraju vrijednost zaglavlja.
    - Ako pozivatelj zahtjeva pristup zaglavlju odgovora koji nije u skupu jednostavnih zaglavlja odgovora (kontrola predmemorije, jezik sadržaja, vrsta sadržaja, ističe, zadnja izmjena, pragma), dodati Access-Control-Expose-Headers zaglavlje koje sadrži popis imena dodatnih odgovora za klijenta kojem bi klijent trebao pristupiti.
    - Ako zahtjev zahtijeva kolačiće, dodati zaglavlje Access-Control-Allow-Credentials i postaviti na "true".
    - Ako je zahtjev bio prije isporuke zahtjeva, usluga MORA:
        - Dodati zaglavlje odgovora Access-Control-Allow-Headers koji sadrži popis imena zaglavlja zahtjeva koje klijent smije koristiti.
        - Dodati zaglavlje odgovora Access-Control-Allow-Methods koji sadrži popis HTTP metoda koje pozivatelj smije koristiti.

### 4.1.4. Kolekcije

Usluge MOGU podržavati trajne identifikatore za svaku stavku u zbirci, a taj bi identifikator trebao biti predstavljen u JSON-u kao id. Ovi trajni identifikatori često se koriste kao ključevi predmeta. Zbirke koje podržavaju trajne identifikatore MOGU podržati delta upite. Zbirke su predstavljene u JSON koristeći standardnu matricu polja. Zbirke se nalaze izravno ispod korijena usluge kada su na najvišoj razini ili kao segment pod drugim resursom kada se dodijele tom izvoru. Kako podaci rastu, tako se povećavaju i zbirke stoga je planiranje paginacije važno za sve usluge. (Microsoft)

POST zahtjevi nisu *idempotent*. To znači da dva POST zahtjeva poslana u zbirni izvor s točno istim korisnim opterećenjem MOŽE dovesti do stvaranja više predmeta u toj kolekciji. To se često događa za operacije umetanja na stavkama s ID-om generiranim na strani poslužitelja. (Microsoft)

Rezultati upita u zbirci MOGU se sortirati na temelju vrijednosti svojstava. Svojstvo određuje vrijednost parametra upita $ orderBy. Izraz MOŽE sadržavati sufiks *asc* za uzlazni ili *desc* za silazni, odvojen od naziva svojstva jednim ili više razmaka. Ako *asc* ili *desc* nije naveden, usluga MORA naručiti navedeno svojstvo uzlaznim redoslijedom. NULL vrijednosti MORAJU razvrstati kao "manje od" vrijednosti koje nisu NULL. RESTful API-ji koji vraćaju zbirke MOGU vratiti djelomične skupove. Potrošači ovih usluga MORAJU očekivati djelomične rezultate rezultata i ispravno pregledavanje stranica kako bi preuzeli cijeli skup. Parametri sortiranja i filtriranja MORAJU biti dosljedni na svim stranicama, jer su i pozivi na strani klijenta i na poslužitelju u potpunosti kompatibilni i s filtriranjem i sa sortiranjem. Usluge MOGU odabrati da podrže delta upite. (Microsoft)

### 4.1.5. Verzije

Svi API-ji sukladni Microsoft REST API smjernicama MORAJU podržavati eksplicitnu verziju. Presudno je da klijenti mogu računati na to da će usluge biti stabilne tijekom vremena, a ključno je da usluge mogu dodavati značajke i unositi promjene. (Microsoft)

### 4.1.6. Operacije s dugim trajanjem

Dugotrajne operacije, koje se ponekad nazivaju i asinkrone operacije, obično znače različite stvari kod različitih ljudi. Ovaj je odjeljak izložio smjernice o različitim vrstama dugotrajnih operacija i opisao žičane protokole i najbolje prakse za ove vrste operacija. (Microsoft)

- Jedan ili više klijenata MORAJU biti u stanju istovremeno nadzirati i raditi na istom resursu.

- Stanje sustava POTREBNO je u svakom trenutku otkriti i ispitati. Klijenti bi trebali biti u stanju odrediti stanje sustava, čak i ako resurs za praćenje rada više nije aktivan. Sam čin ispitivanja stanja dugotrajne operacije trebao bi sam utjecati na načela weba. tj. dobro definirani resursi s ujednačenom semantikom sučelja. Klijenti MOGU izdati GET na nekom resursu kako bi utvrdili stanje dugog rada.

- Dugotrajne operacije TREBAJU raditi za klijente koji žele "Zapaliti i zaboraviti" i za klijente koji žele aktivno nadzirati i djelovati na rezultate.

- Otkazivanje ne znači izričito povrat. Za slučaj koji je određen po API-ju to može značiti povrat, kompenzaciju ili dovršetak ili djelomični završetak, itd. Nakon otkazane operacije NE MORA biti odgovornost klijenta da uslugu vrati u dosljedno stanje što omogućuje nastavak usluge.

### 4.1.7. Provlačenje, kvote i limiti

Usluge bi trebale biti što prihvatljivije kako ne bi blokirali pozivatelje. U pravilu, svaki API poziv za koji se očekuje da traje duže od 0.5 sekundi, trebao bi razmotriti korištenje obrasca dugotrajnih operacija za te pozive. Očito je da usluge ne mogu jamčiti ova vremena odaziva, u slučaju potencijalno neograničenog opterećenja od pozivatelja. Usluge bi stoga trebale osmisliti i dokumentirati ograničenja zahtjeva za

pozive za klijente i odgovarati na odgovarajuće, djelotvorne pogreške i poruke o pogrešci ako su ta ograničenja prekoračena. (Microsoft)

Usluge bi trebale reagirati brzo s pogreškom kada su obično preopterećene, a ne jednostavno sporo odgovarati. Konačno, mnoge će usluge imati kvote za pozive, možda i niz operacija po satu ili danu, obično povezane s planom usluge ili cijenom. Kada se ove kvote prekorače, usluge moraju pružiti neposredne i djelotvorne pogreške. Kvote i ograničenja trebaju se dodijeliti korisničkoj jedinici: pretplata, zakupnik, aplikacija, plan ili bez ikakve druge identifikacije niz ip adresa… prema potrebi uslužnim ciljevima kako bi se opterećenje pravilno podijelilo i jedna jedinica ne ometa drugu. (Microsoft)

### 4.1.8. Nepodržani zahtjevi

Klijenti RESTful API-ja mogu zatražiti funkcionalnost koja trenutno nije podržana. RESTful API-ji MORAJU odgovoriti na važeće, ali nepodržane zahtjeve u skladu s ovim odjeljkom. RESTful API-ji često biraju ograničiti funkcionalnost koju mogu obavljati klijenti. Na primjer, revizorski sustavi omogućuju stvaranje zapisa, ali ne i modificiranje ili brisanje. Slično tome, neki API-ji izložit će kolekcije, ali zahtijevaju ili na neki drugi način ograničavaju kriterije filtriranja i naručivanja ili NE MOGU podržati paginaciju koju vodi klijent. (Microsoft)

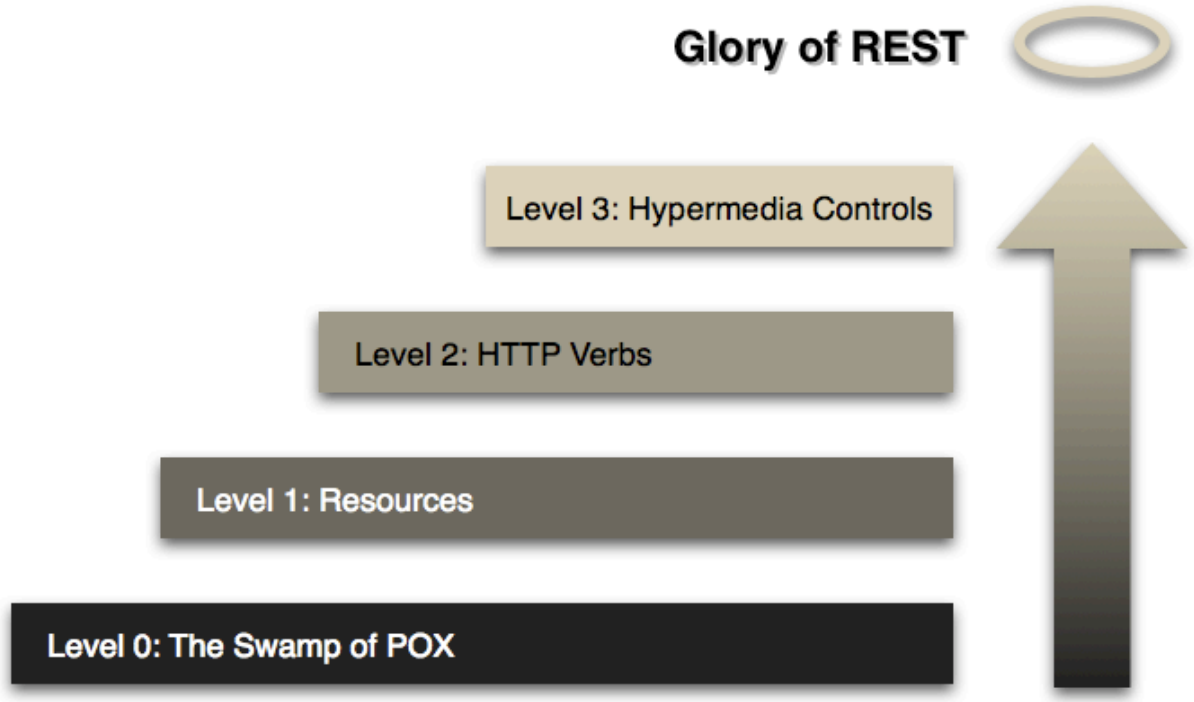
### 4.1.9. Smjernice za imenovanje

Politike imenovanja trebale bi pomoći programerima u otkrivanju funkcionalnosti bez potrebe za stalnim pozivanjem na dokumentaciju. Upotreba uobičajenih obrazaca i standardnih konvencija uvelike pomaže programerima u ispravnom nagađanju uobičajenih imena i značenja svojstava. Usluge TREBAJU koristiti „verbose“ obrasce imenovanja, a NE TREBAJU koristiti skraćenice osim akronima koji su dominantni način izražavanja u domeni koju predstavlja API (npr. Url). (Microsoft)

# 5. RICHARDSONOV MODEL ZRELOSTI

Richardson je koristio tri faktora za određivanje zrelosti usluge, tj. URI, HTTP metode i HATEOAS (*Hypermedia*). Što usluga više koristi ove tehnologije, to će se smatrati zrelijom.

*Slika 5. Razine zrelosti prema Richardsonovom modelu*




Izvor: https://martinfowler.com/articles/richardsonMaturityModel.html

Iako se Richardsonov model zrelosti često smatra ezoteričnijim u odnosu na poznatije konkurente, on može poslužiti kao smjernica za postizanje doista cjelovitih i korisnih API-ja.

## 5.1. Razina 0 - The Swamp of POX

Nulta razina ne koristi nijednu od URI, HTTP metoda i HATEOAS mogućnosti. Ove usluge imaju jednu URI i koriste jednu HTTP metodu (obično POST). Na primjer, većina usluga temeljenih na Web uslugama (WS - *) koristi jedan URI za identifikaciju

krajnje točke, a HTTP POST za prijenos opterećenja temeljenog na SOAP-u, učinkovito ignorirajući ostatak HTTP metoda.

Slično tome, usluge temeljene na XML-RPC-u koje šalju podatke kao Plain Old XML (POX). Ovo je najprimitivniji način izgradnje SOA aplikacija jednom metodom POST i korištenjem XML-a za komunikaciju između usluga.

## 5.2. Razina 1 - Resursi

Od tri moguća faktora zrelosti prva razina zrelosti koristi URI-je. Ove usluge koriste mnogo URI-ova, ali samo jednu HTTP metodu - uglavnom HTTP POST. Svaki resurs zasebno je identificiran pomoću jedinstvenog URI-ja što ih čini boljim od nulte razine.

Dizajn API-ja na razini 1 odnosi se na korištenje različitih URL-ova za interakciju s različitim resursima u aplikaciji. Umjesto da se prođe sve kroz http://example.org/articles, može se razlikovati http://example.org/article/1 i http://example.org/article/2.

## 5.3. Razina 2 – HTTP metode

Druga razina zrelosti koristi URI i HTTP metode. Usluge druge razine pružaju brojne resurse koji se mogu uputiti putem URI-ja. Takve usluge podržavaju nekoliko HTTP metoda na svakom izloženom izvoru - Stvaranje, čitanje, ažuriranje i brisanje (CRUD) usluga. Ovdje se stanje resursa, koje obično predstavljaju poslovne subjekte, može manipulirati mrežom. Ovdje dizajner usluge očekuje da će ljudi uložiti neki napor u savladavanje API-ja uglavnom čitanjem isporučene dokumentacije.

Razina 2 je dobar primjer REST principa, koji se zalažu za upotrebu različitih metoda temeljenih na HTTP metodama zahtjeva, a sustav može imati više resursa.

## 5.4. Razina 3 – Hypermedia

Treća razina zrelosti koristi sve tri značajke, tj. URI, HTTP metode i HATEOAS. Ovo je najzrelija razina Richardsonovog modela koji potiče lako otkrivanje i omogućava sam po sebi razumljiv odgovor korištenjem HATEOAS-a. Usluga vodi potrošače na tragu resursa, što uzrokuje prijelaze stanja aplikacije.

# 6. HATEOAS

HATEOAS (Hypermedia as the Engine of Application State) ograničenje je REST arhitekture aplikacija koja zadržava jedinstvenu arhitekturu RESTful stila u odnosu na većinu ostalih mrežnih aplikacijskih arhitektura. Izraz *hypermedia* odnosi se na bilo koji sadržaj koji sadrži poveznice na druge oblike medija, kao što su slike, filmovi i tekst. Ovaj arhitektonski stil omogućuje korištenje hipermedijske veze u sadržaju odgovora kako bi klijent mogao dinamički prijeći na odgovarajući resurs prelazeći hipermedijske veze. To je konceptualno isto kao i web korisnik koji se kreće po web stranicama klikom na odgovarajuće hiperveze kako bi postigao konačni cilj.

Na primjer, dolje navedeni JSON odgovor može biti od API-ja poput HTTP GET-a http://api.domain.com/management/employees/4

*Slika 6. Primjer sadržaja odgovora HATEOAS-a*


```
{
    "FirstName": "Razvan",
    "LastName": "Stetcu",
    "BirthDate": "07-08-1994",
    "Links": [
        {
            "Href": "http://localhost:62782/Employee/4",
            "Rel": "get_employee",
            "Method": "GET"
        },
        {
            "Href": "http://localhost:62782/Employee/4",
            "Rel": "delete_employee",
            "Method": "DELETE"
        },
        {
            "Href": "http://localhost:62782/Employee",
            "Rel": "edit_employee",
            "Method": "PUT"
        }
    ]
}
```


Izvor: https://wayfare.ro/ive-used-hateoas-internal-project

U prethodnom primjeru, odgovor koji poslužitelj vraća sadrži hipermedijske veze na resurse zaposlenika 4 / zaposlenike, koje klijent može preći da bi pročitao zaposlenike koji pripadaju odjelu. Prednost gore navedenog pristupa je u tome što

hipermedijske veze vraćene s poslužitelja pokreću stanje aplikacije, a ne obrnuto. Ne postoji opće prihvaćeni format za predstavljanje veza između dva izvora u JSON-u.

## 6.1. HATEOAS Implementacija

Kada se posjeti web stranica, najprije se otvori početna stranica koja predstavlja neke snimke i veze do drugih dijelova web stranice. Klikom na njih dobije se više informacija zajedno s više povezanih linkova koji su relevantni za kontekst.

Slično klijentskoj interakciji s web stranicom, REST klijent doseže početni URI API i koristi veze na poslužitelju kako bi dinamički otkrio dostupne radnje i pristupio resursima koji su mu potrebni. Klijent ne mora imati prethodno znanje o usluzi ili različite korake koji su uključeni u tijek rada. Uz to, klijenti više ne moraju ručno unositi URI strukture za različite resurse. To omogućuje poslužitelju da promijeni URI kako se API razvija bez problema, bez da utječe na radnju klijenta.

HATEOAS donosi ovaj koncept prikazivanja veza do određenog resursa RESTful uslugama. Kad se vrate detalji određenog resursa, vraćaju se i veze do radnji koje se mogu izvoditi na resursu, kao i veze na povezane resurse. Ako potrošač usluga može upotrebljavati veze iz odgovora za obavljanje transakcija, ne bi trebalo unositi sve veze unutar koda. (Karanam, 2019)

Prethodna interakcija API-ja moguća je samo pomoću HATEOAS-a. Svaki REST okvir pruža vlastiti način stvaranja HATEOAS veza korištenjem mogućnosti okvira. Postoje dva popularna formata za specificiranje JSON REST API hipermedijskih veza.

### 6.1.1. RFC 5988 (web povezivanje)

RFC 5988 donosi okvir za izgradnju veza koji definira odnos između resursa na webu. Svaka veza u RFC 5988 sadrži sljedeća svojstva:

- Ciljni URI: Svaka veza treba sadržavati ciljane IRI-je (Internationalized Resource Identifier). Ovo je predstavljeno atributom *href*.
- Vrsta odnosa veze opisuje kako se trenutni kontekst (izvor) odnosi na ciljni resurs. To je predstavljeno atributom *rel.*
- Atributi za ciljni IRI uključuju *hreflang*, medij, naslov i vrstu i sve parametre veze proširenja.

### 6.1.2. JSON Hypermedia API Language (HAL)

JSON HAL je obećavajući prijedlog koji postavlja konvencije za izražavanje hipermedijskih kontrola, poput veza, s JSON ili XML-om. HAL je jednostavan format koji daje dosljedan i jednostavan način hiperveze između resursa u API-ju. (Kelly, 2013) Svaka veza u HAL-u može sadržavati sljedeća svojstva:

- Ciljani URI: Ukazuje ciljani URI. Ovo je predstavljeno atributom *href.*
- Odnos veze: Vrsta odnosa veze opisuje kako se trenutni kontekst odnosi na ciljni resurs. To je predstavljeno atributom *rel*.
- Tip: Označava očekivanu vrstu medija resursa. To je predstavljeno atributom *type.*

# 7. OPĆENITO O RAPiD SUSTAVU

Skladišta kao objekti predstavljaju vrlo važan segment u opskrbnom lancu. Posjedovanjem neadekvatne organizacije, opreme ili zaposlenika dolazi do nastanka nepotrebnih troškova. Skladišni procesi moraju se izvršavati efektivno kako bi se postigla potrebna razina zaštite robe, izbjegla nepotrebna kašnjenja i čekanja robe te optimizacija troškova. (Jangjel, 2018)

U prijem robe pripadaju poslovi i zadaci: istovar, kontrole i evidencije primanja robe u skladištu. Naime, roba se zaprima na temelju prijevoznog dokumenta koji može biti: teretnica (konsoman) kod pomorskog, tovarni list kod željezničkog prijevoza, otpremnica kod cestovnog, sprovednica kod dopreme robe poštom i zrakoplovni tovarni list zračnog. U skladište se roba zaprima na temelju kvantitativne i kvalitativne kontrole. (Garc, 2017)

RAPiD[21] je sustav kreiran da optimizira postupak skladištenja robe, te znatno ubrza prijem robe. Skoro svi danas imaju smartphone, stoga kako bi se optimizirao, te ubrzao i olakšao postupak prijema robe napravljen je sustav koji se sastoji od dva dijela. Sastoji se od mobilne aplikacije i web aplikacije. Za izradu mobilne aplikacije korišten je Swift 5.0 u Xcode razvojnom okruženju. Web aplikaciju može se dodatno podijeliti na sučelje i pozadinu. Laravel kao jedan od najpopularnijih PHP okvira korišten je za pozadinu, dok se za sučelje koristi React. Kao razvojno okruženje za web aplikaciju korišten je PHP Storm. Sustav je namijenjen svim proizvodnim i trgovačkim poduzećima koja imaju skladišta. Mogu ga koristiti dva tipa korisnika:

- Poduzeće
- Zaposlenik

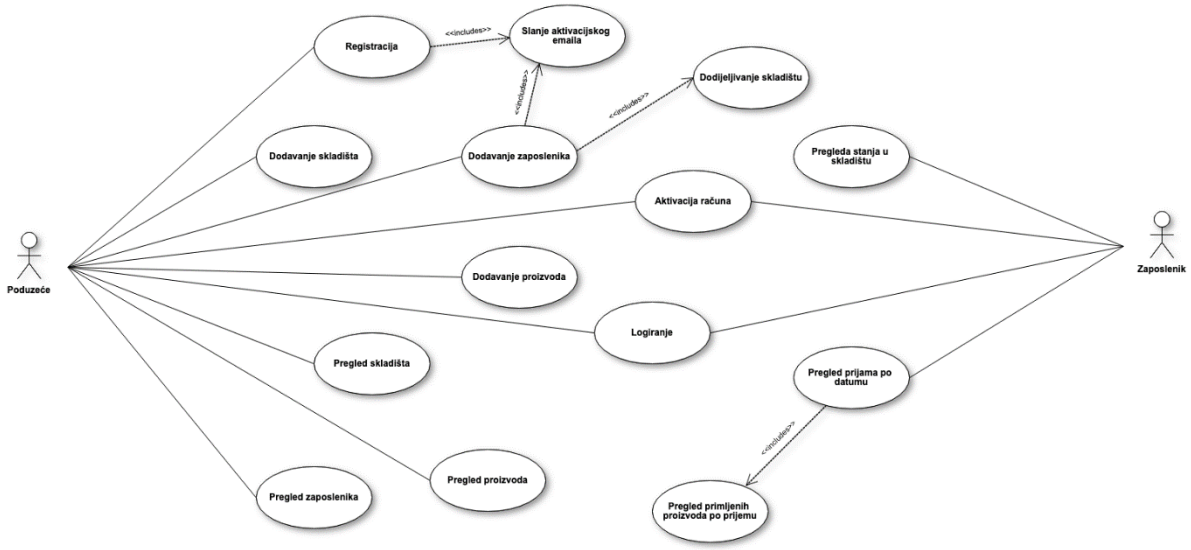
U sljedećem poglavlju će se ukratko opisati funkcionalnosti sustava, koje će se dodatno razraditi u uputama gdje će se detaljnije vizualno predstaviti, te opisati korištenje sustava.

[21] RAPiD (Restful Application of Products in Demand) – naziv sustava u sklopu ovog rada

# 8. FUNKCIONALNOSTI

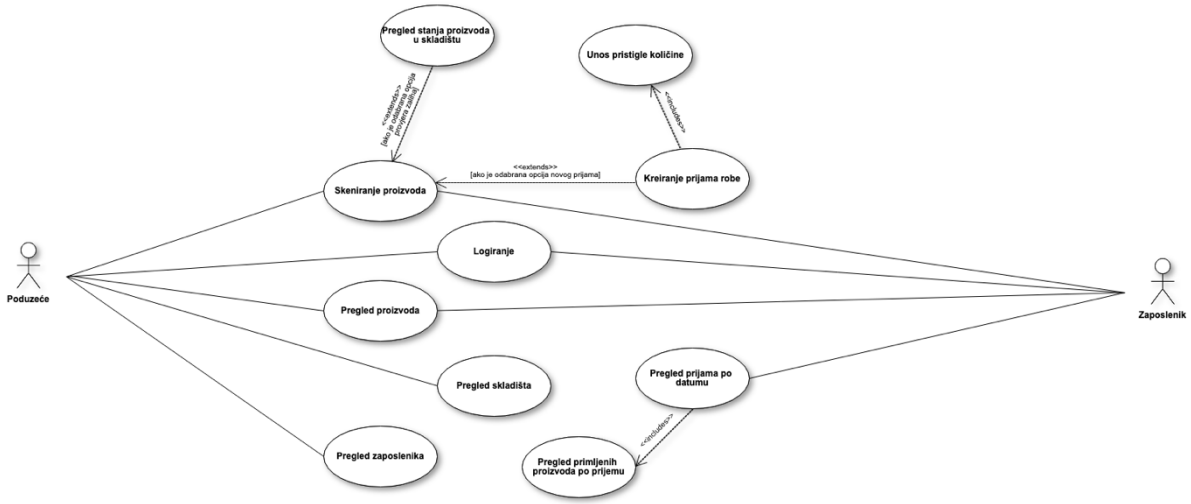
Kako bi se sustav započeo koristiti, najprije se poduzeće mora registrirati na web aplikaciji, gdje će nakon ispunjenog obrasca s potrebnim podacima dobiti mail s kojim se aktivira račun za korištenje sustava.

*Slika 7. Use - case dijagram: Mogućnosti na Web aplikaciji*




Izvor: izradio autor

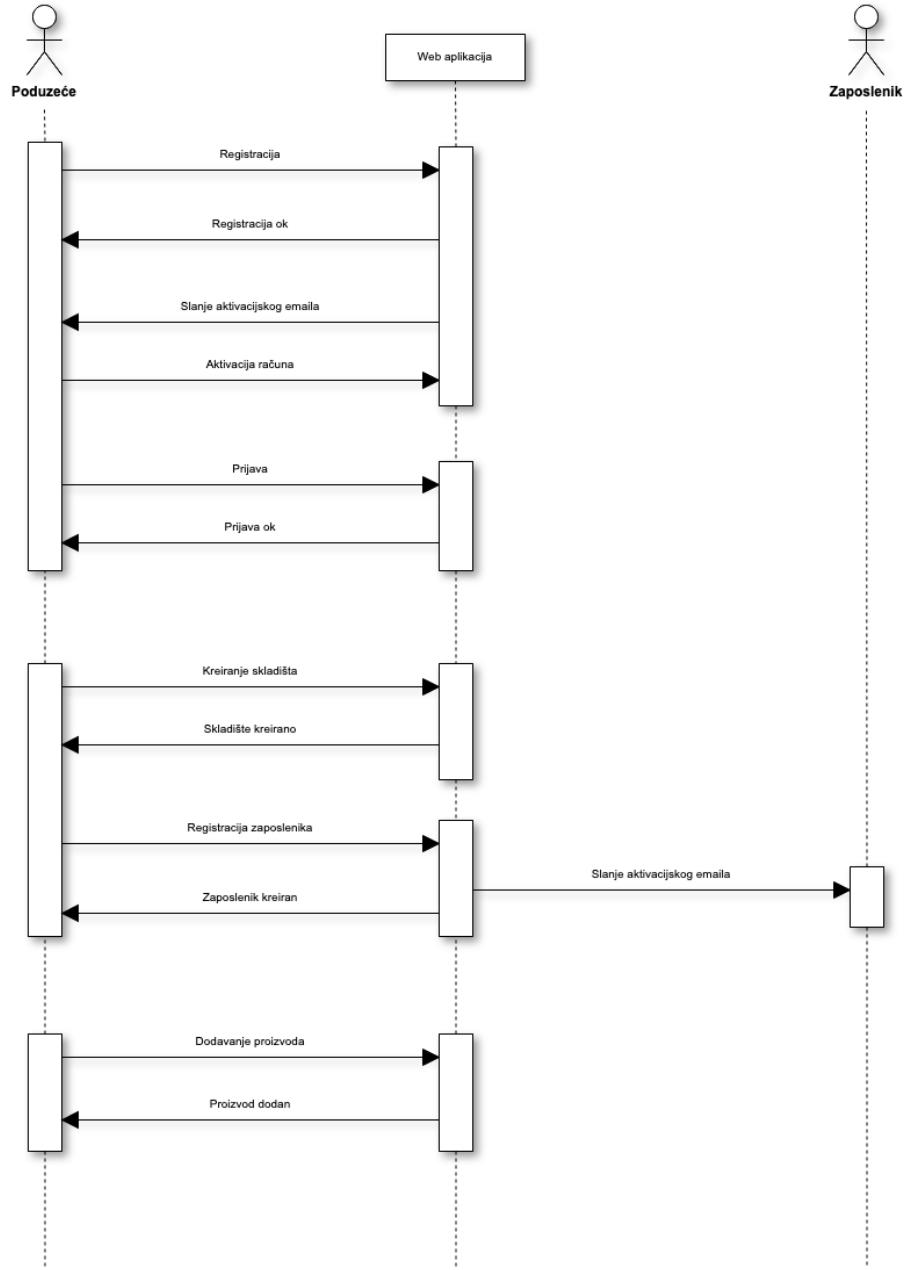
*Slika 8. Use - case dijagram: Mogućnosti na Mobilnoj aplikaciji*




Izvor: izradio autor

## 8.1. Poduzeće

*Slika 9. Sekvencijski dijagram: Tijek korištenja web aplikacije za poduzeće*




Izvor: izradio autor

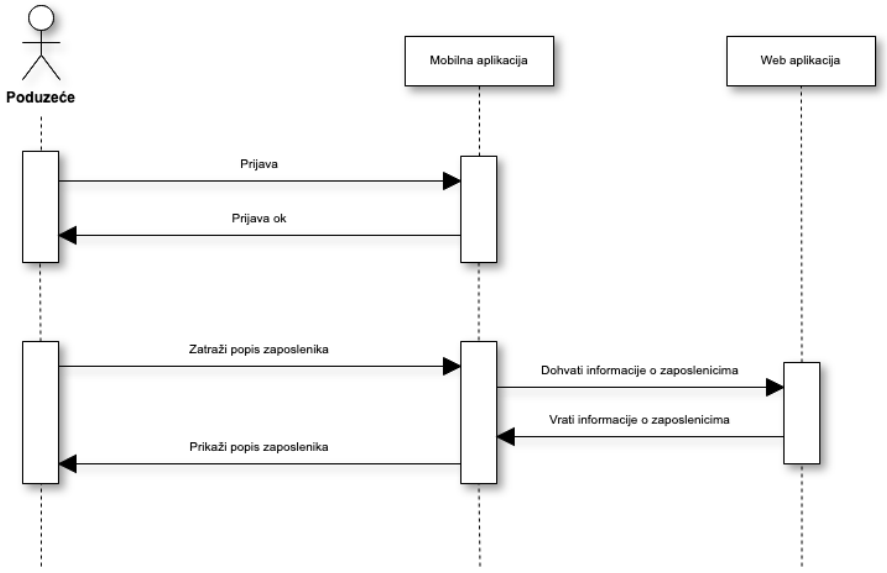
Nakon uspješne registracije i aktivacije računa poduzeće se može logirati u sustav. Kako bi se započelo korištenje sustava, prvo što je bitno za poduzeće je da doda skladišta koja poduzeće posjeduje. Kad se uspješno dodaju skladišta u sustav, poduzeće može dodavati zaposlenike i dodijeliti ih određenom skladištu u kojem rade.

Zaposlenici će nakon toga primiti mail s kojim će aktivirati svoj račun, te će u mail-u dobiti privremenu zaporku koju zaposlenik može u bilo kojem trenutku promijeniti na web aplikaciji. Sve opisano se može vidjeti na Slici 9 gdje je prikazan tijek korištenja web aplikacije. Uz dodavanje skladišta i zaposlenika, poduzeće dodaje i proizvode kojima raspolaže. Za svaki proizvod se mora:

- upisati naziv proizvoda
- upisati barkod proizvoda
- odabrati tip proizvoda
- opcionalno odabrati sliku proizvoda

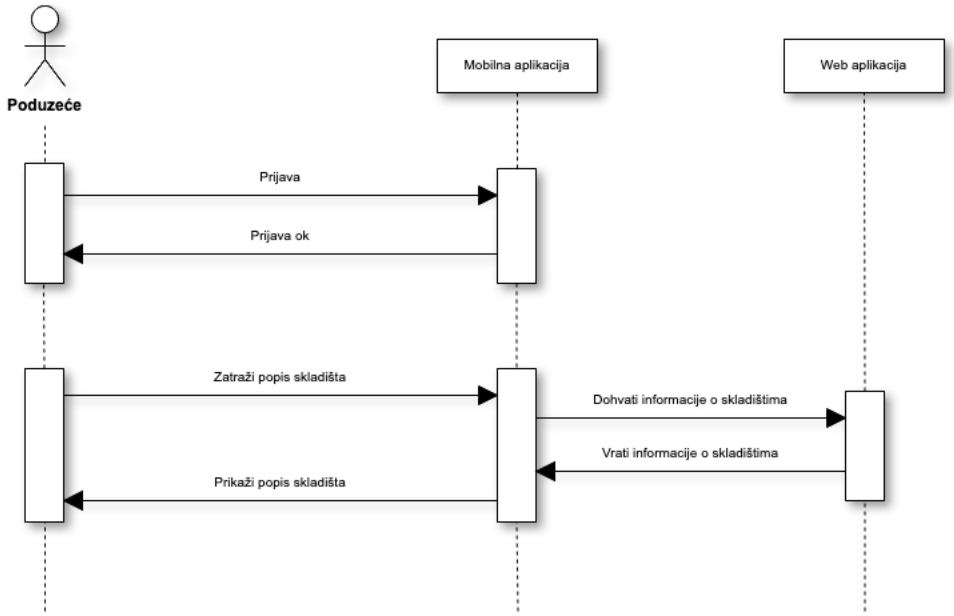
Kako bi se olakšalo upisivanje barkoda, postoji mogućnost korištenja mobilne aplikacije za skeniranje bar koda koji se automatski pojavljuje u polju za upisati barkod. Uz korištenje web aplikacije, poduzeće također ima na raspolaganju korištenje mobilne aplikacije. Kao što je već spomenuto umjesto ručnog upisivanja svakog barkoda pri dodavanju proizvoda na web aplikaciji može se skenirati barkod proizvoda koji će se nakon toga poslati na web, te će web aplikacija imati pristup skeniranom barkodu. Time se znatno ubrzava i olakšava dodavanje novih proizvoda u sustav. Kroz sljedećih nekoliko slika može se vidjeti tijek korištenja mobilne aplikacije, gdje je svaka mogućnost zasebno prikazana.

*Slika 10. Sekvencijski dijagram: Prikazivanje popisa zaposlenika u mobilnoj aplikaciji*



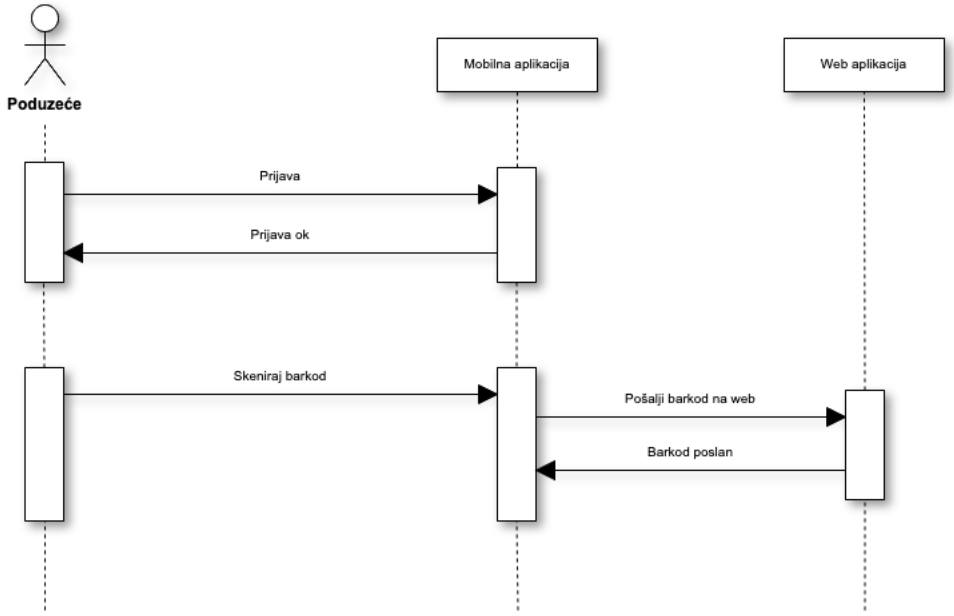

Izvor: izradio autor

*Slika 11. Sekvencijski dijagram: Prikazivanje popisa skladišta u mobilnoj aplikaciji*




Izvor: izradio autor

*Slika 12. Sekvencijski dijagram: Skeniranje barkoda za dodavanje proizvoda na web aplikaciji*

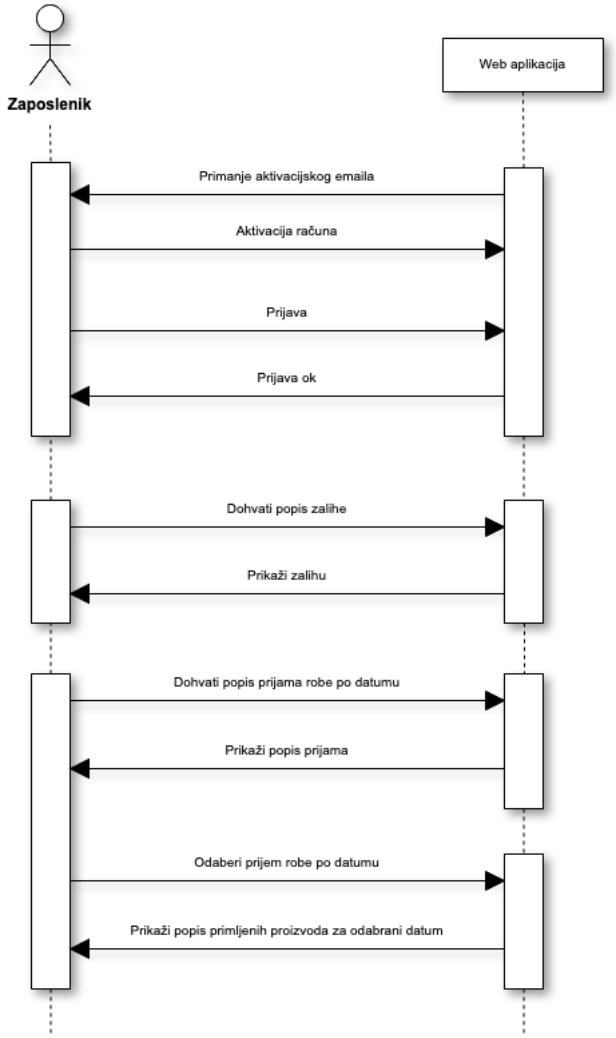



Izvor: izradio autor

## 8.2. Zaposlenik

Zaposlenici nakon aktivacije svog računa mogu početi koristiti web i mobilnu aplikaciju. Na web aplikaciji mogu pratiti stanje na zalihi, te pregledavati prijem robe po datumu i vremenu što se može vidjeti na Slici 13.

*Slika 13. Sekvencijski dijagram: Tijek korištenja web aplikacije za zaposlenika*



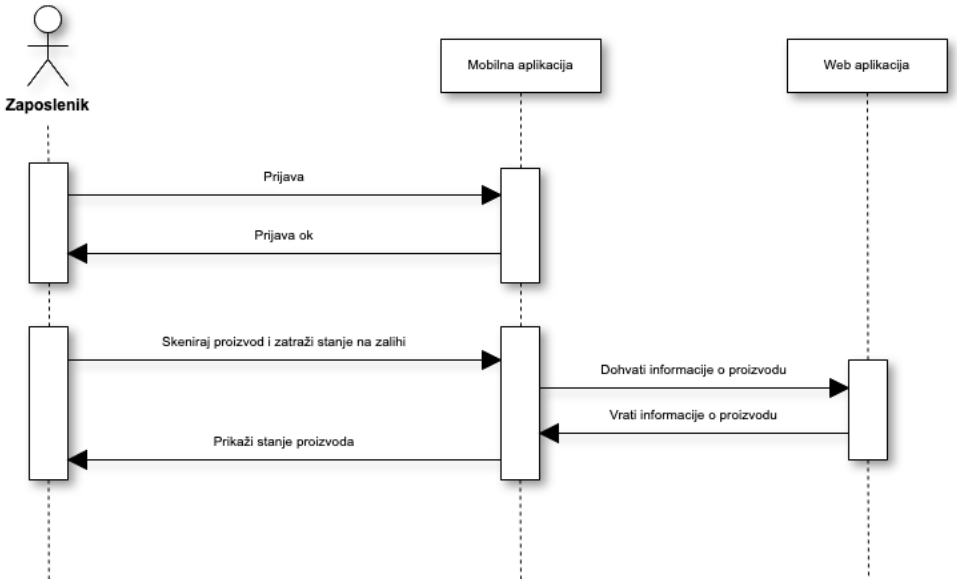

Izvor: izradio autor

Također te stvari zaposlenik ima dostupno u aplikaciji, stoga je zaposleniku dovoljno imati aplikaciju koja ima i dodatne mogućnosti poput:

- kreiranje novog prijema robe te skeniranje i dodavanje količine robe koja je pristigla
- skeniranje proizvoda da bi se vidjelo stanje na zalihi

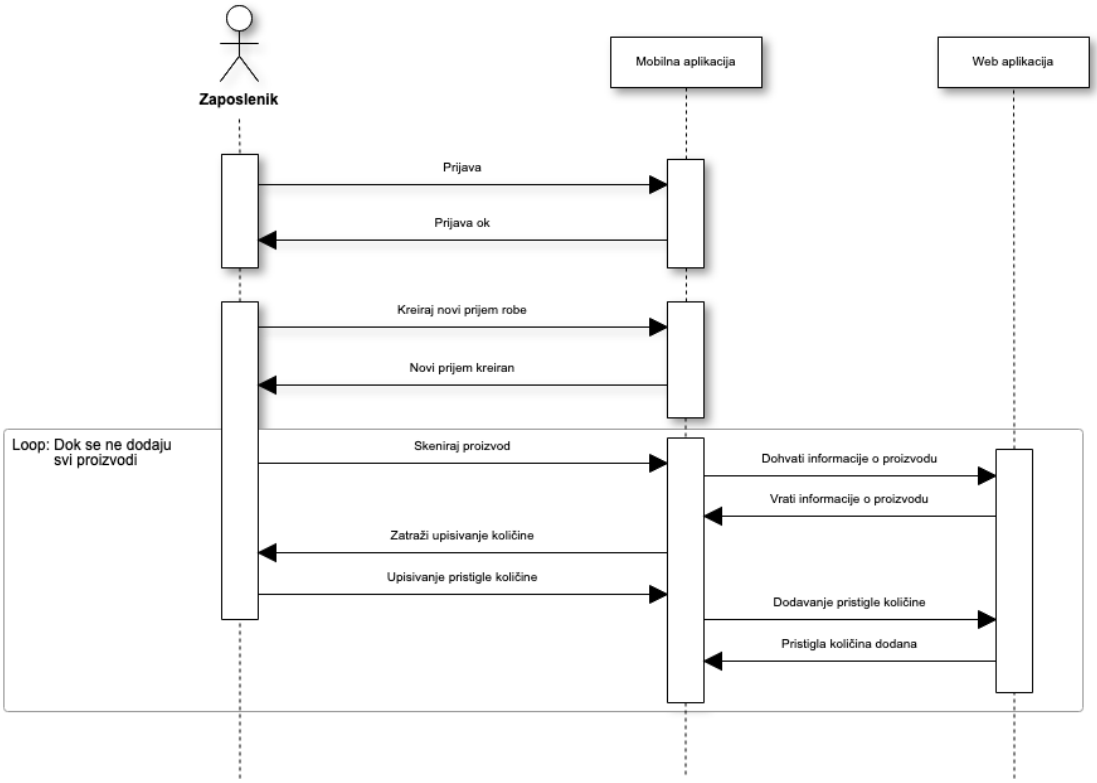
Sve mogućnosti koje zaposlenik može koristiti prikazane su kroz nekoliko sljedećih slika.

*Slika 14. Sekvencijski dijagram: Skeniranje proizvoda da se sazna stanje na skladištu*



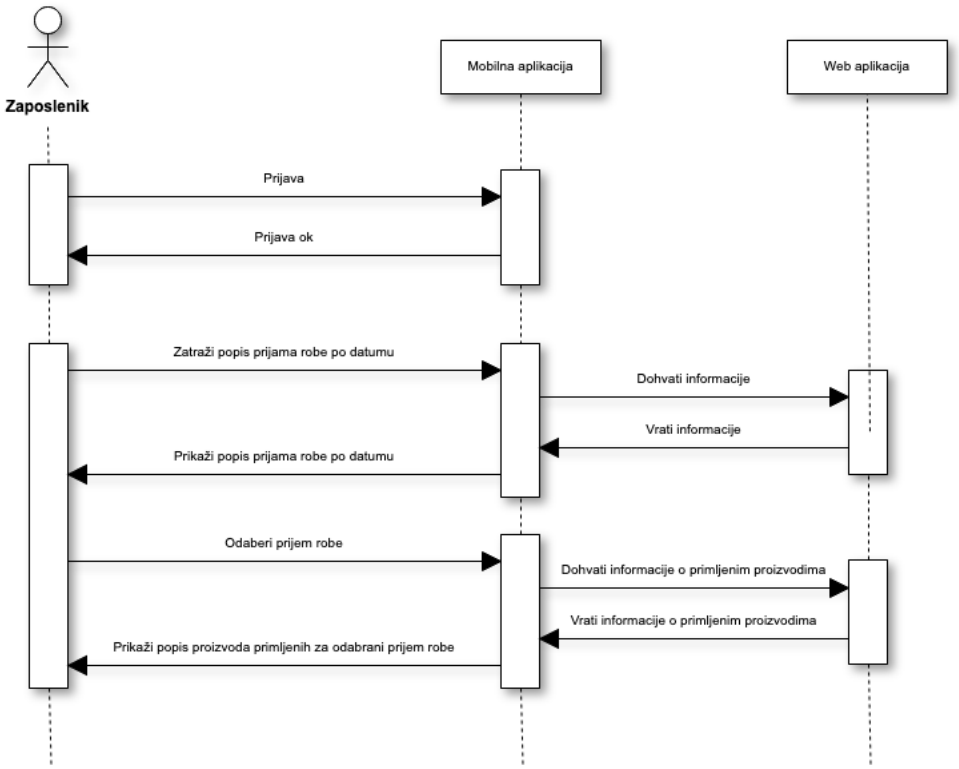

Izvor: izradio autor

*Slika 15. Sekvencijski dijagram: Kreiranje novog prijema robe*




Izvor: izradio autor

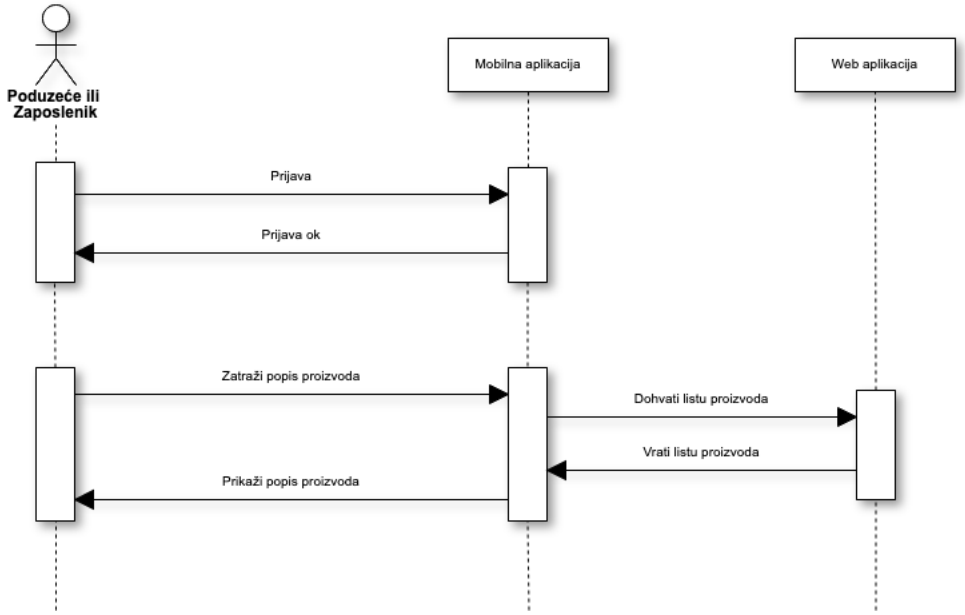
*Slika 16. Sekvencijski dijagram: Prikaz prijema robe u mobilnoj aplikaciji*




Izvor: izradio autor

Ono što je zajedničko za poduzeće i zaposlenika je da oboje mogu u aplikaciji vidjeti popis proizvoda s kojima poduzeće raspolaže, što je prikazano na Slici 17.

*Slika 17. Sekvencijski dijagram: Prikaz proizvoda u mobilnoj aplikaciji*




Izvor: izradio autor

# 9. IMPLEMENTACIJA

U ovom poglavlju bit će objašnjeno što se koristilo pri izradi praktičnog rada te na koji način funkcionira sustav i pojedine značajke. Za izradu sučelja web aplikacije korišten je React okvir. React ne može komunicirati s bazom direktno. Iz tog razloga React komunicira s pozadinom koja je napravljena u Laravelu.

## 9.1. Baza podataka

Za bazu podataka korišten je odabran MySQL zbog jednostavnosti i već prethodnog znanja u korištenju ovog relacijskog sustava. Postoje dvije glavne tablice u bazi podataka. Prva je Migrations koja sadrži sve migracije iz Laravel okvira koje su kreirane za potrebe projekta, tako da se na serveru može pokrenuti naredba koja kreira sve potrebne tablice te njihove relacijske veze. Druga glavna tablica je Users, koja je za ovaj sustav napravljena tako da sadrži dva tipa korisnika, a to su kompanija i zaposlenik. Stoga veza tablice prikazana na Slici 18. koja se povezuje na sebe označava povezanost kompanije i zaposlenika, tj. jedna kompanije može imati nula ili više zaposlenika, dok jedan zaposlenik može pripadati samo jednoj kompaniji. Ovisno koji tip korisnika se dodaje, popunjavaju se različiti stupci na različite načine. Na primjer, najprije se doda kompanija te se za nju ispune:

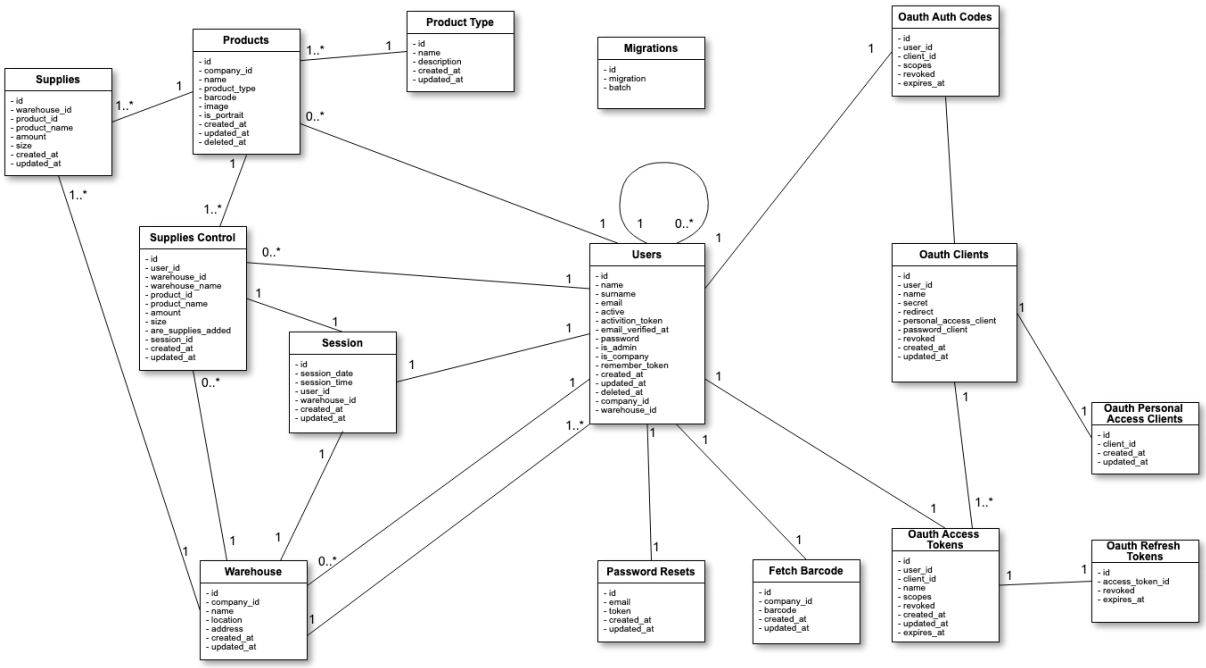
- name – naziv kompanije
- surname – sjedište kompanije (mjesto)
- is_company – postavlja se na *true*

Uz gore navedena polja također se popunjava email i password koji je isti bez obzira na tip korisnika. Nakon što se doda kompanija, ona može dodavati svoje zaposlenike, te se onda popunjavaju ova polja:

- name – ime zaposlenika
- surname – prezime zaposlenika
- is_company – postavlja se na *false*
- company_id – ovo je *foreign key* koji pokazuje kojoj kompaniji pripada zaposlenik

- warehouse_id – odabire se skladište u koji je smješten zaposlenik

*Slika 18. Baza podataka - Relacije između tablica za RAPiD sustav*




Izvor: izradio autor

Na desnoj strani Slike 18 nalaze se Oauth tablice koje su kreirane za autorizaciju korisnika, te poboljšanje sigurnosti. S lijeve strane  Users tablice prikazane su tablice koje koristi sustav za dobivanje i spremanje informacija potrebnih za rad sustava. Jedina relacija koja može biti zbunjujuća je dvostruka veza između tablica Users i Warehouse.

S obzirom da u tablici postoje dva tipa korisnika, ovisno koji je korisnik, povezanost između te dvije tablice se razlikuje. U slučaju da je korisnik kompanija tablice su povezane na način da jedan korisnik, tj. kompanija posjeduje nula ili više skladišta. Dok u slučaju da je zaposlenik, tada je povezano na način da jedno skladište ima jednog ili više zaposlenika, tj. jedan ili više zaposlenika pripadaju jednom skladištu.

## 9.2. Web aplikacija - Backend

Sve rute koje se koriste nalaze se u mapi routes u web.php i api.php datotekama. U web datoteci nalazi se samo početna ruta preko koje se dohvaća prikaz web

stranice, sve ostale poveznice za snalaženje i preusmjeravanje kroz web stranicu nalaze se na sučelju. Osim tih ruta u API datoteci nalaze se sve rute koje komuniciraju s bazom podataka.

*Slika 19. Prikaz svih ruta*

```
Route::group(['middleware' => ['json.response']], function () {

    // public routes
    Route::post( uri: '/login',  action: 'Auth\AuthController@login')->name( name: 'login.api');
    Route::post( uri: '/register',  action: 'Auth\AuthController@register')->name( name: 'register.api');
    Route::get( uri: '/signup/activate/{token}',  action: 'Auth\AuthController@signupActivate');
    Route::post( uri: '/create',  action: 'Auth\PasswordResetController@create');
    Route::get( uri: '/password/find/{reset_token}',  action: 'Auth\PasswordResetController@find');
    Route::post( uri: '/reset',  action: 'Auth\PasswordResetController@reset');
    Route::post( uri: '/refresh',  action: 'Auth\AuthController@refresh')->name( name: 'refresh.api');

    // private routes
    Route::middleware( middleware: 'auth:api')->group(function ($router) {

        Route::get( uri: '/logout',  action: 'Auth\AuthController@logout')->name( name: 'logout');
        Route::post( uri: '/info',  action: 'Auth\AuthController@getUserInfo');
        Route::post( uri: '/change_password',  action: 'Auth\PasswordResetController@changePassword');
        Route::post( uri: '/upload/{id}',  action: 'Api\ImagesController@uploadImage');
        Route::delete( uri: '/delete_image/{id}/{name}',  action: 'Api\ImagesController@deleteImage');
        Route::resource( name: 'get_barcode',  controller: 'Api\BarcodeController')->only(['store', 'show']);

        RouteHelper::make($router)
            ->resource( name: 'users',  controller: 'Api\UsersController')
            ->searchable()
            ->done()

            ->resource( name: 'products',  controller: 'Api\ProductsController')
            ->searchable()
            ->done()

            ->resource( name: 'product_type',  controller: 'Api\ProductTypesController')
            ->searchable()
            ->done()

            ->resource( name: 'supplies',  controller: 'Api\SuppliesController')
            ->searchable()
            ->done()

            ->resource( name: 'transactions',  controller: 'Api\SuppliesControlController')
            ->searchable()
            ->done()

            ->resource( name: 'session',  controller: 'Api\SessionController')
            ->searchable()
            ->done()

            ->resource( name: 'warehouse',  controller: 'Api\WarehouseController')
            ->searchable()
            ->done();

    });
});
```

Izvor: izradio autor

Kao što se može vidjeti na Slici 19 sve rute se nalaze unutar grupe koja ima json.responce middleware što znači da su automatski svi odgovori u JSON formatu. Route::middleware služi kako bi određene rute bile zaštićene, tj. da samo autorizirani korisnici mogu pristupiti tim rutama. U ovom slučaju koristi se auth:api.

Osnovna funkcionalnost svakog sustava je login i registracija. Laravel već olakšava provjeru autentičnosti putem tradicionalnih oblika za prijavu, no što je s API-jevima? API-ji obično koriste tokene za autentifikaciju korisnika i ne održavaju stanje sesije između zahtjeva. Laravel omogućuje laganu provjeru autentičnosti pomoću

Laravel Passporta, koji omogućuje potpunu implementaciju OAuth2 poslužitelja. Sve rute unutar „auth:api“ *middlewarea* koriste Laravel Passport autorizaciju.

OAuth 2 je okvir za autorizaciju koji omogućuje aplikacijama da dobiju ograničeni pristup korisničkim računima na HTTP servisu, kao što su Facebook, GitHub i DigitalOcean. (Anicas, 2014)

OAuth ne dijeli podatke o zaporkama već umjesto toga koristi tokene za autorizaciju za dokazivanje identiteta između potrošača i davatelja usluga. OAuth je protokol za provjeru autentičnosti koji omogućuje da se odobri interakcija jedne aplikacije s drugom za specifičnog korisnika bez davanja zaporke.

### 9.2.1. OAuth 1.0 vs OAuth 2.0

OAuth 2.0 je potpuni redizajn u odnosu na OAuth 1.0, i nisu kompatibilni stoga se Oauth 2.0 može smatrati kao potpuno novi protokol. OAuth 2.0 fokusira se na jednostavnost između klijenta i razvojnih programera na način da pruža specifični tijek autorizacije za web aplikacije, desktop aplikacije, mobilne aplikacije i ostale uređaje. (OAuth)

OAuth 1.0 uvelike se temeljio na dva postojeća vlasnička protokola: Flickr-ov autorizacijski API i Googleov AuthSub. Rad koji je postao OAuth 1.0 bilo je najbolje rješenje temeljeno na tadašnjem iskustvu u implementaciji. Tijekom nekoliko godina mnoga poduzeća koja su koristile OAuth 1.0 API-je i mnogi programeri koji su pisali kod za konzumaciju API-ja, shvatili su gdje protokol daje izazov ljudima. Nekoliko je specifičnih područja identificirano da se trebaju poboljšati jer su ili ograničavali sposobnosti API-ja ili su bili previše izazovni za implementaciju.

OAuth 2.0 predstavlja višegodišnju raspravu između širokog spektra kompanija i pojedinaca, uključujući Yahoo !, Facebook, Salesforce, Microsoft, Twitter, Deutsche Telekom, Intuit, Mozilla i Google.

Tamo gdje OAuth 2.0 definira četiri uloge (klijent, poslužitelj autorizacije, poslužitelj resursa i vlasnik resursa), OAuth 1 koristi drugačiji skup izraza za ove uloge. OAuth 2.0 "klijent" poznat je i kao "potrošač", "vlasnik resursa" poznat je i kao "korisnik", a "server poslužitelja" je poznat i kao "pružatelj usluga". OAuth 1 također izričito ne odvaja uloge poslužitelja resursa i poslužitelja autorizacije.

### 9.2.2. Dodatni paketi

Za kreiranja tablica na bazi Laravel ima rješenje koristeći migracije (*migrations)*. Migracije omogućavaju da se shema baze podataka aplikacije jednostavno izradi ili izmijeni. Primjer migracije prikazan je na Slici 20.

*Slika 20. Primjer migracije*

```
class CreateProductsTypeTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create( table: 'products_type', function (Blueprint $table) {
            $table->increments( column: 'id');
            $table->string( column: 'name');
            $table->string( column: 'description')->nullable();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists( table: 'products_type');
    }
}
```

Izvor: izradio autor

Mailtrap je lažni SMTP (Simple Mail Transfer Protocol) server koji se koristi za razvoj aplikacija u fazi testiranja za slanje elektroničke pošte. Sustav nudi besplatnu registraciju, te funkcionira na način da se poslani mailovi dostavljaju u virtualni *inbox*. U *inboxu* se može vidjeti pošiljatelj, primatelj maila te njegov sadržaj. Jedini nedostatak je što besplatna licenca daje korisniku na korištenje jednog virtualnog *inboxa* koji može primiti samo 50 mailova, što može znatno ograničavati testiranje u slučaju da se testira više korisnika.

## 9.3. Web aplikacija - Frontend

Kada se posjeti web stranicu, najprije se dolazi na glavni URL koji dohvaća podatke iz MainComponent komponente prikazane na Slici 21. Ona ovisno o URL-u preusmjerava na ostale komponente. Izuzetak je „/dashboard“ gdje se za svaki URL

koji kreće sa *dasboard* najprije provjerava je li URL autoriziran, tj. je li korisnik prijavljen i ima li pristup korištenju. Provjera je prikazana na Slici 22 gdje ako je korisnik logiran, dopušta mu pristup te ga preusmjerava na zatraženi URL. U slučaju da nije logiran preusmjerava ga na Login komponentu.

*Slika 21. Glavna komponenta Korisničkog sučelja*

```
class MainComponent extends Component{

    render () {
        return(
            <Provider store={store}>
                <PersistGate loading={null} persistor={persistor}>
                    <Router>
                        <Switch>
                            <Route exact path='/' component={Homepage}/>
                            <Route path='/login' component={Login}/>
                            <Route path='/register' component={Register}/>
                            <Route path='/forgotpassword' component={Forgot}/>
                            <Route path='/password/reset/:token' component={Reset}/>
                            {/*<Route path='/successful' component={PasswordResetSuccessful}/>*/}
                            <Route path='/signup/activate/:token' component={ActivateAccount}/>
                            <AuthRoute path='/dashboard' component={Dashboard}/>
                            <Route path='*' component={NotFound}/>
                        </Switch>
                    </Router>
                </PersistGate>
            </Provider>
        )
    }
}

ReactDOM.render(<MainComponent />, document.getElementById( elementId: 'app'));
```

Izvor: izradio autor

*Slika 22. Provjera autorizacije korisnika*

```
const checkAuth = () =>{

    const state = store.getState().user;

    if (state.access_token){
        if(Date.now() > state.expires){
            store.dispatch(refreshToken(state.refresh_token));
            return true
        }else{
            return true
        }
    }else{
        return false;
    }

};


function AuthRoute({ component: Component, ...rest }) {


  return (
    <Route
      {...rest}
      render={props =>
        checkAuth() ? (
          <Component {...props} />
        ) : (
          <Redirect
            to={{
              pathname: "/login",
              state: { from: props.location }
            }}
          />
        )
      }
    />
  );
}
```

Izvor: izradio autor

Kao primjer *frontenda* prikazat će se login stranica. Login komponenta pri učitavanju provjerava je li korisnik prethodno logiran. U slučaju da je već logiran, preusmjerava na *dashboard* aplikacije. Kada nije logiran, komponenta prikazuje login formu na korisničkom sučelju gdje korisnik mora upisati email i zaporku. Nakon što korisnik upiše potrebne podatke, podaci se šalju u pozadinu da bi se provjerili te se vrati odgovor. Komunikacija sa  pozadinomkoristi dodatni paket za React, a to je Redux. Nije potrebno koristiti Redux samo zato što je netko rekao da treba, potrebno je odvojiti malo vremena da se shvati potencijalne prednosti i koristi od korištenja. (Redux, 2018) Redux je JavaScript biblioteka otvorenog koda za upravljanje stanjem aplikacija. Najčešće se koristi s bibliotekama poput React ili Angular za izgradnju korisničkih sučelja. Da bi se koristio Redux kreira se *store* funkcija.

*Slika 23. Store funkcija*

```
import { createStore, applyMiddleware, compose } from "redux";
import thunk from 'redux-thunk';
import rootReducer from './reducers';
import { composeWithDevTools } from 'redux-devtools-extension';
import  {persistStore, persistReducer} from 'redux-persist';
import storage from 'redux-persist/lib/storage';

const initialState = {};
const persistConfig = {
    key: 'root',
    storage,
    whitelist: ['user']
};

const middleware = [thunk];

const store = createStore(
    persistReducer(persistConfig, rootReducer),
    composeWithDevTools(
        applyMiddleware(...middleware)
    )
);

const persistor = persistStore(store);

export { store, persistor };
```

Izvor: izradio autor

Nakon toga kreira se types.js datoteka kako bi se imalo pregledniji prikaz svih dostupnih akcija. Na slici 24. prikazane su dostupne akcije za login i registraciju.

*Slika 24. Primjer kreiranih akcija za korištenje Reduxa*

```
export const LOGIN = 'LOGIN';
export const GET_USER_INFO = 'GET_USER_INFO';
export const REGISTER = 'REGISTER';
export const LOGOUTUSER = 'LOGOUTUSER';
export const REFRESH_TOKEN = 'LOGOUT';
export const GET_ALL_USERS = 'GET_ALL_USERS';
export const LOGIN_ERROR = 'LOGIN_ERROR';
export const REGISTER_ERROR = 'REGISTER_ERROR';
export const RESET_PASSWORD = 'RESET_PASSWORD';
```

Izvor: izradio autor

Zatim se kreira *reducer* koji će upravljati stanjima aplikacije. Na Slici 25 prikazana je kompletna Redux funkcija za registraciju, te nekoliko dodatnih akcija.

*Slika 25. Primjer Redux reducera*

```
const initialState = {
    access_token: '', refresh_token: '', expires: '', user_info: '', new_user: '', password_reset_data: ''
};

export default function(state = initialState, action) {

    switch (action.type) {
        case LOGIN:
            return {
                ...state,
                access_token: action.access_token,
                refresh_token: action.refresh_token,
                expires: action.expires,
                user_info: action.user_info
            };
        case GET_USER_INFO:
            return {
                ...state,
                user_info: action.user_info
            };
        case LOGOUTUSER:
            return {
                ...state,
                access_token: '',
                refresh_token: '',
                expires: '',
                user_id: ''
            };
        case REGISTER:
            return {
                ...state,
                new_user: action.payload
            };
        case REFRESH_TOKEN:
            return {
                ...state,
                access_token: action.access_token,
                refresh_token: action.refresh_token,
                expires: action.expires
            };
        case RESET_PASSWORD:
            return {
                ...state,
                password_reset_data: action.password_reset_data
            };
        default:
            return state;
    }
};
```

Izvor: izradio autor

Kada je sve to napravljeno jedino što je preostalo je da se napravi funkcija koja će pozivati određenu akciju da se izvrši te spremi stanje, tako da se kasnije kroz cijelu aplikaciju može jednostavno dohvatiti podatke. Na Slici 26 prikazana je login funkcija koja koristi više akcija, međutim glavna akcija je „LOGIN“. Sa *dispatch*, *type: LOGIN* poziva se LOGIN akcija u loginReduceru koja zamjenjuje inicijalno stanje sa podacima proslijeđenim kroz *dispatch*.

*Slika 26. Login funkcija*

```
export const loginFunction = (data, history) => dispatch => {


    axios.post( url: '/api/login', data, header )
        .then( onfulfilled: user => {
            let user_id_header = {
                headers: {
                    Authorization: 'Bearer ' + user.data['access_token'],
                    Accept: 'application/hal+json'
                }
            };

            axios.post( url: '/api/info', data, user_id_header )
                .then( onfulfilled: user_id => {
                    // If request is good...

                    const users_data = JSON.parse(user_id.data);

                    axios.get( url: 'api/users/'+users_data.id, user_id_header )
                        .then( onfulfilled: user_info => {

                            dispatch({
                                type: LOGIN,
                                user_info: user_info.data,
                                access_token: user.data['access_token'],
                                refresh_token: user.data['refresh_token'],
                                expires: Date.now() + (user.data['expires_in'] * 1000)
                            });

                            history.push("dashboard");
                        })
                        .catch( onrejected: (error) => {
                            console.log('error ' + error);
                            dispatch({
                                type: ADD_FLASH_MESSAGE,
                                type_of_error: 'error',
                                text: error.response.data
                            });
                        });
                })
                .catch( onrejected: (error) => {
                    console.log('error ' + error);
                    dispatch({
                        type: ADD_FLASH_MESSAGE,
                        type_of_error: 'error',
                        text: error.response.data
                    });
                });
        })
```

Izvor: izradio autor

U login funkciji koriste se tri zahtjeva u pozadini. Najprije se šalju login podaci ispunjeni u formi na rutu /api/login koja vraća *access token*, *refresh token* i vrijeme kada *access token* prestaje vrijediti, gdje se nakon tog vremena koristi *refresh token* kako bi se dobio novi *access token* da se mogu dohvaćati potrebni podaci.

U ovom trenutku korisnik je prijavljen u sustav i može imati pristup svim autoriziranim podacima. Međutim *access token* se mora prosljediti uz svaki zahtjev koji ima *middleware* koji koristi Laravel Passport. Prosljeđuje se na način da se *access token* dodaje u *header* zahtjeva, što se može vidjeti na Slici 26 gdje je kreirana varijabla uer_id_header te se kasnije prosljeđuje sa zahtjevima. Sljedeći zahtjev se koristi da se dobije id korisnika kako bi se u zadnjem zahtjevu ove funkcije dobili svi potrebni podaci vezani uz prijavu korisnika. Kada je sve to prošlo uspješno, poziva se LOGIN akcija te se spremaju svi dobiveni podaci da se mogu kasnije jednostavno dohvatiti iz bilo kojeg dijela web aplikacije.

Nakon prijave ovisno koji je korisnik prijavljen, je li kompanija ili zaposlenik, korisnik se preusmjerava na početni URL *dashboarda* s kojeg kasnije može ići i na ostale URL-ove.

*Slika 27. Dashboard komponenta koja preusmjerava na ostale komponente ovisno o URL-u*

```
class Dashboard extends Component {
    render() {

        const location = this.props.match.url;

        return (
            <Fragment>
                    <Sidebar location={this.props}/>

                    <Main navWidthCollapsed={this.props.navWidthCollapsed}>
                        <Route exact path={`${location}`} component={DashboardHomepage}/>
                        <Route exact path={`${location}/register`} component={RegisterCompany}/>
                        <Route exact path={`${location}/companies`} component={Companies}/>
                        <Route  path={`${location}/employees/list`} component={Employees}/>
                        <Route  path={`${location}/employees/add`} component={RegisterEmployee}/>
                        <Route path={`${location}/products/list`} component={Products}/>
                        <Route path={`${location}/product/edit/:id`} component={ProductEdit}/>
                        <Route exact path={`${location}/products/add`} component={AddProducts}/>
                        <Route path={`${location}/warehouse/list`} component={Warehouses}/>
                        <Route exact path={`${location}/warehouse/add`} component={AddWarehouse}/>
                        <Route exact path={`${location}/supplies`} component={Supplies}/>
                        <Route exact path={`${location}/product/details/:id`} component={ProductDetails}/>
                        <Route exact path={`${location}/workflow`} component={Workflow}/>
                        <Route exact path={`${location}/transaction/:id`} component={Transaction}/>
                        <Route exact path={`${location}/settings`} component={Settings}/>
                    </Main>
            </Fragment>
        );
    }
}
```

Izvor: izradio autor

### 9.3.1. Dodatni paketi

Reactstrap je React biblioteka koja sadrži Botstrap 4 komponente. Reactstrap je sličan Bootstrapu, ali kao samostalne komponente obilježavanja  s imenima klase poput <Button>. U osnovi, sve potrebne komponente uvoze se kao hrpa elemenata koji su potrebni da bi se započela izgradnja korisničkog sučelja. Uz reactstrap korištena je biblioteka styled-components koja dodatno stilizira elemente.

*Slika 28. Primjer oblikovanja elementa korištenjem styled-components biblioteke*

```
const Main = styled.main`
    position: absolute;
    top: 0;
    right: 0;
    bottom: 0;
    left: ${props => props.navWidthCollapsed}px;
    transition: all .15s;
    padding: 0 20px;
    background: 'inherit';
    transition: background-color .35s cubic-bezier(.4, 0, .2, 1);
`;
```

Izvor: izradio autor

## 9.4. Mobilna aplikacija

Za izradu mobilne aplikacije korišteno je Xcode razvojno okruženje. Napisani kod mobilne aplikacije je u Swift 5 jeziku, trenutno zadnja verzija Swift jezika koji se konstantno razvija i poboljšava. Glavna funkcionalnost mobilne aplikacije je korištenje skenera za barkod, međutim ima i dodatne funkcionalnosti koje su već prikazane u prijašnjem poglavlju. Kako bi aplikacija mogla skenirati barkodove, koristi se već predefinirani AVFoundation okvir za audiovizualne medije.

AVFoundation ima AVCaptureSession koji pruža sučelje kamere potrebno za skeniranje koda. Sve što je potrebno napraviti je dodati prilagođenu verziju AVCaptureMetadataOutput u varijablu tipa AVCaptureSession i na istu pozvati startRunning ().

*Slika 29. Kreiranje AVCaptureMetadataOutput varijable*

```
let metadataOutput = AVCaptureMetadataOutput()

if (captureSession.canAddOutput(metadataOutput)) {
    captureSession.addOutput(metadataOutput)

    metadataOutput.setMetadataObjectsDelegate(self, queue: DispatchQueue.main)
    metadataOutput.metadataObjectTypes = [.ean8, .ean13, .pdf417]
} else {
    failed()
    return
}
```

Izvor: izradio autor

Kada se pokrene sesija, sesija će kontinuirano tražiti dane metadataObjectTypes i pozvat će delegatsku metodu ako se pronađe bilo koja vrsta podudaranja. Tada se može pozvati stopRunning() metoda na varijablu instanceSession ili se može nastaviti skenirati ovisno o potrebi.

*Slika 30. Delegatska metoda*

```
func metadataOutput(_ output: AVCaptureMetadataOutput, didOutput metadataObjects: [AVMetadataObject], from connection: AVCaptureConnection) {
    captureSession.stopRunning()

    if let metadataObject = metadataObjects.first {
        guard let readableObject = metadataObject as? AVMetadataMachineReadableCodeObject else { return }
        guard let stringValue = readableObject.stringValue else { return }
        AudioServicesPlaySystemSound(SystemSoundID(kSystemSoundID_Vibrate))
        found(code: stringValue)
    }
}

func found(code: String) {
    if(typeOfScan == "create") {
        scannerModel.sendBarcodeForNewProduct(barcode: code)
    } else if(typeOfScan == "add") {
        displayProductAddPopup(barcode: code)
    } else if(typeOfScan == "check") {
        scannerModel.checkSuppliesForProduct(barcode: code)
    }
}
```

Izvor: izradio autor

## 9.5. REST API

REST API razvijen je u Laravel okviru. Za potrebe projekta korišten je HAL-API paket kojeg je razvio Jari Schäfer baziranom na Laravel 5. Ovaj paket je znatno olakšao kreiranje RESTful API-ja. Kako bi se paket mogao koristiti unutar Laravel projekta, potrebno ga je instalirati pomoću *composera*, te dodati nekoliko stvari na različita mjesta u projektu. Nakon što se to napravi može se započeti koristiti paket te sve njegove pogodnosti. Na Slici 31 prikazane su kreirane rute u Laravelu koje koriste HAL-API paket kako bi se napravio RESTfull API. Svaka ruta je resurs koji poziva određeni *controller* ovisno o ruti. Kako bi se bolje objasnilo korištenje paketa, uzet će se primjer za warehouse rutu, koju koristi WarehouseController.

*Slika 31. Dostupne rute koje koriste HAL-API paket*

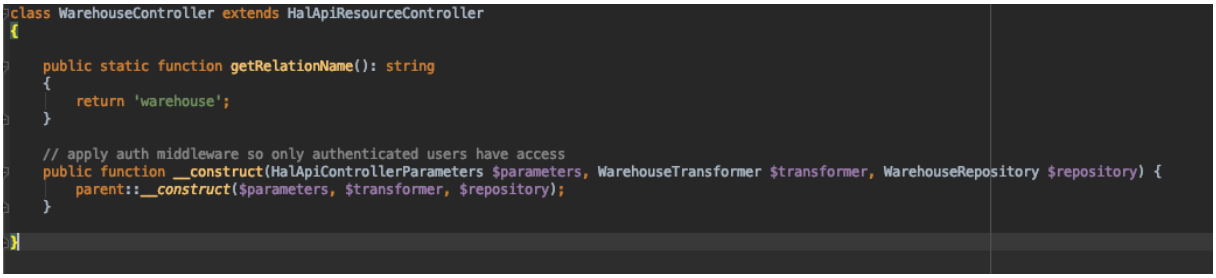
```
RouteHelper::make($router)
    ->resource( name: 'users',  controller: 'Api\UsersController')
    ->searchable()
    ->done()

    ->resource( name: 'products',  controller: 'Api\ProductsController')
    ->searchable()
    ->done()

    ->resource( name: 'product_type',  controller: 'Api\ProductTypesController')
    ->searchable()
    ->done()

    ->resource( name: 'supplies',  controller: 'Api\SuppliesController')
    ->searchable()
    ->done()

    ->resource( name: 'transactions',  controller: 'Api\SuppliesControlController')
    ->searchable()
    ->done()

    ->resource( name: 'session',  controller: 'Api\SessionController')
    ->searchable()
    ->done()

    ->resource( name: 'warehouse',  controller: 'Api\WarehouseController')
    ->searchable()
    ->done();
```

Izvor: izradio autor

Na Slici 32 prikazan je WarehouseController koji zahtjeva tri dodatne komponente kako bi funkcionirao, a to su:

- *Model* – podaci resursa koji se nalaze unutar modela
- *Repository* – repozitorij dohvaća i sprema modele
- *Transformer* – transformira model u HAL prikaz

*Slika 32. WarehouseController – primjer HAL-API resource kontrolera*



```
class WarehouseController extends HalApiResourceController
{

    public static function getRelationName(): string
    {
        return 'warehouse';
    }

    // apply auth middleware so only authenticated users have access
    public function __construct(HalApiControllerParameters $parameters, WarehouseTransformer $transformer, WarehouseRepository $repository) {
        parent::__construct($parameters, $transformer, $repository);
    }

}
```

Izvor: izradio autor

Jedna od tri komponente potrebne kako bi *controller* funkcionirao je *model* koji komunicira s bazom podataka. Svaka tablica u bazi ima svoj odgovarajući *model* koji se koristi za interakciju s tom tablicom. *Model* dozvoljavaju da se vrše upiti na tablicu, te također dodavanje novih zapisa u tablicu. U ovom slučaju model prikazan na Slici 33 uz poveznicu na tablicu iz baze i polja koja se mogu mijenjati, ima i event koji se pokrene kada se kreira novi zapis te povezanost s drugim tablicama kroz funkcije.

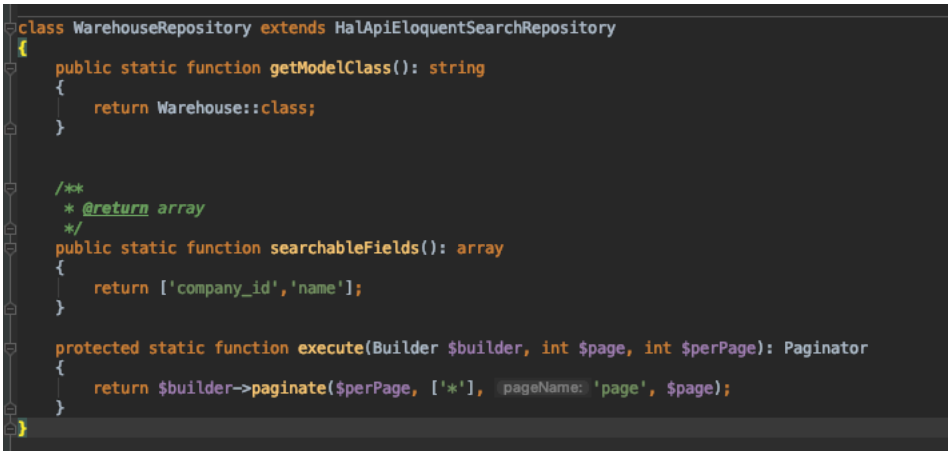
*Slika 33. Primjer modela*

```
class Warehouse extends Model
{
    protected $fillable = ['name', 'company_id', 'location', 'address'];
    protected $table = 'warehouse';

    protected $dispatchesEvents = [
        'created' => Events\NewWarehouseIsAddedEvent::class
    ];

    public function user() {
        return $this->belongsTo( related: User::class);
    }

    public function users() {
        return $this->hasMany( related: User::class);
    }

    public function supplies() {
        return $this->hasMany( related: Supplies::class);
    }
}
```

Izvor: izradio autor

Sljedeća bitna komponenta je *Repository*, gdje se dohvaća model. Sve što je potrebno je kreirati getModelClass funkciju prikazano na Slici 34. U funkciji searchableFields kao što naziv funkcije sam po sebi ukazuje, definira se koja polja iz tablice se mogu pretraživati kao bi dobili specifičnije rezultate. Zadnja funkcija na slici je dodatak zbog prilagodbe prikaza podataka.

*Slika 34. WarehouseRepository - primjer HAL-API repozitorija*

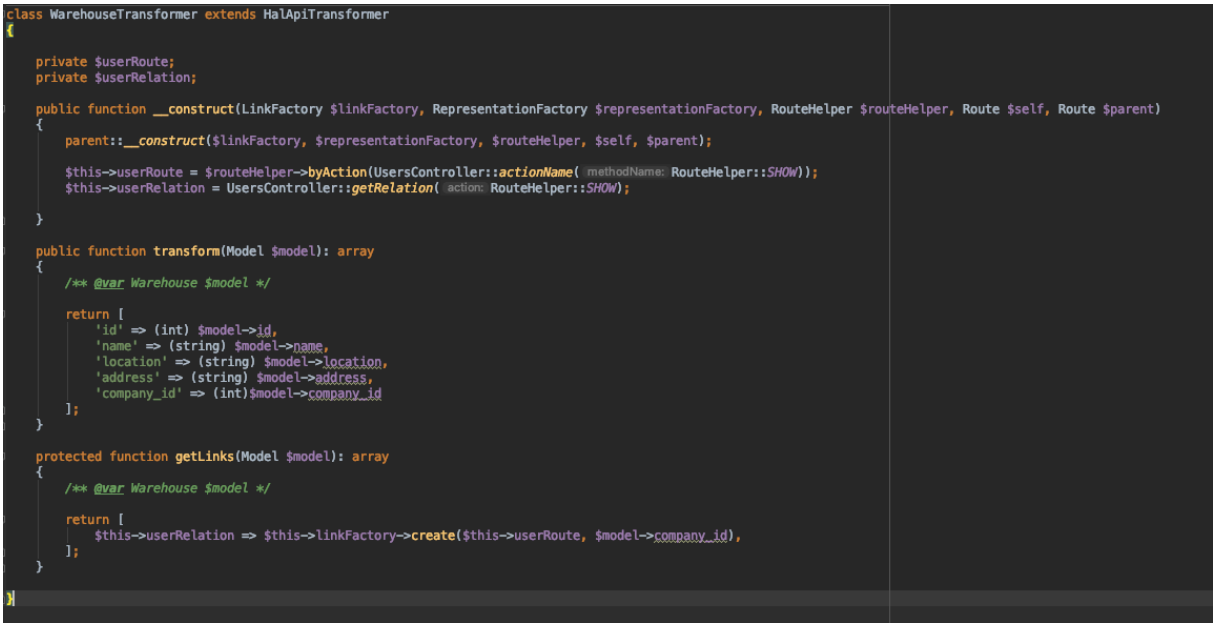


```php
class WarehouseRepository extends HalApiEloquentSearchRepository
{
    public static function getModelClass(): string
    {
        return Warehouse::class;
    }


    /**
     * @return array
     */
    public static function searchableFields(): array
    {
        return ['company_id','name'];
    }

    protected static function execute(Builder $builder, int $page, int $perPage): Paginator
    {
        return $builder->paginate($perPage, ['*'], pageName: 'page', $page);
    }
}
```

Izvor: izradio autor

Zadnja komponenta je *Transformer*. Na Slici 35. u funkciji transform može se vidjeti da se podaci dohvaćaju iz Warehouse modela te se vraćaju kao odgovor na zahtjev kojeg je korisnik uputio na rutu. Dohvaćeni podaci iz modela mogu se nazvati po potrebi, tj. dohvaćeni podatak poput lokacije skladišta ne mora se dohvaćati po istom ključu kako se naziva polje u tablici, već se može odrediti novi naziv za tak podatak te po njemu ga dohvatiti.

*Slika 35. WarehouseTransformer - primer HAL-API transformer-a*



```php
class WarehouseTransformer extends HalApiTransformer
{

    private $userRoute;
    private $userRelation;

    public function __construct(LinkFactory $linkFactory, RepresentationFactory $representationFactory, RouteHelper $routeHelper, Route $self, Route $parent)
    {
        parent::__construct($linkFactory, $representationFactory, $routeHelper, $self, $parent);

        $this->userRoute = $routeHelper->byAction(UsersController::actionName( methodName: RouteHelper::SHOW));
        $this->userRelation = UsersController::getRelation( action: RouteHelper::SHOW);

    }

    public function transform(Model $model): array
    {
        /** @var Warehouse $model */

        return [
            'id' => (int) $model->id,
            'name' => (string) $model->name,
            'location' => (string) $model->location,
            'address' => (string) $model->address,
            'company_id' => (int)$model->company_id
        ];
    }

    protected function getLinks(Model $model): array
    {
        /** @var Warehouse $model */

        return [
            $this->userRelation => $this->linkFactory->create($this->userRoute, $model->company_id),
        ];
    }

}
```

Izvor: izradio autor

Uz sve gore navedeno potrebno je i dodati poveznicu na *Transformer* u AppServiceProvider u funkciju *boot* na način prikazan na Slici 36 te isti postupak ponoviti za svaki *Transformer*.

*Slika 36. Primjer poveznice u AppServiceProvider klasi*

```
$this->app->singleton( abstract: WarehouseTransformer::class, function (Container $container) {
    $linkFactory = $container->make( abstract: LinkFactory::class);
    $representationFactory = $container->make( abstract: RepresentationFactory::class);
    $routeHelper = $container->make( abstract: RouteHelper::class);
    $self = $routeHelper->byAction(WarehouseController::actionName( methodName: RouteHelper::SHOW));
    $parent = $routeHelper->parent($self);

    return new WarehouseTransformer($linkFactory, $representationFactory, $routeHelper, $self, $parent);
});
```

Izvor: izradio autor

Također obavezno treba povezati sve parametre rute u RouteServiceProvider. Povratni poziv prikazan dolje obrađuje nedostajuće parametre, ovisno o metodi zahtjeva. Na primjer, GET zahtjev za nepostojeći zapis baze podataka trebao bi dati odgovor 404. Isto vrijedi i za sve ostale HTTP metode osim za PUT. PUT jednostavno stvara resurs ako nije postojao prije.

*Slika 37. Primjer povezivanja parametara rute*

```
public function boot(Router $router)
{
        parent::boot($router);

        $callback = RouteHelper::getModelBindingCallback();
        $router->model('users', User::class, $callback);
        $router->model('posts', Post::class, $callback);
}
```

Izvor: https://packalyst.com/packages/package/jarischaefer/hal-api

Međutim za potrebe ovog projekta koristio se malo drugačiji pristup povezivanja parametra prikazan na Slici 38.

*Slika 38. Povezivanje parametra rute u projektu*

```
parent::boot();

/** @var Router $router */
$router = $this->app['router'];


$router->group([
    'prefix' => 'api',
    'as' => 'api.',
    'middleware' => ['json.response','auth:api'],
], function () {


    Route::bind( key: 'users', function ($value) {
        return User::whereId($value)
            ->first();
    });
    Route::bind( key: 'products', function ($value) {
        return Products::whereId($value)
            ->first();
    });
    Route::bind( key: 'product_type', function ($value) {
        return ProductsType::whereId($value)
            ->first();
    });
    Route::bind( key: 'supplies', function ($value) {
        return Supplies::whereId($value)
            ->first();
    });
    Route::bind( key: 'transactions', function ($value) {
        return SuppliesControl::whereId($value)
            ->first();
    });
    Route::bind( key: 'warehouse', function ($value) {
        return Warehouse::whereId($value)
            ->first();
    });
    Route::bind( key: 'session', function ($value) {
        return Session::whereId($value)
            ->first();
    });

});
```

Izvor: izradio autor

Konačni rezultat svega postavljanja može se vidjeti u JSON odgovoru prikazanom na Slici 39, gdje se šalje GET upit serveru, tj. API-ju te se dobiveni podaci vraćaju u JSON obliku koji se može podijeliti u tri glavna dijela, a to su:

- meta – ovaj dio opisuje koliko ima zapisa po stranici, koliko ima ukupno zapisa i ukupno stranica, koja je trenutna stranica koja se prikazuje
- _links – ovo su linkovi koji su povezani sa tom tablicom iz koje se dobivaju podaci preko kojih možemo dobiti poveznicu na drugu tablicu koja je u relaciji s trenutnom, osim toga ima linkove koji služe za dobivanje ostalih stranica kako bi se mogli vidjeti i ostali zapisi
- _embedded – ovaj dio sadrži zapise iz tablice, sve njegove podatke, te njegove poveznice s drugim tablicama

*Slika 39. Primjer JSON odgovora*

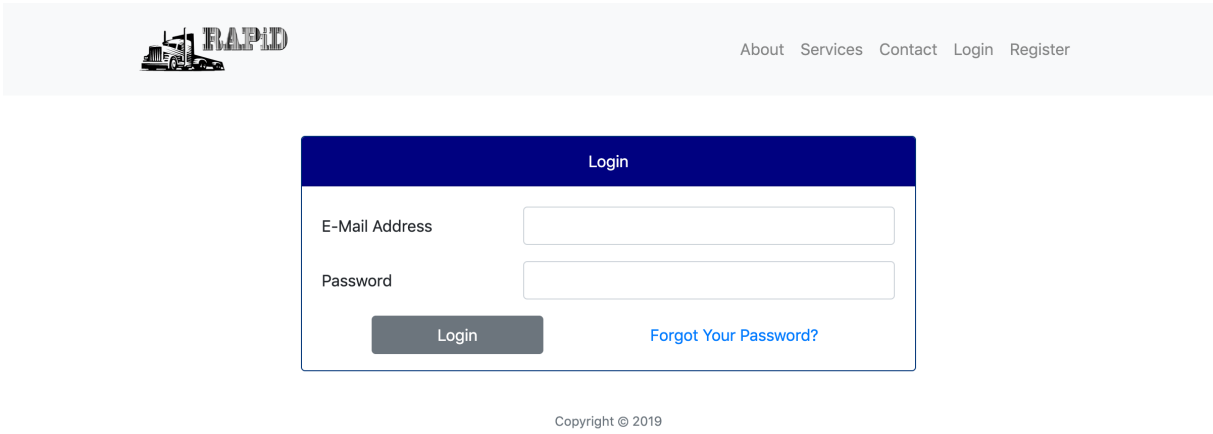

```
{
    "meta": {
        "pagination": {
            "page": 1,
            "per_page": 10,
            "count": 1,
            "total": 1,
            "pages": 1
        }
    },
    "_links": {
        "self": {
            "href": "http:\/\/rapid.gq\/api\/warehouse?page={page}&per_page={per_page}",
            "templated": true
        },
        "parent": {
            "href": "http:\/\/rapid.gq\/api\/warehouse?page={page}&per_page={per_page}",
            "templated": true
        },
        "first": {
            "href": "http:\/\/rapid.gq\/api\/warehouse?page=1&per_page={per_page}",
            "templated": true
        },
        "last": {
            "href": "http:\/\/rapid.gq\/api\/warehouse?page=1&per_page={per_page}",
            "templated": true
        },
        "warehouse.search": {
            "href": "http:\/\/rapid.gq\/api\/warehouse\/search?page={page}&per_page={per_page}",
            "templated": true
        },
        "warehouse.show": {
            "href": "http:\/\/rapid.gq\/api\/warehouse\/{warehouse}",
            "templated": true
        },
        "warehouse.store": {
            "href": "http:\/\/rapid.gq\/api\/warehouse",
            "templated": false
        },
        "warehouse.update": {
            "href": "http:\/\/rapid.gq\/api\/warehouse\/{warehouse}",
            "templated": true
        },
        "warehouse.destroy": {
            "href": "http:\/\/rapid.gq\/api\/warehouse\/{warehouse}",
            "templated": true
        }
    },
    "_embedded": {
        "warehouse.show": [
            {
                "data": {
                    "id": 2,
                    "name": "Wiva",
                    "location": "Pula",
                    "address": "Medulinska 1",
                    "company_id": 1
                },
                "_links": {
                    "self": {
                        "href": "http:\/\/rapid.gq\/api\/warehouse\/2",
                        "templated": true
                    },
                    "parent": {
                        "href": "http:\/\/rapid.gq\/api\/warehouse?page={page}&per_page={per_page}",
                        "templated": true
                    },
                    "users.show": {
                        "href": "http:\/\/rapid.gq\/api\/users\/1",
                        "templated": true
                    },
                    "warehouse.update": {
                        "href": "http:\/\/rapid.gq\/api\/warehouse\/2",
                        "templated": true
                    },
                    "warehouse.destroy": {
                        "href": "http:\/\/rapid.gq\/api\/warehouse\/2",
                        "templated": true
                    }
                },
                "_embedded": []
            }
        ]
    }
}
```


Izvor. izradio autor

# 10. KORISNIČKE UPUTE

## 10.1. Web aplikacija

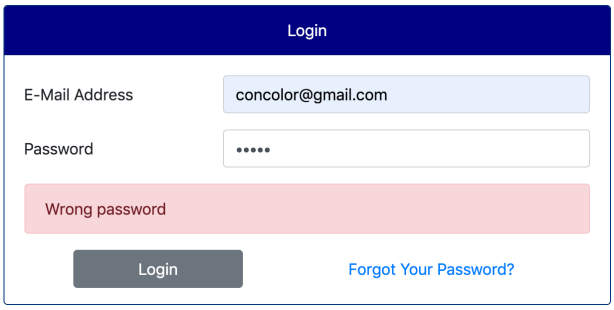
*Slika 40. Prijava u web aplikaciju*




Izvor: izradio autor

Kako bi se započelo korištenje web aplikacije, nakon uspješne registracije te aktivacije računa, korisnik se treba logirati u sustav na način da unese svoj email i zaporku. U slučaju da unese krivi email ili krivu zaporku, javlja se poruka koja obavještava korisnika što je krivo upisao, primjer je prikazan na Slici 41 gdje je korisnik krivo upisao svoju zaporku i poruka se pojavi da ga obavijesti. Zatim korisnik mora ponovo unijeti zaporku da bi se prijavio u sustav.

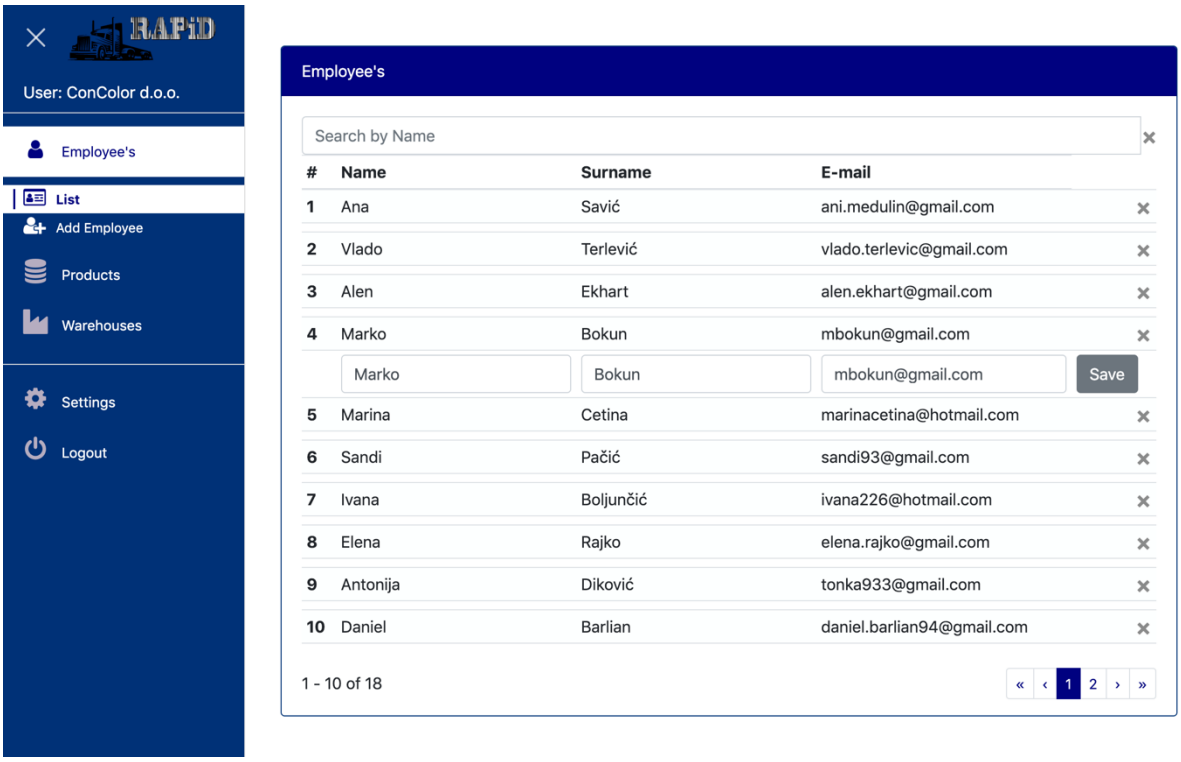
*Slika 41. Primjer neuspješnog logiranja u sustav*




Izvor: izradio autor

Nakon što se uspije prijaviti, web aplikacija ga preusmjerava na jednu od dvije moguće početne stranice. Prva moguća početna stranica je lista zaposlenika, koja se prikazuje korisniku kada on ima status poduzeća.

*Slika 42. Početna stranica za poduzeće nakon uspješne prijave*



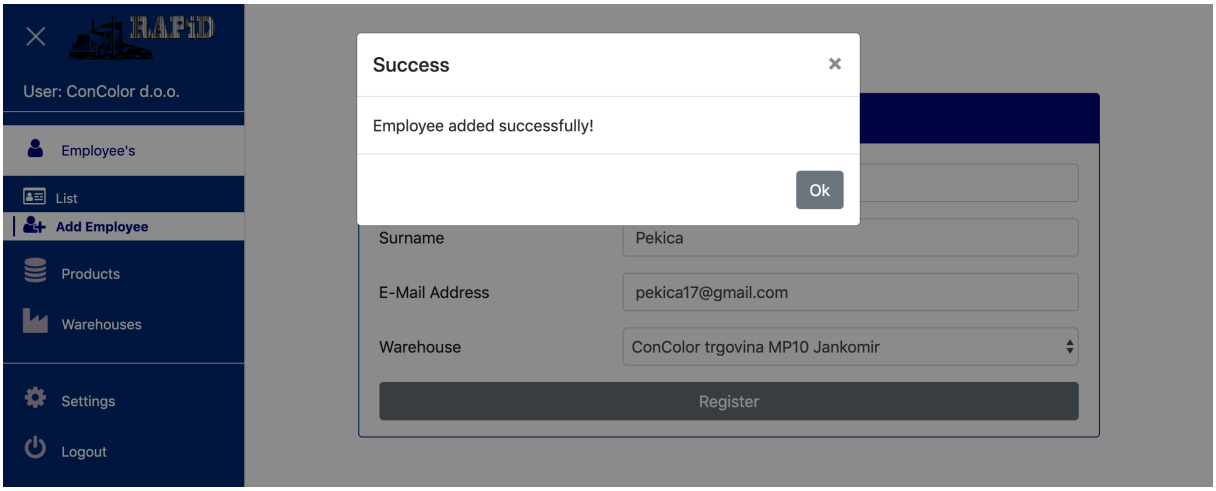

Izvor: izradio autor

Uz popis zaposlenika gdje se prikazuje po 10 zaposlenika u tablici, korisnik odnosno poduzeće može na tom mjestu ažurirati osnovne podatke o zaposleniku na način da pozicionira strelicu miša na zaposlenika kojem želi promijeniti podatke, te klikne na taj redak u tablici. Zatim se otvara novo polje u tablici sa poljima koji imaju informacije o odabranom korisniku te se mogu mijenjati. Pritiskom na *Save* gumb informacije se ažuriraju te se bez osvježavanja stranice prikazuju novi podaci o zaposleniku. Osim što se mogu mijenjati informacije o zaposleniku, zaposlenici se mogu brisati pritiskom na *x* gumb sa desne strane.

Osim početne stranice gdje poduzeće može vidjeti popis svojih zaposlenika te mijenjati informacije, sa lijeve strane može se vidjeti izbornik pomoću kojeg se poduzeće može navigirati kroz ostale mogućnosti koje su joj na raspolaganje, pa je tako jedna od tih i dodavanje zaposlenika.

Kako bi se uspješno dodao novi zaposlenik, potrebno je ispuniti polja s odgovarajućim podacima, a to su ime, prezime i email zaposlenika kojeg se želi dodati. Kao što se vidi na Slici 43 nakon tih podataka ostaje još odabir kojem skladištu će novi zaposlenik biti dodijeljen, stoga da bi se uspješno dodao novi zaposlenik potrebno je najprije dodati skladišta kojima poduzeće raspolaže.

*Slika 43. Primjer uspješnog dodavanja zaposlenika*

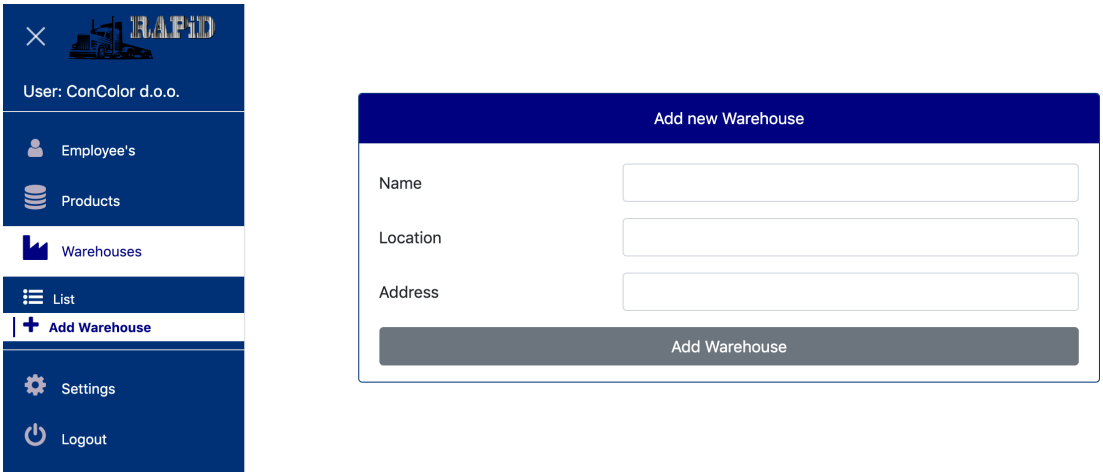



Izvor: izradio autor

Ako su skladišta prethodno dodana, odabere se skladište u kojem će novi zaposlenik raditi te se registrira korisnika pritiskom na gumb register nakon čega se može vidjeti poruka prikazana na Slici 43 kako je zaposlenik uspješno dodan. Sustav zatim šalje email zaposleniku koji sadrži aktivacijski link te privremenu zaporku koju se preporuča korisniku da promjeni nakon što se prvi put logira u sustav.

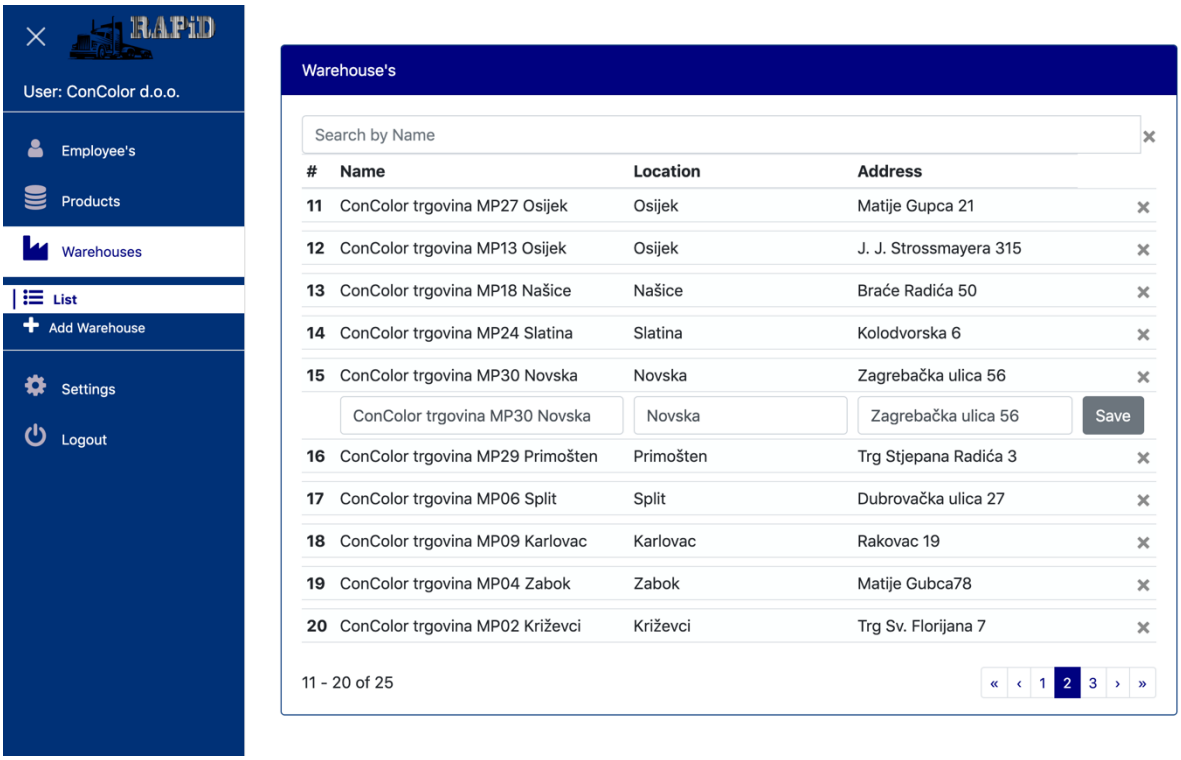
Na sljedeće dvije slike se može vidjeti obrazac što je potrebno da bi se dodalo novo skladište i popis skladišta kojima poduzeće raspolaže. Također za skladišta kao i za zaposlenike mogu se direktno u tablici ažurirati podaci o skladištima.

*Slika 44. Obrazac za dodavanje skladišta*




Izvor: izradio autor

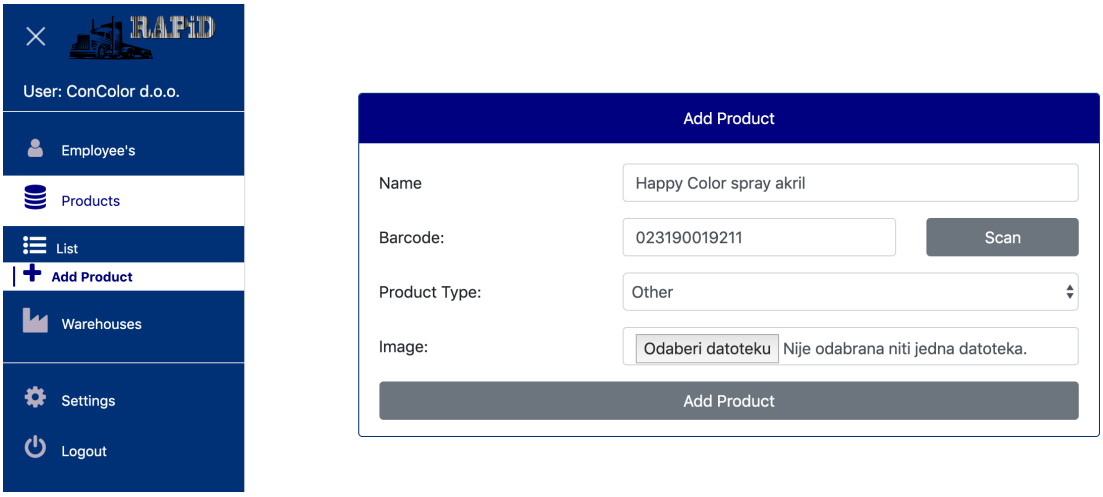
*Slika 45. Prikaz tablice s listom skladišta*




Izvor: izradio autor

Zadnja bitna stvar i zapravo najbitnija za poduzeće su njihovi proizvodi. Sustav je i napravljen za rad s proizvodima kako bi se olakšao prijem robe u skladištima, što će biti dodatno prikazano kroz upute za mobilnu aplikaciju. Da bi se sustav koristio, potrebno je dodati proizvode, što se može napraviti ispunjavanjem obrasca na Slici 46.

*Slika 46. Primjer ispunjenog obrasca za dodavanje proizvoda*

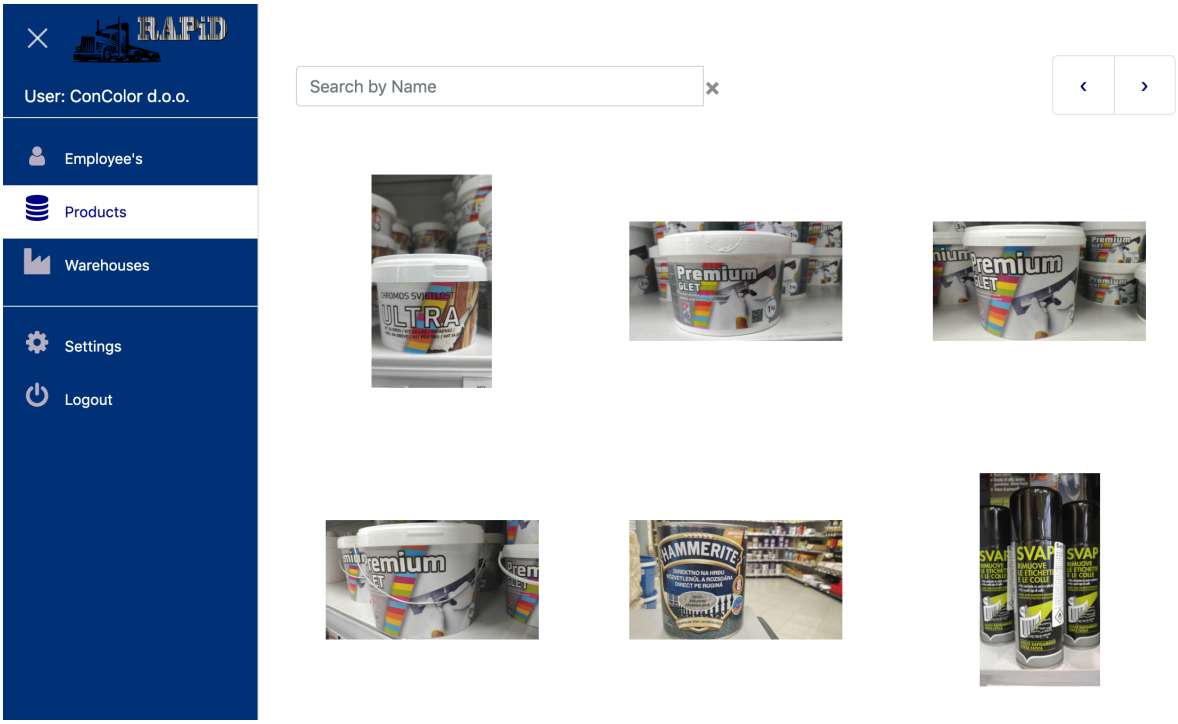



Izvor: izradio autor

Na slici je prikazan primjer ispunjenog obrasca, naziv proizvoda, barkod i tip proizvoda su obavezni za ispuniti kako bi se dodao novi proizvod, dok je odabir slike opcionalan. Barkod se može ručno upisati, ali postoji lakši način. Korištenjem mobilne aplikacije može se skenirati barkod i poslati na web, gdje se pritiskom na gumb *Scan* započinje tražiti  je li barkod skeniran i poslan na web. Ukoliko je skeniran, tada ga web aplikacija nalazi te prikazuje u predviđeno polje i zaustavlja pretraživanje.

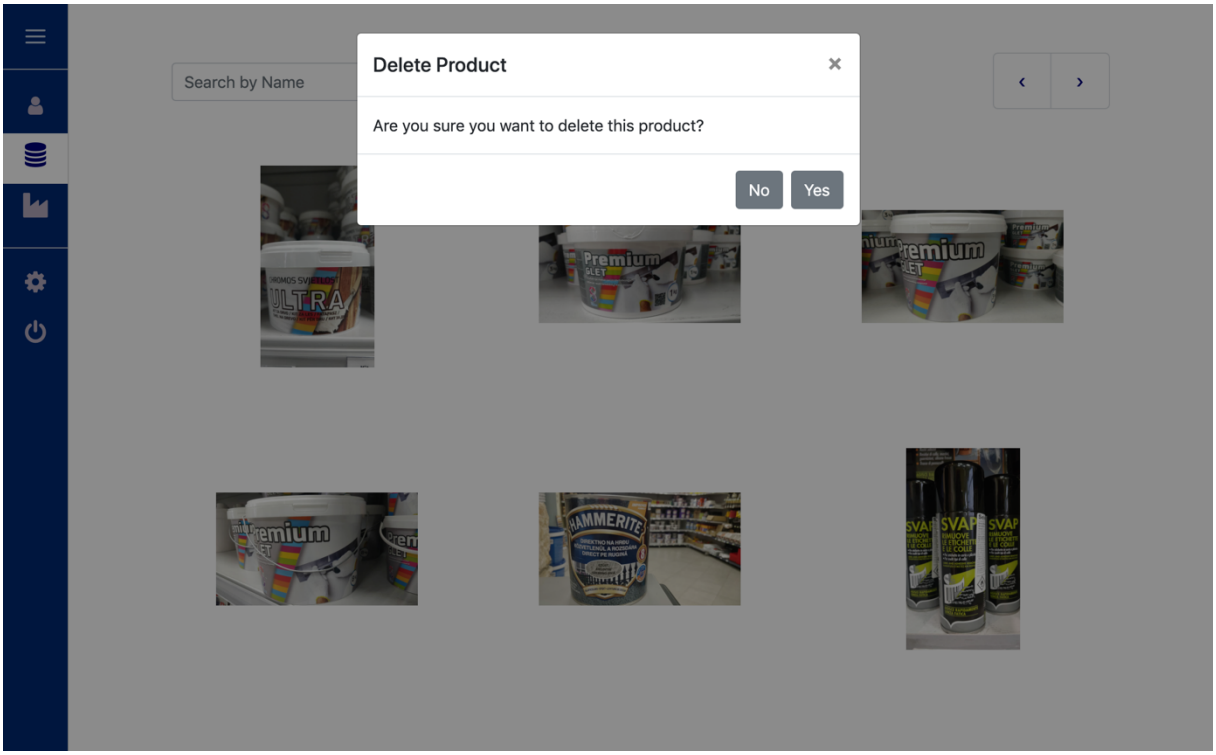
Kada se proizvodi dodaju, svi dodani proizvodi se mogu vidjeti na listi proizvoda, u ovom slučaju prikaza slika proizvoda, gdje se prelaskom miša preko slike može vidjeti naziv proizvoda, te dva gumba. *Edit* gumb preusmjerava korisnika na stranicu gdje može vidjeti i ažurirati sve podatke o proizvodu, a *x* gumb služi za brisanje proizvoda. Kada se klikne *x* gumb pojavi se skočni prozor koji traži potvrdu brisanja, što se može vidjeti na Slici 48.

*Slika 47. Primjer liste proizvoda*




Izvor: izradio autor

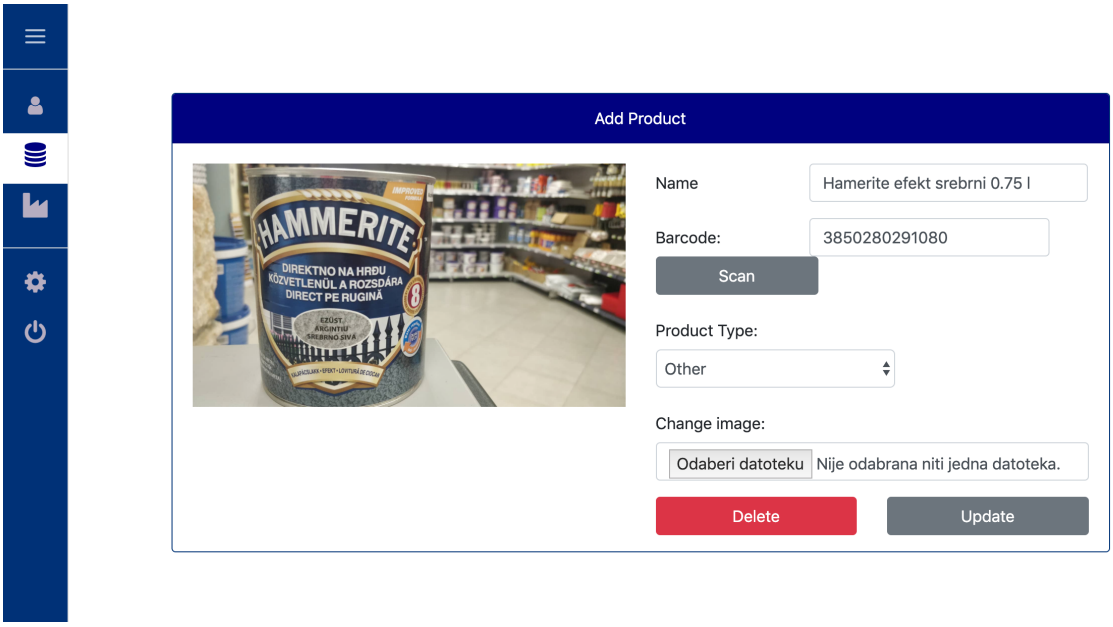
*Slika 48. Potvrda brisanja proizvoda*




Izvor: izradio autor

Na Slici 49 prikazane su sve informacije o proizvodu, te se sve informacije mogu promijeniti pa tako i skenirati novi barkod ili zamijeniti sliku proizvoda.

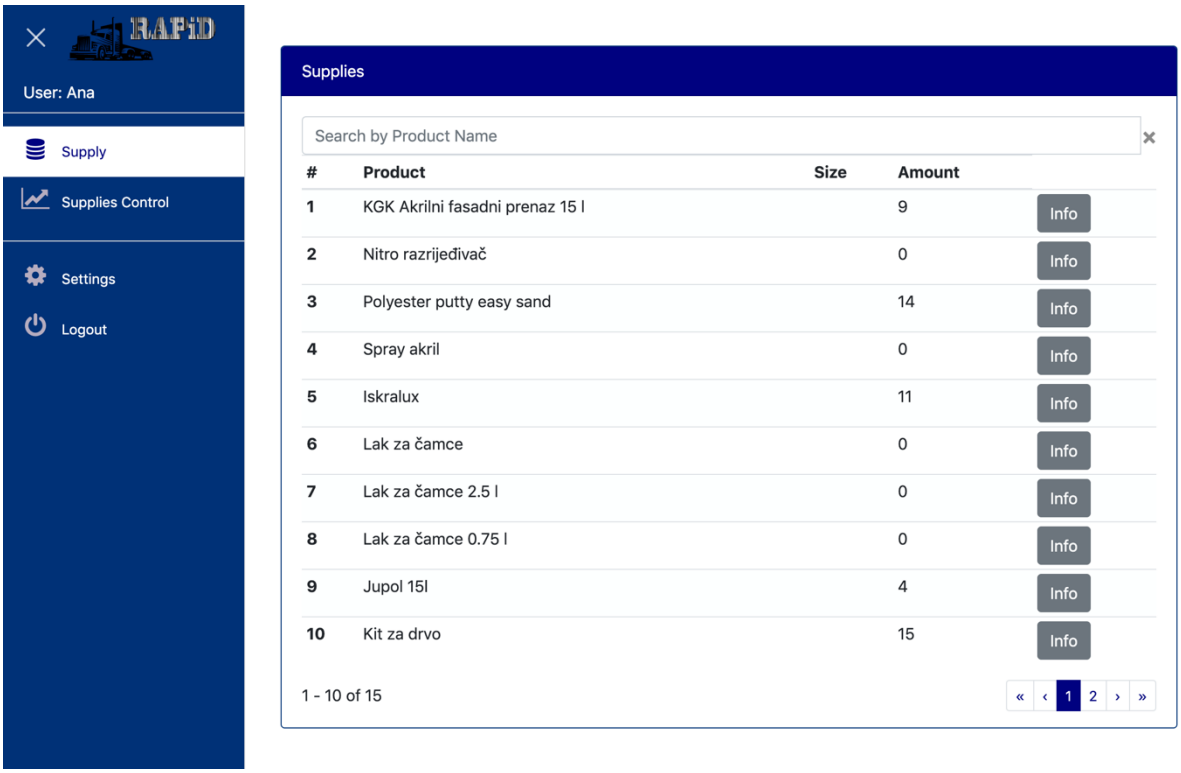
*Slika 49. Prikaz informacija o proizvodu koje se mogu ažurirati*




Izvor: izradio autor

Druga moguća početna stranica je prikaz trenutnog stanja zaliha u skladištu. Ova stranica postaje početna kada korisnik koji se prijavljuje ima status zaposlenika. Uz prikaz trenutnog stanja može se kliknuti na gumb *Info* koji preusmjerava na novu stranicu te prikazuje informacije o proizvodu.

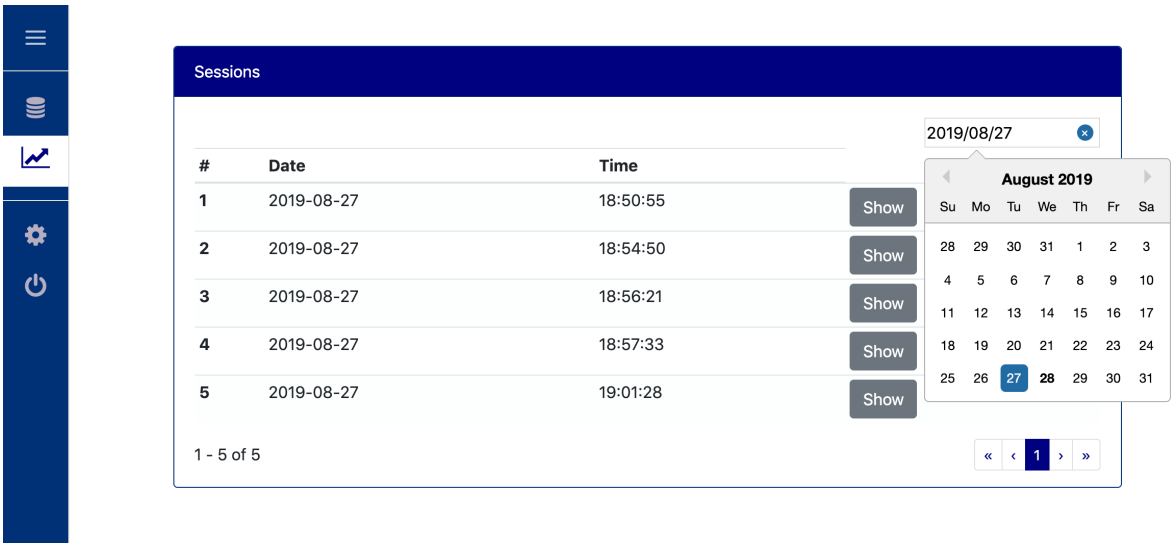
*Slika 50. Primjer stanja zaliha u skladištu*




Izvor: izradio autor

Osim prikaza stanja u skladištu, korisnik, tj. zaposlenik može vidjeti svaki prijem robe prikazan u tablici po datumu i vremenu. Također se može filtrirati rezultate koji se prikazuju, na način da se odabere datum za koji se želi vidjeti prijem robe. Klikom na gumb *Show* za odabrani redak preusmjerava se na sljedeću stranicu gdje se mogu vidjeti svi proizvodi i količine koje su se unesle u stanje zalihe za odabrano vrijeme prijama robe.

*Slika 51. Primjer liste prijema robe za odabrani datum*



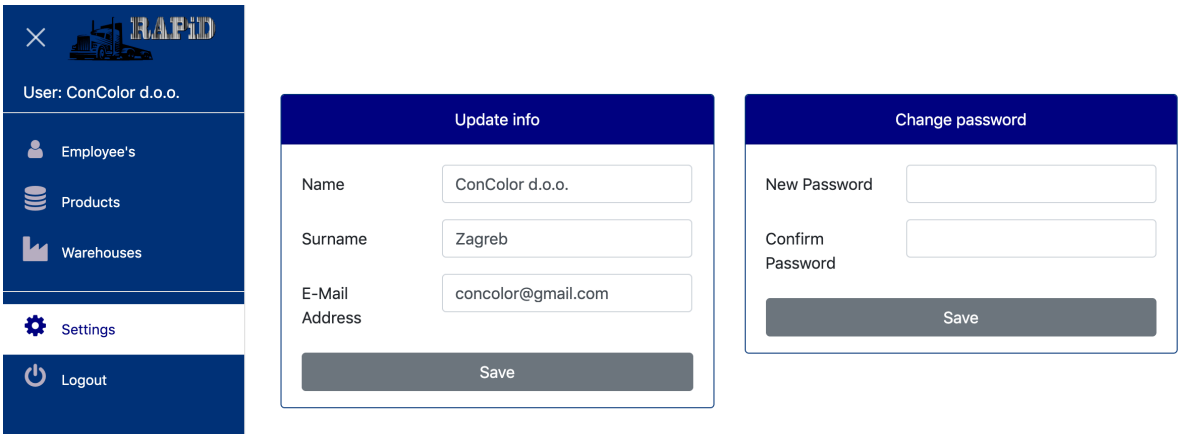

Izvor: izradio autor

*Slika 52. Primjer popisa proizvoda pristiglih za odabrani prijem robe*

Izvor: izradio autor

Ono što je zajedničko za oba tipa korisnika je stranica gdje mogu promijeniti svoje podatke i zaporku, što se može vidjeti na slici 53.

*Slika 53. Izgled stranice za promjenu osobnih podataka*




Izvor: izradio autor

## 10.2. Mobilna aplikacija

*Slika 54. Prijava u mobilnu aplikaciju*

Izvor: izradio autor

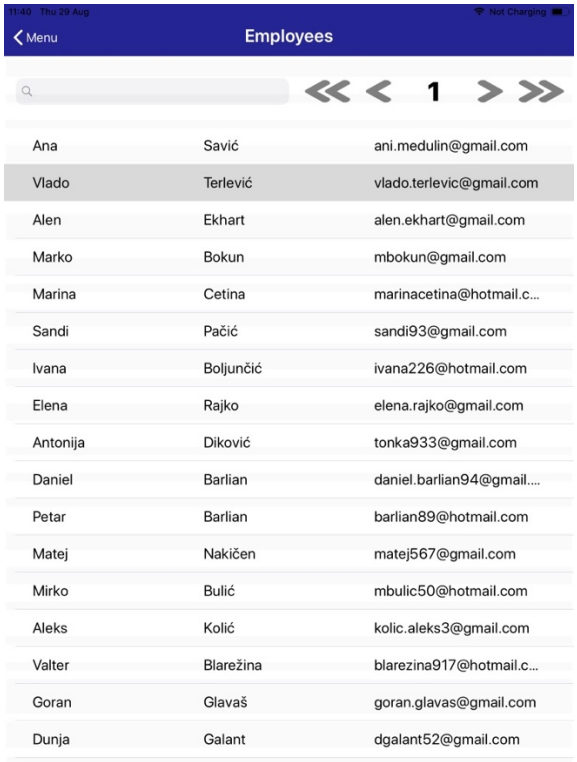
Da bi se započelo korištenje mobilne aplikacije, korisnik najprije treba aktivirati svoj račun na web aplikaciji, te kada to uspije može se logirati u mobilnu aplikaciju i započeti s korištenjem. Nakon uspješnog logina korisnik se preusmjerava na glavni izbornik, te ovisno radi li se o poduzeću ili zaposleniku, nude se različite funkcionalnosti.

*Slika 55. Prikaz glavnog izbornika za poduzeće*

Izvor: izradio autor

Ako mobilnu aplikaciju koristi poduzeće, onda ima na raspolaganju vidjeti popis zaposlenika, popis proizvoda kojima raspolaže, popis skladišta koja ima u vlasništvu i skeniranje proizvoda, tj. skeniranje barkoda koji se šalje na web aplikaciju kako bi se olakšalo dodavanje proizvoda i izbjeglo ručno upisivanje barkoda za svaki proizvod. Kroz sljedeće tri slike prikazat će se kako u aplikaciji izgleda kada korisnik, tj. poduzeće želi vidjeti tko su mu zaposlenici, koja skladišta ima u posjedu i kako izgleda kad se uspješno skenira barkod i pošalje na web kako bi se olakšalo dodavanje proizvoda.

*Slika 56. Primjer popisa zaposlenika*

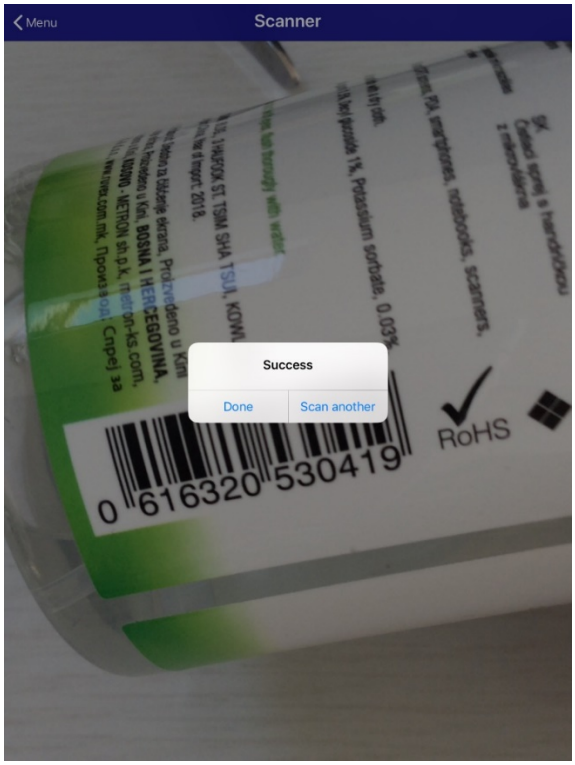



Izvor: izradio autor

*Slika 57. Primjer popisa skladišta*

Izvor: izradio autor

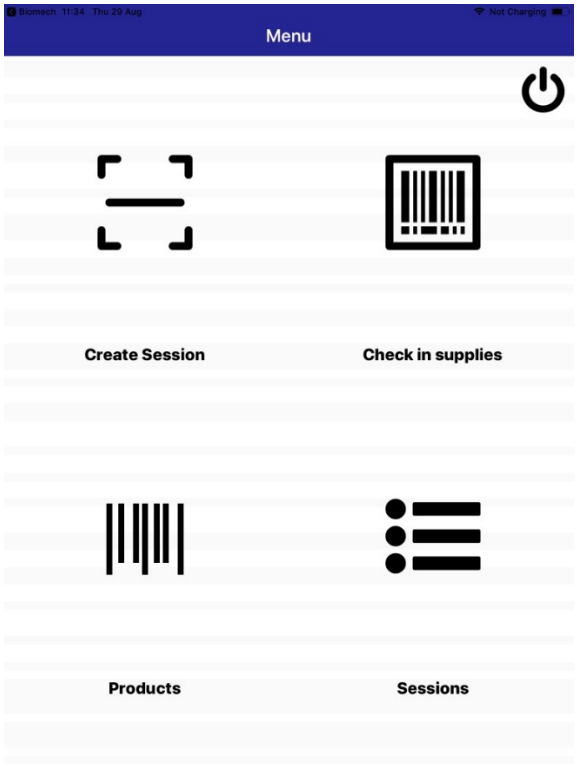
*Slika 58. Primjer uspješnog skeniranja barkoda*




Izvor: izradio autor

Glavni izbornik za zaposlenika prikazan je na Slici 59 i ima također četiri funkcionalnosti poput izbornika kod poduzeća. Imaju i jednu zajedničku funkcionalnost, a to je popis proizvoda, što se može vidjeti kako izgleda na Slici 60. Oba tipa korisnika mogu vidjeti kojim proizvodima poduzeće raspolaže pa tako mogu i filtrirati, tj. pretraživati proizvode po nazivu kako bi jednostavno bili u mogućnosti potražiti određeni proizvod.

*Slika 59. Prikaz glavnog izbornika za zaposlenika*

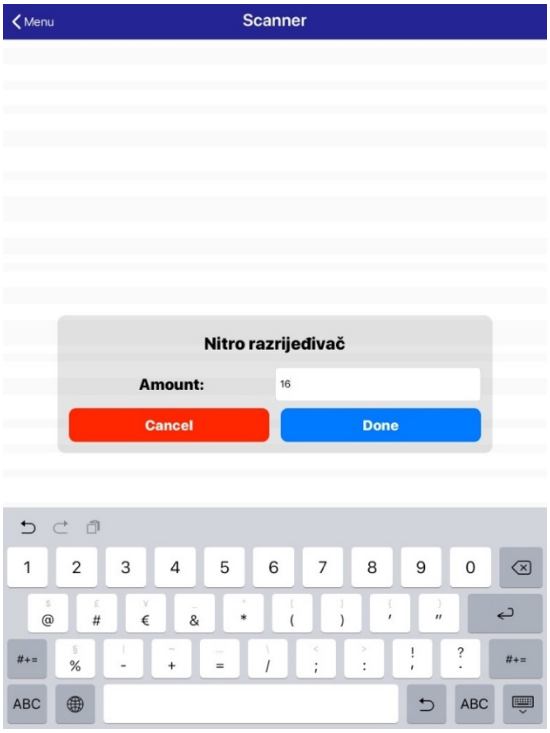



Izvor: izradio autor

*Slika 60. Primjer prikaza liste proizvoda*

Izvor: izradio autor

*Slika 61. Primjer skeniranja kod prijema robe i dodavanja količine*



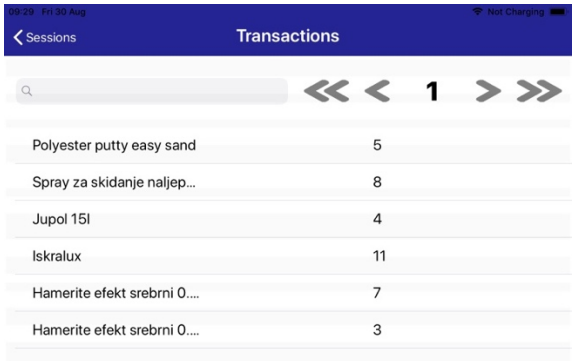

Izvor: izradio autor

Korisnik može kreirati sesiju odabirom *Create Session* iz glavnog izbornika. Tada skenira pojedini proizvod koji je potrebno dodati na stanje skladišta. Barkod proizvoda se skenira i uspoređuje s bazom podataka. U slučaju da je barkod pronađen, korisniku se vraća forma koja traži od korisnika da upiše količinu koja je pristigla i treba se dodati na stanje zaliha u skladištu. Kada se upiše količina, pritisne se gumb *Done* čime se potvrđuje količina, dodaje na stanje, potom vraća na skener gdje se može skenirati barkod sljedećeg proizvoda koji je pristigao.

*Slika 62. Primjer prikaza kreiranih sesija*

Izvor: izradio autor

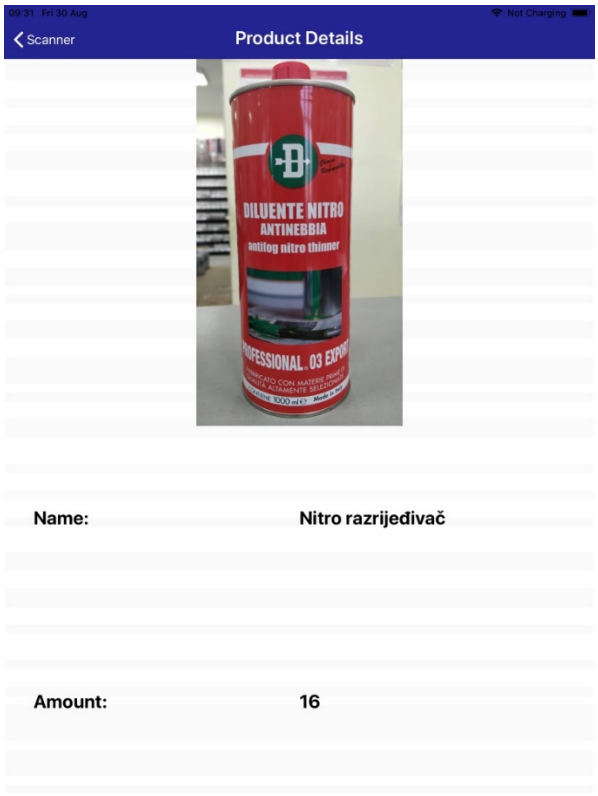
*Slika 63. Primjer skeniranih proizvoda za odabranu sesiju*




Izvor: izradio autor

Nakon što se kreira sesija te dodaju pristigli proizvodi, može se vidjeti koje je sve sesije kreirao logirani korisnik. Pritiskom na redak koji sadrži datum i vrijeme kreirane sesije, korisnik se preusmjerava na popis proizvoda koji su se dodali na stanje skladišta u odabranoj sesiji. Zadnja opcija koju zaposlenik ima je skeniranje proizvoda kako bi se provjerilo trenutno stanje u skladištu. Rezultat se može vidjeti na Slici 64.

*Slika 64. Primjer skeniranja proizvoda da se provjeri količina na stanju*




Izvor: izradio autor

# ZAKLJUČAK

RAPiD sustav zahtjeva integriranje više različitih tehnologija u jednu smislenu cjelinu. Da bi se proces prijama robe što više automatizirao, proučen je postupak prijema robe, postupak skladištenja robe, te razni problemi koji se mogu pojaviti. Kvalitetno vođenje evidencije skladišta je vrlo bitno kako bi se smanjili gubitci kod prijema te skladištenja robe. Razvijen sustav namijenjen je poduzećima i njihovim skladištima kako bi ubrzali postupak prijama robe i kvalitetno vodili evidenciju, te imali mogućnost da brzo i jednostavno pristupe podacima koji prikazuju trenutno stanje zaliha u pojedinom skladištu. Podaci su dostupni svim djelatnicima poduzeća za skladište kojem su dodijeljeni, tj. za skladište u kojem rade. Sustav se sastoji od mobilne aplikacije i web aplikacije kojima je potrebna komunikacija s bazom podataka, gdje se nalaze svi potrebni podaci.

Da bi naposljetku sve komponente sustava mogle međusobno komunicirati i razmjenjivati podatke, opisan je RESTful API. RESTful API čini ključnu ulogu u RAPiD sustavu za prijem robe zato što omogućuje komunikaciju između mobilne aplikacije i web aplikacije preko kojeg korisnici mogu spremati, dodavati, izmjenjivati ili dohvatiti potrebne informacije iz baze podataka kako bi mogli pratiti prijem robe u skladištima, te stvarno stanje zaliha po skladištima. Slično klijentskoj interakciji s web stranicom, REST klijent koji koristi RESTful API doseže početni URI i koristi veze na poslužitelju kako bi dinamički otkrio dostupne radnje i pristupio resursima koji su mu potrebni. Klijent ne mora imati prethodno znanje o usluzi ili različite korake koji su uključeni u tijek rada. Uz to, klijenti više ne moraju ručno unositi URI strukture za različite resurse. To omogućuje poslužitelju da promijeni URI kako se API razvija bez problema, bez da utječe na radnju klijenta.

Rezultat rada je sustav koji olakšava i ubrzava prijem robe te uspješno koristi RESTful API-je u komunikaciji između komponenta.

# LITERATURA

**Knjige:**


[1.] Doglio, F. (2018) *REST API Development with Node.js*. Apress

[2.] Karanam, R. R. (2019) *Mastering Spring 5*. 2nd edition. Birmingham. Packt Publishing.

[3.] Kyrnin, J. (2015) *Sams Teach Yourself HTML, CSS & JavaScript*. 7-enth Edition. United States of America. Sams

[4.] Friesen, J (2019) *Java XML and JSON*. Apress.

[5.] Flanagan, D. (2011) *JavaScript: The Definitive Guide*. 6th Edition. O'Reilly Media, Inc.

[6.] Robson, E. (2014) *Head First JavaScript Programming*. O'Reilly Media, Inc.

[7.] HAVERBEKE, M. (2015) *Eloquent JavaScript a Modern introduction to programming.* 2nd edition, San Francisco, California, No Starch Press

[8.] MALATESTA, F. (2015), *Learning Laravel's Eloquent*, Livery Place, Ujedinjeno Kraljevstvo, Packt Publishing Ltd

[9.] Anthony, A. (2017). *Fullstack react: The complete guide to reactjs and friends*.

[10.] Wieruch, R. (2017). *The Road to Learn React: Your Journey to Master Plain Yet Pragmatic React. Js*. CreateSpace Independent Publishing Platform.

[11.] Bray, T. (2017). *The javascript object notation (json) data interchange format*. Tec- hnical report.


**Web izvori:**


[12.] https://laravel.com/docs/5.5 [Pristupljeno: 12.4.2019.]

[13.] Garc, R. (2017) *SKLADIŠNI SUSTAV KAO LOGISTIČKI PODSUSTAV PODUZEĆA.* Završni rad. Požega. Veleučilište u Požegi https://zir.nsk.hr/islandora/object/vup:378/preview [Pristupljeno: 21.4.2019.]

[14.] Jangjel, K. (2018) *ANALIZA SKLADIŠNOG PROCESA PRIJAMA ROBE NA PRIMJERU IZ PRAKSE*. Diplomski rad. Zagreb. Fakultet Prometnih Znanosti. https://repozitorij.fpz.unizg.hr/islandora/object/fpz%3A1594/datastream/PDF/view [Pristupljeno: 27.4.2019.]

[15.] http://www.webtech.com.hr/html.php [Pristupljeno: 29.7.2019.]

[16.] Anicas, M (2014) *An Introduction to OAuth 2*. DigitalOcean. https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2 [Pristupljeno: 24.5.2019.]

[17.] Holjevac, I. (2010) *KLIJENTSKE WEB TEHNOLOGIJE*. Diplomski rad. Varaždin. Fakultet organizacije i informatike Varaždin. https://www.irenaholjevac.com/dipl/ [Pristupljeno: 3.8.2019.]

[18.] *HATEOAS Driven REST APIs*. https://restfulapi.net/hateoas/?fbclid=IwAR0GgftTB9BRVz-CpCPbaxNEiQqQ9Dogsx-7Z64aXB4sQNIBkYGV89mJDcU [Pristupljeno: 27.4.2019.]

[19.] Fielding Roy Thomas (2000), *Architectural Styles and the Design of Network-based Software Architectures*, Sveučilište u Kaliforniji. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf?fbclid=IwAR0DP4eROlaobQwL9Q6BxM_2aE37Dm6yS2SGJjM-R8xWwYu-1yXn0mIXouQ [Pristupljeno: 28.7.2019.]

[20.] Microsoft. *Microsoft REST API Guidelines*. https://github.com/Microsoft/api-guidelines/blob/vNext/Guidelines.md?fbclid=IwAR0di_IgopJDSHNfpK-uwZPr0jiRD_-I6vwMFDcjUepnV8HdLotvMzSttzo [Pristupljeno: 25.3.2019.]

[21.] OAuth 2.0. OAuth. https://oauth.net/2/ [Pristupljeno: 6.8.2019.]

[22.] React. https://reactjs.org/ [Pristupljeno: 4.5.2019.]

[23.] Redux. (2018) https://redux.js.org/introduction/getting-started [Pristupljeno: 2.6.2019.]

[24.] Kelly, M. (2013) HAL - *Hypertext Application Language*. http://stateless.co/hal_specification.html [Pristupljeno: 3.5.2019.]

[25.] Rouse, M. (2019) *REST (REpresentational State Transfer)*. https://searchapparchitecture.techtarget.com/definition/REST-REpresentational-State-Transfer [Pristupljeno: 7.8.2019.]

[26.] Ben Avraham , S. (2017) *What is REST — A Simple Explanation for Beginners*. https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-1-introduction-b4a072f8740f [Pristupljeno: 7.8.2019.]

[27.] Rouse, M. (2019) *RESTful API*. https://searchapparchitecture.techtarget.com/definition/RESTful-API [Pristupljeno: 5.8.2019.]

# POPIS SLIKA

Slika 1. Primjer HTML dokumenta ..... 2
Slika 2. Primjer PHP koda ugrađen u HTML ..... 4
Slika 3. Primjer React komponente ..... 7
Slika 4. MVC Arhitektura ..... 8
Slika 5. Razine zrelosti prema Richardsonovom modelu ..... 33
Slika 6. Primjer sadržaja odgovora HATEOAS-a ..... 36
Slika 7. Use - case dijagram: Mogućnosti na Web aplikaciji ..... 40
Slika 8. Use - case dijagram: Mogućnosti na Mobilnoj aplikaciji ..... 40
Slika 9. Sekvencijski dijagram: Tijek korištenja web aplikacije za poduzeće ..... 41
Slika 10. Sekvencijski dijagram: Prikazivanje popisa zaposlenika u mobilnoj aplikaciji ..... 42
Slika 11. Sekvencijski dijagram: Prikazivanje popisa skladišta u mobilnoj aplikaciji ..... 43
Slika 12. Sekvencijski dijagram: Skeniranje barkoda za dodavanje proizvoda na web aplikaciji ..... 43
Slika 13. Sekvencijski dijagram: Tijek korištenja web aplikacije za zaposlenika ..... 44
Slika 14. Sekvencijski dijagram: Skeniranje proizvoda da se sazna stanje na skladištu ..... 45
Slika 15. Sekvencijski dijagram: Kreiranje novog prijema robe ..... 46
Slika 16. Sekvencijski dijagram: Prikaz prijema robe u mobilnoj aplikaciji ..... 46
Slika 17. Sekvencijski dijagram: Prikaz proizvoda u mobilnoj aplikaciji ..... 47
Slika 18. Baza podataka - Relacije između tablica za RAPiD sustav ..... 49
Slika 19. Prikaz svih ruta ..... 50
Slika 20. Primjer migracije ..... 52
Slika 21. Glavna komponenta Korisničkog sučelja ..... 53
Slika 22. Provjera autorizacije korisnika ..... 53
Slika 23. Store funkcija ..... 54
Slika 24. Primjer kreiranih akcija za korištenje Reduxa ..... 54
Slika 25. Primjer Redux reducera ..... 55
Slika 26. Login funkcija ..... 56
Slika 27. Dashboard komponenta koja preusmjerava na ostale komponente ovisno o URL-u 57
Slika 28. Primjer oblikovanja elementa korištenjem styled-components biblioteke ..... 57
Slika 29. Kreiranje AVCaptureMetadataOutput varijable ..... 58
Slika 30. Delegatska metoda ..... 58

Slika 31. Dostupne rute koje koriste HAL-API paket ..... 59
Slika 32. WarehouseController – primjer HAL-API resource kontrolera ..... 60
Slika 33. Primjer modela ..... 60
Slika 34. WarehouseRepository - primjer HAL-API repozitorija ..... 61
Slika 35. WarehouseTransformer - primer HAL-API transformer-a ..... 61
Slika 36. Primjer poveznice u AppServiceProvider klasi ..... 62
Slika 37. Primjer povezivanja parametara rute ..... 62
Slika 38. Povezivanje parametra rute u projektu ..... 63
Slika 39. Primjer JSON odgovora ..... 64
Slika 40. Prijava u web aplikaciju ..... 65
Slika 41. Primjer neuspješnog logiranja u sustav ..... 65
Slika 42. Početna stranica za poduzeće nakon uspješne prijave ..... 66
Slika 43. Primjer uspješnog dodavanja zaposlenika ..... 67
Slika 44. Obrazac za dodavanje skladišta ..... 68
Slika 45. Prikaz tablice s listom skladišta ..... 68
Slika 46. Primjer ispunjenog obrasca za dodavanje proizvoda ..... 69
Slika 47. Primjer liste proizvoda ..... 70
Slika 48. Potvrda brisanja proizvoda ..... 70
Slika 49. Prikaz informacija o proizvodu koje se mogu ažurirati ..... 71
Slika 50. Primjer stanja zaliha u skladištu ..... 72
Slika 51. Primjer liste prijema robe za odabrani datum ..... 73
Slika 52. Primjer popisa proizvoda pristiglih za odabrani prijem robe ..... 73
Slika 53. Izgled stranice za promjenu osobnih podataka ..... 74
Slika 54. Prijava u mobilnu aplikaciju ..... 74
Slika 55. Prikaz glavnog izbornika za poduzeće ..... 75
Slika 56. Primjer popisa zaposlenika ..... 76
Slika 57. Primjer popisa skladišta ..... 76
Slika 58. Primjer uspješnog skeniranja barkoda ..... 77
Slika 59. Prikaz glavnog izbornika za zaposlenika ..... 78
Slika 60. Primjer prikaza liste proizvoda ..... 78
Slika 61. Primjer skeniranja kod prijema robe i dodavanja količine ..... 79
Slika 62. Primjer prikaza kreiranih sesija ..... 79
Slika 63. Primjer skeniranih proizvoda za odabranu sesiju ..... 80
Slika 64. Primjer skeniranja proizvoda da se provjeri količina na stanju ..... 80

# SAŽETAK

Ovaj rad se temelji na teorijskom i praktičnom djelu razvoja aplikacije za vođenje evidencija skladišta. Kroz teorijski dio objašnjeni su svi pojmovi vezani uz jezike i tehnologije koje se koriste za izradu praktičnog dijela, gdje na najbitniji dio REST API. U praktičnom je dijelu opisana izrada aplikacije. Da bi korisnici mogli voditi evidenciju skladišta realizirana je web aplikacija razvijena u Laravel okviru kao pozadina i React okviru kao sučelje web aplikacije, te mobilna aplikacija razvijena u Swift okviru. Obje aplikacije komuniciraju s REST API-jem kako bi se što više zaštitili podaci i olakšalo korištenje API-ja, gdje se URI u bilo kojem trenutku može promijeniti bez utjecaja na radnju aplikacija.

**Ključne riječi**: prijam robe, RESTful servisi, API, mobilna aplikacija, web aplikacija, baza podataka

# SUMMARY

This paper builds on the theoretical and practical work of developing a warehouse records management system. Throughout the theoretical part, all the concepts related to the languages and technologies used to create the practical part are explained, where the most important part is REST API. The practical part describes how system was build. To enable users to keep records of the warehouse, a web application was developed in the Laravel framework as a backend and a React framework as a frontend of web application that was implemented, and a mobile application developed in the Swift framework. Both applications interact with the REST API to protect the data as much as possible and to facilitate the use of the API, where the URI can be changed at any time without affecting the actions of the applications.

Keywords: goods receipt, RESTful services, API, mobile application, web application, database