etf+grb **Универзитет у Источном Сарајеву**

**Електротехнички факултет**

**Развој прогресивних веб апликација у *Vue.js framework-*у**

ЗАВРШНИ РАД

Студент: Ментор:

*Огњен Лубарда Доц. др. Данијел Мијић*

Источно Сарајево, децембар 2021.

Садржај

[1. Увод 2](#_Toc90104032)

[1. Прогресивне веб апликације 3](#_Toc90104033)

[2.1. Карактеристике прогресивне веб апликације 3](#_Toc90104034)

[2.2. Кад веб-сајт постаје прогресивна веб апликација? 4](#_Toc90104035)

[2.3. Алати који се користе за развијање прогресивних веб апликација 6](#_Toc90104036)

[3. Ларавел 9](#_Toc90104037)

[3.1. Кратка историја Ларавела 10](#_Toc90104038)

[3.2. Ларавел у односу на друге оквире и серверске jeзике 10](#_Toc90104039)

[3.2.1. Ларавел у односу на друге *PHP* оквире 11](#_Toc90104040)

[3.2.2. Ларавел у односу на друге серверске језике 12](#_Toc90104041)

[3.3. Структура Ларавел пројекта 13](#_Toc90104042)

[3.4. Основни концепти Ларавела 15](#_Toc90104043)

[3.4.1. Рутирање 15](#_Toc90104044)

[3.4.2. *Middleware* 16](#_Toc90104045)

[3.4.3. *Laravel* *Blade* 17](#_Toc90104046)

[3.4.4. *Laravel Sanctum* 18](#_Toc90104047)

[*4.* *Vue.js* 20](#_Toc90104048)

[4.1. Поређење са другим Јаваскрипт оквирима 20](#_Toc90104049)

[4.2. Основни дијелови *Vue.js* компоненте 22](#_Toc90104050)

[4.3. Динамичко везивање података (енгл. *data binding*) 25](#_Toc90104051)

[4.4. *Vue* и прогресивне веб апликације 26](#_Toc90104052)

[5. Практична реализиација веб апликације 27](#_Toc90104053)

[5.1. Мотивација за израду апликације 27](#_Toc90104054)

[5.2. Технички опис апликације 28](#_Toc90104055)

[5.3 Функционалности апликације 32](#_Toc90104056)

[Литература 38](#_Toc90104057)

[Додаци 39](#_Toc90104058)

[Слике 39](#_Toc90104059)

[Табеле 40](#_Toc90104060)

[Кодни листинзи 40](#_Toc90104061)

# Увод

Циљ рада је практична реализација прогресивне веб апликације за читање мјерења различитих величина у пластенику. Апликација треба да омогући приказ мјерења у реалном времену као и тренд промјене у временском периоду који корисник изабере.

Прогресивне веб апликације постепено постају свакодневница развоја веб-сајтова због своје ефикасности и исплативости приликом развоја. Помоћу ове апликације, корисници имају увид у стање у пластенику/башти без да буду присутни на лицу мјеста.

Рад је организован у четири дијела. У другом поглављу је описано шта су прогресивне веб апликације и како се развијају, као и како постојећи веб-сајт претворити у прогресивну веб апликацију.

У трећем поглављу је описан Ларавел и његове функционалности. Упоређен је са другим серверским алатима и описани неки основни концепти.

У четвртом поглављу је описан *Vue.js.* Упоређен је са другим Јаваскрипт оквирима и кратко је описана структура и функционалност *Vue.js*. Такође је објашњено зашто је *Vue* погодан за развој прогресивних веб апликација.

У петом поглављу је описана практична реализација апликације. Побројане су и укратко описане друге технологије и алати који су кориштени. На крају је објашњена функционалност и начин употребе апликације.

# Прогресивне веб апликације

Прогресивне веб апликације представљају тип софтвера креиран помоћу стандардних веб алатa. У основи, прогресивне веб апликације би се могле дефинисати као веб-сајтови који кориснику пружају осјећај коришћења изворних (енгл. *native)* мобилних апликација.

## 2.1. Карактеристике прогресивне веб апликације

Да би се приказале карактеристике прогресивне веб апликације, најбоље је кренути од предности и мана коришћења веб-сајтова у односу на изворне мобилне апликације.

Предност веб-сајтова представља њихова доступност. Веб-сајтови су доступни свакоме са интернет конекцијом и инсталираним веб читачем. Лако се дијеле међу корисницима, без потребе за додатним инсталирањем. Један код је довољан, независно од уређаја којим ће корисник приступити. Међутим, нису увијек *user-friendly,*  и зависе од интернет конекције, без интернета нема ни учитавања веб-сајта.

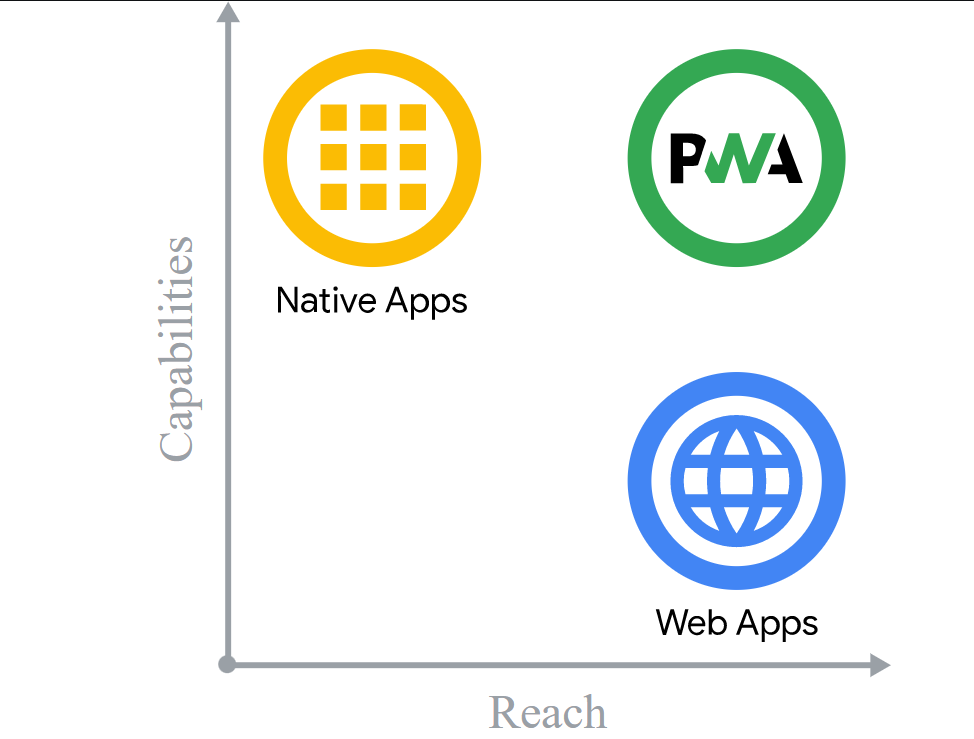
За разлику од веб-сајтова, предност изворних апликација је њихова могућност. Изворне апликације кориснику пружају богат доживљај приликом коришћења. Не зависе од интернет конекције, могу читати податке са локалног складишта уређаја, приступити уређају повезаним преко *USB-*а*,* користити његову камеру, блутут, податке из других апликација (нпр. користити календар, контролисати музику и сл.). Изворне веб апликације је могуће поставити на почетни екран гдје су увијек на дохват руке за поновно покретање.

Међутим, иако пружају много веће могућности од веб-сајтова, изворне апликације су скупe за реализацију и одржавање, и потребна је инсталација. То значи да је за изворну апликацију потребно издвојити меморију на уређају, што, посебно код старијих уређаја представља слабу тачку.

Креирање и одржавање изворних апликација које су доступне свима заправо представља креирање и одржавање двије апликације, *Android*  и *iOS,*  самим тим су неопходна и два тима која ће развијати и одржавати те апликације. Такође, да би уопште биле доступне кориснику за преузимање потребно их је пласирати на неку од продавница (нпр. *Google Play*).

Ту се долази до предности прогресивних веб апликација које су узеле најбоље од обје технологије.

На слици 1.1. су приказане могућности и доступност прогресивних веб апликација у односу на веб и изворне апликације. Прогресивне веб апликације представљају могућности изворне апликације. Могу се поставити на почетни екран, претраживати на уређају. Оне раде и без интернет конекције, и корисник може поново ући у странице у које је ушао док је постојала конекција, прогресивне веб апликације ће приказати задње доступне вриједности и новости. Иако се и оне морају сачувати да би биле доступне на почетном екрану, прогресивне веб апликације заузимају и до 25 пута мање меморије у односу на изворну апликацију, и до 15 пута се брже учитају[1]. Прогресивне веб апликације могу слати такозвана *push* обавјештења, и синхронизовати податке у позадини.



Слика 1.1. могућности и доступност апликација

Али, пошто су прогресивне веб апликације заправо веб-сајтови, оне се могу лако дијелити преко линкова, доступне су свима и потребно је развити само један код који ће радити на свим уређајима. Сачуване прогресивне веб апликације се покрећу у независном прозору (а не у веб читачу), што им омогућава да буду много приступачније кориснику него стандрадни веб-сајт.

Можда и најбољи показатељ ефикасности прогресивних веб апликација је статистика коју су подијелиле компаније које су прешле на прогресивне веб апликације. Твитер је пријавио 65% већу посјећеност страницама, 75% више твитова, и 20% мањи *bounce – rate*  односно број корисника који је напустио страницу, и то све док су смањили величину апликације за чак 97%. *Nikkei* је примјетио 2.3 пута већи саобраћај, 58% више претплатника и 49% више корисника дневно. [2] Joш неке од познатих компанија које користе прогресивне веб апликације су Пинтерест, Убер, Форбс, и можда најпопуларнији међу њима, иако још није имплементирао све функционалности изворне апликације – Инстаграм.

## 2.2. Кад веб-сајт постаје прогресивна веб апликација?

Иако званично није дефинисано, постоје 3 основна услова која је потребно испунити да би се неки веб-сајт сматрао прогресивном веб апликацијом:

* Креирање респонзивног веб-сајта
* Креирање *JSON* манифеста
* Учитавање услужних скрипти

Креирање респонзивног веб-сајта у суштини значи креирање сајта чији ће се садржај прилагодити величини екрана корисника.

Креирање *JSON* манифеста је креирање једноставне *JSON* датотеке која у себи садржи податке попут назива апликације, иконицa које се користе и почетнe урл адресe. Сврха ове датотеке је посједовање инструкција за веб читаче, односно како да покрену прогресивну веб апликацију када је она сачувана на уређају корисника.

{

"name": "Ocitavanje mjerenja - diplomski rad",

"short\_name": "Ocitavanje mjerenja",

"start\_url": "/",

"display": "standalone",

"theme\_color": "#0476F2",

"background\_color": "#fff",

"icons": [

{

"src": "/img/icons/android-chrome-192x192.png",

"type": "image/png",

"sizes": "192x192",

"purpose": "any maskable"

},

{

"src": "/img/icons/android-chrome-512x512.png",

"type": "image/png",

"sizes": "512x512"

}

]

}

Кодни листинг 1.1. Примјер JSON манифеста

Оно што неку прогресивну веб апликацију чини изворном су услужне скрипте (енгл. *service worker*). То су (најчешће) Јаваскрипт датотеке које у себи садрже податке попут које ће се информације сачувавати на уређају, а које ће се учитавати приликом конекције са интернетом. Услужне скрипте су у потпуности асинхроне, тако да синхрони захтјеви и локална меморија не могу бити кориштени као услужне скрипте. Услужне скрипте раде независно од главне нити. Оне могу примити *push* поруке од сервера и кад апликација није покренута, што омогућује прогресивној веб апликацији да прикаже *push* обавјештења и кад корисник не користи апликацију.Међутим, услужне скрипте не могу приступити директно *DOM-у (*енгл. *Document Object Model),* па за комуникацију са страницом, услужне скрипте користе  *postMessage()* методу за слање порука и догађај *message* за примање порука.

Рад услужне скрипте је регулисан догађајима. Неки од њих су *fetch, push, sync, activate, install, message.* Да би апликацији омогућила да ради без интернет конекције, услужна скрипта користи два *API-ja, Fetch* и *Cache. Fetch* је стандардан начин дохваћања података са мреже, док *Cache* представља складиште за дохваћене податке. Ово кеш складиште није исто што и кеш складиште веб читача. Животни циклус услужне скрипте се састоји од регистрације, инсталације и активације. Једном активирана, услужна скрипта ослушкује све догађаје страница за које је регистрована.

Услужне скрипте раде само преко *HTTPS* протокола, jer су због своје немогућности да пресретну мрежне захтјеве и модификују одговоре подложни нападима *man-in-the-middle.* [3]

Гугл је поставио следеће услове које је неопходно да сајт испуни да би се могао сматрати прогресивном веб апликацијом:

* Сајт користи *HTTPS* протокол
* Сајт је респонзиван (садржај се прилагођава величини екрана)
* Кад се приступа без интернет конекције, неопходно је приказати бар почетну страницу
* Може се додати иконица на почетни екран на телефону
* Интеракција је могућа за мање од 10 секунди кад се приступа преко 3Г мреже
* Сајт је подржан и оптимизован за прегледање у различитим веб читачима
* Свака страница унутар апликације посједује посебну урл адресу
* Док се чека учитавање странице, страница није замрзнута (покреће се нека иконица или анимација као и на изворним апликацијама) [4]

Иако је један од услова да апликација буде прилагођена свим веб читачима, неки од читача не подржавају прогресивне веб апликације. Мозила је на почетку подржавала прогресивне веб апликације, али је у децембру 2020. године укинула ту подршку. [5] Ово наиме важи за Мозила веб читач – десктоп верзију, Мозила веб читач верзија за телефоне и даље пружа подршку за прогресивне веб апликације.

## 2.3. Алати који се користе за развијање прогресивних веб апликација

Пошто су прогресивне веб апликације заправо веб-сајтови, за њихово развијање се могу користити основни алати који се користе и за развијање веб-сајтова:

* *Html*
* *CSS*
* *Javascript*

*Html (*енгл. *HyperText Markup Language)* представља основни градивни блок сваког веб-сајта. *Html* документи имају екстензију .*html*. *Html* је описни језик, и самим тим нема могућност креирања динамичког садржаја. Он дефинише структуру веб странице у облику блокова. *Html* елементи су одвојени почетним и завршним таговима, а основна струкура *Html* документа и уједно примјери неких тагова су приказани у кодном листингу 1.2.

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

</head>

<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>

</html>

Кодни листинг 1.2. Структура Html документа и примјер тагова

*CSS (*енгл. *Cascading Style Sheet)* представља језик којим се дефинише изглед веб-сајта и уз *Html* је основа сваког веб-сајта. Јавио се као потреба да се код којим се дефинише изглед веб-сајта издвоји од његове структуре. Омогућава дефинисање стилова који се могу примјенити и промијенити на више елемената одједном.

html **{**

**font-family: sans-serif;**

**line-height: 1.15;**

**-webkit-text-size-adjust: 100%;**

**-webkit-tap-highlight-color: rgba(0, 0, 0, 0);**

**}**

article**,** aside**,** figcaption**,** figure**,** footer**,** header**,** hgroup**,** main**,** nav**,** section **{**

**display: block;**

**}**

body **{**

**margin: 0;**

**font-family:** "Nunito"**, sans-serif;**

**font-size: 0.9rem;**

**font-weight: 400;**

**line-height: 1.6;**

**color: #212529;**

**text-align: left;**

**background-color: #f8fafc;**

**}**

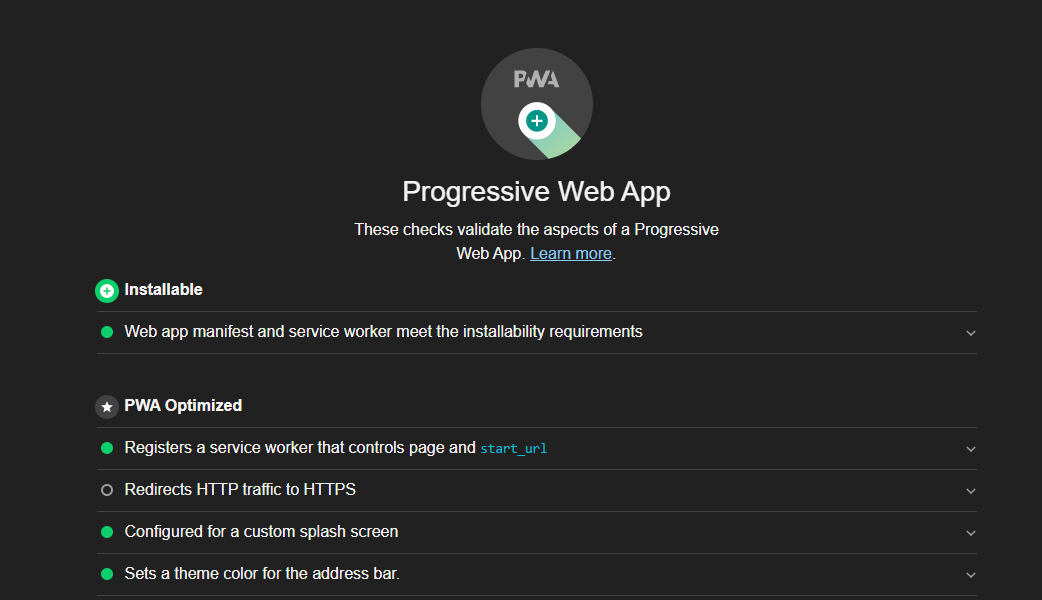
Кодни листинг 1.3. Примјер CSS кода

Јаваскрипт је широко распрострањен објектно-орјентисан језик првенствено намијењен за развој веб-сајтова. Омогућава извршавање динамичких радњи у статичким *Html* страницама.

Из Јаваскрипт програмског језика су временом настали многи оквири и библиотеке које су програмерима омогућили лакши развој страница са више функционалности и визуелних ефеката. Са развојем ових оквира омогућен је и развој прогресивних веб апликација.

Напреднији алати који се користе данас за развој прогресивних веб апликација су:

* *AngularJS* представља један од најстаријих Јаваскрипт оквира, развијен од стране Гугла. Има једну од највећих база корисника и редовна ажурирања.
* *React PWA* нуди добру подлогу за развој прогресивних веб апликација и представља библиотеку *ReactJS* оквира, уз *Angular* и *Vue* најпопуларнијег Јаваскрипт оквира на свијету. Развијен је од стране Фејсбука.
* *Vue.js* је један од најмлађих Јаваскрипт оквира који је стекао огромну популарност међу програмерима својом једноставношћу. Један је од оквира који се најбрже развијају и постиже велику брзину рендеровања. Више о овом оквиру ће бити у поглављима испод.
* *Ionic* представља моћан алат за развој прогресивних веб апликација, настао на *Apache Cordovi* и *AngularJS* окружењу. Пружа повољности попут *UI* елемената, тестирања, рутирања..
* *Polymer* је један од многих алата развијених од стране Гугла који се могу користити за развијање прогресивних веб апликација.
* *PWA Builder* је алат подржан од стране *Microsoft-a* чија је основна намјена претварање веб-сајтова у прогресивне веб апликације уз мало рада. Користе га већина веб компанија које се баве развојем прогресивних веб апликација.
* *Google Lighthouse* је алат који процијењује колико је неки веб-сајт заправо прогресивна веб апликација, и препоручује измјене које би довеле до побољшања прогресивне веб апликације. Извјештај овог алата је приказан на слици 1.2.



Слика 1.2. Извјештај Lighthouse алата

# Ларавел

У овом пројекту, серверска страна је одрађена помоћу Ларавел оквира. Ларавел је бесплатни *PHP* оквир отвореног кода(енгл. *open source*), развијен од стране Тејлора Отвела (енгл. *Taylor Otwell*). Главна намјена Ларавела је развој веб-сајтова користећи *Model-View-Controller(*скраћено *MVC)* архитектуру. Најпознатији веб-сајтови који користе Ларавел су *BBC, 9GAG, Pfizer.*

## Кратка историја Ларавела

Ларавел је настао 2011. године као напреднија алтернатива *CodeIgniter* оквиру који није имао уграђену подршку за одређена својства, попут аутентификације и ауторизације.

Ларавел 1 је настао у јуну 2011. године а имао је уграђену подршку за аутентификацију, локализацију, модел, поглед, сесије и друге погодности, али није имао уграђену подршку за контролер, па се није могао сматрати правим *MVC* оквиром.

Ларавел 2 је објављен у септембру 2011. са многим побољшањима од којих је најзначајнија подршка за контролер, док је главни недостатак Ларавела 2 укидање подршке за пакете из других извора.

Ларавел 3 је објављен фебруара 2012. године. Главнe одлике Ларавел 3 су додавање командног интерфејса *Artisan* и веће управљање са системима базе. Са верзијом 3, Ларавел стиче све већу популарност међу корисницима.

Ларавел 4 је објављен у мају 2013. године. Кодни назив му је *Illuminate* и представља комплетну реформу дотадашњег оквира. Распоређен је у пакете којим управља *Composer.* Осим реформе оквира, у овој верзији су додате подршке за електронску пошту и почетно попуњавање базе података (енгл. *seeding*).

Ларавел 5 је верзија која је временски најдуже трајала. Са својим подверзијама, трајала је од фебруара 2015. до септембра 2019. Неке од могућности које су додате у овој верзији су *Dusk, Mix, Blade* компонент и др.

Ларавел 6 је објављен у септембру 2019. године и одликује се побољшањем дотадашњих могућности. Такође, фронтенд је пребачен и главног пакета у пакет *laravel/ui.*

Ларавел 7 је објављен марта 2020. године са додацима попут *Sanctuma.*

Најновија верзија је Ларавел 8, објављена 8. септембра 2020. године. Нови додаци су *Jetstram, model factory* класе, побољшање миграције и друго. [6]

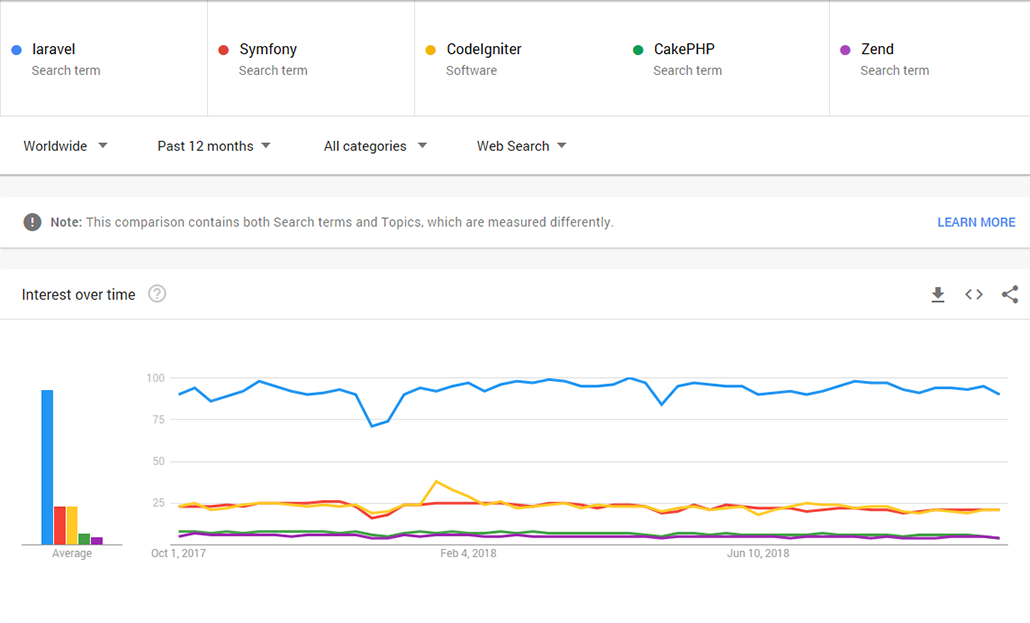
Овај пројекат користи верзију 8.65.

## Ларавел у односу на друге оквире и серверске jeзике

Постоје многе технологије за развијање серверске стране, а у овом поглављу ће Ларавел укратко бити упоређиван са најпознатијима. Кроз ово упоређивање биће објашњене и неке основне карактеристике Ларавела.

### Ларавел у односу на друге *PHP* оквире

*PHP* је један од најстаријих и најпопуларнијих програмских језика за развијање серверске стране веб-сајтова. Као такав, подлога је за развој многих оквира од којих су најпопуларнији *Symfony, CodeIgniter, CakePHP,* Ларавел и  *Zend.* На слици 2.1. је график који показује процентуалан број претрага ових оквира у задњих дванаест мјесеци, гдје је Ларавел увелико најпретраживанији *PHP* оквир.



Слика 2.1. Процентуалан број Гугл претрага *PHP оквира* у задњих годину дана

*Symfony* је други најпопуларнији *PHP* оквир и највећи ривал Ларавелу. Има велику базу корисника. Добро познате платформе попут Магента користе *Symfony,* и *Symfony* се ажурира редовно. Међутим, сналажење и учење у *Symfony*-у је много теже за почетнике у односу на Ларавел. *Symfony* је добар за велике и комплексне пројекте, док је Ларавел бољи за мање пројекте који захтјевају брз развој апликације. У Ларавелу је много лакше подесити ауторизацију и аутентификацију, потребно је само подесити базу и моделе. Што се тиче брзине, Ларавел је доста бржи у односу на *Symfony,* док је код *Symfony-*а чистији и организованији. Ларавел користи *Blade* као шаблонску машину(енгл. *template engine)*, док *Symfony* користи *Twig.* Главна разлика између ова два шаблона је што *Blade* подржава *PHP* код. Код локализације и превођења, Ларавел подржава само *PHP* и *JSON* датотеке, док *Symfony* осим ових подржава још и *Qt, .po, .mo, CSV, INI... Symfony* користи компоненте које се могу употријебити вишекратно, али сам *Symfony* није прави *MVC* оквир за разлику од Ларавела. Што се тиче цијене, *Symfony*  је доста скупљи за одржавање и документација је шкртија у односу на Ларавел. [7] [8]

*CodeIgniter* je једноставнији од Ларавела и брже се учи, међутим, нема већину функционалности које Ларавел посједује. *CodeIgniter* је бољи само када су изузетно мали и брзи пројекти, док је Ларавел бољи у сваком другом случају. Неке од функционалности које Ларавел посједује,а *CodeIgniter* не посједује су објектно-релационо мапирање, шаблонска машина, уграђену аутентификацију, модуле пакирањa..[9]

И док је и *CakePHP* код отвореног типа као и Ларавел, за *CakePHP* је потребна лиценца. *CakePHP* за разлику од Ларавела који користи објектно орјентисан модел, користи документационо оријентисан модел. *CakePHP* користи хијерархијску *MVC* архитектуру (скраћено *НMVC)* и не подржава процес везивања података (Ларавел подржава двосмјерно везивање података). *CakePHP* је бољи за мање пројекте и пружа већу сигурност. Међутим, Ларавел је много бољи у смислу брзог развоја, резервне копије податакa и скалирања. У суштини, Ларавел је бољи за све осим за мале пројекте.[10]

Иако је по структури сличан Ларавелу, *Zend* се теже учи, процесуира мање захтјева одједном и спорији је у односу на Ларавел. Још један од недостатака *Zend*-а је што се за сваку функционалност мора инсталирати посебан пакет.

### Ларавел у односу на друге серверске језике

Према Гугловој претрази, 5 најпопуларнијих језика за развијање серверске стране су Јава, *C#, PHP, Phyton* и *Node.* У овом поглављу ће Ларавел бити упоређиван са најпознатијим оквирима ових језика.

*Spring* представља Јава оквир за развој апликација. Као и Ларавел, и *Spring* користи *MVC* архитектуру. Иако су наизглед слични оквири, *Spring* има предност, јер потиче од богатијег језика који је способнији за више ствари – Јаве. Ларавел има предност само код мањих пројеката гдје се његова лака примјена истиче.

*Asp.net* је Мајкрософтов оквир развијен за развој веб апликација. За разлику од Ларавела који пружа хоризонтално скалирање, *Asp.net* пружа вертикално скалирање. *Asp.net* је генерално моћније окружење, али је и скупље од Ларавела, тако да се за *Asp.net* одлучују веће компаније са великим пројектима и буџетима. Ларавел је више намијењен за *user-friendly* сајтове док је *Asp.net* више намијењен за безбиједност и функционалност. Ларавел је отвореног типа док *Asp.net* није, и потребна је Мајкрософтова лиценца. Још један од недостатака *Asp.net*-а у односу на Ларавел је теже учење самог оквира.

*Django* представља оквир програмског језика *Phyton*. Истиче се бољом сигурношћу у односу на Ларавел. Има једноставну криву учења и велику базу корисника. Међутим, монолитан је (и фронтенд и бекенд су на истом серверу) и није погодан за развијање мањих пројеката.

*NodeJS*  је не само Јаваскрипт окружење већ и сервер. Одличан је за изградњу *API*-ja, има велику базу библиотека и перфомансе апликација направљених помоћу *NodeJS*  су веће него помоћу Ларавела. Недостатак *NodeJS* је што рад са *NodeJS*  укључује асинхроно програмирање и није баш најпоузданији у смислу сигурности. *NodeJS*  је покренут догађајима.

## Структура Ларавел пројекта

Структура Ларавел пројекта и кратко објашњење је приказана у табели 1.

|  |  |
| --- | --- |
| app | Директоријум који садржи основни код Ларавел апликације |
| bootstrap | Садржи bootstrap скрипте кориштене у апликацији |
| config | Садржи све конфигурационе фајлове (.*config*) |
| database | Садржи датотеке везане за базу података |
| public | Директоријум у који се најчешће смјештају .*css* и .*js* датотеке, као и слике и други мултимедијални садржај |
| resources | Садржи .*sass* датотеке, датотеке језика( за превођење) као и шаблоне |
| routes | Садржи све дефиниције рутирања фајлова као што су *console.php, api.php , channels.php* |
| storage | Садржи разне датотеке генерисане од стране окружења, као и датотеке сесије, кеша и слично. |
| tests | Садржи тестне случајеве |
| vendor | Садржи датотеке зависности *Composer*-a |

Табела 1. Структура Ларавел пројекта

Структура *app* директоријума и кратко објашњење је приказано у табели 2.

|  |  |
| --- | --- |
| Console | Садржи све *artisan* команде |
| Events | Садржи дефиниције за догађаје |
| Exceptions | Садржи дефиниције изузетака које Ларавел избацује током дебаговања |
| Http | Садржи различите филтере, захтјеве и контролере |
| Jobs | Садржи све поставне послове. Потребно га је креирати ручно командом *make:job* |
| Listeners | Садржи дефиниције за све ослушкиваче који примају и рјешавају догађаје |
| Mail | Садржи све мејлове послате кроз Ларавел пројекат. Потребно га је креирати ручно командом *make:mail* |
| Notifications | Садржи обавјештења послата кроз Ларавел пројекат. Потребно га је креирати ручно командом *make:notification* |
| Policies | Садржи различите полисе везане за пројекат |
| Providers | Садржи различите сервисне провајдере |
| Rules | Садржи различита правила дефинисана од стране програмера. Потребно га је креирати ручно командом *make:rule* |

Табела 2. Структура App директоријума

## Основни концепти Ларавела

### 3.4.1. Рутирање

Рутирање је један од основних концепата Ларавела и омогућује повезивање апликације са одговарајућим контролером или путањом. Све датотеке рутирања се налазе у директоријуму *routes.*

Основни облик руте која позива неки контролер је:

use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'index']);

Кодни листинг 2.1. примјер позивања руте у Ларавелу

Осим *get* методе, Ларавел подржава *post, put, patch, delete* и *options.* Такође, могуће је позвати више метода на једној рути која тад има облик:

Route::match(['get', 'post'], '/', function () {

//

});

Кодни листинг 2.2. повезивање више метода у рути

Уколико се желе искористити све методе, умјесто кључне ријечи *match* користи се кључна ријеч *all.*

Помоћу рутирања је могуће послати и параметре који се означавају витичастим заградама {}.

Route :: get ('emp/{id}', function ($id) {

echo 'Emp '.$id;

});

Кодни листинг 2.3. прослијеђивање параметра рутирању

Опциони параметри се означавају додавањем знака „?“ послије параметра (нпр. {id?}). [11]

### 3.4.2. *Middleware*

*Middleware*  представља механизам који испитује и филтрира *HTTP* захтјеве који улазе у апликацију. На примјер, Када корисник пошаље захтјев за аутентификацију *middleware* провјерава да ли је корисник аутентификован. Уколико није, *middleware* ће усмјерити корисника на страницу за пријављивање (енгл. *login page),* у супротном, *middleware* ће дозволити захтјеву да настави са процесуирањем.

*Middleware*  се креира командом:

php artisan make:middleware<middleware\_name>

*Middleware*  се може регистровати глобално и локално (*route middleware*). Глобално регистровање *middleware*-а се врши у датотеци *app/Http/Kernel.php,*  у пропертију *$middleware,* док се појединачни(локални) *middleware* региструје додавањем жељене руте у листу параметра *$routeMiddleware.*

protected $routeMiddleware = [

'auth' => \Illuminate\Auth\Middleware\Authenticate::class,

'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,

'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,

'userAuth' => \Illuminate\Routing\Middleware\UserAuthRequests::class,

];

Кодни листинг 2.4. Повезивање middleware-а са рутом

Као и приликом рутирања, параметри се могу послати *middleware-*у. На примјер, уколико се жели омогућити кориснику преглед странице у зависности од његове улоге, онда је слање параметра *middleware-*у пожељна ствар. Жељени параметри се наводе након *$next*  аргумента. [11]

<?php

namespace App\Http\Middleware;

use Closure;

class EnsureUserHasRole

{

/\*\*

\* Handle the incoming request.

\*

\* @param \Illuminate\Http\Request $request

\* @param \Closure $next

\* @param string $role

\* @return mixed

\*/

public function handle($request, Closure $next, $role)

{

if (! $request->user()->hasRole($role)) {

// Redirect...

}

return $next($request);

}

}

Кодни листинг 2.5. Слање параметара middleware-у

### 3.4.3. *Laravel* *Blade*

За разлику од неких других *PHP* оквира, Ларавел користи *Blade* шаблонску машину. Главна карактеристика *Blade*-а је могућност кориштења *PHP* кода унутар саме *Blade* датотеке. Ово је могуће зато што се *Blade* шаблони компајлирају у чисти (енгл. *plain)* *PHP* код и кеширају све док се не модификују. *Blade* датотеке се обично налазе у директоријуму *resources/view* и имају наставак *.blade.php.*

Да би се приступило овим датотекама потребно их је прво регистровати у рутама.

То се ради на следећи начин:

Route::get('/', function () {

return view('greeting', ['name' => 'Finn']);

});

Кодни листинг 2.6. Регистровање view-a

Параметар ['name' => 'Finn'] је опцион, и представља параметар који се може прослиједити *Blade*-у, а у самој *Blade* датотеци се приступа помоћу двоструких витичастих заграда *{{name}}.* Прослијеђени параметар може бити и функција. Пошто Ларавел подржава рад са Јаваскрипт оквирима, од којих неки користе исту ознаку, потребно је прослиједити симбол „*@“* испред заграда *@{{name}}* чиме се *Blade* обавјештава да назначени параметар треба остати нетакнут*.* Изрази у витичастим заградама су аутоматски послати кроз *PHP* *htmlspecialchars* функцију, чиме се спријечавају *Cross-Site Scripting*  (скраћено *XSS*)напади. [11]

Поред класичног *Html* кода, у *Blade*-у се могу писати *PHP* петље и изрази.

@for ($i = 0; $i < 10; $i++)

The current value is {{ $i }}

@endfor

@foreach ($users as $user)

<p>This is user {{ $user->id }}</p>

@endforeach

@forelse ($users as $user)

<li>{{ $user->name }}</li>

@empty

<p>No users</p>

@endforelse

@while (true)

<p>I'm looping forever.</p>

@endwhile

Кодни листинг 2.7. Примјер кориштења Html и PHP кода у Blade шаблону

### 3.4.4. *Laravel Sanctum*

*Sanctum* представља изузетно лаган систем аутентификације за апликације са једном страницом (енгл. *Single Page Applications – SPA),* мобилне апликације, као и једноставне *API*-je засноване на токенима. Рад *Sanctum*-а је објашњен на примјеру регистрације новог корисника. У новијим верзијама Ларавела *Sanctum* долази као преинсталирана функционалност.

Да би модел корисника могао да користи *Sanctum* потребно је да посједује токен што се провјерава помоћу *HasApiTokens.*

// app/Models/User.php

use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable

{

use HasApiTokens, HasFactory, Notifiable;

}

Кодни листинг 2.8. User модел

Након тога је потребно креирати AuthController помоћу php artisan make:controller AuthController, и потребно је регистровати API у routes/api.php.

// routes/api.php

use App\Http\Controllers\AuthController;

Route::post('/register', [AuthController::class, 'register']);

Кодни листинг 2.9. Регистровање руте AuthController-а

Сам *AuthController* треба да изгледа овако:

// app/Http/Controllers/AuthController.php

use Illuminate\Support\Facades\Hash;

public function register(Request $request)

{

$validatedData = $request->validate([

'name' => 'required|string|max:255',

'email' => 'required|string|email|max:255|unique:users',

'password' => 'required|string|min:8',

]);

$user = User::create([

'name' => $validatedData['name'],

'email' => $validatedData['email'],

'password' => Hash::make($validatedData['password']),

]);

$token = $user->createToken('auth\_token')->plainTextToken;

return response()->json([

'access\_token' => $token,

'token\_type' => 'Bearer',

]);

}

Кодни листинг 2.10. AuthController

Прво се провјерава да ли су примљени параметри валидни ($validateData). Уколико одговарају траженим условима, параметри се уписују у базу података. Када се креира нови корисник, креира се нови лични токен методом createToken(). Пошто ова метода враћа објекат типа Laravel\Sanctum\NewAccessToken помоћу plainTextToken се тај токен претвара у текст. Коначно, враћа се објекат типа JSON који у себи садржи сам токен, као и тип токена. [11]

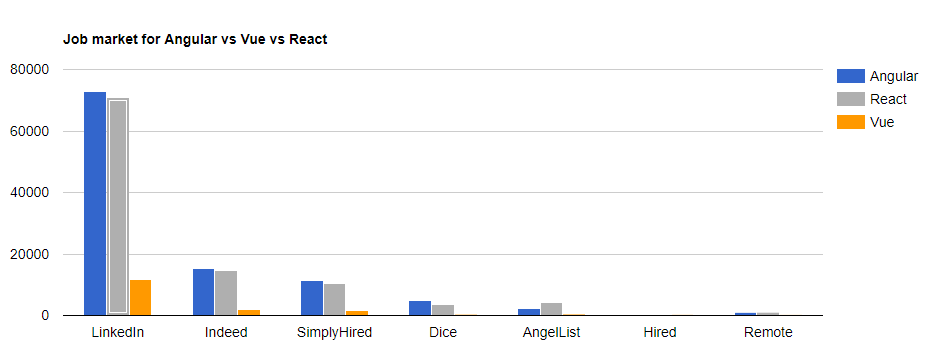
# *Vue.js*

Алат који је кориштен у овом пројекту за развој клијентске стране апликације је *Vue.js*. *Vue.js* (скраћено *Vue*) представља прогресивни оквир Јаваскрипта намјењен за креирање корисничког интерфејса и веб-сајтова са једном страницом. Настао је 2014. а креирао га је Eвн Ју (енгл. *Evan You*), бивши инжењер у Гуглу. Ју је створио овај оквир користећи дијелове Ангулара. Тренутна верзија оквира је 3.2.20 а у овом пројекту је кориштена верзија 2.6.12.

За разлику од других сличних оквира, *Vue* је у потпуности прилагодљив и усвојив. Основна бибилиотека се заснива само на слоју погледа (енгл. *view layer*), и самим тим је врло једноставно уградити *Vue* оквир у већ постојеће бибилиотеке и пројекте. Све напредне могућности које пружа *Vue* су омогућене са додатним пакетима и библиотекама, од којих ће оне које су кориштене у раду бити обрађене у овом поглављу.

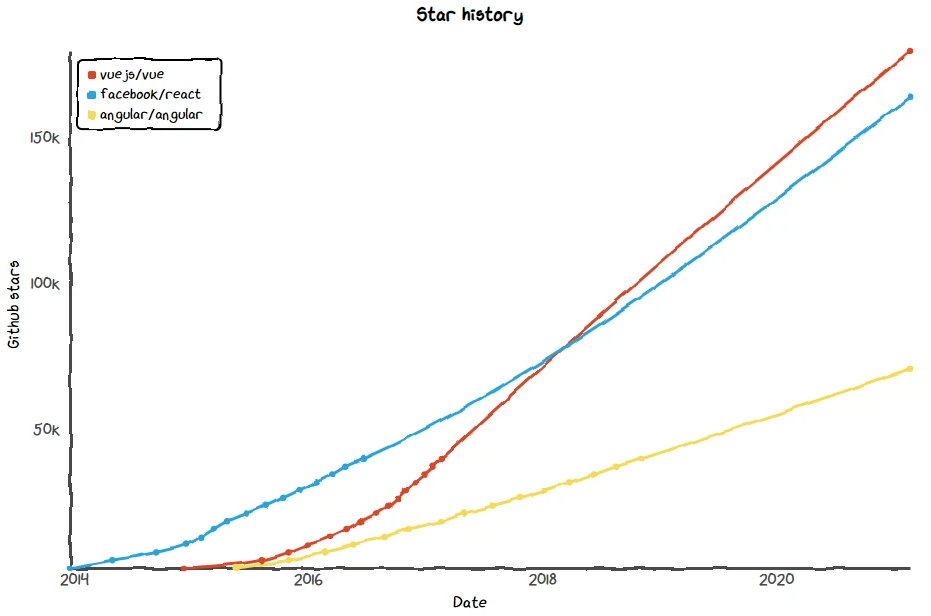
## 4.1. Поређење са другим Јаваскрипт оквирима

Три најпопуларнија фронтенд Јаваскрипт оквира су *React, Angular* и *Vue.* Од наведених, *Vue* је најмлађи, и самим тим има најмању базу корисника и потражњу за послом, као што се види на слици 4.1.



Слика 4.1. Број послова на популарним сајтовима

Међутим, због своје једноставности, и прилично лаког учења *Vue* је процентуално најпопуларнији, са највећим растом, што и потврђује слика 4.2. која приказује број „звијездица“ односно лајкова ових оквира на *Git Hub-*у.



Слика 4.2. Број звијезда на Git Hub-у

*AngularJS*  представља фронтенд оквир отвореног типа заснован на Јаваскрипту развијен од стране Гугла. За рад са Ангуларом је потребно познавати *Typescript*-а*.* Пошто је развијен од стране Гугла, има више ресурса и подршку у сваком тренутку, као и велику базу корисника. Међутим није баш флексибилан. Теже је разумјети постојеће пројекте, и пројекти рађени у АнгуларЈС се не могу претворити у Ангулар2. Предност Ангулара је што је прилично добар за развијање и одржавање комплекснијих апликација. [12] Користи се за развијање апликација у реалном времену, као што су апликације за дописивање. Заснован је на *MVC* архитектури, што му смањује флексибилност.

По структури и историји, *Vue* се може сматрати млађим братом Ангулара, што је и истина јер је настао из њега. Највећа разлика је што Ангулар користи стварни *DOM* (*Document Object Model*) док *Vue* користи виртуелни (*Vue* поново рендује само оне објекте који су се промијенили, а не читаву страницу.)Идеалан је за прављење прогресивних веб апликација као и веб апликација са једном страницом. Годишњи раст *Vue* корисника иде и до 135%. Постоји више разлога зашто је тако, а највећи разлози су флексибилност и једноставност учења. Много је једноставнији за учење и за почетнике, и доста је једноставније наставити постојеће пројекте. Тривијалне веб апликације се у *Vue* могу направити за пар сати, док то није случај са Ангуларом. Иако подржава *Typescript*, главни алат који користи је Јаваскрипт. Највећа предност *Vue* оквира је његова флексибилност. Највећа мана *Vue* оквира је његова флексибилност. Почетници могу врло лако да од флексибилног кода заправо направе код који је тежак за разумијети и отклонити грешке (*garbage code).* Сам *Vue* оквир и веб-сајтови развијени у *Vue* заузимају мање меморије него Ангулар. Што се тиче скалабилности и великих апликација, Ангулар има предност, док се за мање и средње апликације препоручује *Vue*. Што се тиче сигурности, Ангулар има предност. Неки од популарних сајтова који користе Ангулар су *Upwork, IBM, Freelencer.com, Deutche Bank, PayPal, Forbes..*

*React* је оквир развијен од стране Фејсбука. До данас држи примат најпопуларнијег оквира. Као и *Vue,* и  *React* користи виртуелни *DOM*. Иако имају доста сличности највећа разлика је начин на који је слој погледа изграђен. *React* користи само *ЈSX* *(Javascript XML* – могућност да се *Html* и *CSS* користе унутар Јаваскрипт кода*).* Са друге стране, *Vue* користи *Html* шаблоне, али се може користити и *ЈSX.* Раздвајање кода у три цјелине (*Html*, *CSS,* Јаваскрипт) омогуће да *Vue* буде приступачнији за читање и разумијевање, и самим тим се смањује тежина учења. Kao што је случај и са Ангуларом, и *React* је препоручено користити код великих и комплексних пројеката, док је код малих и средњих боље користити *Vue. React* има бољу подршку, али се документација сматра лошијом. Главни недостатак *Vue-*а је што иза њега не стоји велика компанија, и што је још увијек релативно нов оквир, тако да су број алата и заједница мали у односу на друга два оквира. Са станишта сигурности, *React* се сматра бољим оквиром. Неки познати веб-сајтови који су направљени помоћу *React*-а су Фејсбук, Инстаграм, Нетфликс, Редит, Твитер, док су познати сајтови направљени помоћу *Vuе* оквира Дрибл, *Adobe,Zoom*, *Vice, Alibaba..*

## Основни дијелови *Vue.js* компоненте

*Vue* компоненте имају наставак .*vue*. Скелет *Vue* компоненте чине три тага *<template>, <script>,* и *<style>.* Као што им и име назначује, у *<template>* таг се смјешта *Html* код односно сама структура веб странице. У овај таг се може смјестити само једно дијете (енгл. *child container*). Тек у овaj дијете таг (које може бити било који *Html* таг) је могуће ставити контејнере који су у односу брат-сестра (енгл. *siblings*).

<template>

<div class="child-div">

<div id="sibling-one"></div>

<div id="sibling-two"></div>

</div>

</template>

Кодни листинг 3.1.1. Примјер исправно дефинисаног template тага

<template>

<div id="sibling-one"></div>

<div id="sibling-two"></div>

</template>

Кодни листинг 3.1.2. Примјер неисправно дефинисаног template тага

У *<style>* таг се смјештaju стилови за ту компоненту. У таг *<script>* се смјешта сва функционалност компоненте. Подијељена је у више својстава (енгл. *properties*) од којих су у овом раду кориштени:

- ***name*** представља име компоненте, потребно приликом регистровања и рутирања компоненте

*-* ***computed*** користи се за различито рачунање података*.* Када се промијени вриједност *computed* варијабле која је везана за неку варијаблу у *<template>* дијелу, *Vue* поново рендерује тај дио.

*-* ***methods*** својство у ком се дефинишу све методе компоненте, и које је касније могуће позвати у различитом дијелу компоненте (било *template* или *script*)

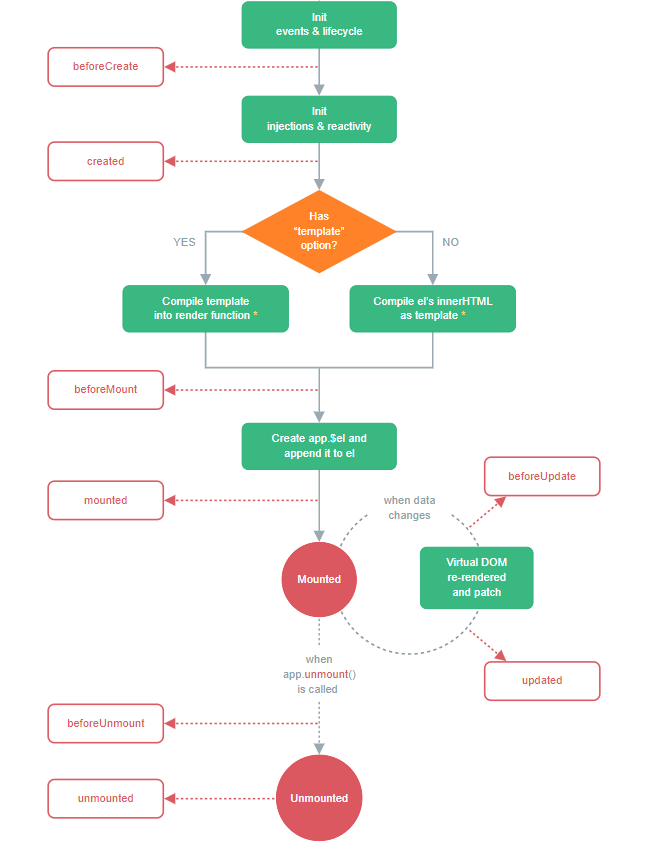
*-* ***data*** својство у ком се дефинишу варијабле које се користе у компоненти

- ***props*** својство преко ког дијетекомпонента добија податке од родитељ компоненте. Приликом слања тих података битно је да се назив *props*-а у дијете компоненти поклапа са називом варијабле која је везана приликом слања у родитељ компоненти. Могуће је послати примитивни тип податка, објекат или функцију.

Животни циклус компоненте је:

* ***beforeCreate*** позива се синхроно одмах након иницијализације инстанце компоненте, а прије учитавања података и подешавања догађаја.
* ***created*** позива се синхроно након креирања компоненте. Подаци су учитани, али компонента још није рендерована
* ***beforeMount*** позива се пред првобитно рендеровање компоненте
* ***mounted*** позива се након што се компонента први пут рендерује
* ***beforeUpdate*** позива се када се неки везани податак промијени, а прије промјене *DOM*-а
* ***updated*** позива се након што је и *DOM* промијењен и поново рендерован
* ***beforeDestroy***позива се прије уништења компоненте. У *Vue3* се назива *beforeUnmount()*
* ***destroyed*** позива се након уништења компоненте. У *Vue3* се назива *unmounted*

Дијаграм животног циклуса компоненте је приказан на слици 3.3.



Слика 3.3. Животни циклус Vue компоненте

## Динамичко везивање података (енгл. *data binding*)

Основа *Vue* оквира је систем динамичког везивања података који чини синхронизацију података и *DOM* изузетно једноставном. Постоји више начина везивања података. Најједноставнији начин је помоћу двоструких витичастих заграда *<span>Message: {{ msg }}</span>* гдје је *msg* дефинисано у својству *data,* у *script* тагу*.* Осим варијабли, могуће је везати и читаве изразе.

Директиви у *Vue* оквиру имају префикс „*v-„. v-bind* се користи за везивање података. Линија *<div v-bind:example=”myData”>* ће за параметар *example* везати вриједност параметра *myData* а не буквално низ карактера *myData.* Наравно, *myData* је потребно прво дефинисати у својству *data,* у *script* тагу*.* Овако је могуће динамички везати и неке особине блока као што су класа и сл. За позивање догађаја користи се *v-оn.* На примјеру, то би изгледало *<div v-оn:click=”myMethod()”>.* Скраћено позивање директива *v-bind* је двотачка , а скраћено позивање директива *v-оn* је симбол „@“ (нпр. *:example, @click*). Још један изузетно битан директив који се користи у *Vue* оквиру је *v-model. v-model* представља двосмјерно везивање података за разлику од *v-bind* које је једносмјерно. *v-model* се може сматрати као комбинација *v-bind* и *v-оn.*

*<input v-model="something">*

је исто што и

*<input*

*v-bind:value="something"*

*v-on:input="something = $event.target.value"*

*>*

*v-model* се користи приликом уноса неке вриједности кроз форму.

*Vue* оквир подржава условно рендеровање и рендеровање на основи неке листе или низа.

За условно рендеровање се користи директив *v-if.* Блок са овим директивом ће се приказати само ако је испуњен услов задат у условном директиву. Осим *v-if* директива, за условно рендеровање се користе *v-else-if* и *v-else,* и користе се као и класични *if-else* израз у неком програмском језику.

*<div v-if="type === 'A'">*

*A*

*</div>*

*<div v-else-if="type === 'B'">*

*B*

*</div>*

*<div v-else>*

*Not A/B/C </div>*

За рендеровање листе неких елемената се користи директив *v-for.*

*<ul id="example-1">*

*<li v-for="item in items" :key="item ">*

*{{ item }}*

*</li>*

*</ul>*

У примјеру изнад ће се исписати појединачно сви чланови низа *items. :key* служи као јединствени идентификатор сваког од елемената креираних помоћу овог директива. Овај директив може проћи кроз низ или кроз објекат.

***Vue Router*** представља официјалну библиотеку *Vue* оквира. Ова библиотека помаже успоставити везу између урл веб читача и *Vue* компоненте и омогућава *Vue* да изузетно лако креира апликације са једном страницом (странице које се не учитавају сваки пут изнова, него се учитавају само промијењени дијелови).

***Vuex***  представља шему управљања стањем (енгл. *state management pattern*) и библиотеку којом се примјењује та шема. Стање (енгл. *state*) су подаци који се чувају у складишту(енгл. *store*). Промјена тих података доводи до ажурирања компоненте. Складиштем је могуће управљати преко акција и мутација (енгл. *actions*  и *mutations*). Акције су асинхроне и њих не занима промјена стања већ логика. Акција ће позвати мутацију. Мутација је синхрона и она је директно одговорна за мјењање стања. [13]

## *Vue* и прогресивне веб апликације

Због својих особина, *Vue* представља идеално рјешење за креирање прогресивних веб апликација. Ове апликације требају да буду лаке, брзе и прилагодљиве. Требају бити поуздане и јефтине за развити. То све омогућава *Vue* који је једноставан оквир малих димензија. Једноставност прављења апликација са једном страницом је управо најбоље својство које *Vue* посједује а тиче се прогресивних веб апликација. Осим ручног додавања манифеста и услужних скрипти, *Vue* посједује додатну компоненту (енгл. *plugin)* за развој прогресивних апликација, међутим ова додатна компонента тренутно ради само у режиму продукције за верзију *Vue3.*

# Практична реализиација веб апликације

Практични дио овог рада је била израда прогресивне веб апликације која кориснику графички приказује очитавања са различитих сензора који се налазе у пластенику. Мјерења могу бити тренутна, или график временског периода које корисник одабере. Такође потребно је било реализовати двије врсте корисника, администратор, који одређује сваком кориснику која мјерења ће пратити, и обични корисник који може само да чита податке Мјерења се преузимају преко различитих *API*-а са *ThingSpeak* платформе. У овом дијелу је детаљно описана апликација као и додатни алати који су кориштени за њену израду.

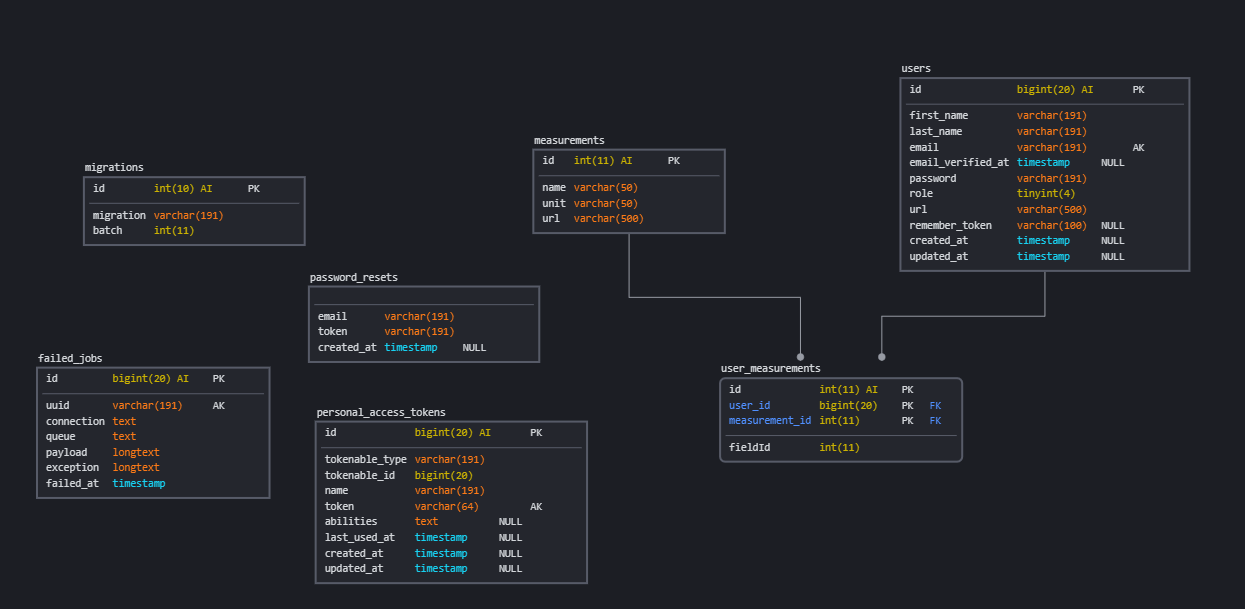
## 5.1. Мотивација за израду апликације

Потреба за аутоматизацијом постоји од кад постоји и сам људски рад, а највише је изражена у данашње вријеме усљед развоја модерних технологија. Од највећих компанија па до појединачних случајева, људи покушавају аутоматизацијом уштедити вријеме и ресурсе и побољшати квалитет услуге којом се баве као и сам квалитет живота. Од најмањих детаља, попут сијалица које се укључују на покрет у стубиштима зграда па до највећих индустријких постројења, немогуће је замислити живот без очитавања неког сензора. Паметни фрижидери, телевизори, видео надзор, регулатори топлоте и сл. су све присутнији и у малим државама попут Босне и Херцеговине. Заједнички назив за све уређаје који очитавања са сензора шаљу преко неке мреже (најчешће интернета), и на основу тих очитавања се дешавају неке акције је Интернет ствари (енгл. *Internet of Things,* скраћено *IoT*). У свијету је то развијено у толикој мјери да постоје читави паметни градови који помоћу Интернета ствари регулишу саобраћај, сигурност, загађеност и сл. Босна и Херцеговина као мала и плодна држава највише шанси и могућности има аутоматизацијом пољопривреде и сточарства. Чињеница да скоро свака кућа у БиХ има своју башту или пластеник је главна мотивација за израду ове апликације. Постављањем једноставних сензора, који су, опет захваљјући интернету доступни по приступачним цијенама, омогућује се праћење основних параметара за узгој биљака а кориштењем ове апликације, корисник не мора увијек бити присутан на лицу мјеста, довољно му је да има интернет конекцију и имаће све потребне информације у реалном времену, као и графички приказ промјене величина у жељеном временском раздобљу. Апликација није строго везана за праћење мјерења у пољопривреди. Уз малу измјену графичких елемената (попут промјене позадине у неку неутралну) може се искористити за праћење било којих мјерених величина.

## Технички опис апликације

Апликација се састоји од два дијела: серверског – који је рађен у Ларавелу и клијентског који је рађен у *Vue.js.* Серверски дио је искориштен за комуникацију са базом.

Систем који је кориштен за реализацију базе је *MySql.* To je вишенитни вишекориснички систем за управљање релационим базама података. Сама структура базе је приказана на слици 4.1. Значајније табеле су *users* – листа регистрованих корисника*, measurements* – каталог свих потенцијалних мјерења*, user\_measurements –* мјерења која корисник може да види. Oстале табеле које нису наведене су аутоматски креиране од стране Ларавеловог система.



Слика 4.1. дијаграм базе података

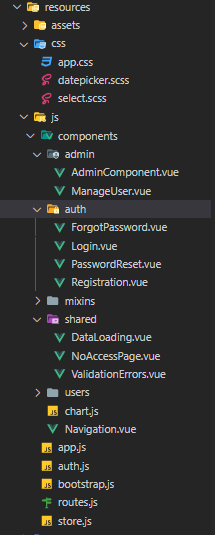
У Ларавелу су креирани контролери који дохваћају корисника и његова мјерења из базе, као и контролер који сачувава у базу мјерне величине додијељене од стране админа.

На слици 4.2. је приказан *welcome.blade.php,* шаблон у ком је повезан *Vue.js* са Ларавелом. Мета тагови показују шаблону како да се понаша у случају различитих уређаја а контејнер *<div id=”app”>* je заправо контејнер у који се увози *Vue.* Таг *navigation* представља *Vue* компоненту која је у овом случају основна, и садржи навигацију до осталих компоненти. Осим стилова и *Vue* оквира, у овом дијелу је и увезен *JSON* манифест описан у поглављу 2.



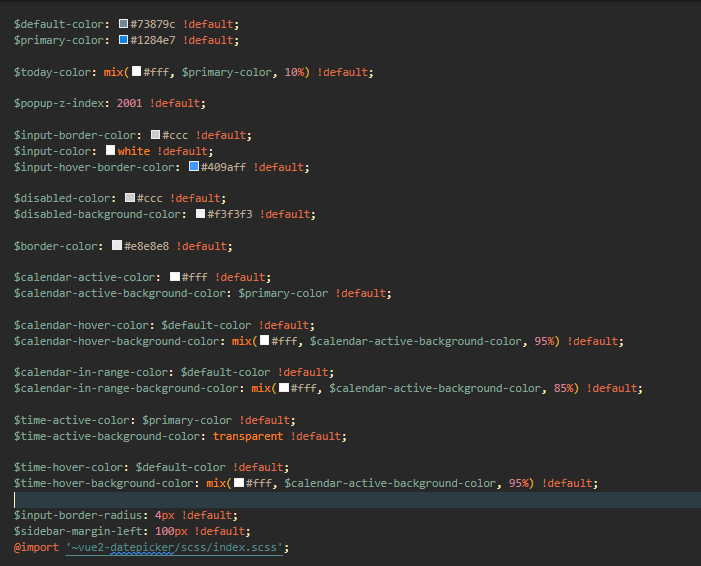
Слика 4.2. welcome.blade.php датотека

На клијентској страни су креиране различите *Vue* компоненте.Такође су предефинисани стилови (слика 4.3.). Основна скрипта је *app.js* у којој су увезени компонента *Navigation,* библиотеке *Vue Router*, *Vuex*  и *Axios*, као и услужна скрипта која ову апликацију чини прогресивном.



Слика 4.3. структура Vue пројекта

Осим генеричког *.css* фајла, за уређивање стилове је кориштен и *.scss* (*Sassy Cascading Style Sheet*) који представља напредни суперсет *CSS-*а. Осим свих могућности које има *CSS, SCSS* има додатне могућности попут дефинисања варијабли, које доста олакшавају и убрзавају посао. Такође, *SCSS* омогућује увоз и предефинисње других *SCSS* датотека, што у многоме олакшава прилагођавање компоненте, нпр. код промјене теме (слика 4.4.).

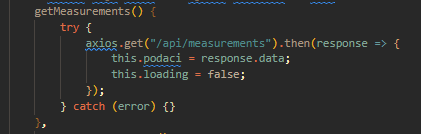


Слика 4.4. прилагођавање стила неке компоненте у SCSS фајлу

*Axios* је Јаваскрипт библиотека заснована на обећањима (енгл. *promise*) и користи се за асинхроно слање *HTTP* захтјева. Oбећање је Јаваскрипт објекат који се користи као чувар мјеста за крајњи резултат асинхроног захтјева. Има три узајамно искључива стања:

* **чекање** чека се на крајњи резултат
* **испуњено** захтјев је обрађен и резултат је спреман
* **одбијено** захтјев је обрађен и постоји грешка

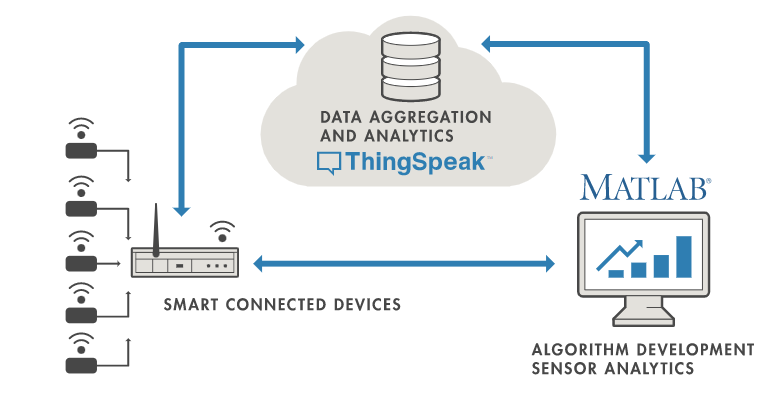
Најчешће коришћени захтјеви код *Axiosa* су *GET, POST, PUT, DELETE.* [14]



Слика 4.5. примјер axios захтјева

Да би се могао користити у *Vue* пројекту, *Axios* је потребно инсталирати, а затим увести у пројекат командом *const axios = require('axios')*. Да би се користио *await/async, Axios* је потребно ставити у  *try catch*  блок.

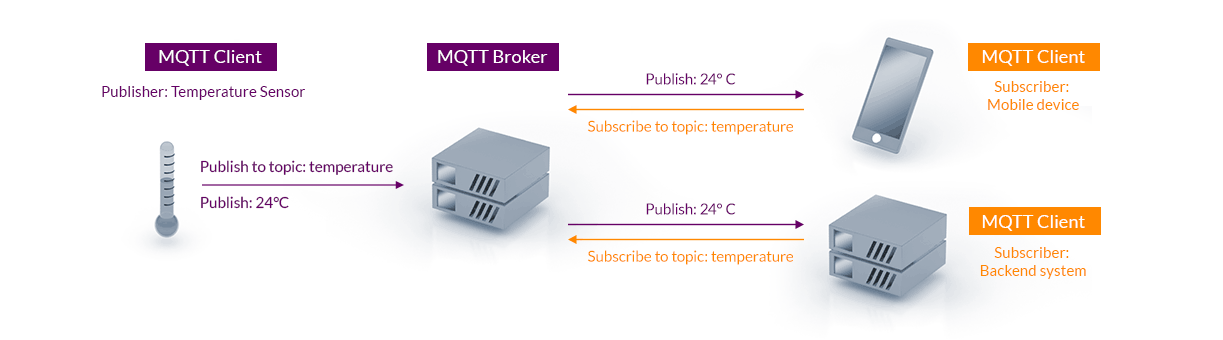
Вриједности мјерења које апликација користи су смјештени на *cloud* платформу *ThingSpeak.* Сама платформа омогућује складиште, анализу и визелизацију података. Такође подржава *MatLab* код.



Слика 4.6. Дијаграм рада ТhingSpeak платформе

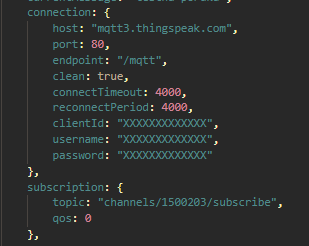
Апликација шаље и прихвата мјерења преко два *API –* а, и то *REST* и *MQTT. REST* је посредник између апликације и платформе. Омогућује размјену података у *JSON, XML* и  *CSV* форматима. У овом пројекту је искориштен за преузимање вриједности мјерења у неком прошлом временском периоду.

За преузимање и ажурирање измјерених вриједности у реалном времену искориштен је  *MQTT*  протокол. *MQTT* представља лагани протокол за слање и преузимање података. Нашао је велику примјену у *IoT* свијету због минималне потрошње пропусног опсега. Принцип рада *MQTT*  протокола је илустрован на слици 4.7.



Слика 4.7. Принцип рада MQTT протокола

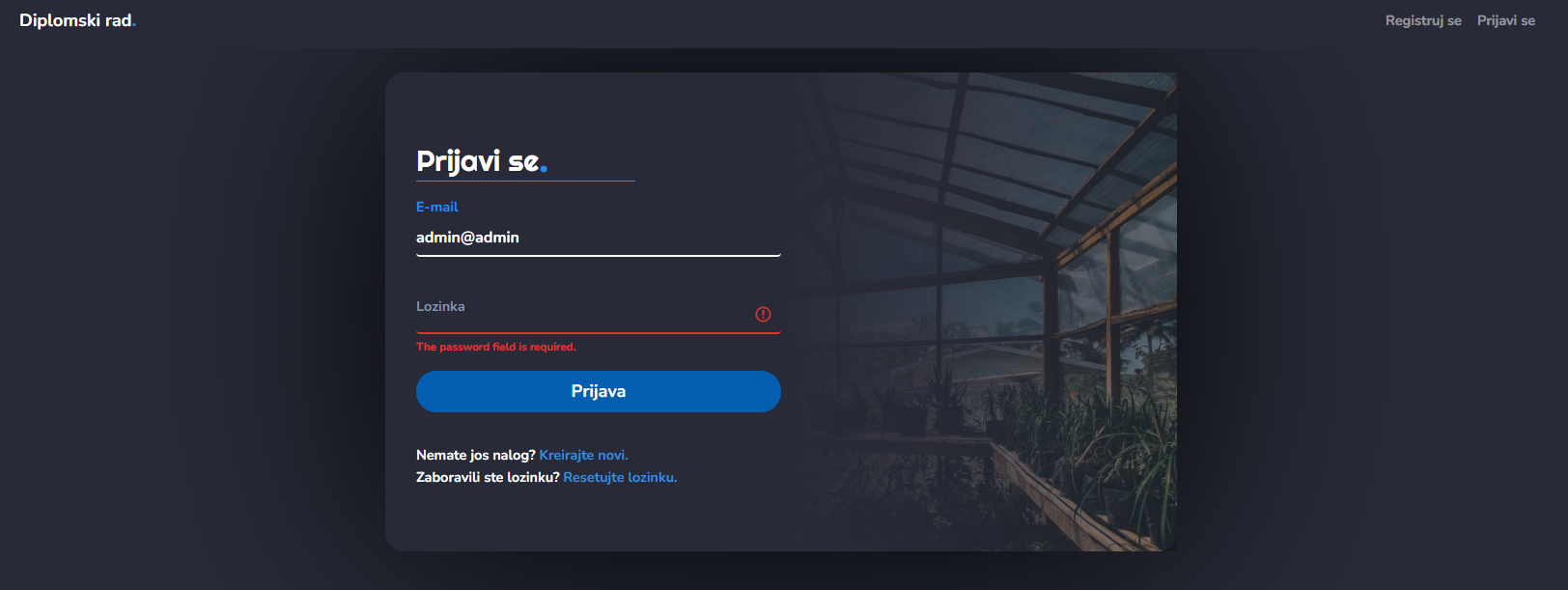
Основу протокола чине брокер и клијент. Брокер представља неки тип сервера који је подијељен на теме (енгл. *topic).* Клијент може слати податке (енгл. *publish*) у жељену тему, или читати из ње. Да би читао поруке из неке теме, потребно је прво да се претплати на ту тему (енгл. *subscribe*). Када добије поруку, брокер ту поруку шаље свим клијентима који су конектовани и који су се претплатити на тему у коју је дошла порука. Овај принцип рада омогућује слање и читање порука у реалном времену. Параметри који су потребни за претплату и читање са *ThingSpeak* платформе су приказани на слици 4.8.



Слика 4.8. Параметри MQTT протокола

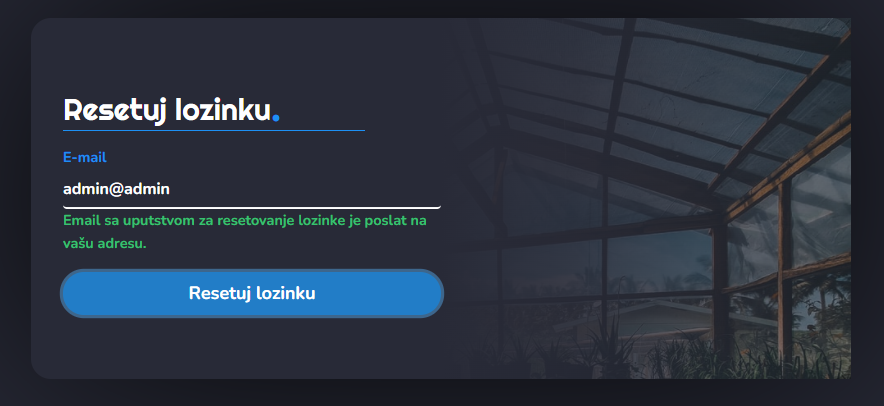
## 5.3 Функционалности апликације

Прва страница коју корисник види када посјети апликацију је страница за пријављивање (слика 4.9).



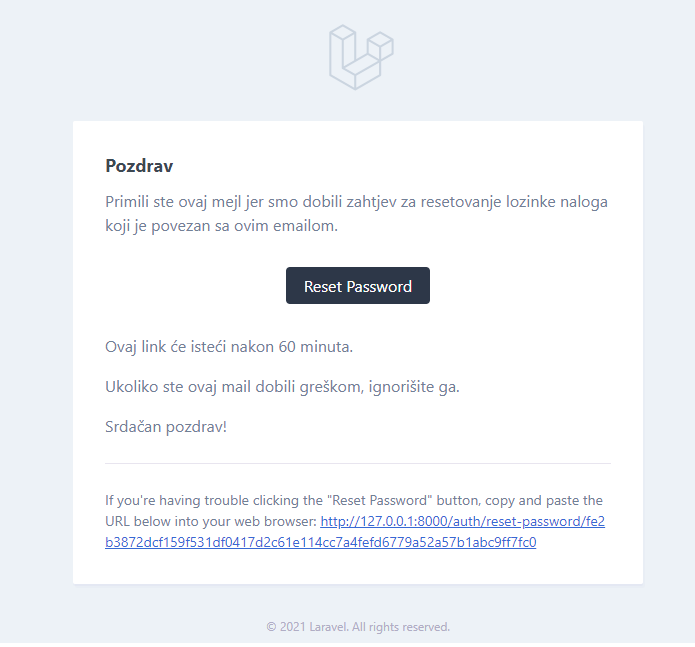
Слика 4.9. Страница за пријављивање

Уколико неки од параметара нису валидни, кориснику ће се приказати одговарајућа порука. Осим форме за пријаву, ту су још и линкови који воде до креирања новог налога, или ресетовања заборављене лозинке преко мејла. Кликом на линк *Ресетујте лозинку* учитава се компонента приказана на слици 4.10.



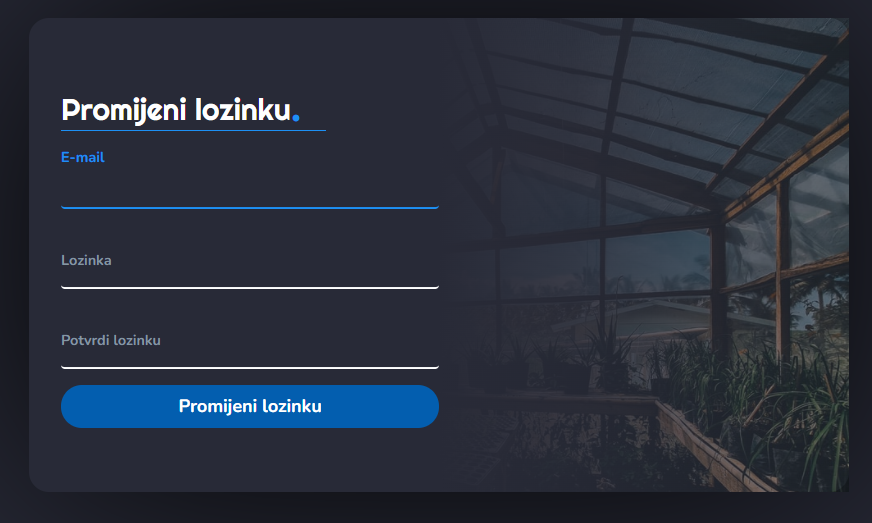
Слика 4.10. Страница за ресетовање лозинке

Потврдом на дугме *Ресетуј лозинку* се шаље мејл адреса са упутством за ресетовање лозинке (слика 4.11.)



Слика 4.11. Мејл за ресетовање лозинке

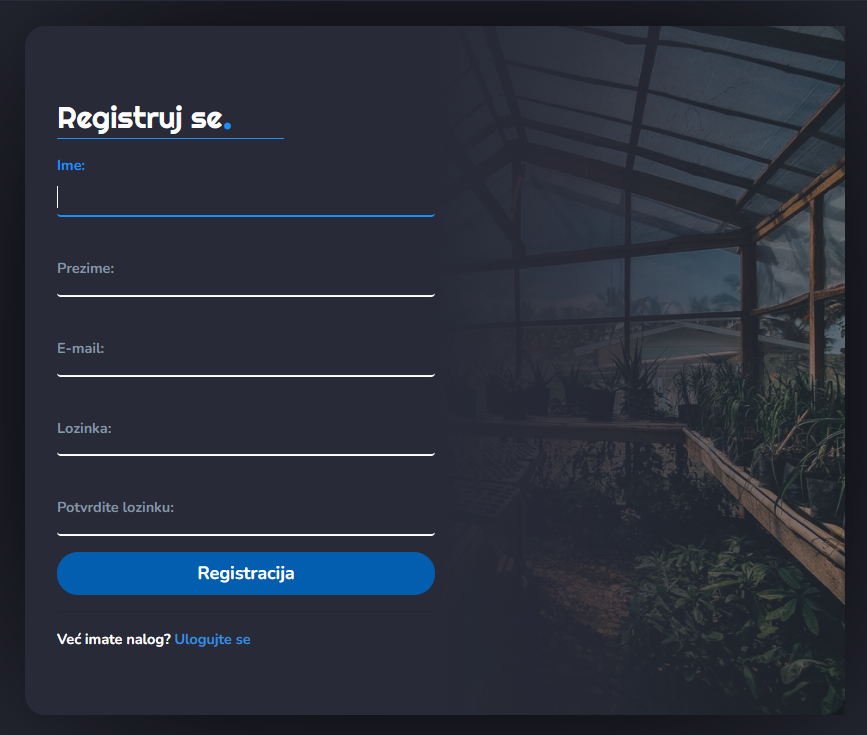
Дугме *Reset Password*  води до нове компоненте приказане на слици 4.12.



Слика 4.12. Завршни корак промјене лозинке

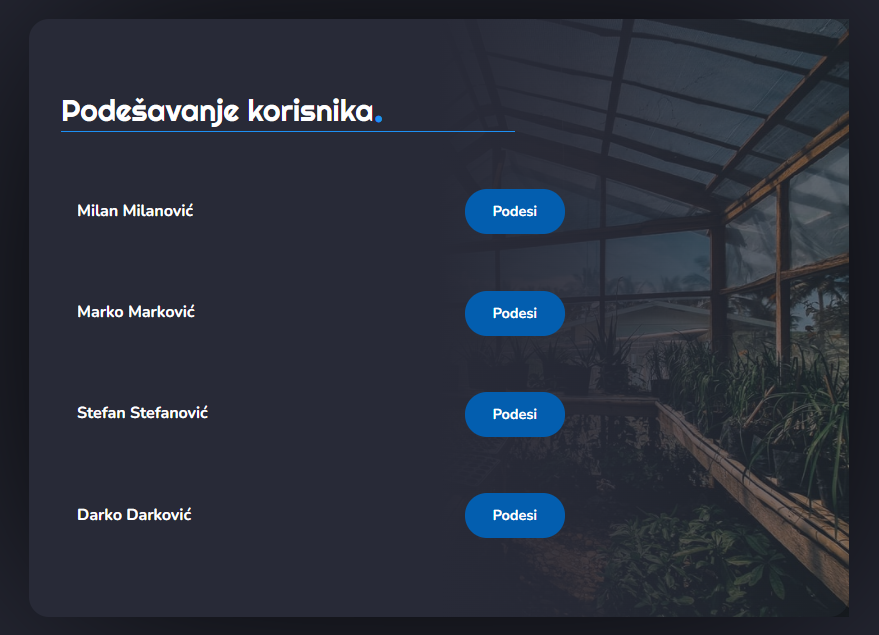
Корисник сада треба поново да унесе своју адресу, нову лозинку и потврду нове лозинке. Уколико унесе неку другу адресу (а не ону на којој је примио мејл ресетовања) појавиће се грешка.

Коначно, компонента за регистровање је приказана на слици 4.13.



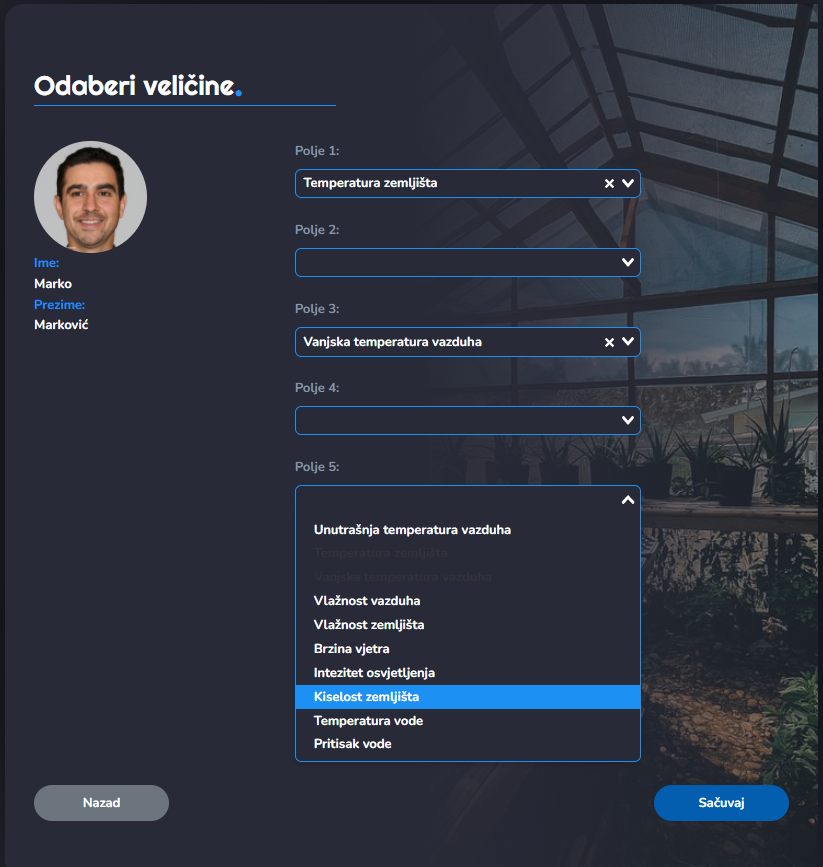
Слика 4.12. Компонента за регистрацију

Када се корисник улогује на свој налог, он види навигациони мени са везама: *Тренутна мјерења*, *Тренд мјерења* као и *Приказ корисника* уколико је админ. Приказ корисника је приказан на слици 4.13. и ту админ види све регистроване кориснике.



Слика 4.13. Приказ корисника

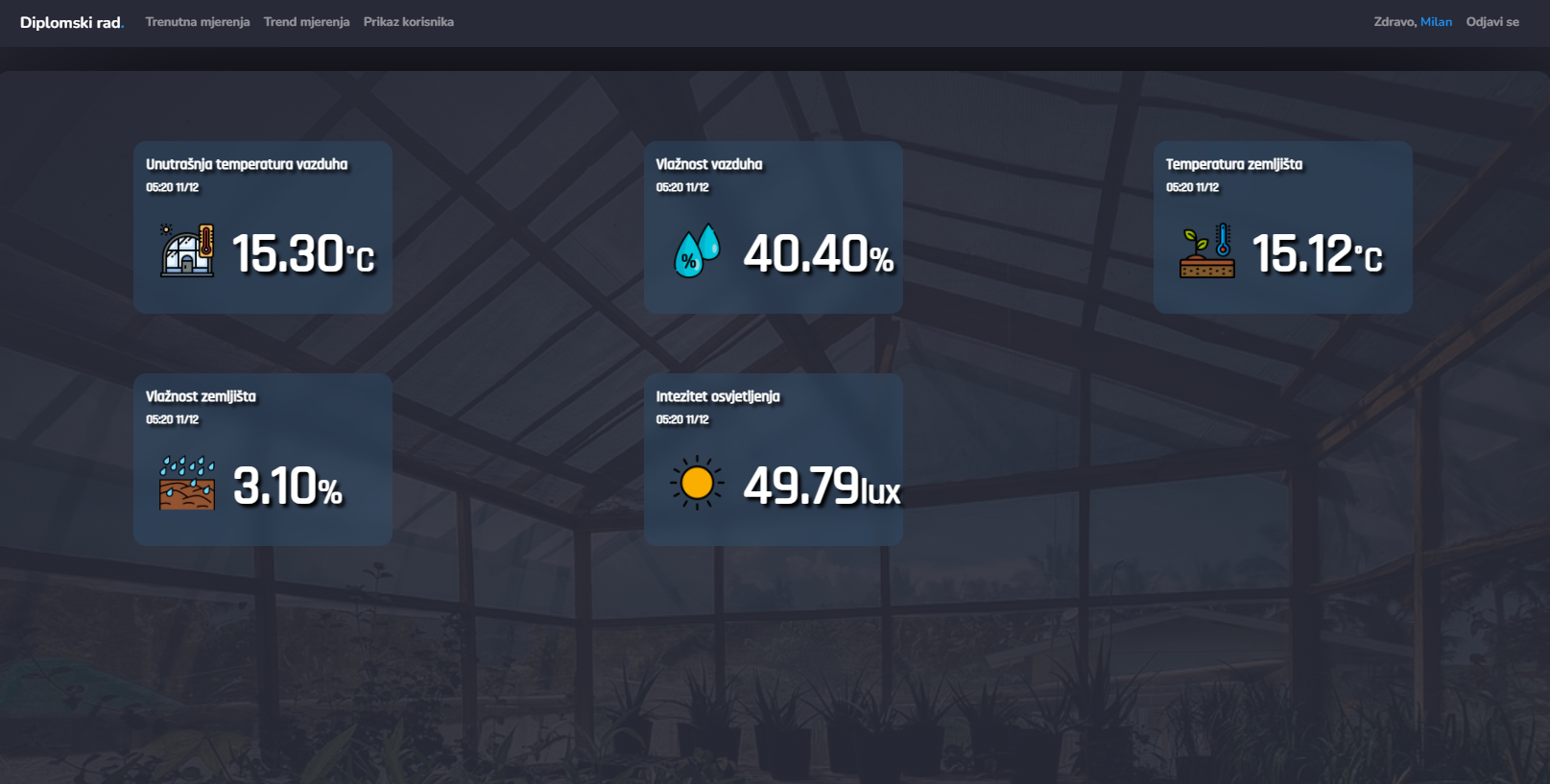
Након одабира неког од корисника, админу се приказује нова компонента приказана на слици 4.14.



Слика 4.14. Подешавање корисника

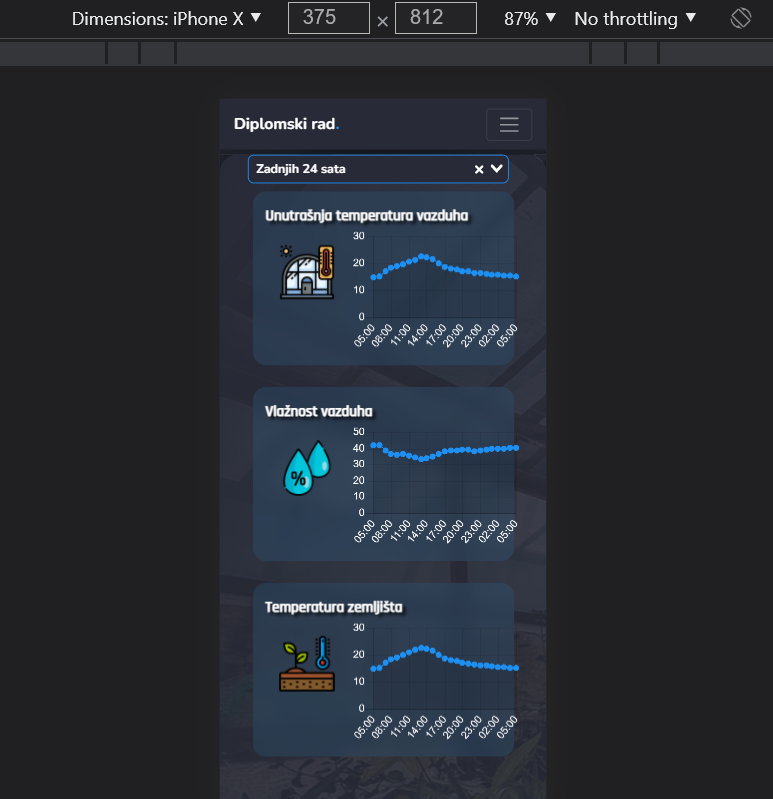
У овој компоненти админ подешава које величине ће корисник пратити и са ког поља ће читати измјерене вриједности. Пошто *ThingSpeak* платформа дозвољава праћење максимално осам величина, админ може додијелити највише осам величина.

*Тренутна мјерења* је компонента која кориснику приказује вриједности измјерених величина у реалном времену (слика 4.15.)



Слика 4.15. Приказ тренутниих величиина

Слика 4.16. приказује компоненту Тренд мјерења. Кориснику се приказује график промјене у изабраном временском периоду а понуђене опције су: задњих 24 сата, задњих 7 дана, задњих 30 дана и произвољан временски период. Одабиром последње опције из падајућег менија, појављује се календар (енгл. date-picker) који кориснику омогућује да изабере који год период жели. Уједно, слика 4.16. приказује како би апликација изгледала на мобилном телефону.



Слика 4.16. Приказ тренда величиина

# Закључак

Апликација реализована у практичном дијелу је изузетно лагана апликација која кориснику пружа увид у све неопходне информације о његовој башти. Развијена је помоћу нових, изузетно популарних алата. Алати су једноставни за научити, а сва неопходна документација је изузетно богата и дескриптивно и примјерима.

Ларавел је оквир који *PHP* држи и даље у животу, а сам оквир је драстично популарнији од свих осталих *PHP* оквира. *Vue* тек треба да ступи на сцену као најпопуларнији Јаваскрипт оквир, а како вријеме пролази и расте његова заједница, тако ће се и његови недостаци наведени у раду отклањати.

Апликација није уско везана само за пољопривреду, може се примјенити на било коју сферу живота, односно на праћење било којих мјерења што је чини изузетно флексибилном, као и оквири у којима је развијена.

Могућа проширења која нису била дио овог задатка су обавјештења која ће се кориснику појавити на телефону уколико неко мјерење пређе критичну границу и подешавање те критичне границе. Такође, неко од могућих проширења, које не зависи само од ове апликације, него и од сензора и реализације сензорске мреже је подузимање акције уколико дође до критичне границе. На примјер, да корисник може помоћу апликације да покрене систем за наводњавање, вентилацију и слично.

# Литература

[1] <https://www.popwebdesign.net/popart_blog/2019/01/kako-da-kreirate-progresivne-veb-aplikacije-i-zasto/> Брзина и потребна меморија изворне и прогресивне веб апликације -децембар 2021.

[2] <https://web.dev/what-are-pwas/> Статистички подаци компанија које су прешле на прогресивне веб апликације – децембар 2021.

[3] <https://developers.google.com/web/ilt/pwa/introduction-to-service-worker> - децембар 2021.

[4] <https://www.popwebdesign.net/popart_blog/2019/01/kako-da-kreirate-progresivne-veb-aplikacije-i-zasto/> Услови Гугла да неки сајт буде сматран прогресивном веб апликацијом – децембар 2021.

[5] <https://en.wikipedia.org/wiki/Progressive_web_application> Подаци о подршци Мозиле прогресивним веб апликацијама – децембар 2021.

[6] <https://en.wikipedia.org/wiki/Laravel#cite_note-maxoffsky-1> Историја Ларавела – децембар 2021.

[7] <https://asperbrothers.com/blog/laravel-vs-symfony/> - децембар 2021.

[8] <https://www.orangemantra.com/laravel-vs-symfony/> - децембар 2021.

[9] https://www.bacancytechnology.com/blog/laravel-vs-codeigniter - децембар 2021.

[10] <https://www.educba.com/laravel-vs-cakephp/> - децембар 2021.

[11] <https://laravel.com/docs/8.x> - децембар 2021.

[12] <https://www.educba.com/angular-js-vs-vue-js/> - децембар 2021.

[13] [https://itnext.io/vuex-made-simple-getting-started-6bf229d432cf - децембар 2021](https://itnext.io/vuex-made-simple-getting-started-6bf229d432cf%20-%20децембар%202021).

[14] <https://stackabuse.com/making-asynchronous-http-requests-in-javascript-with-axios/> - децембар 2021.

# Додаци

## Слике

Слика 1.1. могућности и доступност апликација

извор - <https://web.dev/what-are-pwas/> децембар 2021.

Слика 1.2. Извјештај *Lighthouse* алата

Слика 2.1. Процентуалан број Гугл претрага *PHP* оквирау задњих пет година, извор *Google Trends* децембар 2021.

Слика 3.1. Број послова на популарним сајтовима

извор - <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/> децембар 2021.

Слика 3.2.Број звијезда на *Git Hub-*у

извор - <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/> децембар 2021.

Слика 3.3. Животни циклус *Vue* компоненте

извор <https://v3.vuejs.org/guide/instance.html#lifecycle-hooks> децембар 2021.

Слика 4.1. дијаграм базе података

Слика 4.2. *welcome.blade.php* датотека

Слика 4.3. структура *Vue* пројекта

Слика 4.4. прилагођавање стила неке компоненте у *SCSS* фајлу

Слика 4.5. примјер *axios* захтјева

Слика 4.6. Дијаграм рада *ТhingSpeak* платформе

извор - [https://thingspeak.com/pages/learn\_more децембар 2021](https://thingspeak.com/pages/learn_more%20децембар%202021).

Слика 4.7. Принцип рада MQTT протокола,

извор - <https://mqtt.org> децембар 2021.

Слика 4.8. Параметри MQTT протокола

Слика 4.9. Страница за пријављивање

Слика 4.10. Страница за ресетовање лозинке

Слика 4.11. Мејл за ресетовање лозинке

Слика 4.12. Завршни корак промјене лозинке

Слика 4.12. Компонента за регистрацију

Слика 4.13. Приказ корисника

Слика 4.14. Подешавање корисника

Слика 4.15. Приказ тренутниих величиина

Слика 4.16. Приказ тренда величиина

## Табеле

Табела 1. Структура Ларавел пројекта, извор:

<https://www.w3schools.in/laravel-tutorial/application-directory-structure/> - децембар 2021.

Табела 2. Структура App директоријума, извор:

<https://www.w3schools.in/laravel-tutorial/application-directory-structure/> - децембар 2021.

## Кодни листинзи

Кодни листинг 1.1. примјер *JSON* манифеста

Кодни листинг 1.2. Структура Html документа и примјер тагова

Кодни листинг 1.3. Примјер CSS кода

Кодни листинг 2.1. примјер позивања руте у Ларавелу

Кодни листинг 2.2. повезивање више метода у рути

Кодни листинг 2.3. прослијеђивање параметра рутирању

Кодни листинг 2.4. Повезивање *middleware*-а са рутом

Кодни листинг 2.5. Слање параметара *middleware*-у

Кодни листинг 2.6. Регистровање *view*-a

Кодни листинг 2.7. Примјер кориштења *Html* и *PHP* кода у *Blade* шаблону

Кодни листинг 2.8. *User* модел

Кодни листинг 2.9. Регистровање руте *AuthController*-а

Кодни листинг 2.10. *AuthController*

Кодни листинг 3.1.1. Примјер исправно дефинисаног *template* тага

Кодни листинг 3.1.2. Примјер неисправно дефинисаног *template* тага