**Универзитет у Београду**
**Грађевински факултет**
**Одсек за геодезију и геоинформатику**


# РАЗВОЈ *Rshiny* АПЛИКАЦИЈЕ ЗА ОБЕЛЕЖАВАЊЕ ПРОМЕНА НА ПОЉОПРИВРЕДНИМ ПАРЦЕЛАМА НА ОСНОВУ БИОФИЗИЧКИХ ПАРАМЕТАРА

**дипломски рад - мастер рад**

Кандидат:
Дарко Марковић 1528/18

Ментор:
др Милан Килибарда дипл. инж. геод.

**Београд, 2021**

ГРАЂЕВИНСКИ ФАКУЛТЕТ
КАТЕДРА ЗА ГЕОДЕЗИЈУ И ГЕОИНФОРМАТИКУ
УЖА НАУЧНА ОБЛАСТ: ГЕОДЕТСКА КАРТОГРАФИЈА

МАСТЕР РАД:
Развој Rshiny апликације за обележавање промена на пољопривредним парцелама на основу биофизичких параметара

КАНДИДАТ: Дарко Марковић 1528/18

ЗАДАТАК:
Полазне претпоставке за израду мастер рада су:

1. Веб картографија има релативно кратку, али убрзану историју, са импресивним развојем повезаним како са динамиком науке и технологије, тако и са повећаним интересовањем стручњака и грађана за конзумирање нових врста карата створених услугама Веб картографије.
2. Биофизички параметри базирани на подацима Sentinel 2 мисије могу показати промене на працели узрокована појавом болести, нападом штеточина или елемантарном непогодама.
3. Користећи R и Rshiny могуће је развити систем за обележавање парцела на којима се догодила промена биофизичких параметара. Излаз из апликације треба да буде спреман за даље процесирање користећи технике машинског учења.

На основу изложених претпоставки, у оквиру овог рада предвиђају се следеће активности:

1. Описати R окружење са акцентом на пакете за растерске податке укључујући и просторно временске растерске податке какви су временска серија сателитских снимака или временска серија биофизичких параметара.
2. Описати Rshiny.
3. Описати процес креирања апликације.
4. Након лабелирања снимака и доступног сета података креирати модел базиран на некој од техника машинског учења нпр SVM или RandomForest и презентовати резултате.

У Београду, уторак, 10.11.2020. Предметни наставник

---

---

В.проф. др Милан Килибарда, дипл, инж. геод.

# Садржај

# 1. Увод

Развој информационих технологија променио је, не само начин на који се чувају подаци, већ и начин на који се они интерпретирају и приказују. Један од главних задатака информационих технологија јесте да се успостави одрживи систем података и процеса у циљу доношења паметних одлука у свим областима човековог делања.

За овај рад, од нарочитог значаја су временске серије просторних података о којима ће бити речи у даљем тексту. Просторни подаци, пре рачунарске ере, физички су постојали у форми аналогне карте која је представљала најмоћније средство за визуелизацију просторних феномена. Рапидни развој информационих технологија донео је са собом пуно различитих начина и формата на који подаци могу бити чувани, обрађивани, интерпретирани и на крају визуализовани. Са развојем технологија за аквизицију података, простор за обраду и анализу података је вишеструко увећан, али исто тако, са ширењем простора могућности проширена је и област проблема процесирања, интерпретације и визуализације просторно-темпоралних података.

Бављење темом израде једне овакве веб апликације мотивисано је, пре свега, потребом да се комплексни подаци на што разумљивији начин представе крајњем кориснику и да се представи модерна технологија која омогућава лакше креирање веб апликација помоћу само једног, *R* програмског језика, као и постављање апликације на веб и тестирање њених кључних перформанси помоћу аутоматизованог алата *Lighthouse*.

Тема овог мастер рада, може се рашчланити на два сегмента које ћемо најпре независно посматрати а онда пронаћи и спрегу између њих. Први сегмент јесу временске серије просторних података, начин њиховог прикупљања, специфичности оваквог типа података, њихова расположивост и формати у којима су доступни. Други сегмент на који ће бити обраћена посебна пажња јесте израда веб апликације са посебним осврtом на веб фрејмворк *Shiny* и реактивну архитектуру оваквог решења. Након разматрања ове две целине, увидећемо зашто је *Shiny* адекватна технологија за овакав тип података и које су предности коришћења овакве технологије.

У овом конкретном случају, реч је о временским серијама просторних података. Просторне податке тешко је перципирати на нивоу алфанумеричких података. Комплетнија, јаснија и информативнија слика добија се када се у контекст нумеричких података укључе графички подаци. Визуелизација просторних подака сама по себи је широка и готово неисцрпна област, а када се на то дода темпорална димензија, онда се отвара и нови простор могућности анализа и предикција које је изазовно визуализовати. Графички вид интерпретације података је најинтуитивинији приступ али има и својих мана.

Убрзани напредак веб технологија и развојних алата проширило је круг корисника у многим областима па и у области израде веб картографских апликација. Решење за претходни описани проблем визуелизације временске серије просторних података успешно се решава веб апликацијом као медијумом између корисника и података. Главна предност манипулисања података на овај начин јесте сама интерактивност. Поред интерактивности, апликација нуди увоз података, егзаминацију података, визуелизацију временске серије растера кроз графике, хистограме и најзад, могућност лабелирања података по одређеном критеријуму. Под претпоставком да је растер георефернциран, корисник у посебном панелу може видети где се тај растер налази на мапи света, затим истражити индекс вегетације - *NDVI* (енг. *Normalized difference vegetation index*) за сваки растер датасета и посматрати како се мења у времену на одређеном подручју односно пикселу чије се координате могу ишчитати из

апликације и чија се просторна резолуција може видети у апликацији, а потом извршити лабелирање узимајући у обзир и просечан *NDVI* који апликација срачунава за сваки растер понаособ.

Подаци које ова апликација може интерпретирати јесу растерски подаци који су прикупљени у различитим временским серијама. Главна идеја апликације јесте, да се поред визуелизације података и закључака које је могуће донети на основу фунцкионалности које служе за егзаминацију биофизичких параметара, да се омогући лабелирање које би се могло користити као улазни податак за различите предикционе моделе у машинском учењу.

Продукти сателитских мисија нашироко су заступљени као извор информација о простору у различитим гранама привреде и научним областима. За ову апликацију најинтересантнији је дериват ових продуката, односно вегетациони индекс који је срачунат на основу спектралних карактеристика. У пољопривреди, нормализовани индекс вегетације је веома значајан за мониторинг вегетације кроз време. Као пример показних података за овај рад одабрани су продукти сателитске мисије Sentinel-2 на основу којих је срачунат *NDVI*. Засебно поглавље *Временска серија растерских података* посвећено је прикупљању и препарацији података за апликацију.

Додатни циљ овог рада јесте указати на важност интерпретације временских серија просторних података од стране крајњих корисника и указати на чињеницу да је подједнако важно имати информацију и презентовати је заинтересованој страни, као и да је круцијално бити информисан о модерним технологијама које су нам располагању у постизању тог циља.

У поглављу *Технологије коришћене за израду веб апликације* објашњено је уз помоћ којих технологија је могуће релативно једноставно и брзо направити веб базирану апликацију за геоспацијалне податке и самим тим направити битну окосницу и показати да израда веб апликације не мора бити финансијски скуп и компликован производ. У наведеном поглављу биће речи о програмском језику *R* као полазне тачке за разумевање свих библиотека које су биле коришћене да би се изградила сама логика апликације.

Кључна библиотека за равој веб апликације је свакако *Shiny* модул који представља популарни веб фрејмворк намењен широком кругу *R* корисника, а не искључиво професионалним веб програмерима као што је то случај са већином осталих веб фрејмворка. Посебан део текста посвећен је арихтектури апликације и реактивности система који је суштински концепт *Shiny*-а.

У поглављу *Израда веб картограгфске апликације за обележавање промена на пољопривредним парцелама на основу биофизчких параметара* представљен је сам графички интерфејс апликације, принцип функционисања и посебно издвоједни искоментарисани делови кода главних функционалности уз одређени степен генерализације зарад читљивости кода и његовог суштинског разумевања.

Најзад, у поглављу *Постављање веб апликације на интернет* уверићемо се у највећи бенефит који нуди клауд решење и показати да постављање веб апликације на интернет никада није било лакше.

# 2. Временска серија растерских података

С обзиром да се тема овог рада односи на временску анализу растерских података и њихово лабелирање, потребно је објаснити о каквим подацима је заправо реч. Просторно-временски подаци представљају спој две врсте података. Појам просторних података, односи се на све оне податке који садрже информације о конкретним локацијама на Земљиној површи. Временски подаци су подаци који се односе на неко стање у одређеном временском тренутку. Оваква врста података најчешће се прикупља у циљу анализе и мониторинга климатских феномена и промена, фенолошких појава, саобраћајних услова, демографских трендова, итд.

У циљу што бољег разумевања концепта просторно-временских података, у даљем тексту биће објашњени просторни и временски подаци - са јаснијим разумевањем просторних и временских података, лакше је схватити и њихову каузалност.

## 2.1 Просторни подаци

Просторни подаци могу се представити у различитим форматима и они садрже много више него просту информацију о локацији одређене просторне појаве. Кључна подела просторних података јесте на векторске и растерске податке.

**Векторски подаци** веома прецизно описују одређене просторне појаве које се могу представити на основу три типа векторских података:

- Тачка
- Линија
- Полигон

Иако је направљена оваква подела, суштински тачка је основни векторски податак из кога су изведени остали подаци - линије су састављене од тачака, а полигони од линија. Векторски подаци су одлични начин за генерализацију и дискретизацију просторних ентитета који се налазе на Земљиној површини.

**Растерски подаци** су подаци који се представљају преко грида пиксела где сваки пиксел чува одређену вредност. Вредност која је садржана у пикселу може се односити на боју тог пиксела, одређену јединицу или представљати меру нечега. То је изузетно паметан начин за складиштење информација јер на тај начин пиксели, односно растер, представља веома моћног преносника најразличитијих информација. Најчешћи тип растерских података са којима се већина корисника сусреће јесу сателитски снимци.

Формати за чување просторних података не садрже само информације о локацијама, такви подаци не би били од нарочитог значаја јер не бисмо знали шта је то што је представљено у простору. За локације се често везује сет непросторних атрибута који чувају текстуалне, нумеричке и логичке вредности који су битни за контекст разумевања просторне појаве.

Појам локације није једноставан и локација се може описати на разне начине. Из тог разлога, користи се географски координатни систем помоћу кога се на једнозначан начин дефинишу локације на Земљиној површи. Овај систем не садржи *x* и *y* осу, као што је случај у математичком координатном систему, већ се састоји од латитуде и лонгитуде. Латитуда и лонгитуда се изражавају у степенима и заправо представљају географску ширину и дужину.

Није стран случај да просторни подаци постоје, али ван географског координатног система односно ван пројекције. Просторни подаци могу постојати чак и када не знамо њихово место и оријентацију на глобусу. Међутим, ти подаци се могу георефернцирати, односно такав се растер може сместити у реалне координате

координатног система. Чест је случај и да немамо егзактне координате, али имамо одређене дескриптивне податке на основу којих се може на различите начине утврдити локација. Овакав поступак проналажење локације назива се геокодирање.

## 2.2 Временски подаци

Временски или темпорални подаци су подаци који представљају стање у одређеном временском тренутку. Временски подаци се прикупљају најчешће у сврху различитих и често веома сложених анализа. Постоје различите методе за прикупљање временских података. Најпримитивнија метода је свакако мануелно прикупљање временских података. Постоје и аутоматизовани начини као што су опсервација сензорима или временски подаци који су продукт симулационих модела.

Темпорални подаци могу бити чувани у различитим форматима, у базама података као посебан атрибут и у оквиру темпоралних база података. Темпорална база података чува податке које се односе на неку временску инстанцу. Временска инстанца се односи на прошлост садашњост или будућност.

Узгред ћемо поменути класификацију темпоралних база података:

- Уни-темпоралне
- Би-темпоралне
- Три-темпоралне

Три важна појма везујемо за разумевање времена у контексту базе података:

1. Валидно време – представља временски период током ког је чињеница тачна у реалном свету
2. Трансакционо време – представља временски тренутак када се чињеница бележи у бази података
3. Време одлуке – је време када је донета одлука о некој чињеници

**Уни-темпоралне** базе података имају само једну осу која представља време односно интервал валидног времена или системски интервал времена.
**Би-темпорална** база података садржи две осе времена:

1. Валидно време
2. Трансакционо време или време одлуке

**Три-темпоралне** базе података садрже три временске осе:

1. Валидно време
2. Трансакционо време
3. Време одлуке

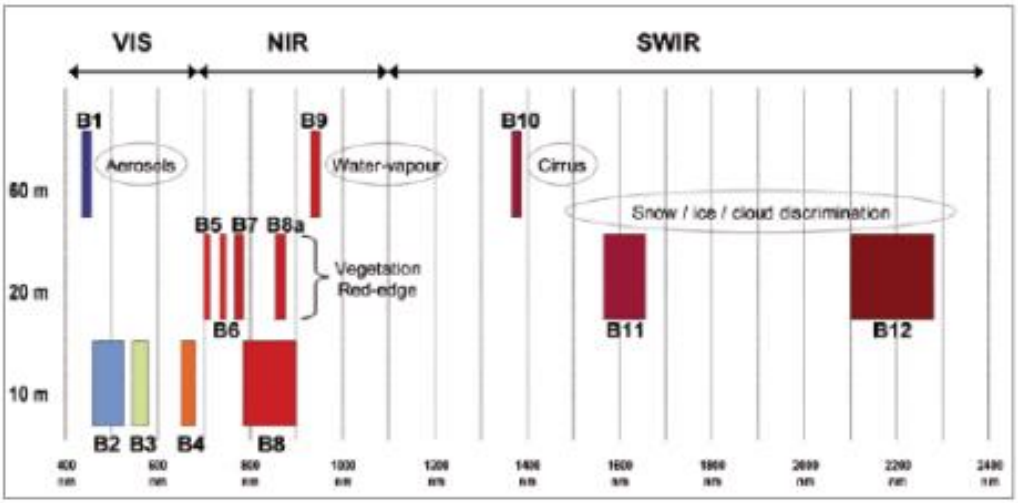
Темпоралне базе података наилазе на све ширу примену у свим гранама привреде као и у областима вештачке интелигенције и машинског учења.

Временске серије растерских података који су коришћени представљају продукте сателитске мисије Sentinel-2 који су преузети са званичне платформе за преузимање података *Sentinel hub.*

## 2.3 Sentinel-2 мисија

"*Copernicus* програм се бави како аквизицијом података који се односе на осматрање Земље, тако и пружањем вредних додатних инфромација кроз Copernicus сервисе намење праћењу земљишта, мора, атмосфере и климатских промена, као и управљању ванредним ситуацијама и цивилној безбедности." – *Philippe Brunet*

Европска Свемирска Агенција развила је *Sentinel-2* мисију као део *Copernicus* програма. Мисија се састоји од 2 сателита у сунцу синхроној орбити на висини од 786 км, са 180° разлике у фази, који на себи носе сензор за прикупљање мулти-спектралних снимака у 13 спектралних канала. Најдужи период од поновног осматрања неке тачке под истим углом је 5 дана. Покривене су све копнене површине између 56° S и 84° N, као и приобалне воде и цело Медитеранско море. Ширина захвата тј. видног поља сателита је 290 km, снимци су доступни у 3 резолуције: 10, 20 и 60 m, зависно од спектралног канала (слика .). Радиометријска резолуција снимака је 12 бит-а.




*Слика 1: Геометријска и спектрална резолуција канала Sentinel-2 мисије*

У оквиру Sentinel-2 мисије постоје различити нивои обраде података. Разликујемо следеће нивое обраде:

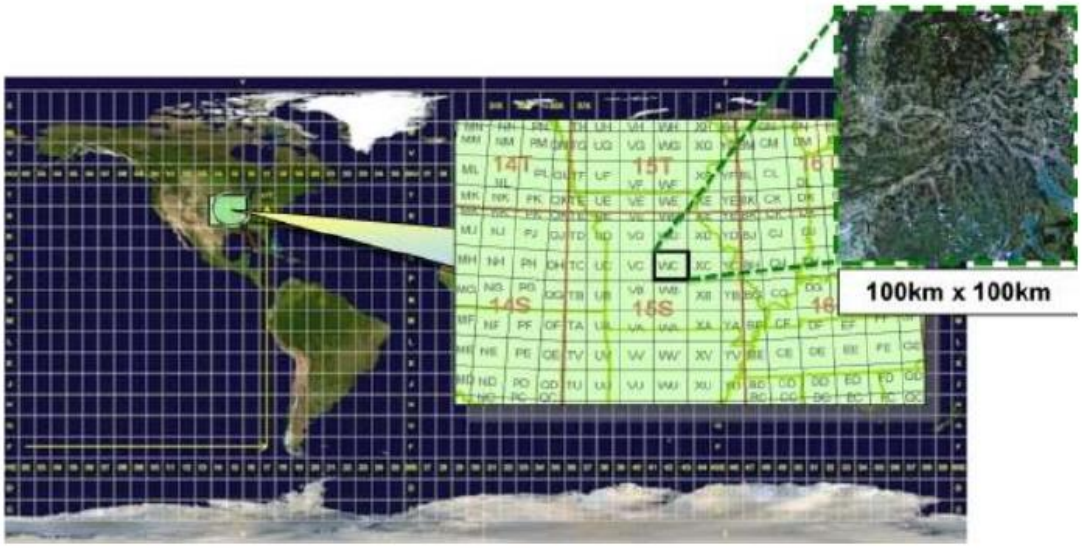
- L0
- L1A
- L1B
- L1C
- L2A

Само поједини нивои доступни су широком кругу корисника:

| Нивои | Опис |
|---|---|
| 1B | На овом нивоу обраде отклоњени су радиометријски и сензорни утицаји и извршена је подела на гранулe на основу релативне позиције у односу на правце сателита. Последица овога јесте та да снимци нису георефернцирани (*слика 2*). |
| 1C | На овом нивоу обраде подаци су георефернцирани и орторектификовани у UTM/WGS84 картографској пројекцији (*слика 3*). |

*Табела 1: Нивои продуката*


*Слика 2: Гранулe L1B снимака*

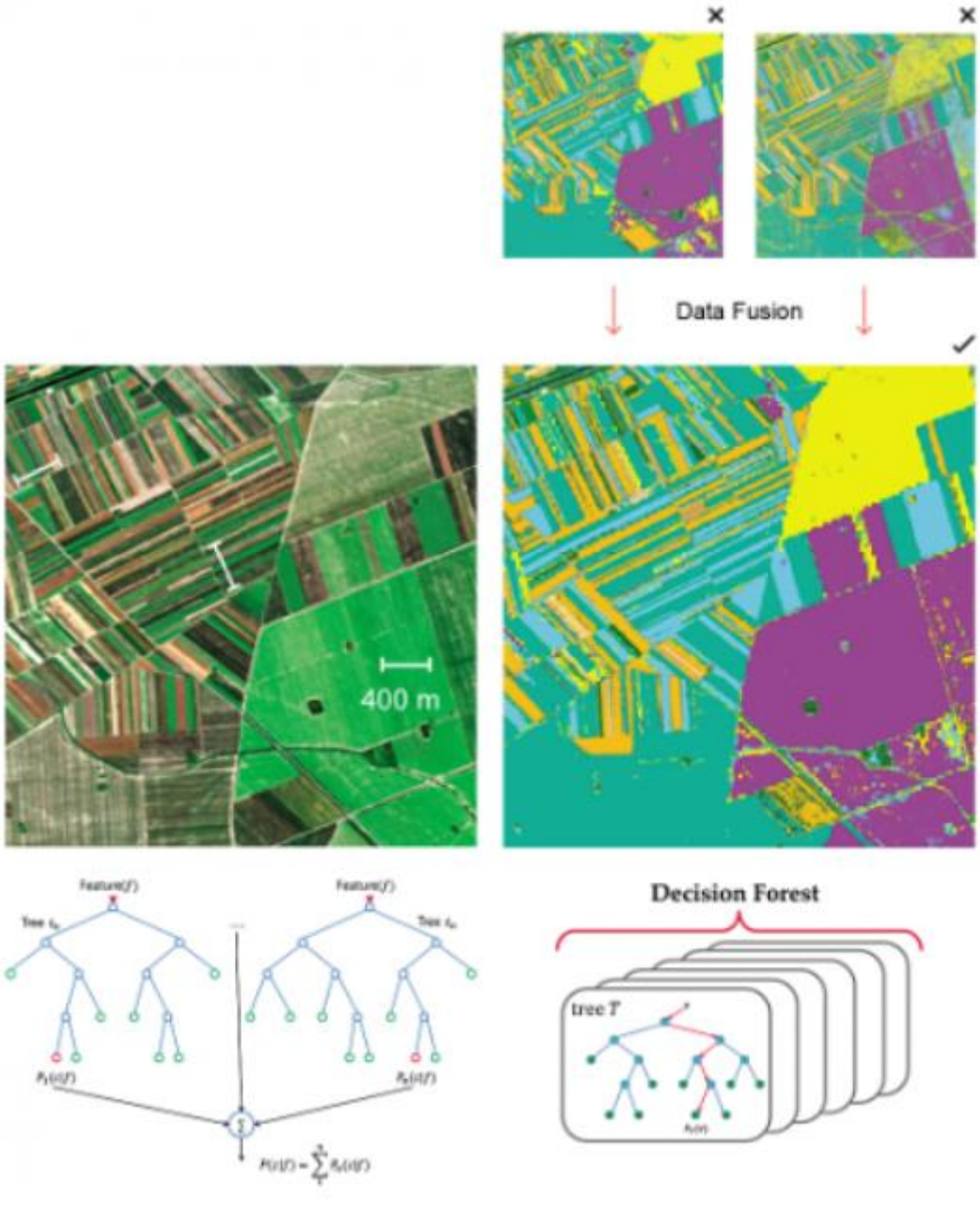



*Слика 3: Георефернцирани и тајловани L1C снимци*

Сви бесплатни снимци доступни су за преузимање на следећем линку: https://scihub.copernicus.eu/dhus/#/home.

### 2.3.1 Продукти

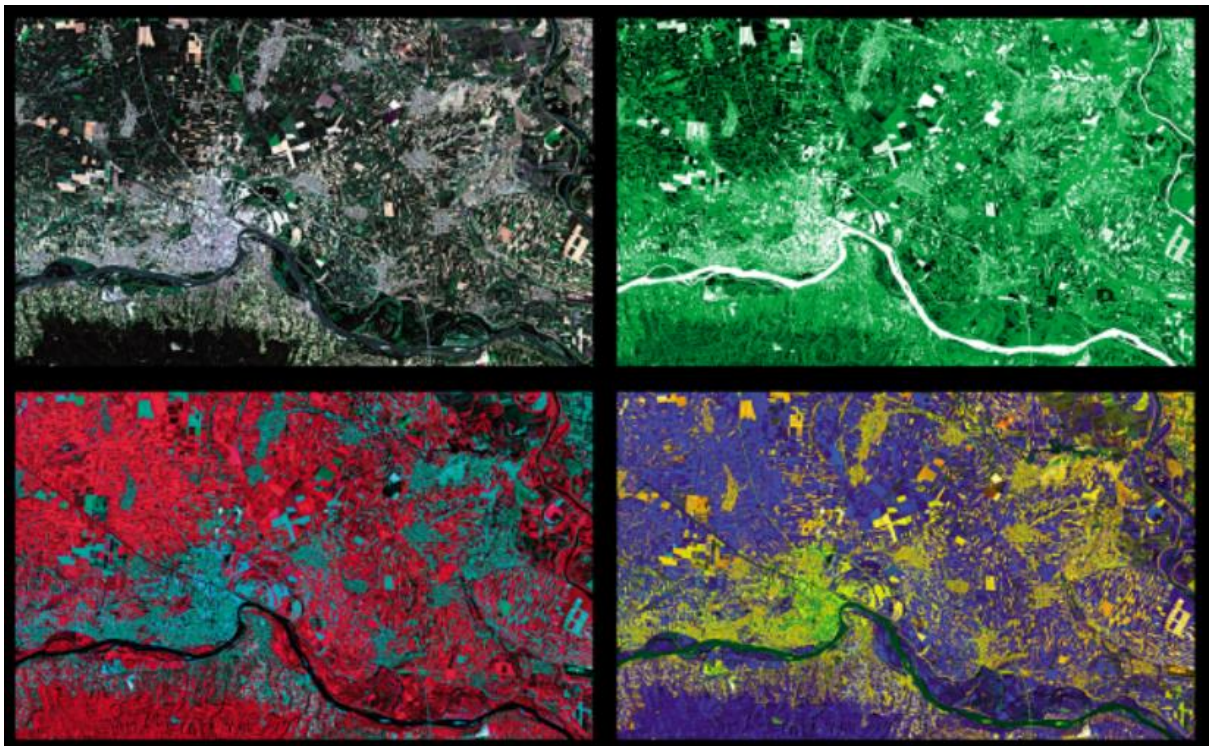
Велика доступност сателитских снимака, како комерцијалних, тако и бесплатних, значајно је утицало на квалитет информација који добијамо из истих. Сада се много прецизније могу детектовати и мапирати како земљишни покривач, тако и вештачки објекти на њему. Врло прецизно могу се детектовати пољопривредне парцеле са јасном границом структуре усева (*слика 4*).




*Слика 4: Алгоритам за фузију података*

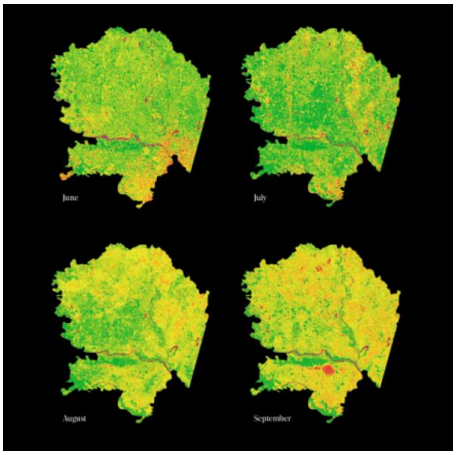
На слици 4. види се мапа класификације усева која је добијена из временске серије мултиспектралних сателитских опажања. На основу алгоритма за фузију података врши се процес фузије мерења која су прикупљена помоћу различитих инструмената. На основу алгоритма за фузију података добија се више информација и већа просторна

резолуција. Комбинацијом различитих спектралних канала сателитског снимка могуће је појачати вегетациони сигнал. Такви продукти називају се вегетациони индекси. Ови продукти представљају улазне податке веб апликације која је предмет овог рада.



*Слика 5: Нови Сад и река Дунав*

Овакви продукти преносе информацију о стању вегетационог покривача на одређеном географском подручју. На слици . у горњем левом углу можемо видети колор композит; горе-десно *NDVI* индекс; доле-лево псеудо-колор слика добијена на основу мерења у блиско-инфрацрвеном, плавом и зеленом каналу; доле-десно слика добијена на основу мерења у краткоталасном-инфрацрвеном, блиско-инфрацрвеном и плавом каналу.




*Слика 6: NDVI за подручје Војводине у периоду лето-јесен*

**NDVI** (*слика 6.*) представља акроним за нормализовану разлику вегетационог индекса. NDVI се креће у опсегу од -1 до 1 и то представља нормализовани вегетациони

индекс. Позитивне вредности указују на вегетационо подручје док са друге стране негативне вредности указују на водене површине. Вредности блиске нули указују на вештачке објекте, урбана подручја, итд. Вредности блиске нули указују на веома густо вегетационо подручје. NDVI се рачуна по следећем изразу.

$$\mathrm{NDVI} = \frac{\mathrm{NIR} - \mathrm{RED}}{\mathrm{NIR} + \mathrm{RED}}$$

## 2.4 Подаци коришћени у апликацији

Подаци коришћени у апликацији представљају сателитске снимке *Sentinel-2* мисије за парцеле које се налазе на подручју аутономне покрајине Војводина у Републици Србији. Поред добијених сателитских снимака, испоручен је *shp* фајл са векторским подацима који представљају границе парцела и сетом атрибута који ближе описују власништво над њима, начин коришћења, број и подброј.

За тест податке апликације, одабрано је седамнаест парцела односно седамнаест временских серија растера. Сваки растер насловљен је датумом опажања. С обзиром на чињеницу да растерски подаци заузимају доста меморије, а у апликацију је потребно учитати читаву временску серију која обухвата и до 500 растера за поједине парцеле, било је потребно урадити предпроцесирање и уклонити све оне канале из растера који се не користе за рачунање *NDVI.* На овај начин, остварена је меморијска уштеда чак и до петнаест пута а брзина учитавања података у апликацију значајно се убрзало док би са оригиналним уносом трајали минутима и самим тим негативно се одразило на корисничко искуство. Други корак предпроцесирања састојао се у опсецању растера. Имајући у виду да веб апликација срачунава просечан *NDVI* за сваки растер у временској серији, било је потребно урадити опсецање растера како би се добило само подручје захваћено парцелом.

Предпроцесирање је извршено уз помоћ *R* програмског језика и *raster, rgdal* и *sf* библиотека.

# 3. Технологије коришћене за израду веб апликације

У овом поглављу биће описане технологије односно библиотеке коришћене за израду веб апликације. За израду веб апликација најчешће се користи микс различитих технологија, од технологија за сервирање просторних података, преко окружења за развој клијентске веб апликације, алата за управљање и инсталацију пакета и контрола развоја апликација до специјализованих библиотека за развој интерактивних карата. Апликација која је тема овог рада развијена у потпуности у R екосистему. R екосистем представља велики број библиотека написаних у R програмском језику.

У наредним поглављима биће описане кључне библиотеке на чијим темељима се базира развој веб апликације која је тема овог рада:

- *Shiny* веб фрејмворк
- *Shinydashboard* библиотека
- *Raster* библиотека
- *Leaflet* библиотека
- *Rgeos* библиотека
- *Tidyverse* библиотека
- *Rsconnect* програмски интерфејс за публиковање *Shiny* апликације

## 3.1 R – програмски језик

*R* није само програмски језик већ и синоним за *R* развојно оркужење чија је примарна сврха статистички прорачуни и графике. *R* је дериват *S* програмског језика који је развијен од стране Џон Чејмберса у склопу *Bell Laboratories-a.* Многи делови *S* језика, односно сам код, остао је у потпуности нетакнут и покреће се у склопу *R*-а. Овај језик је део *GNU* projekta (*General Public License*) што је од изузетне важности за *R* заједницу. *GNU* пројекат је најпознатија и најшире коришћена лиценца за слободан софтвер. Оваква лиценца представља озбиљну гаранцију и обавезује кориснике да сви софтвери који настану као модификација овог софтвера такође морају гарантовати исте слободе, другим речима, све што настане као модификација мора бити дистрибуирано под истом лиценцом.

*R* нуди раскошне статистичке и графичке технике моделирања као што су линеарно и нелинеарно моделирање, класични статистички тестови, анализе временских серија, класификација, кластерисање и многе друге. Велика предност *R*-а јесте велика једноставност вискоквалитетне визуелизације података где корисник има потпуну контролу над графиком. *R* јесте зависан од платформе али развијен је за све водеће платформе као што су *FreeBSD, Linux, Windows* i *MacOS.*

*R* окружење представља интегрисане софтверске алате који омогућавају манипулисање подацима, калкулације и графички приказ. Укључује следеће могућности:

- Ефективно руковање података и складиштење
- Сет оператора намењених за калкулације над листама односно матрицама
- Раскошан сет кохерентних и интегрисаних колекција алата намењених за анализу података

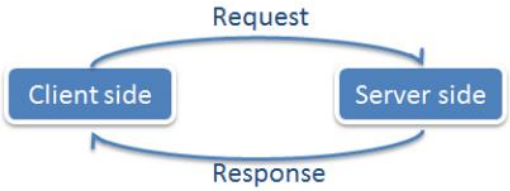
- Алате за графички приказ
- Једноставан и ефективан програмски језик који обухвата тестирање услова, петље, кориснички дефинисане рекрузивне функције, улазне и излазне податке

Лепота *R* окружења огледа се у томе што алати и библиотеке које га чине нису развијане у индивидуалном маниру. Сама реч окружење упућује на пажљиво осмишљен плански развијан кохерентни систем где алати и библиотеке прате стазу развоја *R* окружења чији је фокус на кооперативности међу функционалним целинама. *R* је језик високог нивоа и карактерише га тромост у поређењу са језицима нижег нивоа. Брзина извршавања је знатно спорија у односу на језике ниског нивоа, међутим предности овог језика, као и свих осталих језика високог нивоа, јесте што су релативно лаки за учење, концепти и синтакса су знатно једноставнији. Ова *мана* не представља непремостив проблем за *R* с обзиром да омогућава захтевнијим корисницима могућност писања кода у C, C++ и *Fortran-y*. Напредни корисници могу писати *C* код и директно манипулисати *R* објектима. *R* представља више од статистичког компјутерског језика и тенденција ширења *R*-а ка разнородним областима огледа се у појављивању најразличитијих библиотека. Најпознатије развојно окружење је *R Studio*.

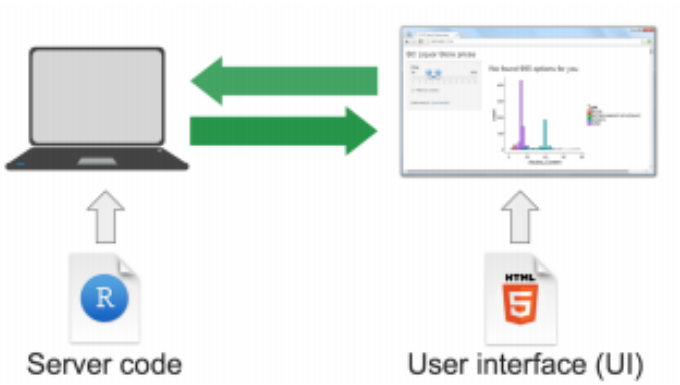
## 3.2 *Shiny* веб фрејмворк

*Shiny* представља веб фрејмворк чија је намена развој реактивних веб апликација. Апликације направљене уз помоћ овог веб фрејмворка су потпуно респонсивне на свим уређајима захвљујући томе што се користи *Bootstrap 3 (*верзије старије 0.11 подржавају *Bootstrap 2). Shiny* реактивна архитектура омогућава релативно лако прављење веб апликација у односу на друге доступне технологије за развој веб апликација. Иако се *Shiny* користи за брзу и лаку имплементацију, никако не треба потценити његове могућности код развијања и значајно сложенијих апликација. У даљем тексту, упознаћемо се са архитектуром овог веб фрејмоврка који омогућава висок степен реактивности, размотрићемо предности и антипатерне којима једна оваква архитектура подлеже.

*R* процес подржава само једну *Shiny* апликацију. Свака *Shiny* базирана апликација користи клијент сервер дистрибуирану архитектуру (*слика* 7), сви ресурси и сервиси послужују се клијенту на основу захтева клијента и одговора сервера. Сервер на коме се покреће *R* скрипта, назива се веб сервер јер он опслужује клијента односно веб претраживач.




*Слика 7: Клијент сервер архитектура*




*Слика 8: Логичка организација R кода*

*R* код састоји се из две логичке целине:

1. Инструкције које се извршавају на серверској страни
2. Инструкције које се односе на кориснички интерфејс

*Shiny* логика је униформна и структура сваке апликације заснива се на следећем шаблону:

```
library(shiny)

ui <- fluidPage()

server <- function(input, output, session) {
        }

shinyApp(ui = ui, server = server)
```

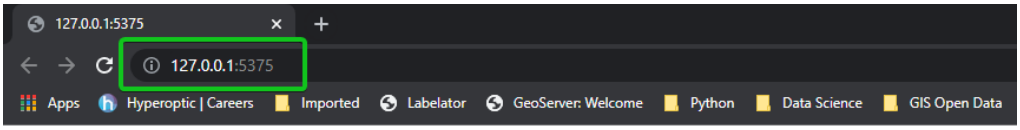
Предуслов за коришћење *Shiny* модула јесте да се претходно учита у сесију како би се омогућио приступ његовим ресурсима. Уколико *R* интерпретер врати грешку да таква библиотека није пронађена, онда је потребно инсталирати ову библиотеку путем следеће команде:

```
install.packages(shiny)
```

Уколико покренемо горенаведени код, истог тренутка покреће се апликација на локалној машини, на насумичном порту уколико порт није предефинисан. *Shiny* апликације подржавају следећи веб претраживачи:

- *Google Chrome*
- *Mozila Firefox*
- *Safari*
- *Internet Explorer*


*Слика 9: Shiny апликација*




*Слика 10: Shiny апликација*

У адресном простору налази се URL (енг. *Uniform Resource Locator*) где можемо видети хоста и порт на коме се слуша апликација (*слика 10*).

### 3.2.1 Реактивна архитектура

Главна идеја реактивног програмирања јесте остварити висок ниво интерактивности апликације кроз креирање реактивних варијабли и реактивних израза који су повезани са другим реактивним варијаблама и изразима. Другим речима, свака промена улазне вредности неке варијабле у апликацији покреће *R* код који рефлектује промену на излазну вредност.

```
input values => R code => output values
```

Сваки реактивни израз у позадини води евиденцију о реактивним објектима и другим реактивним изразима које позива. Уколико нека реактивна вредност није ажурна, реактивни израз ће послати инструкцију свим директно или индиректно

повезаним изразима да се ланчано изврше. Креирање реактивног израза је једноставно, следећи пример демонстрира креирање једноставне апликације која се базира на реактивности.

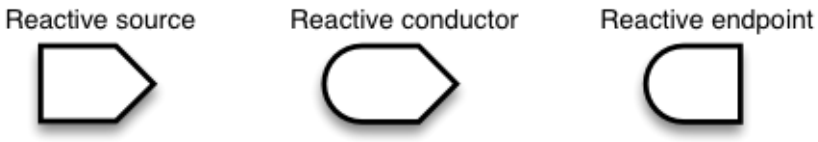
```
library(shiny)

polja <- c("predmet","obavezan","semestar","espb")
ui <- fluidPage(



  titlePanel("Lista predmeta"),
  sidebarLayout(
    sidebarPanel(
      textInput("predmet","Naziv predmeta"),
      selectInput("obavezan","Obaveznost",c("da","ne")),
      textInput("semestar","Redni broj semestra"),
      textInput("espb","Broj ESPB"),
      actionButton("sacuvaj","Dodaj")

    ),
    mainPanel(
      mainPanel(
        DT::dataTableOutput("odgovori", width = 1000), tags$hr()
      )
    )
  )
)

server <- function(input, output,session) {
  sacuvajPodatke <- function(data) {
    data <- as.data.frame(t(data))
    if (exists("odgovori")) {
      odgovori <<- rbind(odgovori, data)
    } else {
```

```
      odgovori <<- data
    }
  }
  ucitajPodatke <- function() {
    if (exists("odgovori")) {
      odgovori
    }
  }
  podaci <- reactive({
    data <- sapply(polja, function(x) input[[x]])
    data
  })
  observeEvent(input$sacuvaj, {
    sacuvajPodatke (podaci())
  })
  output$odgovori <- DT::renderDataTable({
    input$sacuvaj
    ucitajPodatke()
  })
}

shinyApp(ui = ui, server = server)
```

Улазни подаци у овом конкретном примеру су реактивни и њих уноси корисник кроз интерфејс претраживача. Реактивни излазни податак је табела која се ажурира након што корисник унесе информације и притисне дугме за додавање. Реактивни израз заправо трансформише реактивни улазни података у излазни.

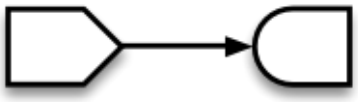
Реактивни модел састоји се из три типа објекта (*слика 11*):

- Реактивни извор
- Реактивни кондуктор
- Реактивна крајња тачка




*Слика 11: Објекти реактивног модела*

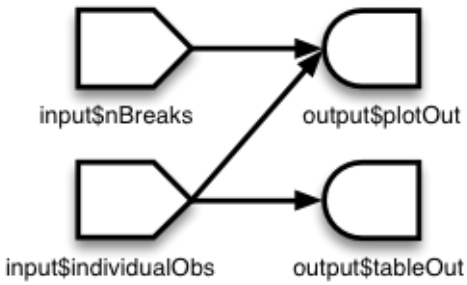
Најједноставнији модел састоји се из само два елемента, реактивног извора и реактивне крајње тачке (*слика 12*):



*Слика 12: Пример једноставног реактивног модела*

У типичној *Shiny* апликацији, реактивни извор представља кориснички унос у веб претраживачу. То може бити унос са тастатуре у виду текста или бројева, као и селектовање или клик на дугме. Свака од ових акција представља иницирање реактивног извора. Реактивна крајња тачка, представља плот, мапу, график или једноставно табелу као у демонстрираном примеру. Реактивни извор доступан је кроз *input* објекат, док се реактивној крајњој тачки приступа кроз *output* објекат у *R* коду. У традиционалним реактивним моделима, да би се креирао интерактивни кориснички интерфејс, потребно је дефинисати *event handler* који би препознавао догађаје на корисничком интерфејсу и читао вредности и вршио трансфер података од једне до друге компоненте. Међутим, хибридни модел *Shiny-a* то решава сам.

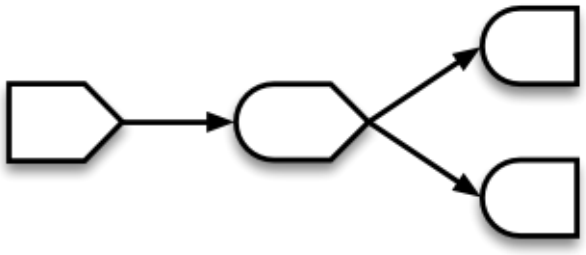
Реактивни извор не мора бити повезан са само једном реактивном крајњом тачком, он може бити повезан са више крајњих тачака које имају и друге реактивни изворе (*слика 13.*). Реактивни извор који је повезан са више реактивних крајњих тачака, приликом промене, покренуће извршавање реактивних израза за све крајње тачке.




*Слика 13: Крајње тачке повезане са више реактивних извора*

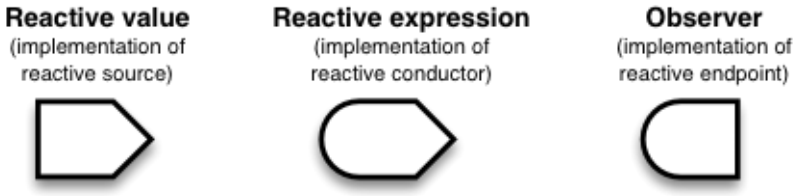
Последњи елемент реактивног модела је реактивни кондуктор. Кондуктор служи да се операције које захтевају велики временски период за изврашавање енкаспулирају између реактивног извора и крајње тачке (*слика 14.*). Другим речима, уколико није

неопходно покретати реактивне експресије за сваку крајњу тачку, онда треба додати кондуктор.



*Слика 14: Реактивни кондуктор*

*Shiny* за сваки елемент реактивног модела поседује класу са објектима који су из ње изведени. Реактивни вредности представљају класу реактивног извора, реактивни изрази заправо представљају реактивне кондукторе док *observers* представљају реактивне крајње тачке који могу да приступају реактивним изворима и реактивним изразима и не враћају никакву вредност већ се користе само за рефлектовање промена.




*Слика 15: Имплементација елемената реактивног модела*

### 3.2.2 Антипатерни реактивне архитектуре

Свака архитектура, поред својих предности, има и слабе тачке на које увек треба обратити посебну пажњу, нарочито када су у питању перформансе и сигурност података. Реактивна архитектура *Shiny-a* омогућава коришћење *observer()* на јако једноставан начин, ослобађајући *R* програмера да експлицитно пише код за сваки догађај покренут на страни корисничког интерфејса. Ово је предност уколико је апликација једноставна односно нема превише реактивних вредности које покрећу код. Међутим уколико апликација временом постане јако комплексна, добићемо претерану реактивност која ће резултирати падом перформанси.

Поред пада перформанси, сам код у *observer()* често нарасте до границе непрегледности па даље развијање, тестирање и одржавање апликације постаје временски комплексније, а сам програм подложнији грешкама. Апликација *Обележавање промена на пољопривредним парцелама на основу биофизичких параметара* временом се усложњавала, и код у *observer()* је вишеструко нарастао резултујући манама које су управо наведене.

Антидот изложеном проблему је управо обрнути приступ. Уместо гомилања реактивних израза у *observer()* било је потребно сваки реактивни догађај посебно хендловати уз помоћ *observeEvent()*. Код апликације било је потребно рефакторисати

тако да се свака реактивна акција у апликацији што грануларније посматра, односно да се свака реактивност експлицитно веже за реактивну крајњу тачку на основу промена реактивних вредности.

Рефакторисани код допринео је бољим перформансама, бољој читљивости, лакшем тестирању и бржем развоју који је потом уследио.

## 3.3 *Shinydashboard* библиотека

*Shinydashboard* библиотека је направљена са циљем да се омогући једноставно креирање *dashboard-a* у *Shiny-u.* У овом поглављу, упознаћемо се са компонентама ове библиотеке, као и са њеним могућностима и ограничењима. Предуслов за коришћење ове библиотеке јесте инсталирана верзија *Shiny-a* која не сме бити старија од верзије 0.11. *Shinydashboard* инсталира се као и свака друга R базирана библиотека:

```
install.packages("shinydashboard")
```

Како бисмо користили ресурсе ове библиотеке потребно је учитати је као и сваку другу R библиотеку. Сваки *Dashboard* састоји се од *header-a, sidebar-a i body-a.*

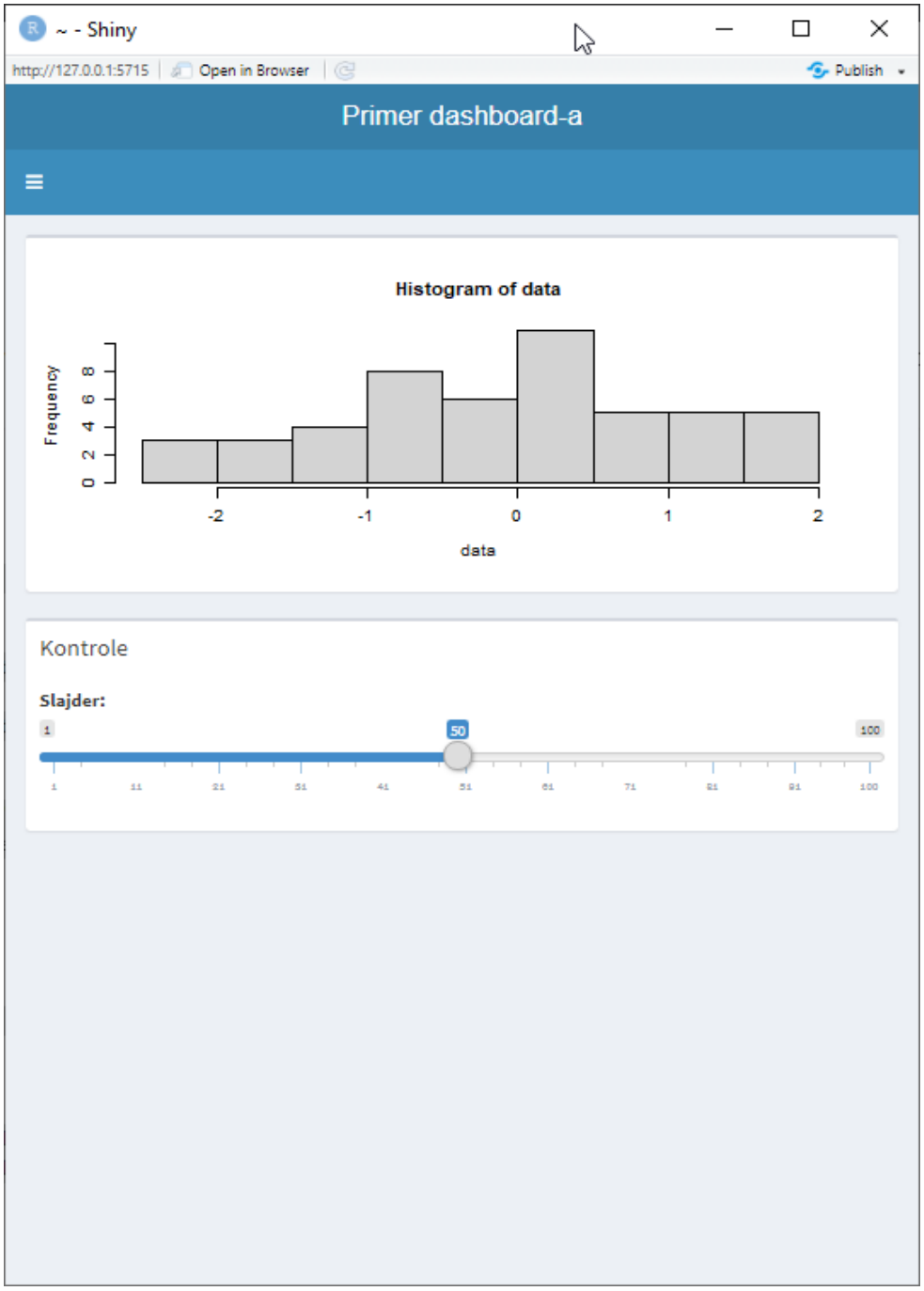
```
library(shinydashboard)


dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)
```

С обзиром да је *Shiny* конципиран да се састоји из две компоненте, корисничког интерфејса и сервера, пратећи ту логику *dashboard* смештамо у објекат корисничког интерфејса:

```
library(shiny)
library(shinydashboard)
ui <- dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)
server <- function(input, output) { }
shinyApp(ui, server)
```

На основу следећег кода креираћемо једноставан *dashboard (слика 5.)* у којем је садржан хистограм и слајдер:

```
library(shinydashboard)
ui <- dashboardPage(
  dashboardHeader(title = "Basic dashboard"),
  dashboardSidebar(),
  dashboardBody(
    fluidRow(
      box(plotOutput("plot1", height = 250)),

      box(
        title = "Controls",
        sliderInput("slider", "Number of observations:", 1, 100, 50)
      )
    )
  )
)
server <- function(input, output) {
  set.seed(122)
  histdata <- rnorm(500)

  output$plot1 <- renderPlot({
    data <- histdata[seq_len(input$slider)]
    hist(data)
  })
}
shinyApp(ui, server)
```




*Слика 16: Пример dashboard-a*

У циљу спознавања могућности и ограничења оваквог креирања веб апликација, требало би се позабавити не само кодом, него и логиком која стоји иза тог кода. Свака веб страница састоји се од барем HTML-а *(Hypertext Markup Language)*, а ту су и CSS *(Cascading Style Sheets)* и Javascript-а. На основу једноставних позива функција, креира се заправо HTML структура. Узећемо за пример често употребљаван HTML таг *div:*

```
div(class = "my-class", "Div content")

## <div class="my-class">Div content</div>
```

Ако упоредимо ове две линије кода, видимо да је за нијасну синтаксно лакше креирати *div* елемент на основу div() позива са одговарајућим параметрима. Међутим, код компликованијих случајева, један позив функције може заменити писање многоструких линија кода. На пример, код креирање панела, много је једноставније позвати функцију и проследити параметре, него писати HTML код и креирати логику за функционисање панела:

```
sidebarPanel(
  div("First div"),
  div("Second div")
)
## <div class="col-sm-4">
##   <form class="well">
##     <div>First div</div>
##     <div>Second div</div>
##   </form>
## </div>
```

Код компликованијих апликација, добра пракса је раздвојити све три неизостване компоненте као посебне објекте, а затих их касније повезати путем dashboardPage() функције:

```
header <- dashboardHeader()
sidebar <- dashboardSidebar()
body <- dashboardBody()
dashboardPage(header, sidebar, body)
```

### 3.3.1 *Header* компонента

Заглавље апликације састоји се из наслова и падајућег менија. Путем следеће функције можемо брзо и лако подесити наслов:

```
dashboardHeader(title = "Primer dashboarda")
```

Прављење падајућих менија никад није било лакше и уз то имамо предефинисане меније за поруке и нотификације и таскове. Пример таквог менија:

```
dropdownMenu(type = "messages",
  messageItem(
    from = "Sales Dept",
```

```
    message = "Sales are steady this month."
  ),
  messageItem(
    from = "New User",
    message = "How do I register?",
    icon = icon("question"),
    time = "13:45"
  ),
  messageItem(
    from = "Support",
    message = "The new server is ready.",
    icon = icon("life-ring"),
    time = "2014-12-01"
  )
)
```

Овакав начин прављења менија је статичан и није погодан за сваки сценарио. У највећем броју случајева, на веб страницама креирани садржај се динамички генрише на основу датих параметара. Другим речима, динамичко креирање садржаја подразумева да се HTML садржај генерише на страни сервера и да се клијенту шаље на рендеровање. Следи пример креирања динамичког менија:

```
# UI
dashboardHeader(dropdownMenuOutput("messageMenu"))
# Server
output$messageMenu <- renderMenu({
  msgs <- apply(messageData, 1, function(row) {
    messageItem(from = row[["from"]], message = row[["message"]])
  })
  dropdownMenu(type="messages", msgs[[1]], msgs[[2]], ...)
  dropdownMenu(type = "messages", .list = msgs)
})
```

Уколико нам заглавље апликације није потребно, могуће га је у потпуности искључити:

```
DASHBOARDHEADER(DISABLE = TRUE)
```

### 3.3.2 *Sidebar* компонента

*Sidebar* компонента може да садржи табове и улазне вредности у форми текста или слајдера (sliderInput и textInput). Обично се користи за брзи и лаку навигацију. Постоји могућност реферисања на екстерне линкове.

```
sidebar <- dashboardSidebar(
  sidebarMenu(
    menuItem("Dashboard", tabName = "dashboard", icon = icon("dashboard")),
    menuItem("Widgets", icon = icon("th"), tabName = "widgets",
             badgeLabel = "new", badgeColor = "green")
  )
)
body <- dashboardBody(
  tabItems(
    tabItem(tabName = "dashboard",
      h2("Dashboard tab content")
    ),

    tabItem(tabName = "widgets",
      h2("Widgets tab content")
    )
  )
)
dashboardPage(
  dashboardHeader(title = "Simple tabs"),
  sidebar,
  body
```

```
)
menuItem("Source code", icon = icon("file-code-o"),
           href = "http://osgl.grf.bg.ac.rs/en/ ")
```

*Sidebar* подржава динамичко креирање путем renderMenu и sidebarMenuOutput-a.

```
ui <- dashboardPage(
  dashboardHeader(title = "Dynamic sidebar"),
  dashboardSidebar(
    sidebarMenuOutput("menu")
  ),
  dashboardBody()
)
server <- function(input, output) {
  output$menu <- renderMenu({
    sidebarMenu(
      menuItem("Menu item", icon = icon("calendar"))
    )
  })
}
shinyApp(ui, server)
```

Као и претходна компонента, ни ова није обавезна те се може, по потреби, искључити: `dashboardSidebar(disable = TRUE)`.

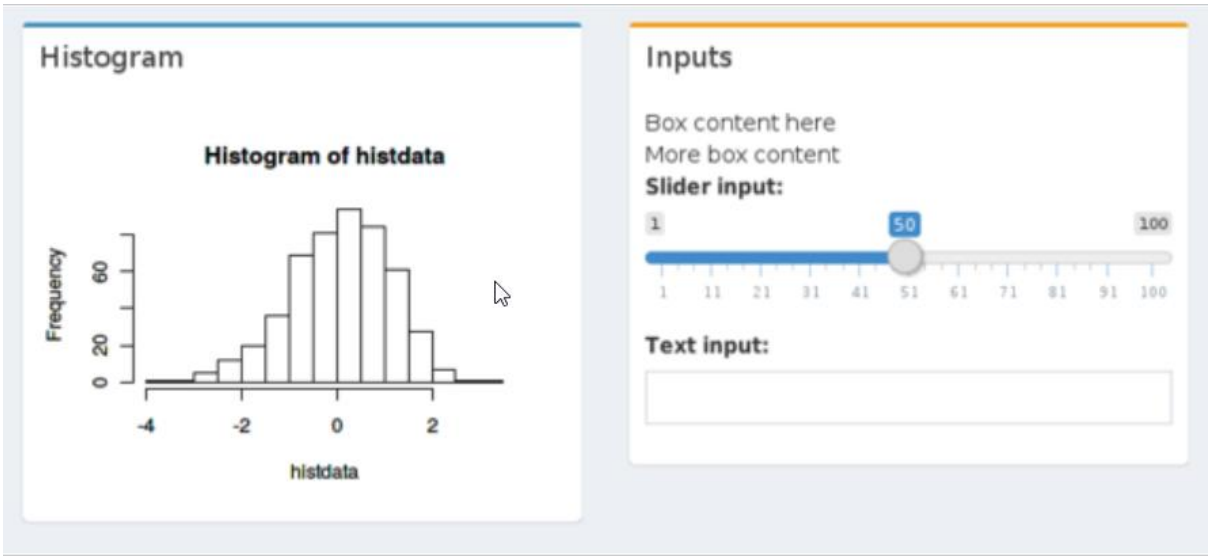
### 3.3.3 Body компонента

Тело апликације може садржати било коју *Shiny* компоненту. С обзиром да је *dashboard* осмишљен као високо структуиран начин грађења апликације, пожељно је користити основни градивни блок односно *box* компоненту. *Box* може садржати било који *Shiny* садржај, велика предност коришћења *box-a* јесте засигурно обезбеђена респонсивност која је одлика свих квалитетно израђених апликација.

#### 3.3.3.1 Box

*Box* представља главни градивни блок *dashboarda* и креира се позивом функције *box()* а садржај може бити било који објекат *Shiny UI*. У типичном *dashboard-u box* се смешта у *fluidRow()*. Пример *box-a* дат је у следећем блоку кода:

```
dashboardBody(
  fluidRow(
    box(plotOutput("plot1")),

    box(
      "Box content here", br(), "More box content",
      sliderInput("slider", "Slider input:", 1, 100, 50),
      textInput("text", "Text input:")
    )
  )
)
```




*Слика 17. Пример креирања box-а*

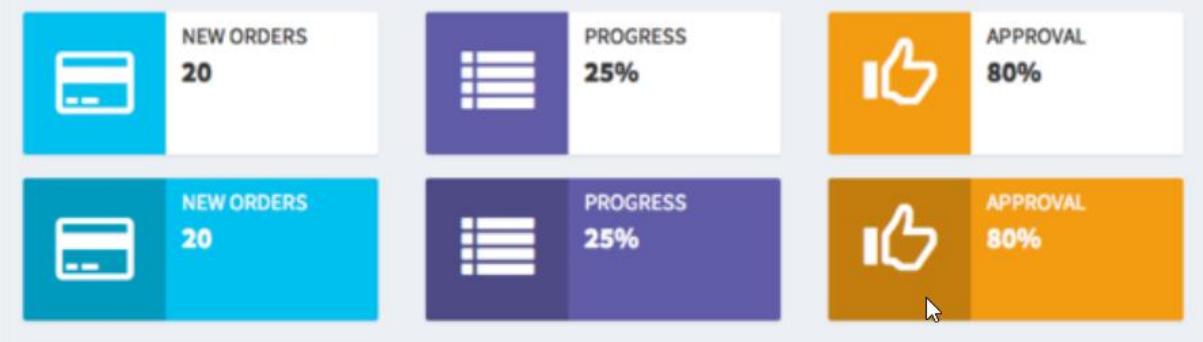
Иако су наслови и боје заглавља *box-а* предефинисани преко статус аргумента функције, могуће је и произвољно подесити стилизацију. У оквиру *box-а* могу се подесити следећи параметри стилизације: боја заглавља и наслова, дугме за миниманизацију *box-а* (*collapse button*) и позадинска боја.

Разликујемо три подврсте односно проширења *box* компоненте:

- tabBox
- infoBox
- valueBox

**tabBox** нуди кориснику могућност поделе *box-a* на више картица. Свакој картици могуће је доделити *id* на основу кога се успоставља контрола на серверској страни. Примера ради, уколико је *id* додељен, у коду који се одвија на серверској страни, могуће му је приступити са долар нотацијом *input$kartica1*. Постоји могућност подешавања висине, ширине и наслова, као и стране на којој ће се картици појављивати у оквиру *box-a* (лево или десно).

**infoBox** превасходно служи за приказивање нумеричких и текстуалних вредности уз додатак иконице (*слика 18.*).




*Слика 18: Пример infoBox-a*

*infoBox* може садржати статички али и динамички садржај. Најчешће се јавља потреба за динамичким садржајем и за те потребе имамо помоћне фунцкије *infoBoxOutput* и *renderInfoBox.*

**valueBox** веома личи на *infoBox,* разликује се у приказу (*слика 19.*).


*Слика 19: Пример valueBox-a*

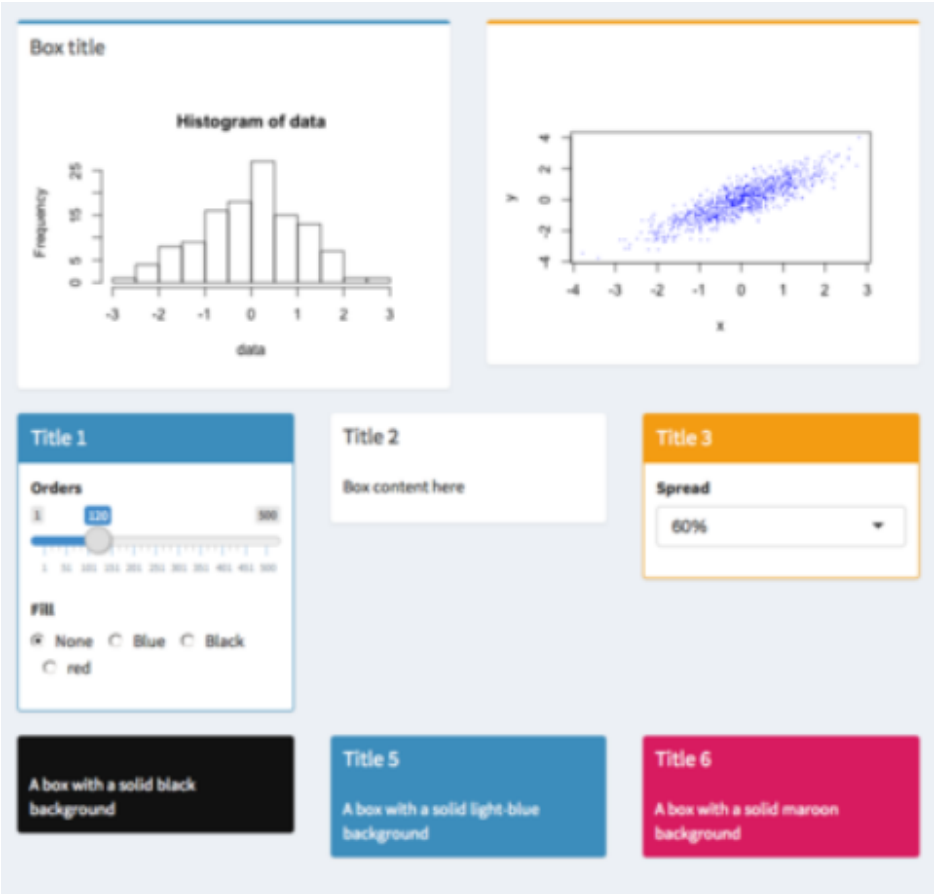
### 3.3.3.2 Распоред

Распоред *Shiny* апликација креираних помоћу *shinydashboard* библиотеке, базиран је на систему *Bootstrap* грида. Овакав систематичан распоред подразумева поделу тела апликације на 12 колона једнаке ширине, док висина зависи од броја редова. Имајући у виду да је *body* подељен на тачно 12 колона, величину *box-a*, односно његову ширину дефинишемо тако што одабирамо колики број колоне ће *box* заузети.

Постоје три начина на које се може контролисати распоред *box-eva* на екрану:

1. Распоред на основу редова
2. Распоред на основу колона
3. Распоред на основу редова и колона

**Распоред на основу редова** овакав приступ подразумева коришћење *fluidRow()* фунцкије. Сваки ред има грид ширине 12, примера ради, ако *box* има *width = 6* то значи да ће заузети пола екрана. Код распоређивања *box-ева* на овај начин, *box-еви* ће бити поравнати на основу њихове горње ивице. Доња ивица *box-ева* не мора нужно бити поравната из разлога што *box-еви* могу бити креирани различитих висина. На следећој слици (*слика 20.)* можемо видети да су сви *box-еви* поравнати по горњој ивици, а у другом реду можемо видети како изгледа поравнање када су висине *box-ева* међусобно различите. Наравно, могуће је усагласити висине да буду идентичне у циљу остваривања истих висина колона. Висина се подешава преко *height* параметара. Висину је потребно подесити у пикселима. Ова разлика произилази из чињенице да HTML/CSS другачије дефинишу висину и ширину.




*Слика 20: Пример распореда на основу редова*

**Распоред на основу колона** заснива се на следећем поступку. Најпре се креира колона, а онда се у њу смештају *box-еви.*

**Распоред на основу редова и колона** овакав приступ представља комбинацију претходна два приступа.

## 3.4 *Raster*

Растер библиотека представља групу функција које служе за креирање, читање, манипулисање и писање растерских података. Поред најразличитијих функција, ова библиотека омогућава развијање сопствених функција кроз надоградњу или модификацију постојећих. Имплементирана је и растерска алгебра, односно математичке операције над растерским типом података.

Велика предност растер пакета јесте што може радити са сетом просторних података који су складиштени на диску, односно који су превелики да се учитају у RAM меморију. То даље имплицира да ова библиотека може радити са веома великим фајловима зато што објекти који се креирају за те фајлове садрже само информације о структури објеката. Под структуром растерског објекта сматра се: број редова и колона, екстензија, име датотеке. Током процесирања са овим објектима подаци се прослеђују у деловима, а уколико излазна датотека није дефинисана, а растер је превелики за RAM меморију, онда ће се створити привремена датотека.

Растер библиотека се темељи на S4 класама.

Најважније класе које се односе на саму структуру података су:

- RasterLayer
  - Ова класа представља растерски податак са једним атрибутом. Такође, садржи и све фундаменталне параметре једног растерског објекта: број редова и колона, просторни обухват података и параметри координатног референтног система.
- RasterBrick
  - Класа RasterBrick подржава више атрибута у оквиру растерских података. Нпр. сателитски снимци са више канала. Важна карактеристика ове класе јесте да се односи на само један фајл.
- RasterStack
  - Ова класа представља листу растерских лејера. Веома слична класи RasterBrick, с тим што не постоји ограничење у погледу фајлова – дакле, фајлови могу бити сачувани на диску или у RAM меморији, а могућа је и комбинација ова два случаја.

## 3.5 *Leaflet*

Leaflet је водећа *open-source* библиотека написана у *JavaScript*. Служи за креирање картографског садржаја за различите платформе. Такође, подржава HTML5 i CSS3 које подржавају паметни телефони. Попут OpenLayers-a, Leaflet је под лиценцом BSD клаузуле 2. Користи се од 2011. године и поред OpenLayers библиотеке и Google мапа, једна је од најчешће коришћених библиотека. Лепеза функција које нуди ова билиотека је сасвим задовољавајућа већини веб програмера и веб картографа. Само библиотека заузим само 39 KB. Велики број веб сајтова користе управо ову библиотеку

која је по свом дизајну једноставна, а која нуди изузетне перформансе и на коју се могу ослонити и програмери који желе направити различите типове веб карата без напредног познавања функционалности ГИС софтвера. Leaflet подржава следеће ГИ формате:

| Стандард | Подршка |
|---|---|
| GeoJSON | Подршка кроз geoJson функцију |
| KML, CSV, WKT, TopoJSON, GPX | Подршка кроз *Leaflet-Omnivore* проширење |
| WMS | Подршка кроз TileLayer.WMS подтип |
| WFS | Није подржано, али постоје *3rd party* проширења |
| GML | Није подржано |

*Табела 2: Leaflet стандарди*

Главни Leaflet објекти:
- Растерски типови (TileLayer и ImageOverlay)
- Векторски типови (Path, Polygon, Circle)
- Групни типови (LayerGroup, FeatureGroup, GeoJSON)
- Контроле (Zoom, Layers…)

## 3.6 *rgeos*

У склопу ове библиотеке имплементиране су функционалности за манипулисање просторним геометријама користећи *Geometry Engine – Open Soruce (GEOS)* библиотеку. Неке од често коришћених *rgeos* функција:

- gContains – проверава да ли се одређена геометрија налази у склопу друге геометрије. Да би се користила ова функција, потребно је координате тачке трансформисати у *SpatialPoint* објекте
- gDistance – ова фунцкија рачуна дистанцу између две геометрије у мерној величини пројекције која се користи
- gConvexHull – креира конвексни полигон
- gIsValid - користи се за проверу валидности геометрије
- gCentroid – користи се за рачунање центроида
- gIntersection – одређује пресек две геометрије и враћа *sp* објекат као резултат
- gLength – рачуна тренутну дужину неке геометрије у пројекцији
- gBoundary – одређује границу одређене геометрије
- gArea – ова функција рачуна површину одређене геометрије у датој пројекцији

## 3.7 *tidyverse*

Представља колекцију библиотека за управљање подацима и неизовастани је алат за све оне који се баве науком о подацима. Колекција библиотека које се налазе под сугестивним именом *tidyverse* може се функционално груписати по следећим логичким целинама:

- Чишћење, структурирање и трансформација података
  - dplyr - подржава филтрирање и груписање података и комбиновање различитих функција уз помоћ пајп функције **%>%**
    - **select() –** селектује колоне датасета

        - **filter() –** филтрира редове по задатом критеријуму
        - **group_by() –** групише на начин да се оригинални датасет не мења
        - **summarise() –** сумира у комбинацији са горепоменутим функцијама
        - **arrange() –** аранжирање колона по растућем или опадујућем критеријуму
        - **join() –** подржава све врсте придруживање табела
        - **mutate() –** подржава креирање нових колона
    - tidyr – нуди додатну групу функција која проширује могућности dplyr-a:
        - **gather() –** конвертује више колона одређеног датасета у кључ-вредност парове
        - **spread() –** као улазни податак узима две колоне и враћа их као више колона
        - **separate() –** подржава рашчлањивање једне колоне у више колона
        - **unit() –** подржава спајање више колона у једну
    - stringr – подржава манипулисање нискама
        - **str_sub() –** екстракција подниске из ниске
        - **str_trim() –** одсецање ниске
        - **str_length() –** провера дужине ниске
        - **str_to_lower() –** конвертовање ниске у мале карактере
        - **str_to_upper() –** конвертовање ниске у велике карактере
    - forcats – ова библиотека намењена је за управљање са факторима односно са категоричким подацима
- Учитавање података и управљање
    - tibble – омогућава конвертовање *dataframe-a* у формат који је погоднији за многе функционалности доступне у оквиру *tidyversa*
    - readr – ова библиотека служи за олакшано читање фајлова и њихово парсирање у горепоменути формат *tibble*
    - readxl – подржава учитавање и управљање ексел форматима
    - haven – подржава учитавање SPSS, STATA и SAS податке
    - googledrive – подржава учитавање гугл драјв фајлова
- Функционално програмирање
    - purr – олакшава функционално програмирање, олакшавајући корисницима овог пакета да избегавају петље осањајући се на функције за мапирање
- Визуализација и егзаминација података
    - ggplo2 – представља најбољу библиотеку за визуелзацију података, широк спектар могућности учинио је ову библиотеку популарном да је последично направљена и пајтон верзија ове библиотеке

## 3.8 *rsconnect*

Представља програмски интерфејс за публиковање *R* продуката за *shinyapps.io, RStudio Connect* и *RPubs.* У *R* продукте спадају *Shiny* апликације, *Rmarkdown* извештаји, плотови и статички веб садржај.

Преглед главних функција за управљање апликацијом:

- deployApp() – публикација апликације на сервер
- configureApp() – конфигурација апликације публиковане на серверу
- restartApp() – поново покретање апликације на серверу
- terminateApp() – уклањање апликације која је покренута на серверу
- deployments() – листа информација о апликацији публиковане на серверу

Преглед главних функција за управљање налозима и корисницима:

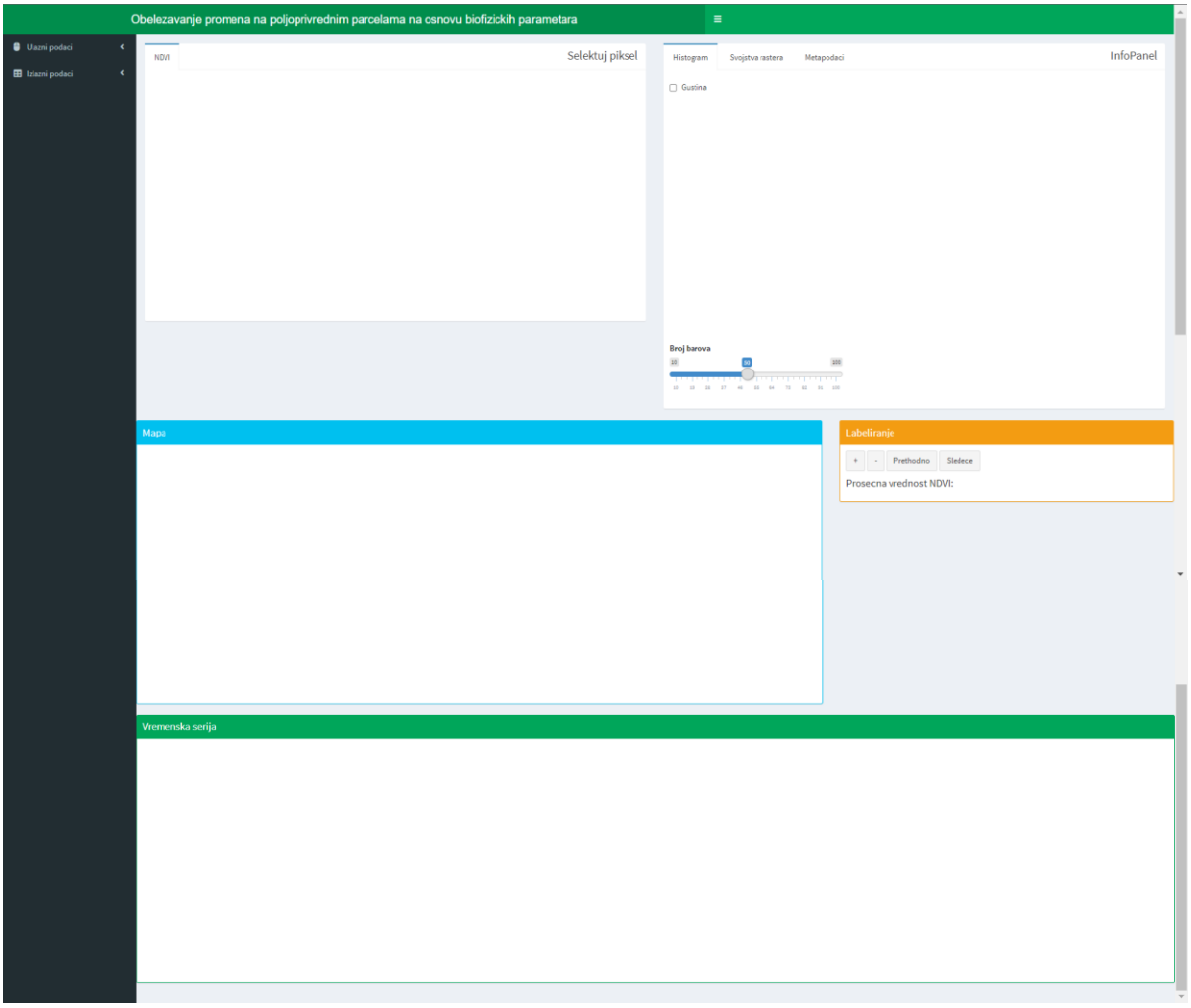
- setAccountInfo() – регистровање налога
- removeAccount() – уклањање регистрованог налога
- accountInfo() – информације о регистрованом налогу

Конкретна употреба *rsconnect* програмског интерфејса приказана је у поглављу *Постављање веб апликације на клауд (shinyapps.io).*

# 4. Израда веб картографске апликације за обележавање промена на пољопривредним парцелама на основу биофизичких параметара

У овом поглављу биће објашњена израда веб картографске апликације за обележавање промена на пољопривредни парцелама на основу биофизичких параметара. Интерфејс веб картографске апликације састоји се од пет целина. Целине представљају панеле са следећим функцијама:

1. Бочни панел за учитавање и извоз података
2. Панел за приказивање учитаних растера и селекцију пиксела
3. Инфо панел подељен у три таба:
    a. Хистограм
    b. Својства растера
    c. Метаподаци
4. Лабелирање
5. Временска серија
6. Панел *Leaflet* мапа




*Слика 21: Почетни екран апликације*

У поглављу 3.2 и 3.3, детаљно је објашњен поступак креирања веб апликације помоћу *Shiny* и *Shinydashboard* модула. У наредном коду, дата је структура апликације и дизајн корисничког интерфејса. Апликација је функционално подељена на кориснички интерфејс ui и серверску логику *server <- function(input, output)*. Кориснички интерфејс представља *dashboardPage* који се састоји из *dashboardHeader*, *dashboardSidebar* и *dashboardBody*. У *dashboardHeader* дефинисан је наслов веб апликације и његова ширина, *dashboardSidebar* садржи мените за увоз и извоз података, *dashboardBody* се састоји из три *fluidRow* компоненте. У првом реду налази се панел за приказивање учитаних растера и селекцију пиксела и инфо панел; у другом реду налази се панел *Leaflet* мапе у којој се учитавају векторски подаци (границе парцела) и панел за лабелирање; у трећем реду налази се панел временске серије. У *server* делу креирана је функција са два аргумента, *input* и *output* за контролу улазних и излазних података.

```
ui <-  dashboardPage(

  skin = "green",

  dashboardHeader(

    title = "Obelezavanje promena na poljoprivrednim parcelama na osnovu biofizick
ih parametara

    titleWidth = "100%"

  ),

  dashboardSidebar(

    disable = TRUE

    ),

  dashboardBody(

    fluidRow(

        ...

    ),

    fluidRow(

        ...

    ),

    fluidRow(

        ...

      )

    )
```

```
  )
)
server <- function(input, output) {
}
shinyApp (ui = ui, server = server)
```

У наредним поглављима за израду сваког панела биће описана имплементација како корисничког интерфејса тако и кода који се покреће на серверу са одређеним степеном генерализације.

## 4.1 Бочни панел за учитавање и извоз података

Бочном панелу приступа се путем бургер менија који је лоциран са горње десне стране поред наслова апликације. Бочни панел креиран је уз помоћ `sidebarMenu()`, и садржи два менија која се односе на учитавање података у апликацију и извоз података. У менију за учитавање података могуће је учитати две врсте података, растерске податке и векторске. Обе врсте података односе се на парцелу, с тим што се за растерске податке учитава читав сет података односно временска серија, а за парцелу *shp* фајл са полигоном. Редослед учитавања није битан из перспективе апликације. Други мени, односи се на експортовање *csv* фајла у коме се чува листа са лабелама за сваки датум временске серије.

```
#UI
dashboardSidebar(
    sidebarMenu(
      menuItem("Ulazni podaci", icon = icon("mouse"),


               fileInput("file1", "Ucitaj rastere",
                         multiple = TRUE,
                         accept = c("image/*"),
                         buttonLabel = "Ucitaj...",
                         placeholder = "Nema ucitanih rastera"
               ),
               fileInput("filemap", "Ucitaj poligone",
                         multiple = TRUE,
                         accept = c('.shp','.dbf','.sbn','.sbx','.shx",".prj"),
```

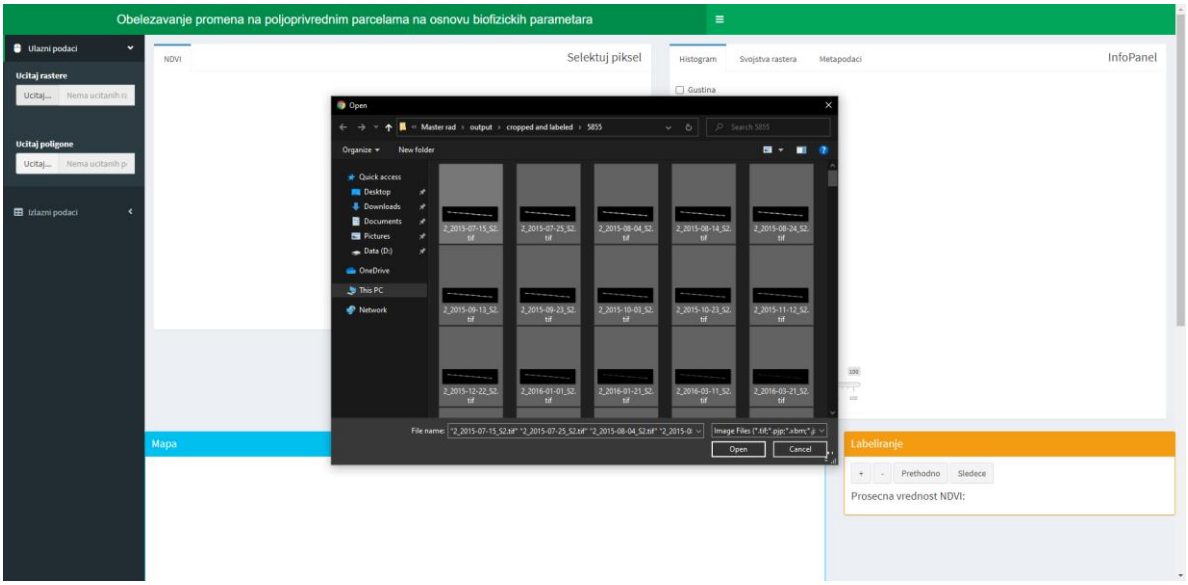
```
                        buttonLabel = "Ucitaj...",
                        placeholder = "Nema ucitanih poligona"
                )
      ),
      menuItem("Izlazni podaci", icon = icon("table"),
                textInput("outputFile", h6("Sacuvaj kao"),
                        value = "parcela"),
                downloadButton("downloadData", "Preuzmi")
      )
    )
  )
#Server


options(shiny.maxRequestSize=700*1024^2)
  map <- reactive({
    req(input$filemap)
    shpdf <- input$filemap
    tempdirname <- dirname(shpdf$datapath[1])
    # Rename files
    for (i in 1:nrow(shpdf)) {
      file.rename(
        shpdf$datapath[i],
        paste0(tempdirname, "/", shpdf$name[i])
      )
    }
    map <- readOGR(paste(tempdirname,
                        shpdf$name[grep(pattern = "*.shp$", shpdf$name)],
                        sep = "/"
    ))
```

```
    observeEvent(input$file1,{
    chr = character()
    dates <- list()
    pth <- input$file1
    pth_sub <- pth$datapath[1]
    tmpdirname <- str_remove(pth_sub, "/0.tif")
    for(nr in 1:length(input$file1[, 1])){
      raster_name_tif <- input$file1[nr, 'name']
      chr[nr] <- input$file1[nr, 'datapath']
      raster_name <- str_remove(raster_name_tif, ".tif")
      dates[nr] <- raster_name
      tmpdirname <- gsub("\\\\", "/", tmpdirname)
      to_rename <- pth$datapath[nr]
      file.rename(
        to_rename,
        paste0(tmpdirname, "/", raster_name_tif)
      )
      datapath_raster <- paste(tmpdirname, "/", raster_name_tif, sep="", collapse=
NULL)
      chr[nr] <- datapath_raster
    }
    lista_rastera <<- stack(chr)
        output$downloadData <- downloadHandler(
      filename = function() {
        paste(input$outputFile, ".csv", sep = "")
      },

      content = function(file) {
        write.csv(df_fn_label, file, row.names = FALSE)
      }
```

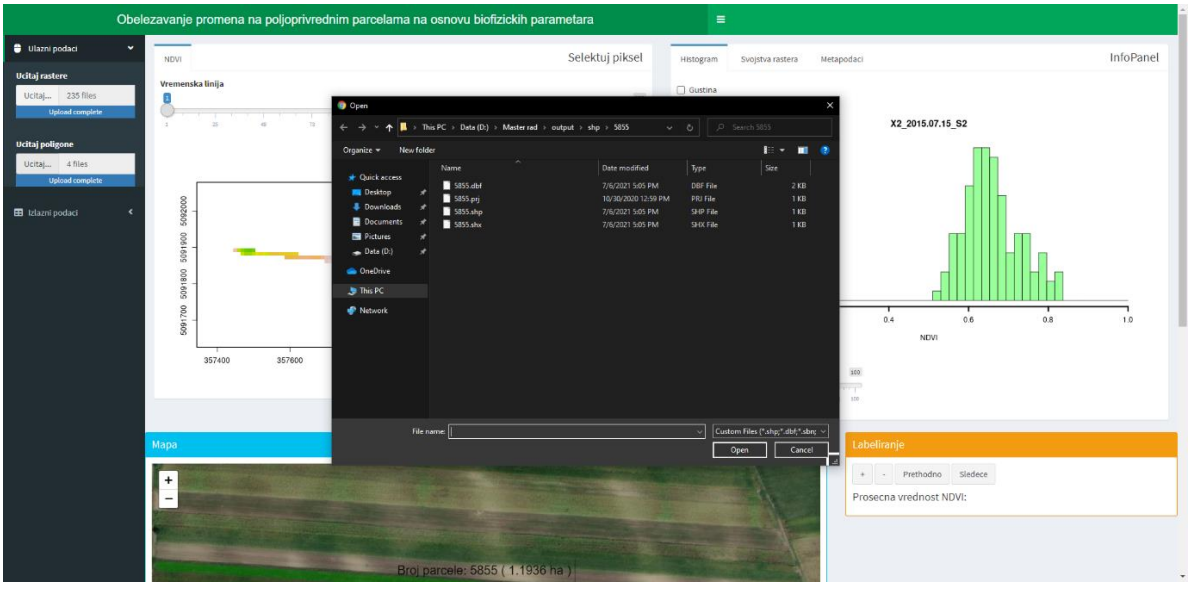
```
)
```

На серверској страни прво је дефинисана максимална количина података која се може учитати у апликацију. Подразумевана вредност за учитавање података износи 5MB, али се може променити путем **`options`** методе где MB изражавамо као $N*1024^2$.

Реактивност овог веб фрејмворка примарно је дизајниран да срачунава вредности и покреће акције инициране неким догађајем при свакој пмени улазног податка. У овом случају догађај је клик на дугме за унос података, а као улазни података уносе се датотеке односно растерске слике и векторски подаци који ће бити приказани у апликацији помоћу **`observeEvent`**.

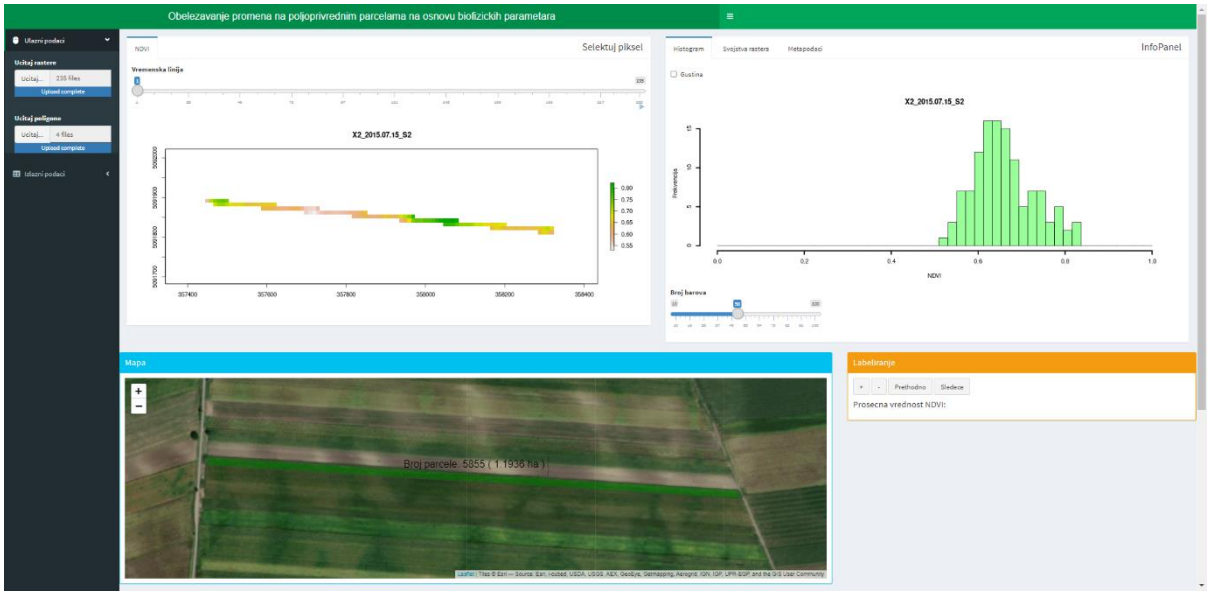



*Слика 22: Учитавање растерских*




*Слика 23: Учитавање векторских података*

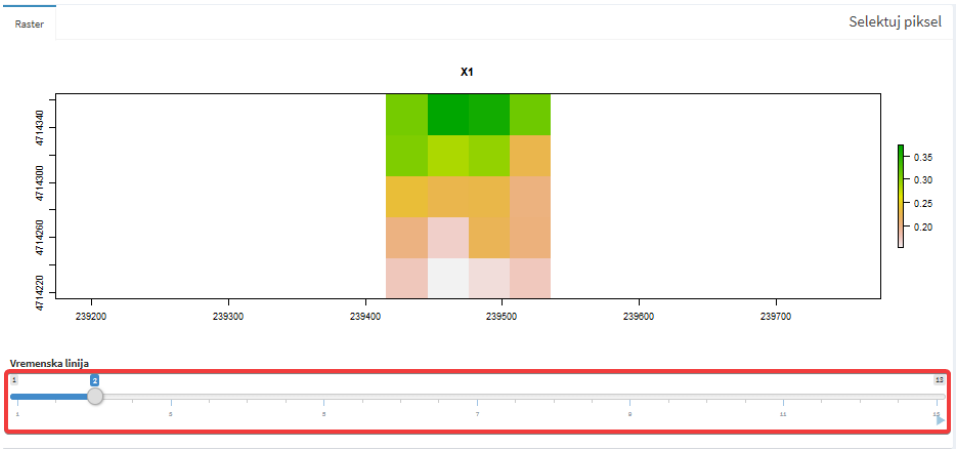
Учитане датотеке се смештају у посебан растерски објекат класе *stack*. *Rasterstack* представља колекцију *RasterLayer* објеката који деле исти просторни обухват и резолуцију. *RasterStack* може бити креиран од *RasterLayer* објеката, растерских фајлова или њихове комбинације. *RasterStack* објекат може бити креиран и на основу *SpatialPixelsDataFrame* објекта или *SpatialGridDataFrame* објекта.




*Слика 24: Учитани подаци*

## 4.2 Панел за приказ учитаних растера и селекцију пиксела

Овај панел, заједно са инфо панелом, креиран је са главном компонентом *body* компоненте **`tabBox`** у оквиру које је дефинисан назив панела и ширина. На слици 25. можемо видети приказ првог учитаног растера из колекције растера, функција за овај приказ **`plotOutput`**. Одмах испод, налази се временска линија, односно слајдер, помоћу кога корисник може смењивати на екрану учитане податке. Слајдер је креиран помоћу функције **`sliderInput`** у оквиру кога подешавамо минималну и максималну вредност слајдера, корак слајдера, ширину и оквир **`tabBox`** и анимацију. У десном доњем углу налази се дугме за пуштање анимације. Приликом покретања анимације, учитани растери ће се смењивати један за другим, од почетка до краја.




*Слика 25: Временска серија података*

На серверској страни покреће се код за креирање **SpatialPoints** објекта коме се затим додељује координатни систем учитаних података помоћу **crs** и затим му се додељује пројекција – **spTransform**. Функција **eventReactive** омогућава да се селектује пиксел директно са растера односно координатни пар (x,y).

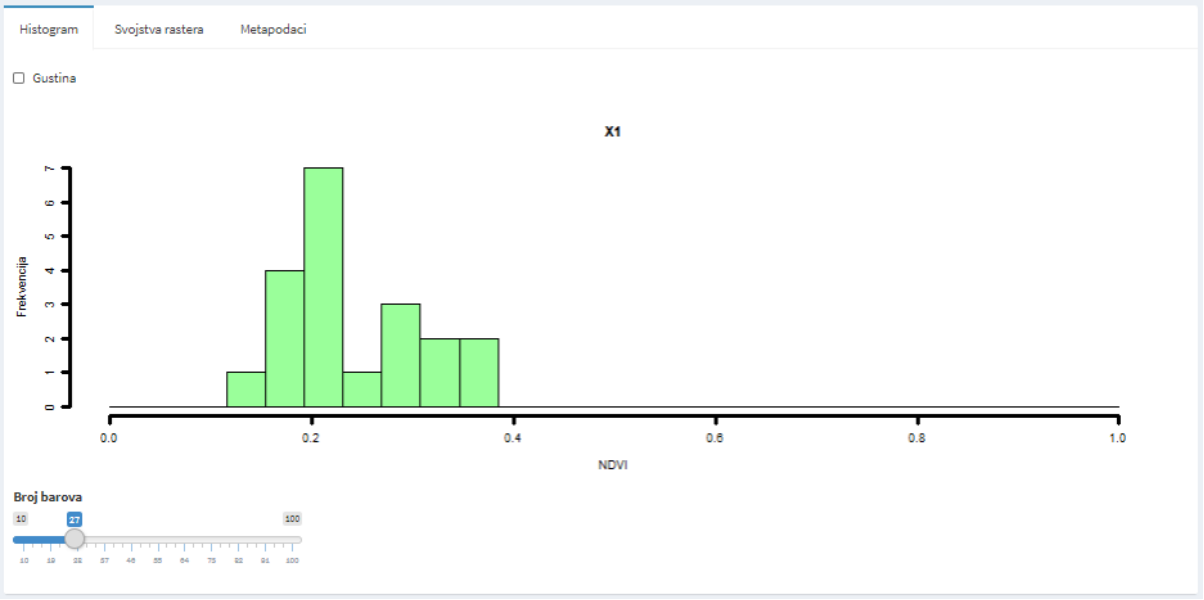
```
#UI

fluidRow (

      tabBox (

        title = "Selektuj piksel", id = "panel1",

        tabPanel ("Raster",

                  plotOutput("raster_iz_liste", click = "raster_klik"),

                  sliderInput("layer", "Vremenska linija", min = 1, max = 13, value
= 1, step = 1, width="100%", animate = TRUE))

      )

#Server strana

# Kreiranje objekta SpatialPoints i dodeljivanje projekcije

    xy <- SpatialPoints(data.frame(x_sr ,y_sr))

    crs(xy) <- crs(lista_rastera)

    xy <- spTransform(xy, "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs") #par
ametri projekcije

Selekcija_koordinata <- reactive({

      req(input$raster_klik$x)

      c(input$raster_klik$x, input$raster_klik$y)

    })

    value <- eventReactive(input$raster_klik$x,{

      extract(lista_rastera,cellFromXY(lista_rastera, Selekcija_koordinata()))

    })
```

## 4.3 Инфо панел (Хистограм, Својства, Метаподаци)

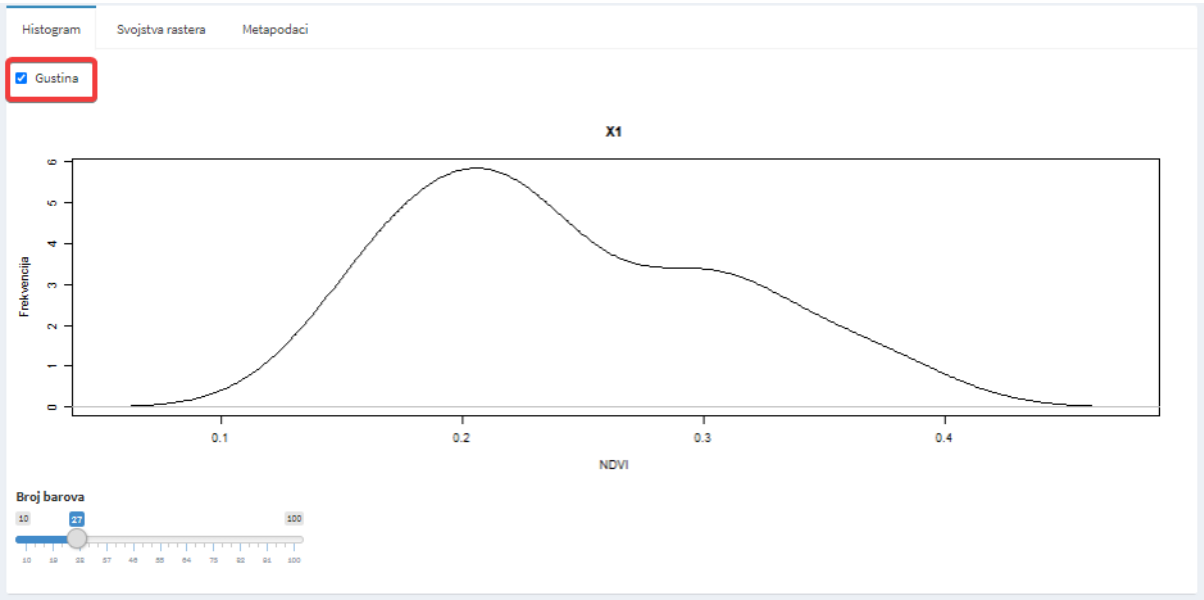
Инфо панел састоји се из три таба. Први таб односи се на приказ хистограма оног растера који је приказан на екрану. Променом растера на временској линији, мења се и хистограм за дати растер. Фреквенција хистограма је по подразумеваном подешавању приказан на основу барова (*слика 26.*), али корисник може изабрати и приказ густине (*слика 27.*). Ова опција за смењивање омогућена је помоћу **`conditionalPanel-a`**.

```
#UI

tabBox(

        title = "InfoPanel", id = "panel2",

        tabPanel("Histogram",

              checkboxInput(inputId = 'opcija', label = 'Gustina', value = FALSE),

                 conditionalPanel(

                   condition = 'input.opcija == true',

                   plotOutput('gustinaPlot')

                   ),

                 conditionalPanel(

                   condition = 'input.opcija == false',

                   plotOutput('histo')

                 ),

                 sliderInput("bin", "Broj barova", min = 10, max = 100, value = 50
)

                 ),

        tabPanel("Svojstva rastera",

                 tableOutput("tablicaPx")

                 ),

        tabPanel("Metapodaci",

                 HTML("<b>Broj ucitanih slojeva:</b>"),

                 textOutput("brojLejera"),

                 HTML("<b>Rezolucija:</b>"),

                 textOutput("rezolucija"),
```

```
                  HTML("<b>Kartografska projekcija:</b>"),

                  textOutput("projekcija"),

                  h4("Selektovane koordinate"),

                  textOutput("ispisKoord")

                  )

    )

#Server

output$histo <- renderPlot({

      hist(lista_rastera[[input$layer]], ylab = "Frekvencija", xlab = "NDVI", col
= 'palegreen1', border = "black", lwd = 3, breaks = seq(0, 1, l = input$bin))

    })



    output$gustinaPlot <- renderPlot({

      plot(density(lista_rastera[[input$layer]]),  main = names(lista_rastera)[inp
ut$layer], xlab = "NDVI", ylab ="Frekvencija")

    })

  output$tablicaPx <- renderTable({

      svojstvaRastera <-  data.frame("Layer" = names(lista_rastera), "Value" = unl
ist(value())[1,])

      colnames(svojstvaRastera) <- c("Naziv rasterskog fajla","Vrednost piksela")

      rownames(svojstvaRastera) <- 1:nlayers(lista_rastera)

      svojstvaRastera

    })

output$projekcija <- renderText(projection(lista_rastera))

output$rezolucija <- renderText(res(lista_rastera))

output$brojLejera <- renderText(nlayers(lista_rastera))

output$ispisKoord <- renderText(formatC(round(Selekcija_koordinata(),2), format =
"f", digits = 2))
```

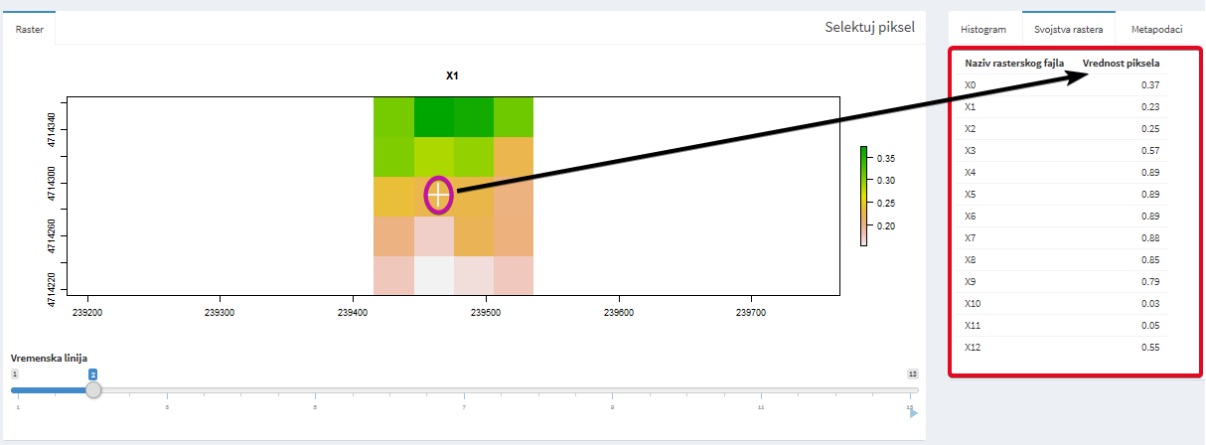



*Слика 26: Хистограм (степенасти)*



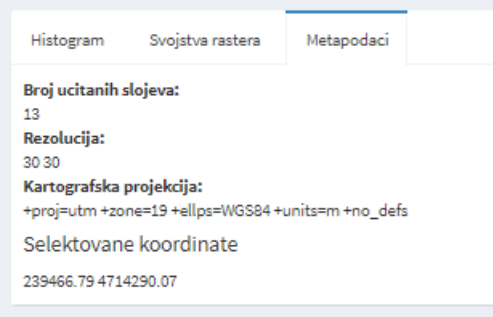

*Слика 27:Хистограм (густина)*

Други таб односи се на приказ својства растера одабраног пиксела (слика 28.). Овај таб приказује вредност одбраног пиксела за све растере који се налазе у стацку а резултат се приказује у табеларном приказу. За приказ коришћен је **`tableOutput`** а табела се креира са новим вредностима на серверској страни при сваком клику на растер **`tablicaPx`**.




*Слика 28: Својства растера*

Трећи таб намењен је за приказ метаподатака учитаних растера (*слика 28.*). Метаподаци који се приказују односе се на број учитаних растера, резолуцију и картографску пројекцију (*слика 29.*). Ови метаподаци представљају заправо атрибуте растерских података. На основу функција из растерске библиотеке, врло лако можемо доћи до ових атрибута коришћењем фунцкија нлауерс, рес, пројецтион. За приказ коришћен је **`textOutput`** у комбинацији са **`HTML`** функцијом која нам омогућава непосредно додавање хтмл елемената на корисничком интерфејсу.




*Слика 29: Метаподаци*

## 4.4 Лабелирање

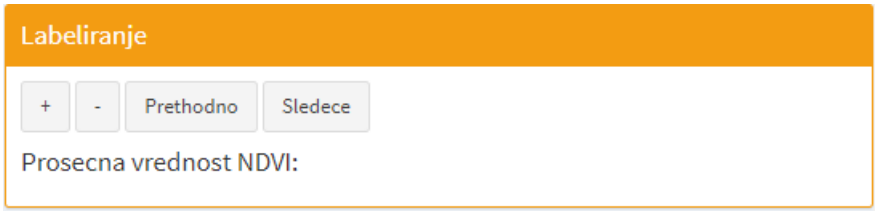
Панел лабелирање састоји се из четири дугмета. Замисао панела јесте да се уз помоћ њега могу смењивати растери у инфо панелу и да се на основу аутоматске срачунате просечне вредности *NDVI* и осталих доступних информација, могу лабелирати парцеле односо растери који задовољавају одређени критеријум односно не задовољавају га. Дугме + лабелира датум снимка са 0 који задовољавају дефинисани праг, а дугме - лабелира са вредношћу 1.

```
#UI

box(

    title = „Labeliranje“, status = „warning“, solidHeader = TRUE, width = 4,
```

```
          actionButton(inputId = „labela_0“, „+“),
          actionButton(inputId = „labela_1“, „-„),
          actionButton(inputId = „previousImage“, „Prethodno“),
          actionButton(inputId = „nextImage“, „Sledece“),
          h4(„Prosecna vrednost NDVI: „),
          h4(textOutput(„average_ndvi“))
        )
#Server

observeEvent(input$previousImage,{
      previous_value = input$layer – 1
      num_of_layers <- nlayers(lista_rastera)
      if (previous_value == 0 ) {
        previous_value <-  num_of_layers
      }
        output$raster_iz_liste <- renderPlot({
        updateSliderInput(session, inputId = „layer“, value = previous_value)
        plot(lista_rastera[[input$layer]], main = names(lista_rastera)[input$layer
])
      })
      layermeans <- cellStats(lista_rastera[[previous_value]], stat='mean', na.rm=
TRUE)
      sredina <- mean(layermeans)
      sredina <- round(sredina, 2)
      output$average_ndvi <- renderText({toString(sredina)})
    })
    observeEvent(input$nextImage,{

      next_value = input$layer + 1
      num_of_layers <- nlayers(lista_rastera)
```

```
      if (next_value > num_of_layers ) {

        next_value <- 1

      }

      output$raster_iz_liste <- renderPlot({

        updateSliderInput(session, inputId = „layer“, value = next_value)

        plot(lista_rastera[[input$layer]], main = names(lista_rastera)[input$layer
])

      })

      layermeans <- cellStats(lista_rastera[[next_value]], stat='mean', na.rm=TRUE
)

      sredina <- mean(layermeans)

      sredina <- round(sredina, 2)

      output$average_ndvi <- renderText({toString(sredina)})

      updateActionButton(session, inputId = „labela_0“, „+“)

    })

  })

  df_fn_label <- data.frame(fn = character(), file_name = character(), label = num
eric())

    observeEvent(input$labela_0,{

      extract_fn <- filename(lista_rastera[[input$layer]])

      f_name  <- basename(extract_fn)

      df_fn_label <<- df_fn_label %>% add_row(fn = extract_fn, file_name = f_name,
label = 0)

    })

    observeEvent(input$labela_1,{

      extract_fn <- filename(lista_rastera[[input$layer]])

      f_name  <- basename(extract_fn)



      df_fn_label <<- df_fn_label %>% add_row(fn = extract_fn, file_name = f_name,
label = 1)

    })
```

У приложеном коду, можемо видети акције које се покрећу када се одређено дугме притисне. Акција за смењивање слика напред и назад, покреће код за рачунање средње вредности *NDVI* за једну парцелу. Када се слика лабелира одређено лабелом (0 или 1), креира се *dataframe* објекат и у њему се додају редови у којима се бележи назив растера и придодата лабела. Касније, ова табела ће бити извежена као *csv* датотека (*табела 3*).




*Слика 30: Лабелирање*

| file_name | label |
|---|---|
| 2_2015-07-15_S2.tif | 0 |
| 2_2015-07-25_S2.tif | 0 |
| 2_2015-08-04_S2.tif | 0 |
| 2_2015-08-14_S2.tif | 0 |
| 2_2015-08-24_S2.tif | 0 |
| 2_2015-09-13_S2.tif | 0 |
| 2_2015-09-23_S2.tif | 0 |
| 2_2015-10-03_S2.tif | 1 |
| 2_2015-10-23_S2.tif | 1 |
| 2_2015-11-12_S2.tif | 1 |
| 2_2015-12-22_S2.tif | 1 |
| 2_2016-01-01_S2.tif | 1 |
| 2_2016-01-21_S2.tif | 1 |
| 2_2016-03-11_S2.tif | 1 |
| 2_2016-03-21_S2.tif | 1 |
| 2_2016-04-20_S2.tif | 0 |
| 2_2016-04-30_S2.tif | 1 |
| 2_2016-05-20_S2.tif | 1 |
| 2_2016-05-30_S2.tif | 1 |
| 2_2016-06-09_S2.tif | 0 |
| 2_2016-06-19_S2.tif | 1 |
| 2_2016-06-29_S2.tif | 0 |
| 2_2016-07-09_S2.tif | 1 |
| 2_2016-07-19_S2.tif | 1 |
| 2_2016-07-29_S2.tif | 0 |
| 2_2016-08-08_S2.tif | 0 |
| 2_2016-08-18_S2.tif | 0 |
| … | … |

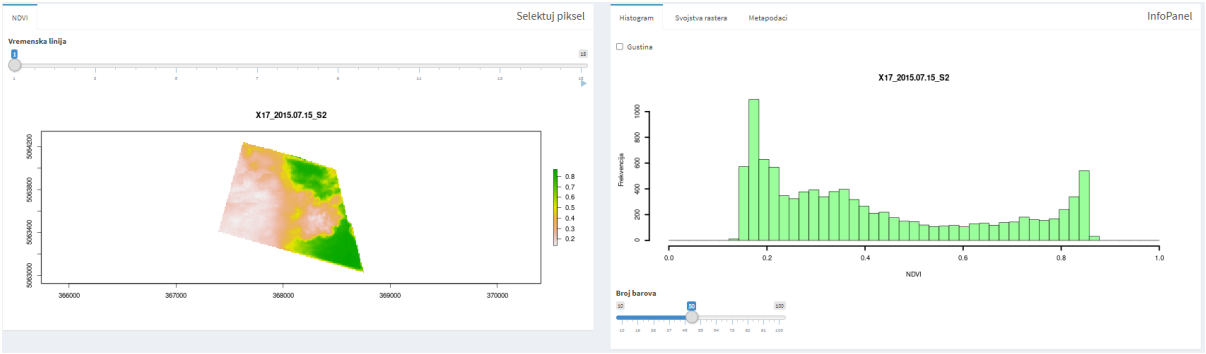
*Табела 3: Пример лабела у излазном фајлу*

## 4.5 Временска серија

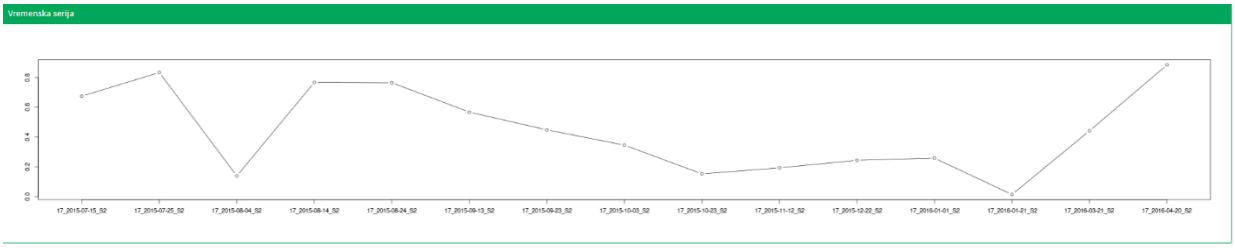
Панел временска серија предвиђен је да пружи приказ есенцијалних информација, говорећи наравно, у контексту ове апликације. Овај плот приказује како се вредност одабраног пиксела мењала кроз временску серију. У овом конкретном примеру, за селектовано подручје, можемо видети како се вредност *NDVI-a* мењала кроз период од годину дана (*слика 31.*). На *y*-оси нанет је опсег *NDVI* за учитане податке, а на *x*-оси, нанети су бројеви који означавају месеце. Из овог плота корисник може изанализирати тренд *NDVI-a* на овом подручју. У августу месецу, *NDVI* је био средње низак, да би током јесени забележио благи порасти а од јануара до фебруара имао нагли пад, а потом нагли пораст, све до априла где има свој пик.

Временска серија смештена је у трећи ред апликације заједно са *Leaflet* мапом. Као и до сада, коришћен је **`fluidRow`** за креирање новог реда, и **`box`** за смештање овог плота који је изгенерисан помоћу **`renderPlot`** функције у којој се, на основу координата, екстрактују вредности индекса свих учитаних растера из гомиле.

```
#UI strana

fluidRow(

      box(

        title = "Vremenska serija", status = "success", solidHeader = TRUE, width=
7,

        plotOutput("vremenskaSerija")

      )

   )

)

#Server strana

output$vremenskaSerija <- renderPlot({

      req(input$raster_klik$x)

      plot(1:nlayers(lista_rastera),value(), type = "b", xlab="Vreme", ylab="Vredn
ost")

}

)
```




*Слика 31: Селектовање пиксела*




*Слика 32: Временска серија растерских података*

## 4.6 Панел *Leaflet* мапа

Функција панела Leaflet мапе јесте да се аутоматски детектује географска локација учитаних растера. Да би користили мапу, морамо обезбедити извор података. У конкретном примеру, изабрани провајдер тајлова за ову мапу је *Esri World Imagery*, провајдер се дефинише уз помоћ **`addProviderTiles`**. На основу просте аритметичке средине координата срачуната је средња вредност која је одабрана за центар мапе **`setView`** у коме се дефинише и ниво зума. Позивом **`renderLeaflet`** креира се *Leaflet* мапа.

```
#UI strana

fluidRow(

         ...

      box (

        title = "Mapa", status = "info", solidHeader = TRUE, width = 5, leafletOut
put("Map")

        )

      )

#Server strana

output$Map <- renderLeaflet({

      leaflet() %>%

        setView(xy@coords[1],xy@coords[2], 8) %>%
```

```
        addProviderTiles("Esri.WorldImagery")

    })
```


*Слика 33: Leaflet мапа*

# 5. Постављање веб апликације на интернет

По завршетку развијања веб апликације, следећи логични корак је постављање апликације на интернет како би била доступна предвиђеним корисницима.

Постоји више начина како се апликација може поставити на веб:

- **RStudio Connect**
- **Shiny сервер отвореног кода**
- **shinyapps.io**

Избор опције за публиковање апликације треба да проистакне из детаљне анализе доступних технологија, финансијских ограничења и кадра који ће радити на евентуалном одржавању или даљем развоју апликације.

## 5.1 RStudio Connect

*RStudio connect* је комерцијална платформа која омогућава веб публикацију не само *shiny* веб апликација већ и *R Mardown* извештаја, *Plumber* програмских интерфејса и модела. Ова платформа омогућава интеракцију и са корисницима који не користе искључиво *R* програмски језик, инсталира се на серверу који може имати конфигурисан фајервол што представља важан слој заштите података унутар пословне организације или дефинисане групе корисника. *Rstudio* сервер омогућава централизовани приступ, скалирање на захтев у циљу побољшања рачунарских перформанси и приступ *Rstudio* интегрисаном развојном окружењу кроз веб претраживач.

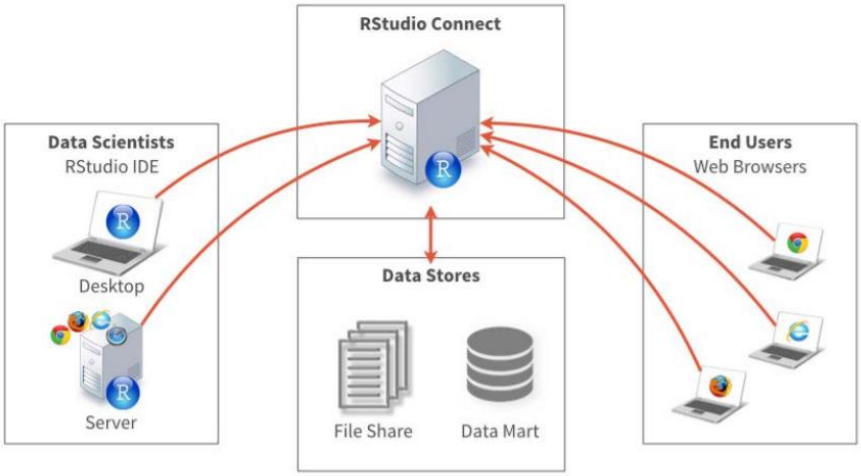
*Rstudio connect* платформа предвиђена је за Линукс оперативни систем и захтева инсталиран *R* програмски језик. Хардверски захтеви зависе од броја апликације, њихових хардверских захтева и броја активних корисника. Стандардна спецификација за продукциони сервер креће се између 8 и 16 процесорских језгара и 32-128 гигабајта РАМ-а.

Погодности *Rstudio connect-a:*

- Публикација притиском на дугме
  - Простим кликом на дугме за публикацију из *Rstudio Desktopa, Rstudio Server-a* и *Rstudio Server Pro-a* дели се било који статички или динамички садржај креиран у *R*-у са предефинисаном групом људи која има приступ.
- Управљање садржајем
  - Омогућава контролу дељења садржаја између *R* корисника – приступ читања, приступ колаборације, планирање ажурирање и приступ логовима.
- Планско ажурирање и дистрибуција

- Подешавање аутоматског генерисања *R Markdown* извештаја и дистрибуције последње верзије путем електронске поште члановима тима.

- Сигурност
  - Контрола приступа подацима, подршка системима за заштиту идентитета, LDAP/Активни директоријум, *Google Oauth, PAM,* прокси аутентификација, базе података.




*Слика 34: Rconnect Server дијаграм*

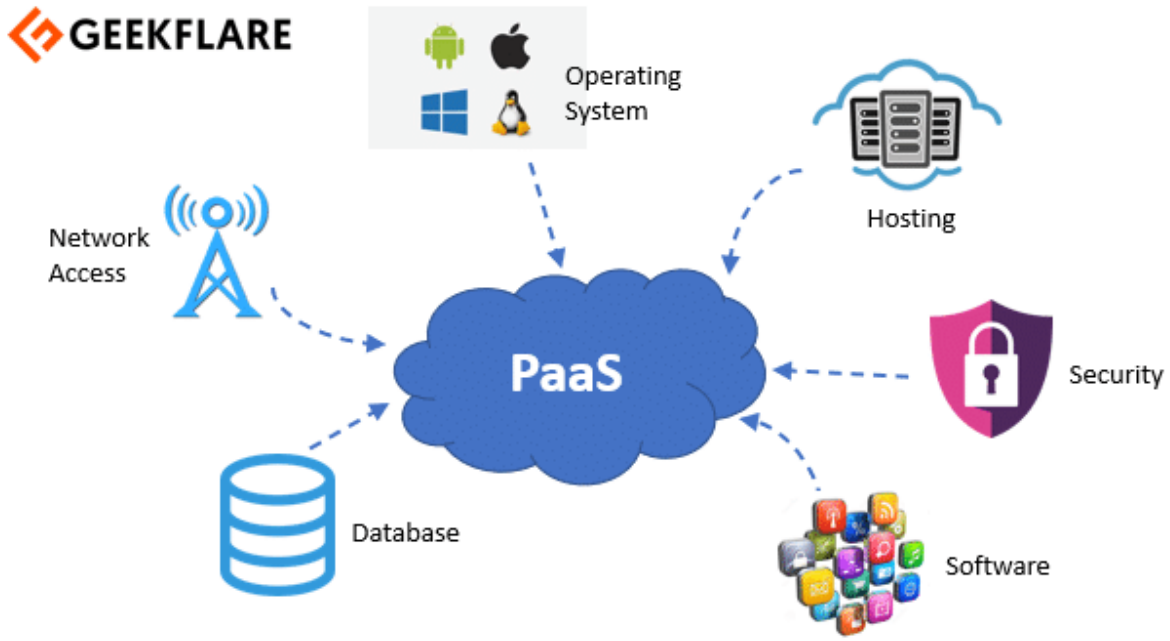
## 5.2 *Shiny* сервер отвореног кода

Ова платформа је алтернативно решење отвореног кода развијена за Линукс оперативни систем доступна под AGPLv3 лиценцом. Доступна је за следеће Линкус ОС Ubuntu 16.04+, Red Hat/CentOS 6+, and SUSE Linux Enterprise Server 12+. За разлику од RStudio connect, ова платформа је јако ограничена у погледу онога што пружа. Захтева мануелну публикацију, не подржава више од једног R процеса, као ни интеграцију са аутентификационим провајдерима, нити подржава Plumber програмски интерфејс као и R Markdown извештаје. Подржава више shiny апликација на једном серверу где свака има свој УРЛ односно порт.

## 5.3 shinyapps.io

Представља клауд решење за хостовање апликација развијен од стране *RStudia,* доступан у комерцијалном али и бесплатном режиму. *Shinapps* базиран је на PaaC (платформа као сервис) технологији. PaaC је еволуирао из CaaC (софтвер као сервис) технологији и представља интегрисано решење које је доступно путем интернета. Овакав тип производа представља скуп компонената и софтверских подсистема који омогућавају услугу или развој потпуно функционалног производа односно сервиса. Бенефит оваквог приступа огледа се у томе што се вендор стара о ресурсима односно

оперативном систему, одржавању софтвера, печовању и инфраструктури, док са друге стране корисник ПааС услуге сву своју пажњу може посветити развијању бизнис логике.




*Слика 35: Платформа као сервис*

### 5.3.1 Постављање веб апликације на клауд (shinyapps.io)

Предуслови за публикацију апликација преко ове платформе су поседовање налога на *shinyapps.io, Rstudio* интегрисано развојно окружење и најсвежија верзија *rsconnect* модула.

Приликом креирања налога, треба имати у виду да ће се име овог налога користити и у домену за сваку публиковану апликацију.

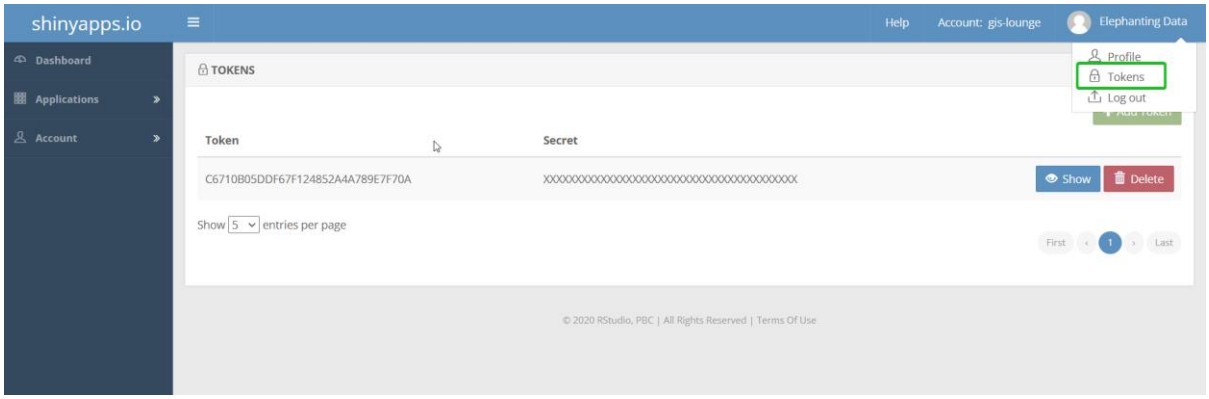
Инсталација модула:

```
install.packages('rsconnect')
```

Након што се модул инсталира, потребно га је учитати у *R* сесију:

```
library(rsconnect)
```

Након инсталације модула и креирања налога, потребно је конфигурисати *rsconnect* у *Rstudio-y* како би се повезали са налогом преко кога желимо да публикујемо апликацију. Повезивање се врши преко генерисаног токена и генерисане тајне на *shinyapps.io*.




*Слика 36: Управљање токенима за ауторизацију*

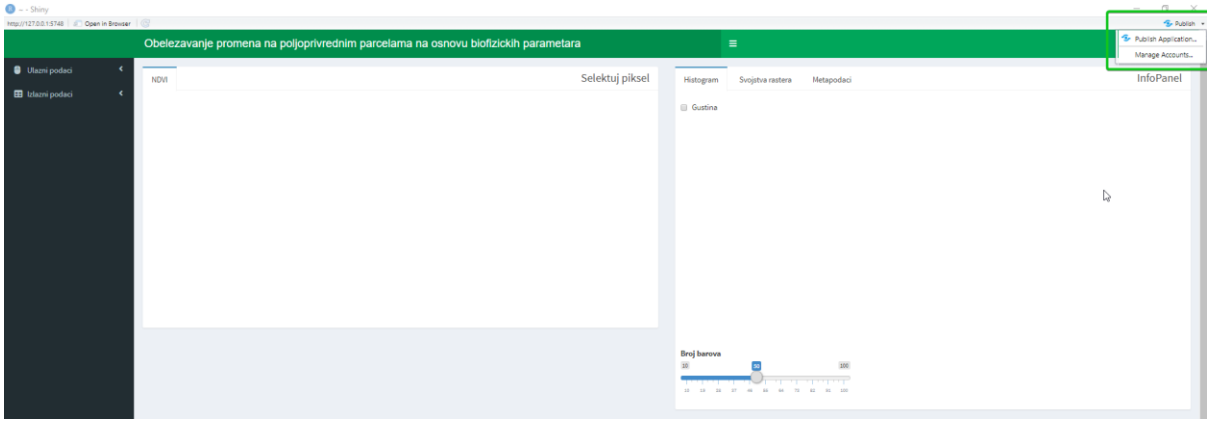
На онај начин извршава се ауторизација налога на корисничком уређају. Ауторизација се извршава следећом командом у конзоли:

```
rsconnect::setAccountInfo(name='korisnicko_ime',
                          token='OvajTokenJeUpravoIzmisljen',
                          secret='OvaTajnaJeMozdaPrava')
```

Корисник налога може управљати токенима, генеришући нове и бришући их.

Процес развоја апликације и њено покретање већ је објашњено у ранијим поглављима. Након што је апликација добро истестирана и оперативна, спремна је за публиковање. Постоје две методе публиковање и обе су лако изводљиве.

Прва метода јесте путем *Publish* дугмета, које је доступно након што се покрене апликација




*Слика 37: Публиковање апликације директно из Shiny претраживача*

Друга метода подразумева публиковање апликације директно из конзоле у развојном корисничком окружењу:

```
library(rsconnect)

deployApp()
```

Након покретање горенаведене команде, *rsconnect* прави листу свих модула који су коришћени у апликацији, затим се апликација и листа модула шаље *shinyapps.io* сервису који врши инсталације и подиже апликацију на веб. Сви модули доступни на најпознатијим репозиторијумима као што су *CRAN, GitHub,* и *BioConductor* могу се инсталирати.

Након публиковања, прави се виртуализовани сервер са копијом кода апликације, односно инстанца. Могуће је направити више инстаци исте апликације, али треба имати у виду да ће свака од њих користити посебни фајл систем.

У следећој табели можемо видети опције које нуди *shinyapps.io* за апликације различитих величина:

| Тип инстанце | Меморија |
|---|---|
| Мала | 256 MB |
| Средња (подразумевана) | 512 MB |
| Велика | 1024 MB |
| хВелика | 2048 MB |
| ххВелика | 4096 MB |

*Табела 4: Меморије предвиђене сходно величини апликације*

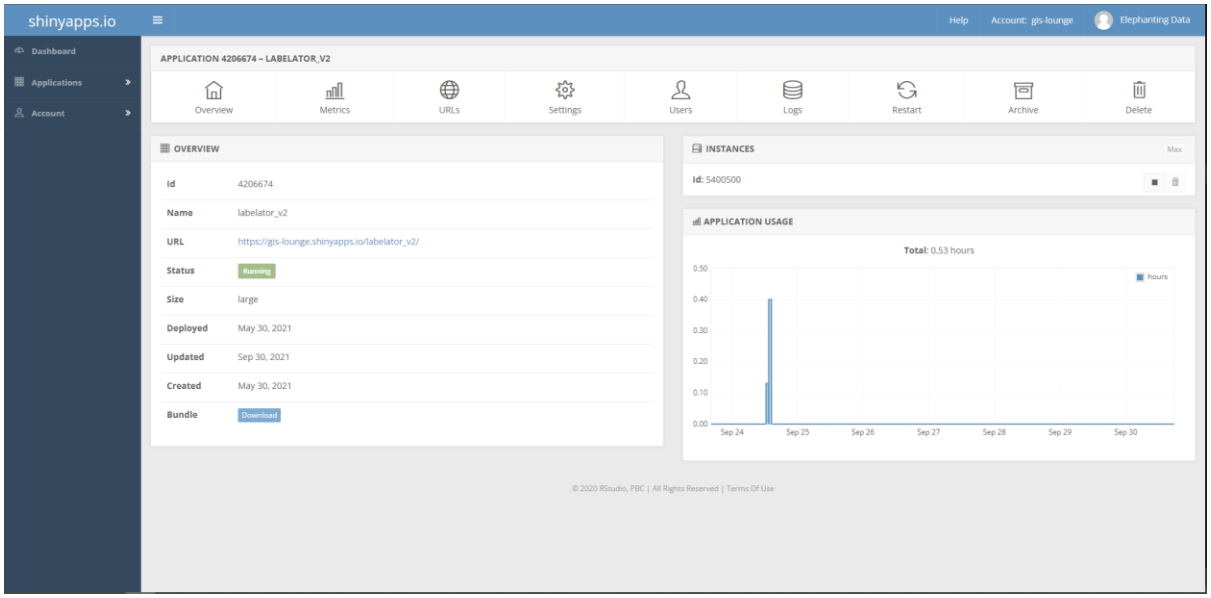
С времена на време, потребно је по захтеву или потреби реконфигурисати апликацију. Примера ради, променити јој име или променити меморију резервисану за њу:

```
rsconnect::configureApp("APPNAME", size="small")
```

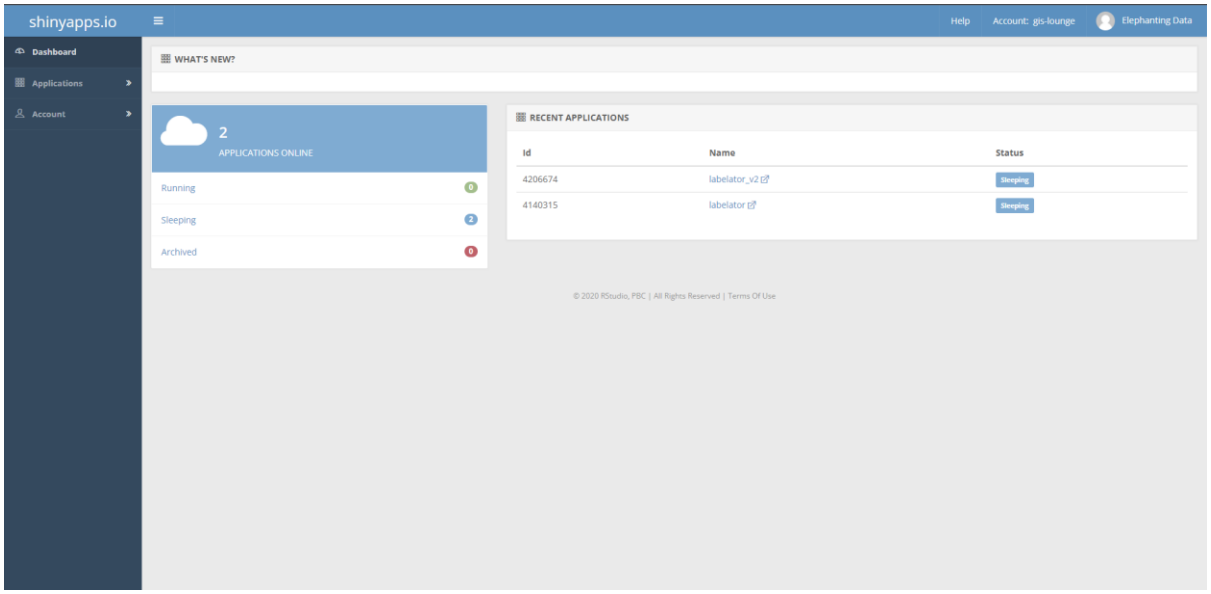
Током живота апликације, постоји оправдана потреба да се с времена на време прегледају логови уколико се појави одређени проблем у раду апликације. Лог садржи и stdout (print, cat) stderr (message, warning, stop), такође је могуће увести и константи мониторинг путем следећег параметра streaming = TRUE.

```
rsconnect::showLogs()
```

На контролној табли *shinyapps.io* могуће је управљати апликацијама и посматрати ресурсе које оне користе као и управљати аутентификацијом за различите кориснике.




*Слика 38: Апликација*



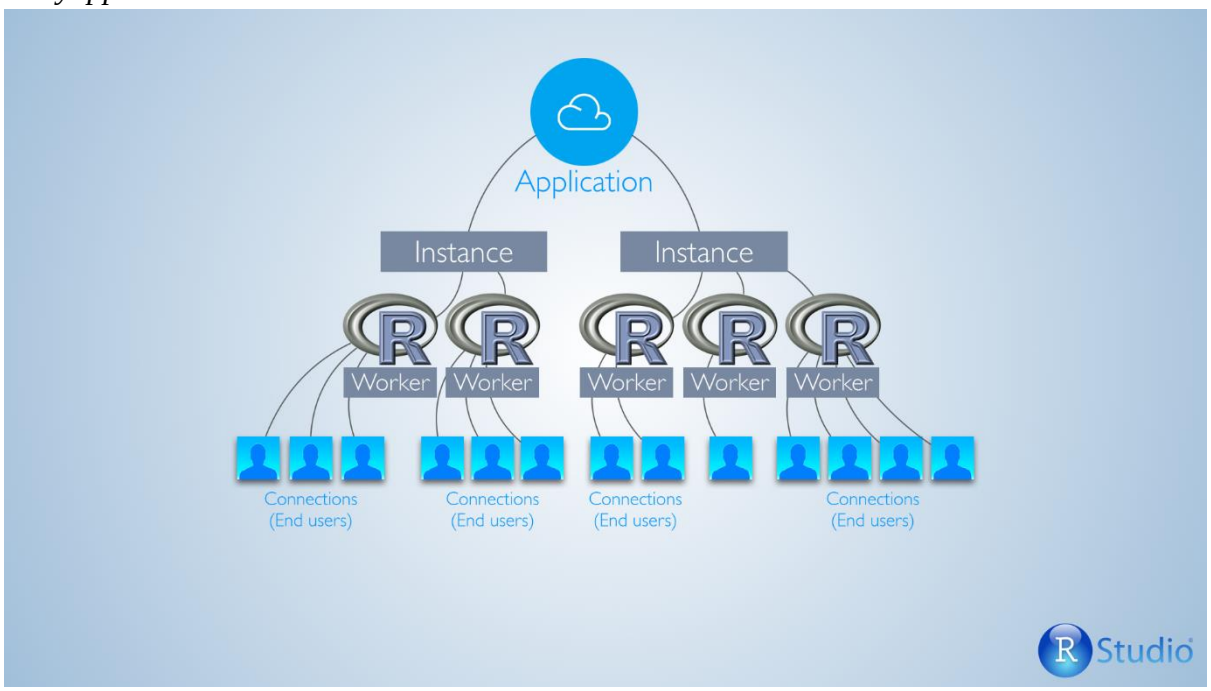

*Слика 39: Контролна табла*

Уклањање инстаце апликације могуће је извршити у оквиру саме командне табле или путем конзоле у интегрисаном развојном окружењу користећи следећу команду:

```
terminateApp("<ImeAplikacije>")
```

### 5.3.2 Скалирање и перформансе

Скалабилност и перформансе представљају важне принципе архитектуралног дизајна. *R* апликација не може у истом тренутку да опслужује више корисника односно више захтева. У већини случајева, ово не представља проблем, јер процесирање захтева на страни сервера се мери милисекундама. Међутим, са порастом комплексности апликације и броја захтева, перформансе ипак могу бити угрожене. У просеку, *R* процес може процесирати између 5 и 30 захтева у секунди. У даљем тексту, упознаћемо се са кључним идејама решавања проблема перформанси односно скалабилности са *shinyapps.io.*

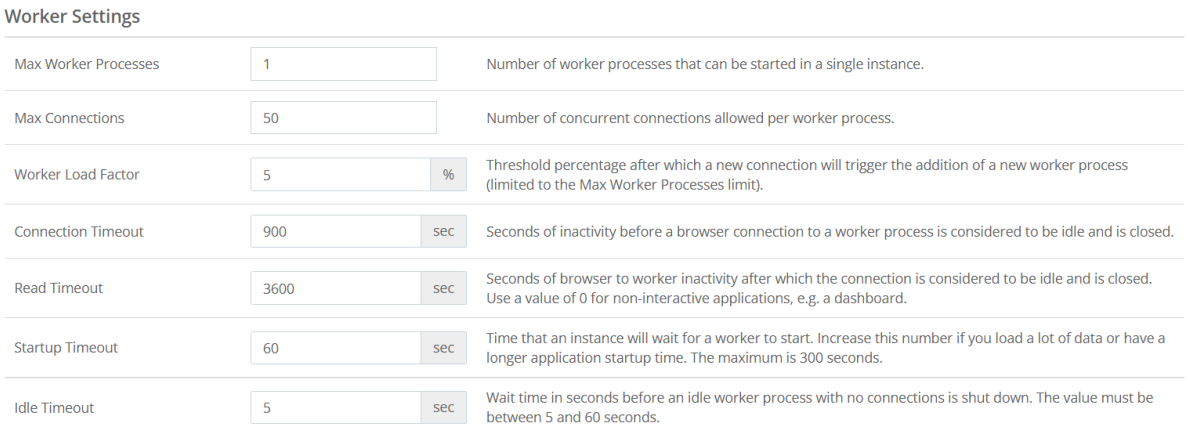



*Слика 40: Скалабилност дијаграм*

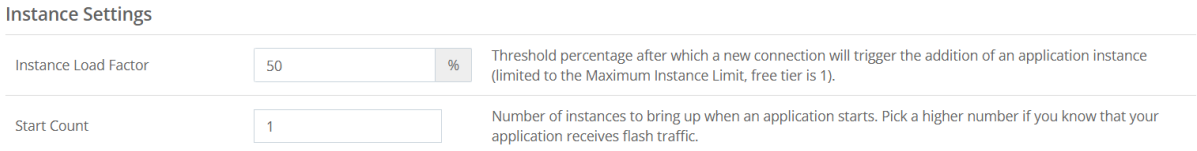
Апликација се састоји из фајлова који дефинишу кориснички интерфејс и инструкције за сервер. Приликом публиковања апликације, ови фајлови се учитавају на *shinyapps.io,* апликација мора имати барем једну инстанцу и она се креира када корисник пошаље захтев. Инстанца представља сервер који треба да прими и обради захтев корисника. Уколико ниједан корисник не користи апликацију, односно не шаље захтеве ка њој, ова инстаца ће се аутоматски угасити све док се поново не пошаље захтев. Инстанца покреће посебан тип *R* процеса (енг. *worker*) који процесирају одређени број захтева који су дефинисани конфигурацијом. Инстанца није ограничена на један овакав процес, на основу броја захтева, ако један процес постане преоптерећен, аутоматски ће се креирати нови процес који ће обрађивати захтеве. Конекције су креиране кроз веб претраживач оног тренутка када је захтев послат или када се освежи страница у веб претраживачу.

Два фактора користе се како би се одредили фактори за скалирање ресурса:

- Оптерећење на *R* процесу (енг. *worker*) (*слика*)
- Оптерећење инстанце (*слика*)



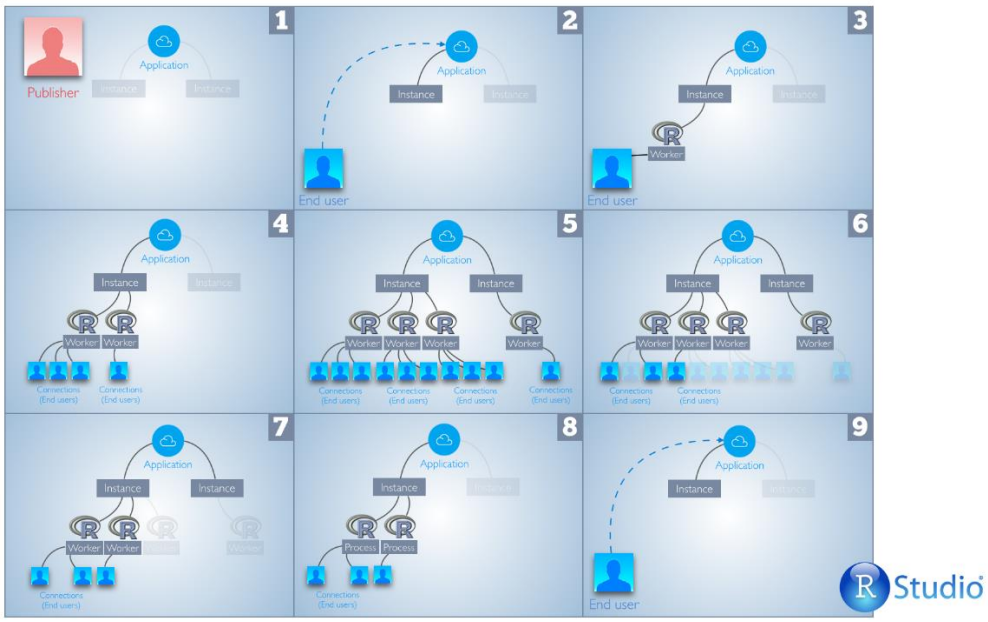

*Слика 41: Подешавања R процеса*




*Слика 42: Подешавања инстанце*

Код првог фактора дефинише се процентуални праг након кога ће се додати још један *R* процес уколико се дефинисани број конекција пређе.

Други фактор дефинише процентуални праг након кога ће се повећати број инстанци уколико се пређе дефинисани број конекција. Поред овог фактора, потребно је дефинисати максимални број инстанци. Битно је напоменути да код бесплатне верзије, само једна инстанца је доступна за апликацију.

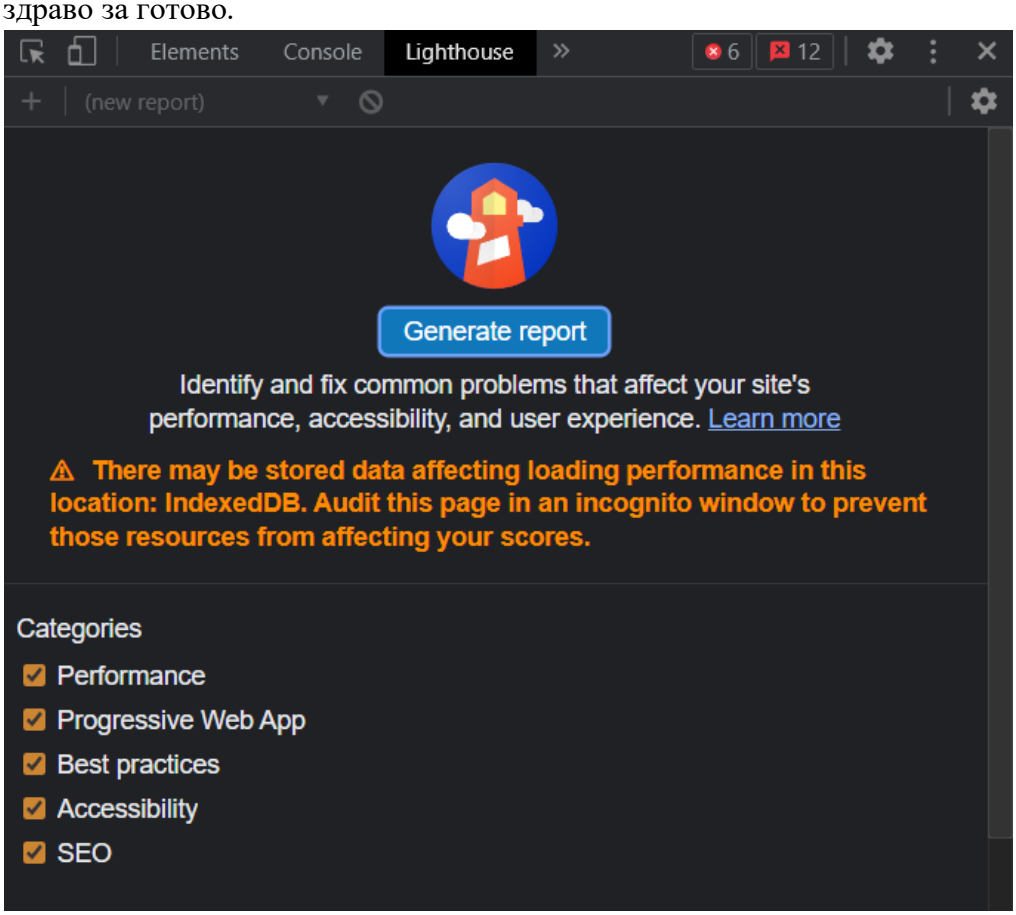



*Слика 43: Животни циклус апликације*

### 5.3.3 *Lighthouse* тестирање

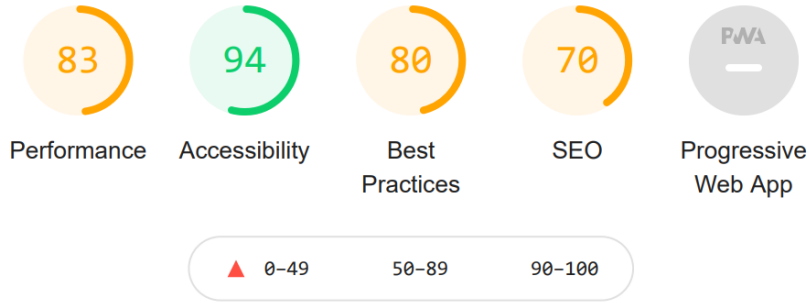
*Lighthouse* представља аутоматизовани алат отвореног кода који служи за тестирање кључних перформанси веб апликације односно веб страница једне апликације. Овај алат покреће се из *Chrome DevTools-a*, корисник треба одабрати категорије за које жели да покрене тестирање, на располагању су:

- Перформансе – главни фокус је на брзини приказа веб странице из перспективе очекивања корисника
- Прогресивност веб апликације – ова метрика не даје скор као све остале, већ даје пролазну оцену да ли је апликација прогресивна или не (проверава преусмеравање између HTTP и HTTPS, брзину учитавања на 3Г мрежи, итд.)
- Најбоља пракса – ова метрика покрива низ добрих пракси које се тичу развоја веб апликације (проверава да ли постоје грешке логоване у конзоли, да ли се користе програмски интерфејси који више нису у употреби, да ли је кеш апликације валидан, да ли је приступ за геолокацију послат, да ли се слике приказују у доброј резолуцији, итд.)
- Приступачност – ова метрика проверава атрибуте *html* елемената, контраст позадина, доступност линкова на веб страници, итд.
- CEO оптимизација – ова метрика даје оцену о CEO оптимизацији веб странице, премда треба узети у обзир да је ово само површан тест који не треба узети здраво за готово.

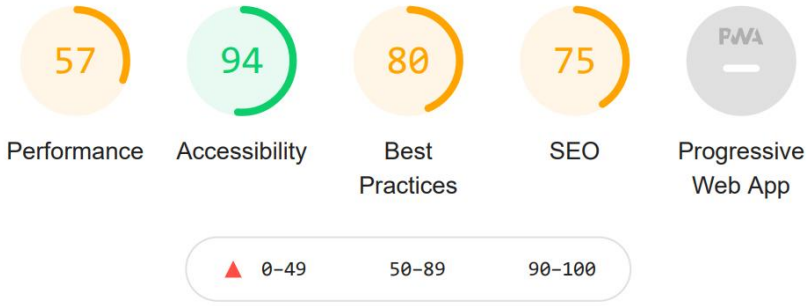



*Слика 44: Lighthouse* тест

С обзиром на респонсивност *shiny* веб апликације, тестирање је извршено и за десктоп и за мобилну верзију апликације. Са *слике 45.* може се видети да десктоп верзија има боље перформансе док су остале метрике идентичне. На основу добијених резултата, може се извести закључак да апликација задовољава високе захтеве корисника у погледу перформанси, као и да задовољава високе веб стандарде.




*Слика 45: Тест десктоп верзије*




*Слика 46: Тест мобилне верзије*

# 6. Закључак

Веб картографија има кратку али богату историју јер је уско повезана са развојем информационих технологија. Обиље бесплатних просторних података на вебу, као и обиље доступних веб технологија отвореног кода, допринело је рапидном развоју картографских апликација за којима постоји потреба готово у свим гранама привреде и у многим научним областима.

Главни медијум данашњице за умрежавање, рад, едукацију и научна истраживања је управо веб коме се приступа са рачунара, таблета и паметних телофона те је сасвим логично да су потребе грађанства и стручњака за конзумирањем нових картографских производа управо у складу са развојем хардвера којима се приступа вебу и доступних веб технологија.

Када је потребно донети суд о одређној технологији, онда је најбоље проверити зашто је та технологија постала популарна. У овом раду, фокус је био на *Shiny* веб фрејмворку који истовремено представља и методологију развоја реактивне веб апликације. Применом ове технологије, решен је један конкретан задатак који се односио на лабелирање парцела на којима постоје биофизичке промене које су уз помоћ ове веб картографске апликације детектоване. *Shiny* технологија на јединствен начин заокружује *R* екостистем јер омогућава креирање веб апликације у изворном *R* коду без неопходног познавања веб језика као што су *Javascript* и HTML и CSS који представљају веб стандарде. Неопходно је напоменути да уз лакоћу израде веб апликације још је лакше постављање исте на интернет. На овај начин, задовољена је масивна потреба корисника за поделом података путем веба и колаборацијом. Управо је ова једноставност оно што ову технологију чини популарном у *R* заједници која непрестано расте.

# 7. Литература


1. Handbook of Mathematical Geosciences, B.S. Daza Sagar, Qiuming Cheng, Frits Agteberg
2. https://www.safe.com/what-is/spatial-data/#spatial-data-graphics
3. Геовизуелизација и Веб картографија, Милан Килибарда, Драгутин Протић
4. Hands-On Dashboard Development with Shiny (https://englianhu.files.wordpress.com/2018/10/hand-on-dashboard-development-with-shiny.pdf)
5. https://leafletjs.com/reference-1.7.1.html
6. https://en.wikipedia.org/wiki/Leaflet_(software)
7. https://sentinel.esa.int/web/sentinel/missions/sentinel-2
8. Sentinel-2 - User Handbook
9. ESA, Copernicus - Sentinels Scientific Data Hub
10. Solutions Architect Handbook, Saurabh Shrivastava, Neelanjali Srivastav

# 8. Прилози

Код апликације:

```
library(raster)
library(shiny)
library(shinydashboard)
library(leaflet)
library(rgdal)
library(tidyverse)
library(rgeos)
library(rsconnect)



server <- function(input, output, session) {

  options(shiny.maxRequestSize=700*1024^2)


  map <- reactive({

    req(input$filemap)

    shpdf <- input$filemap


    tempdirname <- dirname(shpdf$datapath[1])


    # Rename files
    for (i in 1:nrow(shpdf)) {
      file.rename(
        shpdf$datapath[i],
        paste0(tempdirname, "/", shpdf$name[i])
      )
    }


    map <- readOGR(paste(tempdirname,
                         shpdf$name[grep(pattern = "*.shp$", shpdf$name)],
                         sep = "/"
```

```
  ))

  map <-  spTransform(map, CRS("+proj=longlat +datum=WGS84"))
  map

})

output$map <- renderLeaflet({

  if (is.null(map())) {

    return(NULL)

  }

  map <- map()
  centers <- data.frame(gCentroid(map, byid = TRUE))

  parcel_info <- as.data.frame(map)

  centers$region <- paste("Broj parcele: ",parcel_info$NUMBER,"(",parcel_info$AREA,"
ha )")


  l <- leaflet(map) %>%

    setView(view_extent@coords[1],view_extent@coords[2], 17) %>%

    addProviderTiles("Esri.WorldImagery") %>%
    #addTiles() %>%
    #setView(lat=10, lng=0, zoom=2) %>%
    addPolygons(data = map, color = "#444444", weight = 1, smoothFactor = 0.5,

                opacity = 1.0, fillOpacity = 0.5,

                fillColor = "green",

                highlightOptions = highlightOptions(color = "white", weight = 2,

                                                    bringToFront = TRUE),
                ) %>%
    addLabelOnlyMarkers(data = centers,

                        lng = ~x, lat = ~y, label = ~region,
```

```
                    labelOptions = labelOptions(noHide = TRUE, direction = 'top', textOnly =
TRUE,style = list(
                      "color" = "black",
                      "font-style" = "bold",
                      "font-size" = "20px",

                      "box-shadow" = "2px 2px rgba(0,0,0,0.25)",
                      "border-color" = "rgba(0,0,0,0.5)"
                    )))

  })
  observeEvent(input$file1,{
    chr = character()
    dates <- list()

    pth <- input$file1
    pth_sub <- pth$datapath[1]
    tmpdirname <- str_remove(pth_sub, "/0.tif")

    for(nr in 1:length(input$file1[, 1])){
      #str(input$file1)
      raster_name_tif <- input$file1[nr, 'name']
      chr[nr] <- input$file1[nr, 'datapath']
      raster_name <- str_remove(raster_name_tif, ".tif")
```

```
    dates[nr] <- raster_name

    tmpdirname <- gsub("\\\\", "/", tmpdirname)

    to_rename <- pth$datapath[nr]

    file.rename(

      to_rename,

      paste0(tmpdirname, "/", raster_name_tif)
    )

    datapath_raster <- paste(tmpdirname, "/", raster_name_tif, sep="", collapse=NULL)

    chr[nr] <- datapath_raster

  }


  lista_rastera <<- stack(chr)


  # Racunanje koordinata za leaflet mapu
  prostorni_obuhvat <- bbox(lista_rastera)

  x_sr <- (prostorni_obuhvat[1]+prostorni_obuhvat[3])/2

  y_sr <- (prostorni_obuhvat[2]+prostorni_obuhvat[4])/2

  # Kreiranje objekta SpatialPoints i dodeljivanje projekcije
  xy <- SpatialPoints(data.frame(x_sr ,y_sr))

  crs(xy) <- crs(lista_rastera)

  xy <- spTransform(xy, "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
#parametri projekcije

  view_extent <<- xy
  # Outputs
  output$raster_iz_liste <- renderPlot({

    plot(lista_rastera[[input$layer]], main = names(lista_rastera)[input$layer])
```

```
  })

  output$projekcija <- renderText(projection(lista_rastera))

  output$rezolucija <- renderText(res(lista_rastera))

  output$brojLejera <- renderText(nlayers(lista_rastera))

  nummax <- nlayers(lista_rastera)

  output$ispisKoord <- renderText(formatC(round(Selekcija_koordinata(),2), format = "f",
digits = 2))

  Selekcija_koordinata <- reactive({

    req(input$raster_klik$x)

    c(input$raster_klik$x, input$raster_klik$y)

  })


  output$tablicaPx <- renderTable({

    svojstvaRastera <-  data.frame("Layer" = names(lista_rastera), "Value" =
unlist(value())[1,])

    colnames(svojstvaRastera) <- c("Naziv rasterskog fajla","Vrednost piksela")

    rownames(svojstvaRastera) <- 1:nlayers(lista_rastera)

    svojstvaRastera

  })

  value <- eventReactive(input$raster_klik$x,{

    raster::extract(lista_rastera,cellFromXY(lista_rastera, Selekcija_koordinata()))

  })

  output$histo <- renderPlot({
```

```
    hist(lista_rastera[[input$layer]], ylab = "Frekvencija", xlab = "NDVI", col = 'palegreen1',
border = "black", lwd = 3, breaks = seq(0, 1, l = input$bin))

    })

  output$gustinaPlot <- renderPlot({

     plot(density(lista_rastera[[input$layer]]),  main = names(lista_rastera)[input$layer], xlab
= "NDVI", ylab ="Frekvencija")

     })


  output$vremenskaSerija <- renderPlot({

    req(input$raster_klik$x)

    plot(1:nlayers(lista_rastera), value(), type = "b", xlab="Vreme", ylab="Vrednost",
xaxt='n', ann=FALSE)

    axis(1, at=1:nlayers(lista_rastera), labels=dates)

  })

  output$downloadData <- downloadHandler(

    filename = function() {

      paste(input$outputFile, ".csv", sep = "")

    },

    content = function(file) {

      write.csv(df_fn_label, file, row.names = FALSE)

    }
  )



  observeEvent(input$avg_ndvi,{
```

```
    layermeans <- cellStats(lista_rastera[[input$layer]], stat='mean', na.rm=TRUE)

    sredina <- mean(layermeans)

    output$average_ndvi <- renderText({toString(sredina)})


  })

  count <- 1

  df_fn_label <- data.frame(fn = character(), file_name = character(), label = numeric())

  observeEvent(input$labela_0,{

    extract_fn <- filename(lista_rastera[[input$layer]])

    f_name  <- basename(extract_fn)

    df_fn_label <<- df_fn_label %>% add_row(fn = extract_fn, file_name = f_name, label =
0)


  })

  observeEvent(input$labela_1,{

    extract_fn <- filename(lista_rastera[[input$layer]])

    f_name  <- basename(extract_fn)

    df_fn_label <<- df_fn_label %>% add_row(fn = extract_fn, file_name = f_name, label =
1)


  })


  output$all_rasters <- renderUI({
    sliderInput(inputId = "layer", "Vremenska linija", min = 1, max = nummax, value = 1,
step = 1, width="100%", animate = TRUE)

  })
```

```
observeEvent(input$previousImage,{

  previous_value = input$layer - 1

  num_of_layers <- nlayers(lista_rastera)

  if (previous_value == 0 ) {
    previous_value <-  num_of_layers
  }


  output$raster_iz_liste <- renderPlot({
    updateSliderInput(session, inputId = "layer", value = previous_value)

    plot(lista_rastera[[input$layer]], main = names(lista_rastera)[input$layer])
  })


  layermeans <- cellStats(lista_rastera[[previous_value]], stat='mean', na.rm=TRUE)

  sredina <- mean(layermeans)

  sredina <- round(sredina, 2)

  output$average_ndvi <- renderText({toString(sredina)})


})

observeEvent(input$nextImage,{

  next_value = input$layer + 1


  num_of_layers <- nlayers(lista_rastera)

  if (next_value > num_of_layers ) {

    next_value <- 1

  }

  output$raster_iz_liste <- renderPlot({
```

```
      updateSliderInput(session, inputId = "layer", value = next_value)

      plot(lista_rastera[[input$layer]], main = names(lista_rastera)[input$layer])


    })

    layermeans <- cellStats(lista_rastera[[next_value]], stat='mean', na.rm=TRUE)

    sredina <- mean(layermeans)

    sredina <- round(sredina, 2)

    output$average_ndvi <- renderText({toString(sredina)})

    updateActionButton(session, inputId = "labela_0", "+")


  })

 })

}


ui <-  dashboardPage(

 skin = "green",

 dashboardHeader(

  title = "Obelezavanje promena na poljoprivrednim parcelama na osnovu biofizickih
parametara",
  titleWidth = "60%"

 ),
 dashboardSidebar(

  sidebarMenu(

    menuItem("Ulazni podaci", icon = icon("mouse"),


           fileInput("file1", "Ucitaj rastere",
```

```
                multiple = TRUE,
                accept = c("image/*"),
                buttonLabel = "Ucitaj...",
                placeholder = "Nema ucitanih rastera"
            ),

            fileInput("filemap", "Ucitaj poligone",
                multiple = TRUE,
                accept = c('.shp','.dbf','.sbn','.sbx','.shx',".prj"),
                buttonLabel = "Ucitaj...",
                placeholder = "Nema ucitanih poligona"
            )
    ),
    menuItem("Izlazni podaci", icon = icon("table"),
            textInput("outputFile", h6("Sacuvaj kao"),
                value = "parcela"),
            downloadButton("downloadData", "Preuzmi")
    )
  )
),

 dashboardBody(
  fluidRow(
    tabBox(
      title = "Selektuj piksel", id = "panel1",
```

```
  tabPanel("NDVI",
           uiOutput("all_rasters"),
           plotOutput("raster_iz_liste", click = "raster_klik")
           )
),
tabBox(
  title = "InfoPanel", id = "panel2",
  tabPanel("Histogram",
           checkboxInput(inputId = 'opcija', label = 'Gustina', value = FALSE),
           conditionalPanel(
             condition = 'input.opcija == true',
             plotOutput('gustinaPlot')
           ),
           conditionalPanel(
             condition = 'input.opcija == false',
             plotOutput('histo')
           ),
           sliderInput("bin", "Broj barova", min = 10, max = 100, value = 50)
  ),
  tabPanel("Svojstva rastera",
           tableOutput("tablicaPx")
  ),
```

```
    tabPanel("Metapodaci",
             HTML("<b>Broj ucitanih rastera:</b>"),
             textOutput("brojLejera"),
             HTML("<b>Rezolucija:</b>"),
             textOutput("rezolucija"),
             HTML("<b>Kartografska projekcija:</b>"),
             textOutput("projekcija"),
             h4("Selektovane koordinate"),
             textOutput("ispisKoord")
    )
  ),
  fluidRow(
    box(
      title = "Mapa", status = "info", solidHeader = TRUE, width = 8, leafletOutput("map")
    ),
    box(
      title = "Labeliranje", status = "warning", solidHeader = TRUE, width = 4,
      actionButton(inputId = "labela_0", "+"),
      actionButton(inputId = "labela_1", "-"),
      actionButton(inputId = "previousImage", "Prethodno"),
      actionButton(inputId = "nextImage", "Sledece"),
      h4("Prosecna vrednost NDVI: "),
      h4(textOutput("average_ndvi"))
```

```
      )

    ),
    fluidRow(

      box(

        title = "Vremenska serija", status = "success", solidHeader = TRUE, width= 12,

        plotOutput("vremenskaSerija")

      )
    )
  )
)




shinyApp(ui = ui, server = server)
```