



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ
НАУКА У НОВОМ САДУ




Никола Дакић

**Имплементација апликације
за предикцију цене лаптоп
рачунара**

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, 2019

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА ИЗРАДУ ЗАВРШНОГ (BACHELOR) РАДА	Датум:

(Податке уноси предметни наставник - ментор)

СТУДИЈСКИ ПРОГРАМ:	Софтверско инжењерство и информационе технологије
РУКОВОДИЛАЦ СТУДИЈСКОГ ПРОГРАМА:	Др Бранко Милосављевић

Студент:	Никола Дакић	Број	SW 59/2013
Област:	Примењене рачунарске науке и информатика		
Ментор:	Др Милан Сегединач, доцент		
НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА МАСТЕР РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА: - проблем – тема рада; - начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;			

НАСЛОВ ДИПЛОМСКОГ РАДА:

Имплементација апликације за предикцију цене лаптоп рачунара

ТЕКСТ ЗАДАТКА:

Реализовати веб апликацију за прикупљање, обраду, приказ и предикцију цене лаптоп рачунара. Deployment апликације имплементирати употребом Docker контејнера. Предвиђање цене имплементирати употребом вишеструке линеарне регресије и методе најближих суседа.

Руководилац студијског програма:	Ментор рада:

Примерак за: <input type="checkbox"/> - Студента; <input type="checkbox"/> - Ментора
--

САДРЖАЈ

1.	УВОД	1
2.	ТЕХНИКЕ И АЛАТИ.....	3
2.1	Maven	4
2.2	Spring радни оквир	5
2.3	Beautiful Soup	7
2.4	Selenium Webdriver	8
2.5	Psycopg	8
2.6	Docker	9
2.6.1	Docker архитектура	9
2.6.2	Containers vs VMs	10
2.6.3	Docker Images	11
2.6.4	Dockerfile датотека	11
2.6.5	Docker Container	12
2.6.6	Изградња и покретање Docker слике	13
2.7	Једнострука линеарна регресија	14
2.7.1	Модел	15
2.7.2	Фитовање модела	15
2.7.3	Функција грешке	16
2.7.4	Gradient descent	16
2.8	Вишеструка линеарна регресија	17
2.8.1	Модел	17
2.8.2	Евалуација модела	18
2.9	Метода најближих суседа	18
3.	СПЕЦИФИКАЦИЈА АПЛИКАЦИЈЕ	19
3.1	Модел података	19
3.2	Дијаграм активности апликације	20
3.3	Дијаграм класа	21
3.4	Модули	22
3.4.1	Модул за прикупљање података	23
3.4.2	Модул за обраду података	25
3.4.3	Модул за приказ података	26
3.4.4	Модул за предикцију цене	26
3.4.5	Deployment апликације	26
4.	ИМПЛЕМЕНТАЦИЈА	27
4.1	Имплементација модула за прикупљање подата.....	27
4.2	Имплементација модула за обраду података	33
4.3	Имплементација модула за приказ података	34
4.4	Имплементација модула за предикцију цене	36
4.5	Deployment	39
4.5.1	Deployment Angular апликације	39
4.5.2	Deployment Spring апликације	41
4.5.3	Deployment Django апликације	42

5. ВЕРИФИКАЦИЈА РЕШЕЊА	45
6. ЗАКЉУЧАК	46
7. ЛИТЕРАТУРА	47
8. БИОГРАФИЈА	49

1. Увод

У овом раду биће представљена апликација намењена за предикцију цене половних лаптоп рачунара. Апликација се бави прикупљањем, обрађивањем и анализирањем података о компонентама (спецификацијама) лаптопова.

Цена половног лаптопа се одређује на основу других прикупљених података о лаптоп рачунарима и њихових спецификација. Спецификације лаптоп рачунара формирају се парсирањем огласа који се преузимају са вебсајта Купиндо.

Апликација функционише тако што корисник одабира компоненте које га интересују, након чега се као резултат рада апликације добија предиктована цена лаптопа.

Апликација се састоји од 4 модула. Први модул служи за прикупљање података. Други модул служи за складиштење и обраду података. Трећи модул служи за приказ података. И четврти модул служи за предикцију цене лаптопа.

У склопу рада је приказано који подаци су прикупљани и на који начин. Затим су представљене методе машинског учења, које су коришћене за предвиђање цене лаптопа. Приказан је начин обраде података као и кориснички интерфејс апликације. И на крају је приказано како је имплементиран *deployment* (стављање апликације у рад) свих модула апликације.

Рад је подељен у шест поглавља. Прво поглавље је увод.

У другом поглављу представљене су технике и алати који су коришћени при имплементацији сваког појединачног модула апликације. Укратко је објашњено за шта служи сваки алат и како функционишу технике машинског учења које су примењене у овом раду.

У трећем поглављу је кроз класни дијаграм и дијаграм активности дата формална спецификација модела података и пословних процеса апликације. Такође у овом поглављу је објашњен сваки модул тако што је укратко наведено од чега се састоји и за шта је сваки модул задужен.

У четвртом поглављу представљена је конкретна имплементација сваког модула апликације. Наведене су технологије

које су коришћене при имплементацији модула, као и које су главне методе сваког модула.

У петом поглављу приказани су резултати апликације и извршена је валидација добијених резултата.

У шестом поглављу наведени су могући правци даљег истраживања.

2. Технике и алати

У овом поглављу биће описани алати, технике и технологије које су коришћене при имплементацији апликације која се описује у овом раду.

Апликација се састоји од 3 независне апликације и модула за прикупљање података. Манипулација прикупљених података је реализована уз ослонац на Maven и Spring радни оквир. Интерфејс и интеракција са апликацијом је реализована уз помоћ Angular апликације. Предвиђање цене лаптопа је реализовано уз ослонац на Django радни оквир.

Модул за прикупљање података се састоји од две Python скрипте (класе) које се периодично извршавају. Једна скрипта је задужена за прикупљање података а друга за њихово складиштење у базу података.

За прикупљање података са интернета коришћена је библиотека BeautifulSoup и Selenium радни оквир. Прикупљени подаци се складиште у PostgreSQL релациону базу података која се налази на AWS (Amazon Web Services) облаку (cloud).

За имплементацију ове апликације коришћене су следеће технологије. Свака од наведене технологије биће описана у засебној секцији:

Списак коришћених алата и библиотека:

- Maven build toolkit 4.0.0
- Spring framework 5.0
- PostgreSQL 10.6
- Angular 7.3.8.
- Django framework 2.2.1
- BeautifulSoup 4.6.0
- Selenium 3.13.0
- Psycopg 2.8.2
- Docker 19.03.1

За предвиђање цене лаптопа коришћене су две технике машинског учења:

- Вшеструка линеарна регресија (MLR) и
- Метода најближих суседа (KNN)

Deployment свих делова апликације имплементиран је уз ослонац на Docker.

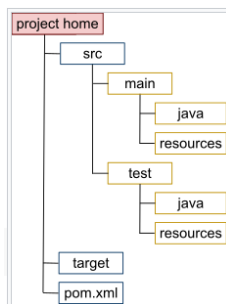
2.1 Maven

Maven је алат који служи за управљање софтверским пројектима и углавном су то Java пројекти. Maven build toolkit користи XML нотацију и бави се решавањем два аспекта израде софтвера. Први аспект који описује како се софтвер гради и други који описује које зависности користи. То подразумева да води рачуна о животном циклусу Java пројекта (компајлирању, покретању тестова, креирању `.jar` фајлова) као и да на основу дефинисаних зависности, динамички преузима неопходне библиотеке са удаљеног репозиторијума (Maven Central Repository) и чува их локално.

Животни циклус покретања апликације се састоји из неколико фаза. Стандардни животни циклус укључује следеће фазе и то оним редоследом којим су набројане [1]:

- валидација
- копирање ресурса
- компајлирање
- тестирање
- паковање
- интеграционо тестирање
- верификација
- инсталирање
- подизање апликације

Maven сваком пројекту додељује подразумевана подешавања пројекта, како би се касније лакше покретао. Оваква идеја се назива *Convention over Configuration*, што у слободном преводу значи да ако се испоштују одређене конвенције, нема потребе за додатним подешавањем пројекта. Због ове идеје, сваки Maven пројекат има јасно прописану хијерархију фолдера. На слици 1. је приказана хијерархија фолдера Maven пројекта.



Слика 1. Maven хијерархија фолдера

2.2 Spring радни оквир

Spring је радни оквир који служи за развој пословних апликација. Оквир омогућује развој и садржи инфраструктуру за једноставнији развој Јава ЕЕ апликација. Spring управља инфраструктуром тако да програмер може да се фокусира на реализацију доменских проблема уместо на технологију. Java Spring радни оквир је замишљен као лаган и неинвазиван оквир. Идеја оквира је да се програмски код апликације што мање прилагођава Spring API-ју, а то се реализује претежним коришћењем обичних (POJO) Java објеката, као и коришћењем анотација или конфигурационих фајлова којима се дефинише како Spring управља објектима.

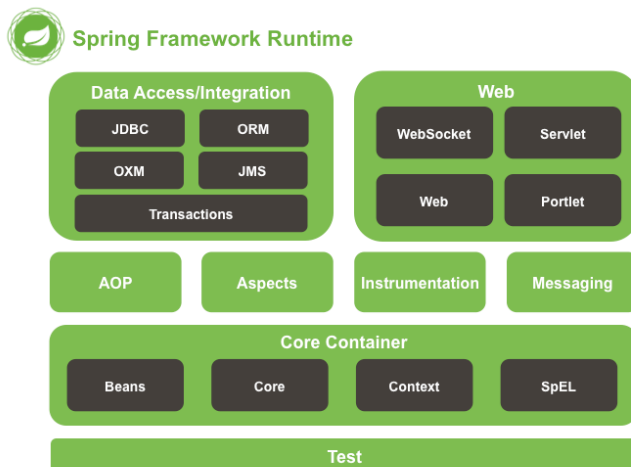
Један од главних концепата Spring радног оквира је повезивање објеката кроз Dependency Injection (DI) механизам. Овај механизам је битан јер се функционалности обично реализују заједничким радом међусобно повезаних објеката, те постоји потреба да Spring олакша тај посао и организује објекте у функционалну целину. Задаци којима се бави Spring су:

- Повезивање објеката
- Управљање животном циклусом објеката
- Додављање тражених објеката

Аспектно оријентисано програмирање (AOP) је још један битан концепт Spring радног оквира. AOP подразумева поновно искоришћење често коришћених функционалности. AOP се реализује тако што се одређена функционалност независно дефинише, а затим се као аспект примени у потребним деловима апликације.

Spring радни оквир се састоји од разних функционалности које су груписане у модуле [2]. Модули су груписани у следеће слојеве (слојеви се могу видети на слици 2):

- Core Container
- Data Access/Integration
- Web
- AOP (Aspect Oriented Programming)
- Aspects
- Instrumentation
- Test



Слика 2. Слојеви Spring радног оквира

Core Container је најважнији слој и садржи основне модуле за рад са Spring радним оквиром. Сви Spring модули се темеље на овом слоју. Core модул управља животним циклусом објеката и омогућује Dependency Injection механизам. Поред тога модул садржи application context који учитава конфигурацију апликације и све њене објекте.

Data Access/Integration слој обезбеђује JDBC (Java Database Connectivity) и уклања потребу за класичним приступањем базама података, коришћењем JDBC управљачких програма. Коришћење JDBC-а подразумева писање кода за прибављање конекције, формирање наредби (statements), обраду резултата и затварање конекције, док се коришћењем JDBC оквира целокупан посао апстрактује и поједностављује. ORM модул обезбеђује слој за интеграцију Spring оквира са популарним ORM алатима, као што су: Hibernate, JPA, JDO, iBatis, TopLink.

Web модул је модул који садржи комплетну имплементацију MVC оквира који се користи за развој Web апликација. Spring Web MVC обезбеђује потпуно раздвајање презентационог слоја од пословне логике апликације, уз коришћење свих особина оквира приликом развоја Web апликације. Поред основне имплементације MVC оквира, Spring нуди подршку и за интеграције са другим MVC оквирима.

Messaging и Instrumentation модули омогућују размену порука између делова апликације као и прилагођавање рада JVM (садржи измењене начине учитавања класа).

AOP модул пружа подршку за аспектно оријентисано програмирање које програмерима омогућава да на једноставан начин раздвоје компоненте система, имплементирајући функционалности, које су логички одвојене, засебно.

Тест модул пружа подршку за јединично и интеграционо тестирање Spring апликације коришћењем Junit и TestNG библиотека

Поред самог Spring радног оквира развијен је велики број других оквира и библиотека које се користе у оквиру Spring апликације, неки од њих су:

- *Spring Security* који омогућује једноставну имплементацију контроле приступа у Spring апликацијама
- *Spring Data* који омогућује једноставно управљање базама података
- *Spring Boot* који поједностављује, коришћење самог Spring радног оквира тако што ослобађа програмера значајног дела конфигурисања Spring-а и писања стандардних делова апликације.

2.3 Beautiful Soup

Beautiful Soup (BS) је Python библиотека за парсирање и извлачење података из HTML и XML фајлова. BS парсира HTML или XML у DOM стабло, као што то раде модерни веб претраживачи.

Пример употребе BS библиотеке:

- прибављање и парсирање HTML садржаја са одређеног url-а
- проналажење и извлачење података, коришћењем DOM стабла или CSS селектора
- манипулација HTML елемената, атрибута и текста

Тренутна верзија ове библиотеке у тренутку писања рада је Beautiful Soup 4.4.0. Ради подједнако за Python 2.7 и 3.2.

2.4 Selenium WebDriver

Selenium је радни оквир (framework) који служи за тестирање веб апликација.

Састоји се неколико компоненати од којих свака има своју специфичну улогу. Једна од најчешће коришћених компоненти је Selenium WebDriver.

Ова компонента омогућава да се прочита одређена веб страница уз помоћ жељеног претраживача и да се накнадно парсира. Овакав начин читавања страница је веома згодан код прикупљања података јер омогућава да се изврши и JavaScript програмски код вебсајта, а то омогућава да се веб страница парсира у потпуности.

WebDriver-у је могуће проследити широк спектар подешавања која говоре како веб претраживач да се понаша. Нека од тих подешавања су коришћење претраживача без корисничог интерфејса (headless browser) или рецимо постављање проксија (проху) уколико постоји потреба за тим.

Selenium WebDriver функционише тако што прима жељени *url* а затим уз помоћ жељеног претраживача отвара вебстраницу, прави од ње DOM стабло и враћа је на коришћење. Тако добијено стабло странице се користи за извлачење података од интереса,

Webdriver има своје уграђене методе за парсирање података, али се може користити и у комбинацији са Beautiful soup библиотеком.

2.5 Psycopg

Psycopg је најпопуланији адаптер за PostgreSQL базу података за програмски језик Python. Његова главна предност је комплетна имплементација Python DB API 2.0 спецификације и безбедно коришћење нити (више нити могу да деле исту конекцију).

Адаптер је дизајниран да подржи више-нитне апликације које користе велики број конекција и извршавају велики број INSERT и UPDATE упита.

2.6 Docker

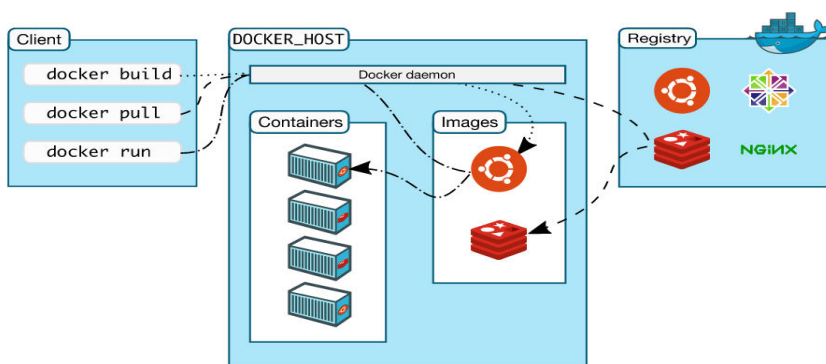
Docker је платформа отвореног кода (open source) за развој, дистрибуцију и покретање апликација. Docker омогућава раздвајање апликације од њене архитектуре што омогућава бржу испоруку софтвера. Docker служи за лакше креирање, објављивање (deploy) и покретање апликација, употребом контејнера (containers). Docker контејнер је покретљива инстанца слике. Docker контејнер омогућава паковање свих делова апликације и свих њених зависности, а тиме се омогућава лако и конзистентно покретање апликација на различитим системима.

Docker је доступан у два издања:

- Community Edition (CE) - предвиђено за тестирање Docker-a
- Enterprise Edition (EE) - предвиђено за тимове који испоручују пословно оријентисане апликације

2.6.1 Docker архитектура

Docker користи клијентско-серверску архитектуру. Docker клијент комуницира са Docker механизмом (*daemon*) који обавља послове изградње, покретања и дистрибуције Docker контејнера. Docker клијент и механизам се могу покретати на истом систему а могуће је и повезивање клијента са удаљеним Docker механизмом. Docker клијент и механизам комуницирају користећи REST API, преко UNIX прикључка или мрежног интерфејса [5].



Слика 3. Docker архитектура

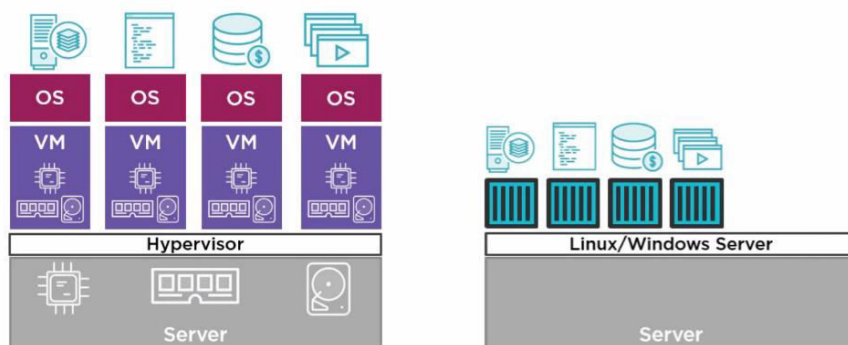
Docker архитектура се састоји од више компоненти:

- Docker механизам - слуша Docker API захтеве и управља Docker објектима као што су слике, контејнери и мреже.
- Docker клјент - примарни начин за интеракцију корисника са Docker-ом.
- Docker регистри - место где се чувају Docker слике (images).

2.6.2 Containers vs VMs

Docker контејнер за разлику од виртуалне машине (VM) омогућава апликацијама да користе исти Linux Kernel који користи и host систем апликације, те на тај начин ефикасније користи ресурсе оперативног система (OS).

На слици 4 се може видети зашто је Docker који се базира на Container моделу ефикаснији.



Слика 4. VM vs Container модел

Код VM модела свака виртуална машина има свој RAM (Random-Access Memory), процесор, меморију и свој оперативни систем.

Код Container модела постоји један физички сервер са једним оперативним системом, што доводи до ефикаснијег коришћења ресурса.

2.6.3 Docker Images

Docker слика је фајл са инструкцијама које служе за креирање Docker контејнера. Често, Docker слика је базирана на неког другој Docker слици са додатним подешавањима. Пример тога је рецимо Docker слика која се базира на *ubuntu* слици, али поред тога садржи и Apache веб сервер као и саму апликацију. Свака Docker слика се састоји од слојева. Слој представља промену на слици.

Docker слику је могуће самостално направити али могу се користити и друге Docker слике које се налазе на Docker Hub складишту (repository). Docker Hub је cloud базирано складиште за чување Docker слика које могу бити приватно или јавно доступне.

Да би се направила Docker слика, потребно је креирати Dockerfile датотеку са једноставном синтаксом која дефинише кораке неопходне за креирање слике. Свака инструкција у Dockerfile-у креира слојеве слике, па када се промени Dockerfile датотека и обнавља (rebuild) Docker слика, само слојеви који су измењени се поново обнављају.

2.6.4 Dockerfile датотека

Dockerfile је текстуални фајл који садржи све команде и инструкције неопходне за креирање Docker слике. Састоји се од команди и аргумената уз помоћ којих се формира Docker слика.

На слици 5 се може видети пример Dockerfile датотеке.

```
FROM python:3
RUN mkdir /code
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/

EXPOSE 9000
CMD ["python3", "manage.py", "runserver", "0.0.0.0:9000"]
```

Слика 5. Dockerfile за покретање Django апликације

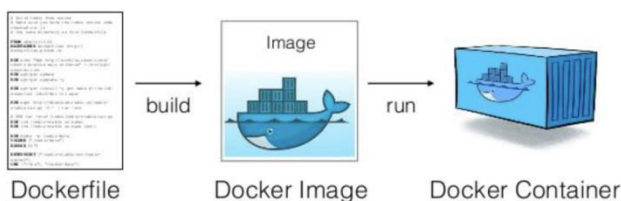
Свака инструкција у Dockerfile-у представља један слој. Фајл се дефинише тако што се прво наведе Docker кључна реч, након које иде жељена инструкција. Конкретно у претходном примеру коришћене су следеће кључне речи:

- FROM - кључна реч која дефинише основну слику на којој ће се базирати апликација.
- RUN - кључна реч која се користи за извршавање команди приликом изградње Docker слике. Dockerfile датотека може да садржи више RUN команди.
- WORKDIR - кључна реч којом се поставља радни директоријум у којем ће се извршавати све будуће команде.
- COPY - кључна реч која служи за потребе копирања садржаја.
- EXPOSE - кључна реч која служи као документација на ком порту контејнер слуша.
- CMD - кључна реч која служи за извршавање инструкције која покреће апликацију.

Поред претходно наведених кључних речи (команди), постоје и друге команде као нпр: ADD, ENV, VOLUME и ENTRYPOINT. Комплетан списак команди и њихов начин коришћења се може пронаћи на званичној веб страници Docker-а [5].

2.6.5 Docker Container

Као што је већ речено, Docker омогућава паковање свих делова апликације и свих њених зависности у контејнер. Када је апликација конфигурисана уз помоћ Docker-а, тј. када је дефинисана Dockerfile датотека уз помоћ које се прави (build) Docker слика, покретањем (run) те слике добија се Docker контејнер који представља изоловано окружење, видети слику 6.



Слика 6. Фазе креирања Docker контејнера

Употребом Docker контејнера, олакшавају се проблеми покретања апликација на различитим система.

Поред те чињенице треба посебно напоменути да је покретање апликације (употребом Docker-а), на локалном рачунару, релативно лако, и да прави изазов заправо лежи у скалирању real-world апликација, које се састоје од много Docker контејнера, и које могу бити поприлично комплексне. Али пошто та тема превазилази опсег овог рада, биће споменуто само да Docker има свој уграђен модул за управљање контејнерима који се зове Docker Swarm, а да се тренутно најпопуларнији систем за управљање контејнерима зове Kubernetes.

2.6.6 Изградња и покретање Docker слике

Након креирања Dockerfile датотеке следи изградња Docker слике. Docker слику правимо помоћу Docker командног интерфејса са командом `docker build -f Dockerfile -t prediction-api..`

```
Sending build context to Docker daemon 203.3kB
Step 1/8 : FROM python:3
--> a4cc999cf2aa
Step 2/8 : RUN apt-get update
--> Using cache
--> c604074920b7
Step 3/8 : WORKDIR /usr/src/app
--> Using cache
--> 906f6a34cc84
Step 4/8 : COPY requirements.txt ./
--> de19342d26fb
Step 5/8 : RUN pip3 install -r requirements.txt
--> Running in 980d14901947
Removing intermediate container 980d14901947
--> 30bd563a52cb
Step 6/8 : COPY . .
--> 34620a33a962
Step 7/8 : EXPOSE 8000
--> Running in 31922d512648
Removing intermediate container 31922d512648
--> 389f3f6ae907
Step 8/8 : CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
--> Running in 322918a064ad
Removing intermediate container 322918a064ad
--> a645f7c62bca
Successfully built a645f7c62bca
Successfully tagged prediction-api:latest
```

Слика 7. Испис конзоле након изградње слике

На слици 7 се може видети испис командне линије након изградње Docker слике. Приказана изградња слике се састоји од 8 корака (steps) који су дефинисани унутар Dockerfile датотеке. Свака команда мора бити успешно извршена како би се Docker слика могла изградити. Након успешне изградње Docker слике, слика нам је доступна. Преглед свих Docker слика и њихових информација се може видети извршавањем команде `docker images`.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
prediction-api	latest	a645f7c62bca	9 minutes ago	1.37GB
client-app	latest	e580d93066bd	19 minutes ago	1.37GB
server-app	latest	fba4188b1bca	20 minutes ago	660MB

Слика 8. Приказ свих Docker слика

Након успешне изградње Docker слике, командом `docker run -d --name prediction-api prediction-api` покрећемо Docker слику и креирамо Docker контејнер. Приказ свих активних и неактивних Docker контејнера и њихових информација може се видети извршавањем команде `docker ps -a`.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
25047d88cec1	server-daka-api:latest	"java -jar original-..."	2 seconds ago	Up 1 second	9005/tcp	server-app
ef8682217633	client-daka-app:latest	"/bin/sh -c 'npm ins...'"	13 seconds ago	Up 12 seconds	4200/tcp	client-app
ef93424e5db3	prediction-api	"python3 manage.py r..."	About a minute ago	Up 38 seconds	8000/tcp	prediction-api

Слика 9. Приказ свих Docker контејнера

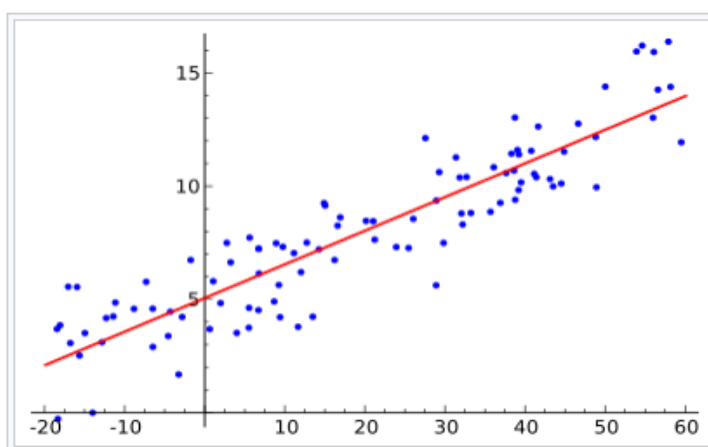
2.7 Једнострука линеарна регресија

Једнострука линеарна регресија је техника која омогућава да се процени једна променљива на основу неке друге. Претпоставља се да на циљну варијаблу у утиче само једна улазна варијабла $y=h(x)$. Такође претпоставка линеарне регресије јесте да је зависност циљне у варијабле од улазне варијабле x линеарна.

Иако је овај модел веома једноставан, линеарна регресија је изузетно корисна и концептуално и практично [3].

2.7.1 Модел

Функција хипотезе за линеарну регресију је: $h(x) = \theta_0 + \theta_1 x$
Приликом рачунања, потребно је одредити непознате параметре θ_0 и θ_1 , на такав начин да се права коју добијамо на најбољи начин уклапа у податке које имамо. Пример праве која се најбоље уклапа у податке се може видети на слици 10.



Слика 10. Регресиона права

Параметар θ_0 представља пресек (intercept) а параметар θ_1 нагиб (slope) праве.

2.7.2 Фитовање модела

Правна која се добро уклапа у податке се одређује тако што се уводи термин "цена коштања" (Cost function) праве. Цена је изражена као RSS (Residual Sum of Squares) и она представља разлику између тачне вредности променљиве Y и вредности P коју би предвидео модел у некој тачки X , тј. представља разлику између стварне и предвиђене цене. Циљ задатака је одредити параметре модела за које је цена праве најмања.

Цена изражена као RSS се рачуна уз помоћ формуле:

$$\min \frac{1}{2N} \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$(x^{(i)}, y^{(i)})$ - представља i -ту опсервацију

N - представља број опсервација

y - представља тачну вредност циљне варијабле

p - представља предикцију модела

Цену праве се минимизује због једноставности рачунања извода функције и ради смањења утицаја outliers-а тј. података који знатно одступају од већине.

2.7.3 Функција грешке

Функција грешке представља резултате свих цена.

$$J(\theta) = \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Права која се најбоље уклапа у податке добија се тако што се одреди минимум функције грешке.

2.7.4 Gradient descent

Gradient descent (GD) је оптимизациона техника за рачунање минимума функције.

GD се користи за одређивање вредности коефицијената θ_0 и θ_1 , тако што се изаберу неке вредности θ_0 и θ_1 и итеративно се коригују те вредности док се не добије најнижа цена.

Главни елементи ове технике су:

- Одабир почетног решења
- Одабир величине корака α
- Провера конвергенције
- Скалирање (нормализација) обележја

Тачке великог утицаја су тачке које негативно утичу на модел и њих је потребно уклонити како би се модел боље уклапао у податке. Кандидати за тачке од великог утицаја су екстремно велике или екстремно мале вредности x .

2.8 Вишеструка линеарна регресија

Код вишеструке линеарне регресије, две или више независних променљивих (x_1, x_2, x_3) се користе како би се предвидела зависна у променљива. Претпоставља се да на циљну варијаблу у утиче више улазних варијабли $y=h(x_1, \dots, x_D)$ [3].

2.8.1 Модел

Функција хипотезе за вишеструку линеарну регресију је:

$$h(x) = \sum_{d=0}^D \theta_d f_d(x)$$

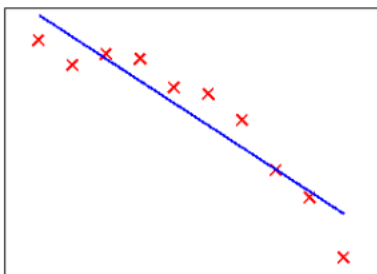
D - представља број обележја скупа података

$f_j(x)$ - представља обележје j

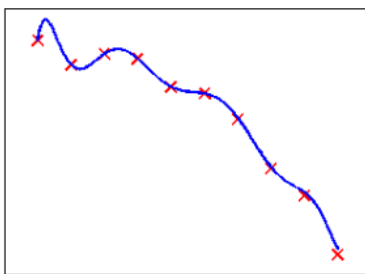
Повећањем броја улазних варијабли тј. броја обележја, повећава се комплексност модела.

Модели ниске комплексности могу бити недовољно флексибилни да опишу податке тј. могу имати велико системско одступање (bias) а малу варијансу. Насупрот томе, комплекснији модели су флексибилнији али могу да пате од overfitting -а тј. варијанса им може бити велика а системско одступање мало.

На слици 11. се може видети пример модела ниске комплексности а на слици 12. пример модела превелике комплексности [3].



Слика 11. Пример великог одступања (high bias)



Слика 12. Пример малог одступања (high variance)

Стога адекватан избор обележја је од велике важности за добре перформансе алгорита.

2.8.2. Евалуација модела

Модел се евалуира рачунањем грешке модела тј. рачунањем RMSE (Root Mean Square Error) на тест подацима (скупу).

Претходно је потребно податке расподелити на тренинг и тест скуп и то се обично ради у односу 90/10 или 80/20.

RMSE се не рачуна на тренинг скупу јер би се добијени модел могао превише прилагодити тренинг подацима, што би могло да узрокује лошу генерализацију тј. модел би вршио лоше предвиђање за податке које није видео приликом одређивања модела.

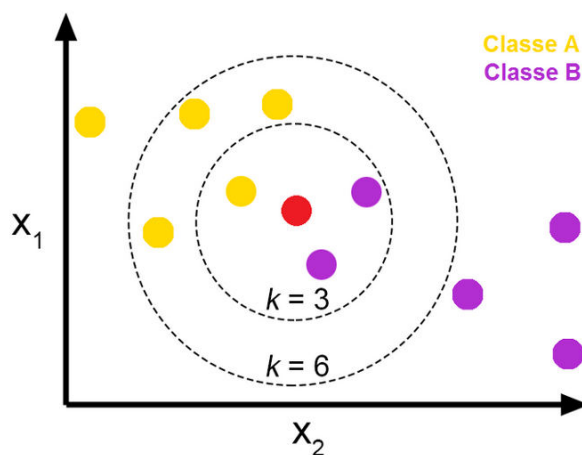
RMSE представља просечну грешку израчунату на тест скупу и рачуна се по формули:

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^n (y^{(i)} - h_{\theta}(x^{(i)}))^2}$$

2.9 Метода најближих суседа (KNN)

KNN (K-Nearest Neighbors) или ти метода најближих суседа је метода која функционише тако што KNN алгоритам пролази кроз цео тренинг сет и налази K најсличних инстанци.

На слици 13 се може видети како KNN алгоритам обухвата податке [4].



Слика 13. Пример KNN са 3 и 6 суседа

KNN врши предикцију користећи податке директно, не постоји никакав модел нити параметри.

Предикција се израчунава директно рачунајући сличност између улазног параметра и сваког тренинг податка, најчешће као средња вредност к сличних инстанци. Сличност је заправо дистанца између две тачке и постоји више начина да се она израчуна, рецимо коришћењем Еуклидске удаљености (сума разлике квадрата) у једнодимензионалном простору, али постоје и друге функције као нпр. Manhattan и Hamming.

KNN добро ради за мали број улазних параметара (D), док за велик број улазних параметара има лоше перформансе. Разлог зашто KNN има лоше перформансе са великим бројем улазних параметара је проклетство димензионалности тј. најближи суседи могу бити далеко када имамо велики број димензија.

3 Спецификација апликације

Софтверска архитектура подразумева фундаменталну структуру софтверског система и дисциплину креирања такве структуре и система. Свака структура обухвата софтверске елементе, везе између њих, као и њихова својства. Архитектура софтверског система је метафора аналогна архитектури зграда.

Софтверска архитектура подразумева прављење фундаменталних структуралних одлука које је скупо накнадно променити када се имплементирају.

У овом поглављу биће описана архитектура апликације.

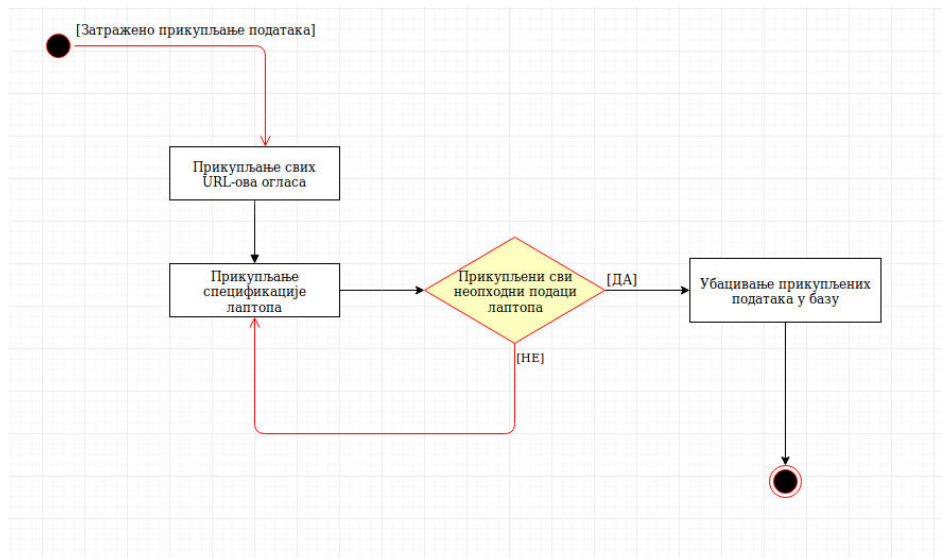
3.1 Модел података

За моделовање различитих апликација користи се UML језик за моделовање. UML (Unified Modeling Language) језик помаже при изради спецификације као и при визуелизацији, конструкцији и документовању производа софтверског система. UML представља скуп практично примењених инжињерских модела, који се користе у развоју великих и сложених система. UML је постао стандардни језик моделовања, између осталог, и због чињенице да је независан од програмских језика, Подржава богат скуп графичких елемената. Расположиви графички елементи омогућавају да се на разумљив начин опишу: класе, компоненте, чворови, активности, случајеви коришћења, објекти, стања и слично, као и да се моделују релације између наведених елемената [6].

3.2 Дијаграм активности апликације

Дијаграми активности су намењени моделирању динамичких аспеката (понашања) система. Приказују понашање користећи моделе тока контроле и тока података. Такође приказују секвенцијалне и конкурентне кораке у процесу обраде [6]. На слици 10 је приказан дијаграм активности везан за прикупљање података.

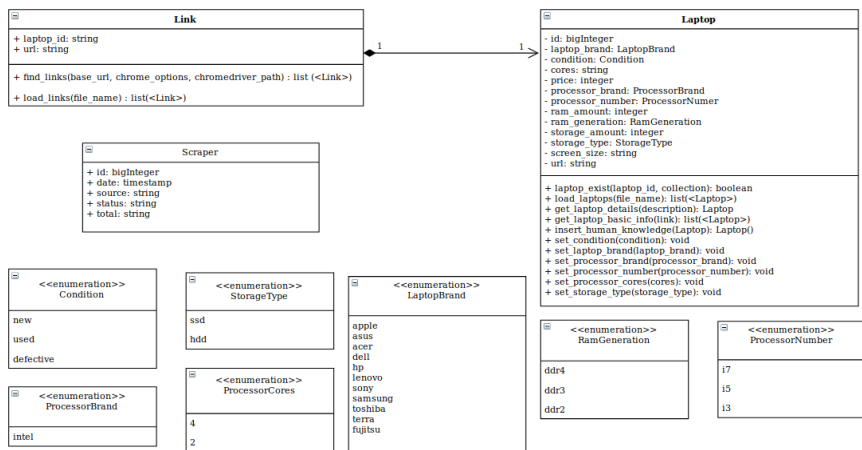
Активност прикупљања податак започиње тако што корисник покреће (затражи) скрипту која се бави прикупљањем података о лаптоп рачунарима. Скрипта прво прикупља све URL-ове огласа, након чега прелази на прикупљање појединачиних података о сваком лаптоп рачунару. Када су подаци прикупљени, скрипта провера да ли су прикупљени сви неопходни подаци, и уколико јесу, подаци се убацују у базу података. Уколико нису прикупљени сви подаци о лаптоп рачунару, подаци тог лаптопа се одбацују као непотпуни.



Слика 14. Дијаграм активности прикупљања података

3.3 Дијаграм класа

Дијаграм класа приказује скуп класа, интерфејса, сарадњи, шаблона, пакета и других елемената, повезаних релацијама. Представља граф направљен од темена (елемената) повезаних гранама (релацијама). Специфицира логичке и статичке аспекте модела. На основу њега је могуће изгенерисати скелет кода који се користи за имплементацију програма [6]. Апликација представљена у овом раду има три модела (са њиховим еnumerацијама) која се могу видети на слици 15.

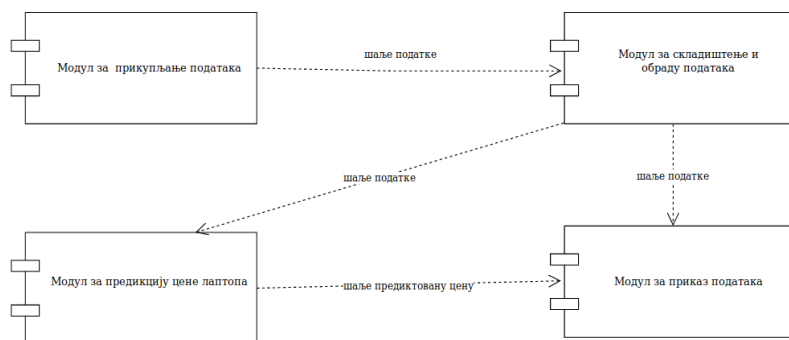


Слика 15. Дијаграм класа модела података

3.4 Модули

Модуларност апликације доводи до смањења зависност између делова апликације. Сваки модул се може посматрати као засебна апликација. Промене у једном модулу, неће проузроковати промене у неком другом. За реализацију пројекта користе се 4 модула:

- модул за прикупљање података
- модул за обраду података
- модул за приказ података
- модул за предикцију цене



Слика 16. Модули апликације

На слици 16. приказани су модули апликације и како су они међусобно повезани.

3.4.1 Модул за прикупљање података

У овој секцији биће описано како тачно функционише модул за прикупљање података. Циљ овог модула је прикупљање спецификација за што више лаптопова.

Прикупљање података обавља се уз помоћ Selenium Webdriver-а и BeautifulSoup библиотеке. Selenium Webdriver омогућава читавање комплетне веб странице коришћењем ChromeDriver-a, а затим прослеђује ту страницу BeautifulSoup библиотеци уз помоћ које се врши њено парсирање.

Подаци се прикупљају са вебсајта Купиндо и прикупљање започиње тако што се учита следећа вебстраница https://www.kupindo.com/Racunari-i-oprema/Laptop-racunari-delovi-i-oprema/Laptopovi/artikli/1473/cena_DESC_strana_1. На тој страници се налази списак огласа лаптопова, опадајуће сортирани по њиховој цени, што се може видети и по параметрима URL-а. Пошто су огласи подељени по странама, прво што се ради је утврђивање колико огласа односно страница тренутно постоји.

Затим након тога се извлаче URL-ови свих огласа по странама, док се не дође до последње стране. Сви URL-ови који представљају јединствену адресу огласа се чувају у CSV (comma-separated values) фајлу, заједно са јединственом (ID) вредношћу огласа.

Када су сви URL-ови активних огласа прикупљени, прелази се на појединачно читавање и парсирање огласа.

Пример како изгледа један оглас лаптопа се може видети на слици број 17.



LENOVO laptop - NOVO, NEKORISCENO

Cena: 78.990 din kupi odmah

ubaci u privatnu listu želja
ubaci u 'kupi mi' listu želja

Stanje: Nekorišćen
 Garancija: Ne
 Isporuka: Post Express
 Plaćanje: Tekuci račun (pre slanja)
 Grad: Kragujevac, Kragujevac-Stari Grad

107 pregleda
besplatna dostava

Dijagonala ekrana: **15.6"**
 Proizvođač: **Lenovo**

LENOVO laptop - NOVO, NEKORISCENO - Intel® Core™ i5-7200U, 2.50 GHz - 3.10GHz/AMD Radeon 530 sa 2GB/8GB/256 GB SSD

Karakteristike:

Ekran: 15.6" FullHD, AntiGlare
 Procesor: Intel® Core™ i5-7200U, 2.50 GHz - 3.10GHz
 Grafička kartica: AMD Radeon 530 sa 2GB
 Memorija: 8GB DDR4
 Hard disk: 256GB SSD
 Optički uređaj: DVD±RW DL
 Wi-Fi: WiFi 802.11ac
 Bluetooth: Bluetooth 4.1
 Mrežna karta: 1x 10/100/1000 Ethernet
 Web kamera: 0.3Mpix
 HDMI: HDMI x 1
 USB 2.0: USB 2.0 x 1
 Čitač memorijskih kartica: 4 u 1 čitač kartica

Слика 17. Пример огласа

На основу слике се може закључити да постоје неки основни подаци који су фиксни и које је могуће релативно лако парсирати, као нпр: цена, стање, дијагонала екрана или произвођач лаптопа. Међутим све остале податке лаптопа је неопходно парсирати из описа огласа. То представља изазовнији део модула јер не постоји јасна структура описа огласа, тј. сваки корисник може на различит начин да опише свој лаптоп.

Парсирање описа је урађено коришћењем регуларних израза, и то на такав начин да се прво тражи одређена реч у огласу (нрп. SSD) након које се претпостави (и валидира регуларним изразом) да иде

тачна спецификација лаптопа, пошто је то типичан начин описа спецификације лаптопа.

Спецификација лаптопа се састоји од следећих компоненти:

- бренд лаптопа
- бренд процесора
- модел процесора
- број језгара процесора
- генерација RAM меморије
- количина RAM меморије
- тип екстерне меморије
- количина екстерне меморије
- величина екрана
- стање лаптопа
- цена лаптопа

Само уколико се извуку све набројане спецификације лаптопа, парсирање огласа се сматра успешним, и лаптоп са његовим компонентама се чува у фајл.

Треба напоменути да су уведена одређена ограничења парсирања ради смањења комплексности апликације, те се парсирају само они лаптопови који имају *Intel* процесор. Такође приликом парсирања уграђено је одређено људско знање ради повећања броја прикупљених лаптопова. Конкретно, на основу цене лаптопа и величине екстерне меморије могуће је претпоставити о којем типу екстерне меморије се ради као и тачном број језгара процесора.

Када су прикупљене и сачуване у фајл спецификације свих лаптопова, покреће се посебна скрипта која учитава све лаптопове и серијски их убацује (уз помоћ *psycopg2* библиотеке) у удаљену PostgreSQL базу података, коју на располагање даје AWS (Amazon Web Services).

У зависности од успешности убацивања података у базу података, убацују се такође и информације о статусу убацивања, укупном броју прикупљених, извору као и тачном датуму прикупљања података.

3.4.2 Модул за обраду података

Модул за обраду података се бави учитавањем података из базе података и њиховим обрађивањем. Модул за обраду података омогућава модулу за приказ података да прикаже жељене податке.

Конкретно овај модул омогућава приказ свих лаптопова, затим омогућава њихову претрагу на основу бренда лаптопа, као и приказ

информација о успешности скрејповања и убацивања података у базу података.

Такође, пошто је апликација имплементирана на такав начин да је неопходно за предвиђање цене лаптопа изабрати његове спецификације, овај модул даје на коришћење све опције које је могуће одабрати за сваку појединачну спецификацију лаптопа.

3.4.3 Модул за приказ података

Модул за приказ података приказује резултате рада претходних модула. Омогућава приказ и филтрирање свих прикупљених података о лаптоп рачунара, као и приказ података који указују на успешност прикупљања података.

Такође овај модул омогућава кориснику да одабере спецификације свог лаптоп рачунара и да графички види резултате предвиђања.

3.4.4 Модул за предикцију цене

Овај модул се бави суштином апликације која је представљена у овом раду, а то је предвиђањем цене лаптопа. Модул предвиђа податке на основу свих прикупљених података и на основу одабране спецификације лаптопа. Користе се две технике машинског учења, које функционишу на различит начин. Резултат рада ових метода је предиктована цена лаптопа.

3.4.5. Deployment апликације

Сви претходно описани модули су имплементирани као засебне апликације уз помоћ различитих технологија и алата. Тиме је постигнута модуларност апликације, која олакшава њен развој и одржавање.

Независност делова апликација изискује посебан deployment (пуштање апликације у рад) сваког модула.

Deployment апликација имплементиран је употребом Docker-а тако што је за сваку апликацију написан адекватан Dockerfile којим је описано како се апликација покреће.

4 Имплементација

У овом делу рада биће детаљно објашњена имплементација сваког појединачног модула, као и пример употребе апликације уз поређење добијених резултата. Такође биће објашњено и како је реализован deployment сваке појединачне апликације.

4.1 Имплементација модула за прикупљање података

За имплементацију овог модула коришћен је програмски језик Python уз ослонац на Selenium радни оквир, BeautifulSoup библиотеку и Psycorg адаптер.

Метода `get_links` је имплементирана са циљем прикупљања свих линкова. Метода започиње дефинисањем Selenium Chrome Webdriver-а и постављањем времена (`set_page_timeout`) чекања да се учита жељена страница. Након тога, пошто се сви огласи налазе на истој страници, одређује се тачан број странице до које је могуће ићи приликом листања огласа. Затим се уз помоћ BeautifulSoup билбиотеке креће у парсирање свих линкова на тренутној страници, док се не дође до последње. Сви линкови заједно са јединственом вредношћу (ID) огласа се убацују у *links* листу, која се накнадно чува у CSV фајл.

Поред кључне функционалности исписују се и поруке које индикују цео процес прикупљања линкова, како би се парсирање могло пратити.

У наставку овог поглавља, после сваког описа, следиће листинг методе.

```

def get_links(base_url):

    print "Collectiong urls started.."

    driver = webdriver.Chrome(executable_path=root_path + "scraper/chromedriver", chrome_options=get_chrome_options())
    driver.set_page_load_timeout(30)

    pagination_limit = "1"

    driver.get(base_url + pagination_limit)
    bs_page_limit_page = BeautifulSoup(driver.page_source, 'html.parser')

    pagination_limit = bs_page_limit_page.find('div', attrs={'class': 'holder_pagination'}).find_all('a')[1].text

    print "Total pages: %d" % int(pagination_limit)

    limit = 0

    for page_number in range(1, int(pagination_limit)):

        if limit <= int(pagination_limit):

            driver.get(base_url + str(page_number))

            bs_page_urls = BeautifulSoup(driver.page_source, 'html.parser')

            laptops_container = bs_page_urls.find_all('div', attrs={'class': 'product'})

            for laptop_item in laptops_container:

                link = laptop_item.find('a', attrs={'class': 'item_link'})['href']

                get_id_regex = re.search(r'(/Laptopovi/)\d+.', link)
                if get_id_regex:
                    laptop_id = re.search(r'\d+.', get_id_regex.group()).group().split("_")[0]

                    if check_does_exist(laptop_id, "link") == False:
                        links.append(Link(laptop_id, link))

            limit += 1

    print "Total links found: %d" % len(links)

```

Листинг 1. метода get_links

Метода `get_laptop_info` је имплементирана са циљем парсирања огласа и формирања спецификације лаптопа.

Метода започиње дефинисањем Selenium Chrome Webdriver-а и постављањем времена (`set_page_timeout`) чекања да се учита жељена страница. У наставку методе, поред провере да ли је линк већ обрађен, метода најпре парсира фиксне податке који се налазе на сваком огласу, а затим уз помоћ `get_details` методе парсира опис огласа и извлачи све остале податке. Прикупљени подаци се убацују у *laptops* листу која се накнадно чува у CSV фајл.

Пошто власници огласа неретко не напишу све информације лаптопа, ради што већег броја прикупљених података, уведена је метода `insert_human_knowledge` која претпоставља одређене спецификације.

```

def get_laptop_info():

    print "Getting laptops info started.."

    driver = webdriver.Chrome(executable_path=root_path + "scraper/chromedriver", chrome_options=get_chrome_options())
    driver.set_page_load_timeout(30)

    count = 0

    for link in links:
        if check_does_exist(link.laptop_id, "laptop") == False:
            try:
                time.sleep(1)

                driver.get(link.url)
                laptop_page = BeautifulSoup(driver.page_source, 'html.parser')

                # get latop price
                price = ""
                price_search = re.search(r'\d+[\.]\d+', laptop_page.find("span", attrs={"class": "input-group-addon"}).text)
                if price_search:
                    print price
                    price = int(price_search.group()).replace(".", "")

                # get laptop condition
                condition = ""
                condition_search = laptop_page.find("table", attrs={"class": "table table-predmet table-predmet-info table-clear"}).find_all('tr')[0].find_all('td')

                if condition_search[0].text.lower() == "stanje:":
                    condition = condition_search[1].text
                else:
                    condition = laptop_page.find("table", attrs={"class": "table table-predmet table-predmet-info table-clear"}).find_all('tr')[1].find_all('td')[1].text

                condition = translate_condition(condition)

                search_status = True
                description = laptop_page.find('div', attrs={'id': 'opis'}).find_all('p')

                try:
                    screen_size = re.search(r'\d+[\.]\d+', (description[0].find_all('strong')[0].text).strip()).group().replace(".", ".")
                    laptop_brand = description[0].find_all('strong')[1].text
                    processor_brand, processor_number, number_of_cores, ram_generation, ram_amount, storage_type, storage_amount = get_details(description[1].text)
                except Exception as e:
                    print e
                    search_status = False

                # insert human knowledge
                number_of_cores, ram_generation, ram_amount = insert_human_knowledge(number_of_cores, price, ram_generation, ram_amount)

                if search_status and processor_number != "" and price != "" and storage_amount != "" and ram_amount != "":
                    laptops.append(Laptop(laptop_brand, processor_brand, processor_number, number_of_cores, ram_generation, ram_amount,
                                          storage_type, storage_amount, screen_size, price, condition, link.url, link.laptop_id))

                print count, link.display()
                count += 1

            except Exception as e:
                print e

    else:
        print "Laptop already exist: %s" % link.laptop_id

```

Листинг 2. метода get_laptop_info

Метода `get_details` уз помоћ регуларних израза парсира опис огласа и враћа прикупљене податке.

```
def get_details(description):

    processor_brand, processor_number, number_of_cores, ram_generation, ram_amount, storage_type, storage_amount = laptop_default_values()

    description_lower = description.lower()

    # get processor brand, number and number_of_cores
    if "quad" in description_lower:
        number_of_cores = 4

    processor_number_regex = re.search(r'[i][357]', description_lower)
    if processor_number_regex:
        processor_number = processor_number_regex.group()
        processor_brand = "intel"

    # get ram_generation and ram_amount
    ram_generation_regex = re.search(r'(memorija|ram)+.*(ddr)\d?', description_lower)
    if ram_generation_regex:
        ram_gen = re.search(r'(ddr)[234]', ram_generation_regex.group())
        if ram_gen:
            ram_generation = ram_gen.group()

    ram_regex = r'(memorija|ram)[^\S\n\t]*[:]?[^\S\n\t]*(ddr)?[234]?[^\S\n\t]*d+[^^\S\n\t]*(gb)'
    ram_amount_regex = re.search(ram_regex, description_lower)
    if ram_amount_regex:
        ram_r = re.search(r'\d+[^^\S\n\t]*(gb)', ram_amount_regex.group())
        if ram_r:
            ram_amount = re.search(r'\d+', ram_r.group()).group()

        if int(ram_amount) > 32:
            ram_amount = "8"

    # get storage type and amount
    ssd_regex = re.search(r'(ssd)', description_lower)
    if ssd_regex:
        storage_type = "ssd"

    storage_amount_regex = re.search(r'(disk)[^\S\n\t]*[:]?[^\S\n\t].*\d+[^^\S\n\t]*.*(gb)*', description_lower)
    if storage_amount_regex:
        storage_temp = re.search(r'\d+[ ]?(gb)*', storage_amount_regex.group())
        if storage_temp:
            storage_amount = re.search(r'\d+', storage_temp.group()).group()

    if storage_amount == "":
        storage_amount_regex = re.search(r'(hdd)[^\S\n\t]*[:]?[^\S\n\t].*\d+[^^\S\n\t]*.*(gb)*', description_lower)
        if storage_amount_regex:
            storage_temp = re.search(r'\d+[ ]?(gb)*', storage_amount_regex.group())
            if storage_temp:
                storage_amount = re.search(r'\d+', storage_temp.group()).group()

    if storage_amount == "":
        storage_amount_regex = re.search(r'(ssd)[^\S\n\t]*[:]?[^\S\n\t].*\d+[^^\S\n\t]*.*(gb)*', description_lower)
        if storage_amount_regex:
            storage_temp = re.search(r'\d+[ ]?(gb)*', storage_amount_regex.group())
            if storage_temp:
                storage_amount = re.search(r'\d+', storage_temp.group()).group()

    storage_amount_regex_terabyte = re.search(r'\d+[^\S\n\t]*?(tb)', description_lower)
    if storage_amount_regex_terabyte:
        storage_tb = re.search(r'\d+', storage_amount_regex_terabyte.group())
        if storage_tb:
            storage_amount = re.search(r'\d+', storage_tb.group()).group()

    if storage_amount != "":
        if int(storage_amount) == 1:
            storage_amount = "1000"
            storage_type = "hdd"

        if int(storage_amount) == 2:
            storage_amount = "2000"
            storage_type = "hdd"

        if int(storage_amount) < 100 or int(storage_amount) > 2000:
            storage_amount = ""

    return processor_brand, processor_number, number_of_cores, ram_generation, ram_amount, storage_type, storage_amount
```

Листинг 3. метода get details

Метода `insert_human_knowledge` претпоставља одређене спецификације на основу цене, стања и других карактеристика лаптопа, све у циљу прикупљања што већег броја података.

```
def insert_human_knowledge(number_of_cores, ram_generation, ram_amount, storage_amount, processor_number, storage_type, condition, price):

    # check number of cores
    if number_of_cores == "":
        if processor_number == 'i5' and price < 80000: number_of_cores = 2
        if processor_number == 'i5' and price >= 80000 and condition == 'used': number_of_cores = 4
        if processor_number == 'i7': number_of_cores = 4
        if processor_number == 'i3': number_of_cores = 2

    # check ram generation
    if ram_generation == "":
        if price < 50000: ram_generation = "ddr2"
        if price >= 50000 and price < 100000: ram_generation = "ddr3"
        if price > 100000: ram_generation = "ddr4"

    # check ram amount
    if ram_amount == "":
        if price < 50000 and condition == "new": ram_amount = 4
        if price < 50000 and condition == "used": ram_amount = 8
        if price >= 50000 and price < 100000: ram_amount = 8
        if price >= 100000 and price < 150000 and condition == "used": ram_amount = 16
        if price >= 100000 and price < 150000 and condition == "new": ram_amount = 8
        if price >= 150000: ram_amount = 16

    # check storage amount
    if storage_amount == "":
        if price < 50000: storage_amount = 250
        if price >= 50000: storage_amount = 500

    # check condition
    if processor_number == 'i5' and condition == 'used' and ram_amount == 8 and storage_type == 'ssd' and price < 25000:
        condition = 'defective'
    if processor_number == 'i5' and condition == 'used' and ram_amount == 8 and storage_type == 'hdd' and price < 20000:
        condition = 'defective'
    if processor_number == 'i7' and condition == 'used' and ram_amount <= 8 and price < 25000: condition = 'defective'

    # set typical if empty
    if number_of_cores == "":
        number_of_cores = 2
    if storage_type == "":
        storage_type = 'hdd'

    return number_of_cores, ram_generation, ram_amount, storage_amount, processor_number, storage_type, condition
```

Листинг 4. метода `insert_human_knowledge`

Све претходне описане методе започињу своје извршавање првенствено након позива методе `get_laptop_info` која се позива из `main` методе. Такође у `main` методи позивају се методе за учитавање и чување података.

```
if __name__ == "__main__":

    urls_filename = "urls"
    laptop_filename = "laptops"
    link_base_url = 'https://www.kupindo.com/Racunari-i-oprema/Laptop-racunari-delovi-i-oprema/Laptopovi/artikli/1473/cena_DESC_strana_'

    if check_does_file_exist('urls'):
        load_links(urls_filename)
        get_links(link_base_url)
        save_data(urls_filename, "urls")
    else:
        get_links(link_base_url)
        save_data(urls_filename, "urls")

    if check_does_file_exist('laptops'):
        load_laptops(laptop_filename)

    get_laptop_info()
    save_data(laptop_filename, "laptops_info")
```

Листинг 5. метода `main`

Након прикупљана свих података и њиховог чувања у CSV фајлу, подаци се накнадно учитавају и убацују у базу података. За ту потребу је имплементирана метода *insert_data_in_db* која уз ослонац на psycорг адаптер убацује све прикупљене податке у базу.

```
def insert_data_in_db():
    load_laptops('laptops')

    try:
        conn = psycopg2.connect("host=server-api-db.clyne3bmdpup.us-east-2.rds.amazonaws.com dbname=server_db user=ndakic password=postgres")
    except:
        print "I am unable to connect to the database"

    seq_con = conn.cursor()
    laptop_con = conn.cursor()
    scraper_con = conn.cursor()

    today = datetime.now()
    format_today = today.strftime("%Y-%m-%d %H:%M:%S") # 2019-04-21 00:00:00.000000
    seq_con.execute("SELECT last value FROM scraper_seq")
    next_value = seq_con.fetchall()[0][0]
    scraper_status = "success"

    try:
        for laptop in laptops:
            check_laptop = "select l.brand from laptop l where id = %d" % long(laptop.laptop_id)
            laptop_con.execute(check_laptop)

            if laptop.ram_amount == '':
                laptop.ram_amount = 0
            if laptop.storage_amount == '':
                laptop.storage_amount = 0
            if laptop.price == '':
                laptop.price = 0

            if len(laptop_con.fetchall()) == 0:
                values = '\'%s\', \'%s\', \'%s\', %d, \'%s\', \'%s\', %d, \'%s\', \'%s\', %d, \'%s\', \'%s\', \'%s\'' % (laptop.laptop_id,
                    laptop.laptop_brand, laptop.number_of_cores, int(laptop.price), laptop.processor_brand, laptop.processor_number, int(laptop.ram_amount),
                    laptop.ram_generation, laptop.screen_size, int(laptop.storage_amount), laptop.storage_type, laptop.condition, laptop.url)
                insert_query = "INSERT INTO laptop (id, brand, cores, price, processor_brand, processor_model, ram_amount, + \
                    ram_generation, screen_size, storage_amount, storage_type, condition, url) VALUES (%s)" % values
                print "Insert laptop:", insert_query
                laptop_con.execute(insert_query)
            else:
                print "Laptop %d already exist in DB." % long(laptop.laptop_id)

    except Exception as e:
        print e
        scraper_status = "error"
        transaction.rollback()

    scraper_values = '\'%d\', \'%s\', \'%s\', \'%s\', \'%s\'' % (next_value, "kupindo", format_today, str(len(laptops)), scraper_status)
    scraper_con.execute("INSERT INTO scraper (id, source, date, total, status) values (%s)" % scraper_values)

    set_seq = "SELECT setval('scraper_seq', %d, true)" % (long(next_value) + 1)
    seq_con.execute(set_seq)

    conn.commit()
    conn.close()
```

Листинг 6. метода insert_data_in_db

4.2 Имплементација модула за обраду података

Имплементација овог модула реализована је уз помоћ Java програмског језика уз ослонац на Spring радни оквир. Модул је реализован сагласно са MVC (model-view-controller) архитектуром и главна улога му је да обезбеди податке за приказ модулу који је задужен за њихову презентацију.

Пошто се суштини овог модула своди на читање података из PostgreSQL базе података и давања истих на коришћење, не постоји нека значајнија логика модула, али пошто је већ споменуто у раду да овај модул даје на коришћење све опције које је могуће одабрати за сваку појединачну спецификацију лаптопа, биће представљен интерфејс LaptopRepository који све то омогућује.

Интерфејс LaptopRepository наслеђује класу JpaRepository која садржи стандардне CRUD (create, read, update and delete) операције за Laptop класу, али поред њих имплементирани су и додатне операције везане за појединачне опције спецификације лаптопа које је могуће одабрати.

```
@Repository
public interface LaptopRepository extends JpaRepository<Laptop, Long> {

    Optional<Laptop> getId(Long id);

    List<Laptop> findAllByOrderByPriceDesc();

    List<Laptop> findByBrandIgnoreCaseContaining(String brand);

    @Query(value = "SELECT l from Laptop l where price > :lbound and price < :ubound and condition = :condition")
    List<Laptop> findAllByPriceBoundsAndCondition(@Param("lbound") Integer lbound, @Param("ubound") Integer ubound,
        @Param("condition") String condition);

    @Query(value = "SELECT DISTINCT brand from Laptop WHERE brand <> '' order by brand asc")
    List<String> findAllLaptopBrands();

    @Query(value = "SELECT DISTINCT processorBrand from Laptop WHERE processorBrand <> '' order by processorBrand asc")
    List<String> findAllLaptopProcessorBrands();

    @Query(value = "SELECT DISTINCT processorModel from Laptop WHERE processorModel <> '' order by processorModel asc")
    List<String> findAllLaptopProcessorModels();

    @Query(value = "SELECT DISTINCT cores from Laptop WHERE cores <> '' order by cores asc")
    List<String> findAllLaptopCores();

    @Query(value = "SELECT DISTINCT ramGeneration from Laptop WHERE ramGeneration <> '' order by ramGeneration asc")
    List<String> findAllLaptopRamGenerations();

    @Query(value = "SELECT DISTINCT ramAmount from Laptop order by ramAmount asc")
    List<String> findAllLaptopRamAmounts();

    @Query(value = "SELECT DISTINCT storageType from Laptop WHERE storageType <> '' order by storageType asc")
    List<String> findAllLaptopStorageTypes();

    @Query(value = "SELECT DISTINCT storageAmount from Laptop order by storageAmount asc")
    List<String> findAllLaptopStorageAmounts();

    @Query(value = "SELECT DISTINCT screenSize from Laptop WHERE screenSize <> '' order by screenSize asc")
    List<String> findAllLaptopScreenSizes();

    @Query(value = "SELECT DISTINCT condition from Laptop WHERE condition <> '' order by condition asc")
    List<String> findAllLaptopConditions();
}
```

Листинг 7. Класа LaptopRepository

4.3 Имплементација модула за приказ података

Презентација апликације имплементира је помоћу Angular радног оквира и Flexbox layout-a.

Због робусности неће бити приказан листинг апликације али ће бити представљано како изгледа апликација и које су њене функционалности.

Притиском на опцију Laptops, у навигационом бару, могу се видети сви прикупљени лаптопови односно њихове спецификације, слика 12. Подаци су приказани у форми табеле коју је могуће листати по странама. Laptops табела се може претраживати куцањем жељеног брента лаптопа и притиском на дугме search. Претрагу је могуће поништити притиском на дугме reset.

Поред лаптопова, са десне стране, постоји и друга табела Scraper, која указује на статус прикупљања података. У њој се може видети датум, извор и број прикупљених података.

The screenshot displays a web application interface with a dark blue header containing a home icon and the text 'Home Laptops'. The main content area is divided into two sections. The left section, titled 'Laptops', features a search bar with the placeholder 'Laptop brand...', a green 'SEARCH' button, and a green 'RESET' button. Below this is a table with 11 columns: Laptop Brand, Processor Brand, Processor Model, Processor Cores, RAM Generation, RAM Amount, Storage Type, Storage Amount, Screen size, Price, and Condition. The table lists various laptop models from brands like Dell, Apple, and Hewlett Packard. The right section, titled 'Scraper', contains a table with 4 columns: Source, Date, Total, and Status, showing data for 'kugrdo' from 2007 to 2019.

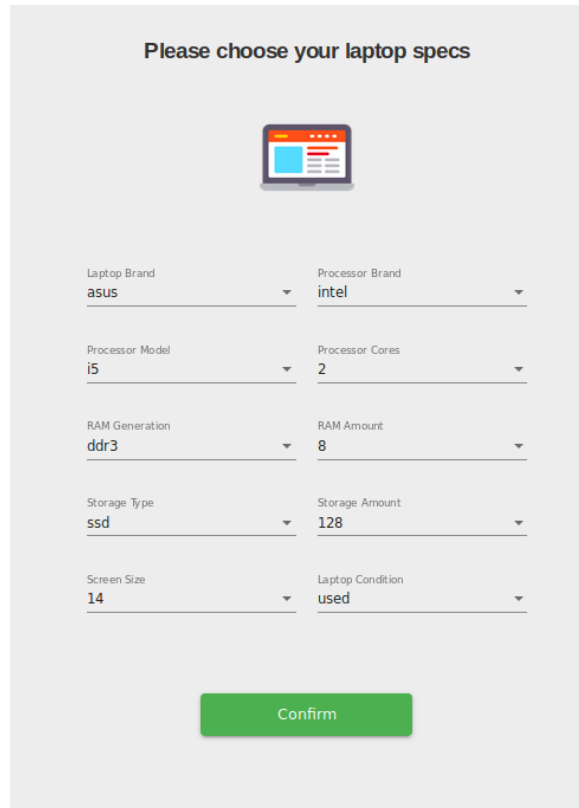
Laptops										
Laptop Brand	Processor Brand	Processor Model	Processor Cores	RAM Generation	RAM Amount	Storage Type	Storage Amount	Screen size	Price	Condition
Dell	Intel	i5	2	ddr2	8	sdd	128	12	47250000	new
Apple	Intel	i7	2	ddr2	4	sdd	250	15	47000000	refurbish
Dell	Intel	i7	2	ddr3	8	sdd	128	14	47000000	new
Lenovo	Intel	i7	2	ddr3	8	sdd	256	14	46000000	new
Hewlett Packard	Intel	i3	2	ddr4	4	hdd	500	15	45900000	new
Dell	Intel	i5	2	ddr3	4	hdd	500	15.6	45000000	new
Hewlett Packard	Intel	i5	2	ddr2	8	sdd	256	12.5	45400000	new
Hewlett Packard	Intel	i7	2	ddr3	8	hdd	500	14	44000000	new
Apple	Intel	i7	2	ddr3	8	hdd	250	15	44000000	new
Apple	Intel	i5	2	ddr3	8	sdd	120	13	43800000	new

« Previous 1 ... 30 31 32 ... 49 Next »

Scraper			
Source	Date	Total	Status
kugrdo	2007/2019	468	success
kugrdo	22/09/2019	385	success
kugrdo	14/05/2019	373	success
kugrdo	12/09/2019	356	success
kugrdo	09/05/2019	331	success

Слика 18. Laptops приказ

Притиском на опцију Home, приказује се форма за бирање спецификације лаптопа. Након одабира спецификације, притиском на дугме Confirm (потврди) врши се предикција цене лаптопа. Приказ форме се може видети на слици 19.



Please choose your laptop specs

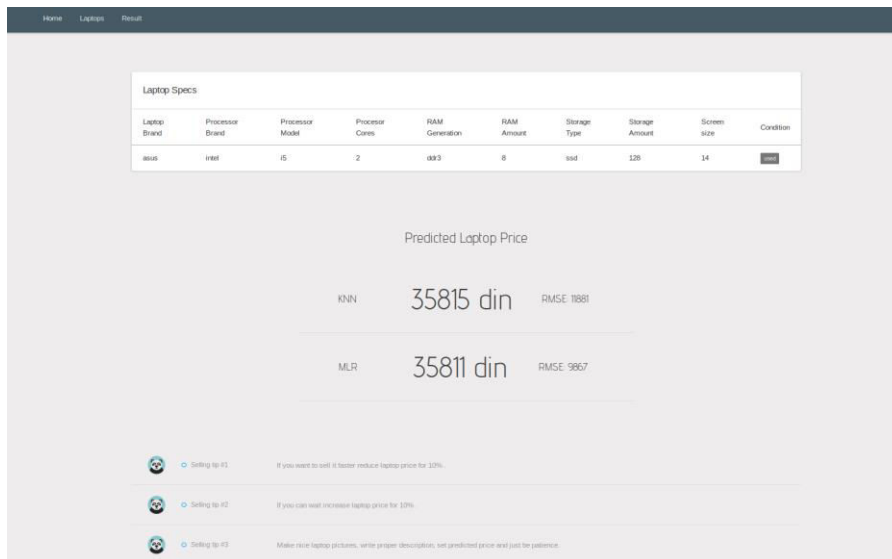
Laptop Brand	asus	Processor Brand	intel
Processor Model	i5	Processor Cores	2
RAM Generation	ddr3	RAM Amount	8
Storage Type	ssd	Storage Amount	128
Screen Size	14	Laptop Condition	used

Confirm

Слика 19. Одабир спецификације

Након одабира спецификације и притиском на дугме *confirm*, врши се предвиђање цене лаптопа. Као што се може видети на слици број 14, као резултат рада апликације, кориснику се приказују цене лаптопа. Поред сваке цене стоји која метода је предвидела ту цену и колика је њена просечна грешка (RMSE).

Поред цене, у табели *laptop specs* може се видети и која је спецификација одабрана. Такође испод цене додати су неки корисни предлози који се могу применити приликом продаје лаптопа.



Слика 20. Предвиђене цене лаптопа

4.4 Имплементација модула за предикцију цене

Модул за предикцију цене лаптопа имплементиран је употребом Python програмског језика уз ослонац на Django радни оквир, *pandas* и *scikit-learn* библиотеку.

На листингу под редним бројем 7, могу се видети како су имплементиране крајње тачке (endpoints) које су задужене за обраду захтева и позивање адекватних метода предикције.

Endpoint-и примају захтеве користећи тело захтева, (request body) које је у JSON формату који омогућава позивање ових метода без обзира на технологију.

Оба endpoint-а су имплементирана тако да се прво преузимају одабране карактеристике спецификације, а затим се те карактеристике адекватно трансформишу, како би се касније могле користити. Када су подаци адекватно трансформисани, прослеђују се методама које врше предвиђање и враћају цену лаптопа као и RMSE.

```

from django.http import HttpResponse
from django.views.decorators.csrf import csrf_exempt
from django.http.response import JsonResponse

from . import prediction

import json
import pandas as pd

columns = ["condition", "cores", "processor_model", "ram_amount", 'ram_generation', "storage_type"]

@csrf_exempt
def predict_price_mlr(request):
    params = json.loads(request.body.decode("utf-8"))

    data = [params['condition'], params['processorCores'], params['processorModel'], params['ramAmount'],
            params['ramGeneration'], params['storageType']]

    user_input = prediction.transform_input(pd.DataFrame(data=[data], columns=columns))

    result, rmse = prediction.predict_price_mlr(user_input)

    return JsonResponse({'result': round(result[0]), "RMSE": round(rmse)})

@csrf_exempt
def predict_price_knn(request):
    params = json.loads(request.body.decode("utf-8"))

    data = [params['condition'], params['processorCores'], params['processorModel'],
            params['ramAmount'], params['ramGeneration'], params['storageType']]

    data_frame = prediction.format_data(pd.DataFrame(data=[data], columns=columns))

    result, rmse = prediction.predict_price_knn(data_frame)

    return JsonResponse({'result': round(result[0]), "RMSE": round(rmse)})

```

Листинг 7. Prediction endpoints

Вишеструка линеарна регресија (MLR) имплементирана је тако да се прво из базе читају прикупљени подаци, који се након тога адекватно трансформишу, из разлога што је већина прикупљених података категоричног типа.

Затим се тако трансформисани подаци користе за подешавање (fitting) модела линеарне регресије уз помоћ којег се врши предвиђање цене лаптопа. Метода поред цене лаптопа, враћа и RMSE вредност која се израчунава на тренинг скупу.

Због робусности модула биће приказане само кључне методе.

```

# Multiple Linear Regression
def predict_price_mlr(data):

    loaded_data = load_data_from_db()
    data_with_dummies_var = transform_data(loaded_data)

    mlr = LinearRegression()

    # Split the data in training and test sets
    x_train, x_test, y_train, y_test = train_test_split(data_with_dummies_var, loaded_data.price,
                                                         test_size=0.10, random_state=5)

    mlr.fit(x_train, y_train) # Fitting a linear model

    print('Coefficients: \n', mlr.coef_, "Score: ", mlr.score(x_train, y_train))

    y_test_pred = mlr.predict(x_test)

    rmse = math.sqrt((mean_squared_error(y_test, y_test_pred)))

    result = mlr.predict(data.iloc[0].values.reshape(1, -1))

    return result, rmse

```

Листинг 8. Имплементација Вишеструке линеарне регресије

Метода најближих суседа (KNN) је такође имплементирана тако што се прво учитају прикупљени подаци који се адекватно трансформишу. Затим се врши стандардна подела података на тренинг и тест скуп, након које се уз помоћ KNeighborsClassifier методе одређује најбоље k (број најближих суседа), које се заједно са тренинг подацима проследи KNN моделу, ради његовог подешавања. Коришћењем тог модела, врши се предикција цене.

Метода такође враћа предвиђену цену и просечну грешку мерену на тест скупу.

```

# K-Nearest Neighbors
def predict_price_knn(data):

    formatted_data = format_data(load_data_from_db())

    # Drop columns
    x = formatted_data.drop(['price', 'processor_brand', 'id', 'url', 'screen_size', 'brand', 'storage_amount'], axis=1)

    x_train, x_test, y_train, y_test = train_test_split(x, formatted_data.price, test_size=0.10, random_state=5)

    num_list = list(range(1, 50)) # creating odd list of K for KNN

    neighbors = list(filter(lambda i: i % 2 != 0, num_list)) # subsetting just the odd ones

    cv_scores = [] # empty list that will hold cv scores

    # perform n-fold cross validation
    for k in neighbors:
        knn = KNeighborsClassifier(n_neighbors=k) # Finds the K-neighbors of a point.
        scores = cross_val_score(knn, x_train, y_train, cv=9)
        cv_scores.append(scores.mean())

    mse = [1 - cv for cv in cv_scores] # changing to misclassification error

    optimal_k = mse.index(min(mse)) # determining best k
    print("The optimal number of neighbors is %d" % optimal_k)

    neigh = KNeighborsRegressor(n_neighbors=optimal_k, weights='distance')
    neigh.fit(x_train, y_train)

    rmse = math.sqrt(mean_squared_error(y_test, neigh.predict(x_test)))
    result = neigh.predict(data.iloc[0].values.reshape(1, -1))

    return result, rmse

```

Листинг 9. Имплементација Методе најближих суседа

4.5 Deployment

Као што је већ речено у претходном делу рада Deployment апликација имплементиран је употребом Docker-а тако што је за сваку апликацију написан адекватан Dockerfile којим је описано како се апликација покреће.

У овом поглављу биће приказан и објашњен Dockerfile клијентеске Angular, серверске Spring апликације и Dockerfile Django апликације која се бави предикцијом цене лаптопа.

4.5.1 Deployment Angular апликације

За покретање клијентеске Angular апликације неопходно је имати инсталиран Node.js, те из тог разлога користи се званична Docker слика која садржи Node.js и све његове зависности. Сliku користимо користећи кључну реч FROM.

Затим се употребом кључних речи RUN и WORKDIR креира директоријум *app* и поставља се као радни.

Након тога се употребом кључних речи RUN и COPY прво глобално инсталира Angular а потом се сви апликацијски фајлови копирају у *app* радни директоријум.

На крају се уз помоћ кључне речи RUN врши позиционирање у *usr* директоријум а након тога се уз помоћ кључне речи CMD врши инсталирање свих неопходних зависности и покреће се апликација.

Све претходно описано се може видети на слици 21.

```
FROM node:11.14.0

RUN mkdir /usr/src/app
WORKDIR /usr/src/app
RUN npm install -g @angular/cli@7.3.8
COPY . /usr/src/app
RUN cd /usr/
CMD npm install && ng serve --host 0.0.0.0 --port 4200 --disable-host-check
```

Слика 21. Dockerfile Angular апликације

Након формирања описане Dockerfile датотеке неопходно је изградити Docker слику. Docker слику правимо извршавањем Docker команде `docker build -f Dockerfile -t client-app .` Docker build команда садржи велик број опција које је могуће

проследити build команди а начин њихове употребе као и списак свих опција је могуће видети на званичном сајту Docker-а [5]. Приликом креирање client-app Docker слике коришћене су опције `-f` након које се наводи жељена Dockerfile датотека и опција `-t` која поставља назив Docker слике.

```
Sending build context to Docker daemon 409.6MB
Step 1/7 : FROM node:11.14.0
--> 9289251188de
Step 2/7 : RUN mkdir /usr/src/app
--> Running in 68fd3a8637dd
Removing intermediate container 68fd3a8637dd
--> c2809126cfa6
Step 3/7 : WORKDIR /usr/src/app
--> Running in fe29a5ab768b
Removing intermediate container fe29a5ab768b
--> b17c0fce4141
Step 4/7 : RUN npm install -g @angular/cli@7.3.8
--> Running in 9a484clf2088
/usr/local/bin/ng -> /usr/local/lib/node_modules/@angular/cli/bin/ng
+ @angular/cli@7.3.8
added 289 packages from 181 contributors in 13.553s
Removing intermediate container 9a484clf2088
--> 4fddd83954b6
Step 5/7 : COPY . /usr/src/app
--> 99282a2682f8
Step 6/7 : RUN cd /usr/
--> Running in cf1409dfc32a
Removing intermediate container cf1409dfc32a
--> 7233b89f4f51
Step 7/7 : CMD npm install && ng serve --host 0.0.0.0 --port 4200 --disable-host-check
--> Running in 62fddc9bff33
Removing intermediate container 62fddc9bff33
--> 0bd91b6237c7
Successfully built 0bd91b6237c7
Successfully tagged client-app:latest
```

Слика 22. Испис конзоле након изградње client-app Docker слике

На слици 22 се може видети испис командне линије након изградње клијентске (client-app) Docker слике.

Након успешне изградње Docker слике, командом `docker run -d --name client-app client-app`, покрећемо клијентску Docker слику и креирамо Docker контејнер. Информације о активним контејнерима можемо видети извршавањем команде `docker ps`, слика 23.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
client-app	latest	e580d93066bd	19 minutes ago	1.37GB

Слика 23. Информације о client-app контејнеру

4.5.2 Deployment Spring апликације

За покретање серверске Java апликације неопходно је имати инсталиран JDK (Java Development Kit) који омогућава креирање Java апликације. Из тог разлога се користи званична Docker слика која садржи JDK верзије 8.

Да би се Java апликација могла покренути, претходно је потребно изгенерисати `.jar` фајл који садржи све фајлове и зависности апликације. То је могуће урадити на неколико начина а један од тих начина је додавање `maven-shade-plugin` додатка у `maven-ов.rom` фајл. Apache Maven Shade Plugin омогућава да се упакују све зависности апликације у `.jar` фајл. Након тога, да би се креирао `.jar` фајл потребно је извршити Maven команду `mvn package`.

Пример како изгледа Dockerfile уз помоћ којег се покреће Java апликација се може видети на слици 24.

Кључном речи `ADD` копира се `.jar` фајл апликације који се налази у директоријуму `target` у подразумевани (default) `root` радни директоријум Docker-а.

Кључном речи `EXPOSE` наводи се који `port` ће се користити приликом покретања апликације.

Кључном речи `ENTRYPOINT` наводи се команда која ће увек бити извршена приликом покретања контејнера, а она у овом случају покреће `.jar` фајл апликације тј. саму апликацију.

```
FROM openjdk:8
ADD target/original-server-api-0.0.1-SNAPSHOT.jar original-server-api.jar
EXPOSE 9005
ENTRYPOINT ["java", "-jar", "original-server-api.jar"]
```

Слика 24. Dockerfile Java апликације

Након формирања описане Dockerfile датотеке неопходно је изградити Docker слику. Docker слику правимо извршавањем Docker команде `docker build -f Dockerfile -t server-app .`

```

Sending build context to Docker daemon 100.3MB
Step 1/4 : FROM openjdk:8
--> bec43387959a
Step 2/4 : ADD target/original-server-api-0.0.1-SNAPSHOT.jar original-server-api.jar
--> bec35408bfe8
Step 3/4 : EXPOSE 9005
--> Running in 0dcbl704ead4
Removing intermediate container 0dcbl704ead4
--> 9b0420b506aa
Step 4/4 : ENTRYPOINT ["java", "-jar", "original-server-api.jar"]
--> Running in 49c8b5449199
Removing intermediate container 49c8b5449199
--> edd7a1a3f82f
Successfully built edd7a1a3f82f
Successfully tagged server-app:latest

```

Слика 25. Испис конзоле након изградње server-app слике

На слици 25 се може видети испис командне линије након изградње серверске (server-app) Docker слике.

Након успешне изградње Docker слике, командом `docker run -d --name server-app server-app` покрећемо серверску (server-app) Docker слику и креирамо Docker контејнер. Информације о активним контејнерима можемо видети извршавањем команде `docker ps`, слика 26.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
server-app	latest	fba4188b1bca	20 minutes ago	660MB

Слика 26. Информације о server-app контејнеру

4.5.2. Deployment Django апликације

За покретање Django апликације која се бави предикцијом цене лаптопа неопходно је имати инсталиран програмски језик Python, верзије 3. Због те потребе, као базична Docker слика користи се званична Docker слика која садржи Python 3, преузета са Docker Hub складишта. Сliku користимо користећи кључну реч `FROM`.

Пример како изгледа Dockerfile уз помоћ којег се покреће Django апликација се може видети на слици 27.

Уз помоћ кључне речи `RUN` покреће се команда `apt-get update` која служи за ажурирање листе најновијих верзија пакета.

Затим се уз помоћ кључне речи `WORKDIR` поставља радни директоријум у којем ће се извршавати све будуће команде.

Након тога се кључном речи COPY копира *requirement.txt* фајл у тренутни радни директоријум, након чега се употребом кључне речи RUN извршава команда за инсталирање свих неопходних зависности које Django апликација захтева.

Кључном речи COPY копира се целокупни садржај Django апликације у тренутни радни директоријум.

Кључна реч EXPOSE служи као документација на ком порту ће контејнер слушати.

Кључном речи CMD извршава се инструкција која покреће Django апликацију.

```
11 lines (8 sloc) | 197 Bytes
1 FROM python:3
2
3 RUN apt-get update
4
5 WORKDIR /usr/src/app
6 COPY requirements.txt ./
7 RUN pip3 install -r requirements.txt
8 COPY . .
9
10 EXPOSE 8000
11 CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
```

Слика 27. Dockerfile Django апликације

Након формирања описане Dockerfile датотеке неопходно је изградити Docker слику. Docker слику правимо извршавањем Docker команде `docker build -f Dockerfile -t prediction-api. .`

```

Sending build context to Docker daemon 203.3kB
Step 1/8 : FROM python:3
---> a4cc999cf2aa
Step 2/8 : RUN apt-get update
---> Using cache
---> c604074920b7
Step 3/8 : WORKDIR /usr/src/app
---> Using cache
---> 906f6a34cc84
Step 4/8 : COPY requirements.txt ./
---> de19342d26fb
Step 5/8 : RUN pip3 install -r requirements.txt
---> Running in 980d14901947
Removing intermediate container 980d14901947
---> 30bd563a52cb
Step 6/8 : COPY . .
---> 34620a33a962
Step 7/8 : EXPOSE 8000
---> Running in 31922d512648
Removing intermediate container 31922d512648
---> 389f3f6ae907
Step 8/8 : CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
---> Running in 322918a064ad
Removing intermediate container 322918a064ad
---> a645f7c62bca
Successfully built a645f7c62bca
Successfully tagged prediction-api:latest

```

Слика 28. Испис конзоле након изградње prediction-api слике

На слици 28 се може видети испис командне линије након изградње prediction-api Docker слике.

Након успешне изградње Docker слике, командом `docker run -d --name prediction-api prediction-api` покрећемо Docker слику и креирамо Docker контејнер. Информације о активним контејнерима можемо видети извршавањем команде `docker ps`, слика 29.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
prediction-api	latest	a645f7c62bca	9 minutes ago	1.37GB

Слика 29. Информације о prediction-api контејнеру

5 Верификација решења

Ради верификације успешности рада апликације, имплементирани су две методе машинског учења, метода вишеструке линеарне регресије (MLR) и метода најближих суседа (KNN). Обе методе користе исте прикупљене податаке али функционишу на сасвим другачији начин. Из тог разлога је могуће упоредити добијена предвиђања. Поред саме цене, обе методе враћају и просечну грешку предвиђања (RMSE), која говори колико добро методе функционишу.

На слици 14 из претходног поглавља, може се видети да обе методе предвиђају сличну цену и имају сличну просечну грешку за исту одабрану спецификацију лаптопа. MLR предвиђа цену од 35815 дин ($RMSE: 11881$ дин), док KNN предвиђа цену од 35811 дин ($RMSE: 9867$ дин).

Иако добијени резултати задовољавају сврху апликације, на основу добијених резултата може се уочити да је просечна грешка предвиђања код обе методе велика.

Разлог зашто је просечна грешка предвиђања велика, лежи највећим делом у самим подацима и проблему који се решава, као и у недовољном броју прикупљених података.

```
35268219,asus,used,4,71500,intel,i7,8,ddr3,15,240,hdd  
55826655,asus,used,4,70200,intel,i7,8,ddr3,17,1000,hdd  
47007701,asus,used,4,47000,intel,i7,8,ddr3,15.6,500,hdd  
46995633,asus,used,4,33500,intel,i7,8,ddr3,15.6,500,hdd  
55849687,asus,used,4,27500,intel,i7,8,ddr3,17.3,320,hdd
```

Слика 30. Примери лаптопова са сличном спецификацијом

Анализирајући прикупљене податке на слици 30 се може видети да иако лаптопови имају сличну, понекад и идентичну спецификацију, њихова цена се може драстично разликовати, па то знатно отежава предвиђање методама.

Један од разлога зашто се цене толико разликују лежи у самој чињеници у каквом стању се налази лаптоп. Стање *коришћено* (used) представља релативан појам и не описује довољно у каквом стању се тачно налази лаптоп. Такође постоје и разни други фактори који утичу на цену лаптопа, а тешко их је представити у информационом систему, као нпр. чињеница да се власнику из неког разлога жури да прода лаптоп (а то није нигде назначено), те из тог разлога даје нижу цену од стварне.

6. Закључак

У овом раду је развијена веб апликација за предикцију цена лаптоп рачунара на основу одабраних компоненти. Апликација је модуларна и састоји се од четири независна модула. Модул за прикупљање података задужен је за добављање података о лаптоп рачунарима. Прикупљени подаци су скрејповани са неколико веб сајтова. Модул за обраду података нуди REST API преко којег је могуће добавити прикупљене податке. За потребе овог рада развијена је и клијентска апликација која конзумира податке са поменутог REST API-ја. Најзначајнији модул у овом раду бави се предикцијом цене лаптопа применом техника машинског учења. Тај модул је проширив што је и приказано тиме што је подржано више техника предикције. С обзиром на чињеницу да читава апликација ради у веб окружењу, посебна пажња у овом раду посвећена је стављању апликација у продукцију. За ту намену коришћени су Docker контејнери - по један за сваки модул.

Циљ овог рада био је да се предложи ефикасно продукционо окружење за апликације машинског учења. То је и показано на примеру две методе - линеарна регресија и метода најближих суседа. Овај скуп метода је проширив па с тога један од праваца даљег истраживања подразумева имплементацију и других метода. Тиме би се добило да се методе једноставно могу поредити и било би потребно подржати да корисник сам може изабрати адекватну методу. То истраживање би пре свега требало да посвети пажњу конфигурабилности апликације и омогући кориснику без великог знања програмирања да користи имплентиране методе.

Модул за предикцију цена снажно се ослања на библиотеку `scikit-learn`. Даљи правци истраживања подразумевали би развој унимфорног интерфејса за друге библиотеке намењене машинском учењу. С обзиром на захтевност оваквих библиотека, архитектура продукционог окружење захтевала би додатну разраду.

7. Литература

- [1] G. Savić, Konstrukcija i testiranje softvera, materijali sa predavanja, Univerzitet u Novim Sadu, FTN 2018.
- [2] Spring framework, preuzeto sa: <https://docs.spring.io/spring/docs/current/spring-framework-reference/>
- [3] J. Slivka, Nastavni materijali na predmetu Mašinsko učenje
- [4] Italo José, <https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d>
- [5] Docker documentation, preuzeto sa: <https://docs.docker.com>
- [6] B. Perišić, Specifikacija i modeliranje softvera, materijali sa predavanja, Univerzitet u Novim Sadu, FTN 2014.

8. Биографија

Никола Дакић је рођен 18.10.1994. године у Новом Саду. Основну школу "Петар Кочић" завршио је 2009. године у Темерину. Економско-трговинску школу, смер економски техничар, завршио је 2013. године у Бечеју. Исте године уписао се на Факултет техничких наука у Новом Саду, смер Софтверско инжењерство и информационе технологије. Положио је све испите предвиђене планом и програмом.

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, RBR :	
Identifikacioni broj, IBR :	
Tip dokumentacije, TD :	Monografska publikacija
Tip zapisa, TZ :	Tekstualni štampani dokument
Vrsta rada, VR :	Diplomski rad
Autor, AU :	Nikola Dakić
Mentor, MN :	Dr Milan Segedinac
Naslov rada, NR :	Implementacija aplikacije za predikciju cene laptop računara
Jezik publikacije, JP :	Srpski
Jezik izvoda, Jl :	Srpski / engleski
Zemlja publikovanja, ZP :	Srbija
Uže geografsko područje, UGP :	Vojvodina
Godina, GO :	2019
Izdavač, IZ :	Autorski reprint
Mesto i adresa, MA :	Novi Sad, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6
Fizički opis rada, FO :	6 / 53 / 0 / 0 / 27 / 8 / 0
Naučna oblast, NO :	Primenjene računarske nauke i informatika
Naučna disciplina, ND :	Napredne web tehnologije
Predmetna odrednica / ključne reči, PO :	Deployment, docker, mašinsko učenje, predikcija cene laptop računara
UDK	
Čuva se, ČU :	Biblioteka Fakulteta tehničkih nauka, Trg Dositeja Obradovića 6, Novi Sad
Važna napomena, VN :	
Izvod, IZ :	U radu je prikazana aplikacija koja se bavi predikcijom cene laptop računara. Pored metoda mašinskog učenja koje su primenjene, akcenat u radu je stavljen na puštanje aplikacije u produkciju. Izlaz iz aplikacije je prediktovana cena laptop računara.
Datum prihvatanja teme, DP :	

Datum odbrane, DO :	
Članovi komisije, KO :	
predsednik	prof. dr. Goran Savić, vanredni profesor, FTN Novi Sad
član	prof. dr. Jelena Slivka, docent, FTN Novi Sad
mentor	prof. dr. Milan Segedinac, docent, FTN Novi Sad
Potpis mentora	

KEY WORDS DOCUMENTATION

Accession number, ANO :	
Identification number, INO :	
Document type, DT :	Monographic publication
Type of record, TR :	Textual material
Contents code, CC :	Bachelor Thesis
Author, AU :	Nikola Dakić
Mentor, MN :	Milan Segedinac, Professor, PhD
Title, TI :	Implementation of application for laptop price prediction
Language of text, LT :	Serbian
Language of abstract, LA :	Serbian / english
Country of publication, CP :	Serbia
Locality of publication, LP :	Vojvodina
Publication year, PY :	2019
Publisher, PB :	Author's reprint
Publication place, PP :	Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6
Physical description, PD :	6 / 53 / 0 / 0 / 27 / 8 / 0
Scientific field, SF :	Software engineering
Scientific discipline, ND :	Advanced web technologies
Subject / Keywords, S/KW :	Deployment, Docker, machine learning, laptop price prediction
UDC	
Holding data, HD :	Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad
Note, N :	
Abstract, AB :	The paper presents application which use machine learning methods to predict laptop price. Beside that, the emphasis of the paper is put on deployment of application. Application output is predicted laptop price.

Accepted by sci. board on, ASB :	
Defended on, DE :	
Defense board, DB :	
president	Goran Savić, Phd, associate prof., FTN Novi Sad
member	Jelena Slivka, Phd, assist prof., FTN Novi Sad
mentor	Milan Segedinac, Phd, assist prof., FTN Novi Sad
Mentor's signature	