PMF Cirilica.wmf

ИНСТИТУТ ЗА МАТЕМАТИКУ И ИНФОРМАТИКУ  
ПРИРОДНО – МАТЕМАТИЧКИ ФАКУЛТЕТ  
УНИВЕРЗИТЕТ У КРАГУЈЕВЦУ

Завршни рад

Сајт за преглед цена ИТ производа

**Ментор**

**др Владимир Цвjетковић**

**Студент**

**Андрија Лазић 53/20**

Крагујевац, 2024.

Садржај:

[Значење скраћеница и непознатих речи 4](#_Toc161495534)

[1 Увод 5](#_Toc161495535)

[2 Коришћене технологије 6](#_Toc161495536)

[2.1 MongoDB 6](#_Toc161495537)

[2.2 React 6](#_Toc161495538)

[2.3 Node.js 7](#_Toc161495539)

[2.4 Express.js 7](#_Toc161495540)

[2.5 TypeScript 7](#_Toc161495541)

[2.6 Redis 7](#_Toc161495542)

[2.7 Python 8](#_Toc161495543)

[2.8 Scrapy 8](#_Toc161495544)

[2.9 Git 8](#_Toc161495545)

[2.10 Postman 8](#_Toc161495546)

[3 Структура пројекта 9](#_Toc161495547)

[3.1 Архитектура система 9](#_Toc161495548)

[3.2 Листа функционалности 10](#_Toc161495549)

[3.3 Дијаграми случајева коришћења 10](#_Toc161495550)

[3.4 МОВ дијаграм 12](#_Toc161495551)

[4 Структура кода и организација 13](#_Toc161495552)

[4.1 WEB Scraper 13](#_Toc161495553)

[4.1.1 Пример креирања spider-a 13](#_Toc161495554)

[4.1.2 Рад са базом података 16](#_Toc161495555)

[4.1.3 Scrapy pipeline 18](#_Toc161495556)

[4.1.4 ProductMenager – Класа за управљање производима 18](#_Toc161495557)

[4.1.5 Распоред scrape-овања 20](#_Toc161495558)

[4.2 Node.js WEB сервер 21](#_Toc161495559)

[4.2.1 Руте 21](#_Toc161495560)

[4.2.2 Комуникација са базом података 22](#_Toc161495561)

[4.2.3 Контролер 23](#_Toc161495562)

[4.2.4 Сервиси 23](#_Toc161495563)

[4.2.5 Redis 24](#_Toc161495564)

[4.3 React frontend. 25](#_Toc161495565)

[4.3.1 Hooks 25](#_Toc161495566)

[4.3.2 Рутирање 26](#_Toc161495567)

[5 Функционалности 27](#_Toc161495568)

[5.1 Претрага производа 28](#_Toc161495569)

[5.2 Избор категорија 28](#_Toc161495570)

[5.3 Приказ појединачних производа 29](#_Toc161495571)

[6 Предлози за унапређење 30](#_Toc161495572)

[7 Литература 30](#_Toc161495573)

[Кратка биографија кандидата 31](#_Toc161495574)

# Значење скраћеница и непознатих речи

1. MERN stack - скуп од четири технологије: MongoDB, ExpressJs, React ,NodeJs
2. SCRAPER - назив за софтвер који се користи за преузимање података са WEB-страна
3. NoSQL - тип базе података која не користи SQL као језик за управљање базом и подацима
4. Framework – структура која служу као основа за изградњу софтвера
5. JSON(JavaScript Object Notation) - текст формат
6. BSON (Binary JSON) - бинарно енкодиран JSON
7. XML (Extensible Markup Language) - формат и језик за дефинисање података унутар тагова
8. JSX(JavaScript XML) - формат који нам омогућава да пишемо HTML елементе унутар JavaScript кода
9. DOM(Document Object Model) - представља страну и омогућава мењање структуре, стила…
10. Open-source - означава да је софтвер отвореног кода односно да је изворни код доступан свима који желе да га користе или мењају
11. API - Део Frontend-a и Backend-a који омогућава узајамну комуникацију
12. HTTP - један од многих интернет протокола
13. Spider - једна од алатки које се користе за Scrape-овање података
14. Pipeline - ток података у апликацији или процесу

# Увод

Савремени човек, који је посвећен свом послу и напретку, има мало слободног времена. Ово доводи до тога да човек не може или не жели да издвоји мало времена како би се што боље информисао пре куповине производа.

Продавци ово користе како би својим купцима продали застареле и неповољне производе. Најбољи пример ове појаве код нас је “Акцијска распродаја” за Црни петак, где продавци месец дана раније подигну цене производа само да би их за Црни петак вратили и тврдили нереалне попусте. Ова манипулација цена остаје незапажена од стране купаца који се надају повољним ценама и великим уштедама.

Апликација “EzDeals” је настала као решење за горе наведене проблеме. Назив „EzDeals“ је сленг односно скраћеница од „Easy Deals“ што би могло да се преведе као „Лаке добре понуде“ односно „Што лакше до добрих понуда“. Она ће омогућити корисницима да врше претрагу жељених производа и имају увид у тренутне цене у различитим продавницама као и промене цене датог производа кроз одређени временски период.

У тренутку писања овог документа постоји пар сличних веб апликација чији је највећи проблем лоша ажурност података, што доводи до тога да добар део производа није на стању или су цене истих нетачне. Овај проблем је један од главних повода за развој апликације „EzDeals“.

# Коришћене технологије

Пројекат се састоји из два дела: WEB апликација и SCRAPER сервис.

У основи WEB апликације се налази “MERN stack” који се састоји од следећих технологија:

* MongoDB
* Express.js
* React.js
* Node.js

Уз “MERN stack” је коришћена и REDIS база података за кеширање података у радној меморији. SCRAPER сервис је развијен у Python окружењу на основу Scrapy framework-a.

## MongoDB

MongoDB је NoSQL база података која складишти податке у BSON формату. Свака база се састоји од једне или више колекција а свака колекција у себи садржи документе у BSON формату.

Предности NoSQL база су:

* Флексибилност- NoSQL базе података лако могу обрадити било који формат података, као што су структурисани, полу-структурисани и неструктурисани подаци, у једном складишту података. Ово омогућава брз и поједностављен развој апликације.
* Скалабилности- у случају да нам је потребан већи меморијски простор можемо цео систем скалирати у ширину(изнајмљивање нових сервера) што је јефтиније и лакше од скалирања у висину(надоградња тренутног сервера).
* Високе перформансе- због честих измена података очекују се много боље перформансе у односу на традиционалне базе података. Ово је један од главних разлога што је одабрана NoSQL база података
* Лака имплементација- из базе можемо читати и уписивати JSON објекте што олакшава приступ подацима.

## React

React је библиотека за развој интерактивних и динамичних веб апликација. У тренутку писања овог документа је и најпопуларнија frontend технологија.

Кориснички интерфејс се састоји од мањих компонента које су представљене у JSX формату. JSX нам омогућава да пишемо HTML елементе унутар JavaScript кода. Више компонената се комбинују у појединачне стране и једна компонента се може искористити у креирању више различитих страна. Могућност поновне употребе компонената је једна од главних особина React-а која знатно убрзава развој апликације. Модуларна структура такође омогућава лакше одржавање кода што додатно утиче на време потребно за израду апликације.

Virtual DOM представља копију DOM-а и омогућава React-у да утврди промене настале од стране корисника и позадинских процеса и на основу датих промена одлучује које компоненте ће се поново рендеровати.

## Node.js

Node.js је open-source JavaScript окружење које нам омогућава да покренемо JavaScript код изван интернет претраживача. Ово је постигнуто помоћу Гугловог V8 engine-a. Node.js се може користити на различитим платформама: Windows, Linux, Unix, Mac OS X…

Кључне карактеристике Node.js-а су:

* Неблокирајућа петља догађаја са асинхроним извршавањем у једној нити, што га чини погодним за управљање конкурентним конекцијама и извршавање задатака као што су упис/читање података из фајлова, мрежни захтеви и операције са базама података без блокирања извршавања читавог програма.
* NPM (Node Package Manager), екосистем отворених библиотека и модула доступних за JavaScript. Лака инсталација и надоградња пакета што додатно омогућава лак транспорт пројекта са једног система на други.
* Заједница и екосистем: Node.js има активну заједницу која доприноси развоју библиотека и тиме олакшава процес израда веб апликација.

## Express.js

Express је framework имплементиран унутар Node.js-а који се користи за развој серверске стране веб апликације коришћењем JavaScript језика. Такође се користи и за послуживање статичких фајлова као што су слике,css и JavaScript фајлови. У овом пројекту је конкретно коришћен за дефинисање HTTP рута и middleware-a.

## TypeScript

Бесплатан и open-source језик који представља надоградњу JavaScript-а. Главна разлика између TypeScript-а и JavaScript-а је коришћење статичких типова насупрот динамичких што омогућава рано детектовање грешака и лакши развој апликације.

TypeScript подржава наслеђивање што омогућава креирање хијерархије класа.

TypeScript код се компајлира у JavaScript код који може бити извршен на било којем JavaScript окружењу. Ова особина нам је омогућила да TypeScript користимо при развоју React и Node.js апликација.

## Redis

Open-source, in-memory база података која се може користити за кеширање, стримовање, сервис за дистрибуцију порука…

У овом пројекту Redis је коришћен као систем за кеширање резултата одређених HTTP захтева што позитивно утиче на перфромансе. Једина мана овог система је што Redis није подржан на windows системима па је потребно користити виртуелну машину са Linux оперативним системом.

## Python

Python је одабран као основа scraping сервиса због великог броја доступних библиотека, framework-ова и алата за обраду података.

## Scrapy

Scrapy је open-source framework који се користи за преузимање различитих врста података са веб-а. Највећа предност овог framework-а је лако одржавање и надоградња што је у овом пројекту битно због честих измена и додавања нових производа. Пошто је framework написан у Python-у то му омогућава да ради на различитим окружењима као што су : Linux, Windows, Mac…

Scrapy се састоји од појединачних радника(Spider) који су задужени за рад на појединачним продавницама. Ово доводи до бољих перформанси и лакшег одржавања.

## Git

Git је бесплатан и open-source систем за контролу верзије кода. Главна функционалност Git-a је праћење измена у коду и другим фајловима током времена. Ово нам омогућава да видимо развој пројекта током времена и да се вратимо на претходне верзије ако је то потребно услед техничких проблема.

## Postman

Postman је познати алат који се користи за тестирање веб API-a које може бити ручно или аутоматско.

# Структура пројекта

Пре него што започнемо рад на самом пројекту добра пракса је да се уради спецификација дизајна софтвера у којој ће бити описане следеће ставке:

* Листа функционалности
* Архитектура система
* Дијаграми случајева коришћења
* Модел објекти везе
* Изглед корисничког интерфејса

## Архитектура система

Апликација „EzDeals“ ће се састојати из три самосталне целине: клијентски део, серверски део и микросервис за „web scraping“, као и базе података којој ће приступати и серверски део и микросервис. Све три целине ће бити на истом серверу ради практичности, али постоји могућност да се налазе на дистрибуираном систему уз минималне исправке конфигурационих фајлова. Ова карактеристика нам омогућава да лако проширимо цео систем ако апликација буде захтевала више рачунарских ресурса. Комуникација између њих ће се одвијати путем интернета користећи HTTP протоколе.

A diagram of a diagram

Description automatically generated

## Листа функционалности

У оквиру система ће постојати једна врста корисника а то је **нерегистровани** корисник. Они ће моћи да врше претрагу производа и имаће увид у тренутне цене код различитих продаваца као и историју цена.

Апликација мора да испуни следећу листу функционалности:

1. Обавезне функционалности
   1. Претрага производа
   2. Визуелизација цена производа
2. Пожељне функционалности
   1. Поређење производа
   2. Систем за обавештавање корисника о попустима
3. Опционе функционалности
   1. Логин и регистрација корисника
   2. Остављање коментара о производу

## Дијаграми случајева коришћења

A diagram of a diagram

Description automatically generatedПрви случај коришћења[Дијаграм 1] је претрага одређеног производа од стране корисника. У зависности од тога да ли корисник зна тачно који производ жели има могућност да унесе име производа у поље за претрагу или изабере одређену категорију производа

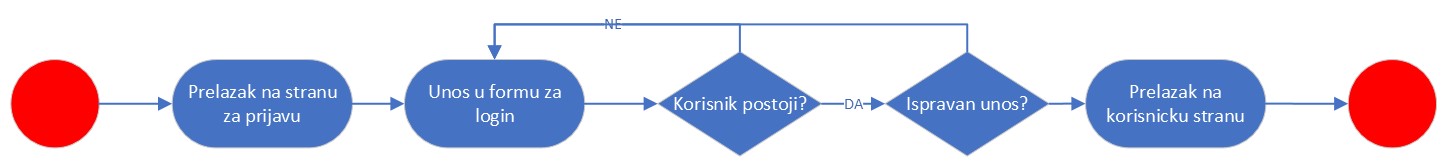
Дијаграм 2: Дијаграм активности претраге производа

На страни производа корисник ће моћи да остави коментар везан за дати производ[Дијаграм 3]. Корисник мора бити пријављен да би оставио коментар, ако није онда се пребацује на страну за регистрацију[Дијаграм 3]

A diagram of a diagram

Description automatically generated

Дијаграм 4: Дијаграм активности писања коментара



Дијаграм 5: Дијаграм активности пријаве корисника

Када се корисник успешно региструје потребно је да изврши верификацију емаил-а и да се улогује у свој налог[Дијаграм 4].

A diagram of a flowchart

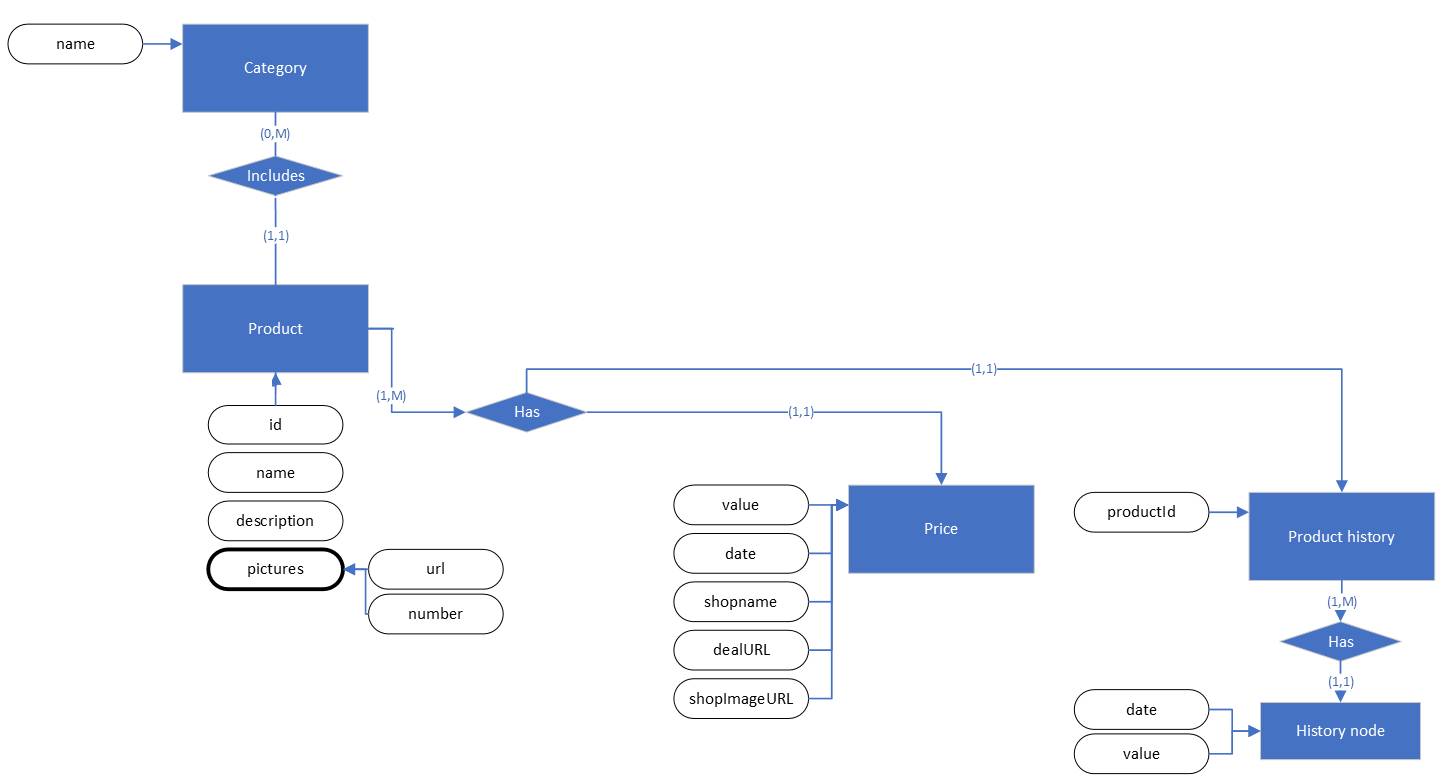
Description automatically generated

Дијаграм 6: Дијаграм активности пријаве корисника

## МОВ дијаграм

МОВ(Модел објекти и везе) дијаграм представља визуелну репрезентацију базе података. Кључне компоненте ER(Entity relation) дијаграма су:

* Ентитети: Објекти односно концепти у систему. Представљени помоћу правоугаоника
* Атрибути: Описују објекте
* Везе: Између објеката постоје везе које могу имати различите кардиналности. Везе су имплементиране тако што се памте ID-еви одређених докумената. На пример производ памти ID своје историје и на тај начин јој може приступити.



# Структура кода и организација

## WEB Scraper

WEB Scraper је развијен на Scrapy [[1]](#_Литература) framework-у унутар Python [[2]](#_Литература) развојном окружењу. Главна компонента Scrapy-a су Spider-и који се користе за scraping. Постоје два начина да се један сајт scrape-ује. Један је директно преузимање потребних података са API-a датог сајта, што је и најбржа метода scrape-овања. Друга метода подразумева пролазак кроз појединачне HTML стране и преузимање потребних података помоћу xpath-a и css селектора. Из овог разлога за сваки веб сајт је потребно направити посебан Spider зато што сваки сајт има различиту структуру.

### Пример креирања spider-a

#### Креирање класе

Процес креирања Spider-a започињемо креирањем класе која треба да наследи класу Spider из библиотеке Scrapy. У класи дефинишемо име, почетни url и стране које желимо да scrape-ујемо.

|  |
| --- |
| **Изворни код 4.1: Дефинисање spider класе** |
| 1. class GigatronscraperSpider(scrapy.Spider):  2. name = "gigatronScraper"  3. allowed\_domains = ["gigatron.rs"]  4. start\_urls = ["https://gigatron.rs"]  5. dt\_string=""  6.  7. pagesToScrape=[  8. PageInfo("https://search.gigatron.rs/v1/catalog/get/racunari-i-komponente/komponente/procesori?poredak=opadajuci&strana=",1,"RacunarskeKomponente"),  9. PageInfo("https://search.gigatron.rs/v1/catalog/get/racunari-i-komponente/komponente/maticne-ploce?poredak=opadajuci&strana=",1,"RacunarskeKomponente"),  10. PageInfo("https://search.gigatron.rs/v1/catalog/get/racunari-i-komponente/monitori?poredak=opadajuci&strana=",1,"Monitori"),  11. PageInfo("https://search.gigatron.rs/v1/catalog/get/tv-audio-video/slusalice?poredak=opadajuci&strana=",1,"Slusalice"),  12. PageInfo("https://search.gigatron.rs/v1/catalog/get/prenosni-racunari/laptop-racunari?strana=",1,"Laptopovi"),  13. PageInfo("https://search.gigatron.rs/v1/catalog/get/mobilni-telefoni-i-oprema/mobilni-telefoni?strana=",1,"MobilniTelefoni"),  14. PageInfo("https://search.gigatron.rs/v1/catalog/get/oprema-za-racunare/smestanje-podataka/eksterni-hdd-i-ssd?strana=",1,"EksterniDiskovi")] |

#### Функција за иницијализацију

Ова функција је наслеђена апстрактна функција која се извршава приликом иницијализације spider-a. Прво проверавамо да ли имамо приступ датој страни и памтимо време када је scrape-овање започето. На крају итерирамо кроз претходно дефинисане стране и шаљемо захтеве са њиховим url-овима и креирамо Listener којем ће се резултати захтева проследити када се захтев заврши.

|  |
| --- |
| **Изворни код 4.2: Функција за иницијализацију** |
| 17. def parse(self, initialResponse):  18. if(initialResponse.status!=200):  19. print("Could not access the: "+self.start\_urls[0])  20. return  21. self.now = datetime.now()  22. self.dt\_string = self.now.strftime("%d/%m/%Y %H:%M")  23.  24. for page in self.pagesToScrape:  25. yield scrapy.Request(  26. url=page.getCurrentURL(),  27. callback=self.parsePage) |

#### Пример scrape-овања позивањем API захтева

Када се захтев изврши резултат се прослеђује функцији parsePage. У колико је захтев успешан узимамо податке из response-a и прелазимо на следећу страну ако постоји.

|  |
| --- |
| **Изворни код 4.3: Пример scrape-овања позивањем API захтева** |
| 29. def parsePage(self,response):  30. if(response.status!=200):  31. print("Could not access the: "+response.url)  32. return  33.  34. currentPage=None  35. for page in self.pagesToScrape:  37. if page.URL in response.url:  38. currentPage=page  39. break  40.  41. data=response.json()  42.  43. if(currentPage.maxIndex==-1):  44. currentPage.maxIndex=data["totalPages"]  45.  46. for hit in data["hits"]["hits"]:  47. rowJSON=hit["\_source"]["search\_result\_data"]  48. price=rowJSON["price"].split(".")[0]  49. product=Product(rowJSON["name"],rowJSON["image"],self.dt\_string,currentPage.category)  50. product.addPrice(Price(price,"Gigatron",self.start\_urls[0]+rowJSON["url"],"https://gigatron.rs/images/gigatron.png"))  51. yield product  52.  53. if currentPage.index>currentPage.maxIndex:  54. return #self.products  55.  56. yield scrapy.Request(  57. url=currentPage.getCurrentURL(),  58. callback=self.parsePage) |

#### Пример scrape-овања помоћу xpath-a

XPath (XML Path Language) је програмски језик који нам омогућава навигацију кроз елементе и атрибуте XML документа. Претрагу вршимо помоћу HTML тагова и CSS класа.

|  |
| --- |
| **Изворни код 4.4: Пример scrape-овања помоћу xpath-a** |
| 28. def parsePage(self,response):  29.  30. if(response.status!=200):  31. print("\n")  32. print("Could not access the: "+response.url)  33. print("\n")  34. return  35.  36. currentPage=None  37. for page in self.pagesToScrape:  38.  39. if page.URL in response.url:  40. currentPage=page  41. break  42.  43. productsElements=response.xpath('.//div[@class="shop-product-card relative"]')  44.  45. if len(productsElements)==0:  46. return  47.  48. for productEl in productsElements:  49. pictureURL=(productEl.xpath(".//div[@class='product-image-wrapper']/a/img/@src")).get()  50. if pictureURL is None:  51. pictureURL=""  52.  53. name=(productEl.xpath('.//h2[@class="product-name"]/a/text()').get()).strip()  54. product=Product(  55. name,  56. pictureURL,  57. self.dt\_string,  58. currentPage.category  59. )  60.  61. price=(productEl.xpath('.//div[@class="price-holder"]/div[@class="text-bold"]/span/following-sibling::text()[1]')).get()  62.  63. #if product is on action  64. if price is None:  65. price=(productEl.xpath('.//div[@class="price-holder"]/span[@class="text-bold"]/span/following-sibling::text()[1]')).get()  66.  67. price=price.strip().split(",")[0]  68. price=price.split(".")[0]+price.split(".")[1]  69.  70. product.addPrice(Price(  71. price,  72. "Gstore",  73. (productEl.xpath('.//h2[@class="product-name"]/a/@href').get()).strip(),  74. "https://www.gstore.rs/images/gstore-final-logo-with%20background%20w%20or%20b-21.png"))  75. yield product  76.  77. yield scrapy.Request(url=currentPage.getCurrentURL(),callback=self.parsePage) |

### Рад са базом података

Комуникација са базом података се обавља помоћу Pymongo [[3]](#_Литература) библиотеке. Сва логика везана за базу податак се налази унутар singleton класе Database.

Singleton се постиже тако што override-ујемо статичну методу **\_\_new\_\_()** која се извршава приликом инстанцирања новог објекта дате класе. Функција као параметар прихвата **cls** који представља класу која се иницијализује.

Приликом креирања инстанце може доћи до проблема који се зове **Race Condition**. Овај проблем настаје када се две или више нити „такмиче“ за приступ заједничким ресурсима. У нашем случају може доћи до тога да неке нити у исто време покушају да инстанцирају објекат и у исто време утврде да тренутно не постоји ниједна инстанца дате класе. Тада ће прва нит извршити иницијализацију која ће бити обрисана од стране наредне нити, и тако све док се не дође до последње нити која ће једина имати исправну инстанцу базе.

Овај проблем се решава тако што имплементирамо \_\_new\_\_() методу која је **Thread-Safe.** Ово се постиже коришћењем Lock-а који ће омогућити креирање критичне секције којој може приступити само једна нит у једном тренутку.

|  |
| --- |
| **Изворни код 4.5: Пример singleton класе у Python-у.** |
| 1. class Database:  2. db = None  3. client = None  4. productHistoryCollectionName="productHistory"  5. \_instance = None  6. \_lock = threading.Lock()  7.  8. #\_\_new\_\_ is called whenever Python instantiates a new object of a class  9. def \_\_new\_\_(cls):  10.  11. #create new instance if it doesnt exist  12. with cls.\_lock:  13. #if instance already exists return it  14. if cls.\_instance is not None:  15. return cls.\_instance  16.  17.  18. instance=cls.\_instance = super().\_\_new\_\_(cls)  19.  20. dotenv\_path =os.path.abspath(os.path.join(os.getcwd(),Path('../../.env')))  21. load\_dotenv(dotenv\_path=dotenv\_path)  22.  23. cls.client = MongoClient(os.getenv('MongoDBConnectionString'))  24. cls.db = cls.client[os.getenv('MongoDBName')]  25. return instance |

Када завршимо са иницијализацијом базе можемо да кренемо са креирањем упита. Упит се креира тако што прво изаберемо колекцију над којом желимо да извршимо упит и затим одаберемо операцију коју желимо да извршимо. Примери операција су додавање новог документа[Изворни код 4.5], читање документа из базе[Изворни код 4.6] и измена документа[Изворни код 4.7]

|  |
| --- |
| **Изворни код 4.5: Пример упита за уписивање података у бази.** |
| 1. def insertProduct(self,item:Product,collection\_name:str):  2. """  3. used to insert item into selected collection  4.  5. :param item: item u want to insert  6. :param collection\_name: name of collection u want item to insert into  7. :return: \_id  8. """  9. return (self.db[collection\_name].insert\_one(ItemAdapter(item).asdict())).inserted\_id |

|  |
| --- |
| **Изворни код 4.6: Пример упита за читање података из базе.** |
| 1. def getOneProduct(self,filter:dict,collection\_name:str):  2. """  3. used to get Product from selected collection  4.  5. :param collection\_name: name of collection  6. :param filter: what product you want to get, needs to be in following format:{'\_id':ObjectId(item.\_id)} or any other  7. :return: Product  8. """  9. product=(self.db[collection\_name].find\_one(filter))  10. if product is None:  11. return None  12. return Product.from\_dict(product) |

|  |
| --- |
| **Изворни код 4.7: Пример упита за измену података.** |
| 1. def updateProduct(self,filter:dict ,update:dict,collection\_name:str):  2. """  3. used to update item in selected collection  4.  5. :param filter: what product you want to update, needs to be in following format:{'\_id':ObjectId(item.\_id)}  6. :param update: changes made to a product  7. :param collection\_name: name of collection where item can be found  8. :return: \_id  9. """  10.  11.  12. if 'prices' in update:  13. for price in update.get("prices"):  14. index\_to\_update = update['prices'].index(price)  15. update['prices'][index\_to\_update]=price.serialize\_price()  16.  17. update={  18. "$set":update  19. }  20. return (self.db[collection\_name].update\_one(filter,update)) |

### Scrapy pipeline

Представља везу између spider-a и ProductMenager-а(4.1.4) . Када spider заврши са обрадом производа исти шаље методи process\_item која га прослеђује одређеном ProductMenager-у у зависности од категорије производа.

|  |
| --- |
| **Изворни код 4.8: Пример упита за измену података.** |
| 1. class MongoDBpipeline:  2. mapOfProductMenagers={}  3.  4. for categoryName in os.environ.get("PRODUCT\_CATEGORIES").split(","):  5. mapOfProductMenagers[categoryName]=ProductMenager(categoryName)  6.  7. def open\_spider(self,spider):  8. self.database=Database()  9.  10. def process\_item(self, item:Product, spider:scrapy.Spider):  11. if item.primaryCategory in self.mapOfProductMenagers:  12. self.mapOfProductMenagers[item.primaryCategory].addProduct(item) |

### ProductMenager – Класа за управљање производима

Имплементиран је тако да за сваку категорију производа постоји по један singleton. Сваки појединачни ProductMenager има своју нит што ће убрзати рад са Spider-има и упис података у базу. Да смо се одлучили да користимо само једну нит за све ProductMenager-е то би довело развоја **уског грла** између ProductMenager-a и Spider-a који користе по једну нит.

|  |
| --- |
| **Изворни код 4.9: ProductMenager** |
| 1. class ProductMenager:  2. \_instances = {}  3. \_lock = threading.Lock()  4.  5. productsMap={}  6.  7. #\_\_new\_\_ is called whenever Python instantiates a new object of a class  8. def \_\_new\_\_(cls,productCategory:str):  9. #if instance already exists return it  10. if not cls.\_instances:  11. cls.\_instances=defaultdict(ProductMenager)  12.  13.  14. if productCategory in cls.\_instances:  15. return cls.\_instances.get(productCategory)  16.  17. #create new instance if it doesnt exist  18. with cls.\_lock:  19.  20. # Another thread could have created the instance  21. # before we acquired the lock. So check that the  22. # instance is still nonexistent.  23. if productCategory not in cls.\_instances:  24. cls.\_instances[productCategory] = super().\_\_new\_\_(cls)  25. cls.\_instances[productCategory].productsMap=defaultdict(Product)  26.  27. return cls.\_instances.get(productCategory) |

Производи се чувају у одређеном ProductMenager-у, а као структура података је одабрана мапа. Разлог је тај што мапа има много боље перформансе у односу на листу када је у питању приступ појединачним елементима. У колико дати производ већ постоји онда се само додаје нова цена из дате продавнице.

|  |
| --- |
| **Изворни код 4.10: Додавање производа у ProductMenager** |
| 1. def addProduct(self,product:Product):  2. """  3. Add a product to the map  4. :param product: product you wish to add  5. :return: None  6. """  7. with self.\_lock:  8. productOld:Product=self.productsMap.get(product.name)  9. if productOld is None:  10. self.productsMap[product.name]=product  11. return  12. #check for duplicates, only gigatron store  13. if product.prices[0].shopname=="Gigatron":  14. for price in productOld.prices:  15. if price.shopname==product.prices[0].shopname:  16. return  17. productOld.addPrice(product.prices[0])  18. self.productsMap[product.name]=productOld |

Када се заврши scrape-овање свих продавница онда се врши уписивање свих производа у базу[Изворни код 4.11]. У колико производ већ постоји у бази, врши се измена цена и историје производа.

|  |
| --- |
| **Изворни код 4.11: Додавање производа у базу** |
| 1. def uploadProductsToDatabase(self):  2. """  3. Uploads all products to the selected category  4. :return: None  5. """  7. category=self.getCurrentInstanceCategory()  8.  9. print("\n\n\n\n")  10. print("Started uploading products from category:")  11. print(category)  12. print("\n\n\n\n")  13.  14. database=Database()  15.  16. database.createIndex(1,"name",category)  17.  18. currentTime = datetime.now()  19.  21. #Iterating trough all items and checking if they already exist in database  22. for key, val in self.productsMap.items():  23. oldProduct=database.getOneProduct({'name':key},category)  24. lowestPrice=(min(val.prices, key=lambda x: x.value)).value  25.  26. if oldProduct is None:  27.  28. val.currentBestPrice=lowestPrice  29. val.dateAdded=currentTime  30. val.lastScraped=currentTime  31. productHistory=ProductHistory()  32. productHistory.history.append(ProductHistoryNode(val.lastScraped,lowestPrice))  33.  34. historyID=database.insertHistory(productHistory)  35. val.historyID=historyID  36. database.insertProduct(val,category)  37. continue  38.  39. val.currentBestPrice=lowestPrice  40. changes=Product.returnProductChanges(oldProduct,val)  41. database.updateProduct({'\_id':ObjectId(oldProduct.\_id)},changes,category) |

### Распоред scrape-овања

Сама природа апликације захтева да се scrape-овање извршава више пута у току дана. Овим омогућавамо да корисници увек имају увид у најновије промене цена производа.

Да би ово постигли користили смо библиотеке:

* Multiprocessing
* Threading
* Schedule [[4]](#_Литература)

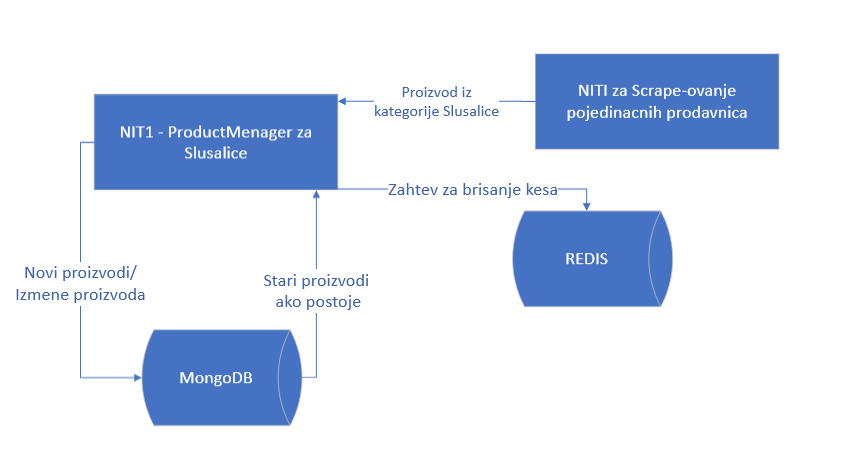
Scrape-овање ће се извршавати сваког сата а попуњавање историје једном дневно[Изворни-код 4.12]. Scrape-овање за појединачне продавнице се извршава паралелно.

|  |
| --- |
| **Изворни код 4.12: Распоред извршавања**  **scrape-овања** |
| 1. def run\_process(job\_func):  2. process = multiprocessing.Process(target=job\_func)  3. process.start()  4.  5. schedule.every().hour.do(run\_process,Scraping)  6. schedule.every().day.at('00:00').do(run\_process,HistoryThreading)  7.  8. Scraping()  10. if \_\_name\_\_ == "\_\_main\_\_":  11. while 1:  12. schedule.run\_pending()  13. time.sleep(60) |

Процес scrape-овања[Изворни код 4.13] започињемо тако што за све производе у бази сетујемо параметар visibility на true и започињемо scrape-овање појединачних продавница. Коришћењем нити ови процеси се извршавају паралелно што повољно утиче на перформансе. Када се процеси заврше врши се уписивање нових производа у базу, односно измена постојећих производа и чишћење Redis кеша.

|  |
| --- |
| **Изворни код 4.13: Извршавање**  **scrape-овања** |
| 1. def Scraping():  2. database=Database()  3.  4. listOfProductMenagers=[]  5.  6. for categoryName in os.environ.get("PRODUCT\_CATEGORIES").split(","):  7. listOfProductMenagers.append(ProductMenager(categoryName))  8.  10. visibilityThread=threading.Thread(setAllProductsVisibility(listOfProductMenagers))  11. visibilityThread.start()  12.  13. settings = get\_project\_settings()  14. process = CrawlerProcess(settings)  15. process.crawl(GigatronscraperSpider)  16. process.crawl(PcPracticSpider)  17. process.crawl(TehnomanijaSpider)  18. process.crawl(GstoreSpider)  19.  20. process.start() # the script will block here until all crawling jobs are finished  21. visibilityThread.join()  22.  23. #Upload all products to a database  24.  25. for menager in listOfProductMenagers:  26. menager.uploadProductsToDatabase()  28. database.close\_db()  30. RedisDatabase().clearDatabase() |

Илустровани приказ Pipeline-a података од WEB стране до базе се може видети на дијаграму доле.



Дијаграм рада Spider-a и ProductMenager-a за категорију слушалице

## Node.js WEB сервер

WEB сервер је развијен у Node.js окружењу, а као језик је коришћен Typescript. Главни задатак веб сервера је да опслужује захтеве који долазе са Front End стране апликације.

### Руте

Рутирање је механизам којим се захтеви наводе до контролера који ће тај захтев да обради. Рутер је имплементиран помоћу Router модула из Express.js [[5]](#_Литература) библиотеке.

|  |
| --- |
| **Изворни код 4.14: Руте** |
| 1. class ProductRoutes {  2. private router: Router;  3.  4. constructor() {  5. this.router = express.Router();  6. }  7. public routes(): Router {  8. this.router.get(  9. "/product/newProducts",  10. ProductControler.prototype.getNewProducts  11. );  12.  13. this.router.get(  14. "/product/history/:historyId",  15. ProductControler.prototype.getProductHistory  16. );  17.  18. this.router.post(  19. "/product/search",  20. ProductControler.prototype.getProductsBySearch  21. );  22.  23. this.router.get(  24. "/product/:productCategory",  25. ProductControler.prototype.getProductsByCategory  26. );  27.  28. this.router.get(  29. "/product/:productCategory/:productId",  30. ProductControler.prototype.getSingleProduct  31. );  32. return this.router;  33. }  34. } |

### Комуникација са базом података

Комуникација је изведена уз помоћ mongoosejs [[6]](#_Литература) библиотеке. У класи Database се налази метода за успостављање везе са базом података. Из .**env** фајла узимамо два параметра, конекциони стринг и име базе, који су нам потребни за успостављање везе са базом. Додатно креирамо **Event listener-**е који ће нас обавестити ако дође од грешака или до проблема са конекцијом.

|  |
| --- |
| **Изворни код 4.15: Комуникација са базом података** |
| 1. class Database {  2. constructor() {  3. this.\_connect();  4. }  5.  6. \_connect() {  7. const MongoDBConnectionString = process.env.MongoDBConnectionString;  8. const MongoDBName = process.env.MongoDBName;  9. if (!MongoDBConnectionString || !MongoDBName) {  10. log.error("MongoDB connection string is not provided.");  11. process.exit(1);  12. }  13. const connString = MongoDBConnectionString + "/" + MongoDBName;  14. mongoose  15. .connect(connString)  16. .then(() => {  17. log.info("Connected to MongoDB:" + connString);  18. RedisFactory.connect();  19. })  20. .catch((error) => {  21. log.error("MongoDB connection error:", error.message);  22. process.exit(1);  23. });  24.  25. mongoose.connection.on("error", (err) => {  26. log.error("MongoDB error:", err);  27. });  28.  29. mongoose.connection.on("disconnected", () => {  30. log.info("MongoDB disconnected, trying to reconnect");  31. this.\_connect();  32. });  33. }  34. } |

Пре него што започнемо имплементацију читања и уписа података потребно је да креирамо шеме наших модела из базе података. У шеми дефинишемо параметре и њихове особине.

|  |
| --- |
| **Изворни код 4.16: Пример шеме производа** |
| 1. const productSchema: Schema = new Schema({  2. name: { type: String, require: true },  3. image: { type: String, require: true },  4. dateAdded: { type: Date, require: true },  5. lastScraped: { type: Date, default: "" },  6. primaryCategory: { type: String, default: "" },  7. prices: { type: [] },  8. historyID: { type: ObjectId }  9. }); |

Шему не можемо користити за директан рад са базом, за то нам је потребно да уз помоћ дате шеме креирамо модел. У класи productCategories се чувају singleton модели појединачних категорија производа, који се после могу користити за читање и уписивање нових података у базу.

|  |
| --- |
| **Изворни код 4.17: Модел категорије производа** |
| 1. class productCategories {  2. private instances = new Map();  3.  4. constructor() {  5. for (let i = 0; i < config.PRODUCT\_CATEGORIES!.length; i++) {  6. const categoryName = config.PRODUCT\_CATEGORIES![i];  7. const newModel: Model<IProductDocument> = model<IProductDocument>(  8. categoryName,  9. productSchema,  10. categoryName  11. );  12. this.instances.set(categoryName, newModel);  13. }  14. } |

### Контролер

Слој задужен за обраду захтева прослеђених од стране рутера. Пре него што започнемо обраду захтева потребно је да проверимо исправност података. За ово је задужен декоратор који представља врсту middleware-a која пролази кроз предефинисане опције појединачних поља. Декоратор може да преправи појединачна поља или да прекине захтев.

|  |
| --- |
| **Изворни код 4.18: Пример декоратора који се налази изнад методе** |
| 1. @productSearchValidation(searchInfoSchema)  2. public async getProductsByCategory(  3. request: Request,  4. response: Response  5. ): Promise<void> |

Када декоратор утврди исправност података исте прослеђује методи над којом се налази.

### Сервиси

Најнижи слој који је задужен за комуникацију са базом података уз помоћ модела. Контролери захтевају од сервиса читање и упис података. Сервис се састоје од асинхроних метода које враћају promise-e. Ти promise-и се затим обрађују у контролерима.

|  |
| --- |
| **Изворни код 4.19: Пример читања података из базе у сервису** |
| 1. products = await ProductCategories.getCategory(  2. searchInfo.productCategory  3. )  4. .find(getDefaultFilter(), ["name", "image", "currentBestPrice","primaryCategory"])  5. .sort(sortParameter)  6. .skip((searchInfo.pageNum - 1) \* searchInfo.numberOfProducts)  7. .limit(searchInfo.numberOfProducts)  8. .lean()  9. .exec(); |

### Redis

За комуникацију са Redis базом је коришћена Redis [[7]](#_Литература) библиотека. Брзина апликације је једна од најбитнијих ствари које утиче на позитивно корисничко искуство. Redis база је коришћена као системски кеш зарад боље одзивности апликације.

|  |
| --- |
| **Изворни код 4.20: Пример методе за кеширање производа** |
| 1. \* @param product product you want to save.  2. \*/  3. public async saveProductToCache(product: IProductDocument): Promise<void> {  4. try {  5. await this.client.set(  6. `products:${product.\_id}`,  7. JSON.stringify(product)  8. )  9. } catch (error) {  10. this.log.error(error);  11. throw new ServerError("Server error try again");  12. }  13. } |

Утицај на перформансе је лако тестирати уз помоћ разних алата као што је на пример **Postman**. Време извршавања HTTP захтева за преузимање најновијих производа је просечно **70ms** а када применимо кеширање само ће први послати захтев имати време извршавања од 70ms док ће сваки наредни бити мањи од **10ms**.

Иако се ова разлика не може приметити од стране једног корисника, она је и те како битна јер са растом броја корисника који у исто време користе апликацију расте и време извршавања захтева. Дуго време извршавања захтева доводи до негативног корисничког искуства.

## React frontend.

Frontend је развијен помоћу React.js библиотеке у Typescript-у. Најмања јединица градње у React-у је компонента. Иста компонента се може користити на више различитих страна. Компоненте се могу креирати на два начина: као методе и као класе. У оба случаја је потребно да повратна вредност буде JSX формата.

|  |
| --- |
| **Изворни код 4.21: Пример компоненте** |
| 1. const ShortProduct = (props:any) => {  2. const product:IShortProduct=props.product as IShortProduct;  3. const productLink="/kategorije/"+product.primaryCategory+"/"+product.\_id;  4.  5. return (  6. <div className='col-12 col-lg-4 col-xxl-2 p-2 '>  7. <div className='shortItem'>  8. <Link to={productLink}>  9. <div className="col-sm productImage">  10. <img className='card-img-top' src={product.image}/>  11. </div>  12. <div className='productData'>  13. <div className="productName">  14. <h5>{product.name}</h5>  15. </div>  16. <div className='productPrice'>  17. {product.currentBestPrice} din.  18. </div>  19. </div>  20. </Link>  21. </div>  22. </div>  23. );  24. }; |

### Hooks

Једна од главних функционалности React-а су Hook-ови. Сваки Hook има одређену намену и најкоришћенији Hook је useState који се користи за декларацију и сетовање параметара. Када дође до промене параметара дефинисаних у useState-у доћи ће и до поновног рендеровања стране што ће омогућити кориснику да види вредност параметара у реалном времену.

У доле наведеном примеру се може видети како се користе два различита Hook-а за учитавање производа.

useState ће извршити поновно рендеровање стране када се производи буду учитали са Node.js сервера, а useEffect ће поново захтевати учитавање производа ако дође до промене тренутне стране.

|  |
| --- |
| **Изворни код 4.22: Пример коришћења Hooks-ова** |
| 1. const [products, setProducts] = useState<IShortProduct[]>([]);  2. useEffect(() => {  3. getProducts();  4. }, [props.currentPage]); |

### Рутирање

За рутирање[Изворни код 4.23] је коришћена react-router-dom библиотека. Рутирање нам омогућава да управљамо приказом различитих страна наше апликације на основу URL-a.

У URL-у се могу налазити URL параметри који се дефинишу тако што се пре назива параметра стави знак „ : “. Пример се може видети на линији 27. где имамо два параметра :category и :productId које користимо приликом претраге производа.

|  |
| --- |
| **Изворни код 4.23: Рутирање** |
| 1. const router = createBrowserRouter([  2. {  3. element: <App />,  4. path: "/",  5. children: [  6. {  7. path: "",  8. element: <NewestProducts />,  9. },  10. {  11. path: "noviProizvodi",  12. element: <NewestProducts />,  13. },  14. {  15. path: "pretraga",  16. element: <ProductSeach />,  17. },  18. {  19. path: "kategorije",  20. element: <CategoriesSelection />,  21. },  22. {  23. path: "kategorije/:category",  24. element: <CategoryProducts />,  25. },  26. {  27. path: "kategorije/:category/:productId",  28. element: <ProductPage />,  29. }  30. ]  31. }  32. ]); |

# Функционалности

Када корисник приступи апликацији на почетној страни су приказани најновији производи у последња 24 сата, ако исти постоје. Са почетне стране корисник има могућност:

-да изабере један од приказаних производа

-пређе на страну за избор категорија

A screenshot of a computer

Description automatically generated -изврши претрагу производа по имену

## Претрага производа

A screenshot of a computer

Description automatically generated Претрага производа се врши тако што се у поље на горњој десној страни странице унесе жељено име за претрагу, и кликом на дугме поред или кликом Enter дугмета на тастатури започињемо претрагу. Када се претрага заврши кориснику се приказују одговарајући производи.

## Избор категорија

A screenshot of a computer

Description automatically generatedКликом на дугме „Proizvodi“ из навигационе траке прелазимо на страну за одабир категорије производа. Кликом на одговарајућу категорију корисник прелази на страну на којој су приказани само производи из те категорије.

## Приказ појединачних производа

Кликом на жељени производ приказује се страна са одређеним информацијама као што су:

-продавнице и цене у датим продавницама

-историја цене производа

A screen shot of a computer

Description automatically generated

# Предлози за унапређење

Једна од првих ствари које је потребно додати је израда администраторског UI-a који треба да олакша креирање нових категорија производа као и само управљање апликацијом.

Што се тиче корисника потребно је имплементирати креирање корисничког налога који ће кориснику омогућити следеће функционалности:

-Остављање коментара испод производа

-Оцењивање производа

-Чување омиљених производа

Пожељно би било и да се креира систем за обавештавање корисника о снижењима његових омиљених производа.

# Литература

|  |  |
| --- | --- |
| [1] | [На мрежи]. https://scrapy.org/. [Последњи приступ 3 3 2024]. |
| [2] | [На мрежи]. https://www.python.org/. [Последњи приступ 3 3 2024]. |
| [3] | [На мрежи]. https://pymongo.readthedocs.io. [Последњи приступ 3 3 2024]. |
| [4] | [На мрежи]. https://schedule.readthedocs.io/en/stable/. [Последњи приступ 3 3 2024]. |
| [5] | [На мрежи]. https://expressjs.com/. [Последњи приступ 3 3 2024]. |
| [6] | [На мрежи]. https://mongoosejs.com/. [Последњи приступ 3 3 2024]. |
| [7] | [На мрежи]. https://www.npmjs.com/package/redis. [Последњи приступ 3 3 2024]. |
| [8] | [На мрежи]. https://www.mongodb.com. [Последњи приступ 3 3 2024]. |
| [9] | [На мрежи]. https://redis.io/. [Последњи приступ 3 3 2024]. |
| [10] | [На мрежи]. https://reactrouter.com/en/main. [Последњи приступ 3 3 2024]. |
| [11] | [На мрежи]. https://www.typescriptlang.org/. [Последњи приступ 3 3 2024]. |

Кратка биографија кандидата

Андрија Лазић је рођен **10.** **априла** **2000**. године у **Крагујевцу**. Основну школу „**Сретен Младеновић**“ у **Десимировцу** **завршио** је **2015**. године, а затим исте године уписује смер **Електротехничар рачунара** у **Првој техничкој школи у Крагујевцу**. Након завршене средње школе уписује основне академске студије на Природно-математичком факултету Универзитета у Крагујевцу, на Институту за математику и информатику, смер информатика.

Универзитет у Крагујевцу

Природно-математички факултет

Институт за математику и информатику

Завршни рад под називом

Сајт за преглед цена ИТ производа

одбрањен је \_\_\_\_\_\_\_\_\_\_\_\_.

МЕНТОР:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

др Владимир Цвjетковић, звање,

Институт за математику и информатику

Природно-математички факултет Крагујевац

ЧЛАНОВИ КОМИСИЈЕ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

др Име Презиме, звање,

Институт за математику и информатику

Природно-математички факултет Крагујевац

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

др Име Презиме, звање,

Институт за математику и информатику

Природно-математички факултет Крагујевац

Завршни рад је оцењен оценом \_\_\_\_\_\_\_\_\_\_\_\_.