|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Марија Кастратовић

GoBooking – портал за туристичку агенцију базиран на микросервисној архитектури

ДИПЛОМСКИ РАД

- Основне академске студије -

Нови Сад, 2022

Садржај

[1 Увод 3](#_Toc111131487)

[2 Теоријске основе 4](#_Toc111131488)

[2.1 Микросервисна архитектура 4](#_Toc111131489)

[2.2 *API Gateway* 6](#_Toc111131490)

[2.3 База по сервису 7](#_Toc111131491)

[3 Коришћене технологије 8](#_Toc111131492)

[3.1 *Go* 8](#_Toc111131493)

[3.2 *Python* и *Flask* 9](#_Toc111131494)

[3.3 *Pharo* 10](#_Toc111131495)

[3.4 *PostgreSQL* 10](#_Toc111131496)

[3.5 *HTTP* протокол 11](#_Toc111131497)

[3.6 *Angular* 12](#_Toc111131498)

[3.7 *Docker* 14](#_Toc111131499)

[4 Имплементација задатка 16](#_Toc111131500)

[4.1 Задатак 17](#_Toc111131501)

[4.1 Архитектура система 18](#_Toc111131502)

[4.2 Микросервиси 19](#_Toc111131503)

[4.2.1 Микросервис за туристичке дестинације 19](#_Toc111131504)

[4.2.2 Микросервис за путовања 20](#_Toc111131505)

[4.2.3 Микросервис за оцене и коментаре 21](#_Toc111131506)

[4.2.4 Микросервис за резервацију путовања 22](#_Toc111131507)

[4.3 Сервис за ауторизацију корисника 22](#_Toc111131508)

[4.4 Сервис за извештаје 23](#_Toc111131509)

[4.5 Клијентска апликација 24](#_Toc111131510)

[4.6 API Gateway 24](#_Toc111131511)

[4.7 Контејнеризација сервиса 26](#_Toc111131512)

[5 Закључак 28](#_Toc111131513)

[6 Литература 29](#_Toc111131514)

[Биографија 30](#_Toc111131515)

# 1 Увод

Тема овог рада је креирање портала за туристичку агенцију – **Go Booking** који је базиран на микросервисној архитектури. Апликацији могу да приступе регистровани корисници који желе да прегледају доступна путовања и резервишу карту за жељено путовање. Такође, корисницима је омогућено оцењивање дестинација као и отказивање резервација. Приступ апликацији имају и администратори система који имају могућност ажурирања података о дестинацијама и путовањима, као и додавање других админа система.

У оквиру апликације постоји неколико пословних микросервиса имплементираних у програмском језику Gо, поред њих постоји и сервис задужен за ауторизацију имплементиран у Flask-у. Клијентска страна имплементирана је у Angular радном окружењу и са сервисима не комуницира директно већ је имплементиран посебан сервис који омогућује ту комуникацију – API Gateway. На самом крају микросервиси су контејнеризовани помоћу специјализованог Docker алата.

Овај рад сачињен је од шест поглавља. Друго поглавље садржи теоријске основе које су неопходне зе разумевање концепта микросервисне архитектуре. У наредном поглавњу описани су програмски језици и технологије коришћене приликом имплементације решења. У оквиру четвртог поглавља представљен је коцепт решења овог задатка. Дат је кратак опис аркитектуре система, представљени су микросервиси у систему као и начин контејнеризације сервиса. Пето поглавље представља резиме до сада урађеног и даљи правци развоја апликације ради њеног побољшања. У последњем поглављу наведен јe списак литературе коришћене приликом писања рада.

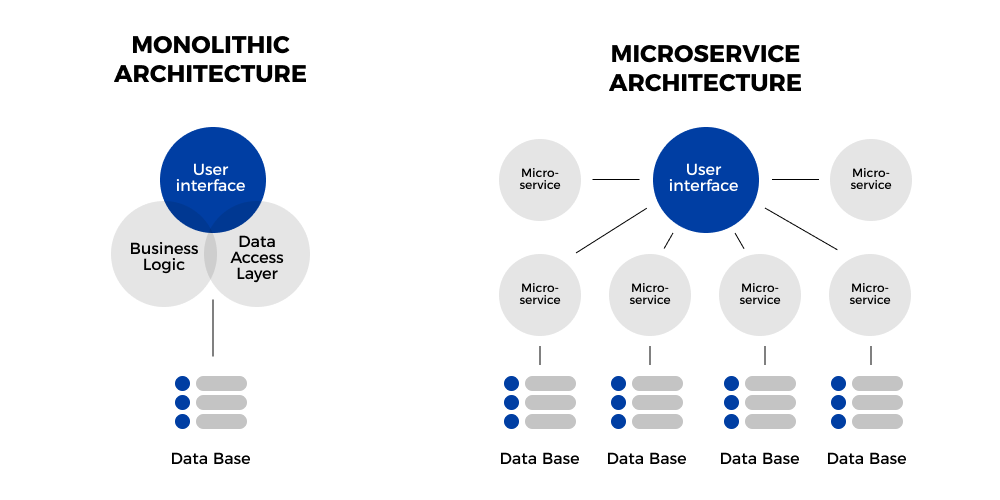
# 2 Теоријске основе

Израда решења захтева предзнање о микросервисима, па ће у овом поглављу бити обрађена ова област. Додатно ће бити обрађена два шаблона (енг. *pattern) API Gateway* и база по сервису, који додатно побољшавају перформансе микросервисне архитектуре.

## 2.1 Микросервисна архитектура

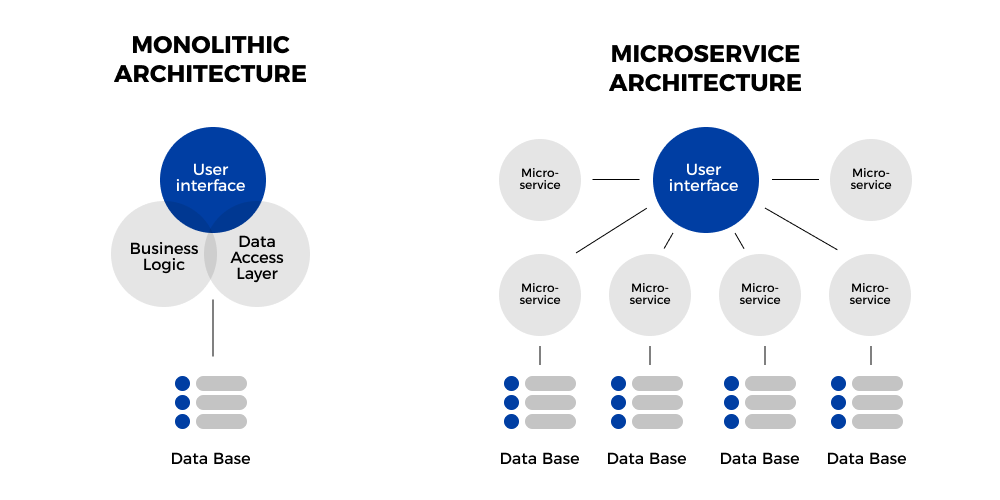
Апликација базирана на микросервисној архитектури је апликација која је прављена као скуп лабаво везаних сервиса који имплементирају функционалности апликације, а сами су независно развијани и независно имплементирани. Сваки од микросервиса покренут је у сопственом процесу и комуницирају преко мреже добро дефинисаним и стандардизованим протоколом, обично HTTP (енг. *Hypertext Transfer Protocol*) протоколом. Микросервиси имају потпуно ограничен контекст - не морају бити свесни никаквих имплементационих детаља и архитектуре других микросервисних модула.

Иако је монолитна архитектура (слика 2.1.1) и данас врло популарна приликом израде веб апликација, она садржи и поједине недостатке који могу послужити као увод у микросервисне архитектуре. Монолитне архитектуре су најчешће сачињене од једне софтверске компоненте која је представљала спрегу између корисничког интерфејса и базе података. Растом и развојем оваквих система јављају се многи проблеми као што су скалирање и одржавање веб апликације, што резултира великом комплексности програмског кода те отежаним будућим одржавањем програмског кода и надоградњом апликације.



Слика 2.1.1 Приказ монолитне архитектуре

Улога микросервисне архитектуре (слика 2.1.2) је да се апликација подели на мање делове, односно сервисе који су задужени за решавање специфичних проблема у апликацији. Ово додатно олакшава имплементацију и одржавање самих сервиса, поделом посла на мање и агилне тимове. На тај начин смањује се време потребно за имплементацију као и број могућих конфликата. Док је код монолитних апликација било неопходно испоручити целу апликацију и приликом најмање измене система, микросервиси могу бити независно испоручивани у продукцију што додатно убрзава сам развојни процес. Још једна предност микросервисне архитектуре у односу на монолитну је отпорност на грешке. За разлику од монолитне архитектуре, микросервисна архитектура би требала моћи поднети пад одређеног сервиса без да та грешка у потпуности сруши читаву апликацију.



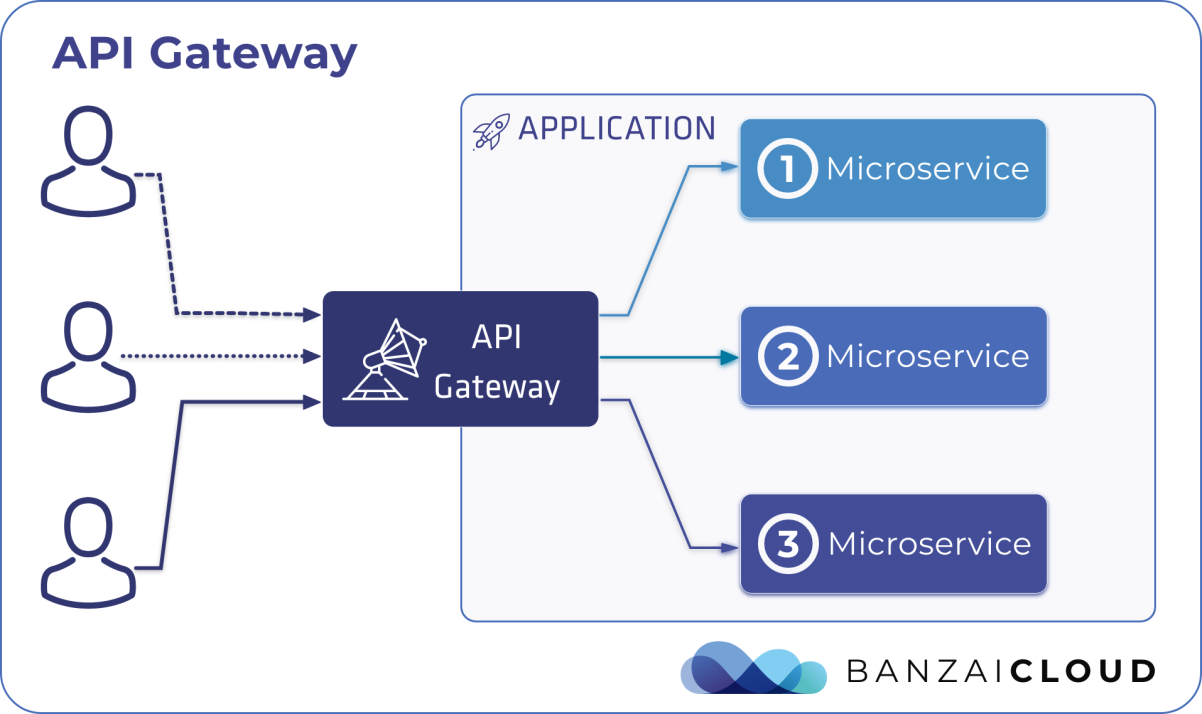
Слика 2.1.2 Приказ микросервисне архитектуре

## 2.2 *API Gateway*

У случају да микросервисну апликацију треба да користе различити клијенти, потребе клијената се могу доста разликовати. Такоће, у великом броју случајева клијенту ће бити потребни подаци од више различитих сервиса. Директан приступ сервисима од стране клијента није добар приступ, јер у великим системима број сервиса може да буде велик па би клијент морао да буде свестан свих сервиса и складишти податке о истим.

Како би се избегли ови проблеми уводи се појам *API* (енг. *Application Programming Interface*) *Gateway pattern* (слика 2.2)који представља улазну тачку у апликацију. Служи као фасада иза које се налазе микросервиси што одаје утисак коришћења једне монолитне апликације. Овај додатни слој представља посебан сервис и једини је којем клијенти имају приступ. Задужен је за рутирање, односно прослеђивање захтева одговарајућим микросервисима.

Још један начин да се додатно унапреди овај шаблон јесте да се креира посебан *API Gateway* сервис за сваку врсту клијента коју апликација опслужује, као што су мобилне апликације и веб апликације. На овај начин додатно прилагођавамо апликацију потребама различитих клијената.



Слика 2.2 Комуникација клијентске стране и микросервиса путем *API Gateway*-a

## 2.3 База по сервису

Код микросервисне архитектуре најпожељнији принцип интеграције са базом података је да сваки од микросервиса има своју базу података коју прилагођава својим потребама. Сваки сервис би требао да има директан приступ само својој бази, а податке из база других сервиса да добија преко њихових *API*-ја.

На овај начин се додатно смањује спрега између микросервиса и побољшава се стабилност система. Уколико један од сервера базе података престане да ради, то неће бити узрок пада читавог система. Применом шаблона база по сервису сваки микросервис има могућност да користи тип складишта података који му највише одговара (релационе или нерелационе базе података).

# 3 Коришћене технологије

У овом поглављу ће бити представљене технологије коришћене приликом имплементације решења.

## 3.1 *Go*

*Go* је програмски језик опште намене, развијен у компанији *Google*. Настао 2007. године, а јавно представљен 2009. године и као такав спада у младе програмске језике. Настао је са циљем да се направи нови језик опште намене, при чему се водило рачуна да се укомбинују лак начин читања и писања као и сигурност језика који се компајлира. Слична синтакса, али знатно поједностављена у односу на програмски језик *C*, омогућила је великом броју програмера да се лако упознају са овим језиком.

*Go* је статички типизирани, компајлирани језик и поседује аутоматско управљање меморијом, односно сакупљање отпада (енг. *garbage-collector*). *Go* није објектно-оријентисан, али преузима неке од концепата објектно оријентисаних језика. У програмском језику *Go*  не постоје класе, концепт наслеђивања, изузеци или анотације. Тиме се доста разликује од многих језика који су додавањем великог броја функционалности постали сувише сложени и често компликовани за рад. Брзо компајлирање као и високе перформансе приликом мрежне комуникације чине *Go* погодним за развој микросервисних апликација. Ово омогућује лакши и бржи развој, као и континуирану испоруку сервиса.

## 3.2 *Python* и *Flask*

### 3.2.1 *Python*

***Python*** је интерпретирани програмски језик високог нивоа настао почетком деведесетих година двадесетог века. Његов творац, холандски програмер Гвидо ван Росум (енг*. Guido van Rossum*) , имао је жељу да направи програмски језик у коме ће акценат бити на читљивом и прегледном коду који ће програмери лако разумети. Подржава процедуралну, објектно-оријентисану и функционалну парадигму програмирања. За програмски језик *Python* развијен је велики број стандардних модула који омогућавају ефикасан рад у многим областима. *Python* подржава динамичке типове, што значи да екплицитно декларисање варијабли није потребно јер програм само одлучује који ће тип придружити којој варијабли.

Иако има лако читљиву и некомплексну синтаксу, *Python* омогућује писање програма за решавање и комплексних проблема. Тренутно спада у једне од најпопуларнијих програмских језика, а најважнији разлози за његову популарност су:

* Једноставна синтакса и веома читљив код.
* Истовремено велике програмске могућности и широка употреба.
* Користе га велике светске организације и компаније, као што су *Google, YouTube, NASA* и *Yahoo.*
* Отвореног је кода, бесплатан и има добру корисничку подршку.

### 3.2.1 *Flask*

***Flask*** развојни оквир (енг. *Microframework* ) написан у програмском језику *Python* и темељи се на *Werkzeug* библиотеци. *Flask* је дизајниран тако да пружа основне услуге али је лако приширив разним надоградњама. *Flask* подржава екстензије које су написане изван самог *Flask*-а, а могу додати могућности апликацијама попут рада с веб формама, приступа базама података, логовања и слично. На тај начин се омогућује кориснику да сам одабере шта ће користити у самој апликацији па самим тим не долази до непотребног гомилања кода.

Главне предности Flask-а су његова флексибилност приликом израде разних врста апликација, једноставан је за одржавање и не захтева пуно процесорске снаге те је једноставан за учење и коришћење. Још једна од предности коришћења *Flask* микровеб развојног оквира је његова скалабилност, односно погодан је за креирање мањих апликација које у будућности имају потенцијал да прерасту у велике и сложене апликације.

## 3.3 *Pharo*

*Pharo* је језик настао по угледу на *Smalltalk* језик са циљем његовог унапређења. Представља прави објекто-оријентисани језик јер је све објекат односно инстанца класе. Модел је динамички типизиран, типови се знају у времену извршавања (енг. *run time*). Методе су јавне и виртуалне, али су сви атрибути заштићени. *Pharo* подржава искључио једноструко наслеђивање. Као што је напоменуто у *Pharu* је све објекат, а тим објектима се управља коришћењем порука. Порука може бити било шта што исказује неку намеру или враћа одређено стање објекта. На поруку објекат одговaра тако што позива одговарајућу методу, а метода описује шта треба урадити када стигне одређена порука. Објекат који прима поруку се назива пријемник (енг. *receiver*).

## 3.4 *PostgreSQL*

*PostgreSQL* или једноставније *Postgres* je врста релационих система за управљање базама података (енг. *RDBMS- Relational Database Management System*), под *open-source* лиценцом (отворени код). Сматра се једном од најпоузданијих база података. Најчешће се користи за веб апликације и веб базе података. *PostgreSQL* садржи моћан објектно-релациони модел података, богат избор врста података, лаку надоградивост, као и надограђени сет наредби *SQL* језика (енг. *SQL - Structured Query Language*).

*SQL* представља структуирани упитни језик за приступ и манипулацију подацима.

* Упитни језик (енг. *Query Language – QL*)
* Језик за манипулацију подацима (енг*. Data Manipulation Language – DML*)
* Језик за дефинисање података (енг. *Data Definition Language – DDL*)

Прва главна предност коришћења *Postgres*-a је то што је *open-source* и може се прилагодити према захтевима програмера. Захтеви за одржавање и подешавање *Postgres* базе података су релативно мали у односу на друге системе за управљање базама података. Такође, *Postgres* је портабилан и преносив са готово свим главним платформама и програмским језицима па је самим тим идеалан за апликације намењене вишеструким платформама.

## 3.5 *HTTP* протокол

*Http* (енг. HyperText Transfer Protocol) представља мрежни протокол који служи за комуникацију између сервера и клијента. Комуникација је заснована на принципу захтев/одговор, при чему је сваки пар захтев/одговор међусобно независан од осталих. Сервер је тај који константно ослушкује захтеве на одговарајућем комуникацијском порту и чека да се клијент повеже и пошаље свој захтев. Захтев који је клијент послао се обрађује на серверу, при ћему се клијенту шаље одговарајући одговор на захтев. *Http* захтев, приказан на слици 3.1, у првом реду садржи податке о методи, путањи и верзији протокола. Затим следе додатни редови који садрже атрибуте облика кључ-вредност. Након тога се налази празан ред и тело поруке које је опционо.



Слика 3.1 Пример HTTP захтева

*Http* дефинише скуп метода захтева за означавање жељене радње коју треба извршити за дати ресурс. Постоји укупно осам *Http* захтева, при чему су најзаступљенији:

* **GET** – Захтева ресурс од веб сервера
* **HEAD** - Захтева само *Http* одговор, без слања самог ресурса
* **POST** – Шаље податке серверу, при чему се најчешће креира нови ресурс
* **PUT** – Замењује тренутну репрезентацију ресурса новом која се налази у захтеву
* **DELETE** – Брише све тренутне репрезентације датог ресурса
* **OPTIONS** – Од сервера тражи списак метода које подржава

Методе HEAD, OPTIONS и GET означене су као сигурне методе јер не врше измене на серверу, већ само имају могућност прегледа ресурса. С друге стране методе POST, PUT и DELETE се сматрају несигурним методама јер могу довести до промена стања сервера.

## 3.6 *Angular*

*Angluar* је радни oквир (енг. *framework*) за развој апликација развијен 2009. године од стране *Google*-a. *Angluar* развојно окружење се користи за креирање клијентског дела веб апликације (енг. *Client-side framework*). Написан је у програмском језику *TypeScript*, развијеном од стране *Microsoft*-а. Његова сврха је да надомести недостатке *JavaScript*-а. Поред строгог типизирања и откривања грешака приликом превођења, најважнија допуна у односу на *JavaScript* јесте увоћење механизма објектно-оријентисаног програмирања. *Angluar* нам омогућује креирање једностраничних апликација (енг. SPA - *Single-Page Application*). Читава апликација смештена је у оквиру једне веб странице са циљем пружања лакше интеракције. Сав потребан код - [*HTML*](https://sr.wikipedia.org/wiki/HTML), *TypeScript* и [*CSS*](https://sr.wikipedia.org/wiki/CSS) преузимају се у оквиру учитавања једне странице. То значи да приликом навигације корисника кроз апликацију нема потребе стално учитавати нове странице већ о навигацији води рачуна рутер (енг*. router*). Приказ архитектуре једностраничне апликације приказан је на слици 3.2.

Слика 3.2 Приказ архитектуре једностраничне апликације

Основни градивни блок *Angluar* аликација су модули. Модул је *TypeScript* класа која групише све функционалности у једну целину. Функционалности које овај радни оквир пружа су компоненте, директиве, филтери, сервиси и модули.

Модул се састоји од компоненти, а једна *Angluar*  компонента обезбеђује део функционалности за апликацију и контролише део екрана који се назива поглед. Компонента се састоји од:

* Класе
* Метаподатака који описују класу и проширују њену функционалност
* Шаблона који се користи за дефинисање *HTML* погледа

Поглед представља низ елемената на страници којима *Angluar* може да приступа и да их модификује. Компоненте користе сервисе, који обезбеђују специфичне функционалности. Сервиси се могу уметнути у компоненте као зависности, при чему доприносе модуларности, чине код поновно искористивим и ефикаснијим.

Један од основних принципа који се веже уз саму компоненту унутар *Angular* оквира је везање података (енг. *Data Binding*).  *Data Binding* је процес прослеђивања података из компоненте ка погледу и обрнуто. Врши се повезивање *DOM* (енг. *Document Object Model*) елемената са својствима класе компоненте .

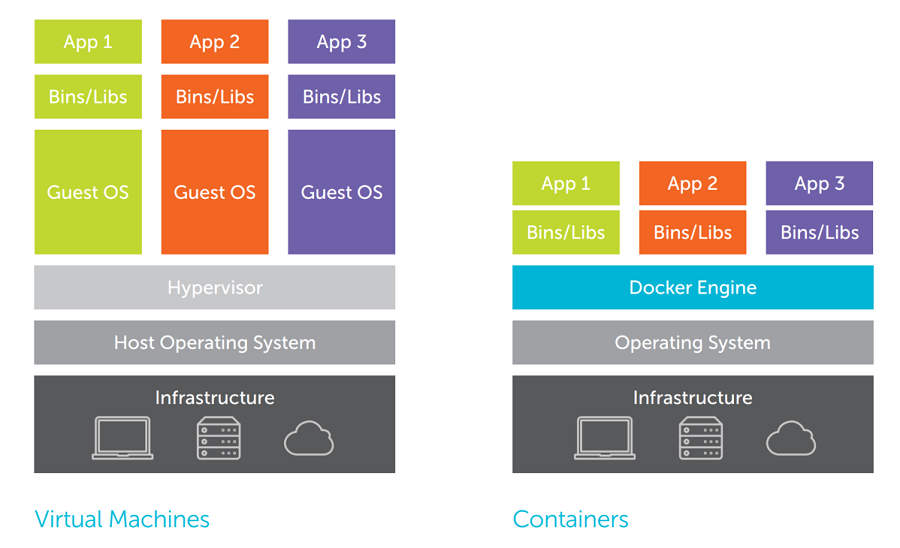
Постоје 4 начина повезивања података у Ангулару:

* Интерполација
* Повезивање својства (енг. *Property binding*)
* Повезивање догађаја (енг. *Event binding*)
* Двосмерно повезивање (енг. *Two Way Binding*)

## 3.7 *Docker*

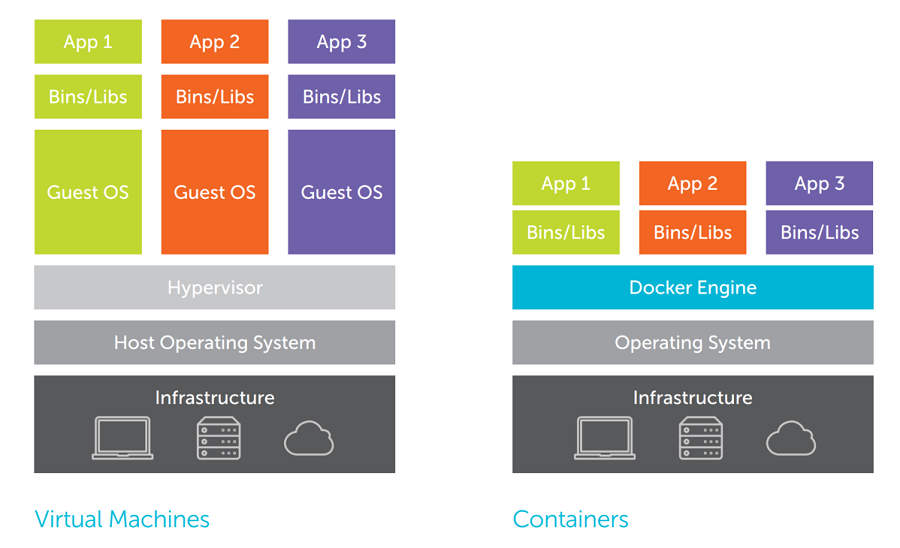
*Docker* је *open-source* платформа дизајнирана да олакша имплементацију, развој и покретање апликација користећи контејнере. Омогућује да се код апликације упакује у контејнере, заједно са свим библиотекама и зависностима неопходним за покретање апликације у било ком окружењу. Контејнери могу међусобно да комуницирају помоћу дефинисаних канала.

Приступ заснован на *Docker*-у функционише слично виртуелној машини. Код традиционалне виртуелизације (слика 3.3), свака појединачна виртуелна машина укључује пуну копију оперативног система, апликације, библиотеке при чему се испод њих налази *hypervisor* који омогућује креирање, покретање и извршавање више виртуелних машина на једном физичком рачунару и омогућује дељење физичких ресурса између њих.



Слика 3.3 - Илустрација више виртуелних машина је једном оперативном систему

За разлику од класичне виртуализације, на једном серверу је могуће извршавати више контејнера истовремено, али све мора да покреће исти оперативни систем (слика 3.4). Поставка оваквог система је знатно једноставнија у односу на виртуелне машине јер је неопхондно да на серверу постоји само основни оперативни систем и софтвер који омогућује употребу контејнера, односно у овом конкретном случају *Docker Engine.* Оно што омогућује овакву техологију је *kernel Linux* оперативног система.Управљање ресурсима у контејнерима се успоставља помоћу контролне групе (енг. *Control groups*) и простора имена (енг. *Namespaces*). Простори имена нам обезбеђују да процеси буду међусобно изоловани, док нам контролне групе омогућују алоцирање ресурса између процеса.



Слика 3.4 Илустрација контејнера је једном оперативном систему

# 4 Имплементација задатка

У овом поглављу ће бити представљен текст задатка, биће приказана архитектура система као и начин имплементације решења.

## 4.1 Задатак

**Goo Booking** представља портал за туристичку агенцију базирану на микросервисној архитектури. Апликацију могу да користе два типа регистрованих корисника, администратор система и регуларни корисници. Поред регистрованих корисника, апликацију могу да користе и нерегистровани корисници како би извршили регистрацију и пријаву на систем.

Функционалности **нерегистрованог корисника**:

* Има могућност уноса личних података с циљем регистрације на систем
* Након успешне регистрације има могућност да се улогује на систем са корисничким именом и лозинком

Функционалности **регуларног корисника**:

* Прегледа доступне дестинације, при чему види детаљне информације о свакој дестинацији и оцене других корисника
* Оцењује доступне дестинације
* Прегледа, додаје и брише коментаре за одабрану дестинацију
* Прегледа доступна путовања за одабрану дестинацију
* Резервише путовања за одабрани датум и број седишта
* Прегледа своје резервације
* Отказује своје резервације

Функционалности **администратора система**:

* Додаје нове администраторе система
* Додаје нове дестинације, мења податке на постојећим дестинацијама и има могућност брисања дестинација
* Додаје нова путовања, мења податке на постојећим путовањима и има могућност брисања путовања
* Прегледа све резравације и има могућност отказивања резервација
* Има увид у коментаре и оцене за сваку дестинацију

## 4.1 Архитектура система

Систем је имплементиран као микросервисна апликација. Апликацију чине четири микросервиса која су имплементирана у програмском језику *Go,* један *Flask* сервис за аутентификацију корисника и *Angular* клијентска апликација. Сваки сервис има своју посебну *PostgreSQL* базу података. На слици 4.1 је приказан дијаграм архитектуре система. За сваки од сервиса је урађена контејнеризација помоћу *Docker* алата.



Слика 4.1 - Дијаграм архитектуре система

## 4.2 Микросервиси

У оквиру система постоје четири микросервиса, а то су:

* Микросервис за туристичке дестинација
* Микросервис за путовања
* Микросервис за оцене и коментаре
* Микросервис за резервацију путовања

Сви микросервиси имплементирани су у програмском језику *Go.* Сваки од микросервиса повезан је на посебну *PostgreSQL* базу података.

Помоћу ***net/http***пакета из стандардне библиотеке, сваки од сервиса покреће сопствени *Http* сервер. Руте се конфигуришу помоћу ***gorilla/mux***пакета који додатно олакшава дефинисање рута и *handler*-а за њих. Такође, уз помоћ пакета ***log*** из стандардне библиотеке омогућено је да се у логове запишу све битне информације и евентуалне грешаке приликом рада.

### 4.2.1 Микросервис за туристичке дестинације

Микросервис за дестинације садржи низ *endopoint-*а који омогућују администраторима система и регуларним корисницима приступ информацијама о доступним дестинацијама. Администратор система поред тога има и могућност додавања, измене и брисања дестинација. Листа свих *endpoint*-a овог микросервиса приказана је у табели 4.1.

Ентитет којим рукује овај микросервис је:

* **Дестинација** – Једну дестинацију описују подаци као што су назив и опис дестинације, држава у којој се дестинација налази као и просечна оцена дестинације.

|  |  |  |
| --- | --- | --- |
| Путања | Метода | Функционалност |
| /destinations | GET | Враћа све доступне дестинације |
| /destination/[id] | GET | Враћа једну конкретну дестинацију |
| /addDestination | POST | Додаје нову дестинацију |
| /updateDestination/[id] | PUT | Ажурира податке о дестинацији |
| /updateDestinationAverage/[id] | PUT | Ажурира се просечна оцена дестинације |
| /delete/[id] | DELETE | Брише се дестинација |

Табела 4.1- Листа *endpoint-*а микросервиса за дестинације

### 4.2.2 Микросервис за путовања

Микросервис за путовања служи да прикаже број доступних места за одређено путовање, као и да пружи додатне информације о путовању. Регуларни корисник има могућност прегледа свих доступних путовања за одабрану дестинацију и датум, као и да види број преосталих слободних места. Администратор система има могућност додавања нових путовања, као и измену и брисање постојећих путовања. Листа свих *endpoint*-a овог микросервиса приказана је у табели 4.2.

Ентитет којим рукује овај микросервис је:

* **Путовање** – Овај ентитет описују подаци као што су наслов путовања, опис, датум, цена и број преосталих слободних места. Такође, садржи референце на конкретну дестинацију на које се путовање односи у виду глобалног *ID*-ја дестинације.

|  |  |  |
| --- | --- | --- |
| Путања | Метода | Функционалност |
| /getTravels | GET | Враћа сва доступна путовања |
| /addTravel | POST | Додаје ново путовање |
| /travels | PUT | Ажурира путовања уколико дође до измене на дестинацији |
| /travelsByDestination/[id] | GET | Враћа листу свих путовања за дестинацију |
| /travel/[id] | GET | Враћа једно конкретно путовање |
| /updateTravel/[id] | PUT | Ажурира податке о путовању |
| /delete/[id] | DELETE | Брише се путовање |

Табела 4.2- Листа endpoint-а микросервиса за путовања

### 4.2.3 Микросервис за оцене и коментаре

Микросервис за оцене и коментаре омогућује корисницима да оцењују и коментаришу дестинације које су посетили. Такође, корисник може да обрише свој коментар. Поред тога омогућује приказ просечних оцена свих доступних дестинација. Листа свих *endpoint*-a овог микросервиса приказана је у табели 4.3.

Ентитети којима рукује овај микросервис је:

* **Оцена** – Садржи вредност оцене коју је корисник дао за дату дестинацију, као и референце на конкретну дестинацију и корисника који је оставио оцену
* **Коментар** – Садржи текст коментара који је корисник оставио за дату дестинацију, као и референце на конкретну дестинацију и корисника који је оставио коментар

|  |  |  |
| --- | --- | --- |
| Путања | Метода | Функционалност |
| /rating/[id] | GET | Враћа просечну оцену дестинације |
| /addRating | PUT | Додаје оцену за дестинацију |
| /addComment | POST | Додаје коментар за дестинацију |
| /comments/[id] | GET | Враћа све коментаре за конкретну дестинацију |
| /deleteComment/[id] | DELETE | Брише се коментар |

Табела 4.3 - Листа endpoint-а микросервиса за оцене и коментаре

### 4.2.4 Микросервис за резервацију путовања

Микросервис за резервацију путовања омогућује корисницима да резервишу путовање за жељени датум и број места. Омогућује кориснику приказ свих његових резервација као и отказивање резервације. Администратор система има могућност прегледа свих резервација као и отказивање резервација. Листа свих *endpoint*-a овог микросервиса приказана је у табели 4.4.

Ентитет којим рукује овај микросервис је:

* **Резервација** – Овај ентитет описују подаци као што су датум, број места за које се резервише путовање као и референце на конкретно путовање и корисника који је извршио резервацију.

|  |  |  |
| --- | --- | --- |
| Путања | Метода | Функционалност |
| /addReservation | POST | Додаје нову резервацију |
| /getReservations | GET | Враћа све резервације |
| /getReservationsByUser/[id] | GET | Враћа све резервације за конкретног корисника |
| /deleteReservation/[id] | DELETE | Брише се резервација |

Табела 4.4 - Листа endpoint-а микросервиса за резервације путовања

## 4.3 Сервис за ауторизацију корисника

Сервис за ауторизацију корисника имплементиран је у програмском језику *Python*, са ослонцем на *Flask* радни оквир. Омогућује регистрацију нових корисника, као и пријаву на систем. Администратору система омогућује додавање нових администратора. Након успешне пријаве на систем креира се *JWT* (енг*. JSON Web Token*) токен. Овај токен садржи податке о кориснику, који се шаље клијенту ради касније аутентификације корисника. Листа свих *endpoint*-a овог сервиса приказана је у табели 4.5.

Ентитет којим рукује овај сервис је:

* **Корисник** – Овај ентитет описују подаци као што су име и презиме, корисничко име, *email* адреса и лозинка.

|  |  |  |
| --- | --- | --- |
| Путања | Метода | Функционалност |
| /sign-in | POST | Омогућује пријаву на систем |
| /sign-up | POST | Омогућује регистрацију новог регуларног корисника |
| /current-user | GET | Враћа тренутно улогованог корисника |
| /add-admin | POST | Додаје новог админа система |
| /[id] | GET | Враћа једног конкретног корисника |
| /registered-users | GET | Враћа све регистроване кориснике |
| /admins | GET | Враћа све администраторе система |

Табела 4.5 - Листа *endpoint-*а сервиса за ауторизацију корисника

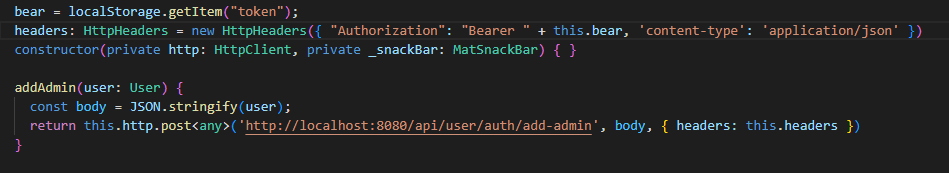
## 4.4 Сервис за извештаје

Администратор система има могућност генерисања извештаја као што су капацитет попуњености седишта за путовања, најбоље оцењиване дестинације и слично. Овај сервис имплементиран је у *Pharo* програмском језику. Подаци за обраду добављају се слањем *HTTP* захтева преко *API Gateway*-а различитим микросервисима у систему. Подаци који стигну се обрађују и графички приказују на дијаграму. За приказ извештаја коришћен је *RTGrapher* из *Roassal*-а. *Roassal* представља алат који омогућује визуелизацију података у различитим форматима као што су *JSON*, *CSV* и *XML* формат.

## 4.5 Клијентска апликација

Клијентска апликација написана је у *Angular* радном оквиру. Како би се олакшало руковање апликацијом, странице као и садржај на страницама се динамички мењају помоћу фајла ***app-routing.module.ts.*** У фајлу ***app.module.ts*** као и у самој *app* компоненти скупља се цела апликација и та компонента представља чвориште за све остале компоненте које се налазе у оквиру апликације.

Комуникација са серверском страном врши се преко *HTTP* протокола. Приликом успешне пријаве на систем, која се одвија помоћу *POST* методе, са серверске стране се пошаље објекат у којем се налази токен. Тај токен се складишти и касније се шаље у оквиру наредног захтева (листинг 4.1) како би методе на серверској страни знале који је корисник тренутно пријављен на систем.



Листинг 4.1 – Пример слања *JWT* токена у оквиру заглавља *HTTP* захтева

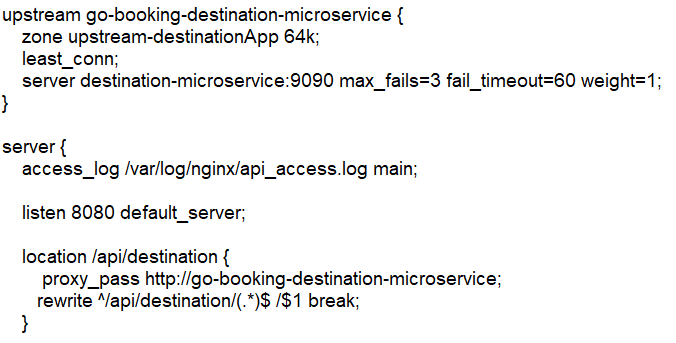
## 4.6 API Gateway

За имплементацију *API Gateway-*а коришћен је веб сервер отвореног кода ***NGINX****.* Конфигурација почиње главним конфигурационим фајлом, **nginx.conf**. За укључивање *API Gateway* конфигурације користи се *include* директива у *http* блоку. Ова директива референцира главни фајл који садржи *gateway* конфигурацију, **api\_gateway.conf**. Садржај *nginx.conf* фајла приказан је у листингу 4.2.



Листинг 4.2 – Главни конфигурациони фајл *nginx.conf*

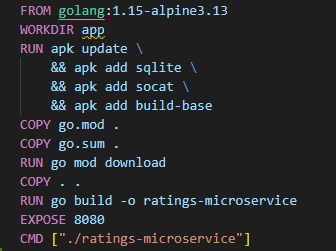
*API gateway* конфигурациони фајл, приказан у листингу 4.3, дефинише виртуелни сервер који омогућује клијентима коришћење *NGINX-*а као *gateway-*ја. Омогућује приступ свим *API*-јима путем једне приступне тачке, у овом примеру сервер слуша на порту 8080. У оквиру *server* блока дефинисани су сви *URI-*ји, који се помоћу *proxy\_pass* директиве рутирају до одговарајуће *upstream* групе*.* У оквиру *upstream* групе за сваки сервис се користе парови адреса-порт како би се дефинисало где је *API deploy -* ован.



Листинг 4.3 – Конфигурациони фајл за *API Gateway*, *api\_gateway.conf*

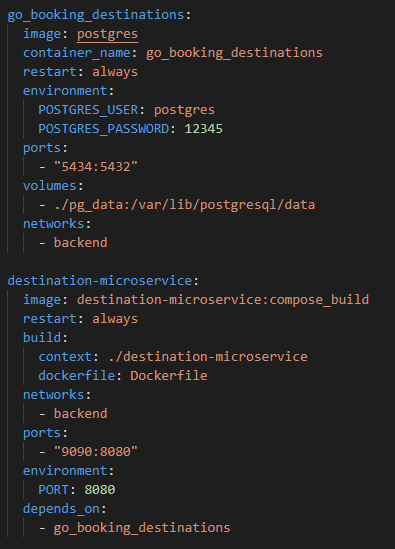
## 4.7 Контејнеризација сервиса

Контејнеризација сервиса извршена је помоћу *Docker* алата. Сваки микросервис упакован је у посебан контејнер. За сваки сервис биће изграђена *Docker* слика (енг. *Docker image) на основу Dockerfile*-ова који се налазе у њиховим конкретним директоријумима. Покретањем слике добија се активни контејнер за тај сервис. Поред микросервиса, посебна *Docker* слика се креира и за клијентску апликацију као и за *Api Gateway*. Пример *Dockerfile*-a приказан је у листингу 4.4.



Листинг 4.4 *Dockerfile* за *Go* микросервис

За оркестрацију контејнера коришћен је *Docker compose* алат. Оркестрација се конфигурише у оквиру **docker-compose.yml** фајла и у њему се дефинишу сви сервиси који треба да буду покренути. У оквиру овог фајла се дефинише мрежа на којој ће се сви контејнери налазити, и то је мрежа типа *bridge*. Сви сервиси који се налазе на истој мрежи могу међусобно да се проналазе и да комуницирају на основу *host name*-a. Пример *docker-compose.yml* фајла приказан је у листингу 4.5.



Листинг 4.5 Конфигурација *docker-compose.yml*

# 

# 5 Закључак

У оквиру овог рада представљена је апликација за туристичку агенцију – GoBooking базирана на микросервисној архитектури. На самом почетку је дат кратак увод у микросервисну архитектуру. Описане су врсте корисника који имају приступ апликацији као и микросервиси од којих је сачињена апликација и њихове функционалности. Приказане су технологије *Go* и *Python* коришћене за имплементацију микросервиса, као и *Angular* радни оквир за имплементацију корисничког интерфејса. Укратко је објашњен начин комуникације клијентске и серверске стране путем *API Gateway*  веб сервера. На самом крају је приказана контејнеризација микросервиса као и њихова оркестрација коришћењем *Docker* алата.

**6 Литература**

1. Salah, Tasneem, et al. "The evolution of distributed systems towards microservices architecture." *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, 2016.
2. Microservices, <https://martinfowler.com/articles/microservices.html>
3. Al-Debagy, Omar, and Peter Martinek. "A comparative review of microservices and monolithic architectures." *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*. IEEE, 2018.
4. [https://miro.medium.com/max/1400/1\*b5vneT\_J4-dKejbYH4o5qg.png](https://miro.medium.com/max/1400/1*b5vneT_J4-dKejbYH4o5qg.png)
5. Pattern:API Gateway, <https://microservices.io/patterns/apigateway.html>
6. The API Gateway Pattern in Microservices, <https://jstobigdata.com/architecture/the-api-gateway-pattern-in-microservices/>
7. <https://banzaicloud.com/blog/backyards-api-gateway/api-gateway.png>
8. The database per service pattern, <https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/database-per-service.html>
9. The database per service pattern, <https://medium.com/design-microservices-architecture-with-patterns/the-database-per-service-pattern-9d511b882425>
10. Meyerson, Jeff. "The go programming language." *IEEE software* 31.5 (2014): 104-104.
11. Go documentation, <https://go.dev/doc/>
12. Go (programming language), <https://en.wikipedia.org/wiki/Go_(programming_language)>
13. Golang microservices, <https://madappgang.com/blog/golang-microservices/>
14. Miškovic, Vladislav. "Osnove programiranja-Python." *Univerzitet Singidunum*  (2017)
15. Python, <https://en.wikipedia.org/wiki/Python_(programming_language)>
16. Aggarwal, Shalabh. *Flask framework cookbook*. Packt Publishing Ltd, 2014.
17. Flask documentation, <https://flask.palletsprojects.com/en/2.1.x/foreword/>
18. Pharo, <http://www.igordejanovic.net/courses/tech/Pharo/>
19. The Pharo MOOC, <https://mooc.pharo.org/>
20. Postgres documentation, <https://www.postgresql.org/about/>
21. SQL, <https://www.techtarget.com/searchdatamanagement/definition/SQL>
22. SQL, <https://en.wikipedia.org/wiki/SQL>
23. <https://www.webdevdrops.com/http-primer-for-frontend-developers-f091a2070637/images/http-2-1024x484.png>
24. HTTP, <https://developer.mozilla.org/en-US/docs/Web/HTTP>
25. HTTP, <https://www.ibm.com/docs/en/cics-ts/5.2?topic=concepts-http-protocol>
26. Angular documentation, <https://angular.io/docs>
27. <https://www.pngwing.com/en/free-png-mewhn>
28. Jadhav, Madhuri A., Balkrishna R. Sawant, and Anushree Deshmukh. "Single page application using angularjs." *International Journal of Computer Science and Information Technologies* 6.3 (2015): 2876-2879.
29. Docker documentation, <https://docs.docker.com/>
30. <https://cdnssinc-prod.softserveinc.com/img/blog/containers-security-virtual-machines.PNG>
31. Potdar, Amit M., et al. "Performance evaluation of docker container and virtual machine." *Procedia Computer Science* 171 (2020): 1419-1428.

# Биографија

Марија Кастратовић рођена је 4. новембра 1998. године у Новом Саду. Основну школу „Жарко Зрењанин“ завршила је 2013. године као одличан ђак. Исте године уписује гимназију „Светозар Марковић“ у Новом Саду, општи смер. 2017. године завршава гимназију са одличним успехом и уписује Факултет техничких наука Универзитета у Новом Саду. Полаже све испите предвиђене планом студијског програма.