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

Ивана Савин

**ВЕБ АПЛИКАЦИЈА ЗA УПРАВЉАЊЕ ИМЕЈЛ ПОРУКАМА**

МАСТЕР РАД

- Мастер академске студије -

Нови Сад, 2019

|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ МАСТЕР (MASTER) РАДА** | Лист/Листова: |
|  |

*(Податке уноси предметни наставник - ментор)*

| Студијски програм: | **Софтверско инжењерство и информационе технологије** |
| --- | --- |
| Руководилац студијског програма: | **Проф. др Мирослав Зарић** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | **Ивана Савин** | Број индекса: | **R1 17/18** |
| Област: | **Веб програмирање** | | |
| Ментор: | **Проф. др Милан Видаковић** | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ  И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА  ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * проблем – тема рада; * начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; * литература | | | |

**НАСЛОВ МАСТЕР (MASTER) РАДА:**

|  |
| --- |
| **Веб апликација за управљање имејл порукама** |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| Задатак рада представља развој једноставне веб апликације за управљање електронском поштом инспирисану већ постојећим решењем (GMail). Клијентски део апликације ће бити развијен помоћу претходно развијене JavaScript библиотеке за креирање компоненти корисничког интерфејса. Cерверски део апликације ће бити реализован у програмском језику Јава, коришћењем Spring окружења и комуницираће са Gmail API-јем. Апликација омогућава основни скуп операција које нуди и оригинално решење попут управљања електронском поштом, креирања и слања поште. Спецификација ће бити представљена UML дијаграмима. |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
|  |  |

|  |
| --- |
| Примерак за:   Студента;   Ментора |

Образац **Q2.НА.15-04** - Издање 1

SADRŽAJ

[1. Uvod 3](#_Toc8736786)

[2. Opis korišćenih tehnologija 5](#_Toc8736787)

[2.1 OldSchoolComponents biblioteka 5](#_Toc8736788)

[2.1.1 Komponente 5](#_Toc8736789)

[2.1.2 Kontejneri 6](#_Toc8736790)

[2.2 Google Gmail API 6](#_Toc8736791)

[2.2.1 Podešavanje Google Gmail API-ja 6](#_Toc8736792)

[2.2.2 API pregled 7](#_Toc8736793)

[2.2.3 Autentifikacija i autorizacija 9](#_Toc8736794)

[2.2.4 Tipičan proces rada sa Gmail API servisom 10](#_Toc8736795)

[2.3 Spring okruženje 10](#_Toc8736796)

[3. Specifikacija aplikacije 12](#_Toc8736797)

[3.1 Dijagram slučajeva korišćenja 13](#_Toc8736798)

[4. Opis implementacije 14](#_Toc8736799)

[4.1 Serverski deo 14](#_Toc8736800)

[4.1.1 Controller 14](#_Toc8736801)

[4.1.2 Service 23](#_Toc8736802)

[4.2 Klijentski deo 27](#_Toc8736803)

[5. Zaključak 33](#_Toc8736804)

[6.KLJUČNA DOKUMENTACIJSKA INFORMACIJA 37](#_Toc8736805)

[7.KEY WORDS DOCUMENTATION 39](#_Toc8736806)

# Uvod

Zadatak rada predstavlja kreiranje JavaScript [1] biblioteke koja će omogućiti jednostavno kreiranje komponenti korisničkog interfejsa i primena te biblioteke za razvoj jednostavne aplikacije. Kao primer za primenu biblioteke kreirana je aplikacija za upravljanje elektronskom poštom inspirisana već postojećim rešenjem (Google Mail) [2].

Budući da je bilo potrebno razviti veb aplikaciju, posebno su razvijeni klijentski i serverski deo, kao i izabrati pogodne tehnologije za implementaciju. Klijentski deo aplikacije je razvijen korišćenjem prethodno implementirane biblioteke dok je serverski deo aplikacije Spring [3] aplikacija koja komunicira sa Gmail API-jem [4].

Prvo poglavlje je uvodno, dok se u drugom nalazi opis izabranih tehnologija koji su korišćeni pri izradi. Treće poglavlje predstavlja detaljnu specifikaciju aplikacije kroz UML dijagrame. Četvrto poglavlje predstavlja opis konkretne implementacije veb aplikacije, dok je u petom poglavlju iznet zaključak.

# Opis korišćenih tehnologija

Za implemetaciju klijentskog dela aplikacije korišćena je prethodno kreirana JavaScript biblioteka za kreiranje komponenti korisničkog interfejsa, *jQuery* biblioteka [5], *Bootstrap* biblioteka za dizajn komponenti [6] i *axios* biblioteka za komunikaciju sa serverom [7].

Serverski deo aplikacije je kreiran zarad lakše komunikacije sa Gmail API-jem. Implementiran je u programskom jeziku Java [8], verzije 1.8, korišćenjem Spring okruženja. Radi pojednostavljenog korišćenja Spring okruženja korišćen je Spring Boot [9] razvojni okvir.

## OldSchoolComponents biblioteka

Komponentama grafičkog interfejsa je moguće rukovati kao objektima zahvaljujuci ECMAScript 6....

Hijerarhija grafičkih komponenti počinje sa komponentom Component i nju nasleđuju sve komponente. Osnovna komponenta sadrži atribute potrebne za grafičko isrtavanje jedne HTML komponente a to su *id* atribut za identifikaciju, *CSSclass* atribut za .. , i node atribut koji predstavlja vezu sa DOM reprezentacijom komponente.

### ECMAScript 6

ECMAScript 6 ili ECMAScript 2015 je unapređena verzija JavaScript-a koja je uvela značajna unapređenja i olakšanja u radu sa JavaScript-om. Značajno unapređenje koje nudi ECMAScript 6 jeste mogućnost kreiranje JavaScript klasa. Kreiranje objekata i njihovo nasleđivanje je sada moguće na mnogo lakši način nego pre postojanja klasa. ES6 klase nude objektno orijentisani model nasleđivanja iako je u pozadini JavaScript prototipsko nasleđivanje. Objektno orijentisani pristup je lakši i jednostavniji za korišćenje i sa ES6 klasama se to postiže i u JavaScript-u.

Jedan način kreiranja klase je korišćenjem class deklaracije, tj. navođenjem ključne reči *class* i imena klase. Primer deklaracije klase se može videti na slici broj.

class Rectangle {

constructor(height, width) {

this.height = height;

this.width = width;

}

}

Slika 2.1 Kreiranje klase pomoću class deklaracije

Drugi način kreiranja klase je korišćenjem class izraza, kao na slici broj.

let Rectangle = class {

constructor(height, width) {

this.height = height;

this.width = width;

}

};

Slika 2.1 Implementacija constructor metode

U projektu klase su kreirane pomoću class deklaracija. ES6 klase su ustvari sintaktički šećer jer je iza njih JavaScript prototipsko nasleđivanje.

### Komponente

#### 2.1.2.1 Constructor

Constructor metoda je specijalana metoda zadužena za kreiranje i inicijalizaciju objekta kreiranih sa ključnom rečju *class*. Prilikom kreiranja objekta klase Component i svih objekata koji nasleđuju ovu klasu kreira se objekat koji ima identifikator i css atribut. Takođe, prilikom kreiranja objektne reprezentacije komponente kreira se i privremeni element DOM stabla.

#### 2.1.2.2 toHTML metoda

Metodu toHTML() implementira svaka klasa i ona vraća string reprezentaciju kreirane HTML komponente. Primer toHTML metode se može videti na slici broj.

Ostale komponente koje nasleđuju osnovnu komponentu Component imaju mogućnost dodavanja specifičnih atributa i metoda za tu komponentu. Jedan primer je klasa InputField koja predstavlja komponetu za unos podataka. Njena implementacija se može videti na slici broj a pored osnovnih atributa ona sadrži i atribute *type* i *placeholder*.

### Kontejneri

Pored osnovnih komponenti prikaza postoje i komponente koje unutar sebe mogu imati neku drugu komponentu. One su specijalizacije klase Component a njihova nad klasa je klasa Container. Klasa Container pored metoda koje ima klasa Component sadži i neke dodatne.

#### Constructor

Conteiner objekat se razlikuje od Component objekta po tome što može sadržati druge Component objekte i samim tim se njegov konstruktor razlikuje. Prilikom kreiranja Container objekta postavlja se atribut *children* koji predstavlja mapu svih komponenti koje taj Container sadrži. Na slici broj može se videti implementacija ovog konstruktora.

constructor(id, CSSclass) {

super(id, CSSclass);

this.children = new Map();

}

Slika 2.1 Implementacija constructor metode

#### Add

Da bi se komponenta prikazala potrebno je dodati je u DOM stablo. Komponenta ne može postojati ukoliko nije dodata u neki Container objekata. Iz tog razloga postoji add metoda koja dodaje komponentu u određeni kontejner na nivou objektne reprezentacija i na nivou DOM stabla zamenjuje privremenu komponentu. Takođe, u ovoj metodi se poziva metoda addListeners koja ce za komponentu dodati sve akcije koje se mogu desiti. Implementacija add metode se može videti na slici broj.

add(component) {

this.children.set(component.id, component);

component.setParent(this);

$('#' + component.id).remove();

$('#' + this.id).append(component.tohtml());

component.node = document.getElementById(component.id);

for (var child of this.children.values()) {

this.addListeners(child);

}

}

Slika 2.1 Implementacija add metode

#### remove

Ukoliko želimo dinamičan prikaz komponenti, bez ponovnog učitavanja kompletnog interfejsa možemo neku komponentu obrisati i na njeno mesto umetnuti novu. Iz tog razloga kreirana je remove metoda koja istovremeno brise objektnu reprezentaciju komponente i briše komponentu iz DOM stabla. Imtementacija ove metode se može videti na slici broj.

remove(component) {

component.children.delete(this.id);

$('#' + this.id).remove();

}

Slika 2.1 Implementacija remove metode

#### toHTML

#### addListeners

## Google Gmail API

Google Gmail API je skup funkcija razvijen od strane Google-a kako bi omogućio drugim aplikacijama da komuniciraju sa Google servisima i da ih integrišu u svoja rešenja. Druge aplikacije mogu koristiti Google Gmail API kako bi poboljšale ili proširile svoja rešenja. [] Implemetacije Google Gmail API-ja postoje u programskim jezicama Java, JavaScript, .NET, Objective C, PHP i Python.

### Podešavanje Google Gmail API-ja

Kao što je već spomenuto, serverski deo aplikacije je kreiran radi lakše komunikacije sa Google Gmail API-jem. Google Gmail API nudi RESTful pristup funkcionalnostima. Za većinu veb aplikacija Gmail API je najbolji izbor jer nudi autorizovan pristup korisnikovim Gmail podacima. U Google Gmail API-ju svaka funkcionalnost predstavlja poseban resurs. Neke od funkcionalnosti koje nudi Gmail API su:

* pregled elektronske pošte,
* manipulisanje elektronskom poštom,
* kreiranje nove pošte,
* slanje pošte,
* upravljanje labelama.

Radi lakšeg kreiranja API zahteva korisnik može izabrati neku od gotovih implementacija Gmail API-ja. U ovom projektu je korišćena Java biblioteka.

Biblioteka je jednostavna za korišćenje i dobro dokumentovana. U Spring projekat se može uvesti preko *maven* [] zavisnosti.

<dependency>

<groupId>com.google.apis</groupId>

<artifactId>google-api-services-gmail</artifactId>

<version>v1-rev65-1.18.0-rc</version>

</dependency>

Slika 2.1 Importovanje Gmail API biblioteke u Spring projekat

Nakon što je biblioteka uvezena u projekat može se koristiti. Gmail API pre korišćenja zahteva autentifikaciju i autorizaciju. [wiki] Google API koristi OAuth 2.0 protokol [] za autentifikaciju. OAuth2 protokol je jednostavan protokol za autentifikaciju, zahteva postojanje *clientId* i *clientSecret* kredencijala. Ove kredencijale je moguće dobiti nakon što se za određeni nalog omogući Gmail API. Postupak dobijanja kredencijala je jednostavan i moguć prolaskom kroz sledeće podešavanje <https://console.developers.google.com/flows/enableapi?apiid=gmail&pli=1>.

### API pregled

Gmail API je veb servis koji koristi RESTful način komunikacije a resursi su predstavljeni u JSON formatu. Resursi koje nudi Google Gmail API su:

* *message* – resurs koji predstavlja poruku, nije promenljiv. Poruka može biti kreirana ili obrisana ali se ni jedno svojstvo poruke ne može izmeniti,
* *draft* – resurs koji predstavlja poruku koja je kreirana ali nije poslana. Draft resurs je vezan za jedan *message* resurs koji može biti izmenjen. Nakon slanja, *draft* resurs se briše i resursu se dodaje sistemska labela SENT
* *label* – resurs koji služi kao sredstvo kategorizacije poruka. Label resurs ima *many-to-many* vezu [] sa porukama. Na jednu poruku može biti primenjeno više labela i jedna labela može biti primenjena na više poruka. Postoje dva tipa labela: system i user. System labele poput INBOX, TRASH, SPAM su automatski kreirane i ne mogu biti izmenjene ili izbrisane. User labele mogu biti dodate od strane korisnika ali i od strane sistema.,
* *history* – resurs koji predstavlja kolekciju izmenjenih poruka u hronološkom redosledu,
* *thread* – resus koji predstavlja kolekciju poruka koje predstavljaju konverzaciju. Kao i poruke i kolekcije poruka mogu imati labele primenjene nad njima dok za razliku od poruka, kolekcija poruka ne može biti kreirana već samo obrisana.,
* *settings* – resurs koji pruža mogućnost kontrole na Google nalogom.

Ključni resursi za razvoj aplikacije su *message* i *label* resursi. Na slici broj je prikazana JSON reprezentacija message resursa, a na slici broj je prikazana reprezentacija label resursa.

{

"id": string,

"threadId": string,

"labelIds": [

string

],

"snippet": string,

"historyId": unsigned long,

"internalDate": long,

"payload": {

"partId": string,

"mimeType": string,

"filename": string,

"headers": [

{

"name": string,

"value": string

}

],

"body": users.messages.attachments Resource,

"parts": [

(MessagePart)

]

},

"sizeEstimate": integer,

"raw": bytes

}

Slika 1.1 JSON reprezentacija *message* resursa

{

"id": string,

"name": string,

"messageListVisibility": string,

"labelListVisibility": string,

"type": string,

"messagesTotal": integer,

"messagesUnread": integer,

"threadsTotal": integer,

"threadsUnread": integer,

"color": {

"textColor": string,

"backgroundColor": string

}

}

Slika 1.1 JSON reprezentacija *label* resursa

### Autentifikacija i autorizacija

Google Gmail API koristi OAuth 2.0 protokol za autentifikaciju i autorizaciju. Aplikacija koja koristi API mora da specifikuje jedan ili više *scope* stringova. Scope služi da se identifikovanje resursa koje je moguće koristiti. Scope zajedno sa grupom tokena obezbeđuje korisnikov pristup resursu. Scope stringovi koji postoje su:

* <https://mail.google.com/> - omogućava čitanje, slanje, brisanje i upravljanje porukama,
* <https://www.googleapis.com/auth/gmail.compose> - omogućava upravljanje porukama za slanje i slanje poruka,
* <https://www.googleapis.com/auth/gmail.insert> - omogućava dodavanje poruke u poštansko sanduče,
* <https://www.googleapis.com/auth/gmail.labels> - omogućava upravljanje labelama,
* <https://www.googleapis.com/auth/gmail.metadata> - omogućava pregled metapodataka,
* <https://www.googleapis.com/auth/gmail.modify> - omogućava pregled i izmenu poruke ali ne i brisanje,
* <https://www.googleapis.com/auth/gmail.readonly> - omogućava pregled poruka i podešavanja,
* <https://www.googleapis.com/auth/gmail.send> - omogućava slanje poruke,
* <https://www.googleapis.com/auth/gmail.settings.basic> - omogućava upravljanje osnovnim podešavanjima,
* <https://www.googleapis.com/auth/gmail.settings.sharing> - omogućava upravljanje osetljivim podešavanjima.

### Tipičan proces rada sa Gmail API servisom

Tipičan rad sa Gmail API servisom se sastoji iz sledećih koraka:

* Autentifikacija,
* Poziv API metode,
* Procesiranje resursa dobijenih u odgovoru.

Metode Gmail API-ja je moguće pozivati putem HTTP metoda, međutim postoji biblioteka koja to olakšava. U razvoju aplikacije korišćena je implementacija ovog servisa u Java programskom jeziku.

## Spring okruženje

Spring je okuruženje za jednostavniji razvoj poslovnih aplikacija u Javi. Trenutno je najpopularniji i najuticajnije okruženje za razvoj Java veb aplikacija. Omogućuje kreiranje J2EE aplikacija na jednostavniji način. Sadrži infrastrukturu za izgradnju Java aplikacija, upravlja infrastrukturom tako da se standardne funkcionalnosti realizuju brzo i lako.

Glavni koncepti ovog okruženja su povezivanje objektata kroz *dependecy injection* mehanizam [9] i deklarativno programiranje korišćenjem aspekata.

Prvi koncept omogućava da se funkcionalnosti realizuju zajedničkim radom međusobno povezanih komponenti, a drugi koncept omogućava izdvajanje često korišćenih funkcionalnosti u aspekte i njihovo ponovno iskorišćenje.

# Specifikacija aplikacije

Zadatak obuhvata izradu veb aplikacije za upravljanje elektronskom poštom. Budući da aplikacija kominicira sa Gmail API-jem korisnik ima mogućnost da koristi osnovni skup funkcionalnosti koje originalna aplikacija nudi. Aplikacija omogućava pregledanje elektronske pošte, rukovanje poštom i njeno ažuriranje. Takođe omogućava kreiranje novih poruka, slanje poruka i odgovaranje na postojeće.

Prilikom dolaska na veb stranicu aplikacije korisniku je omogućena prijava na sistem. Prijava na sistem se vrši pomoću Google-ovog sistema za autentifikaciju.

Nakon uspešne prijave na sistem korisniku se prikazuje glavna stranica aplikacije na kojoj korisnik može da vidi svoju elektronsku poštu po labelama. Korisnik ima mogućnost da kreira novu poruku, odgovori na neku od primljenih ili da poruku prosledi drugom korisniku. Pored glavne stranice sa prikazom primljene pošte postoje još stranica sa prikazom pošte iz svake labele, pošta za slanje, pošta za brisanje… Prikaz glavne stranice je specifičan za svakog korisnika jer je prikaz labela određen korisnikovim podešavanjem labela u originalnoj aplikaciji.

Na stranci sa poštom za brisanje korisnik može da pregleda poštu koju je prebacio u kantu za brisanje i ukoliko se predomisli da opozove tu akciju ili da ih direktno obriše. Na stranici sa poštom pripremljenom za slanje korisnik ima mogućnost da poruku pošalje.

U narednom poglavlju biće prikazan dijagram slučajeva korišćenja.

## Dijagram slučajeva korišćenja

Dijagram slučajeva korišćenja prikazan je na slici 2.1. Kao što se može videti na dijagramu, aplikaciju mogu koristiti dva tipa korisnika. Prvi, neprijavljeni korisnik ima mogućnost da se prijavi na sistem. Nakon uspešne prijave, korisnik postaje ulogovani korisnik i ima mogućnost da pregleda svoje poštu, manipuliše poštom po labelama, kreira novu poštu, šalje poštu. Takođe, korisnik ima mogućnost da odgovara na postojeću ili je briše.

Slika 3.1 Dijagram slučajeva korišćenja

# Opis implementacije

Aplikaciju čine dve celine, serverski i klijentski deo. Serverski deo aplikacije je implementiran u programskom jeziku Java, verzije 1.8 korišćenjem Spring okruženja. Za implementaciju klijentskog dela aplikacije korišćena je prethodno kreirana OldSchoolComponents biblioteka.

## Serverski deo

Kao što je već rečeno serverski deo aplikacije je kreiran radi lakše komunikacije sa Gmail API-jem. Na serverskoj strani se nalaze sloj *controller*-a i *service*-a.

### Controller

Budući da sloj controller-a sadrži samo pozive ka Gmail API-ju svi pozivi su grupisani u jednu klasu Controller. Servisni sloj se sastoji od tri klase u kojima se nalaze metode za procesiranje podataka potrebnih kontroleru. Kao što je rečeno, klasa Controller sadrži metode koji šalju pozive ga Gmail API-ju poput metode za autentifikaciju, metode za prikupljenje korisnikovih labela ili poruka. Na slici broj je prikazana metoda za autentifikaciju aplikacije i prikazvanje Gmail prozora za autentifikaciju korisnika. Budući da se koristi Gmail API, autentifikacija korisnika je realizovana od strane Gmail servisa. Zbog preglednosti na slikama neće biti prikazane *throws* deklaracije kao i *import* sekcija.

@RequestMapping(value = "/login",

method = RequestMethod.***GET***)

**public** RedirectView login(){

AuthorizationCodeRequestUrl authorizationUrl;

**if** (*flow* == **null**) {

Details web = **new** Details();

web.setClientId(clientId);

web.setClientSecret(clientSecret);

*clientSecrets* = **new** GoogleClientSecrets().setWeb(web);

*httpTransport* = GoogleNetHttpTransport.*newTrustedTransport*();

List<String> scopes = service.getScopes();

*flow* = **new** GoogleAuthorizationCodeFlow

.Builder(*httpTransport*, ***JSON\_FACTORY***, *clientSecrets*,Collections.*unmodifiableList*(scopes)).build();

}

authorizationUrl = *flow*.newAuthorizationUrl().setRedirectUri(redirectUri);

String build = authorizationUrl.build();

**return** **new** RedirectView(build);

}

Slika 4.1 Implementacija metode za autentifikaciju aplikacije

Gmail API servis u svom podešavanju nudi mogućnost redirektovanja na adresu korisnikove aplikacije nakon što se aplikacija autentifikuje. Budući da se naša aplikacija sastoji dva dela, klijentski i serverski deo koji su podignuti na različitim serverima Gmail servis nakon autentifikacije aplikaciju redirektuje na URL na serveru koji potom salje poziv klijentskoj aplikaciji. Nakon što se izvrši redirekcija, korisnik je ulogovan i popunjava se polje *credential* koje se koristi za autentifikaciju korisnika pri svakom sledećem Gmail API pozivu. Implementacija ove metode je prikazana na slici broj.

@RequestMapping(value = "/login/gmailCallback",

method = RequestMethod.***GET***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** RedirectView callback(@RequestParam(value = "code") String code) {

TokenResponse response = *flow*.newTokenRequest(code).setRedirectUri(redirectUri).execute();

*credential* = *flow*.createAndStoreCredential(response, "userID");

*client* = **new** com.google.api.services.gmail.Gmail.Builder(

*httpTransport*, ***JSON\_FACTORY***, *credential*)

.setApplicationName(***APPLICATION\_NAME***).build();

RedirectView redirect = **new** RedirectView("http://localhost:5500/SimpleMailApp/client/index.html");

**return** redirect;

}

Slika 4.2 Implementacija metode za redirektovanje

Nakon što je korisnik ulogovan mogu se slati ostali Gmail API pozivi. Na slici broj se nalazi prikaz metode koja traži od Gmail API-ja podatke o ulogovanom korisniku.

@RequestMapping(value = "/me",

method = RequestMethod.***GET***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> getMe(){

Profile profile = *client*.users().getProfile(userId).execute();

JSONObject me = **new** JSONObject();

me.put("email", profile.getEmailAddress());

**return** **new** ResponseEntity<>(me.toString(), HttpStatus.***OK***);

}

Slika 4.3 Implementacija metode za dobavljanje podataka o korisniku

Na slici broj prikazana je GmailController klasa.

@CrossOrigin(origins = "http://localhost:5500")

@RestController

**public** **class** GmailController {

**private** **static** **final** String ***APPLICATION\_NAME*** = "Simple mail app";

**private** **static** **final** String ***USER\_ID*** = "me";

**private** **static** HttpTransport *httpTransport*;

**private** **static** **final** JsonFactory ***JSON\_FACTORY*** = JacksonFactory.*getDefaultInstance*();

**private** **static** Gmail *client*;

**private** **static** GoogleClientSecrets *clientSecrets*;

**private** **static** GoogleAuthorizationCodeFlow *flow*;

**private** **static** Credential *credential*;

@Value("${gmail.client.clientId}")

**private** String clientId;

@Value("${gmail.client.clientSecret}")

**private** String clientSecret;

@Value("${gmail.client.redirectUri}")

**private** String redirectUri;

@Autowired

GmailService service;

@RequestMapping(value = "/login",

method = RequestMethod.***GET***)

**public** RedirectView login(){

AuthorizationCodeRequestUrl authorizationUrl;

**if** (*flow* == **null**) {

Details web = **new** Details();

web.setClientId(clientId);

web.setClientSecret(clientSecret);

*clientSecrets* = **new** GoogleClientSecrets().setWeb(web);

*httpTransport* = GoogleNetHttpTransport.*newTrustedTransport*();

List<String> scopes = service.getScopes();

*flow* = **new** GoogleAuthorizationCodeFlow

.Builder(*httpTransport*, ***JSON\_FACTORY***, *clientSecrets*, Collections.*unmodifiableList*(scopes))

.build();

}

authorizationUrl = *flow*.newAuthorizationUrl().setRedirectUri(redirectUri);

String build = authorizationUrl.build();

**return** **new** RedirectView(build);

}

@RequestMapping(value = "/login/gmailCallback",

method = RequestMethod.***GET***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** RedirectView callback(@RequestParam(value = "code") String code) {

TokenResponse response = *flow*.newTokenRequest(code).setRedirectUri(redirectUri).execute();

*credential* = *flow*.createAndStoreCredential(response, "userID");

*client* = **new** com.google.api.services.gmail.Gmail.Builder(*httpTransport*, ***JSON\_FACTORY***, *credential*) .setApplicationName(***APPLICATION\_NAME***).build();

RedirectView redirect = **new** RedirectView("http://localhost:5500/SimpleMailApp/client/index.html");

**return** redirect;

}

@RequestMapping(value = "/me",

method = RequestMethod.***GET***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> getMe() {

Profile profile = *client*.users().getProfile(***USER\_ID***).execute();

JSONObject me = **new** JSONObject();

me.put("email", profile.getEmailAddress());

**return** **new** ResponseEntity<>(me.toString(), HttpStatus.***OK***);

}

@RequestMapping(value = "/labels",

method = RequestMethod.***GET***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> getLabels(){

ListLabelsResponse labelsResponse = *client*.users().labels().list(***USER\_ID***).execute(); JSONArray labelArray = **new** JSONArray();

**for** (Label l : labelsResponse.getLabels()) {

Label label = *client*.users().labels().get(***USER\_ID***, l.getId()).execute();

JSONObject labelJSON = service.fetchLabel(label);

**if**(labelJSON.getString("labelListVisibility")

.equals("labelShow")) {

labelArray.put(labelJSON);

}

}

**return** **new** ResponseEntity<>(labelArray.toString(), HttpStatus.***OK***);

}

@RequestMapping(value = "/allMessages",

method = RequestMethod.***GET***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> getAllMessages() {

ListMessagesResponse msgResponse = *client*.users().messages().list(***USER\_ID***).execute();

JSONArray messageArray = fetchMessages(msgResponse);

**return** **new** ResponseEntity<>(messageArray.toString(), HttpStatus.***OK***);

}

@RequestMapping(value = "/messages",

method = RequestMethod.***GET***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> getMessages(

@RequestParam(value = "label") String l) {

List<String> labelIds = **new** ArrayList<>();

labelIds.add(l);

ListMessagesResponse msgResponse = *client*.users().messages().list(***USER\_ID***).setLabelIds(labelIds).execute();

JSONArray messageArray = fetchMessages(msgResponse);

**return** **new** ResponseEntity<>(messageArray.toString(), HttpStatus.***OK***);

}

@RequestMapping(value = "/message",

method = RequestMethod.***GET***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> getMessage(

@RequestParam(value = "id") String id) {

Message message = *client*.users().messages().get(USER\_ID, id).setFormat("full").execute();

JSONObject messageJSON = service.fetchFullMessage(message);

**return** **new** ResponseEntity<>(messageJSON.toString(), HttpStatus.***OK***);

}

@RequestMapping(value = "/send",

method = RequestMethod.***POST***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> sendMessage(@RequestBody String body) {

Message temp = service.prepareForSend(body, ***USER\_ID***);

Message send = *client*.users().messages().send(***USER\_ID***, temp).execute();

Message message = *client*.users().messages().get(***USER\_ID***, send.getId()).setFormat("full").execute();

JSONObject messageJSON = service.fetchFullMessage(message);

**return** **new** ResponseEntity<>(messageJSON.toString(), HttpStatus.***OK***);

}

@RequestMapping(value = "/draft",

method = RequestMethod.***POST***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> draftMessage(

@RequestBody String body) {

Message temp = service.prepareForSend(body, ***USER\_ID***); Draft tempDraft = **new** Draft(); tempDraft.setMessage(temp); *client*.users().drafts().create(***USER\_ID***, tempDraft).execute();

List<String> labelIds = **new** ArrayList<>();

labelIds.add("DRAFT");

ListMessagesResponse msgResponse = *client*.users().messages().list(***USER\_ID***)

.setLabelIds(labelIds).execute();

JSONArray messageArray = fetchMessages(msgResponse);

**return** **new** ResponseEntity<>(messageArray.toString(), HttpStatus.***OK***);

}

@RequestMapping(value = "/trash",

method = RequestMethod.***POST***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> trashMessage(

@RequestParam(value = "id") String id) {

*client*.users().messages().trash(***USER\_ID***, id).execute();

List<String> labelIds = **new** ArrayList<>();

labelIds.add("TRASH");

ListMessagesResponse msgResponse = *client*.users().messages().list(***USER\_ID***).setLabelIds(labelIds).execute();

JSONArray messageArray = fetchMessages(msgResponse);

**return** **new** ResponseEntity<>(messageArray.toString(), HttpStatus.***OK***);

}

@RequestMapping(value = "/untrash",

method = RequestMethod.***POST***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> untrashMessage(

@RequestParam(value = "id") String id) {

*client*.users().messages().untrash(***USER\_ID***, id).execute();

List<String> labelIds = **new** ArrayList<>();

labelIds.add("INBOX");

ListMessagesResponse msgResponse = *client*.users().messages().list(***USER\_ID***)

.setLabelIds(labelIds).execute();

JSONArray messageArray = fetchMessages(msgResponse);

**return** **new** ResponseEntity<>(messageArray.toString(), HttpStatus.***OK***);

}

@RequestMapping(value = "/delete",

method = RequestMethod.***POST***,

produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> deleteMessage(

@RequestParam(value = "id") String id) {

*client*.users().messages().delete(***USER\_ID***, id).execute();

List<String> labelIds = **new** ArrayList<>();

labelIds.add("TRASH");

ListMessagesResponse msgResponse = *client*.users().messages().list(***USER\_ID***).setLabelIds(labelIds).execute();

JSONArray messageArray = fetchMessages(msgResponse);

**return** **new** ResponseEntity<>(messageArray.toString(), HttpStatus.***OK***);

}

**private** JSONArray fetchMessages(ListMessagesResponse msgResponse) {

JSONArray messageArray = **new** JSONArray();

**if**(msgResponse.getMessages() == **null**) {

**return** messageArray;

}

**for** (Message msg : msgResponse.getMessages()) {

Message message = *client*.users().messages().get(userId, msg.getId()).execute();

JSONObject messageJSON = service.fetchMessage(message);

messageArray.put(messageJSON);

}

**return** messageArray;

}

**private** JSONArray fetchLabels(ListLabelsResponse labelsResponse) {

JSONArray labelArray = **new** JSONArray();

**for** (Label l : labelsResponse.getLabels()) {

Label label = *client*.users().labels().get(***USER\_ID***, l.getId()).execute();

JSONObject labelJSON = service.fetchLabel(label); **if**(labelJSON.getString("labelListVisibility")

.equals("labelShow")) {

labelArray.put(labelJSON);

}

}

**return** labelArray;

}

}

Slika 4.4 GmailController klasa

### Service

Servisni sloj aplikacije čini 3 servisa GmailService, FetchService i UtilService. GmailService sadrži pomoćne metode specifične za Gmail API. Na slici broj se može videti implemetacija ovog servisa.

@Service

**public** **class** GmailService {

**public** MimeMessage createEmail(String to, String from, String subject, String bodyText) {

Properties props = **new** Properties();

Session session = Session.*getDefaultInstance*(props, **null**);

MimeMessage email = **new** MimeMessage(session);

**if**(from != **null**) {

email.setFrom(**new** InternetAddress(from));

} **else** {

email.setFrom("");

}

**if**(to != **null** && !to.equals("")) {

email.addRecipient(RecipientType.***TO***, **new** InternetAddress(to));

} **else** {

email.addRecipients(RecipientType.***TO***, "");

}

**if**(subject != **null**) {

email.setSubject(subject);

} **else** {

email.setSubject("");

}

**if**(bodyText != **null**) {

email.setText(bodyText);

} **else** {

email.setText("");

}

**return** email;

}

**public** Message createMessageWithEmail(MimeMessage emailContent) {

ByteArrayOutputStream buffer = **new** ByteArrayOutputStream();

emailContent.writeTo(buffer);

**byte**[] bytes = buffer.toByteArray();

String encodedEmail = Base64.*encodeBase64URLSafeString*(bytes);

Message message = **new** Message();

message.setRaw(encodedEmail);

**return** message;

}

**public** Message prepareForSend(String body, String userId) {

JSONObject json = **new** JSONObject(body);

String to = json.getString("to");

String subject = json.getString("subject");

String bodyText = json.getString("bodyText");

MimeMessage emailContent = createEmail(to, userId, subject, bodyText);

Message message = createMessageWithEmail(emailContent);

**return** message;

}

}

Slika 4.5 Implementacija GmailService klase

Radi uštede resursa i lakšeg korišćenja podaci koje vrati Gmail servis se transformisu u odgovarajući JSON objekat. FetchService sadrži metode koje vrše transformaciju a implementacija se nalazi na slici broj.

@Service

**public** **class** FetchService {

**public** JSONObject fetchLabel(Label label) {

JSONObject labelJSON = **new** JSONObject();

labelJSON.put("name", label.getName());

labelJSON.put("messagesTotal", label.getMessagesTotal());

**if**(label.getLabelListVisibility() != **null**) {

labelJSON.put("labelListVisibility", label.getLabelListVisibility());

} **else** {

labelJSON.put("labelListVisibility", "labelShow");

}

**return** labelJSON;

}

**public** JSONObject fetchMessage(Message message) {

JSONObject messageJSON = **new** JSONObject();

messageJSON.put("id", message.getId());

messageJSON.put("snippet", message.getSnippet());

JSONArray labels = **new** JSONArray();

**for** (String label : message.getLabelIds()) {

labels.put(label);

}

messageJSON.put("labels", labels);

JSONObject headersArray = fetchMessageHeaders(message);

messageJSON.put("headers", headersArray);

**return** messageJSON;

}

**public** JSONObject fetchFullMessage(Message message) {

JSONObject messageJSON = **new** JSONObject();

messageJSON.put("id", message.getId());

String content = "";

**if**(message.getPayload().getBody().getData()!= **null**) {

content = StringUtils.*newStringUtf8*(Base64.*decodeBase64*(

message.getPayload().getBody().getData()));

} **else** {

content = StringUtils.*newStringUtf8*(Base64.*decodeBase64*(

message.getPayload().getParts().get(0).getBody().getData()));

}

messageJSON.put("content", content);

JSONArray labels = **new** JSONArray();

**for** (String label : message.getLabelIds()) {

labels.put(label);

}

messageJSON.put("labelIds", labels);

JSONObject headersArray = fetchMessageHeaders(message);

messageJSON.put("headers", headersArray);

**return** messageJSON;

}

**public** JSONObject fetchMessageHeaders(Message message) {

JSONObject headersObject = **new** JSONObject();

**for** (MessagePartHeader header : message.getPayload().getHeaders()) {

**if**(header.getName().equals("Date")) {

SimpleDateFormat f = **new** SimpleDateFormat("EEE, dd MMM yyyy HH:mm:ss Z", Locale.***ROOT***); f.setTimeZone(

TimeZone.*getTimeZone*("UTC"));

Date date = f.parse(header.getValue());

DateFormat df = **new** SimpleDateFormat("dd.MM.yyyy kk:mm",

Locale.***ENGLISH***);

String s = df.format(date);

header.setValue(s);

headersObject.put("date", header.getValue());

}

**if**(header.getName().equals("Subject")) {

headersObject.put("subject", header.getValue());

}

**if**(header.getName().equals("From")) {

headersObject.put("from", header.getValue());

}

**if**(header.getName().equals("To")) {

headersObject.put("to", header.getValue());

}

}

**return** headersObject;

}

**public** JSONObject fetchDraft(Draft draft) {

JSONObject draftJSON = **new** JSONObject();

draftJSON.put("id", draft.getId());

Message message = draft.getMessage();

JSONObject messageJSON = fetchMessage(message);

draftJSON.put("message", messageJSON);

**return** draftJSON;

}

}

Slika 4.6 Implementacija FetchService klase

Gmail servisima bila moguća korisnik mora da specificira *scope* stringove koje će svom nalogu dozvoliti. Na slici broj prikazano je konfigurisanje dozvoljenih scope stringova.

@Service

**public** **class** UtilService {

**public** List<String> getScopes() {

List<String> scopes = **new** ArrayList<>();

scopes.add("https://mail.google.com/");

scopes.add(GmailScopes.***GMAIL\_COMPOSE***);

scopes.add(GmailScopes.***GMAIL\_INSERT***);

scopes.add(GmailScopes.***GMAIL\_LABELS***);

scopes.add(GmailScopes.***GMAIL\_MODIFY***);

scopes.add(GmailScopes.***GMAIL\_READONLY***);

scopes.add(GmailScopes.***GMAIL\_SEND***);

**return** scopes;

}

}

Slika 4.7 Implementacija UtilService klase

## Klijentski deo

Klijentski deo aplikacije predstavlja *view* komponentu u arhitekturi ove aplikacije. Kao što je spomenuto za razvoj klijentskog dela aplikacije korišćena je prethodno kreirana OldSchoolComponents biblioteka za kreiranje komponenti korisničkog interfejsa.

Početno stanje aplikacije je glavni prozor sa prikazom primljene pošte i može se videti na slici broj.

Slika 4.8 Prikaz glavnog prozora

Funkcionalnost koja se nalazi iza ovog prikaza se moze videti na narednim slikama. Na slici broj se vidi iscrtavanje glavnog prozora u koje se umeću iscrtavanje labela (slika broj) i iscrtavanje primljene pošte (slika broj).

export default function drawApp(labels, messages, user) {

var vp1 = new VerticalPanel('vp1', 'container');

var emptyRow1 = new EmptyRow('er1', 'row');

vp1.add(emptyRow1);

var vp2 = new VerticalPanel('vp2', 'row');

vp1.add(vp2);

var vp3 = new VerticalPanel('vp3', 'col-sm-3 col-md-2');

vp2.add(vp3);

var vp4 = new HorizontalPanel('vp4', 'user-head');

vp3.add(vp4);

var ac1 = new AContainer('ac1', 'inbox-avatar', '', '#', '');

vp4.add(ac1);

var image1 = new Image('image1', 'img-responsive', './images/profile.png', user.name, '50px', '50px');

ac1.add(image1);

var vp5 = new VerticalPanel('vp5', 'btn-group pull-right');

vp4.add(vp5);

var b1 = new ButtonContainer('b1', 'btn btn-primary dropdown- toggle', '', 'button', 'dropdown');

vp5.add(b1);

var label1 = new Label('l1', 'caret', '', '');

b1.add(label1);

var ul1 = new UL('ul1', 'dropdown-menu');

vp5.add(ul1);

var li11 = new LI('li11', '');

ul1.add(li11);

var a11 = new A('a11', '', user.email, '#');

li11.add(a11);

var li12 = new LI('li12', '');

ul1.add(li12);

var a12 = new AContainer('a12', '', 'Log out', '');

li12.add(a12);

a12.onclick = function (e) {

e.preventDefault();

e.stopImmediatePropagation();

console.log('LOG OUT');

let axios = window.\_api.axios;

let userManager = new UserManager(axios);

userManager.logout();

}

var vp6 = new VerticalPanel('vp6', 'user-name');

vp4.add(vp6);

var h51 = new H5('h51', '', user.email);

vp6.add(h51);

var vp7 = new VerticalPanel('vp7', 'row');

vp1.add(vp7);

var hr1 = new HR('idhr1');

vp7.add(hr1);

var vp8 = new VerticalPanel('vp8', 'col-sm-3 col-md-2');

vp7.add(vp8);

var buttonCompose = new AContainer('buttonCompose', 'btn btn- danger btn-sm btn-block', 'COMPOSE');

vp8.add(buttonCompose);

var hr2 = new HR('idhr2');

vp8.add(hr2);

var ul2 = drawLabels(labels);

vp8.add(ul2);

var vp9 = drawInbox(messages);

vp7.add(vp9);

buttonCompose.onclick = function (e) {

e.preventDefault();

e.stopImmediatePropagation();

var vp7 = buttonCompose.findById("vp7");

var vp9 = buttonCompose.findById("vp9");

vp9.remove(vp7);

var component = drawCompose();

vp7.add(component);

}

return vp1;

}

Slika 1.1 Iscrtavanje glavnog prozora

export default function drawLabels(labels) {

var ul2 = new UL('ul2', 'nav nav-pills nav-stacked');

if (labels.length > 0) {

for (let index = 0; index < labels.length; index++) {

var container = new LI(labels[index].name, '');

ul2.add(container);

var a = new AContainer('a2' + index, '', labels[index].name, '#');

container.add(a);

var badge = new Label('badge' + index, 'badge pull-right', labels[index].messagesTotal);

a.add(badge);

if (labels[index].name == 'INBOX') {

container.addCSSClass('active');

}

container.onclick = function (e) {

e.preventDefault();

e.stopImmediatePropagation();

changeActiveClass(this.component);

var vp7 = container.findById("vp7");

var vp9 = container.findById("vp9");

vp9.remove(vp7);

let axios = window.\_api.axios;

let messageManager = new MessageManager(axios);

if (labels[index].name == 'TRASH') {

console.log('TRASH')

messageManager.fetchMessages(labels[index].name)

.then(response => {

var component = drawTrash(messageManager.messages);

vp7.add(component);

});

} else {

messageManager.fetchMessages(labels[index].name)

.then(response => {

var component = drawInbox(messageManager.messages);

vp7.add(component);

});

}

}

}

}

return ul2;

}

Slika 1.1 Iscrtavanje panela sa labelama

export default function drawInbox(messages) {

var vp9 = new VerticalPanel('vp9', 'col-sm-9 col-md-10');

var ul3 = new UL('ul3', 'nav nav-tabs');

vp9.add(ul3);

var li31 = new LI('li31', 'active');

ul3.add(li31);

var a31 = new AContainer('a31', '', 'Primary ', '#', 'tab');

li31.add(a31);

var hp1 = new HorizontalPanel('hp1', 'glyphicon glyphicon-inbox');

a31.add(hp1);

var vp10 = new VerticalPanel('vp10', 'tab-content');

vp9.add(vp10);

var vp11 = new VerticalPanel('vp11', 'tab-pane fade in active');

vp10.add(vp11);

var vp12 = new VerticalPanel('vp12', 'list-group');

vp11.add(vp12);

var inbox\_rows = [];

for (let index = 1; index < messages.length + 1; index++) {

var inbox\_rowID = "inbox\_row" + index;

inbox\_rows.push(new AContainer(inbox\_rowID, 'list-group-item', '', '#'));

}

for (let index = 0; index < messages.length; index++) {

var container = inbox\_rows[index];

vp12.add(container);

var trash = new I('i' + index, 'fa fa-trash');

container.add(trash);

trash.onclick = function(e) {

e.preventDefault();

e.stopImmediatePropagation();

console.log('TRASH ' + messages[index].id);

var vp7 = trash.findById("vp7");

var vp9 = trash.findById("vp9");

vp9.remove(vp7);

let axios = window.\_api.axios;

let messageManager = new MessageManager(axios);

messageManager.trashMessage(messages[index].id)

.then(response => {

var component = drawTrash(messageManager.messages);

vp7.add(component);

});

}

container.add(new EmptyCol('ec1' + index, ''));

container.add(new EmptyCol('ec2' + index, ''));

container.add(new Label("sender" + index, 'name', messages[index].headers.from, 'min-width: 160px;display: inline-block;'));

container.add(new Label("title" + index, '', messages[index].headers.subject, ''));

container.add(new Label("time" + index, 'badge', messages[index].headers.date, ''));

container.onclick = function (e) {

e.preventDefault();

e.stopImmediatePropagation();

var vp7 = container.findById("vp7");

var vp9 = container.findById("vp9");

vp9.remove(vp7);

let axios = window.\_api.axios;

let messageManager = new MessageManager(axios);

messageManager.fetchMessage(messages[index].id)

.then(response => {

var component =

drawSingleMail(messageManager.message);

vp7.add(component);

});

}

}

return vp9;

}

Slika 1.1 Iscrtavanje panela sa porukama

# Zaključak

Projekat predstavlja jednostavnu aplikaciju za upravljanje elektronskom poštom inspirisanu već postojećim rešenjem. Dosadašnji rad obuhvata osnovni skup operacija za upravljanje elektronskom poštom poput pregledanja pošte, manipulisanja poštom, kreiranje nove pošte i odgovaranja na postojeću.

Aplikacija je razvijana u trenutno aktuelnim tehnologijama i ima dobru osnovu za dalji razvoj. Biblioteka koja je korišćena za kreiranje komponenti korisničkog interfejsa je u procesu razvoja tako da su proširenja moguća.

LITERATURA

1. *JavaScript*

[*https://en.wikipedia.org/wiki/JavaScript*](https://en.wikipedia.org/wiki/JavaScript)

1. *Google Mail*

[*https://mail.google.com/mail/*](https://mail.google.com/mail/)

1. *Spring*

[*https://spring.io/*](https://spring.io/)

1. *Google Mail API*

[*https://developers.google.com/gmail/api/v1/reference/*](https://developers.google.com/gmail/api/v1/reference/)

1. *jQuery*

[*https://api.jquery.com/*](https://api.jquery.com/)

1. *Bootstrap*

[*https://getbootstrap.com/*](https://getbootstrap.com/)

1. *axios*

[*https://github.com/axios/axios*](https://github.com/axios/axios)

1. *Java*

[*https://en.wikipedia.org/wiki/Java\_(programming\_language)*](https://en.wikipedia.org/wiki/Java_(programming_language))

1. *Spring Boot*

[*https://spring.io/projects/spring-boot*](https://spring.io/projects/spring-boot)

1. *Google API Wiki*

[*https://en.wikipedia.org/wiki/Google\_APIs*](https://en.wikipedia.org/wiki/Google_APIs)

1. *Apache Maven*

[*https://en.wikipedia.org/wiki/Apache\_Maven*](https://en.wikipedia.org/wiki/Apache_Maven)

1. *Many to many (data model)*

[*https://en.wikipedia.org/wiki/Many-to-many\_(data\_model)*](https://en.wikipedia.org/wiki/Many-to-many_(data_model))

1. *Spring Boot*

[*https://spring.io/projects/spring-boot*](https://spring.io/projects/spring-boot)

1. *Arsenije Karpić – Veb aplikacija za upravljanje imejl porukama*
2. *Spring Boot*

[*https://spring.io/projects/spring-boot*](https://spring.io/projects/spring-boot)

# 6.KLJUČNA DOKUMENTACIJSKA INFORMACIJA

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број, **РБР**: | |  | |
| Идентификациони број, **ИБР**: | |  | |
| Тип документације, **ТД**: | | монографска публикација | |
| Тип записа, **ТЗ**: | | текстуални штампани документ | |
| Врста рада, **ВР**: | | дипломски-бечелор рад | |
| Аутор, **АУ**: | | Ивана Савин | |
| Ментор, **МН**: | | Проф Др Милан Видаковић | |
| Наслов рада, **НР**: | | Веб апликација за вођење белешки | |
| Језик публикације, **ЈП**: | | српски | |
| Језик извода, **ЈИ**: | | српски / енглески | |
| Земља публикововања, **ЗП**: | | Србија | |
| Уже географско подручје, **УГП**: | | Војводина | |
| Година, **ГО**: | | 2018 | |
| Издавач, **ИЗ**: | | ауторски репринт | |
| Место и адреса, **МА**: | | Нови Сад, Факултет техничких наука, | |
| Физички опис рада, **ФО**:  (поглавља/страна/цитата/табела/слика/графика/прилога) | | 5 / 58 / 0 / 0 / 47/ 0 / 0 | |
| Научна област, **НО**: | | Информатика | |
| Научна дисциплина, **НД**: | | Рачунарске науке | |
| Предметна одредница/Кључне речи, **ПО**: | | Јava, ВЕБ, REST, Angular 6 | |
| **УДК** | |  | |
| Чува се, **ЧУ**: | | Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, Нови Сад | |
| Важна напомена, **ВН**: | |  | |
| Извод, **ИЗ**: | | Задатак рада представља развој једноставне веб апликације за управљање белешкама инспирисану већ постојећим решењем (Google Keep). Серверски део апликације ће бити реализован у програмском језику Јава, коришћењем Spring окружења а клијентски део у Angular 6 окружењу. Апликација омогућава креирање белешки, размештање по екрану и њихово ажурирање. | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | |  | |
| Чланови комисије, **КО**: | Председник: | др Игор Дејановић, ванредни проф., ФТН Нови Сад |  |
|  | Члан: | др Стеван Гостојић, ванредни проф., ФТН Нови Сад | Potpis mentora |
|  | Члан,ментор: | др Милан Видаковић, ред. проф., ФТН Нови Сад |  |

# 7.KEY WORDS DOCUMENTATION

|  |  |  |  |
| --- | --- | --- | --- |
| Accession number, **ANO**: | |  | |
| Identification number, **INO**: | |  | |
| Document type, **DT**: | | monographic publication | |
| Type of record, **TR**: | | textual material | |
| Contents code, **CC**: | | BSc thesis | |
| Author, **AU**: | | Ivana Savin | |
| Mentor, **MN**: | | Prof Dr Milan Vidaković | |
| Title, **TI**: | | WEB APPLICATION FOR NOTES MANAGEMENT | |
| Language of text, **LT**: | | Serbian | |
| Language of abstract, **LA**: | | Serbian / English | |
| Country of publication, **CP**: | | Serbia | |
| Locality of publication, **LP**: | | Vojvodina | |
| Publication year, **PY**: | | 2018 | |
| Publisher, **PB**: | | author’s reprint | |
| Publication place, **PP**: | | Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6 | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | 5 / 58 / 0 / 0 / 47/ 0 / 0 | |
| Scientific field, **SF**: | | Electrical Engineering | |
| Scientific discipline, **SD**: | | Computer Science | |
| Subject/Key words, **S**/**KW**: | | Јava, ВЕБ, REST, Angular 6 | |
| **UC** | |  | |
| Holding data, **HD**: | | Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad | |
| Note, **N**: | |  | |
| Abstract, **AB**: | | The thesis deals with the implementation of a simple web application for notes management inspired by already existing solution. The back-end part is implemented using Java programming language and Spring framework, and  the front-end is implemented using Angular 6 framework. The application will provide basic managing features. | |
| Accepted by the Scientific Board on, **ASB**: | |  | |
| Defended on, **DE**: | |  | |
| Defended Board, **DB**: | President: | Igor Dejanović, PhD, assoc. prof., FTN Novi Sad |  |
|  | Member: | Stevan Gostojić, PhD, assoc. prof., FTN Novi Sad | Menthor's sign |
|  | Member, Mentor: | Milan Vidaković, PhD, full prof., FTN Novi Sad |  |

Biografija

Ивана Савин, рођена 29.01.1995. у Зрењанину. Завршила Основну школу „Бранко Радичевић" у Александрову и Зрењанинску гимазију у Зрењанину. Завршила основне академске студије на Факултету техничких наука у Новом Саду, смер Софтверско инжењерство и информационе технологије.

Контакт емајл адреса:

ivanasavin95[@gmail.com](mailto:pepicsrdjan134@gmail.com)

Контакт мобилни:

065/84-76-018