



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
У НОВОМ САДУ




Ивана Савин

РАЗВОЈ JavaScript БИБЛИОТЕКЕ ЗА РАД СА КОМПОНЕНТАМА КОРИСНИЧКОГ ИНТЕРФЕЈСА

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 МАСТЕР РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА: <ul style="list-style-type: none"> - проблем – тема рада; - начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; - литература 			

НАСЛОВ МАСТЕР (MASTER) РАДА:

Развој JavaScript библиотеке за рад са компонентама корисничког интерфејса

ТЕКСТ ЗАДАТКА:

Задатак рада представља развој *JavaScript* библиотеке за рад са компонентама корисничког интерфејса. За демонстрацију рада ове библиотеке, биће написана једноставна веб апликација за управљање електронском поштом. Клијентски део апликације ће бити развијен помоћу претходно развијене *JavaScript* библиотеке. Серверски део апликације ће бити реализован у програмском језику Јава, коришћењем *Spring* окружења. Апликација ће омогућити основни скуп операција, попут управљања електронском поштом, креирања и слања поште. Спецификација ће бити представљена UML дијаграмима.

Руководилац студијског	Ментор рада:

Примерак за: ☐ ☐ Студента; ☐ ☐ Ментора

SADRŽAJ

1. Uvod.....	3
2. Opis korišćenih tehnologija	5
2.1 <i>ECMAScript 6</i>	5
2.2 <i>OldSchoolComponents</i> biblioteka	6
2.3 <i>Google Gmail API</i>	15
3. Specifikacija aplikacije	23
3.1 Dijagram slučajeva korišćenja	24
3.2 Dijagram sekvence	25
4. Opis implementacije	27
4.1 Klijentski deo	27
4.2 Serverski deo	52
5. Zaključak.....	65
6.KLJUČNA DOKUMENTACIJSKA INFORMACIJA	69
7.KEY WORDS DOCUMENTATION	73

1. Uvod

Zadatak rada predstavlja kreiranje *JavaScript* [1] biblioteke [2] 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 postojećim rešenjem (*Google Mail*) [3].

Budući da je bilo potrebno razviti veb aplikaciju, posebno su razvijeni klijentski i serverski deo. Klijentski deo aplikacije je razvijen korišćenjem prethodno implementirane biblioteke dok je serverski deo aplikacije *Spring* [4] aplikacija koja komunicira sa *Gmail API*-jem [5].

Prvo poglavlje je uvodno, dok se u drugom nalazi opis izabranih tehnologija koji su korišćene 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.

2.Opis korišćenih tehnologija

Za implementaciju klijentskog dela reprezentativne veb aplikacije korišćena je *OldSchoolComponents* [2] *JavaScript* biblioteka za kreiranje komponenti korisničkog interfejsa, *jQuery* biblioteka za upravljanje elementima DOM stabla [6], *Bootstrap* biblioteka za dizajn komponenti [7] i *axios* biblioteka za komunikaciju sa serverom [8].

Serverski deo aplikacije je kreiran zarad lakše komunikacije sa *Gmail API*-jem. Implementiran je u programskom jeziku Java [9], verzije 1.8, korišćenjem *Spring* okruženja. Radi pojednostavljenog korišćenja *Spring* okruženja korišćen je *Spring Boot* [10] razvojni okvir.

2.1 *ECMAScript* 6

ECMAScript 6 [11] je verzija *JavaScript*-a koja je uvela značajna unapređenja i olakšanja u radu. Unapređenje koje nudi *ECMAScript* 6 jeste objektno-orijentisani pristup radu i mogućnost definisanja *JavaScript* klasa. Kreiranje objekata i njihovo nasleđivanje je sada moguće na mnogo lakši način nego pre postojanja klasa. *ECMAScript* 6 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.

2.1.1 Klase

Jedan način kreiranja klase je korišćenjem *class* deklaracije, tj. navođenjem ključne reči *class* i imena klase. Primer deklaracije klase se može videti na listingu 1.1.

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
```

Listing 1.1 Kreiranje klase pomoću *class* deklaracije

Drugi način kreiranja klase je korišćenjem *class* izraza, kao na listingu 1.2.

```
let Rectangle = class {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
};
```

Listing 1.2 Kreiranje klase pomoću *class* izraza

Class deklaracije su sličan pristup kreiranju klasa kao u ostalim objektno-orijentisanim jezicima pa je ono izabrano kao način deklarisanja.

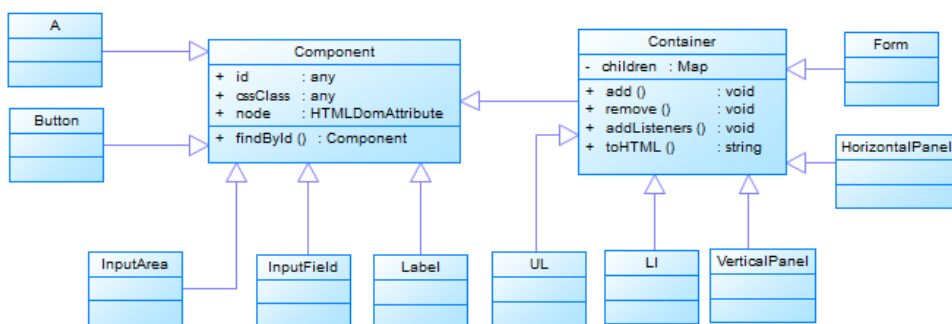
2.2 *OldSchoolComponents* biblioteka

OldSchoolComponents biblioteka [2] je *JavaScript* biblioteka, namenjena radu sa komponentama korisničkog interfejsa. Ona pruža programeru mogućnost da veb aplikaciju pravi programski, upotrebom koda, tako što se svaka komponenta na kraju pretvara u HTML, koji se zatim ugrađuje u osnovnu stranicu.

2.2.1 Komponente

Hijerarhija grafičkih komponenti počinje sa komponentom *Component* i ona je bazna ostalim komponentama i može se videti na slici 1.1. Osnovna komponenta sadrži attribute potrebne za grafičko iscrtavanje jedne HTML

komponente a to su *id* atribut za identifikaciju, *CSSclass* atribut za prikaz komponente, kao i *node* atribut koji predstavlja vezu sa DOM reprezentacijom komponente.



Slika 1.1 Dijagram klasa grafičkih komponenti

2.2.1.1 constructor metoda

Specijalna metoda *constructor* je 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. Na listingu 1.3 se može videti implementacija *constructor* metode u klasi *Component*.

```

constructor(id, CSSclass) {
    this.id = id;
    this.CSSclass = CSSclass;
    this.node = document.createElement("div");
    this.node.id = id;
    this.node.component = this;
    document.body.appendChild(this.node);
}

```

Listing 1.3 Implementacija *constructor* metode

Ostale komponente koje nasleđuju osnovnu komponentu imaju mogućnost dodavanja specifičnih atributa i metoda. Jedan primer je klasa *InputField* koja predstavlja komponentu za unos podataka. Njena implementacija se može videti na listingu 1.4 a pored osnovnih atributa ona sadrži i attribute *type* i *placeholder*.

```
export default class InputField extends Component {
  constructor(id, CSSclass, type, placeholder, value) {
    super(id, CSSclass);
    this.type = type;
    this.placeholder = placeholder;
    this.value = value;
  }

  tohtml() {
    return "<input id='" + this.id + "' type='" + this.type +
      "' class='" + this.CSSclass + "' placeholder='" +
      this.placeholder + "' value='" + this.value
      + "'></input>";
  }
}
```

Listing 1.4 Implementacija InputField klase

2.2.2 Kontejneri

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

2.2.2.1 constructor metoda

Container 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 kontejner sadrži. Na listingu 1.5 može se videti implementacija *constructor* metode.

```

constructor(id, CSSclass) {
    super(id, CSSclass);
    this.children = new Map();
}

```

Listing 1.5 Implementacija *constructor* metode

2.2.2.2 *add* metoda

Da bi se komponenta prikazala potrebno je dodati je u DOM stablo. Komponenta ne može postojati ukoliko nije dodata u neki *Container* objekat. Iz tog razloga postoji *add* metoda koja dodaje komponentu u određeni kontejner na nivou objektno reprezentacije i na nivou DOM stabla zamenjuje privremenu komponentu. Takođe, u ovoj metodi se poziva metoda *addListeners* koja će za komponentu dodati sve akcije koje se mogu desiti. Implementacija *add* metode se može videti na listingu 1.6.

```

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);
    }
}

```

Listing 1.6 Implementacija *add* metode

2.2.2.3 *remove* metoda

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 briše objektnu reprezentaciju komponente i briše komponentu iz DOM stabla. Implementacija ove metode se može videti na listingu 1.7.

```
remove(component) {  
    component.children.delete(this.id);  
    $('#' + this.id).remove();  
}
```

Listing 1.7 Implementacija *remove* metode

2.2.2.4 *tohtml* metoda

Prilikom dodavanja komponenti poziva se *tohtml* metoda koja vraća string reprezentaciju komponente. Prilikom iscrtavanja kontejnera iscrtaju se i sve komponente koje on sadrži. Implementacija ove metode se može videti na listingu 1.8 .

```
tohtml() {  
    var ret = "";  
    for (var child of this.children.values()) {  
        ret += child.tohtml();  
    }  
    return ret;  
}
```

Listing 1.8 Implementacija *tohtml* metode

2.2.2.5 *addListener* metoda

Kako bi akcije nad elementima DOM stabla bile moguće potrebno je definisati odgovarajuće metode. Implementacija metode koja dodaje odgovarajuće akcije se nalazi na listingu 1.9.

```

addListeners(component) {
    if (component.children) {
        for (var child of component.children.values()) {
            this.addListeners(child);
        }
    }
    if (component.onclick) {
        $("#" + component.id).click(component.onclick);
    }
    if (component.ondblclick) {
        $("#" + component.id).dblclick(component.ondblclick);
    }
    if (component.onChange) {
        $("#" + component.id).change(component.onChange);
    }
    if (component.onSubmit) {
        $("#" + component.id).submit(component.onSubmit);
    }
    if (component.onError) {
        $("#" + component.id).error(component.onError);
    }
    if (component.onselect) {
        $("#" + component.id).select(component.onselect);
    }
    if (component.ontoggle) {
        $("#" + component.id).toggle(component.ontoggle);
    }
    if (component.onhover) {
        $("#" + component.id).hover(component.onhover);
    }
    if (component.onfocus) {
        $("#" + component.id).focus(component.onfocus);
    }
    if (component.onfocusin) {
        $("#" + component.id).focusout(component.onfocusout);
    }
}

```

```

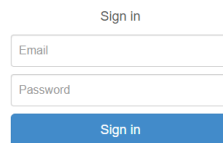
    if (component.onfocusout) {
        $("#" + component.id).focusin(component.onfocusin);
    }
    if (component.onmousedown) {
        $("#" + component.id).mousedown(component.onmousedown);
    }
    if (component.onmouseenter) {
        $("#" + component.id).mouseenter(component.onmouseenter);
    }
    if (component.onmouseleave) {
        $("#" + component.id).mouseleave(component.onmouseleave);
    }
    if (component.onmousemove) {
        $("#" + component.id).mousemove(component.onmousemove);
    }
    if (component.onmouseout) {
        $("#" + component.id).mouseout(component.onmouseout);
    }
    if (component.onmouseover) {
        $("#" + component.id).mouseover(component.onmouseover);
    }
    if (component.onmouseup) {
        $("#" + component.id).mouseup(component.onmouseup);
    }
    if (component.onkeydown) {
        $("#" + component.id).keydown(component.onkeydown);
    }
    if (component.onkeypress) {
        $("#" + component.id).keypress(component.onkeypress);
    }
    if (component.onkeyup) {
        $("#" + component.id).keyup(component.onkeyup);
    }
    var el = $("#" + component.id);
    el[0].component = component;
}

```

Listing 1.9 Implementacija *addListener*s metode

2.2.3 Primer korišćenja

U ovom poglavlju biće prikazan jednostavan primer korišćenja *OldSchoolComponents* biblioteke za kreiranje forme za prijavu na sistem. Za prijavu na određeni sistem uglavnom je potrebno uneti korisničko ime i lozinku i nakon toga pritisnuti određeno dugme.



Sign in

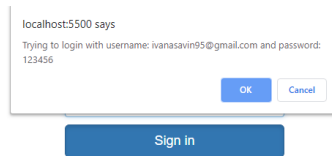
Email

Password

Sign in

Slika 1.2 Forma kreirana pomoću *OldSchoolComponents* biblioteke

Da bi se kreirala jednostavna forma za prijavu na sistem kao na slici 1.2, potrebno je kreirati objekat koji će sadržati polja za unos *email* adrese i lozinke i dugme za izvršavanje akcije. Na ovom primeru je demonstrirano i dodavanje osluškivača akcija od strane korisnika. Kao na slici 1.3 nakon što korisnik klikne na dugme prikazaće se obaveštenje sa unetim podacima na formi.



Slika 1.3 Reakcija na događaj na formi

Na listingu 1.10 biće prikazano kreiranje jednostavne forme.

```
$(document).ready(function () {  
    var mp = new MainPanel('main', '');  
    var vp1 = new VerticalPanel('vp1', 'container');  
    mp.add(vp1);  
    var vp2 = new VerticalPanel('vp2', 'row justify-content-md-center');  
    vp1.add(vp2);  
    var vp3 = new VerticalPanel('vp3', 'col-sm-6 col-md-4 col-md-offset-4');  
    vp2.add(vp3);  
    var h1 = new H1('h1', 'text-center login-title', 'Sign in');  
    vp3.add(h1);  
    var form = new Form('form1', 'form-signin');  
    vp3.add(form);  
    var inputEmail = new InputField('inputEmailId', 'form-control',  
    'email', 'Email', '');  
    inputEmail.onchange = function (e) {  
        var email = inputEmailId.value;  
        if (email.length == 0 || email.indexOf("@") == -1) {  
            alert('Not a valid email address!');  
        }  
    }  
});
```



```

        }
    }
    form.add(inputEmail);
    var inputPassword = new InputField('inputPasswordId', 'form-
control', 'password', 'Password', '');
    inputPassword.onChange = function (e) {
        var password = inputPasswordId.value;
        if (password.length < 6) {
            alert('Password must be a least 6 characters
long!');
        }
    }
    form.add(inputPassword);
    var button1 = new Button('b1', 'btn btn-lg btn-primary btn-block',
'Sign in', 'submit');
    button1.onclick = function (e) {
        alert("Trying to login with username: " +
inputEmailId.value + " and password: " + inputPasswordId.value);
    }
    form.add(button1);
})

```

Listing 1.10 Primer korišćenja *OldSchoolComponents* biblioteke

2.3 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. [12] Implementacije *Google Gmail API*-ja postoje u programskim jezicima Java, JavaScript, .NET, Objective C, PHP i Python.

2.3.1 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 i
- 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* [13] zavisnosti. Na listingu 1.11 može se videti importovanje ove zavisnosti.

```
<dependency>
  <groupId>com.google.apis</groupId>
  <artifactId>google-api-services-gmail</artifactId>
  <version>v1-rev65-1.18.0-rc</version>
</dependency>
```

Listing 1.11 Importovanje *Gmail API* biblioteke u *Spring* projekat

Nakon što je biblioteka uvezena u projekat može se koristiti. *Gmail API* pre korišćenja zahteva autentikaciju i autorizaciju. Google API koristi OAuth 2.0 protokol [14] za autentikaciju. OAuth 2.0 protokol je jednostavan protokol za autentikaciju, 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>.

2.3.2 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 poslata. *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 za kategorizaciju poruka. *label* resurs ima *many-to-many* vezu [15] 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 labela poput INBOX, TRASH, SPAM su automatski kreirane i ne mogu biti izmenjene ili izbrisane. User labela mogu biti dodate od strane korisnika ali i od strane sistema,
- *history* – resurs koji predstavlja kolekciju izmenjenih poruka u hronološkom redosledu,
- *thread* – resurs koji predstavlja kolekciju poruka koje predstavljaju konverzaciju. Kao i poruke i kolekcije poruka mogu imati labela 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 listingu 1.12 je prikazana JSON reprezentacija *message* resursa, a na listingu 1.13 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
}
```

Listing 1.12 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
  }
}
```

Listing 1.13 JSON reprezentacija *label* resursa

2.3.3 Autentikacija i autorizacija

Google Gmail API koristi OAuth 2.0 protokol za autentikaciju i autorizaciju. Aplikacija koja koristi API mora da specifikuje jedan ili više *scope* stringova. 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.

Scope služi za identifikovanje resursa koje je moguće koristiti. *Scope* zajedno sa grupom tokena obezbeđuje korisnikov pristup resursu. Prilikom

prve prijave na sistem aplikacija pita korisnika da dozvoli različite akcije nad svojim *Gmail* nalogom (slika 1.4).



Slika 1.4 Dijalog za dodeljivanje dozvola aplikaciji

Nakon što je prihvatio akcije korisnik dobija obaveštenje koje sve akcije aplikacija može da uradi nad njegovim nalogom (slika 1.5).

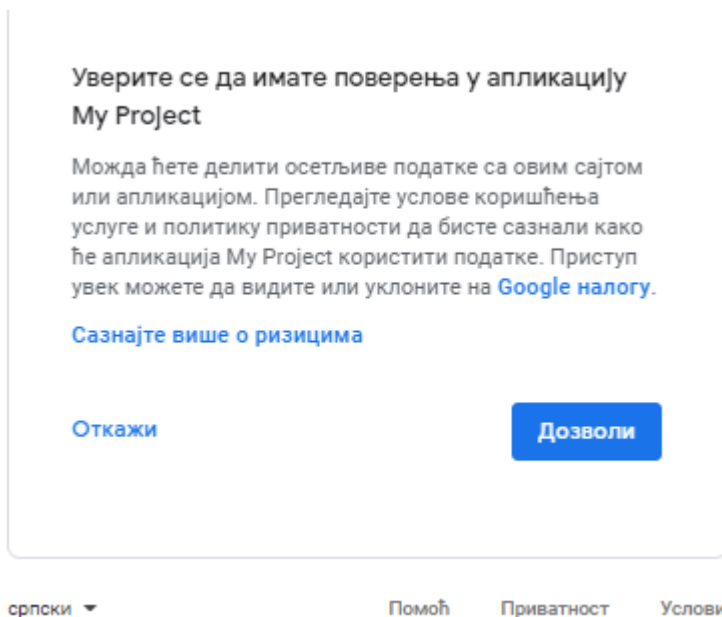
Потврдите изборе



ivana.unitedforce@gmail.com

Апликацији **My Project** дозвољаваће следеће:

- ☒ Читање, писање, слање и трајно брисање свих имејлова из Gmail-а
- ☒ Прегледајте имејлове и подешавања
- ☒ Слање имејлова у ваше име
- ☒ Управљање ознакама поштанског сандучета
- ☒ Убацивање поште у поштанско сандуче
- ☒ Управљање недовршеним порукама и слање порука е-поште
- ☒ Прегледање и мењање е-поште, али не и брисање



Slika 1.5 Obaveštenje za korisnika prilikom prijave na sistem

2.3.4 Tipičan proces rada sa *Gmail API* servisom

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

- autentikacija,
- poziv API metode i
- procesiranje resursa dobijenih u odgovoru.

Metode *Gmail API*-ja je moguće pozivati putem HTTP metoda, međutim postoje klijentske biblioteke koje to olakšavaju. U razvoju aplikacije korišćena je Java biblioteka.

3.Specifikacija aplikacije

Zadatak obuhvata izradu *JavaScript* biblioteke za rad sa komponentama korisničkog interfejsa – *OldSchoolComponents*. Za demonstraciju ove biblioteke, napisana je veb aplikacija za upravljanje elektronskom poštom. Budući da aplikacija komunicira 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 autentikaciju.

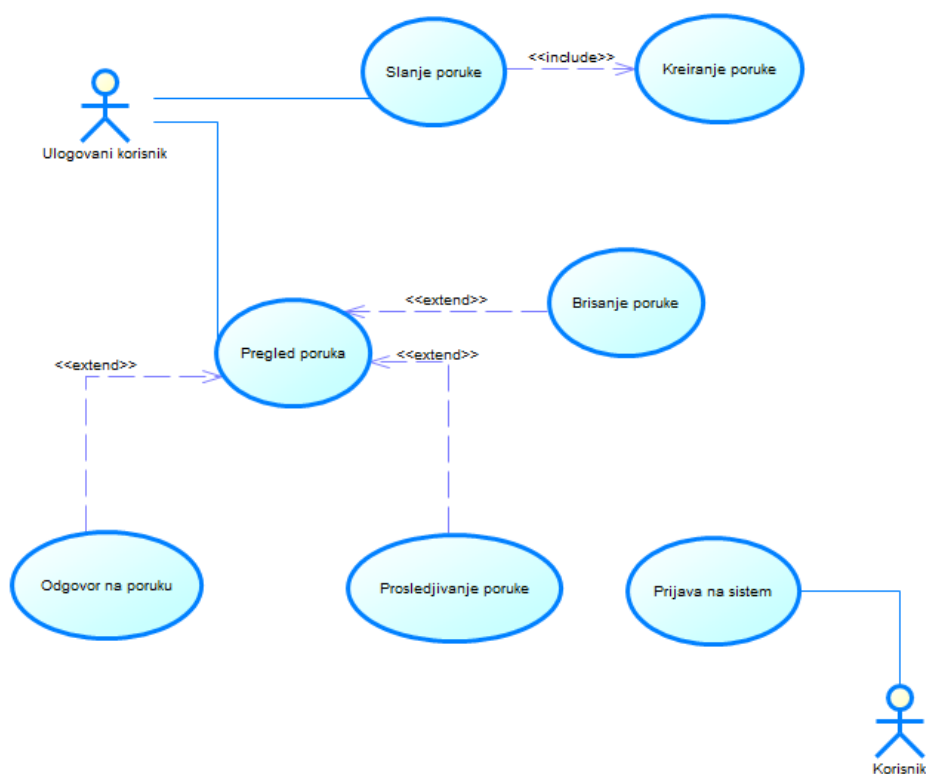
Nakon uspešne prijave na sistem korisniku se prikazuje glavna stranica aplikacije na kojoj korisnik može da pregleda svoju elektronsku poštu po labelama. Korisnik ima mogućnost da kreira novu poruku, odgovori na neku od primljenih ili da poruku prosledi drugom korisniku. Pored glavne stranice sa prikazom 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 i dijagram sekvence.

3.1 Dijagram slučajeja korišćenja

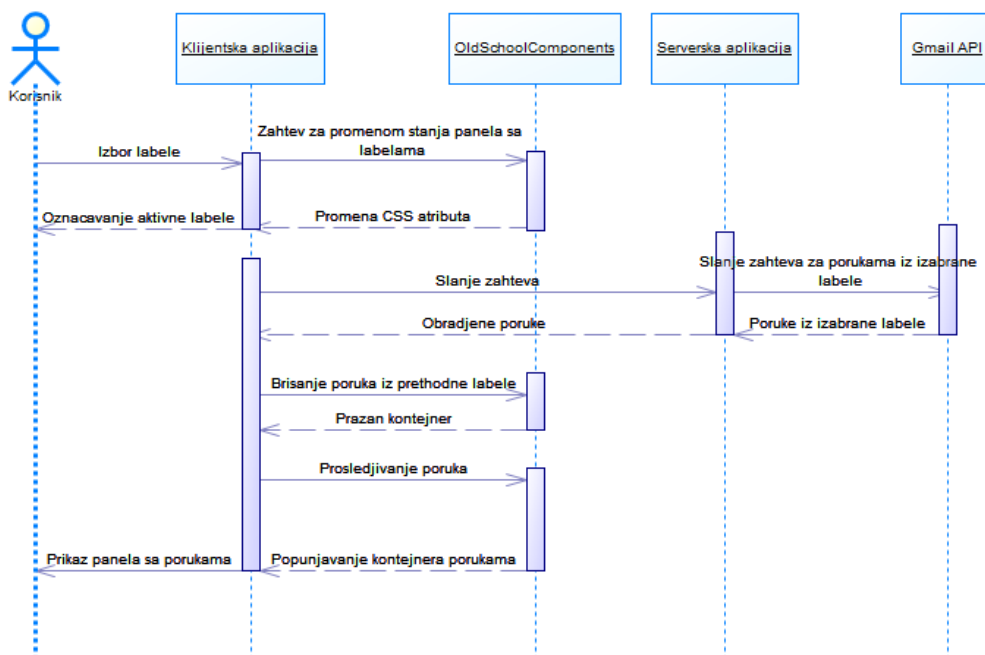
Dijagram slučajeja korišćenja prikazan je na slici 3.1. Kao što se može videti na dijagramu, aplikaciju mogu koristiti dva tipa korisnika. Prvi, neprijavljeni korisnik ima mogućnost da se prijavi na sistem. Nakon uspešne prijave, korisnik postaje ulogovani korisnik i ima mogućnost da pregleda svoje poštu, manipuliše poštom po labelama, kreira novu poštu, šalje poštu. Takođe, korisnik ima mogućnost da odgovara na postojeću ili je briše.



Slika 2.1 Dijagram slučajeja korišćenja

3.2 Dijagram sekvence

Na slici 2.2 prikazan je dijagram sekvence za prikaz poruka iz selektovane labele. Sekvenca događaja počinje korisnikovim izborom željene labele iz panela sa labelama. Nakon što korisnik izabere jednu labelu njoj se promeni CSS atribut i ona postaje aktivna a sve ostale labele imaju CSS atribut postavljen na neaktivan. Zatim klijentska aplikacija šalje zahtev serverskoj aplikaciji sa nazivom izabrane labele koja potom upućuje poziv *Gmail API*-ju za dobavljanje svih poruka iz izabrane labele. *Gmail API* vraća poruke serverskoj aplikaciji koja potom te podatke formatira i šalje klijentskoj aplikaciji. Nakon što pristignu podaci na klijentsku aplikaciju, kontejner sa porukama iz prethodno izabrane labele se briše i potom se iscrtavaja novi kontejner sa komponentama sa podacima.



Slika 2.2 Dijagram sekvence za prikaz poruka iz izabrane labele

4. Opis implementacije

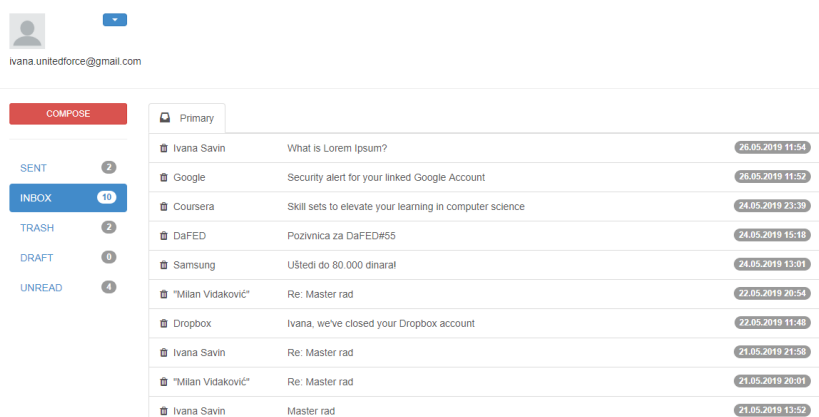
Aplikaciju čine dve celine, serverski i klijentski deo. Serverski deo aplikacije je implementiran u programskom jeziku Java, verzije 1.8 korišćenjem *Spring* okruženja. Za implementaciju klijentskog dela aplikacije korišćena je prethodno kreirana *OldSchoolComponents* biblioteka.

4.1 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. Klijentski deo aplikacije pored metoda za iscrtavanje komponenti čine i metode koje služe za komunikaciju sa serverskim delom aplikacije i dobavljanje podataka.

4.1.1 Metode za iscrtavanje grafičkih komponenti

Početno stanje aplikacije je glavni prozor sa prikazom primljene pošte i može se videti na slici 3.1.



Slika 3.1 Glavni prozor aplikacije

Funkcionalnost koja se nalazi iza ovog prikaza se može videti na narednim listinzima. Na listingu 2.1 se vidi iscrtavanje glavnog prozora u koje se umeću iscrtavanje labela (listing 2.2) i iscrtavanje primljene pošte (listing 2.3).

```
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 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, 'MAIL');
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 = drawComposeReply(undefined, 'COMPOSE');
    vp7.add(component);
}

return vp1;
}

```

Listing 2.1 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.addClass('active');
            }
            container.onclick = function (e) {
                e.preventDefault();
                e.stopImmediatePropagation();
            }
        }
    }
}

```

```

changeActiveClass(this.component);
let axios = window._api.axios;
let messageManager = new MessageManager(axios);
if (labels[index].name == 'TRASH') {
  var vp7 = container.findById('vp7');
  var vp9 = container.findById('vp9');
  vp9.remove(vp7);
  messageManager
    .fetchMessages(labels[index].name)
    .then(response => {
      var component
        = drawTrash(messageManager.messages);
      vp7.add(component);
    });
}
if (labels[index].name != 'TRASH') {
  if (labels[index].name == 'DRAFT') {
    var vp7 = container.findById('vp7');
    var vp9 = container.findById('vp9');
    vp9.remove(vp7);

    messageManager.fetchDrafts()
      .then(response => {
        var component =
drawInbox(messageManager.drafts, 'DRAFT');
        vp7.add(component);
      });
  } else {
    var vp7 = container.findById('vp7');
    var vp9 = container.findById('vp9');
    vp9.remove(vp7);

    messageManager.fetchMessages(labels[index].name)
      .then(response => {
        var component =
drawInbox(messageManager.messages, labels[index].name);
        vp7.add(component);
      });
  }
}
}
}
return ul2;
}

```

Listing 2.2 Isertavanje panela sa labelama


```

export default function drawInbox(data, type) {

  var messages = [];
  if(type == 'DRAFT') {
    for (let index = 0; index < data.length; index++) {
      messages.push(data[index].message);
    }
  } else {
    messages = data;
  }
  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();

      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: 190px;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();

    let axios = window._api.axios;
    let messageManager = new MessageManager(axios);

    if(type == 'DRAFT') {
        var vp7 = container.findById('vp7');
        var vp9 = container.findById('vp9');
        vp9.remove(vp7);

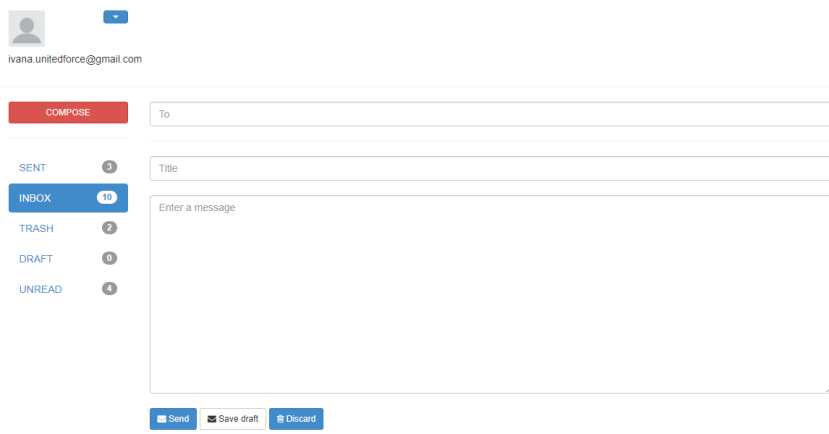
        messageManager.fetchMessage(messages[index].id)
            .then(response => {
                var component =
drawDraftForward(messageManager.message, data[index].id 'DRAFT');
                vp7.add(component);
            });
    } else {
        var vp7 = container.findById('vp7');
        var vp9 = container.findById('vp9');
        vp9.remove(vp7);

        messageManager.fetchMessage(messages[index].id)
            .then(response => {
                var component = drawMessage(messageManager.message,
type);
                vp7.add(component);
            });
    }
}
return vp9;
}

```

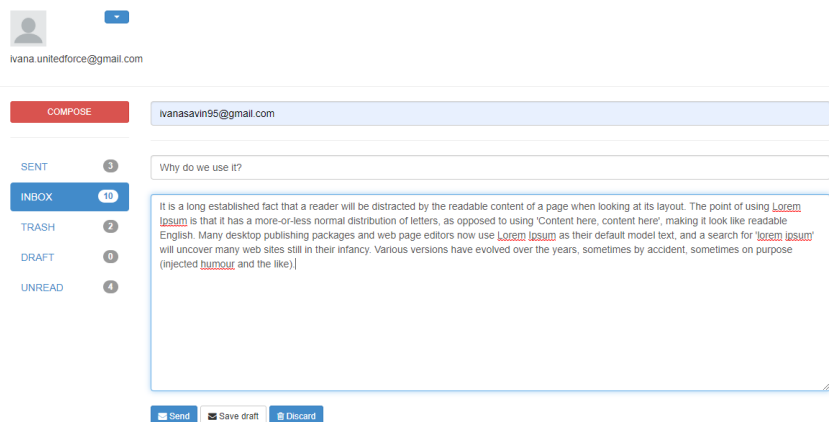
Listing 2.3 Iscrtavanje panela sa porukama

Glavni okvir aplikacije sadrži dugme za kreiranje nove poruke i prikaz korisnikovih labela. Ovaj okvir je vidljiv korisniku u svim situacijama. Ukoliko korisnik izabere kreiranje nove poruke prikazaće mu se prozor kao na slici 3.2.



Slika 3.2 Panel za kreiranje nove poruke

Nakon što korisnik unese podatke, prozor za slanje poruke izgleda kao na slici 3.3.



Slika 3.3 Panel za kreiranje nove poruke 2

Metode za iscrtavanje su pisane tako da se jednom metodom može iscrtati više komponenti. Iscrtavanje ove komponente vrši metoda *drawComposeReply* koja se nalazi na listingu 2.4 i metoda *drawComposeButtons* na listingu 2.5 na kojoj se iscrtavaju dugmad. Takođe, metoda *drawComposeReply* se koristi i za iscrtavanje panela za odgovor na poruku ali sa različitim iscrtavanjem dugmadi.

```
export default function drawComposeReply(message, type) {
  var vp9 = new VerticalPanel('vp9', 'col-sm-9 col-md-10');
  var vp10 = new VerticalPanel('vp10', 'inbox-body');
  vp9.add(vp10);

  var vp11 = new VerticalPanel('vp11', 'heading-inbox row');
  vp10.add(vp11);
  var vp12 = new VerticalPanel('vp12', 'col-md-12');
  vp11.add(vp12);
  if(message) {
    var inputEmail = new InputField('inputEmail', 'form-control',
'email', '', message.headers.from);
    vp12.add(inputEmail);
  } else {
    var inputEmail = new InputField('inputEmail', 'form-control',
'email', 'To', '');
    vp12.add(inputEmail);
  }
  var hr1 = new HR('hr1', '');
  vp12.add(hr1);

  var vp13 = new VerticalPanel('vp13', 'view-mail');
  vp10.add(vp13);
  if(message) {
    var inputTitle = new InputField('inputTitle', 'form-control',
'text', '', message.headers.subject);
    vp13.add(inputTitle);
  } else {
    var inputTitle = new InputField('inputTitle', 'form-control',
'text', 'Title', '');
    vp13.add(inputTitle);
  }
  var emptyRow2 = new EmptyRow('er2', '');
  vp13.add(emptyRow2);
  var inputMessage = new InputArea('inputMessage', 'form-control', 'Enter
a message', 13, 50);
  vp13.add(inputMessage);
  var emptyRow3 = new EmptyRow('er3', '');
  vp13.add(emptyRow3);

  if(type == 'COMPOSE') {
    var vp14 = drawComposeButtons();
    vp10.add(vp14);
  }
}
```

```

    if(type == 'REPLY') {
        var vp14 = drawReplayButtons();
        vp10.add(vp14);
    }
    return vp9;
}

```

Listing 2.4 Iscrtavanje panela za kreiranje poruke

```

export default function drawComposeButtons() {
    var vp14 = new VerticalPanel('vp14', 'compose-btn pull-left');
    var buttonSend = new AContainer('buttonSend', 'btn btn-sm btn-
primary', 'Send ', '', '');
    vp14.add(buttonSend);
    var i1 = new I('i1', 'fa fa-envelope');
    buttonSend.add(i1);
    var ec1 = new EmptyCol('ec1', '');
    vp14.add(ec1);

    buttonSend.onclick = function (e) {
        e.preventDefault();
        e.stopImmediatePropagation();

        var to = inputEmail.value;
        var subject = inputTitle.value;
        var bodyText = inputMessage.value;

        var vp7 = buttonSend.findById('vp7');
        var vp9 = buttonSend.findById('vp9');
        vp9.remove(vp7);

        let axios = window._api.axios;
        let messageManager = new MessageManager(axios);

        messageManager.sendMessage(to, subject, bodyText)
            .then(response => {
                var component = drawMessage(messageManager.message,
'MAIL');
                vp7.add(component);
            });
    }

    var buttonDraft = new AContainer('buttonDraft', 'btn btn-sm btn-
default', 'Save draft ', '', '');
    vp14.add(buttonDraft);
    var i2 = new I('i2', 'fa fa-envelope');
    buttonDraft.add(i2);
    var ec2 = new EmptyCol('ec2', '');
    vp14.add(ec2);

    buttonDraft.onclick = function (e) {
        e.preventDefault();
        e.stopImmediatePropagation();
    }
}

```

```

        var to = inputEmail.value;
        var subject = inputTitle.value;
        var bodyText = inputMessage.value;

        var vp7 = buttonDraft.findById('vp7');
        var vp9 = buttonDraft.findById('vp9');
        vp9.remove(vp7);

        let axios = window._api.axios;
        let messageManager = new MessageManager(axios);

        messageManager.draftMessage(to, subject, bodyText)
            .then(response => {
                var component = drawInbox(messageManager.messages,
'DRAFT');
                vp7.add(component);
            });
    }

    var buttonDiscard = new AContainer('buttonDiscard', 'btn btn-sm btn-
primary', 'Discard ', '', '');
    vp14.add(buttonDiscard);
    var i3 = new I('i3', 'fa fa-trash-o');
    buttonDiscard.add(i3);
    var ec3 = new EmptyCol('ec3', '');
    vp14.add(ec3);

    buttonDiscard.onclick = function (e) {
        e.preventDefault();
        e.stopImmediatePropagation();

        var vp7 = buttonDiscard.findById('vp7');
        var vp9 = buttonDiscard.findById('vp9');
        vp9.remove(vp7);

        let axios = window._api.axios;
        let messageManager = new MessageManager(axios);

        messageManager.fetchMessages('INBOX')
            .then(response => {
                var component = drawInbox(messageManager.messages,
'INBOX');
                vp7.add(component);
            });
    }

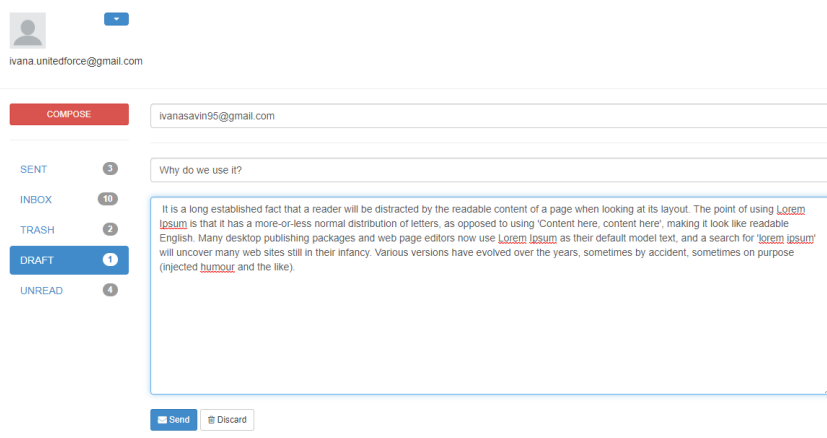
    var hr2 = new HR('hr2', '');
    vp14.add(hr2);

    return vp14;
}

```

Listing 2.5 Iscrtavanje panela za kreiranje poruke 2

Nakon što je popunio polja, korisnik ima mogućnost da poruku pošalje ili je pripremi za slanje. Ukoliko korisnik izabere da poruku pošalje kasnije ona će biti sačuvana pod labelom DRAFT i biće omogućena promena njenih atributa sve dok se ne izabere dugme za slanje. Sačuvana poruka za slanje se može videti na slici 3.4.



Slika 3.4 Panel sa porukom za slanje

Panel sa pripremljenom porukom za slanje iscrstavaju metode *drawDraftForward* (listing 2.6) i *drawDraftButtons* (listing 2.7). Metoda *drawDraftForward* se takođe koristi za iscrtavanje panela za prosleđivanje poruke.

```
export default function drawDraftForward(message, id, type) {
    var vp9 = new VerticalPanel('vp9', 'col-sm-9 col-md-10');
    var vp10 = new VerticalPanel('vp10', 'inbox-body');
    vp9.add(vp10);
    var vp11 = new VerticalPanel('vp11', 'heading-inbox row');
    vp10.add(vp11);
    var vp12 = new VerticalPanel('vp12', 'col-md-12');
    vp11.add(vp12);
```

```

        if(type == 'FORWARD') {
            var inputEmail = new InputField('inputEmail', 'form-control',
'email', 'To', '');
            vp12.add(inputEmail);
        }
        if(type == 'DRAFT') {
            if(message.headers.to != undefined) {
                var inputEmail = new InputField('inputEmail', 'form-
control', 'email', '', message.headers.to);
                vp12.add(inputEmail);
            } else {
                var inputEmail = new InputField('inputEmail', 'form-
control', 'email', '', '');
                vp12.add(inputEmail);
            }
        }
        var hr1 = new HR('hr1', '');
        vp12.add(hr1);

        var vp13 = new VerticalPanel('vp13', 'view-mail');
        vp10.add(vp13);
        var inputTitle = new InputField('inputTitle', 'form-control', 'text',
'', message.headers.subject);
        vp13.add(inputTitle);
        var emptyRow2 = new EmptyRow('er2', '');
        vp13.add(emptyRow2);
        var inputMessage = new InputAreaWithValue('inputMessage', 'form-
control', message.content, 13, 50, 'message.content');
        vp13.add(inputMessage);
        var emptyRow3 = new EmptyRow('er3', '');
        vp13.add(emptyRow3);

        var vp14;
        if(type == 'FORWARD') {
            vp14 = drawForwardButtons(message);
            vp10.add(vp14);
        }
        if(type == 'DRAFT') {
            vp14 = drawDraftButtons(id);
            vp10.add(vp14);
        }
        return vp9;
    }
}

```

Listing 2.6 Iscrtavanje panela sa porukom za slanje


```

export default function drawDraftButtons(id) {

    var vp14 = new VerticalPanel('vp14', 'compose-btn pull-left');

    var buttonSend = new AContainer('buttonSend', 'btn btn-sm btn-
primary', 'Send ', '', '');
    vp14.add(buttonSend);
    var i1 = new I('i1', 'fa fa-envelope');
    buttonSend.add(i1);
    var ec1 = new EmptyCol('emptyCol1', '');
    vp14.add(ec1);

    buttonSend.onclick = function (e) {
        e.preventDefault();
        e.stopImmediatePropagation();

        var vp7 = buttonSend.findById('vp7');
        var vp9 = buttonSend.findById('vp9');
        vp9.remove(vp7);

        let axios = window._api.axios;
        let messageManager = new MessageManager(axios);

        messageManager.sendDraft(id)
            .then(response => {
                var component = drawMessage(messageManager.message,
'MAIL');
                vp7.add(component);
            });
    }

    var buttonDiscard = new AContainer('buttonDiscard', 'btn btn-sm
btn-default', 'Discard ', '', '');
    vp14.add(buttonDiscard);
    var i2 = new I('i2', 'fa fa-trash-o');
    buttonDiscard.add(i2);
    var ec2 = new EmptyCol('emptyCol2', '');
    vp14.add(ec2);

    buttonDiscard.onclick = function (e) {

        var vp7 = buttonDiscard.findById('vp7');
        var vp9 = buttonDiscard.findById('vp9');
        vp9.remove(vp7);

        let axios = window._api.axios;
        let messageManager = new MessageManager(axios);

        messageManager.fetchMessages('INBOX')
            .then(response => {
                var component = drawInbox(messageManager.messages);
                vp7.add(component);
            });
    }
}

```

```

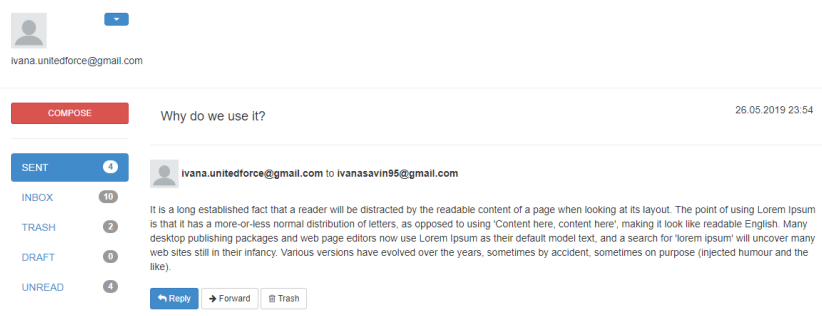
    }

    var hr2 = new HR('hr2', '');
    vp14.add(hr2);
    return vp14;
}

```

Listing 2.7 Iscrtavanje panela sa porukom za slanje 2

Nakon što je poruka poslata ona pripada labeli SENT i može se videti izborom te labele a potom i poruke. Prikaz poslate poruke se može videti na slici 3.5.



Slika 3.5 Panel sa poslatom porukom

Iscrtavanje panela sa porukom vrši metoda *drawMessage* i metode *drawMessageButtons*. Implementacije ovih metoda se nalaze na listinzima 2.8 i 2.9.

```

export default function drawMessage(message, type) {
    var vp9 = new VerticalPanel('vp9', 'col-sm-9 col-md-10');

    var vp10 = new VerticalPanel('vp10', 'inbox-body');
    vp9.add(vp10);

    var vp11 = new VerticalPanel('vp11', 'heading-inbox row');
    vp10.add(vp11);
}

```

```

var vp12 = new VerticalPanel('vp12', 'col-md-12');
vp11.add(vp12);
var h41 = new H4('h41', 'col-md-8', message.headers.subject);
vp12.add(h41);

var vp13 = new VerticalPanel('vp13', 'col-md-4 text-right');
vp12.add(vp13);
var date = new Label('date', 'date', message.headers.date);
vp13.add(date);

var vp14 = new VerticalPanel('vp14', 'col-md-12');
vp13.add(vp14);
var hr1 = new HR('hr1', 'col-md-12');
vp10.add(hr1);

var vp15 = new VerticalPanel('vp15', 'sender-info');
vp10.add(vp15);

var emptyRow2 = new EmptyRow('er2', '');
vp10.add(emptyRow2);

var vp16 = new VerticalPanel('vp16', 'row');
vp15.add(vp16);

var vp17 = new VerticalPanel('vp17', 'col-md-12');
vp16.add(vp17);

var senderImage = new Image('senderImage', '',
'./images/profile.png', message.headers.from, '40px', '40px');
vp17.add(senderImage);
var emptyCol = new EmptyCol('emptyCol1', '');
vp17.add(emptyCol);
var senderName = new Strong('senderName', '', message.headers.from);
vp17.add(senderName);
var textTo = new Label('textTo', '', ' to ');
vp17.add(textTo);
if(message.headers.to == undefined) {
    var textMe = new Strong('textMe', '', '');
    vp17.add(textMe);
} else {
    var textMe = new Strong('textMe', '', message.headers.to);
    vp17.add(textMe);
}

var vp18 = new VerticalPanel('vp18', 'view-mail');
vp10.add(vp18);

var content = new Label('message', '', message.content);
vp18.add(content);

var emptyRow3 = new EmptyRow('er3', '');
vp10.add(emptyRow3);

if (type == 'TRASH') {
    var vp19 = drawTrashButtons(message);

```

```

        vp10.add(vp19);
    } else {
        var vp19 = drawMessageButtons(message);
        vp10.add(vp19);
    }

    return vp9;
}

```

Listing 2.8 Iscrtavanje panela sa porukom

```

export default function drawMessageButtons(message) {

    var vp19 = new VerticalPanel('vp19', 'compose-btn pull-left');

    var button1 = new AContainer('button1', 'btn btn-sm btn-primary',
'Reply ', '', '');
    vp19.add(button1);
    var i4 = new I('i4', 'fa fa-reply');
    button1.add(i4);
    var emptyCol3 = new EmptyCol('ec3', '');
    vp19.add(emptyCol3);

    button1.onclick = function (e) {
        e.preventDefault();
        e.stopImmediatePropagation();

        var vp7 = button1.findById('vp7');
        var vp9 = button1.findById('vp9');
        vp9.remove(vp7);

        var component = drawComposeReply(message, 'REPLY');
        vp7.add(component);
    }

    var button2 = new AContainer('button2', 'btn btn-sm btn-default',
'Forward ', '', '');
    vp19.add(button2);
    var i5 = new I('i5', 'fa fa-arrow-right');
    button2.add(i5);
    var emptyCol4 = new EmptyCol('ec4', '');
    vp19.add(emptyCol4);

    button2.onclick = function (e) {
        e.preventDefault();
        e.stopImmediatePropagation();

        var vp7 = button2.findById('vp7');
        var vp9 = button2.findById('vp9');
        vp9.remove(vp7);
    }
}

```

```

        var component = drawDraftForward(message, 'FORWARD');
        vp7.add(component);
    }
    var button3 = new AContainer('button3', 'btn btn-sm btn-default',
'Trash ', '', '');
    vp19.add(button3);
    var i6 = new I('i6', 'fa fa-trash-o');
    button3.add(i6);

    button3.onclick = function (e) {
        e.preventDefault();
        e.stopImmediatePropagation();

        var vp7 = button3.findById('vp7');
        var vp9 = button3.findById('vp9');
        vp9.remove(vp7);

        let axios = window._api.axios;
        let messageManager = new MessageManager(axios);

        messageManager.trashMessage(message.id)
            .then(response => {
                var component = drawTrash(messageManager.messages);
                vp7.add(component);
            });
    }

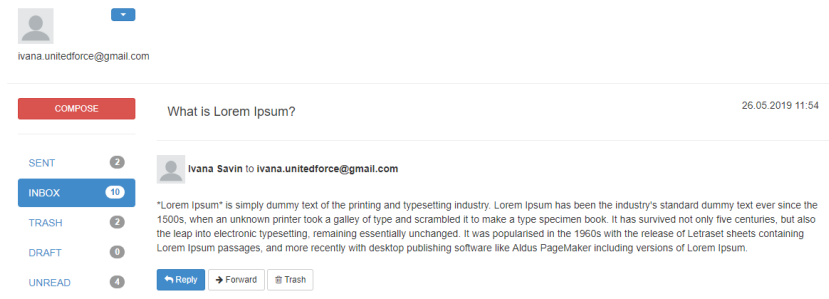
    var hr2 = new HR('hr2', '');
    vp19.add(hr2);

    return vp19;
}

```

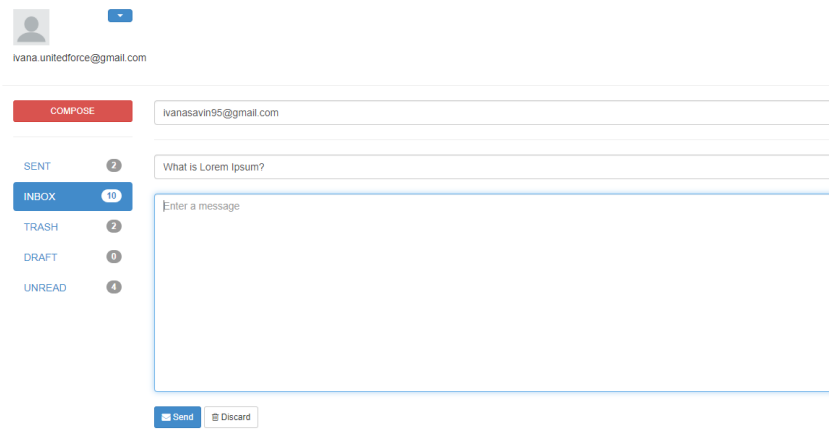
Listing 2.9 Iscrtavanje panela sa porukom 2

Nakon startovanja aplikacije na glavnom prozoru aplikacije su prikazane sve primljene poruke, klikom na neku od njih, poruka će se prikazati u novom prozoru. Prikaz cele poruke se može videti na slici 3.6. Kao i u originalnoj aplikaciji korisnik vidi naslov poruke, vreme slanja, pošiljaoca poruke i samu poruku.

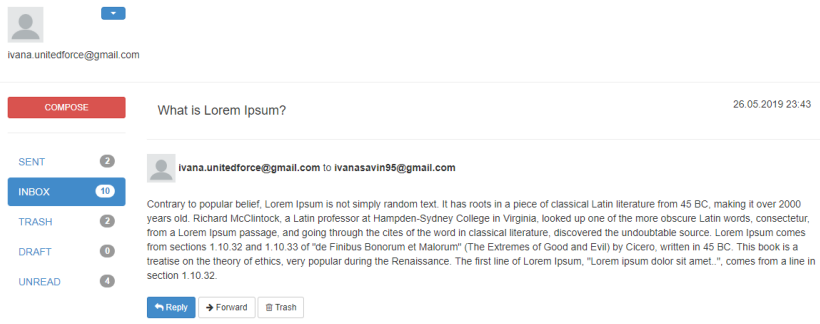


Slika 3.6 Panel sa porukom

Kada je korisniku prikazana poruka nudi mu se mogućnost da odgovori na poruku, da poruku prosledi ili je prebaci u kantu za brisanje. Ukoliko korisnik izabere da odgovori na poruku prikazaće mu se isti prozor kao na slici 3.7. Nakon što popuni polje za poruku i odgovori na poruku prikazaće mu se prozor kao na slici 3.8. Iscrtavanje prozora za odgovor na poruku je isto kao i prilikom slanja poruke i može se videti na listingu 2.4, razlika je samo u iscrtavanju dugmadi koje se može videti na listingu 2.10.



Slika 3.6 Panel za odgovaranje na poruku



Slika 3.7 Panel sa odgovorom na poruku

```
export default function drawReplayButtons() {

    var vp14 = new VerticalPanel('vp14', 'compose-btn pull-left');

    var buttonSend = new AContainer('buttonSend', 'btn btn-sm btn-
primary', 'Send ', '', '');
    vp14.add(buttonSend);
    var i1 = new I('i1', 'fa fa-envelope');
    buttonSend.add(i1);
    var ec1 = new EmptyCol('ec1', '');
    vp14.add(ec1);

    buttonSend.onclick = function (e) {
        e.preventDefault();
        e.stopImmediatePropagation();

        var to = inputEmail.value;
        var subject = inputTitle.value;
        var bodyText = inputMessage.value;

        var vp7 = buttonSend.findById('vp7');
        var vp9 = buttonSend.findById('vp9');
        vp9.remove(vp7);

        let axios = window._api.axios;
        let messageManager = new MessageManager(axios);

        messageManager.sendMessage(to, subject, bodyText)
            .then(response => {
                var component = drawMessage(messageManager.message,
'MAIL');
                vp7.add(component);
            });
    });
}
```

```

    }

    var buttonDiscard = new AContainer('buttonDiscard', 'btn btn-sm btn-
default', 'Discard ', '', '');
    vp14.add(buttonDiscard);
    var i3 = new I('i3', 'fa fa-trash-o');
    buttonDiscard.add(i3);
    var ec3 = new EmptyCol('ec3', '');
    vp14.add(ec3);

    buttonDiscard.onclick = function (e) {
        e.preventDefault();
        e.stopImmediatePropagation();

        var vp7 = buttonDiscard.findById('vp7');
        var vp9 = buttonDiscard.findById('vp9');
        vp9.remove(vp7);

        let axios = window._api.axios;
        let messageManager = new MessageManager(axios);

        messageManager.fetchMessages('INBOX')
            .then(response => {
                var component = drawInbox(messageManager.messages,
'MAIL');
                vp7.add(component);
            });
    }

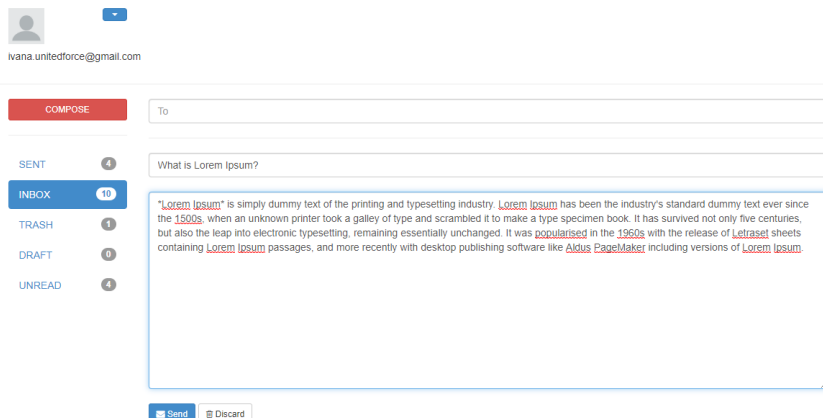
    var hr2 = new HR('hr2', '');
    vp14.add(hr2);

    return vp14;
}

```

Listing 2.10 Iscrtavanje panela sa odgovorom na poruku

Ukoliko korisnik izabere da poruku prosledi prikazaće mu se prozor kao na slici 3.8.



Slika 3.8 Panel za prosleđivanje poruke

Iscrtavanje prozora za prosleđivanje poruke vrši se iz dve metode *drawDraftForward* koja je prikazana na listingu 2.3 i *drawForwardButtons* (listing 2.9).

```
export default function drawForwardButtons(message) {

    var vp14 = new VerticalPanel('vp14', 'compose-btn pull-left');

    var buttonSend = new AContainer('buttonSend', 'btn btn-sm btn-
primary', 'Send ', '', '');
    vp14.add(buttonSend);
    var i1 = new I('i1', 'fa fa-envelope');
    buttonSend.add(i1);
    var ec1 = new EmptyCol('emptyCol1', '');
    vp14.add(ec1);

    buttonSend.onclick = function (e) {
        e.preventDefault();
        e.stopImmediatePropagation();

        var to = inputEmail.value;

        var vp7 = buttonSend.findById('vp7');
        var vp9 = buttonSend.findById('vp9');
        vp9.remove(vp7);

        let axios = window._api.axios;
        let messageManager = new MessageManager(axios);
```

```

        messageManager.sendMessage(to, message.headers.subject,
message.content)
        .then(response => {
            var component = drawMessage(messageManager.message,
'MAIL');
            vp7.add(component);
        });

    }

    var buttonDiscard = new AContainer('buttonDiscard', 'btn btn-sm btn-
default', 'Discard ', '', '');
    vp14.add(buttonDiscard);
    var i2 = new I('i2', 'fa fa-trash-o');
    buttonDiscard.add(i2);
    var ec2 = new EmptyCol('emptyCol2', '');
    vp14.add(ec2);

    buttonDiscard.onclick = function (e) {
        e.preventDefault();
        e.stopImmediatePropagation();

        var vp7 = buttonDiscard.findById('vp7');
        var vp9 = buttonDiscard.findById('vp9');
        vp9.remove(vp7);

        let axios = window._api.axios;
        let messageManager = new MessageManager(axios);

        messageManager.fetchMessages('INBOX')
            .then(response => {
                var component = drawMessage(messageManager.messages);
                vp7.add(component);
            });
    }

    var hr2 = new HR('hr2', '');
    vp14.add(hr2);

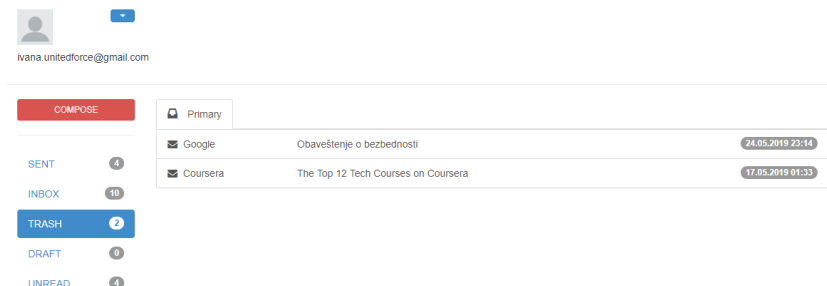
    return vp14;
}

```

Listing 2.9 Iscrtavanje panela za prosledivanje poruke 2

Nakon što korisnik klikne na dugme *trash* kao na slici 3.5 poruka će biti prebačena u poruke za brisanje i biće joj dodeljena sistemska labela TRASH. Klikom na labelu TRASH biće prikazane sve poruke za brisanje (slika 3.9) a izborom jedne od njih prikazaće se panel sa porukom i sa

ponuđenim izborom konačnog brisanja poruke ili opozivanja akcije brisanja. Ovaj prozor se može videti na slici 3.10.



Slika 3.9 Panel sa porukama za brisanje



Slika 3.10 Panel sa porukom za brisanje

4.1.2 Metode za komunikaciju sa serverskom aplikacijom

Kako bi aplikacija radila potrebno je obezbediti komunikaciju između serverske i klijentske strane. Na klijentskoj strani se koristi *axios* biblioteka pomoću koje se šalju HTTP zahtevi ka serveru. Metode za komunikaciju su obuhvaćene u tri klase *MessageManager* za razmenu podataka o porukama, *LabelManager* za razmenu poruka o labelama i *UserManager* za razmenu podataka o korisniku. Implementacija *MessageManager* klase se nalazi na listingu 2.10.

```
export default class MessageManager {
  constructor(axiosApi) {
    this.messages = [];
    this.message = {};
    this.labels = [];

    this.axios = axiosApi;
    this.url = 'http://localhost:9000';
  }

  fetchAllMessages() {
    return this.axios.get(this.url + "/allMessages")
      .then((response) => {
        this.messages = response.data;
      });
  }

  fetchMessages(label) {
    return this.axios.get(this.url + "/messages?label=" + label)
      .then((response) => {
        this.messages = response.data;
      });
  }

  fetchDrafts() {
    return this.axios.get(this.url + "/drafts")
      .then((response) => {
        this.drafts = response.data;
      });
  }

  fetchMessage(id) {
    return this.axios.get(this.url + "/message?id=" + id)
      .then((response) => {
        this.message = response.data;
      });
  }

  draftMessage(to, subject, bodyText) {
    return this.axios.post(this.url + "/draft", {
```

```

        to: to,
        subject: subject,
        bodyText: bodyText
    }).then((response) => {
        this.messages = response.data;
    });
}

sendMessage(to, subject, bodyText) {
    var re =
/^(((^<>()\[\]\.\.,;\s@""]+(\.^[^<>()\[\]\.\.,;\s@""]+)*|(".*"))@((\[[0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\)|((\a-zA-Z\d\-\_)+[a-zA-Z]{2,})))$/;
    if (re.test(String(to).toLowerCase()) && subject.length > 0 &&
bodyText.length > 0) {
        return this.axios.post(this.url + "/send", {
            to: to,
            subject: subject,
            bodyText: bodyText
        }).then((response) => {
            this.message = response.data;
        });
    }
}

sendDraft(id) {
    return this.axios.post(this.url + "/sendDraft?id=" + id)
        .then((response) => {
            this.message = response.data;
        });
}

trashMessage(id) {
    return this.axios.post(this.url + "/trash?id=" + id)
        .then((response) => {
            this.messages = response.data;
        });
}

untrashMessage(id) {
    return this.axios.post(this.url + "/untrash?id=" + id)
        .then((response) => {
            this.messages = response.data;
        });
}

deleteMessage(id) {
    return this.axios.post(this.url + "/delete?id=" + id)
        .then((response) => {
            this.messages = response.data;
        });
}

getMessages() {
    return this.messages;
}

```

```

    getMessage() {
        return this.message;
    }
}

```

Listing 2.10 Metode za komunikaciju sa serverom

4.2 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.

4.2.1 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 već spomenuto, klasa *Controller* sadrži metode koji šalju pozive ga *Gmail API*-ju poput metode za autentikaciju, metode za prikupljanje korisnikovih labela ili poruka. Na listingu 2.11 je prikazana metoda za autentikaciju aplikacije i prikazivanje *Gmail* prozora za autentikaciju korisnika. Budući da se koristi *Gmail API*, autentikacija korisnika je realizovana od strane Gmail servisa. Zbog preglednosti na listinzima 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);
}

```

Listing 2.11 Implementacija metode za autentikaciju aplikacije

Gmail API u svom podešavanju nudi mogućnost redirektovanja na adresu korisnikove aplikacije nakon što se aplikacija autentikuje. Budući da se naša aplikacija sastoji od dva dela, klijentski i serverski deo koji su podignuti na različitim serverima. *Gmail* servis nakon autentikacije aplikaciju redirektuje na URL na serveru koji potom šalje poziv klijentskoj aplikaciji. Nakon što se izvrši redirektovanje, korisnik je ulogovan i popunjava se polje *credential* koje se koristi za autentikaciju korisnika pri svakom sledećem *Gmail API* pozivu. Implementacija ove metode je prikazana na listingu 2.12.

```

@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;
}

```

Listing 2.12 Metoda za redirektovanje

Nakon što je korisnik ulogovan mogu se slati ostali *Gmail API* pozivi. Na listingu 2.13 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);
}

```

Listing 2.13 Metoda za dobavljanje podataka o korisniku

Na listingu 2.14 prikazana je *GmailController* klasa. Budući da su metode *login*, *callback* i *getMe* već prikazane neće biti prikazane na sledećem listingu.

```

@CrossOrigin(origins = "http://localhost:5500")
@RestController
@RequestMapping(value = "")
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
    UtilService utilService;

    @Autowired
    FetchService fetchService;
}

```



```

@Autowired
GmailService service;

.
.
.

@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 = fetchLabels(labelsResponse);

    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 = fetchService.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 = fetchService.fetchFullMessage(message);

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

}

@RequestMapping(value = "/sendDraft",
                method = RequestMethod.POST,
                produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<String> sendDraft(@RequestParam(value = "id")
String id) {

    Draft draft = client.users().drafts().get(USER_ID, id).execute();

    JSONObject messageFromDraft =
fetchService.fetchDraftForSend(draft.getMessage());

    Message temp = service.prepareForSend(messageFromDraft.toString(),
USER_ID);
    Message send = client.users().messages().send(USER_ID,
temp).execute();

    client.users().drafts().delete(USER_ID, id).execute();

    Message message = client.users().messages().get(USER_ID,
send.getId()).setFormat("full").execute();
    JSONObject messageJSON = fetchService.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");

        ListDraftsResponse draftsResponse =
        client.users().drafts().list(USER_ID).execute();

        JSONArray draftArray = fetchDrafts(draftsResponse);

        return new ResponseEntity<>(draftArray.toString(), HttpStatus.OK);
    }

    @RequestMapping(value = "/drafts",
        method = RequestMethod.GET,
        produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<String> getDrafts() {

        ListDraftsResponse draftsResponse =
        client.users().drafts().list(USER_ID).execute();

        JSONArray draftArray = fetchDrafts(draftsResponse);

        return new ResponseEntity<>(draftArray.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("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 = "/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(USER_ID,
msg.getId()).execute();

        JSONObject messageJSON =
fetchService.fetchMessage(message);
        messageArray.put(messageJSON);
    }

    return messageArray;
}

private JSONArray fetchDrafts(ListDraftsResponse draftsResponse) {

    JSONArray draftArray = new JSONArray();
    if(draftsResponse.getDrafts() == null) {
        return draftArray;
    }

    for (Draft d : draftsResponse.getDrafts()) {

        Draft draft = client.users().drafts().get(USER_ID,
d.getId()).execute();

        JSONObject draftJSON = fetchService.fetchDraft(draft);
        draftArray.put(draftJSON);
    }

    return draftArray;
}

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 = fetchService.fetchLabel(label);
        if(labelJSON.getString("labelListVisibility")
.equals("labelShow")) {
            labelArray.put(labelJSON);
        }
    }
    return labelArray;
}
}

```

Listing 2.14 *GmailController* klasa

4.2.2 Service

Servisni sloj aplikacije čine 3 servisa: *GmailService*, *FetchService* i *UtilService*. *GmailService* sadrži pomoćne metode specifične za *Gmail API*. Na listingu 2.15 se može videti implementacija 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;
    }
}

```

Listing 2.15 *GmailService* klasa

Radi uštede resursa i lakšeg korišćenja podaci koje vrati *Gmail* servis se transformišu u odgovarajući JSON objekat. *FetchService* sadrži metode koje transformišu potrebne podatke a implementacija se nalazi na listingu 2.16.

```

@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;
}

public JSONObject fetchDraftForSend(Message message) {

    JSONObject json = new JSONObject();
    for (MessagePartHeader header : message.getPayload().getHeaders())
    {
        if(header.getName().equals("Subject")) {
            json.put("subject", header.getValue());
        }
        if(header.getName().equals("To")) {
            json.put("to", header.getValue());
        }
    }
    String content = "";
    if(message.getPayload().getBody().getData() != null) {
        content = StringUtils.newStringUtf8(Base64.decodeBase64(
            message.getPayload().getBody().getData()));
    }

    json.put("bodyText", content);

    return json;
}
}

```

Listing 2.16 Implementacija FetchService klase

Da bi pristup *Gmail* servisima bio moguć korisnik mora da specificira *scope* stringove koje će svom nalogu dozvoliti. Na listingu 2.17 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;
    }
}

```

Listing 2.17 *UtilService* klasa

5. Zaključak

Osnovni zadatak ovog rada je bio razvoj i demonstracija *JavaScript* biblioteke za rad sa komponentama grafičkog korisničkog interfejsa – *OldSchoolComponents*. Osnovna ideja je bila da se komponente korisničkog interfejsa mogu deklarativno napraviti i dodati na ekran, kako se inače radi u programskim jezicima koji se ne koriste za razvoj veb aplikacija.

Za demonstraciju *OldSchoolComponents* biblioteke, razvijena je jednostavna aplikacija za upravljanje elektronskom poštom, inspirisana 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>
- [2] OldSchoolComponents biblioteka
<https://github.com/iv17/OldSchoolComponents>
- [3] Google Mail
<https://mail.google.com/mail/>
- [4] Spring
<https://spring.io/>
- [5] Google Mail API
<https://developers.google.com/gmail/api/v1/reference/>
- [6] jQuery
<https://api.jquery.com/>
- [7] Bootstrap
<https://getbootstrap.com/>
- [8] axios
<https://github.com/axios/axios>
- [9] Java
[https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))
- [10] Spring Boot
<https://spring.io/projects/spring-boot>
- [11] ECMAScript 6
<https://en.wikipedia.org/wiki/ECMAScript>
- [12] Google API Wiki
https://en.wikipedia.org/wiki/Google_APIs
- [13] Apache Maven
https://en.wikipedia.org/wiki/Apache_Maven
- [14] OAuth 2.0
<https://en.wikipedia.org/wiki/OAuth>
- [15] 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))

6.KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Редни број, РБР :	
Идентификациони број, ИБР :	
Тип документације, ТД :	монографска публикација
Тип записа, ТЗ :	текстуални штампани
Врста рада, ВР :	Мастер рад
Аутор, АУ :	Ивана Савин
Ментор, МН :	Проф Др Милан
Наслов рада, НР :	Развој JavaScript библиотеке за рад са компонентама корисничког интерфејса
Језик публикације, ЈП :	српски
Језик извода, ЈИ :	српски / енглески
Земља публиковања, ЗП :	Србија
Уже географско подручје, УГП :	Војводина
Година, ГО :	2019
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 / 78 / 0 / 0 / 17 / 30 / 0
Научна област, НО :	Информатика
Научна дисциплина, НД :	Рачунарске науке
Предметна одредница/Кључне речи, ПО :	JavaScript, HTML, DOM, Gmail API, ВЕБ, REST
УДК	

Чува се, ЧУ:		Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, Нови Сад	
Важна напомена, ВН:			
Извод, ИЗ:		<p>Задатак рада представља развој <i>JavaScript</i> библиотеке за рад са компонентама корисничког интерфејса. За демонстрацију рада ове библиотеке, биће написана једноставна веб апликација за управљање електронском поштом. Клијентски део апликације ће бити развијен помоћу претходно развијене <i>JavaScript</i> библиотеке. Серверски део апликације ће бити реализован у програмском језику Јава, коришћењем <i>Spring</i> окружења. Апликација ће омогућити основни скуп операција, попут управљања електронском поштом, креирања и слања поште. Спецификација ће бити представљена UML дијаграмима.</p>	
Датум прихватања теме, ДП:			
Датум одбране, ДО:			
Чланови комисије, КО:	Председник:	др Ђорђе Херцег, редовни професор, ПМФ Нови Сад	

	Члан:	др Игор Дејановић, 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 :	Master thesis
Author, AU :	Ivana Savin
Mentor, MN :	Prof Dr Milan Vidaković
Title, TI :	DEVELOPMENT OF JavaScript LIBRARY FOR USER INTERFACE 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 :	2019
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 / 78 / 0 / 0 / 17 / 30 / 0
Scientific field, SF :	Electrical Engineering
Scientific discipline, SD :	Computer Science
Subject/Key words, S/KW :	JavaScript, HTML, DOM, Gmail API, WEB, REST
UC	
Holding data, HD :	Library of the Faculty of Technical Sciences,
Note, N :	

Abstract, AB :		The thesis deals with the implementation of JavaScript library for creating user interfaces. For demonstration how this library works will be created simple web application for email management. The client part of an application will be implemented using the previous created JavaScript library. The server part will be implemented using Java programming language and Spring framework. The application will provide basic managing features as listing, reading, composing and sending email messages.	
Accepted by the Scientific Board on, ASB :			
Defended on, DE :			
Defended Board, DB :	President:	Djordje Herceg, PhD, full prof., PMF Novi Sad	
	Member:	Igor Dejanović, 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. Након завршених основних студија уписала је мастер академске студије на истом факултету, смер Софтверско инжењерство и информационе технологије, модул Електронско пословање. Положила је све испите предвиђене планом и програмом.

Контакт емајл адреса:

ivanasavin95@gmail.com

Контакт мобилни:

065/84-76-018

