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

Арсеније Карпић

**ВЕБ АПЛИКАЦИЈА ЗА УПРАВЉАЊЕ ИМЕЈЛ ПОРУКАМА**

ДИПЛОМСКИ РАД

- Основне академске студије -

Нови Сад, 2019

|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ ⚫ **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА** | Лист/Листова: |
| 2/64 |

*(Податке уноси предметни наставник - ментор)*

| Врста студија: | Основне академске студије  Основне струковне студије |
| --- | --- |
| Студијски програм: | **Рачунарство и аутоматика** |
| Руководилац студијског програма: | **Проф. др Милан Видаковић** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | **Арсеније Карпић** | Број индекса: | **RA55/14** |
| 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 ДИПЛОМСКОГ (BACHELOR) РАДА:**

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

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| Задатак представља развој једноставне веб апликације за управљање имејл порукама која је инспирисана већ постојећим решењем (Google Gmail). Серверски део апликације није посебно развијан док је клијентски део апликације развијен у VueJS 2 радном оквиру. Апликација омогућава листање, читање, писање и претраживање имејл порука. 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 5](#_Toc1331403)

[2. Opis korišćenih tehnologija 7](#_Toc1331404)

[2.1. Vue radni okvir 7](#_Toc1331405)

[2.1.1. Vue instanca 8](#_Toc1331406)

[2.1.2. Komponente 9](#_Toc1331407)

[2.1.3. Rutiranje 11](#_Toc1331408)

[2.1.4. Vuex 12](#_Toc1331409)

[2.1.5. Vue Gapi 14](#_Toc1331410)

[2.2. Google Gmail API 16](#_Toc1331411)

[2.2.1. Podešavanje Google Gmail API 16](#_Toc1331412)

[2.2.2. API pregled 16](#_Toc1331413)

[3. Specifikacija aplikacije 19](#_Toc1331414)

[3.1. Dijagram slučajeva korišćenja 19](#_Toc1331415)

[3.2. Dijagrami sekvence 20](#_Toc1331416)

[3.3. Dijagram Vue komponenti 23](#_Toc1331417)

[4. Opis implementacije 27](#_Toc1331418)

[4.1. Klijentski deo 27](#_Toc1331419)

[4.1.1. Struktura projekta (Webpack i Babel) 27](#_Toc1331420)

[4.1.2. Centralizovano praćenje stanja aplikacije 29](#_Toc1331421)

[4.1.3. Implementacija komponenti 34](#_Toc1331422)

[5. Zaključak 61](#_Toc1331423)

[Literatura 63](#_Toc1331424)

[Ključna dokumentacijska informacija 65](#_Toc1331425)

[Key Words Documentation 67](#_Toc1331426)

[Biografija 69](#_Toc1331427)

# Uvod

Tema rada je razvoj web aplikacije za upravljanje e-mejlovima. Aplikacija je pravljena po uzoru na već postojeće rešenje (Gmail) [1].

Sa obzirom da se radi o web aplikaciji, bilo je potrebno razviti klijentski deo, a za serverski deo korišćen je već postojeći Gmail API [2].

Budući da je bilo neophodno koristiti Gmail API [2], veliki izazov bio je pronaći biblioteku za klijentski deo koja na najlaši način pruža mogućnost rada sa pomenutim API-jem.

Prvo poglavlje je uvodno, dok je u drugom poglavlju predstavljen opis korišćenih tehnologija. Treće poglavlje sadrži specifikaciju aplikacije, potpomognutu UML dijagramima. U četvrtom poglavlju prestavljen je opis konkretne implementacije web aplikacije, dok se u petom poglavlju nalazi zaključak.

# Opis korišćenih tehnologija

Serverski deo aplikacije nije posebno razvijan u okviru rada. Budući da je tema rada aplikacija koja za svrhu ima upravljanje elektronskim porukama, za serverski deo korišćen je Guglov Gmail API [2] koji omogućava lako manipulisanje već postojećim Gugl nalozima.

Klijentski deo aplikacije implementiran je pomoću Vue.js 2 radnog okvira [3] sa Javascript programskim jezikom [4]. Rad sa Guglovim Gmail API je pojednostavljen korišćenjem Vue-Gapi biblioteke [4] u okviru aplikacije. Zbog jednostavnijeg razvoja aplikacije korišćen je Vue CLI (*Command Line Interface*) [5].

U narednom delu teksta za Vue.js 2 će se navoditi samo Vue.

## Vue radni okvir

Vue je radni okvir koji služi za kreiranje klijentskih aplikacija. U svom jezgru, Vue je fokusiran samo na *View* deo MVC (*Model View Controller*) dizajn šablona, što omogućava veoma laku integraciju sa mnogim drugim bibliotekama ili već postojećim projektima. Sa druge strane, kada se koristi uz moderne alate, Vue je moguće koristiti za implementaciju SPA (*Single Page Application*) jer pruža mogućnost kreiranja komponenti.

Svaka Vue aplikacija počinje kreiranjem Vue instance (*Vue Instance).* Tipična Vue aplikacija se sastoji od korenske (*root*) Vue instance i određenog broja komponenti koje su organizovane u hijerarhiju stabla. Komponente je moguće ponovno iskoristiti u bilo kom delu aplikacije. Svaka komponenta je zadužena za obavljanje specifične funkcionalnosti, čime se postiže raščlanjivanje velike aplikacije na manje i lakše razumljive celine.

Prelazak sa jedne komponente na drugu postiže se rutiranjem. Funkcionalnost rutiranja pruža Vue Router [6], koji je oficielno podržan od strane Vue razvojnog tima. Veoma je laka integracija drugih (*3rd party*) rutera.

Velike aplikacije često veoma lako mogu da narastu u složenosti. Delovi stanja mogu biti rasuti širom aplikacije u raznim komponentama. Iz tog razloga Vue pruža mogućnost praćenja i upravljana stanjem aplikacije (*Application state*) na jednom mestu uz pomoć Vuex biblioteke (*State management library*) [7].

### Vue instanca

Kao što je već rečeno, svaka Vue aplikacija počinje kreiranjem Vue instance. Kreiranje Vue instance vrši se pomoću Vue funkcije, kao što je moguće videti na slici 1.

let vm = new Vue({

//options

});

Slika - Kreiranje Vue instance

Prilikom kreiranja Vue instance, Vue funkciji se prosleđuje objekat sa opcijama pomoću kog je moguće parametrizovati ponašanje Vue instance. Neki od tipičnih parametara su:

* el – Omogućuje Vue instanci da se poveže sa određenim DOM (*Document Object Model*) elementom,
* data – Služi kao kontejner za podatke kreirane Vue instance. Vue će svaki property konvertovati u getter/setter par da bi mogao da reaguje na promene podataka,
* methods – Sadrži metode koje je moguće izvršiti na kreiranoj Vue instaci. Svaki method imaće svoj **this** podešen da pokazuje na kreiranu Vue instancu,
* template – Omogućuje prosleđivanje stringa kreiranoj Vue instanci koji će biti korišćen kao njen markap (*markup*).

Da bi korišćenje Vue instance imalo smisla pri kreiranju klijentske aplikacije, Vue instanca mora biti povezana sa određenim markapom, tj. sa odredjenim delom HTML (*Hyper Text Markup Language*). To je moguće uraditi na dva načina, korišćenjem gore pomenutih opcija **el** ili **template**. Razlika je u tome što se prilikom korišćenja el opcije Vue instanca vezuje za postojeći HTML što znači da će se prilikom kreiranja Vue instance outerHTML označenog DOM elementa kompajlirati u templejt (*template*). Prilikom korišćenja template opcije, string koji je prosleđen kompajlira se u render funkciju. Na slici 2 prikazana je razlika u korišćenju ove dve opcije povezivanja Vue instance sa određenim DOM elementima.

**HTML**

<div id="#app">

</div>

**JAVASCRIPT**

let app = new Vue({

el: '#app'

})

let app = new Vue({

template: `<div id="app">

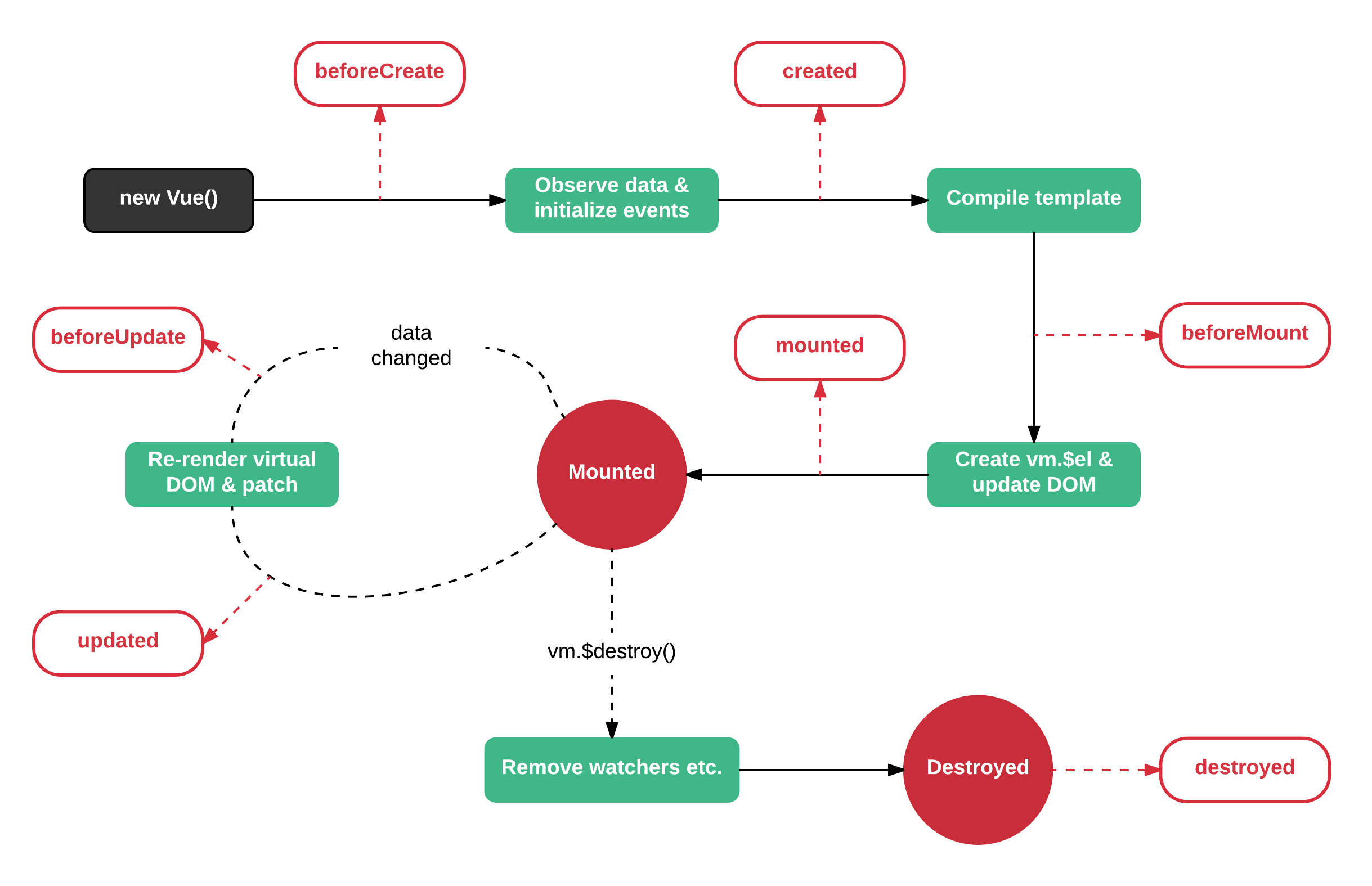
</div>`

});

Slika - Levo: korišćenje el opcije, desno: korišćenje template opcije

U većini slučajeva, na ovaj način se kreira korenska Vue instanca koja služi kao kontejner za komponente, o kojima će biti reči u nastavku teksta.

Vue instanca prolazi kroz nekoliko životnih ciklusa (*Lifecycle hooks*). Za svaki životni ciklus instance moguće je definisati aplikativnu logiku. Na primer, moguće je definisati ponašanje prilikom kreiranja instance (*created hook*) ili prilikom njenog uništenja (*destroyed hook*). Na slici 3 može se videti dijagram životnog ciklusa Vue instance.

[[1]](#footnote-1)

Slika 3 - Životni ciklus Vue instance

### Komponente

Vue pruža mogućnost kreiranja komponenti, kao osnovnih gradivnih jedinica aplikacije. Svaka komponenta zadužena je za specifičnu funkcionalnost.

Vue koristi mehanizam *Single File Components*. Ovaj mehanizam omogućava da se prikaz (HTML *template*), aplikativna logika (*Javascirpt*) i stilovi (CSS) nalaze u jednom fajlu. Ovi fajlovi obično imaju **.vue** ekstenziju.

U daljem tekstu se pod komponentom smatra *Single File Component.*

#### Struktura komponente

Vue komponenta se sastoji iz tri dela:

* template – sadrži sav HTML koji služi za prikaz unutar komponente,
* script – sadrži svu aplikativnu logiku komponente implementiranu pomoću Javascript programskog jezika,
* style – sadrži sve CSS stilove koji se odnose na *template* unutar komponente.

Na slici broj 4 može se videti primer jednostavne Vue komponente koja kroz *string interpolation* mehanizam prikazuje tekst.

Slika - Primer Vue komponente

<template>

<div>

<p>{{ helloWorld }}</p>

</div>

</template>

<script>

export default {

data() {

return {

helloWorld: 'Hello World!'

}

}

};

</script>

<style scoped>

</style>

#### Registrovanje komponente

Vue komponente su organizovane u hijerarhijsku strukturu stabla. Unutar svake komponente se kroz *components* opciju u *script* delu mogu lokalno registrovati komponente. Svaka lokalno registrovana komponenta dostupna je unutar *template* dela kroz svoj definisani *tag*. Na slici broj 5 može se videti primer lokalnog registrovanja komponenti.

Slika - Registrovanje komponenti

<template>

<div>

<app-header></app-header>

<app-sign-in></app-sign-in>

</div>

</template>

<script>

export default {

components: {

appHeader: Header,

appSignIn: SignIn

}

};

</script>

<style scoped>

</style>

Kao što je na slici 5. prikazano registrovane komponente (*Header, SignIn*) su dostupne u *template* delu kroz tagove **<app-header>** i **<app-sign-in>.**

### Rutiranje

Prelazak sa jedne komponente na drugu postiže se rutiranjem. Vue pruža mogućnost integracije različitih (*3rd party*) rutera. Međutim, ruter koji je zvanično podržan od strane Vue razvojnog tima je *Vue Router* [6]*.*

Budući da je aplikacija organizovana kao hijerarhijska struktura komponenti, rutiranje se postiže kroz dva koraka:

* povezivanje komponenti sa rutama (*route mapping*),
* određivanje mesta prikazivanja komponenti uz pomoć <router-view> taga.

Na slici 6 prikazan je primer povezivanja komponenti sa rutama. Kroz *meta* opciju moguće je u rutama preneti razne meta podatke. Pored toga, u rutama je moguće preneti *query* i *route* parametre. Meta podacima kao i parametrima moguće je pristupiti u bilo kojoj komponenti kroz **this.$route**.

Slika - Primer povezivanja komponenti sa rutama

export const routes = [

{

path: '/home',

component: Home,

meta: {

requiresAuth: true

},

children: [

{

path: 'compose',

name: 'compose',

component: Compose,

meta: {

requiresAuth: true

}

}

path: 'all',

name: 'allMessages',

component: AllMessages,

meta: {

requiresAuth: true

}

}

*Vue Router* prikazuje komponente na mestu gde se nalazi <router-view> *tag.* Tag se može nalaziti unutar *template* dela bilo koje komponente.

### Vuex

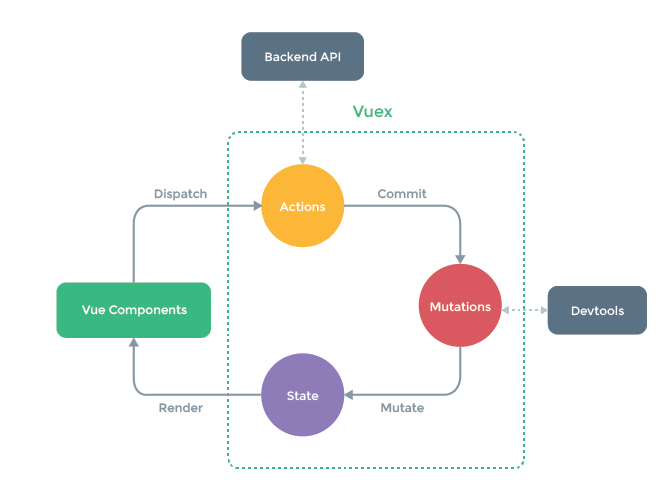
Veliki problem aplikacija koje se baziraju na komponentama (*Single Page Application*) je što se stanje aplikacije (podaci i funkcionalost) nalazi razbacano širom aplikacije, u raznim komponentama. U takvim uslovima teško je pratiti stanje i funkcionalnost aplikacije.

*Vuex* [7] ujedno predstavlja dizajn šablon i bliblioteku (*State management library*) koja rešava upravo gore pomenuti problem.Iskorišćava granularnost Vue radnog okvira kako bi bilo moguće centralizovano praćenje stanja aplikacije.

Suštniski, Vuex je aplikacija unutar aplikacije implementirana kroz *Singleton* šablon. Sastoji se iz nekoliko delova:

* *state* – Vuex koristi *Single State Tree*, tj. omogućava da se stanje aplikacije (suštinski, podaci kojima aplikacija raspolaže) prati kroz jedan Javascript objekat,
* *getters* – Vuex omogućava definisanje aplikativne logike za prikaz podataka,
* *mutations* – jedini način da se stanje aplikacije promeni je kroz mutacije. Mutacije su jednostavne, sinhrone funkcije koje menjaju stanje aplikacije,
* *actions* – akcije su veoma slične mutacijama sa jednom suštinskom razlikom što akcije mogu da sadrže asinhrone pozive. Akcije pozivaju mutacije,
* *modules* – Vuex pruza mogućnost modularizacije praćenja stanja aplikacije.

Korišćenjem ovog šablona u velikim SPA (Single Page Application) omogućeno je da se stanje aplikacije prati i menja centalizovano u bilo kom njenom delu, u bilo kojoj komponenti. Na slici broj 7 prikazan je tok praćenja i menjanja stanja aplikacije unutar Vue komponenti.

[[2]](#footnote-2)

Slika - Tok praćenja i menjanja stanja aplikacije

### Vue Gapi

Kao što je već pomenuto, serverski deo aplikacije nije razvijan. Za upravljanje e-mejlovima korišćen je već postojeci *Google Gmail API*, koji pruža lako upravljanje porukama i *Google* nalozima.

Jedan od problema koje je bilo potrebno rešiti je komunikacija između klijentske aplikacije i *Google Gmail API*. U te svrhe korišćena je *Vue Gapi* biblioteka koja omogućava lako korišćenje svih *endpoint* koje pruža *Google Gmail API*. Više o biblioteci može se pronaći na github stranici: <https://github.com/cedpoilly/vue-gapi> ili na npm stranici: <https://www.npmjs.com/package/vue-gapi>

Biblioteka je laka za korišćenje i poseduje detaljnu dokumentaciju. Biblioteku je moguće instalirati putem *Node Package Manager.*

npm install vue-gapi

Slika - Instalacija Vue Gapi biblioteke

Nakon instalacije, biblioteku je potrebno povezati sa *root* instancom Vue aplikacije. Na slici 9 prikazan je postupak povezivanja biblioteke i Vue instance.

Slika - Povezivanje Vue Gapi biblioteke i Vue instance

import Vue from 'vue'

import VueGAPI from 'vue-gapi';

const apiConfig = {

apiKey: 'API KEY',

clientId: 'CLIENT ID',

discoveryDocs: ["https://www.googleapis.com/discovery/v1/apis/gmail/v1/rest"],

scope: "https://www.googleapis.com/auth/gmail.readonly https://www.googleapis.com/auth/gmail.send https://www.googleapis.com/auth/gmail.modify https://mail.google.com/"

};

Vue.use(VueGAPI, apiConfig);

Prilikom povezivanja *Vue Gapi* biblioteke i Vue instance potrebno je proslediti konfiguracioni Javascript objekat. Konfiguracioni objekat sadrži:

* *apiKey –* ključ pomoću kog *Google Gmail API* identifikuje korisnike,
* *clientId –* identifikaciona oznaka korisnika,
* *discoveryDocs –* putanja na kojoj se nalazi dokumentacija vezana za *Google Gmail API*,
* *scope –* određuje nivo pristupa

Nakon izvršenog povezivanja, na *root* Vue instanci dostupan je novi metod, **$getGapiClient**. Povratna vrednost ovog metoda je *Promise* koji sadrži inicijalizovanu instancu *Google Gmail API* klijenta. Pomoću ovog klijenta se vrše i pozivaju sve funkcionalnosti koje nudi *Google Gmail API*. Primer korišćenja metoda je prikazan na slici 10.

Slika - Primer korišćenja dobijenog metoda

this.$getGapiClient()

.then(gapi => {

//Implementacija bilo koje fukcionalnosti

//koju nudi Google Gmail API pomoću dobijenog

//gapi klijenta

});

## Google Gmail API

*Google Gmail API* [2] pruža mogućnost brzog i fleksibilnog *RESTful* pristupa funkcionalostima kao što su:

* upravljanje elektronskim porukama (*email messages, threads*),
* pretraživanje poruka
* upravljanje labelama (*labels*),
* upravljanje notifikacijama (*push notifications*),
* upravljanje podešavanjima vezanim za *Gmail* nalog.

Postoje implementacije za komunikaciju sa *Gmail API* u programskim jezicima: *Java, C#, PHP, Python, Javascript*. Sa obzirom na to da je za implementaciju klijetskog dela aplikacije korišćen Vue radni okvir baziran na Javascript programskom jeziku, sva komunikacija sa Gmail API je implementirana upravo u tom programskom jeziku.

Gmail API poseduje veoma detaljnu dokumentaciju [2]. Takođe i primere implementirane u svim gore pomenutim programskim jezicima.

### Podešavanje Google Gmail API

Gmail API moguće je koristiti putem bilo kog Gmail naloga (u svrhe razvoja aplikacije napravljen je testni nalog). Neophodno je omogućiti Gmail API. *Setup wizard* na sledećem linku čini taj proces veoma lakim: <https://console.developers.google.com/flows/enableapi?apiid=gmail&pli=1>.

Rezultat podešavanja su *API Key* i *Client ID* koji su neophodni prilikom bilo koje komunikacije sa funkcionalnostima koje nudi *Gmail API*.

Inicijalizacija Google Gmail API klijenta i njegovo korišćenje unutar Vue radnog okvira biće detaljnije opisano u poglavlju *Implementacija aplikacije*.

### API pregled

Google Gmail API je veb servis. Koristi RESTful način komunikacije pri čemu se koriste JSON reprezentacije resursa.

#### Tipovi resursa

*Google gmail API* nudi nekoliko ključnih tipova resursa:

* *message –* resurs koji nije promenjiv. Poruke mogu biti kreirane ili obrisane ali se ni jedno svojstvo poruke ne može izmeniti,
* *label –* resurs koji služi kao sredstvo kategorizacije poruka. Label resurs ima many-to-many odnos sa porukama. Na jednu poruku može biti primenjeno više labela i jedna labela može biti primenjena na više poruka. Postoje sistemske labele kao što su: *INBOX*, *TRASH*, *DRAFT* ali takođe mogu biti kreirane i korisniče labele,
* *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 biva izbrisan i message resursu se dodaje *SENT* sistemska labela,
* *history –* resurs koji predstavlja kolekciju izmenjenih poruka u hronološkom redosledu,
* *thread –* resurs koji predstavlja kolekciju poruka koje predstavljaju konverzaciju,
* *settings -* resurs koji pruža mogućnost kontrole nad Google nalogom.

Od posebne važnosti za razvoj aplikacije su dva resursa: *message* i *label*. Na slici 11 prikazana je JSON [8] reprezentacija *message* resursa, a na slici broj 12 prikazana je reprezentacija *label* resursa.

Slika - JSON reprezentacija *message* 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 - JSON reprezentacija *label* resursa

{

"id": string,

"name": string,

"messageListVisibility": string,

"labelListVisibility": string,

"type": string,

"messagesTotal": integer,

"messagesUnread": integer,

"threadsTotal": integer,

"threadsUnread": integer,

"color": {

"textColor": string,

"backgroundColor": string

}

}

#### Autentikacija i Google Gmail API

*Google Gmail API* koristi Oauth 2.0 [9] za autentikaciju i autorizaciju. Aplikacija koja koristi API mora da specifikuje jedan ili više *scope* stringova. *Scope* služi za identifikovanje resursa koje je moguće koristiti u aplikaciji. Neki od *scope* stringova korišćenih u klijentskom delu aplikacije su:

* <https://www.googleapis.com/auth/gmail.readonly> - omogućava čitanje poruka i labela,
* <https://www.googleapis.com/auth/gmail.send> - omogućava slanje poruka,
* <https://www.googleapis.com/auth/gmail.modify> - omogućava izmenu labela pimenjenih na poruke.

#### Tipičan proces rada as Google Gmail API servisom

Tipičan rad sa API servisom izgleda ovako:

* Autentikacija,
* API pozivi putem HTTP metoda,
* Rukovanje resursima dobijenim u odgovorima API poziva.

Za rad sa Google API servisom nije neophodno imati posebnu biblioteku. Moguće je pozivati API putem HTTP metoda na bilo koji način. Međutim, postoji implementacija (*library*) koja uobličava i maskira API pozive čime olakšava rad sa njima. U razvoju aplikacije korišćena je Javascript implementacija.

# Specifikacija aplikacije

Zadatak obuhvata izradu veb aplikacije za upravljanje elektronskim porukama, rađenu po ugledu na već postojeću Google Gmail aplikaciju (<https://mail.google.com>). Aplikacija omogućava pregled imejlova po labelama, slanje novih imejlova (sa dodacima ili bez njih) i njihovo brisanje.

Prilikom dolaska na veb stanicu korisniku se nudi mogućnost prijave na sistem. Prijava na sistem se vrši putem Google naloga i iz tog razloga nije pružena mogućnost kreiranja novog naloga.

Nakon uspešne prijave na sistem, korisniku se prikazuje glavna stranica na kojoj može da pregleda imejlove. Mejlovi koji su prikazani nakon prijave su poslednji pristigli mejlovi i nisu sortirani po labelama. Pored toga, korisniku je omogućeno da pregleda imejlove i po labelama (*Sent, Inbox, Draft, Important*). Svaki pregled labela vodi korisnika na zasebnu stranicu.

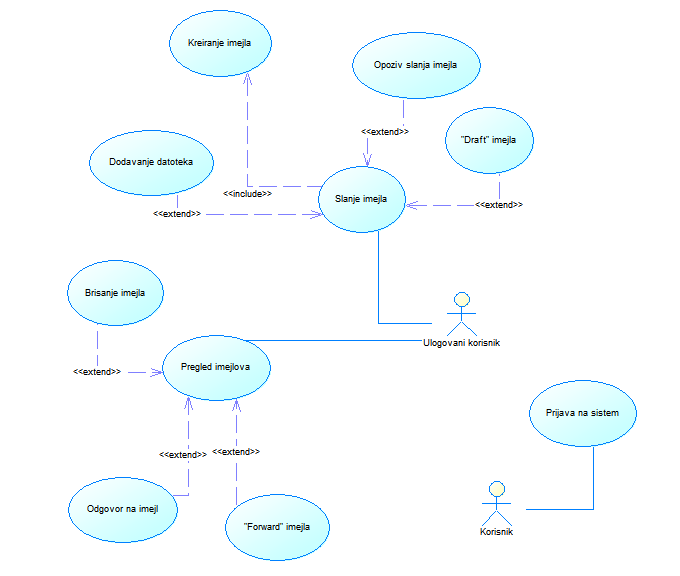
U svim stranicama za pregled, mejlovi su izlistani u tabeli i prikazani su osnovni podaci za svaki od njih (naslov, opis, datum). Detaljni pregled mejlova se vrši klikom na bilo koji od njih, ta akcija korisnika vodi na novu stranicu za detaljni pregled odakle može da odgovori, prosledi ili obriše imejl.

Aplikacija pruža mogućnost kreiranja novog mejla i njegovo slanje. Prilikom kreiranja imejla korisnik unosi naslov, primaoca i samo telo poruke. Takođe, može da doda i dodatak. Kreiranje mejla može da se opozove pre njegovog slanja.

U narednom poglavlju biće prikazan dijagram slučajeva korišćenja aplikacije.

## Dijagram slučajeva korišćenja

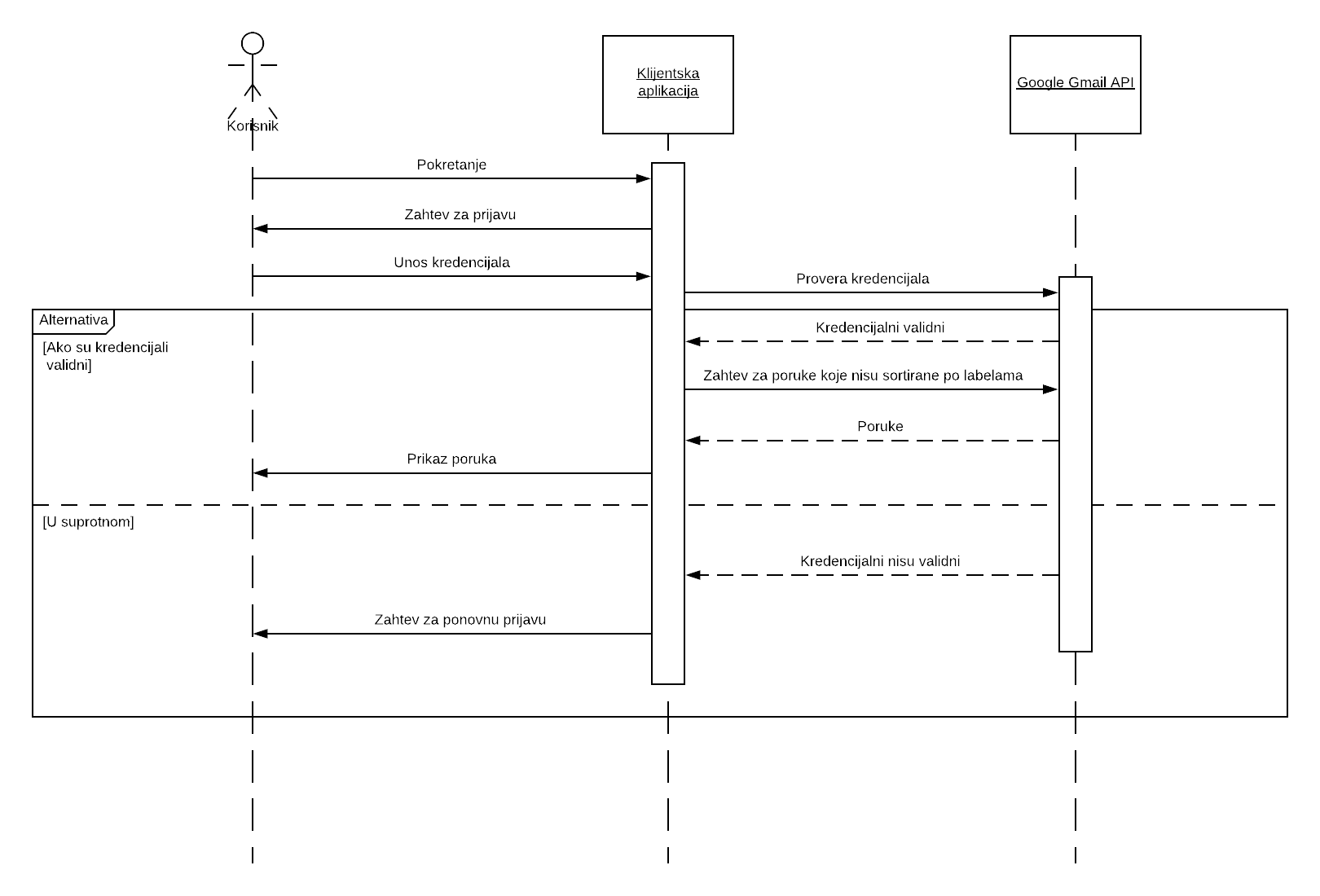
Dijagram slučajeva korišćenja prikazan je na slici broj 11. Kao što se može zaključiti iz dijagrama aplikaciju je moguće koristiti kao neprijavljeni ili kao prijavljeni korisnik. Neprijavljeni korisnik ima samo mogućnost prijave na sistem putem svog *Google* naloga. Nakon uspešne prijave na sistem, neprijavljeni korisnik postaje prijavljeni. Kao takav, korisnik ima mogućnost da pregleda imejlove, da odgovori na njih, prosledi ih nekom drugom ili obriše. Takođe, korisnik ima mogućnost da sastavi novi imejl (sa dodatkom ili bez njega). U svakom trenutku korisnik ima mogućnost da odustane od kreiranja novog mejla ili da isti sačuva kao *Draft*.



Slika 13 - Dijagram slučajeva korišćenja aplikacije

## Dijagrami sekvence

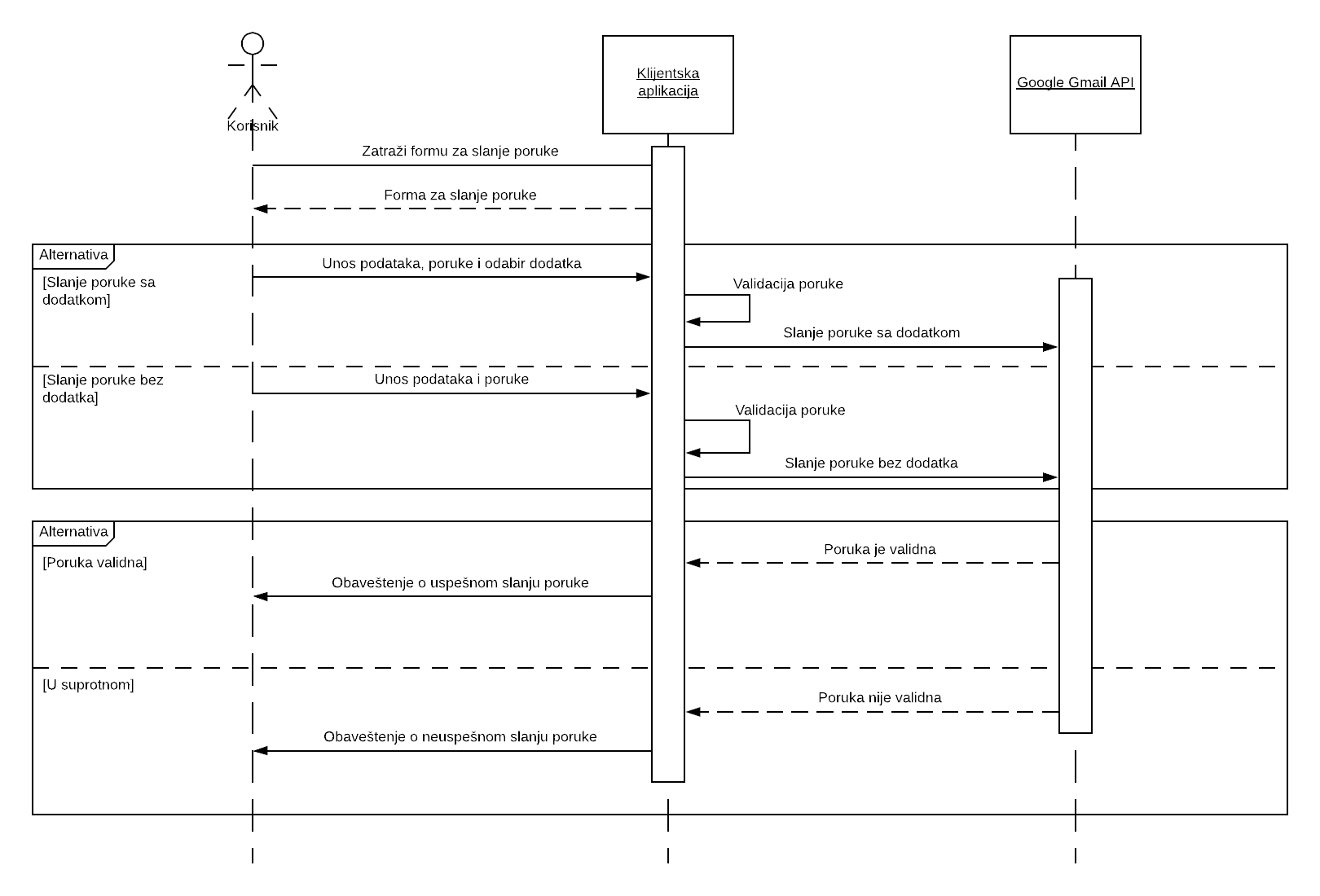
Slede dijagrami sekvence za slucajeve prijave na sistem, slanja novog imejla i odgovora na imejl. Na dijagramima se pod porukom smatra imejl.

Dijagram sekvence za slučaj prijave na sistem prikazan je na slici broj 14. Prvo što je potrebno da korisnik uradi prilikom pokretanja klijentske aplikacije je prijava na sistem (ukoliko već nije prethodno bio prijavljen). Na dijagramu je prikazan niz događaja koji dovode do uspešne ili neuspešne prijave. Korisnik se na sistem prijavljuje upotrebom *Gmail* naloga, prilikom čeka mora da unese lozinku. Proveru kredencijala vrši Google Gmail API servis a klijentska aplikacija reaguje u skladu sa time da li su ti kredencijali validni ili nisu.

Slika - Dijagram sekvence za slučaj prijave na sistem

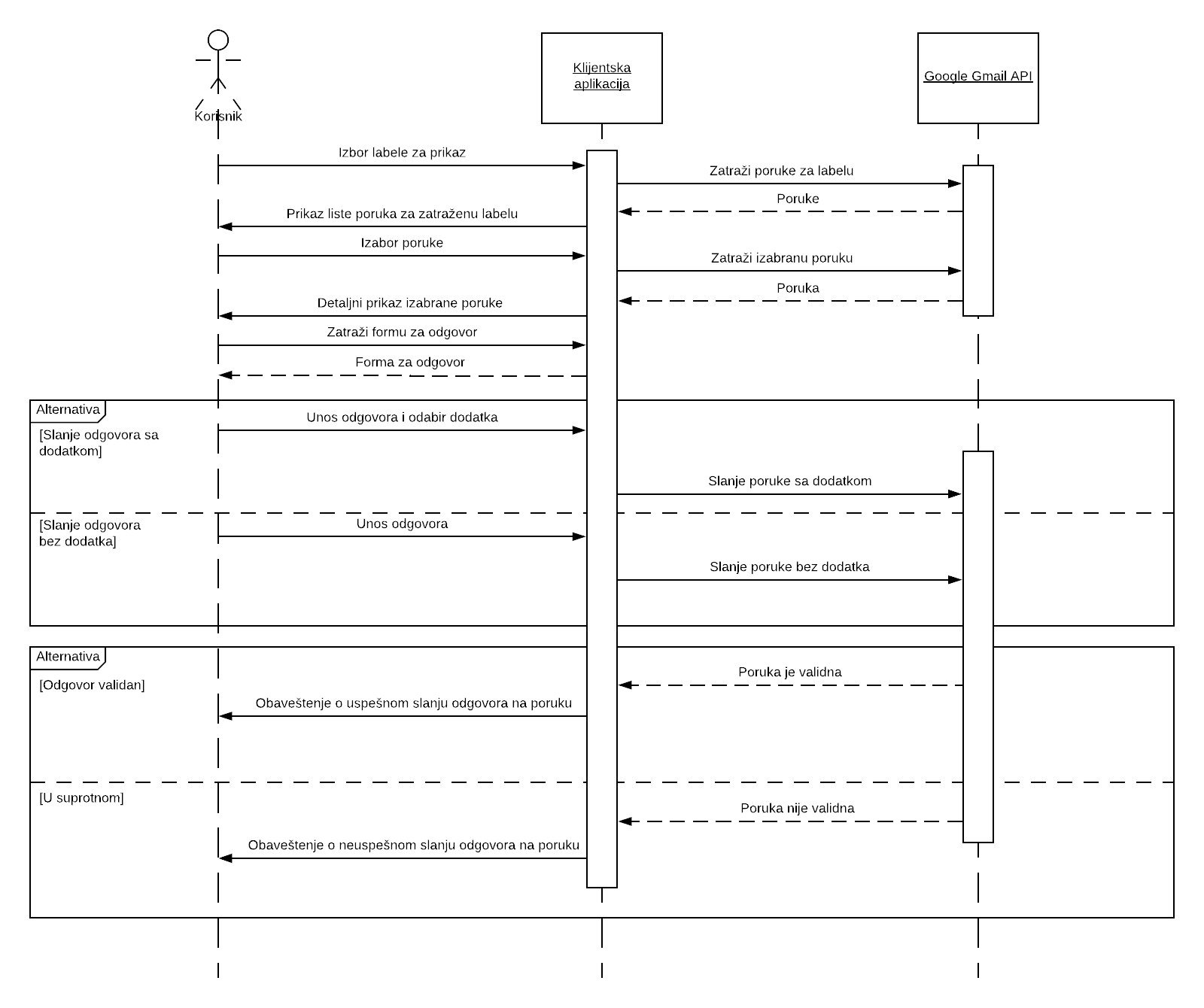
Za naredna dva dijagrama se zbog preglednosti podrazumeva da je korisnik već prijavljen na sistem.

Dijagram na slici broj 15 se odnosi na slučaj slanja novog imejla. Kao što se sa dijagrama može videti, postoje dva alternativne grane dešavanja. Nakon zatraživanja forme za slanje imejla, korisnik se može odlučiti da pošalje jednostavan imejl bez dodataka (*attachment*) ili da u imejl uključi i dodatke. U zavisnosti od toga se dešava sledeće: klijentska aplikacija šalje zahtev za slanje jednostavne poruke *Google Gmail API* servisu ili zahtev za slanje poruke sa dodacima. Validacija koju nad porukom vrši klijentska aplikacija je validacija u smislu ispravno unesene imejl adrese primaoca, da li je upisan naslov poruke i slično, a validacije koje vrsi API servis su validacije ispravnosti tela HTTP zahteva.



Slika - Dijagram sekvence za slučaj slanja novog imejla

Na slici broj 16 je dijagram sekvence za slučaj slanja odgovora na dobijeni imejl. Nakon što korisnik izabere tj. otvori odredjenu poruku postoje, slično kao i kod prethodnog slučaja dva alternativna toka dešavanja. Korisnik može slati jednostavan odgovor bez dodataka ili uključiti dodatke u odgovor, a poruka može biti poslana ili ne u zavisnosti od validnosti HTTP zahteva.



Slika 16 - Dijagram sekvence za slučaj odgovora na poruku

## Dijagram Vue komponenti

Dijagram Vue komponenti prikazan je na slici broj 17. Ovaj dijagram služi kao prikaz hijerarhijske strukture komponenti koje čine klijentski deo aplikacije. Napomena: dijagram Vue komponenti ne treba pomešati sa dijagramom komponenti u komponentalnom razvoju softvera.

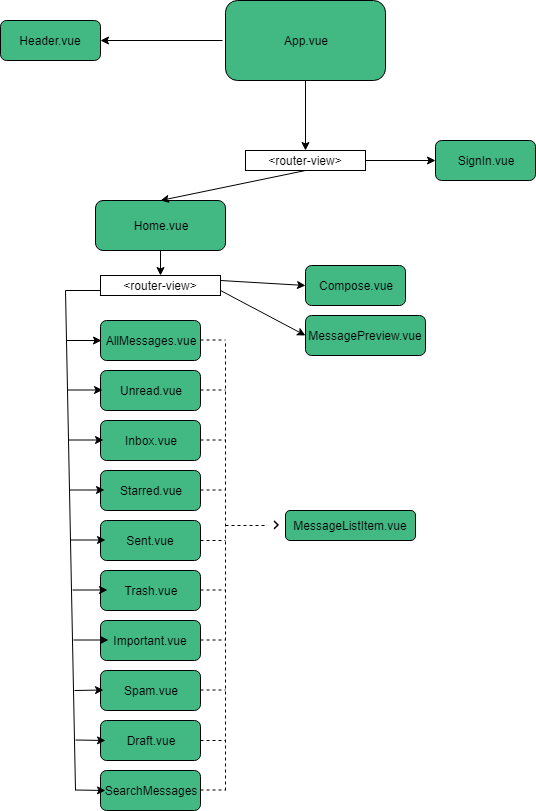
Kao sto se sa dijagrama može zaključiti, korenska komponenta je *App.vue.* Ona služi kao ulazna tačka aplikacije (*entry point*). Korenska komponenta sadrži *Header.vue*, komponentu koja služi za prikaz zaglavlja aplikacije sa opcijama. *Header*.*vue* je komponenta koja je u aplikaciji uvek vidljiva. Putem svog *router-view* kontejnera u korenskoj komponenti mogu biti prikazane *Home.vue* ili *SignIn.vue* komponente. U zavisnosti od toga da li je korisnik prijavljen na sistem prikaz *Home.vue* komponente je zabranjen ili dozvoljen. Ukoliko korisnik nije prijavljen na sistem moguć je samo prikaz *SignIn.vue* komponente koja služi upravo u svrhe prijave.

*Home.vue* komponenta sadrži meni za navigaciju kroz labele. Putem *router-view* kontejnera ove komponente moguć je prikaz sledećih komponenti:

* *AllMessages.vue* – služi za prikaz poruka koje nisu sortirane po labelama,
* *Unread*.*vue* – služi za prikaz poruka koje nisu pročitane,
* *Inbox*.*vue* – služi za prikaz pristiglih poruka,
* *Starred*.*vue* – služi za prikaz poruka koje su sačuvane tj. označene zvezdom,
* *Sent.vue –* služi za prikaz poruka koje su poslane,
* *Trash.vue* – služi za prikaz obrisanih poruka,
* *Important*.*vue* – služi za prikaz poruka koje sadrže *IMPORTANT* sistemsku labelu,
* *Spam.vue –* služi za prikaz poruka koje sadrže *SPAM* sistemsku labelu,
* *Draft.vue –* služi za prikaz nedovršenih poruka,
* *Compose.vue –* služi za slanje nove poruke,
* *MessagePreview.vue* – služi za detaljni prikaz pojedinačnih poruka

Kao što se sa drijagrama može videti, komponente koje služe za prikaz poruka služe se komponentom *MessageListItem.vue*. Svrha ove komponente je osvnovni, manje detaljan prikaz poruka kao liste.

Takođe, može se posumnjati da postoji doza repetetivnosti u komponentama koje prikazuju poruke. Međutim, objašnjenje za to je upravo to što svaka od komponenti koja služi za prikaz poruka može da ima funkcionalnosti koje su jedinstvene za tu komponentu, tako da bi pokušajem prikaza poruka prema labelama kroz jednu, generičku komponentu unelo veliku, nepotrebnu dozu složenosti klijentske aplikacije.



Slika - Dijagram Vue komponenti

# Opis implementacije

Aplikaciju čine dve celine, serverski i klijentski deo. Kao što je već napomenuto, serverski deo aplikacije nije posebno razvijan u svrhe rada. Korišćen je već postojeći Google Gmail API servis koji nudi mogućnosti upravljanja porukama putem bilo kog Google naloga. Klijentska aplikacija je implementirana uz pomoć Vue 2.0 radnog okvira, korišćenjem eksternih biblioteka (*Bootstrap* za raspored elemenata na web stranici, *Vue Gapi* za komunikaciju sa Gmail API servisom i *Alertify* za upravljanje obaveštenjima širom aplikacije.

## Klijentski deo

Klijentska aplikacija razvijana je kao SPA (*Single Page Application*), korišćenjem komponentalnog razvoja veb stranica koje nudi Vue radni okvir. Svaka od komponenti prestavlja jedan *view* koji korisniku prikazuje podatke. Naravno, ovakve aplikacije sežu mnogo dalje od pukog prikazivanja podataka korisniku, sadrže složenu aplikatvnu logiku koja omogućava interakciju korisnika sa prikazanim podacma i na kraju, sa samim Google Gmail API servisom.

### Struktura projekta (Webpack i Babel)

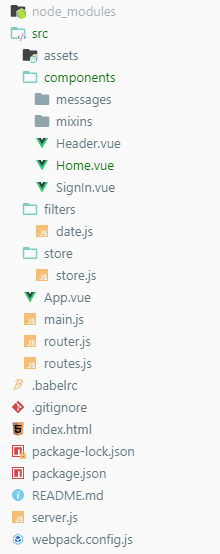
Prilikom izrade SPA, veoma su popularni određeni alati, ne samo kod razvoja aplikacija korišćenjem Vue radnog okvira, već i prilikom rada sa mnogim drugim radnim okvirima. Ti alati su:

* *Webpack [10]*
* *Babel [11]*
* *NodeJS i NPM [12]*

*Webpack* je alat koji služi za pakovanje modula aplikacije u jednu ili više datoteka (*bundle*) koje će biti isporučene internet pretraživaču u trenutku pokretanja aplikacije (zatraživanja veb stranice). Ovo je veoma moćan alat iz razloga što SPA mogu u nekim slučajevima da porastu do enormnih razmera, a *Webpack* omogućava da internet pretraživaču budu dostavljene samo neophodne stvari da bi se veb strana prikazala. *Webpack* radi tako što gradi graf zavisnosti unutar aplikacije.

*Babel* je transpajler koji omogućava da *ES6 Javascript* [13] fukcionalnosti napravi dostupnim na internet pretraživačima koji još uvek nemaju podršku za *ES6 Javascript*. Neke od tih fukcionalnosti su korišćenje *Arrow* funkcija i korišćenje ključne reci *let* koja omogućava blokovski *scope*.

Korišćenjem Vue CLI (*Comman Line Interface*) kreira se projekat koji ima strukturu koja je prikazana na slici 18.



Slika - Struktura klijentskog dela aplikacije

Kao što se sa slike može videti, u *src* direktorijumu projekta se nalaze komponente i sav *Javascript* kod koji definiše aplikativnu logiku aplikacije. Unutar korenskog direktorijuma se nalaze konfiguracioni fajlovi. Bitni konfiguracioni fajlovi su:

* *package.json* – između ostalog, sadrži informacije o tome koje zavisnosti (*dependency*) ima aplikacija, odnosno koje eksterne biblioteke koristi,
* .babelrc – podešavanja *Babel* transpajlera,
* *Webpack.config.js* – podešavanja *Webpack* alata

Za projekat je od izuzetne važnosti upravo *package.json* konfiguracioni fajl. On je prikazan na slici broj 19.

{

"name": "vue-cli",

"description": "A Vue.js project",

"author": "Arsenije Karpic <arsenije.karpic@gmail.com>",

"private": true,

"scripts": {

"dev": "cross-env NODE\_ENV=development webpack-dev-server --open --inline --hot",

"build": "cross-env NODE\_ENV=production webpack --progress --hide-modules",

"start": "node server.js"

},

"dependencies": {

"axios": "^0.18.0",

"babel-preset-stage-2": "^6.24.1",

"bootstrap": "^3.3.7",

"express": "^4.16.4",

"vue": "^2.0.1",

"vue-alertify": "^1.1.0",

"vue-gapi": "0.0.10",

"vue-router": "^3.0.2",

"vuex": "^3.0.1"

},

"devDependencies": {

"babel-core": "^6.0.0",

"babel-loader": "^6.0.0",

"babel-preset-es2015": "^6.0.0",

"cross-env": "^3.0.0",

"css-loader": "^0.25.0",

"file-loader": "^0.9.0",

"vue-loader": "^9.7.0",

"webpack": "2.1.0-beta.25",

"webpack-dev-server": "2.1.0-beta.0"

}

}

Slika - package.json konfiguracioni fajl

Kao što se sa slike može videti, važna stavka unutar package.json fajla je upravo *dependencies* objekat koji sadrži listu svih eksternih biblioteka koje se koriste u projektu zajedno sa njihovim imenima i verzijom.

### Centralizovano praćenje stanja aplikacije

Zbog jednostavnosti praćenja daljeg objašnjenja implementacije aplikacije, najpre je predstavljena implementacija centalizovanog praćenja stanja aplikacije. Za praćenje stanja aplikacije korišćenja je *Vuex* biblioteka o kojoj je detaljnije pisano u poglavlju 2.1.4. Kao što je već pomenuto, Vuex se sastoji od nekoliko delova, sledi objašnjenje konkretne implementacije tih delova u aplikaciji. Uprošćen izgled Vuex.store instance prikazan je na slici broj 20.

Slika - Uprošćen izgled Vuex.Store instance

import Vue from 'vue';

import Vuex from 'vuex';

Vue.use(Vuex);

export default new Vuex.store({

state,

mutations,

actions,

getters

})

U velikim SPA (Single Page Application) bitno je da podaci kojima se upravlja u aplikaciji budu dostupni u svim njenim komponentama. Upravo to pruža *Vuex* i tome služi *state* objekat u okviru Vuex.store instance.Podaci kojima raspolaže aplikacija (*state*) prikazani su na slici broj 21.

state: {

gapi: {},

alertify: {},

labels: [],

allMessages: [],

searchedMessages: [],

messagesByLabel: {

'SENT': [],

'INBOX': [],

'TRASH': [],

'DRAFT': [],

'SPAM': [],

'UNREAD': [],

'IMPORTANT': [],

'STARRED': [],

},

isSignedIn: false,

username: '',

emailAddress: ''

}

Slika - Konkretna implementacija *state* objekta unutar Vuex.Store instance

Ovom implementacijom podaci sačuvai u *state* objektu postaju dostupni širom aplikacije. Kao što se sa slike može videti implementacija API poziva (*gapi*), dopremljene poruke (*allMessages*, *messagesByLabel*) mogu biti dostupne bilo kojoj komponenti. Pristupanje ovim podacima može se vršiti direktno pristupanjem *state* objektu Vuex.store instance, međutim mnogo je bolje rešenje koristiti *getters.* Implementacija *getters* objekta je prikazana na slici 22 (zbog boljeg prikaza objekat je podeljen u dve slike).

getters: {

allLabels: state => {

return state.labels;

},

isSignedIn: state => {

return state.isSignedIn;

},

allMessages: state => {

return state.allMessages;

},

gapi: state => {

return state.gapi;

},

alertify: state => {

return state.alertify;

},

inboxMessages: state => {

return state.messagesByLabel['INBOX'];

},

starredMessages: state => {

return state.messagesByLabel['STARRED'];

},

sentMessages: state => {

return state.messagesByLabel['SENT'];

}

Slika - Implementacija getter objekta unutar Vue.store instance

trashMessages: state => {

return state.messagesByLabel['TRASH'];

},

draftMessages: state => {

return state.messagesByLabel['DRAFT'];

},

spamMessages: state => {

return state.messagesByLabel['SPAM'];

},

unreadMessages: state => {

return state.messagesByLabel['UNREAD'];

},

importantMessages: state => {

return state.messagesByLabel['IMPORTANT'];

},

searchedMessages: state => {

return state.searchedMessages;

},

username: state => {

return state.username;

},

emailAddress: state => {

return state.emailAddress;

}

}

Kao što se na slici može videti, za svaki objekat sačuvan unutar *state* objekta postoji njegov odgovarajući *getter. Getter* je funkcija koja kao argument ima *state* objekat a kao povratnu vrednost može da ima podatke iz *state* objekta, izmenjene ili neizmenjene. Svoju fleksibilnost upravo odražava time što dozvoljava da se nad podacima izvrši aplikativna logika, i da onda takvi, izmenjeni podaci budu prikazivani putem *getter* funkcija.

Podaci unutar *state* objekta se menjaju isključivo putem mutacija (*mutations*). Mutacije su funkcije koje menjaju stanje aplikacije i moraju biti sihnrone zbog lakog praćenja promene stanja aplikacije. Ukoliko bi se omogućilo da fukcije koje menjaju stanje aplikacije mogu da budu asinhrone, ne bi bilo moguće eksplicitno pratiti promenu stanja aplikacije zato što se ne može sa sigurnošću odrediti kada će neki asihnroni poziv biti završen i njegova *handler* funkcija pozvana. Na slici broj 23 može se videti implementacija mutacija unutar aplikacije.

Slika - Implementacija mutations objekta unutar Vuex.store instance

mutations: {

SET\_GAPI\_INSTANCE(state, payload) {

state.gapi = payload;

},

SET\_ALERTIFY\_INSTANCE(state, payload) {

state.alertify = payload;

},

SET\_LABELS(state, payload) {

state.labels = payload;

},

SET\_SIGNED\_IN(state, payload) {

state.isSignedIn = payload;

},

SET\_ALL\_MESSAGES(state, payload) {

state.allMessages = payload;

},

SET\_MESSAGES\_BY\_LABEL(state, payload) {

state.messagesByLabel[payload.label] = payload.messages;

},

SET\_SEARCHED\_MESSAGES(state, payload) {

state.searchedMessages = payload;

}

,

SET\_USERNAME(state, payload) {

state.username = payload;

},

SET\_EMAIL(state, payload) {

state.emailAddress = payload;

}

}

Kao što se sa slike može videti, mutacije su funkcije koje kao argument primaju *state* objekat i opcioni *payload* objekat putem kojeg menjaju stanje. Mutacije se mogu pozivati direktno iz bilo koje komponente, ali je najčeći slučaj sledeći. Akcije (*actions*) su funcije koje se suštinski od mutacija razlikuju po tome što mogu da sadrže asinhrone pozive. Akcije se koriste prilikom slanja zahteva na server. Omogućavaju korišćenje bilo kakve implementacije asihnronih poziva, a kada se dobije odgovor od servera, akcije pozivaju mutacije i prosledjuju im podatke dobijene sa servera. Tako da mutacije i dalje ostaju sinhrone, one su pozvane tek kada se dobije odgovor od servera, a akcije ne menjaju stanje aplikacije direktno već kroz mutacije.

Na slici broj 24 prikazana je jedna akcija (listanje svih poruka sa servera). Radi jednostavnosti prikazana je samo ova akcija. U daljem teksu, kada bude objašnjena implementacija komponenti, biće prikazane i akcije koje te komponente koriste.

Slika - Primer akcije unutar Vuex.Store instance

listAllMessages(context) {

let gapi = context.getters.gapi;

let request;

let allMessages = [];

request = gapi.client.gmail.users.messages.list({

userId: "me",

maxResults: 10

});

request.execute(function(response) {

for (let message of response.messages) {

var messageRequest = gapi.client.gmail.users.messages.get({

userId: 'me',

id: message.id

});

messageRequest.execute(function(resp) {

allMessages.push(resp);

});

}

context.commit('SET\_ALL\_MESSAGES', allMessages);

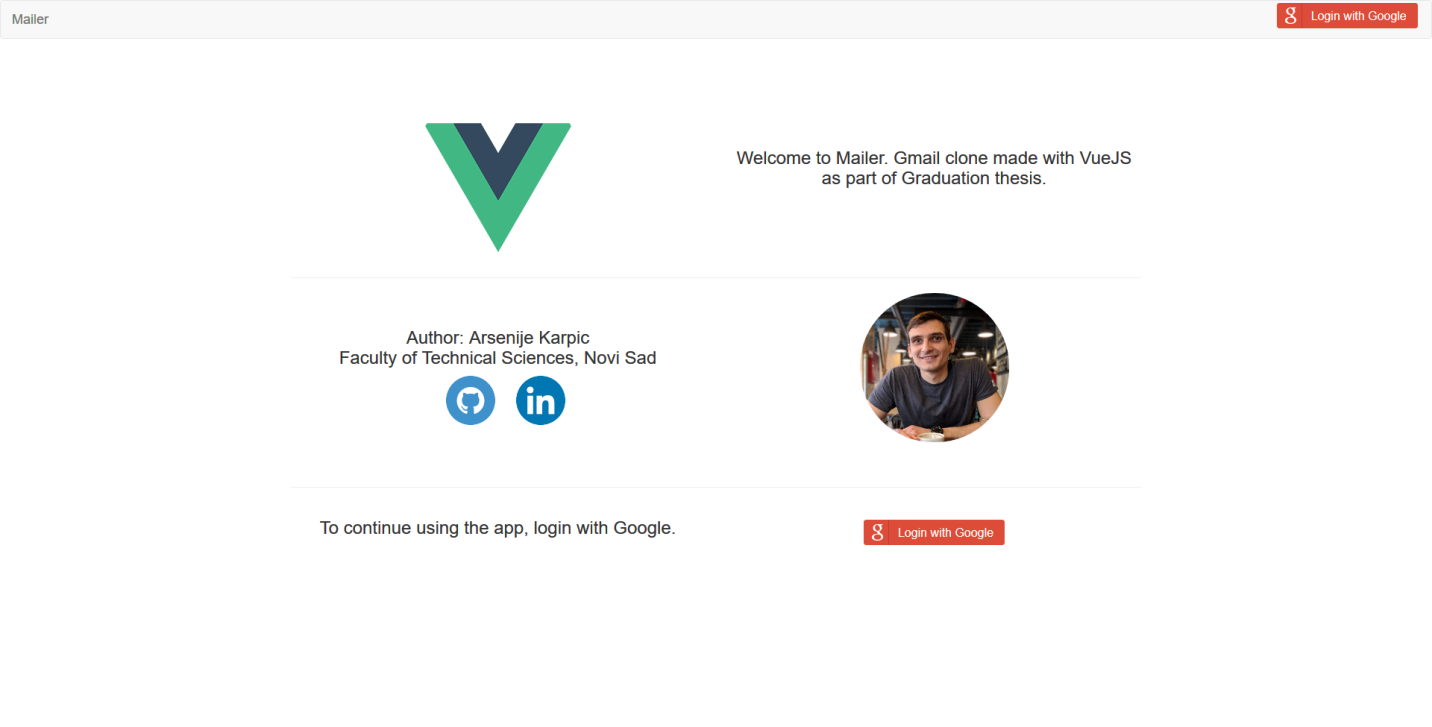
});

}

Kao što se vidi na slici, akcije su takođe funkcije. Kao argument primaju *context* objekat. Putem *context* objekta može se pristupiti stanju aplikacije (state), geterima (getters) i mogu se pozvati mutacije putem funckiije *commit*. U konkretnoj akciji koja je prikazana na slici, kreira se zahtev korišćenjem implementacije *Google Gmail API* poziva za Javascript (*gapi*). Zahtev lista 10 poslednjih poruka za ulogovanog korisnika. API na taj zahtev odgovara listom objekata koji sadrže id svake poruke. Dalje, akcija za svaku poruku kreira i šalje na server novi zahtev za dobijanje kompletne poruke korišćenjem njene id oznake. Dobijene poruke su sačuvane u allMessages objektu, a taj objekat se prosleđuje mutaciji *SET\_ALL\_MESSAGES*. Mutacija dalje koristi taj objekat kako bi promenila stanje, tj. poruke dobijene sa severa sada su dostupne bu bilo kojoj komponenti aplikacije putem odgovarajuće *getter* funkcije. Ovim kratkim opisom objašnjen je šablon koji stoji iza *Vuex* implementacije.

### Implementacija komponenti

Nakon pokretanja aplikacije (nakon što internet pretraživač zatraži stranicu), korisniku se prikazuje stranica prikazana na slici 25.



Slika 25 - Početna stranica aplikacije

Kao što se na slici može videti, ako korisnik prethodno nije prijavljen na sistem, aplikacija mu prikazuje *signin* stranicu putem koje može da se prijavi korišćenjem Google naloga. Za ovu stranicu zadužena je SignIn.vue komponenta. Komponenta je prikazana na slici broj 26 (radi preglednosti komponenta je prikazana iz više delova).

<template>

<div>

<br><br><br><br>

<div class="text-center">

<div class="row">

<div class="col-md-6">

<img src="../assets/logo.png" style="border-radius: 2em;">

</div>

<div class="col-md-6">

<h3><br>Welcome to Mailer. Gmail clone made with VueJS <br> as part of Graduation thesis.</h3>

</div>

</div>

<hr>

<div class="row">

<div class="col-md-6">

<h3><br>Author: Arsenije Karpic <br> Faculty of Technical Sciences, Novi Sad</h3>

<ul class="social-icons">

<li><a href="https://github.com/karpic"><img src="../assets/Github.png" alt="Github logo"></a></li>

<li><a href="https://www.linkedin.com/in/arsenije-karpi%C4%87-a0905516a/"><img src="../assets/Linkedin.png" alt="Linkedin logo"></a></li>

</ul>

</div>

<div class="col-md-6">

<img src="../assets/author.jpg" alt="Author" style="width: 200px; height: 200px; border-radius: 7em;">

</div>

</div>

<br><br>

<hr>

<div class="row">

<div class="col-md-6">

<h3>To continue using the app, login with Google.</h3>

</div>

<div class="col-md-6">

<br>

<button class="loginBtn loginBtn--google" @click="login">

Login with Google

</button>

</div>

</div>

</div>

</div>

</template>

Slika - SignIn komponenta

<script>

import { mapActions } from 'vuex';

export default {

methods: {

...mapActions([

'signIn'

]),

login() {

this.signIn(this);

}

}

};

</script>

Zbog jednostavnosti, stilovi unutar komponenti nisu prikazani. Kao što se može videti u *script* delu komponente, koristi se *mapActions* funkcija. Ta funckija omogućuje pozivanje Vuex akcija unutar komponenti. Konkretno u ovoj komponenti poziva se *signIn* akcija klikom na dugme *Login with Google* unutar template dela. Na slici 27 prikazana je *signIn* akcija.

signIn(context, vueInstance) {

if (vueInstance.$isAuthenticated !== true) {

vueInstance.$getGapiClient().then(gapi => {

gapi.auth2

.getAuthInstance()

.signIn()

.then(googleUser => {

context.commit('SET\_USERNAME', googleUser.w3.ig);

context.commit('SET\_EMAIL', googleUser.w3.U3);

localStorage.setItem('token', googleUser.Zi.access\_token);

vueInstance.$alertify.success(

`Successfully logged in as ${googleUser.w3.ig}`

);

localStorage.setItem('user', JSON.stringify(googleUser));

context.commit('SET\_SIGNED\_IN', true);

context.commit('SET\_GAPI\_INSTANCE', gapi);

context.commit('SET\_ALERTIFY\_INSTANCE', vueInstance.$alertify);

Slika - SignIn akcija

gapi.client.gmail.users.labels

.list({

userId: "me"

})

.then(function(response) {

var labels = [];

for(let label of response.result.labels) {

gapi.client.gmail.users.labels.get({

userId: 'me',

id: label.id

}).then(function(labelResponse) {

labels.push(labelResponse.result);

})

}

context.commit("SET\_LABELS", labels);

});

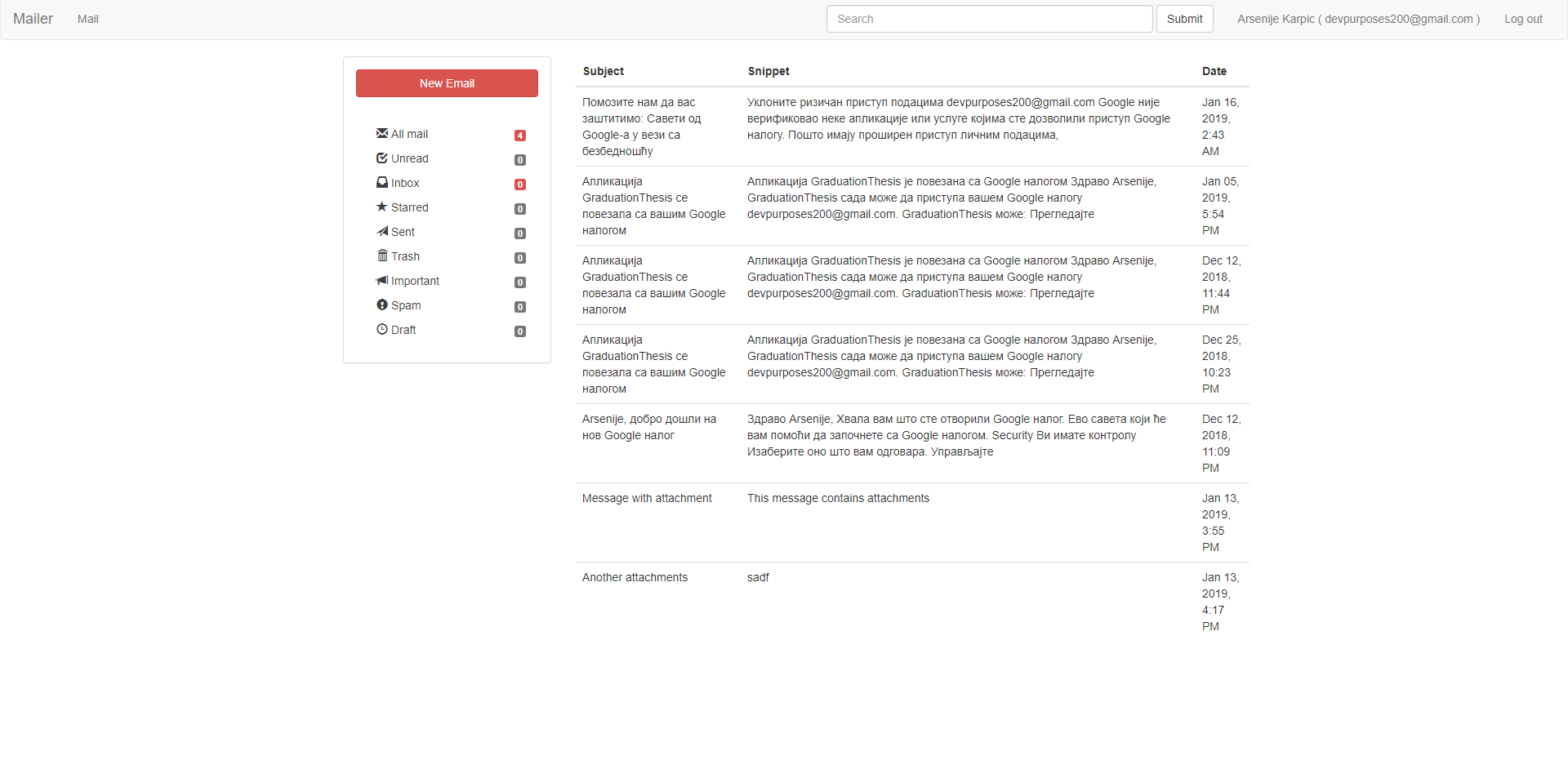
vueInstance.$router.push({ name: 'allMessages' });

});

});

}

Ova akcija podešava inicijalno stanje aplikacije prilikom korisnikove prijave na sistem. Čuva korisničke podatke kao što su imejl adresa, korisničko ime, token za autorizaciju na serveru i labele koje su vezane za korisnikov Google nalog. Nakon promene inicijalnog stanja aplikacije, vrši se rutiranje na *AllMessages.vue* komponentu i korisniku se prikazuje stranica prikazana na slici 28.



Slika 28 - Stranica koja je dostupna nakon prijave na sistem

Nakon uspešne prijave na sistem, kao što se vidi na slici, korisnik može putem navigacionog menija da pretražuje poruke po labelama, da ih čita, da pretražuje poruke koristeći polje za pretragu pri vrhu stranice ili da piše nove poruke. Komponenta koja je zadužena za ovu stranicu je *Home.vue* prikazana je na slici broj 29.

<template>

<div class="container">

<div class="inbox">

<div class="col-lg-3">

<div class="panel panel-default">

<div class="panel-body inbox-menu">

<router-link :to="{ name: 'compose'}"><a class="btn btn-danger btn-block">New Email</a></router-link>

<ul>

       <router-link tag="li" :to="{ name: 'allMessages' }">

<a>

<i class="fa fa-inbox"></i><span class="glyphicon glyphicon-envelope"></span> All mail

<span class="label label-danger">4</span>

</a>

</router-link>

<router-link tag="li" :to="{ name: 'unreadMessages' }">

<a>

<i class="fa fa-inbox"></i><span class="glyphicon glyphicon-check"></span> Unread

<span class="label label-default">{{ getUnreadCountForLabel(allLabels, 'UNREAD')}}</span>

</a>

</router-link>

<router-link tag="li" :to="{ name: 'inboxMessages' }">

<a>

<i class="fa fa-inbox"></i><span class="glyphicon glyphicon-inbox"></span> Inbox

<span class="label label-danger">{{ getUnreadCountForLabel(allLabels, 'INBOX')}}</span>

</a>

</router-link>

<router-link tag="li" :to="{ name: 'starredMessages' }">

<a>

<i class="fa fa-star"></i><span class="glyphicon glyphicon-star"></span> Starred

<span class="label label-default">{{ getUnreadCountForLabel(allLabels, 'STARRED')}}</span>

</a>

</router-link>

<router-link tag="li" :to="{ name: 'sentMessages' }">

<a>

<i class="fa fa-rocket"></i><span class="glyphicon glyphicon-send"></span> Sent

<span class="label label-default">{{ getUnreadCountForLabel(allLabels, 'SENT')}}</span>

</a>

</router-link>

<router-link tag="li" :to="{ name: 'trashMessages'}">

<a>

<i class="fa fa-trash-o"></i><span class="glyphicon glyphicon-trash"></span> Trash

<span class="label label-default">{{ getUnreadCountForLabel(allLabels, 'TRASH')}}</span>

</a>

</router-link>

<router-link tag="li" :to="{ name: 'importantMessages'}">

<a href="#">

<i class="fa fa-bookmark"></i><span class="glyphicon glyphicon-bullhorn"></span> Important

<span class="label label-default">{{ getUnreadCountForLabel(allLabels, 'IMPORTANT')}}</span>

</a>

</router-link>

<router-link tag="li" :to="{ name: 'spamMessages' }">

<a>

<i class="fa fa-inbox"></i><span class="glyphicon glyphicon-exclamation-sign"></span> Spam

<span class="label label-default">{{ getUnreadCountForLabel(allLabels, 'SPAM')}}</span>

</a>

</router-link>

<router-link tag="li" :to="{ name: 'draftMessages' }">

<a>

<i class="fa fa-inbox"></i><span class="glyphicon glyphicon-time"></span> Draft

<span class="label label-default">{{ getUnreadCountForLabel(allLabels, 'DRAFT')}}</span>

</a>

</router-link>

</ul>

</div>

</div>

</div>

   <div class="col-lg-9">

     <router-view></router-view>

   </div>

</div>

</div>

</template>

<script>

import { mapActions, mapGetters } from "vuex";

import mixin from '../components/mixins/mixin.js';

export default {

mixins: [mixin],

methods: {

...mapActions(["listAllMessages"]),

list() {

this.listAllMessages(this);

}

},

computed: {

...mapGetters([

'allLabels'

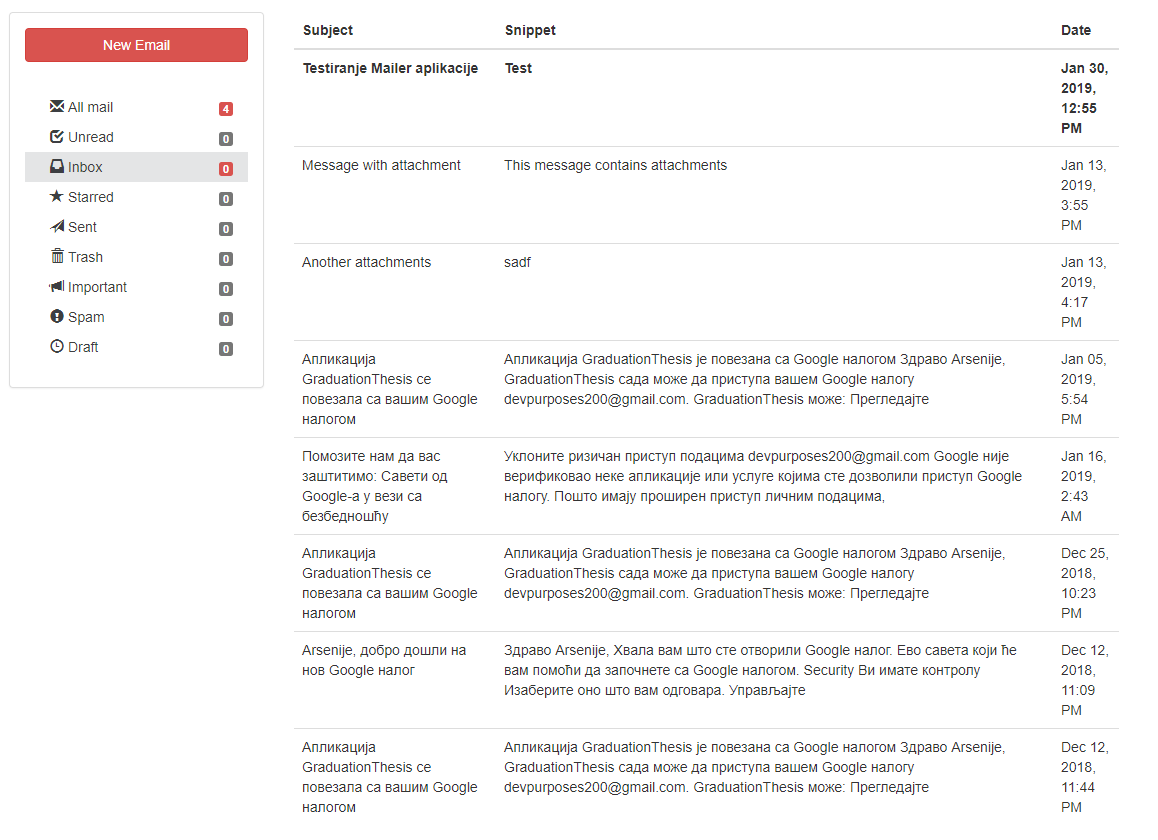
])

}

};

</script>

Slika - Home.vue komponenta

Kao što se sa slike može videti, *Home.vue* komponenta sadrži navigacioni meni putem kog korisnik može da lista imejl poruke prema labelama. U okviru komponente se pored menija nalazi I *router-view* element koji služi da prikaže komponente. U zavisnosti od toga koju labelu je korisnik izabrao, putem mehanizma rutiranja se u router-view elementu prikazuje određena komponenta. Ako korisnik izabere da lista poruke u Inbox labeli, prikazuje se *Inbox.vue* komponenta i listaju se sve poruke sa sitemskom labelom *INBOX*, kao što je prikazano na slici boj 30.

Slika - Izgled Inbox.vue komponente

Implementacija Inbox.vue komponente prikazana je na sledećoj slici (slika 31).

Slika - Implementacija Inbox.vue komponente

<template>

<div>

<table class="table table-inbox table-hover">

<thead>

<tr>

<th scope="col">Subject</th>

<th scope="col">Snippet</th>

<th scope="col">Date</th>

</tr>

</thead>

<tbody>

<app-message-list-item v-for="message in sortedInboxMessages" :message="message" :key="message.id"></app-message-list-item>

</tbody>

</table>

</div>

</template>

<script>

import { mapActions, mapGetters } from 'vuex';

import MessageListItem from './MessageListItem.vue';

import mixin from '../mixins/mixin.js';

export default {

components: {

appMessageListItem: MessageListItem

},

mixins: [mixin],

computed: {

...mapGetters([

'inboxMessages'

]),

sortedInboxMessages() {

return this.inboxMessages.sort(

(a, b) => {

new Date(Number(a.internalDate)) - new Date(Number(b.internalDate))

}

)

}

},

methods: {

...mapActions([

'listMessagesByLabel'

])

},

created(){

this.listMessagesByLabel('INBOX');

}

}

</script>

<style scoped>

</style>

Ova komponenta u svom *template* delu prikazuje imejl poruke tabelarno (kao što je moguće videti na slici 30). Odmah nakon što se komponenta kreira (*created lifecycle hook*) komponenta poziva akciju *listMessagesByLabel* koja je definisana unutar *Vuex.Store* instance. Ova akcija lista poruke sa servera sa zadatom labelom. Implementacija akcije može se videti na slici broj 32.

Slika - Akcija koja lista poruke po labeli

listMessagesByLabel(context, label) {

let gapi = context.getters.gapi;

let request;

let messages = [];

let labels = [];

labels.push(label);

request = gapi.client.gmail.users.messages.list({

userId: 'me',

labelIds: labels,

maxResults: 10

});

request.execute(function(response) {

if(response.messages){

for(let message of response.messages) {

var messageRequest = gapi.client.gmail.users.messages.get({

userId: 'me',

id: message.id

});

messageRequest.execute(function(resp){

messages.push(resp);

});

}

}

context.commit('SET\_MESSAGES\_BY\_LABEL', { label: label, messages: messages });

});

}

Akcija kreira zahtev za Google API servis. Ona lista 10 poslednjih poruka koje imaju prosleđenu labelu (u ovom slučaju *INBOX* labelu), zatim za svaku poruku kreira i šalje pojedinačne zahteve za detaljne poruke. Nakon završetka svih API poziva, akcija poziva *SET\_MESSAGES\_BY\_LABEL* mutaciju koja menja stanje aplikacije tako što poruke sa INBOX labelom čuva u *state* delu. U ovom slučaju bilo koja komponenta može pristupiti pridobijenim porukama, što Inbox.vue komponenta i radi pozivanjem *inboxMessages* getera.

Unutar svoje *computed* opcije u *script* delu komponente, komponenta mapira geter iz Vuex.Store instance i sortira poruke dobijene putem te geter funckije. Korišćenjem ovog mehanizma omogućuje se da se u *template* delu komponente pritupi tim porukama dinamičkim vezivanjem.

Na sledećoj slici prikazan je način pozivanja geter funkcije za poruke (slika broj 33) a na slici broj 34 prikazan je način dinamičkog pristupanja porukama unutar *template* dela komponente.

Slika - Pristupanje geter funkcijama unutar *Inbox*.*vue* komponente

computed: {

...mapGetters([

'inboxMessages'

]),

sortedInboxMessages() {

return this.inboxMessages.sort(

(a, b) => {

new Date(Number(a.internalDate)) - new Date(Number(b.internalDate))

}

)

}

}

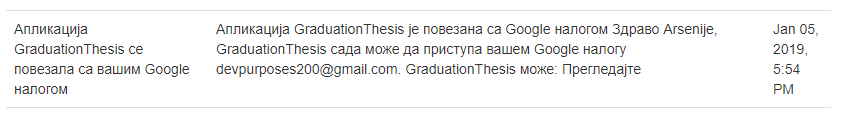
Slika - Dinamičko pristupanje porukama unutar *template* dela *Inbox*.*vue* komponente

<tbody>

<app-message-list-item v-for="message in sortedInboxMessages" :message="message" :key="message.id"></app-message-list-item>

</tbody>

Kao što se sa slike vidi, *for* petljom je moguće proći kroz sve poruke i za svaku posebno napraviti novu komponentu. Komponenta koja predstavlja jednu stavku u tabeli (jednu poruku) je *MessageListItem*.*vue* komponenta. Prilikom kreiranja ove komponente mora joj se proslediti *message* objekat, odnosno objekat koji predstavlja poruku i koji ta komponenta koristi da bi prikazala osnovne informacije o poruci (naslov poruke, datum pristizanja i kratak odsečak poruke). Komponenta koja predstavlja jednu poruku u tabeli prikazana je na slici broj 34.



Slika 35 - Izgled *MessageListItem.vue* komponente

Implementacija *MessageListItem.vue* komponente je prikazana na slici 36.

Slika - Implementacija *MessageListItem*.*vue* komponente

<template>

<tr @click="openMessage(message.id)" :class="{ boldedClass: isUnread }">

<td>{{ getHeader(message.payload.headers, 'Subject') }}</td>

<td>{{ message.snippet }}</td>

<td>{{ message.internalDate | date}}</td>

</tr>

</template>

<script>

import mixin from '../mixins/mixin.js';

export default {

data() {

return {

isUnread: false

}

},

mixins: [mixin],

props: ['message'],

methods: {

openMessage(messageId) {

this.$router.push({ name: 'messagePreview', params: { id: messageId }});

}

},

created(){

for(let i = 0; i < this.message.labelIds.length; i++) {

if(this.message.labelIds[i] === 'UNREAD') {

this.isUnread = true;

}

}

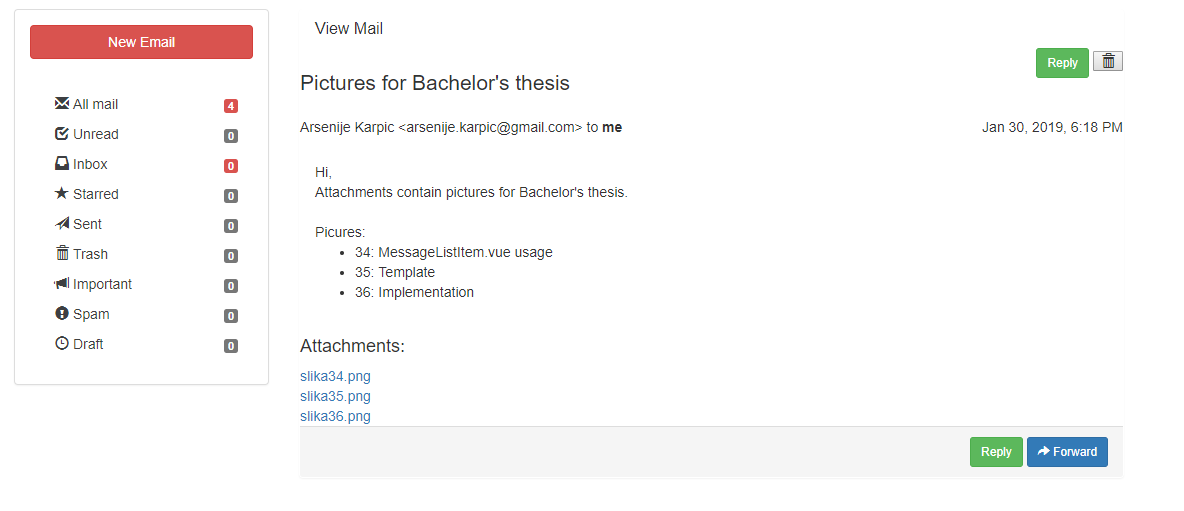
}

}

</script>

Ova komponenta treba da primi *message* objekat i da ga iskoristi kako bi dinamički prikazala osnovne podatke o poruci tabelarno. Kao što se sa slike vidi, u *props* opciji komponenta ima definisan *message* objekat. To znači da je moguće ovoj komponenti proslediti objekat koji sadrži poruku. Takođe ova komponenta nakon svog kreiranja (*created lifecycle hook*) proverava da li poruka sadrži *UNREAD* labelu. Ukoliko je sadrži sav tekst komponente biće podebljan kako bi se naznačilo da poruka nije pročitana. U svom *template* delu komponenta dinamički pristupa *message* objektu i prikazuje naslov (u ove svrhe koristi *getHeader* metodu, više o funkcionalnostima koje dele sve komponente će biti u nastavku teksta), odsečak poruke i datum. Takođe definiše i *hanlder* funkciju za klik, odnosno definiše šta se dešava kada korisnik klikne na ovu komponentu. Prilikom klika na komponentu poziva se *openMessage* metoda definisana u *methods* opciji *script* dela komponente. Ova metoda služi da rutira na koponentu za detaljan prikaz poruka (*MessagePreview.vue*).

Detaljni prikaz poruke može se videti na slici 37.



Slika 37 - Izgled *MessagePreview.vue* komponente

Kao što se sa slike može videti, otvara se detaljni prikaz poruke gde korisnik može da pregleda sadržaj poruke, preuzme dodatke (*attachements*), odgovori na poruku ili je prosledi.

<template>

<div>

<form class="form-horizontal">

<div class="panel mail-wrapper rounded shadow">

<div class="panel-heading">

<div class="pull-left">

<h3 class="panel-title">View Mail</h3>

</div>

<div class="clearfix"></div>

</div><!-- /.panel-heading -->

<div class="panel-sub-heading inner-all">

<div class="pull-left">

<h3 class="lead no-margin">{{ subject }}</h3>

</div>

<div class="pull-right">

<a href="#mail-compose.html" class="btn btn-success btn-sm"><i class="fa fa-reply"></i> Reply</a>

<button class="glyphicon glyphicon-trash" @click.prevent="deleteEmail"></button>

</div>

<div class="clearfix"></div>

</div><!-- /.panel-sub-heading -->

<div class="panel-sub-heading inner-all">

<div class="row">

<div class="col-md-8 col-sm-8 col-xs-7">

<span>{{from}}</span>

to

<strong>me</strong>

</div>

<div class="col-md-4 col-sm-4 col-xs-5">

<p class="pull-right"> {{ message.internalDate | date}}</p>

</div>

</div>

</div><!-- /.panel-sub-heading -->

<div class="panel-body">

<div class="view-mail">

<div v-html="decodedMessage"></div>

</div><!-- /.view-mail -->

</div><!-- /.panel-body -->

<div id="attachmentDownloadLinks">

<h4>Attachments: </h4>

</div>

<div class="panel-footer">

<div class="pull-right">

<a class="btn btn-success btn-sm" @click="toggleReplyClicked"><i class="fa fa-reply"></i> Reply</a>

<a class="btn btn-primary btn-sm" @click="toggleForwardClicked"><i class="glyphicon glyphicon-share-alt"></i> Forward</a>

</div>

<div class="clearfix"></div>

</div><!-- /.panel-footer -->

</div><!-- /.panel -->

</form>

<div v-if="replyClicked">

<textarea name="replyContent" id="replyContent" cols="120" rows="10" placeholder="Type your reply here" v-model="replyMessage" class="form-control"></textarea>

<button class="btn btn-success" @click="sendReply">Send</button>

</div>

<div v-if="forwardClicked">

<div class="form-group">

                     <label for="to" class="col-sm-1 control-label">To:</label>

                     <div class="col-sm-11">

<input type="email" class="form-control select2-offscreen" id="to" placeholder="Type email" tabindex="-1" v-model="forwardTo">

                     </div>

</div>

<br><hr>

<textarea name="forwardContent" id="forwardContent" cols="120" rows="10" v-model="forwardMessage" class="form-control"></textarea>

<button class="btn btn-success" @click="sendForward">Send</button>

</div>

</div>

</template>

<script>

import { mapActions } from 'vuex';

import mixin from '../mixins/mixin.js';

export default {

mixins: [mixin],

data() {

return {

messageId: this.$route.params.id,

message: {},

decodedMessage: '',

replyClicked: false,

forwardClicked: false,

replyMessage: '',

forwardMessage: '',

subject: '',

from: '',

forwardTo: ''

}

},

methods: {

...mapActions([

'sendMessage',

'deleteMessage'

]),

decodeMessage(messageToDecode) {

function getBody(message) {

var encodedBody = '';

if(typeof message.parts === 'undefined')

{

encodedBody = message.body.data;

}

else

{

encodedBody = getHTMLPart(message.parts);

}

encodedBody = encodedBody.replace(/-/g, '+').replace(/\_/g, '/').replace(/\s/g, '');

return decodeURIComponent(escape(window.atob(encodedBody)));

}

function getHTMLPart(arr) {

for(var x = 0; x <= arr.length; x++)

{

if(typeof arr[x].parts === 'undefined')

{

if(arr[x].mimeType === 'text/html')

{

return arr[x].body.data;

}

}

else

{

return getHTMLPart(arr[x].parts);

}

}

return '';

}

this.decodedMessage = getBody(messageToDecode.payload);

},

deleteEmail() {

this.deleteMessage(this.message.id);

},

toggleReplyClicked() {

this.forwardClicked = false;

this.replyClicked = !this.replyClicked;

},

toggleForwardClicked() {

this.replyClicked = false;

this.forwardClicked = !this.forwardClicked;

this.forwardMessage =

`\n\n----Forwarded message --- \nFrom: ${this.getHeader(this.message.payload.headers, 'From')} \nSubject: ${this.getHeader(this.message.payload.headers, 'Subject')}\nTo: ${this.getHeader(this.message.payload.headers, 'To')} \n------------------------- \n ${this.decodedMessage}`

},

sendReply() {

let headers = {

'To': this.getHeader(this.message.payload.headers, 'Return-Path'),

'In-Reply-To': this.message.id

};

this.sendMessage({

headers,

message: this.replyMessage

});

},

sendForward() {

let headers = {

'To': this.forwardTo,

'Subject': this.getHeader(this.message.payload.headers, 'Subject')

};

this.sendMessage({

headers,

message: this.forwardMessage

});

}

},

Slika - Implementacija MessagePreview.vue komponente

created() {

let that = this;

let messageId = this.$route.params.id;

let message = {};

this.$getGapiClient().then(gapi=>{

var messageRequest = gapi.client.gmail.users.messages.get({

userId: 'me',

id: messageId

});

messageRequest.execute(function(resp){

message = resp;

that.decodeMessage(message);

that.message = message;

that.subject = that.getHeader(message.payload.headers, 'Subject');

that.from = that.getHeader(message.payload.headers, 'From');

let modifyRequest = gapi.client.gmail.users.messages.modify({

userId: 'me',

id: message.id,

removeLabelIds: ['UNREAD']

});

modifyRequest.execute(function(respon) {

});

//handle Attachments

let parts = message.payload.parts;

for (var i = 0; i < parts.length; i++) {

let part = parts[i];

if (part.filename && part.filename.length > 0) {

var attachId = part.body.attachmentId;

var request = gapi.client.gmail.users.messages.attachments.get({

'id': attachId,

'messageId': message.id,

'userId': 'me'

});

request.execute(function(attachment) {

let dataBase64Rep = attachment.data.replace(/-/g, '+').replace(/\_/g, '/');

let urlBlob = that.b64toBlob(dataBase64Rep, part.mimeType, attachment.size);

let linkContainer = document.getElementById('attachmentDownloadLinks');

let dlTag = document.createElement('a');

console.log(part.filename);

let linkText = document.createTextNode(part.filename);

dlTag.appendChild(linkText);

dlTag.href = urlBlob;

dlTag.download = part.filename;

linkContainer.appendChild(dlTag);

linkContainer.appendChild(document.createElement('br'));

});

}

}

});

});

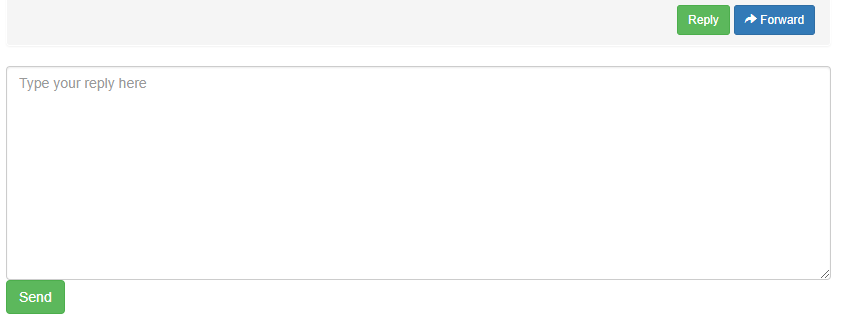
}

}

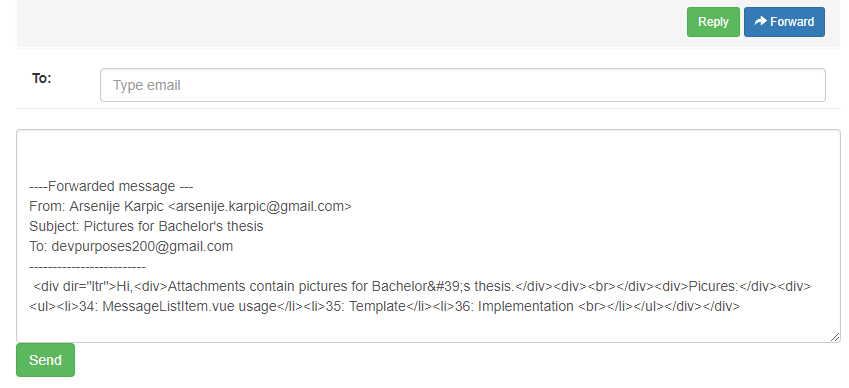
</script>

Kao što se na prethodnim slikama može videti, implementacija Vue komponenti može biti složena. Nakon rutiranja, kreira se *MessagePreview.vue* komponenta. Nakon kreiranja (*created lifecycle hook*) komponente, iz rute se pročita id imejl poruke koja treba da bude prikazana. Kreira se zahtev koji se salje na server i kao odgovor sa servera se dobijaju svi potrebni podaci o konkretnoj imejl poruci. Sledeća naophodna stvar je dekodiranje sadržaja poruke što se radi pozivanjem metode *decodeMessage.* Rezultat dekodiranja poruke je sadržaj poruke u HTML formatu, koji se prikazuje u *template* delu komponente. Nakon dekodiranja poruke, vrši se provera da li postoje dodaci vezani za tu poruku, ako postojie kreiraju se linkovi (slika 37) koji omogućavaju da se ti dodaci preuzmu.

Sa ove stranice korisnik može da odgovori na poruku klikom na dugme Reply ili da je prosledi klikom na dugme Forward. Na slici 39 prikazana je forma koja omogućava slanje odgovora a na slici 40 forma koja omogućava prosleđivanje poruke.



Slika 39 - Forma koja omogućava slanje odgovora na poruku



Slika 40 - Forma koja omogućava prosleđivanje poruke

Nakon što korisnik unese svoj odgovor na poruku i klikne dugme *Send.* Poziva se metoda *sendReply* definisana unutar *methods* opcije *script* dela *MessagePreview*.*vue* komponente (slika 41). Ova metoda kreira *headers* objekat koji služi da odredi parametre poput toga kome se šalje odgovor i id poruke na koju se odgovara. Ova metoda zatim poziva *sendMessage* akciju definisanu unutar Vuex.Store instance (slika 42).

Slika - Metoda *sendReply*

sendReply() {

let headers = {

'To': this.getHeader(this.message.payload.headers, 'Return-Path'),

'In-Reply-To': this.message.id

};

this.sendMessage({

headers,

message: this.replyMessage

});

}

sendMessage(context, { headers, message }) {

let gapi = context.getters.gapi;

let alertify = context.getters.alertify;

var email = "";

for (var header in headers)

email += header += ": " + headers[header] + "\r\n";

email += "\r\n" + message;

var sendRequest = gapi.client.gmail.users.messages.send({

userId: "me",

resource: {

raw: window

.btoa(email)

.replace(/\+/g, "-")

.replace(/\//g, "\_")

}

});

sendRequest.execute(function(resp) {

alertify.success('Successfully sent message!');

router.push({ name: 'allMessages' });

});

}

Slika - Akcija *sendMessage* definisana unutar Vuex.Store instance

Akcija sa slika 42 kreira telo imejl poruke i kreira zahtev koji se šalje na Google Gmail API. Nakon uspešnog slanja poruke, ova akcija redirektuje aplikaciju na AllMessages.vue komponentu koja služi za prikaz svih poruka. Na sličan način funkcioniše i prosleđivanje poruka. Nakon što korisnik doda svoj sadržaj na poruku koju želi da prosledi i unese imejl adresu onoga kome želi da prosledi poruku, poziva se *sendForward* metoda (slika 43) koja takođe poziva *sendMessage* akciju.

Slika - metoda *sendForward*

sendForward() {

let headers = {

'To': this.forwardTo,

'Subject': this.getHeader(this.message.payload.headers, 'Subject')

};

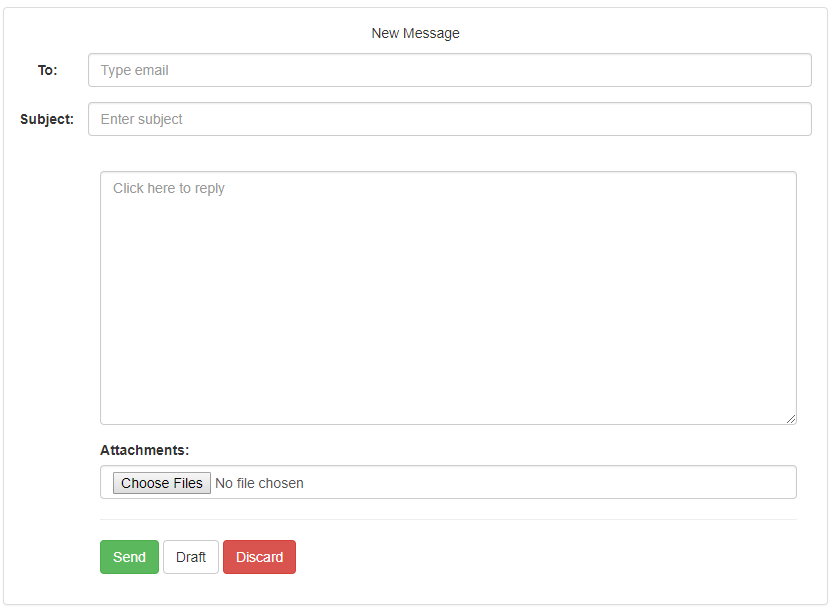
this.sendMessage({

headers,

message: this.forwardMessage

});

}

Jedna velika funkcionalna celina aplikacije je slanje novih imejl poruka. Za ovu funkcionalnost zadužena je *Compose.vue* komponenta. Izgled ove komponente je prikazan na slici broj 44.

Slika - Izgled *Compose.vue* komponente

Nakon što unese imejl adresu primaoca, naslov i telo poruke i (opciono) izabere datoteke koje želi da pošalje kao dodatak, korisnik može poruku ili da pošalje, sačuva kao *draft* ili da odustane i obriše kreiranu poruku (*Discard*). Implementacija *Compose.vue* komponente može se videti na sledećoj slici (slika broj 45).

<template>

     <div>

        <div class="panel panel-default">

            <div class="panel-body message">

                <p class="text-center">New Message</p>

                <form class="form-horizontal" role="form">

                    <div class="form-group">

                     <label for="to" class="col-sm-1 control-label">To:</label>

                     <div class="col-sm-11">

<input type="email" class="form-control select2-offscreen" id="to" placeholder="Type email" tabindex="-1" v-model="to">

                     </div>

                   </div>

                    <div class="form-group">

                     <label for="bcc" class="col-sm-1 control-label">Subject:</label>

                     <div class="col-sm-11">

<input type="email" class="form-control select2-offscreen" id="bcc" placeholder="Enter subject" tabindex="-1" v-model="subject">

                     </div>

                   </div>

                </form>

                <div class="col-sm-11 col-sm-offset-1">

                    <br>

                    <div class="form-group">

                        <textarea class="form-control" id="message" name="body" rows="12" placeholder="Click here to reply" v-model="messageText"></textarea>

                    </div>

                    <div>

                        <div class="form-group">

                            <label for="files">Attachments: </label>

                            <input type="file" name="files" id="files" class="form-control" multiple>

                        </div>

                    </div>

                    <hr>

                    <div class="form-group">

                        <button type="submit" class="btn btn-success" @click="sendEmail">Send</button>

                        <button type="submit" class="btn btn-default" @click="saveDraft">Draft</button>

                        <button type="submit" class="btn btn-danger" @click="navigateDiscard">Discard</button>

                    </div>

                </div>

            </div>

        </div>

    </div>

</template>

<script>

import { mapActions } from 'vuex';

export default {

data() {

return {

to: '',

subject: '',

                messageText: '',

                sendMessagesWithAttachments: false,

                attachmentFiles: []

}

},

methods: {

...mapActions([

                'sendMessage',

                'saveAsDraft',

                'sendMessageWithAttachments'

]),

sendEmail() {

                var headers = {

'To': this.to,

'Subject': this.subject

                    };

if(!this.sendMessagesWithAttachments){

                    this.sendMessage({

                        headers,

                        message: this.messageText

                    });

                }else {

                    this.sendMessageWithAttachments({

                        gapiHaders: headers,

                        messageText: this.messageText,

                        files: this.attachmentFiles}

                    );

                }

            },

            saveDraft() {

                let headers = {

'To': this.to,

'Subject': this.subject

                };

                this.saveAsDraft({

                    headers,

                    message: this.messageText

                });

            },

Slika -Implementacija *Compose*.*vue* komponente

navigateDiscard() {

                if(confirm('Are you sure you want to discard this message?')){

                    this.$router.go(-1);

                }

            },

            handleFileSelection(event) {

                let filesTidy = [];

                function getBase64(file) {

                    return new Promise((resolve, reject) => {

                        const reader = new FileReader();

                        reader.readAsBinaryString(file);

                        reader.onload = () => resolve(reader.result);

                        reader.onerror = error => reject(error);

                    });

                };

                this.sendMessagesWithAttachments = true;

                let files = event.target.files;

                for (let i = 0, f; f = files[i]; i++){

                    getBase64(files[0]).then(function(data) {

                        addToFilesArray(files[0].type, files[0].name, data);

                    })

                };

                function addToFilesArray(type, name, data) {

                    filesTidy.push({

                        type,

                        name,

                        data: window.btoa(data)

                    });

                }

                this.attachmentFiles = filesTidy;

            }

        },

        mounted() {

            let inputFilesTag = document.getElementById('files');

            inputFilesTag.addEventListener('change', this.handleFileSelection, false);

        }

}

</script>

Postoje dva alternativna toka dešavanja prilikom slanja nove poruke. Ukoliko korisnik odluči da pošalje datoteke kao dodatak uz poruku, kreira se drugačiji zahtev za server od onog koji se kreira kada korisnik ne doda nikakve datoteke. Klikom na dugme *Send* poziva se metoda *sendEmail* kao što se može videti na slici gore. Ta metoda podešava *header* objekat (parametre poput adrese primaoca ili naslova) i poziva odgovarajuću akciju definisanu u *Vuex.Store* instanci, *sendMessage* (akciju o kojoj je već pisano i može se videti na slici 42) ili *sendMessageWithAttachments.* Razlika je u tome što se kod ove druge kreira drugačiji zahtev za server, onaj koji uključuje i datoteke iz dodatka. Akcija *sendMessageWithAttachments* može se videti na slici 46.

Slika - *sendMessageWithAttachments* akcija

sendMessageWithAttachments(context, { gapiHeaders, messageText, files }) {

let body = createMimeMessage({

headers: gapiHeaders,

textHtml: messageText,

textPlain: messageText,

attachments: files

});

let token = localStorage.getItem('token');

let headers = {

Authorization: `Bearer ${token}`,

'Content-Type': 'multipart/related; boundary="foo\_bar\_baz"'

};

axios.post('https://www.googleapis.com/upload/gmail/v1/users/me/messages/send?uploadType=media', body, {headers: headers})

.then(function(response){

console.log(response);

})

.catch(function(error){

console.log(error);

});

}

Prilikom čuvanja poruke kao *draft,* pritiskom na dugme *Draft,* poziva se akcija sa slike 47. Ova akcija kreira zahtev za server, suštniski sličan zahtevu za slanje poruke ali na drugom *endpoint.*

Slika - *saveAsDraft* akcija

saveAsDraft(context, { headers, message }) {

let gapi = context.getters.gapi;

let alertify = context.getters.alertify;

var email = "";

for (var header in headers)

email += header += ": " + headers[header] + "\r\n";

email += "\r\n" + message;

var draftRequest = gapi.client.gmail.users.drafts.create({

userId: "me",

resource: {

message: {

raw: window

.btoa(email)

.replace(/\+/g, "-")

.replace(/\//g, "\_")

}

}

});

draftRequest.execute(function(resp) {

alertify.success('Successfully created a draft message!');

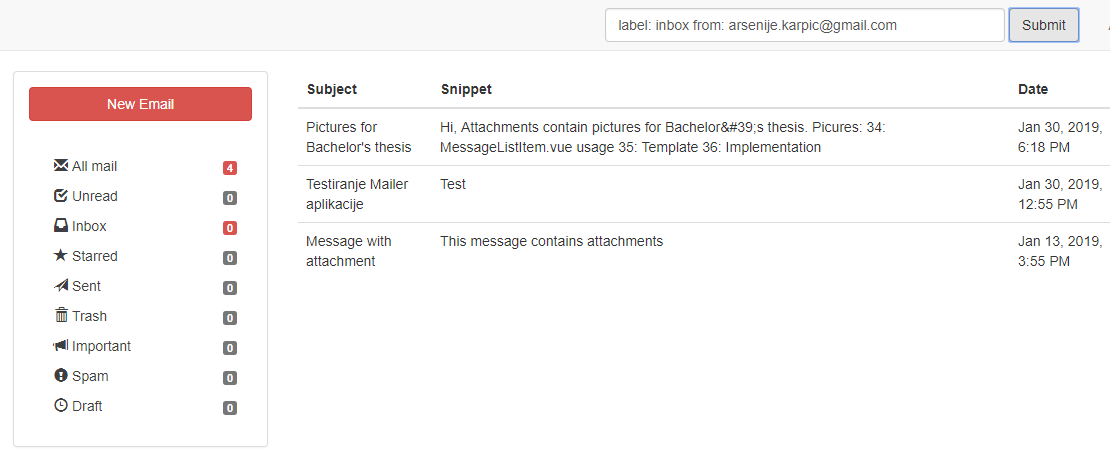
router.push({ name: 'allMessages' });

})

}

Još jedna važna funkcionalna celina aplikacije je pretraživanje poruka. Pretraživanje poruka može da se vrši korišćenjem polja za pretraživanje na vrhu stranice, kao što se može videti na slici ispod (slika broj 48). Za pretraživanje poruka se mogu koristiti svi operatori definisani na sledećoj veb stranici: <https://support.google.com/mail/answer/7190?hl=en>

Primer sa slike, ukoliko se u polje za pretragu upiše *label:inbox from:arsenije.karpic@gmail.com* izlistaće se sve poruke koje imaju sistemsku labelu *INBOX* i koje su poslane sa date imejl adrese.



Slika 48 - Pretraživanje poruka

Iz razloga što se polje za pretragu nalazi unutar *Header.vue* komponente metoda koja se poziva prilikom pretraživanja je definisana u toj komponenti. Metoda je prikazana na slici 49.

Slika - *handleSearch* metoda unutar *Header*.*vue* komponente

handleSearch() {

this.searchMessages(this.query);

this.$router.push({ name: 'searchedMessages'});

}

Ova metoda poziva *searchMessages* akciju i prosleđuje joj *query* (ono što je upisano unutar polja za pretragu) i zatim rutira aplikaciju na *SearchMessages.vue* komponentu koja prikazuje pretražene poruke na sličan načina kao opisana *Inbox.vue* komponenta. *SearchMessages* akcija je prikazana na slici 50.

Slika - *searchMessages* akcija

searchMessages(context, query) {

let gapi = context.getters.gapi;

let request;

let searchResult = [];

request = gapi.client.gmail.users.messages.list({

userId: "me",

maxResults: 10,

q: query

});

request.execute(function(response) {

for (let message of response.messages) {

var messageRequest = gapi.client.gmail.users.messages.get({

userId: 'me',

id: message.id

});

messageRequest.execute(function(resp) {

searchResult.push(resp);

});

}

context.commit('SET\_SEARCHED\_MESSAGES', searchResult);

});

}

# Zaključak

Projekat predstavlja aplikaciju koja je klon Gmail veb aplikacije. Kao takva implementira mali podskup operacija koje pruža Guglovo rešenje. Verzija aplikacije prikazana u ovom radu obuhvata rad sa imejl porukama koji uključuje izlistavanje poruka po labelama, čitanje poruka, odgovaranje na njih, njihovo prosleđivanje ili brisanje. Takođe uključuje i slanje novih poruka i njihovo pretraživanje.

Aplikacija je rađena koristeći aktuelne tehnologije. Zbog velike mogućnosti za proširenjem koje nudi Vue radni okvir i zbog korišćenja komponentalnog razvoja klijentske strane aplikaciju je lako nadograditi ostalim funkcionalnostima koje nudi prava Gmail veb aplikacija.

# Literatura

[1] Google Gmail veb aplikacija

<https://mail.google.com>

[2] Google Gmail API

<https://developers.google.com/gmail/api/>

[3] VueJS 2.0

<https://vuejs.org/>

[4] Vue Gapi

<https://www.npmjs.com/package/vue-gapi>

[5] Vue CLI

<https://cli.vuejs.org/>

[6] Vue Router

<https://github.com/vuejs/vue-router>

[7] Vuex State Management Library

<https://github.com/vuejs/vuex>

[8] JSON

<https://www.json.org/>

[9] Identity wizard

<https://developers.google.com/identity/>

[10] Webpack

<https://webpack.js.org/>

[11] Babel

<https://babeljs.io/>

[12] Node Package Manager

<https://www.npmjs.com>

[13] ES5 Javascript implementation

# 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**: | | Арсеније Карпић | |
| Ментор, **МН**: | | Проф Др Милан Видаковић | |
| Наслов рада, **НР**: | | Веб апликација за управљање имејл порукама | |
| 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) | | 5/69/0/0/50/0/0 | |
| Научна област, **НО**: | | Информатика | |
| Научна дисциплина, **НД**: | | Рачунарске науке | |
| Предметна одредница/Кqучне речи, **ПО**: | | Javascript, ВЕБ, REST, VueJS 2 | |
| **УДК** | |  | |
| Чува се, **ЧУ**: | | Библиотека Факултета  техничких наука, Трг Доситеја  Обрадовића 6, Нови Сад | |
| Важна напомена, **ВН**: | |  | |
| Извод, **ИЗ**: | | Задатак апликације представља развој једноставне веб апликације за управљање имејл порукама. Апликација је инспирисана већ постојећим решењем (Google Gmail). Серверски део апликације није посебно развијан док је клијентски део апликације развијен у VueJS 2 развојном окружењу и користи Google Gmail API. Апликација омогућава листање, читање, писање и претраживање имејл порука. | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | |  | |
| Чланови комисије, **КО**: | Председник: | др Игор Дејановић, ванредни професор, ФТН Нови Сад |
|  | Члан: | др Стеван Гостојић ванредни професор, ФТН Нови Сад | Потпис ментора |
|  | Члан, ментор: | др Милан Видаковић,  редовни професор, ФТН Нови Сад |  |

# 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**: | | Arsenije Karpić | |
| Mentor, **MN**: | | Prof Dr Milan Vidaković | |
| Title, **TI**: | | WEB APPLICATION FOR EMAIL 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, Trg Dositeja Obradovića 6  Sciences, | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | 5/69/0/0/50/0/0 | |
| Scientific field, **SF**: | | Electrical Engineering | |
| Scientific discipline, **SD**: | | Computer Science | |
| Subject/Key words, **S**/**KW**: | | Javascript, WEB, REST, VueJS 2 | |
| **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 email management inspired by an already existing solution (Google Gmail). Application’s server side is not separately implemented and client side application is implemented using VueJS 2 framework. Application handles listing, reading, composing and searching email messages. | |
| Accepted by the Scientific Board on, **ASB**: | |  | |
| Defended on, **DE**: | |  | |
| Defended Board, **DB**: | President: | Igor Dejanović, PhD, associate professor FTN Novi Sad |
|  | Member: | Stevan Gostojić, PhD, associate professor FTN Novi Sad | Menthor's sign |
|  | Member, Mentor: | Milan Vidaković, PhD, full professor,  FTN Novi Sad |  |

# Biografija

Арсеније Карпић, рођен 26.03.1995 у Новом Саду. Завршио Основну Школу „Јован Јовановић Змај“ у Ђурђеву и Гимназију „Јован Јовановић Змај“ у Новом Саду.

Контакт имејл адреса:

[arsenije.karpic@gmail.com](mailto:arsenije.karpic@gmail.com)

Контакт мобилни телефон:

+381640427675

1. Slika preuzeta sa codingexplained.com [↑](#footnote-ref-1)
2. Slika preuzeta sa https://vuex.vuejs.org/ [↑](#footnote-ref-2)