Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/04BE43Savin

РАЗВОЈ JavaScript БИБЛИОТЕКЕ ЗА РАД СА КОМПОНЕНТАМА КОРИСНИЧКОГ ИНТЕРФЕЈСА

DEVELOPMENT OF JavaScript LIBRARY FOR CREATING USER INTERFACE COMPONENTS

Ивана Савин, Милан Видаковић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај: *У овом раду представљено је креирање JavaScript библиотеке која ће омогућити једноставно креирање компоненти корисничког интерфејса. За демонстрацију рада ове библиотеке написана је једноставна веб апликација за управљање електронском поштом. Клијентски део апликације је развијен помоћу претходно креиране JavaScript библиотеке. Серверски део апликације је развијен у програмском језику Јава, коришћењем Spring окружења. Апликација омогућује основни скуп операција, попут управљања електронском поштом, креирања и слања поште.*

Кључне речи: *JavaScript, HTML, DOM, Gmail API, WEB, REST.*

Abstract: *This paper deals with the implementation of JavaScript library for creating user interface components. For demonstration how this library works, a simple web application for email management will be created. The client part of an application will be implemented using the previously created JavaScript library. The server part will be implemented using Java programming language and Spring framework. The application will provide basic managing features like listing, reading, composing and sending email messages.*

Key words: *JavaScript, HTML, DOM, Gmail API, WEB, REST.*

1. УВОД

У овом раду биће описана *JavaScript* [1] библиотека која омогућује једноставно креирање компоненти корисничког интерфејса, као и примена те библиотеке за развој једноставних веб апликација. Као пример за примену библиотеке креирана је апликација за управљање електронском поштом, инспирисана постојећим решењем (*Google Mail* [3]).

Будући да је било потребно развити веб апликацију, посебно су развијени клијентски и серверски део. Клијентски део апликације развијен је коришћењем претходно имплементиране библиотеке док је серверски део апликације Spring апликација која комуницира са *Gmail* API-јем.

НАПОМЕНА:
Овај рад је проистекао из мастер рада чији ментор је био проф. др Милан Видаковић.

2. КОРИШЋЕНЕ ТЕХНОЛОГИЈЕ

За имплементацију клијентског дела репрезентативне веб апликације коришћена је *OldSchoolComponents* [2]. *JavaScript* библиотека за креирање компоненти корисничког интерфејса, *jQuery* библиотека за управљање елементима DOM стабла [4], *Bootstrap* библиотека за дизајн компоненти [5] и *axios* библиотека за комуникацију са сервером [6].

ECMAScript 6

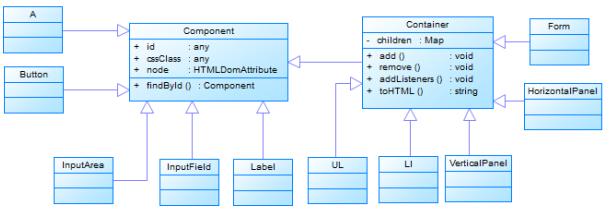
ECMAScript 6 [1] је верзија *JavaScript*-а која је увела значајна унапређења и олакшања у раду. Нека од значајнијих унапређења су објектно-оријентисани приступ раду и креирање *JavaScript* класа. ECMAScript 6 класе нуде објектно-оријентисани модел наслеђивања иако је у позадини JavaScript прототипско наслеђивање. На листингу број 1 се може видети креирање *JavaScript* класа.

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

Листинг 1. *Креирање класе помоћу* class *декларације*

OldSchoolComponents

OldSchoolComponents библиотека [2] је *JavaScript* библиотека намењена раду са компонентама корисничког интерфејса. Она пружа програмеру могућност да веб апликацију прави програмски, употребом кода, тако што се свака компонента на крају претвара у HTML, који се потом уграђује у основну страницу.



Слика 1. *Дијаграм класа*

Компоненте

Као што се може видети на слици 1, хијерархија графичких компоненти почиње са компонентом *Component* и она је базна осталим компонентама.

Основна компонента садржи атрибуте потребне за графичко исцртавање једне HTML компоненте а то су *id* атрибут за идентификацију, *CSSclass* атрибут за приказ компоненте, као и *node* атрибут који представља везу са DOM репрезентацијом компоненте.

***constructor* метода**

Специјална метода *constructor* је задужена за креирање и иницијализацију објеката креираних кључном речју *class*. Приликом креирања објекта класе *Component* и свих објеката који наслеђују ову класу, креира се објекат који има идентификатор и *css* атрибут. Такође, приликом креирања објектне репрезентације компоненте креира се и привремени елемент DOM стабла. На листингу 2 може се видети имплементација *constructor* методе у класи *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);
}
```

Листинг 2. Имплементација *constructor* методе

Остале компоненте које наслеђују основну компоненту имају могућност додавања специфичних атрибута и метода.

Контејнери

Поред основних компоненти приказа постоје и компоненте које унутар себе могу садржати неку другу компоненту. *Container* објекат се разликује од *Component* објекта по томе што може садржати друге *Component* објекте. Приликом креирања Container објекта поставља се атрибут *children* који представља мапу свих објеката које тај контејнер садржи.

Да би се нека компонента приказала потребно је додати је у DOM стабло. Компонента не може постојати уколико није додата у неки *Container* објекат. Из тог разлога постоји метода *add* која додаје компоненту у одређени контејнер на нивоу објектне репрезентације и на нивоу DOM стабла.

Уколико желимо динамичан приказ компоненти, без поновног учитавања комплетног интерфејса можемо неку компоненту обрисати и на њено место уметнути нову. Из тог разлога креирана је *remove* метода која истовремено брише објектну репрезентацију компоненте и брише компоненту из DOM стабла.

Приликом додавања компоненти позива се *tohtml* метода која враћа *string* репрезентацију компоненте. Приликом исцртавања контејнера исцртавају се и све компоненте које он садржи.

Google Gmail API

Google Gmail API [7] је скуп функција развијен од стране Google-а како би омогућио другим апликацијама да комуницирају са *Google* сервисима и да их интегришу у своја решења како би их побољшале или проширили. Имплементације овог сервиса постоје у програмским језицима Java, JavaScript, .NET. За већину веб апликација *Gmail API* је најбољи избор јер нуди ауторизован приступ корисниковим *Gmail* подацима. У *Google Gmail API*-ју свака функционалност представља посебан ресурс и тиме *Gmail API* нуди *RESTful* приступ функционалностима. Неке од функционалности које нуди *Gmail API* су:

- преглед електронске поште,
- манипулисање електронском поштом,
- креирање нове поште,
- слање поште и
- управљање лабелама.

Google API користи OAuth 2.0 протокол [8] за аутентикацију. OAuth 2.0 је једноставан протокол за аутентикацију а апликација која користи API мора да спецификује *scope* стрингове који служе за идентификовање ресурса које је могуће користити.

Gmail API је веб сервис који користи *RESTful* начин комуникације а ресурси су представљени у JSON формату. Google Gmail API нуди неколико различитих ресурса:

- *message* – ресурс који представља поруку. Порука може бити креирана или обрисана али се ни једно својство поруке не може изменити,
- *draft* – ресурс који представља поруку која је креирана али није послата. *Draft* ресурс је везан за један *message* ресурс који може бити измењен. Након слања, *draft* ресурс се брише и ресурсу се додаје системска лабела SENT,
- *label* – ресурс који служи као средство за категоризацију порука. Има many-to-many [9] везу са порукама. На једну поруку може бити примењено више лабела и једна лабела може бити примењена на више порука,
- *history* – ресурс који представља колекцију измењених порука у хронолошком редоследу,
- *thread* – ресурс који представља колекцију порука које представља конверзацију,
- *settings* – ресурс који пружа могућност контроле над Google налогом.

```
{
"id": string,
"name": string,
"messageListVisibility": string,
"labelListVisibility": string,
"type": string,
"messagesTotal": integer,
"messagesUnread": integer,
"threadsTotal": integer,
"threadsUnread": integer,
"color": {
"textColor": string,
"backgroundColor": string
}
}
```

Листинг 3 – JSON репрезентација *label* ресурса

Типичан процес рада са *Gmail API* сервисом се састоји из следећих корака:

- аутентикација,
- позив API методе и

- процесирање ресурса добијених у одговору.

Методе *Gmail* API-ја је могуће позивати путем HTTP метода, међутим постоје клијентске библиотеке које то олакшавају. У развоју апликације коришћена је Јава библиотека.

3. СПЕЦИФИКАЦИЈА АПЛИКАЦИЈЕ

Задатак обухвата израду *JavaScript* библиотеке за рад са компонентама корисничког интерфејса – OldSchoolComponents. За демонстрацију ове библиотеке написана је веб апликација за управљање електронском поштом. Будући да апликација комуницира са *Gmail API*-јем, корисник има могућност да користи основни скуп функционалности које оригинална апликација нуди. Апликација омогућава прегледање електронске поште, руковање поштом и њено ажурирање. Такође, омогућава креирање нових порука, слање порука и одговарање на постојеће.

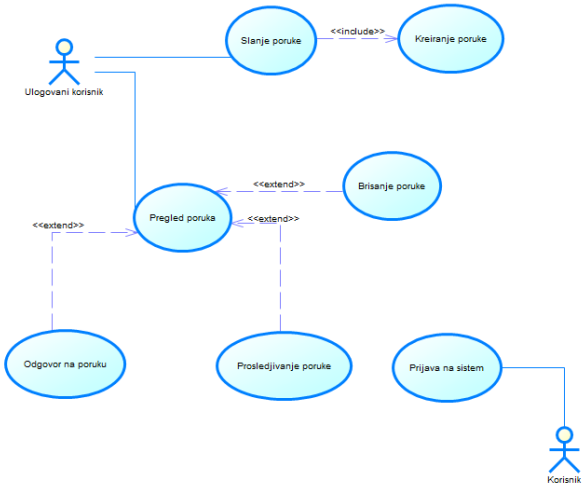
Приликом доласка на веб страницу апликације кориснику је омогућена пријава на систем. Пријава на систем се врши помоћу *Google*-овог система за аутентикацију.

Након успешне пријаве на систем кориснику се приказује главна страница апликације на којој корисник може да прегледа своју електронску пошту по лабелама. Корисник има могућност да креира нову поруку, одговори на неку од примљених порука или да поруку проследи другом кориснику. Поред главне странице са приказом примљене поште постоје још и странице са приказом поште из сваке лабеле.

На страници са поштом за брисање корисник може да прегледа пошту коју је пребацио у канту за брисање и уколико се предомисли да опозове ту акцију или да пошту директно обрише.

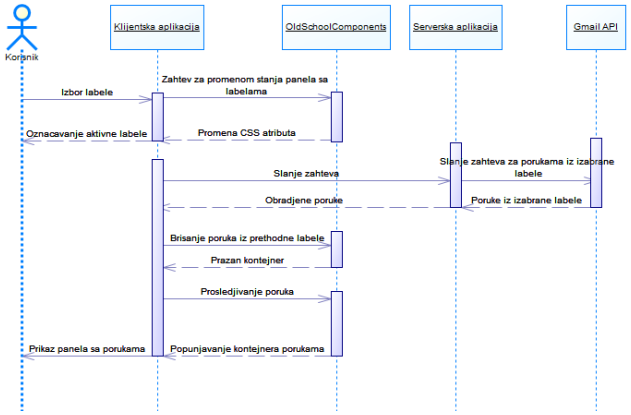
На страници са поштом припремљеном за слање корисник има могућност да поруку пошаље.

На слици 2 је приказан дијаграм случајева коришћења.



Слика 2 - *Дијаграм случајева коришћења*

На слици 3 је приказан дијаграм секвенце за приказ порука из селектоване лабеле. Секвенца догађаја почиње корисниковим избором жељене лабеле из панела са лабелама. Након што корисник изабере једну лабелу њој се промени CSS атрибут и она постаје активна а све остале лабеле имају CSS атрибут постављен на неактиван. Затим клијентска апликација шаље захтев серверској апликацији са називом изабране лабеле која потом упућује захтев Gmail API-ју за добављање свих порука из изабране лабеле. Gmail API враћа поруке серверској апликацији која потом те податке форматира и шаље клијентској апликацији. Након што пристигну подаци на клијентску апликацију, контејнер са порукама из претходне лабеле се бришу и потом се исцртавају нове компоненте за подацима.



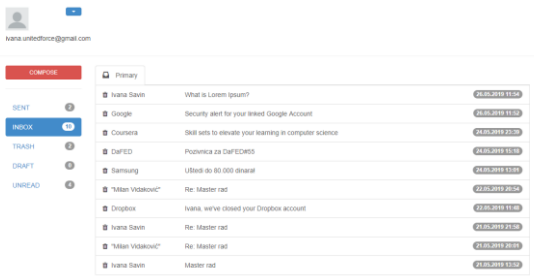
Слика 3. *Дијаграм секвенце*

4. ИМПЛЕМЕНТАЦИЈА

Апликацију чине две целине, серверски и клијентски део. Серверски део апликације је имплементиран у програмском језику Java [10], верзије 1.8 коришћењем *Spring* окружења. За имплементацију клијентског дела апликације коришћена је претходно креирана *OldSchoolComponents* библиотека.

Клијентска апликација

Клијентски део апликације представља *view* компоненту у архитектури ове апликације. Клијентски део апликације чине методе за исцртавање компоненти и методе за комуникацију са серверским делом апликације и добављање порука. Почетно стање апликације је главни прозор са приказом примљене поште и може се видети на слици 4.



Слика 4. *Главни прозор апликације*

Серверска апликација

Серверски део апликације је креиран зарад лакше комуникације са Google API-јем. На серверској страни

се налазе слој *controller*-а и *service*-а. Будући да слој controller-а садржи само позиве ка Gmail API-ју сви позиви су груписани у једну класу. Сервисни слој се састоји од три класе у којима се налазе методе за процесирање података потребних контролеру. Као што је споменуто, класа из слоја *controller* садржи методе за аутентикацију, методе за прикупљање корисникових лабела или порука, креирање и слање порука. Сервисни слој садржи методе које трансформишу потребне податке у одговарајући JSON објекат.

5. ЗАКЉУЧАК

Основни задатак овог рада је био развој и демонстрација JavaScript библиотеке за рад са компонентама графичког корисничког интерфејса – OldSchoolComponents. Основна идеја је била да се компоненте корисничког интерфејса могу декларативно направити и додати на екран, како се иначе ради у програмским језицима који се не користе за развој веб апликација.

За демонстрацију OldSchoolComponents библиотеке, развијена је једноставна апликација за управљање електронском поштом, инспирисана већ постојећим решењем. Досадашњи рад обухвата основни скуп операција за управљање електронском поштом, попут прегледања поште, манипулисања поштом, креирања нове поште и одговарања на постојећу.

Апликација је развијана у тренутно актуелним технологијама и има добру основу за даљи развој. Библиотека која је коришћена за креирање компоненти корисничког интерфејса је у процесу развоја тако да су проширења могућа.

6. ЛИТЕРАТУРА

[1] JavaScript, *https://en.wikipedia.org/wiki/JavaScript*
[2] OldScoolComponents, *https://github.com/iv17/OldSchoolComponents*
[3] Google Mail, https://mail.google.com/mail/
[4] jQuery, *https://api.jquery.com/*
[5] Bootstrap, *https://getbootstrap.com/*
[6] axios, *https://github.com/axios/axios*
[7] Google Mail API, *https://developers.google.com/gmail/api/v1/reference/*
[8] OAuth 2.0, *https://en.wikipedia.org/wiki/OAuth*
[9] many-to-many, *https://en.wikipedia.org/wiki/Many-to-many_(data_model)*
[10] Java, *https://en.wikipedia.org/wiki/Java_(programming_language)*

Кратка биографија:

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.

Милан Видаковић је рођен у Новом Саду 1971. године. На Факултету техничких наука у Новом Саду завршио је докторске студије 2003. године. На истом факултету је 2014. године изабран за редовног професора из области *Примењене рачунарске науке и информатика*.