**UVOD**

Osnovni zadatak ovog rada je bio razvoj i demonstracija *JavaScript* biblioteke za rad sa komponentama grafičkog korisničkog interfejsa – *OldSchoolComponents*.

Osnovna ideja je bila da se komponente korisničkog interfejsa mogu deklarativno napraviti i dodati na ekran, kako se inače radi u programskim jezicima koji se ne koriste za razvoj veb aplikacija.

Ona pruža programeru mogućnost da veb aplikaciju pravi programski, upotrebom koda, tako što se svaka komponenta na kraju pretvara u HTML, koji se zatim ugrađuje u osnovnu stranicu.

Za demonstraciju *OldSchoolComponents* biblioteke, razvijena je jednostavna aplikacija za upravljanje elektronskom poštom, inspirisana već postojećim rešenjem.

**OPIS BIBLIOTEKE**

*Gledajuci DOM stablo, vidimo da je to hijerahija komponenti koje se medjusobno mogu kombinovati. Kao sto je receno, biblioteka je JS biblioteka, tacnije ECMAScript* 6 bibilioteka.

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

COMPONENT

Hijerarhija grafičkih komponenti počinje sa komponentom *Component* i ona je bazna ostalim komponentama i može se videti na slici 1.1. Osnovna komponenta sadrži atribute potrebne za grafičko iscrtavanje jedne HTML komponente a to su *id* atribut za identifikaciju, *CSSclass* atribut za prikaz komponente, kao i *node* atribut koji predstavlja vezu sa DOM reprezentacijom komponente.

Constructor

Specijalna metoda *constructor* je zadužena za kreiranje i inicijalizaciju objekta kreiranih sa ključnom rečju *class*. Prilikom kreiranja objekta klase *Component* i svih objekata koji nasleđuju ovu klasu kreira se objekat koji ima identifikator i *css* atribut. Takođe, prilikom kreiranja objektne reprezentacije komponente kreira se i privremeni element DOM stabla. Na listingu 1.3 se može videti implementacija *constructor* metode u klasi *Component*.

CONTAINER

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

*Container* objekat se razlikuje od *Component* objekta po tome što može sadržati druge *Component* objekte i samim tim se njegov konstruktor razlikuje. Prilikom kreiranja *Container* objekta postavlja se atribut *children* koji predstavlja mapu svih komponenti koje taj kontejner sadrži. Na listingu 1.5 može se videti implementacija *constructor* metode.

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

Ukoliko želimo dinamičan prikaz komponenti, bez ponovnog učitavanja kompletnog interfejsa možemo neku komponentu obrisati i na njeno mesto umetnuti novu. Iz tog razloga kreirana je *remove* metoda koja istovremeno briše objektnu reprezentaciju komponente i briše komponentu iz DOM stabla. Implementacija ove metode se može videti na listingu 1.7.

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

**OPIS APLIKACIJE**

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

Budući da aplikacija komunicira sa *Gmail API*-jem korisnik ima mogućnost da koristi osnovni skup funkcionalnosti koje originalna aplikacija nudi. Aplikacija omogućava pregledanje elektronske pošte, rukovanje poštom i njeno ažuriranje. Takođe omogućava kreiranje novih poruka, slanje poruka i odgovaranje na postojeće

**KRAJ**

Dosadašnji rad obuhvata osnovni skup operacija za upravljanje elektronskom poštom poput pregledanja pošte, manipulisanja poštom, kreiranje nove pošte i odgovaranja na postojeću.

Aplikacija je razvijana u trenutno aktuelnim tehnologijama i ima dobru osnovu za dalji razvoj.

OldSchoolComponents biblioteka koja je korišćena za kreiranje komponenti korisničkog interfejsa je u procesu razvoja, poseduje jednostavan API I lako je nadogradiva tako da su proširenja moguća.

1. LOGOVANJE

2. PRIKAZ SVIH PORUKA

3. PRIKAZ NEPROCITANIH

4. OTVARANJE PORUKE

5. ODGOVARANJE NA PORUKU

6. SLANJE PORUKE

7. PRIKAZ POSLATE PORUKE

8. PROSLEDJIVANJE TE PORUKE

9. COMPOSE, SLANJE NOVE PORUKE

10. PRIKAZ SVIH POSLATIH PORUKA

11. PRIKAZ SVIH PORUKA ZA BRISANJE

12. IZBOR JEDNE PORUKE I PREBACIVANE U TRASH

13 REFRESH