PREDMET IT355 Web Sistemi

Jesenji semestar 2016/2017

Implementacija web sistema za rešenje problema računovođstva

Dušan Stanojević 1940

Ddusan.stanojevic.1940@metropolitan.ac.rs

Beograd, 2016

Abstrakt

Izvor informacija o praćenju i vođenju poslova je nemoguće izvesti bez računovođstva. Standardno računovođstvo na papiru pati od problema fleksibilnosti i pristupa u smislu izmena, pronalaženja podataka i generisanja izveštaja. Jedno od mogućih rešenja ovih problema biće demonstrirano pomoću tehnologija Spring i Ember 2. Rešenje će implementirati SOA arhitekturu.

Ključne reči: Spring, Hibernate, MySQL, REST, JSON, Spring Security, Ember, ES6

Napomena:
U daljem tekstu će biti korišćeni Engleski termini za opis određenih pojmova. Termini koji budu korišćeni predstavljeni su na Tabeli 0.1 Prevodi.

Bill	Račun (koji se plaća)
Account	Račun (za beleženje transakcija)
Vendor/Customer	Dobaljač/Kupac
Invoice	Faktura
Deployment	Puštanje u produkciju

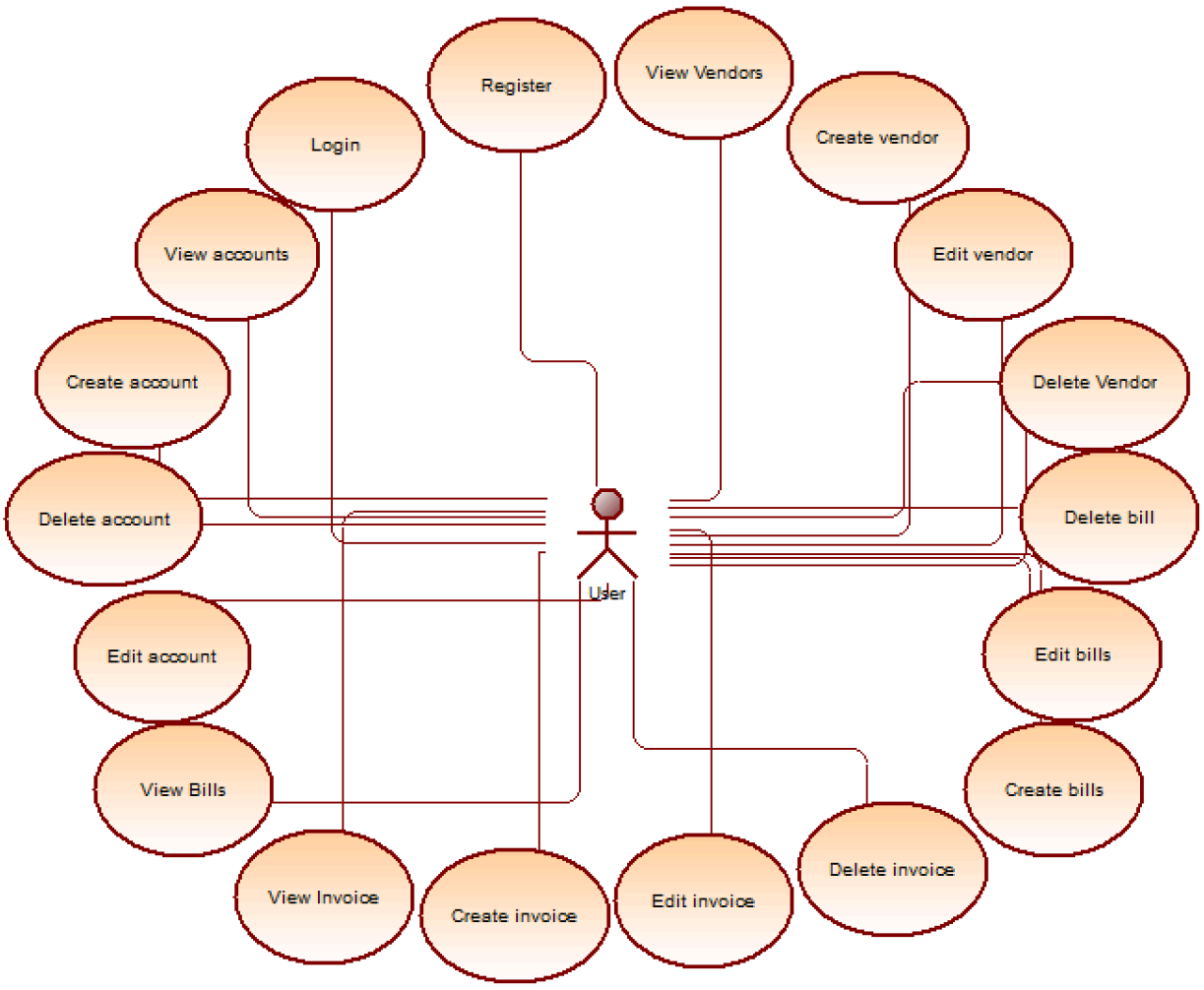
Tabela 0.1 Prevodi

Sadržaj

1. Definicija zahteva

1.1 Funkcionalni zahtevi

Funkcionalni zahtevi predstavljaju zahteve koje sistem treba da ispuni. U ovom poglavlju ćemo ih predstaviti korišćenjem dijagrama korišćenja (Use Case).



Slika 1.1.1 Use Case Dijagram Aplikacije

ID	UC-1
Ime	Login
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik odlazi na stranicu, unosi svoje podatke i loginuje se

ID	UC-2
Ime	Register
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik odlazi na register stranicu i unosi svoje podatke, novi entitet biva kreiran u bazi i beleži podatke korisnika

ID	UC-3
Ime	View Vendors
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik bira opciju Vendors u panelu sa leve strane i dobija prikaz svih vendora koje je dodao

ID	UC-4
Ime	Create Vendors
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik bira opciju Vendors u panelu sa leve strane, pritiska dugme Add Vendors, popunjava polja potrebna za kreiranje vendor-a. Nakon unosa validnih podataka korisnik

ID	UC-5
Ime	Edit Vendors
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik bira opciju Vendors u panelu sa leve strane, pritiska na Vendora koga zeli da izmeni, unosi podatke i pritiska opciju za cuvanje izmena

ID	UC-6
Ime	Delete Vendor
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik odlazi na Vendors stranicu, bira opciju za brisanje Vendor-a i potvrđuje svoju odluki. Vendor biva obrisan.

ID	UC-7

Ime	Delete Bill
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik odlazi na Bill stranicu, bira opciju za brisanje Bill-a i potvrđuje svoju odluki. Bill biva obrisan.

ID	UC-8
Ime	Edit Bills
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik odlazi na Bills stranicu i bira Bill koji želi da izmeni. Korisnik može da dodaje linije na Bill koji je odabrao.

ID	UC-9
Ime	Create Bills
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik bira opciju Bills u panelu sa leve strane, pritiska dugme Add Bills, popunjava polja potrebna za kreiranje Bill-a. Nakon unosa validnih podataka korisnik

ID	UC-10
Ime	Delete Invoice
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik odlazi na Invoice stranicu, bira opciju za brisanje Invoice-a i potvrđuje svoju odluki. Invoice biva obrisan.

ID	UC-11
Ime	Edit Invoice
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik odlazi na Invoices stranicu i bira Invoice koji želi da izmeni. Korisnik može da dodaje linije na Invoice koji je odabrao.

ID	UC-12
Ime	Create Invoice
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik bira opciju Invoices u panelu sa leve strane, pritiska dugme Add Invoice, popunjava polja potrebna za kreiranje Invoice-a. Nakon unosa validnih podataka korisnik

ID	UC-13
Ime	View Invoice

Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik bira opciju Invoice u panelu sa leve strane i dobija prikaz svih Invoice-a koje je dodao

ID	UC-14
Ime	View Bills
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik bira opciju Bills u panelu sa leve strane i dobija prikaz svih vendora koje je dodao

ID	UC-15
Ime	Edit Account
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik bira opciju Accounts u panelu sa leve strane, pritiska na Account koga zeli da izmeni, unosi podatke i pritiska opciju za cuvanje izmena

ID	UC-16
Ime	Delete Account
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik odlazi na Account stranicu, bira opciju za brisanje Account-a i potvrđuje svoju odluki. Account biva obrisan.

ID	UC-17
Ime	Create Account
Aktor	Korisnik
Preduslov	Korisnik ima internet konekciju
Glavni scenario	Korisnik bira opciju Accounts u panelu sa leve strane, pritiska dugme Add Account, popunjava polja potrebna za kreiranje Account-a. Nakon unosa validnih podataka korisnik

1.2 Nefunkcionalni zahtevi

1.2.1 Performanse

Sistem mora da radi na rakorisničkom računaru sa

1GB Memory
1 Core Processor
30GB SSD Disk
2TB Transfer

Svaki zahtev ka bazi mora da bude procesiran u roku od 2.5 sekundi ukoliko na serveru ima ispod 100 zahteva po sekundi.

1.2.2 Bezbednost

Pošto se baza nalazi na serveru i klijenti unose svoje informaciju u bazu, potrebno je dosta pažnje posvetiti bezbednosti. Bazu treba zaštiti od različitih napada, poput injection-a i server od neautorizovanog pristupa. Svi vidovi injection-a moraju da budu primenjeni. Administrator servera mora da bude u stanju da doda SSL sertifikat kako bi mogao da zaštiti od različitih napada presretanjem.

1.2.3 Sigurnost

Pristup uslugama sistema imaju svi registrovani korisnici. Mogućnost dodavanja i editovanja klijenata projekata i zadataka ima samo korisnik kome pripadaju.

1.2.4 Raspoloživost

Sistem ce biti dostupan najmanje 23 časa i 55min, svih 365 dana u godini. Down time servera sme da bude samo 5 minuta na dan.

1.2.5 Robusnost

Sistem će se izvršavati na klijentu. Software mora da radi na svim kompjuterima proizvedenim u poslednjih 4 godina za komercijalno tržište i poslednjoj verziji najzastupljenijih 4 browser-a (Chrome, Firefox, Internet Explorer, Safar).

2. Tehnologije koje ćemo koristiti

2.1 Back End Spring

Back end je obično deo aplikacije koji se izvršava na serverskoj strani. Neki od najčešće korišćenih programskih jezika za razvoj aplikacija koje rade na serverskoj strani su Java, Ruby, Python, C#, PHP, javascript… Svaki od ovih programskih jezika ima svoje prednosti i mane, za izradu serverskog dela aplikacije koristili smo programski jezik Java.

Java je strogo tipiziran Objektno Orisjentisan programski jezik koji je našao primenu u ogromnom broji oblasti od razvoja softvera za ugrađene uređaje do razvoja aplikacija na serverskoj strani. Za izradu ove aplikacije koristićemo Spring framework. Pored Spring-a za razvoj veb aplikacija sa java programskim jezikom često se koristi i Play framework.

Spring je framework baziran na ideji konfiguracije pre konvencije što je suprotno u odnosu na Play framework koji preferira konvenciju radije nego konfiguraciju. Spring framework je za vreme svog postojanja uspeo da postavi standard za kvalitet framework-a u java zajednici i podržava veliku količinu paradigmi prilikom programiranja. Jedna od stvari koju Spring podržava, a Play ne, je mogućnost za aspektno orijentisano programiranje.

Pošto razvijamo SOA aplikaciju naš backend mora da obezbedi autentikaciju, servise, i pristup podacima sa validacijom istih. Ovaj deo izrade je opisan u poglavlju “3. Implementacija aplikacije”.

2.2 Front End EmberJS 2

Ember je open source Javascript framework za razvoj klientskog dela web aplikacije i koristi MVC arhitekturalni patern. U Ember-u rute se koriste za učitavanje model-a dok se kontroler koristi za manipulaciju podacima u model-u.

Originalno se zvao SprutCore MVC framework i razvio ga je Yehuda Katz i publikovao ga je decembra 2011 godine.

Ember je fleksibilan framework licenciran pod MIT licencom i baziran je na konceptu brzih veb stranica. Dozvoljava da se ubrzaju performanse rada aplikacije tako što se ne učitava cela stranica uvek. Koristi handlebars biblioteku za template-in koja je jako slična HTML-u samo što može da ima embedovane izraze.

Ember aplikacije su manje po veličini u odnosu na vanila javascript aplikacije. Data binding je u dva smera i omogućava da izmena s jedne strane da se odmah vidi sa druge strane, strane u ovoj rečenici predstavljaju view, model i kontroler.

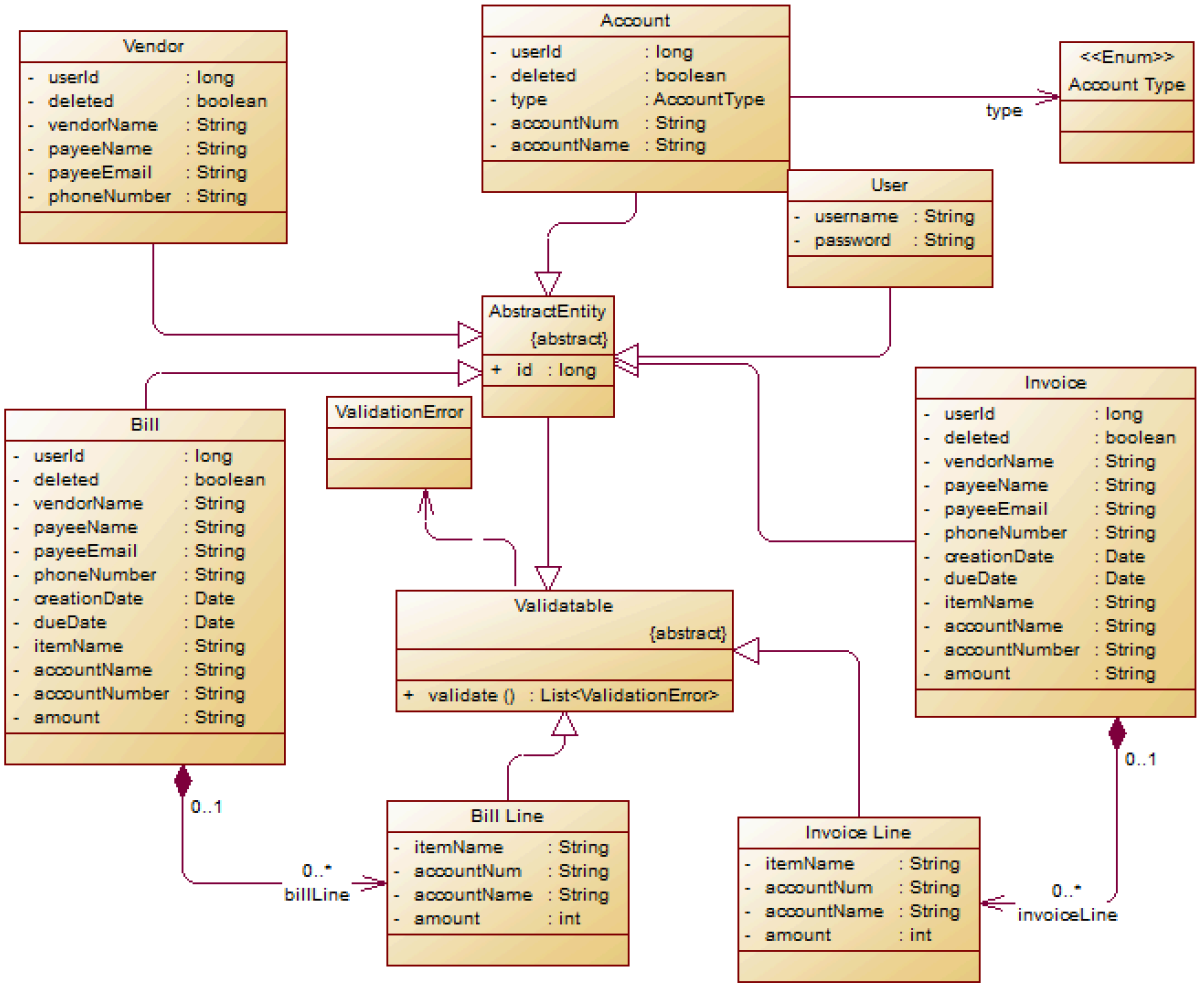
Neke od osnovnih funkcija Ember-a su da se kreiraju standardizovane Javascript front end aplikacije koje je lako održavati. Ruta i kontroler su automatski odabrani te nema potrebe da se ovo ručno radi i/ili proverava.

Ember ima HTML i CSS u srži svog modela za razvoj aplikacije kao i rute koje koristi kroz promenu lokacije (url-a).

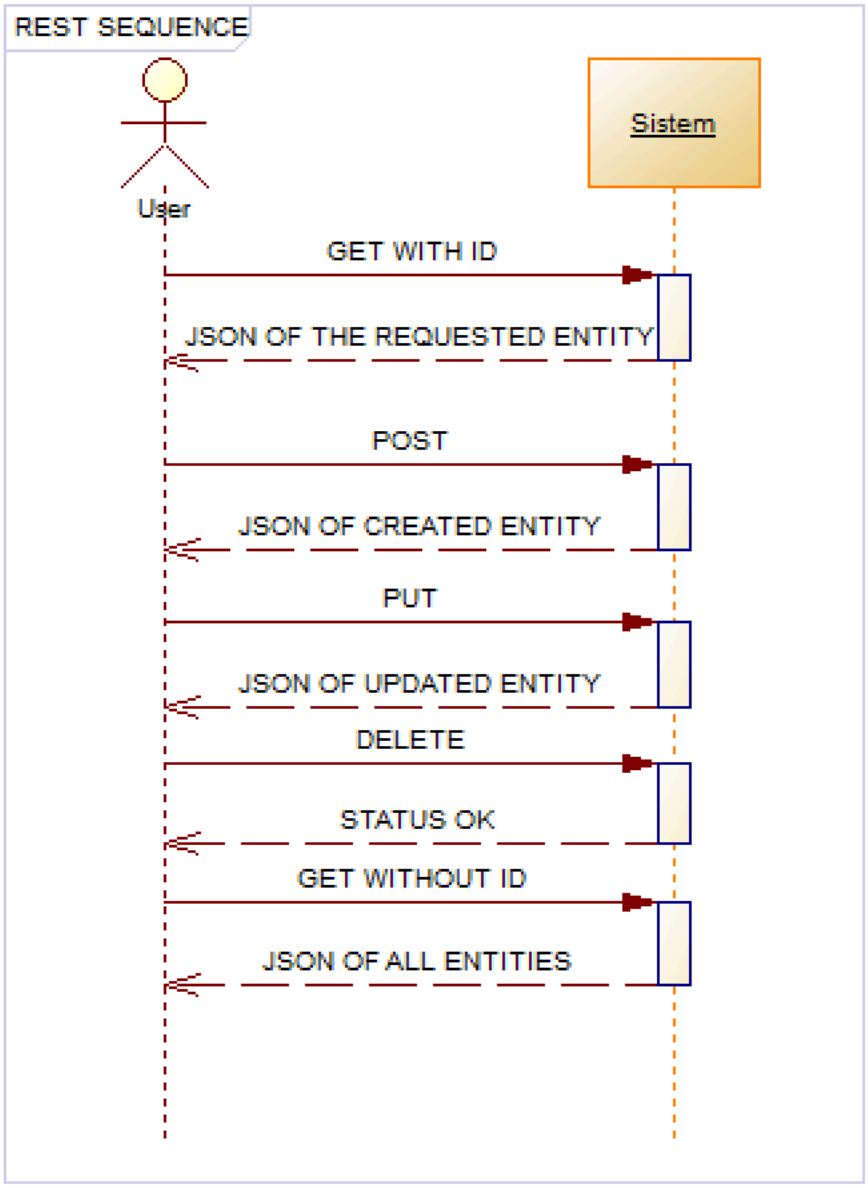
3. Implementacija aplikacije

3.1 Modeli

Podaci predstavljaju srž svake aplikacije, u Hibernate framework-u modeli se prestavljaju svojom definicijom u sklopu /model foldera. Svaki model je definisan u Java programskom jeziku i pomoću ORM-a mapiran ka bazi. Na slici 3.1.1 predstavljen je klasni dijagram aplikacije, a na slici 3.1.2 predstavljen je Sistem Sekvencijalni Dijagram (SSD) aplikacije.



Slika 3.1.1 Klasni Dijagram aplikacije



Slika 3.1.2 SSD aplikacije

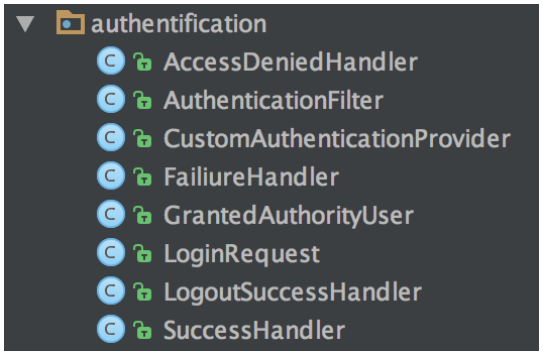
3.2 Autentifikacija

Autentifikacija korisnika je u aplikaciji izvršena pomoću Spring Security-a koji je ponudio osnovu za podešavanje same authentifikacije. Autentifikacija se odvija kroz users tabelu u bazi podataka koja čuva podatke o korisnicima.

Pošto Spring Security sam po sebi nema podešavanje za standardan REST bilo je potrebno implementirati njegove interface-e i abstrakcije. Implementirani interface-i i abstrakcije se mogu videti na slici 3.2.2.

```
<security:authentication-manager alias="authenticationManager">
    <security:authentication-provider ref="customAuthenticationProvider" />
</security:authentication-manager>

<security:http use-expressions="true" auto-config="false" entry-point-ref="http403EntryPoint">
    <security:intercept-url pattern="/bills**" access="hasAuthority('user')" />
    <security:intercept-url pattern="/invoices**" access="hasAuthority('user')" />
    <security:intercept-url pattern="/accounts**" access="hasAuthority('user')" />
    <security:intercept-url pattern="/vendors**" access="hasAuthority('user')" />
    <security:intercept-url pattern="/users**" access="hasAuthority('user')" />

    <security:custom-filter ref="CustomUsernamePasswordAuthenticationFilter" position="FORM_LOGIN_FILTER"/>

    <security:logout invalidate-session="true" success-handler-ref="logoutSuccessHandler" delete-cookies="true"/>

    <security:session-management>
        <security:concurrency-control max-sessions="1" error-if-maximum-exceeded="true" />
    </security:session-management>

    <security:csrf disabled="true"/>

    <security:access-denied-handler ref="accessDeniedHandler" />
</security:http>

<bean id="CustomUsernamePasswordAuthenticationFilter" class="com.it355.authentification.AuthenticationFilter">
    <property name="authenticationManager" ref="authenticationManager" />
    <property name="authenticationSuccessHandler" ref="customSuccessHandler"/>
    <property name="authenticationFailureHandler" ref="failureHandler"/>
    <property name="filterProcessesUrl" value="/j_spring_security_check"/>
    <property name="usernameParameter" value="j_username"/>
    <property name="passwordParameter" value="j_password"/>
</bean>
<bean id="accessDeniedHandler" class="com.it355.authentification.AccessDeniedHandler"></bean>
<bean id="customSuccessHandler" class="com.it355.authentification.SuccessHandler"></bean>
<bean id="failureHandler" class="com.it355.authentification.FailiureHandler"></bean>
<bean id="logoutSuccessHandler" class="com.it355.authentification.LogoutSuccessHandler"></bean>

<bean id="http403EntryPoint" class="org.springframework.security.web.authentication.Http403ForbiddenEntryPoint" />


<bean id="customAuthenticationProvider" class="com.it355.authentification.CustomAuthenticationProvider" autowire="byName"></bean>
```

Slika 3.2.1 XML konfiguracija Spring Security-a



Slika 3.2.2 Osnovne implementacije Spring Security Interface-a i abstrakcija zarad postizanja Rest specifikacije

```java
package com.it355.authentification;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.BufferedReader;

public class AuthenticationFilter extends UsernamePasswordAuthenticationFilter {

    @Override
    public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) {
        String username = "", password = "";

        if ("application/json; charset=UTF-8".equals(request.getHeader("Content-Type"))) {
            try {
                StringBuffer sb = new StringBuffer();
                String line;

                BufferedReader reader = request.getReader();
                while ((line = reader.readLine()) != null){
                    sb.append(line);
                }

                ObjectMapper mapper = new ObjectMapper();
                LoginRequest loginRequest = mapper.readValue(sb.toString(), LoginRequest.class);

                username = loginRequest.getUsername();
                password = loginRequest.getPassword();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        UsernamePasswordAuthenticationToken authRequest = new UsernamePasswordAuthenticationToken(username, password);

        setDetails(request, authRequest);

        return this.getAuthenticationManager().authenticate(authRequest);
    }
}
```

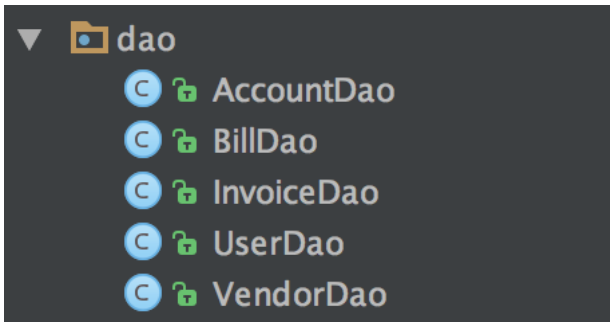
Slika 3.2.3 Primer Authentication Filtera

```java
package com.it355.utils;

import com.it355.models.User;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;

@Component
public class SessionUtils {
    public long getUserId() {
        return ((User) SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getId();
    }

    public String getUsername() {
        return ((User) SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getUsername();
    }
}
```

Slika 3.2.4 Primer Session Utils klase

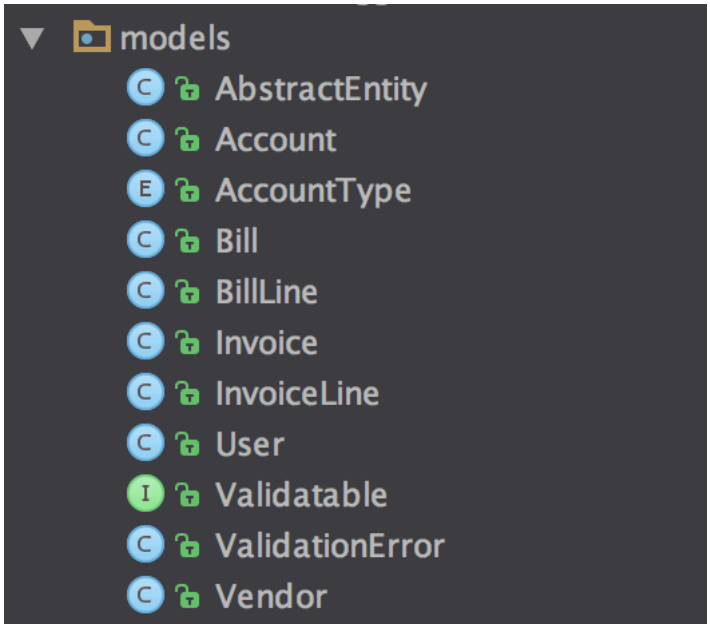
3.3 Modeli i Validacija

Postoji mnogo design pattern-a za rad sa podacima. Dva najzastupljenija u veb programiranju su Active Record i Data Access Object. Oba imaju svojih prednosti i mana u koje ovaj rad neće ulaziti. Za ovaj rad je iskorišćen Data Access Object design pattern, i konkretne DAO klase su prakazani na slici 3.3.1.



Slika 3.3.1 Osnovne DAO klase

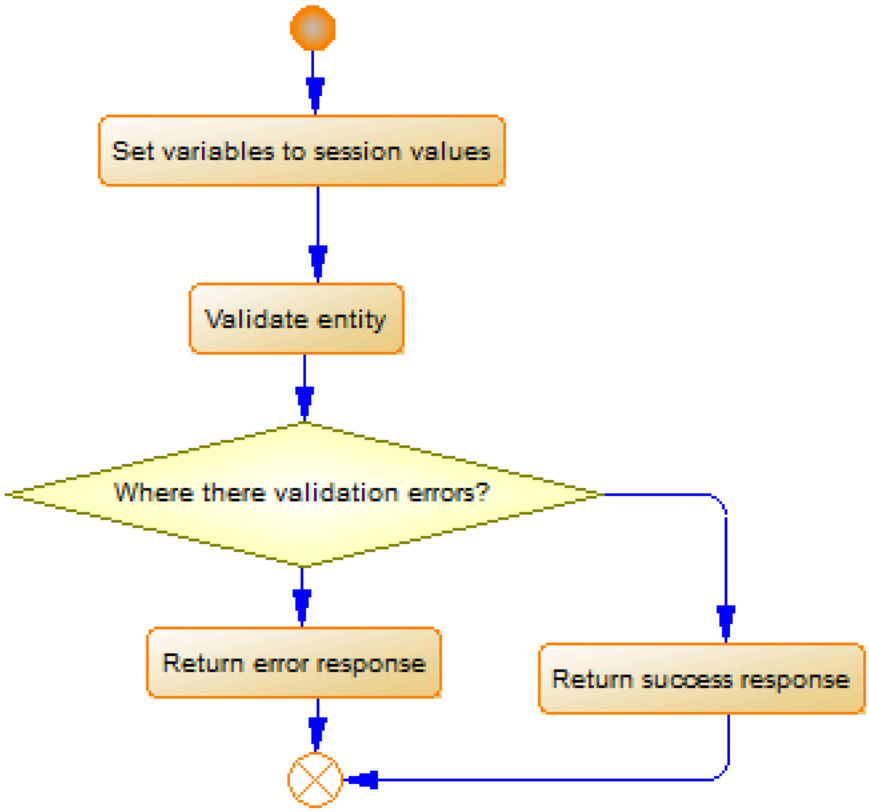
Modeli same aplikacije su prikazani na slici 3.4.1. Postoje dva osnovna nivoa validacije i zaštite implementirana nad ovom aplikacijom. Prvi je na nivou samih podataka kroz interface Validatable. Prvi nivo validacije se stara da ne dođe do korupcije podataka. Drugi nivo validacije koji štiti od pristupa i izmene tuđih podataka je implementiran kroz same metode DAO sloja u vidu zavisnosti od identifikatora korisnika koji pokušava podacima da pristupi.



Slika 3.4.1 Modeli aplikacije

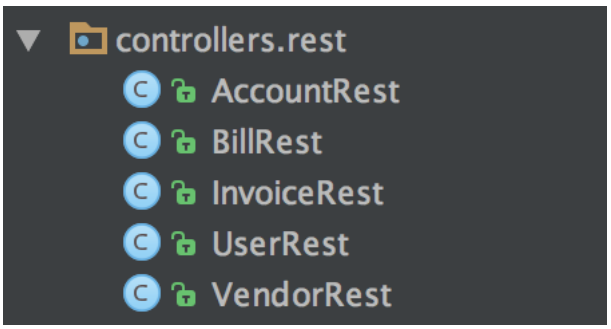
3.4 Kontroleri

Svaki kontroler aplikacije predstavljaće određeni Rest API endpoint sa osnovnim metodama vezanim za njega. Ovo je primer Single Responsibility principa. Svaka metoda koja se bavi baratanjem (CUD operacijama) sa podacima mora da ima tok predstavljen na slici 3.4.1.



Slika 3.4.1 Flow dijagram metoda koje rade CUD operacije

Osnovni kontroleri aplikacije su predstavljeni na slici 3.4.2.



Slika 3.4.2 Osnovni kontroleri aplikacije

```java
import com.it355.utils.ResponseUtils;
import com.it355.utils.SessionUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;


@RestController
@RequestMapping("/bills")
public class BillRest {

    @Autowired
    private BillDao billDao;
    @Autowired
    private SessionUtils sessionUtils;
    @Autowired
    private ResponseUtils responseUtils;


    @GetMapping("")
    public ResponseEntity getBills() {
        return responseUtils.okResponse("bills", billDao.findAll(sessionUtils.getUserId()));
    }

    @GetMapping("/{id}")
    public ResponseEntity getBill(@PathVariable("id") Long id) {
        return responseUtils.okResponse("bill", billDao.findById(id, sessionUtils.getUserId()));
    }

    @PostMapping(value = "")
    public ResponseEntity createBill(@RequestBody Map<String, Bill> billMap) {
        billMap.get("bill").setUserId(sessionUtils.getUserId());
        List<ValidationError> errors = billMap.get("bill").validate();
        if (errors.size() > 0) {
            return responseUtils.errorsResponse(errors);
        }
        return responseUtils.okResponse("bill", billDao.insert(billMap.get("bill")));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity deleteBill(@PathVariable Long id) {
        billDao.delete(billDao.findById(id, sessionUtils.getUserId()));
        return new ResponseEntity(id, HttpStatus.OK);
    }
```

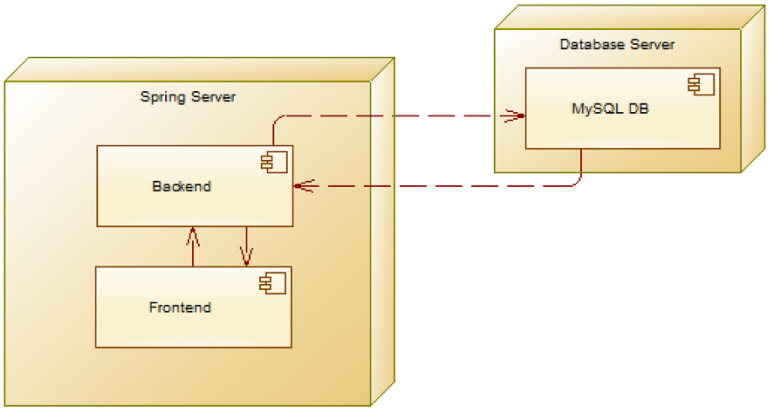
Slika 3.4.3 Primer koda kontrolera Bills entiteta

4. Deployment aplikacije

Kako bismo testirali aplikaciju dovoljno je da uradimo deployment Spring aplikacije u Tomcat web container koji radi na portu 8080 i da pokrenemo komandu “ember server --port=http://localhost:8080/” ovo će nam omogućiti radno okruženje za vreme razvoja same aplikacije. Ember server se po defaultu pali na portu 4200. Ember koristi NodeJS za vreme lokalnog razvoja aplikacije.

Deployment aplikacije u produktivno okruženje ćemo izvesti kombinacijom Sprint MVC-a i Ember JS-a.

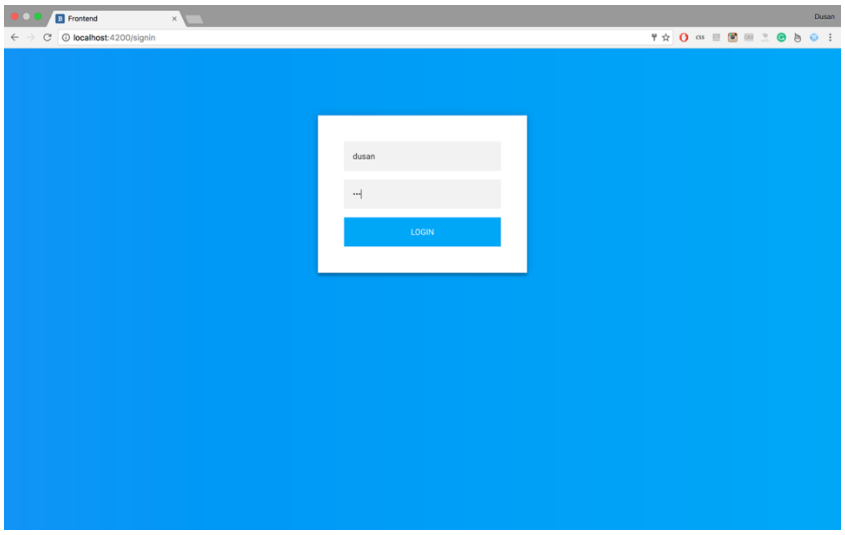
U terminalu ćemo izvršiti poziv aplikaciji ember (ember cli) sa parametrom build koja će generisati finalan produk front end aplikacije u dist folderu. Komanda koju želimo da izvršimo je “ember build”. Nakon ovoga moramo da napravimo servlet i kontroler za ovaj servlet. Servlet treba da učita html, javascript i css fileove iz generisanog ember dist foldera. Za ovakav deployment najbolje je razviti bash skriptu.



Za pokretanje aplikacije na lokalu potrebno je pokrenuti MySQL server i podesiti parametre za konekciju kroz xml konfiguraciju spring projekta. Pokrenuti kontejner sa mogućnostima za deployment Spring aplikacije (npr. Tomcat). Naredni korak je povući bower i npm zavisnosti ember projekta izvršavanjem “npm install && bower install” komandi. Poslednji korak je pokrenuti ember server komandom “ember server --port=http://localhost:8080/”. Aplikacija će se pokrenuti na portu 4200.

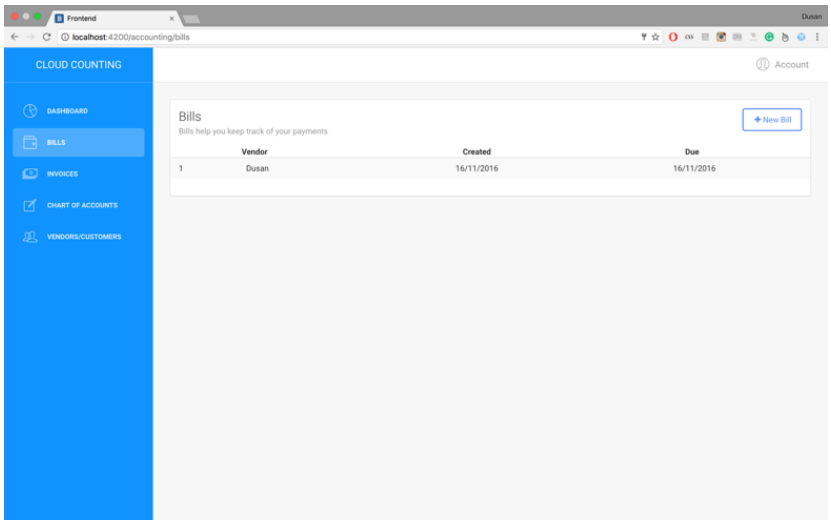
5. Ispunjenost funkcionalnih zahteva i realizacija aplikacije

Na slici 5.1 prestavljena je početna (log in) stranica aplikacije. Ova stranica omogućava korisniku da se uloguje i samim tim i autentifikuje. Ukoliko korisnik unese nevalidne podatke o tome će biti obavešten.



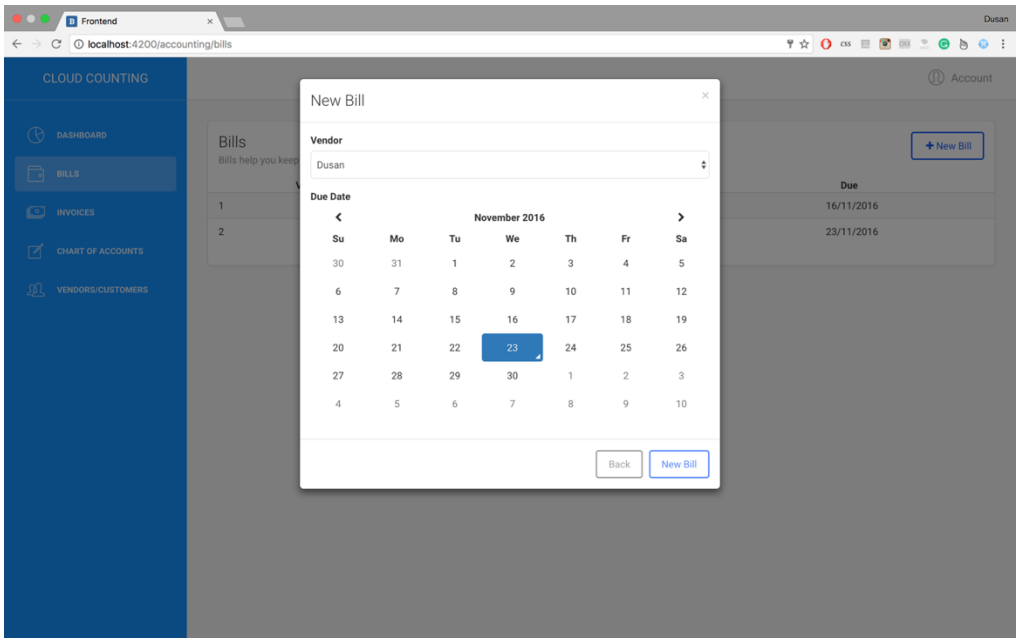
Slika 5.1 Početna strana Aplikacije

Na slici 5.2 prestavljena je Bill stranica aplikacije. Ova stranica omogućava korisniku da pregleda prethodno dodate Bill-ove. Ukoliko korisnik želi da doda novi Bill može to učiniti pritiskom dugmeta "New Bill".



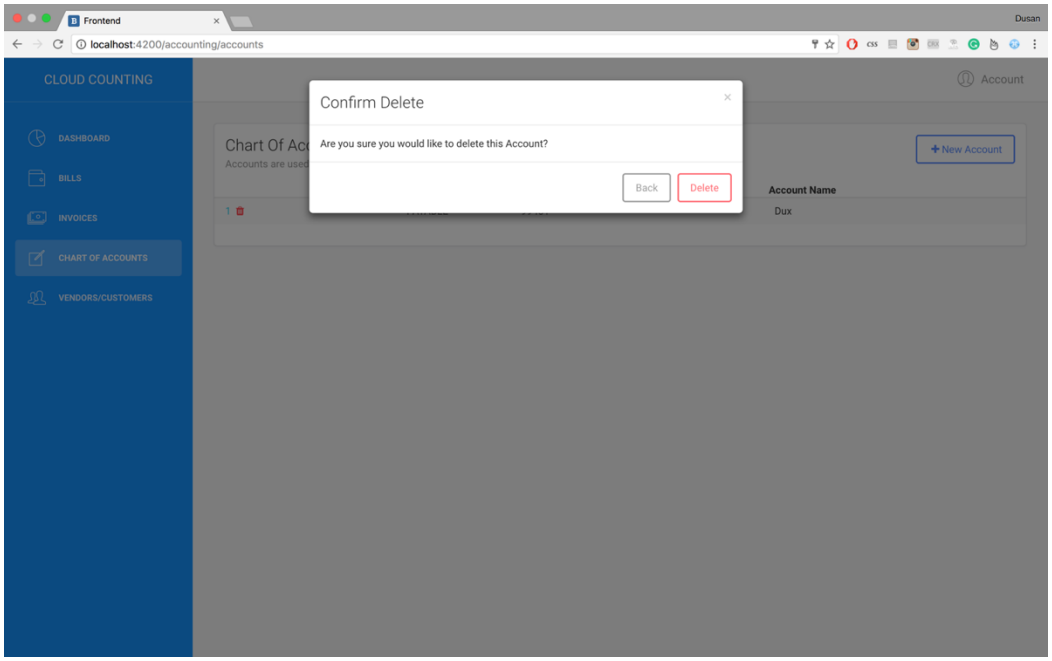
Slika 5.2 Bills strana Aplikacije

Na slici 5.3 prestavljena je Bill Show stranica aplikacije. Ova stranica omogućava korisniku da pregleda sve novčane alokacije Bill-a. Prikazan je primer modal-a koji korisniku omogućava dodavanje novog Bill-a.



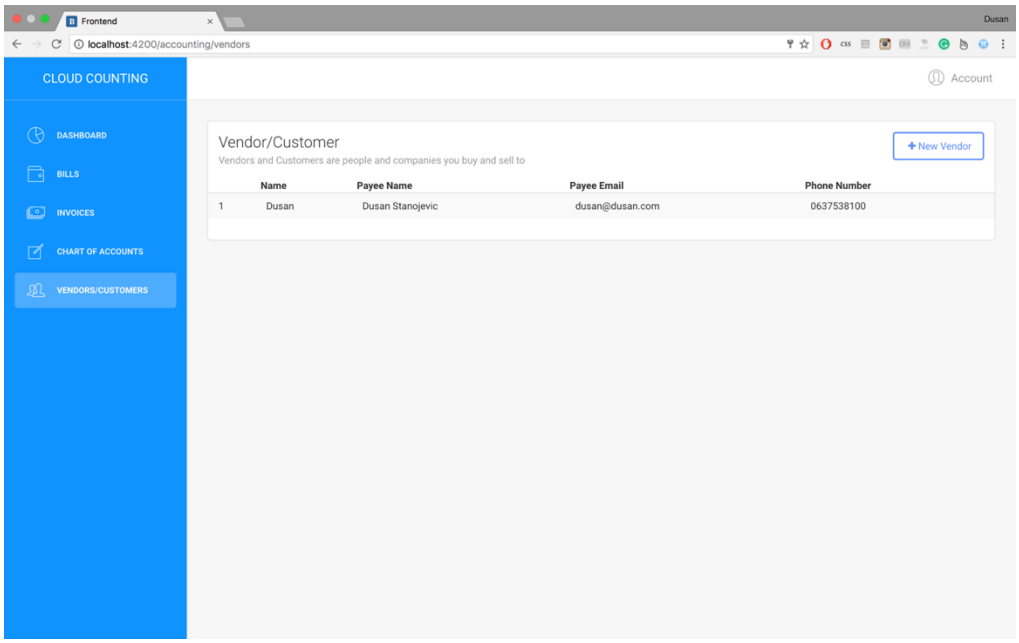
Slika 5.3 Bills strana Aplikacije sa otvorenim Modal-om

Na slici 5.4 prestavljena je Bill stranica aplikacije. Ova stranica omogućava korisniku da pregleda prethodno dodate Bill-ove. Ukoliko korisnik želi da doda novi Bill može to učiniti pritiskom dugmeta “New Bill”.



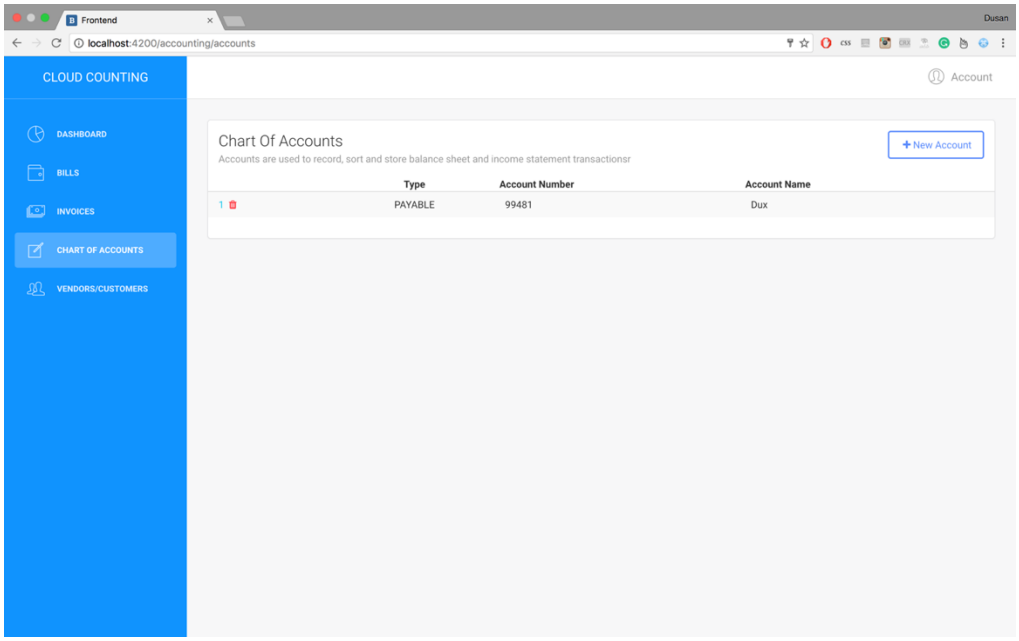
Slika 5.4 Accounts strana aplikacije sa Modal-om

Na slici 5.5 prestavljena je Vendor stranica aplikacije. Ova stranica omogućava korisniku da pregleda prethodno dodate Vendor-e. Ukoliko korisnik želi da doda novog Vendor-a može to učiniti pritiskom dugmeta “New Vendor”.



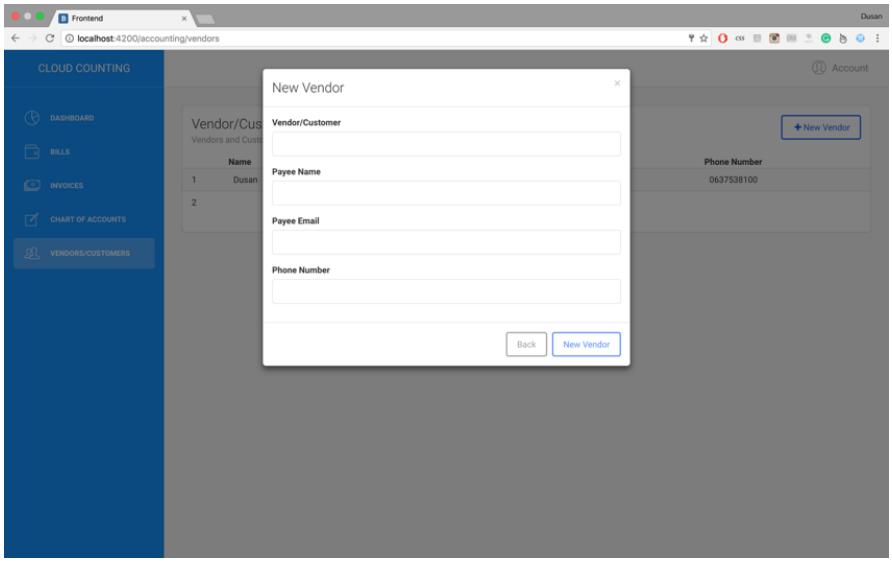
Slika 5.5 Vendors strana aplikacije

Na slici 5.6 prestavljena je Accounts stranica aplikacije. Ova stranica omogućava korisniku da pregleda prethodno dodate Account-e. Ukoliko korisnik želi da obriše account to može učiniti pritiskom kante na liniji Account-a.



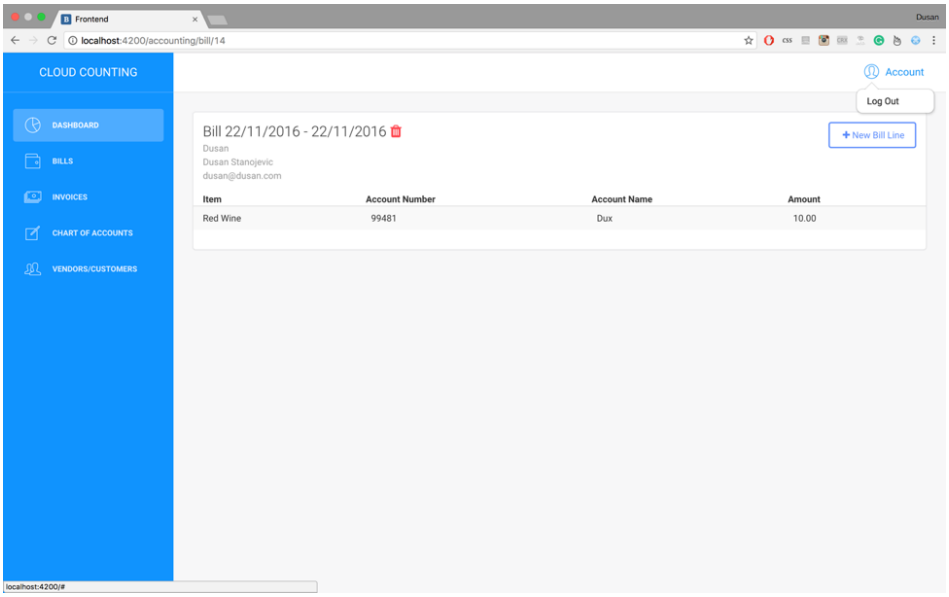
Slika 5.6 Accounts strana aplikacije

Na slici 5.7 prestavljena je Vendors stranica aplikacije sa otvorenim Modal-om za dodavnje Vendor-a.



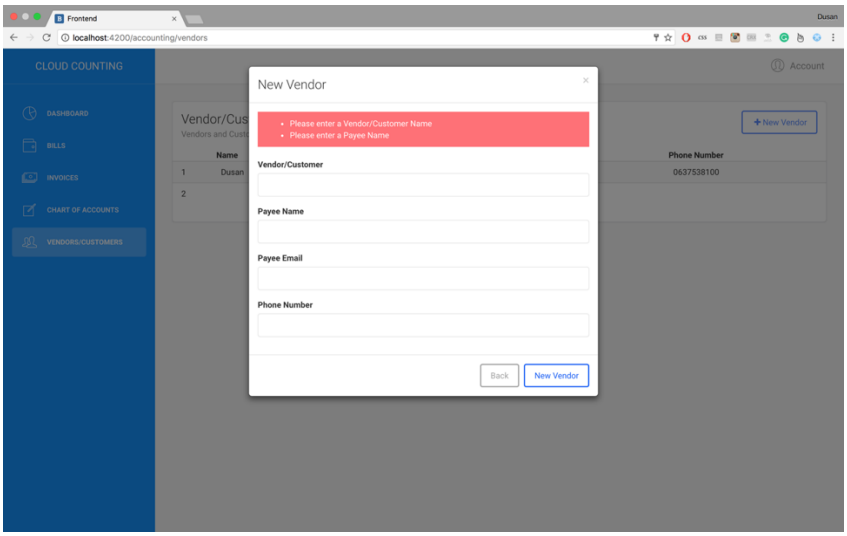
Slika 5.7 Vendors strana aplikacije sa Modal-om

Na slici 5.8 prestavljen je Pop-up koji se otvori pritiskom na Log Out dugme u gornjem desnom ćošku aplikacije.



Slika 5.8 Log out popup primer

Na slici 5.9 prestavljena je demonstracija validacije podataka prilikom dodaje novog Vendor-a.



Slika 5.9 Vendor strana aplikacije sa primerom validacije

5. Literatura

[1] Ember.js: Simple web app creation. Learn Ember.js in a DAY! - Todd Abel

[2] Ember.js cookbook Paperback – February 29, 2016 - Erik Hanchett

[3] Deliver Audacious Web Apps with Ember 2 1st Edition - Matthew White

[4] Ember.js Web Development with Ember CLI Paperback – May 27, 2015 - Suchit Puri

[5] JavaScript and JQuery: Interactive Front-End Web Development 1st Edition - Jon Duckett

[6] Web Design with HTML, CSS, JavaScript and jQuery Set 1st Edition - Jon Duckett

[7] Step By Step Bootstrap 3: A Quick Guide to Responsive Web Development Using Bootstrap - Riwanto Megosinarso

[8] Data-oriented Development with Angularjs Paperback – April 28, 2015 - Manoj Waikar

[9] Spring in Action: Covers Spring 4 4th Edition by Craig Walls

[10] Spring Microservices Paperback – June 28, 2016 - Rajesh RV

[11] Java Network Programming Fourth Edition Edition - Elliotte Rusty Harold

[12] Head First Design Patterns: A Brain-Friendly Guide 1st Edition - Eric Freeman, Bert Bates, Kathy Sierra, Elisabeth Robson

[13] Patterns of Enterprise Application Architecture 1st Edition - Martin Fowler