



PREDMET

IT320 Savremene tehnološke platforme

Prolećni semestar 2016/2017

PROJEKAT

METCHAT

VERZIJA 1.0

AUTOR

MILOŠ MENIĆANIN

Fakultet Informatičkih tehnologija

milos.menicanin.2119@metropolitan.ac.rs

Beograd, 2017

MetChat

Zahtevi klijenta za softver

Za potrebe studenata Metropolitan Univerziteta, treba napraviti sistem za razmenu poruka u realnom vremenu. Sistem treba biti dostupan na različitim uredjajima (desktop, tablet, smartfon).

Funkcionalni zahtevi

Klijentski deo

Klijenti MetChat sistema su studenti koji žele da se dopisuju sa drugim studentima Metropolitan Univerziteta. Klijent se registruje preko sajta ili moze da bude registrovan od strane administratora, gde će mu automatski stići automatski generisana šifra na metropolitan e-mail nalog.

Registracija

Kada student prvi put pristupa sistemu i ukoliko nije već registrovan od strane administratora, student prilikom registracije na početnoj stranici mora da unese sledeće podatke o sebi: korisničko ime, email (@metropolitan.ac.rs), i lozinku.

Prijava

Ukoliko je student registrovan na jedan od prethodno spomenutih načina, student se prijavljuje sa svojim email-om i lozinkom, koju je uneo prilikom registracije ili koja mu je dodeljena od strane administratora.

Dopisivanje

Kada se student prijavi, automatski se usmerava na drugu stranicu na kojoj se nalaze lista soba, lista online studenata, polje u kome se vidi prethodna konverzacija kao i polje za unos nove poruke.

IT320 Savremene Tehnološke Platforme - Projektni Zadatak

Pravljenje soba

Student ima mogućnost da napravi novu sobu (kanal) za komunikaciju sa ostalim studentima (soba može biti na primer šifra predmeta IT320).

Lista korisnika

Student ima mogućnost da vidi ko je online od registrovanih korisnika u sobi.

Promena korisničkog imena

Student ima mogućnost da promeni svoje korisničko ime koje je vidljivo tokom konverzacije sa ostalim studentima u nekoj sobi u datom trenutku.

Odjava

Student mora da ima opciju da se odjavi sa sajta.

Administratorski deo

Administratori sistema imaju mogućnost izmene, unosa i brisanja korisnika i soba sa sistema kao i uvid koliko ima registrovanih korisnika i kreiranih soba.

Unos korisnika

Student ima mogućnost da napravi novu sobu (kanal) za komunikaciju sa ostalim studentima (soba može biti na primer naziv predmeta IT320).

Izmene korisnika

Administrator može da promeni lozinku korisnika.

Brisanje soba

Mogućnost brisanja kreiranih soba.

Statistika

Mogućnost uvida u broj registrovanih korisnika i napravljenih soba.

Dizajn

Lako korišćenje na mobilnim uredjajima

Potrebno je da sistem bude intuitivan za korišćenje i na mobilnim uređajima preko web čitača mobilnog telefona primenom tehnika prilagodljivog dizajna (engl. Responsive web design).

Typefaces

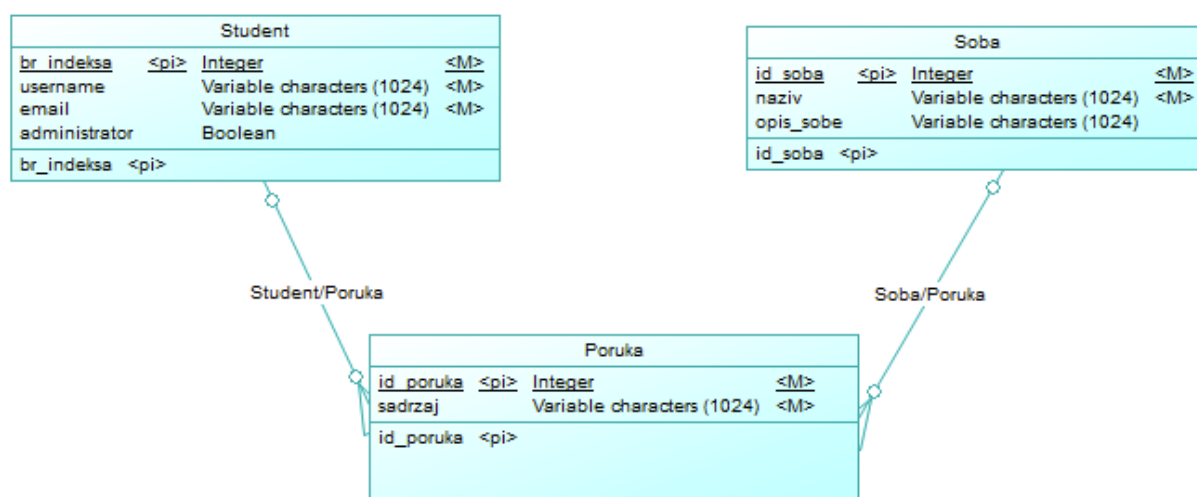
MetChat će koristiti sledeće fontove: **Lato**, **Roboto**.

Paleta Boja

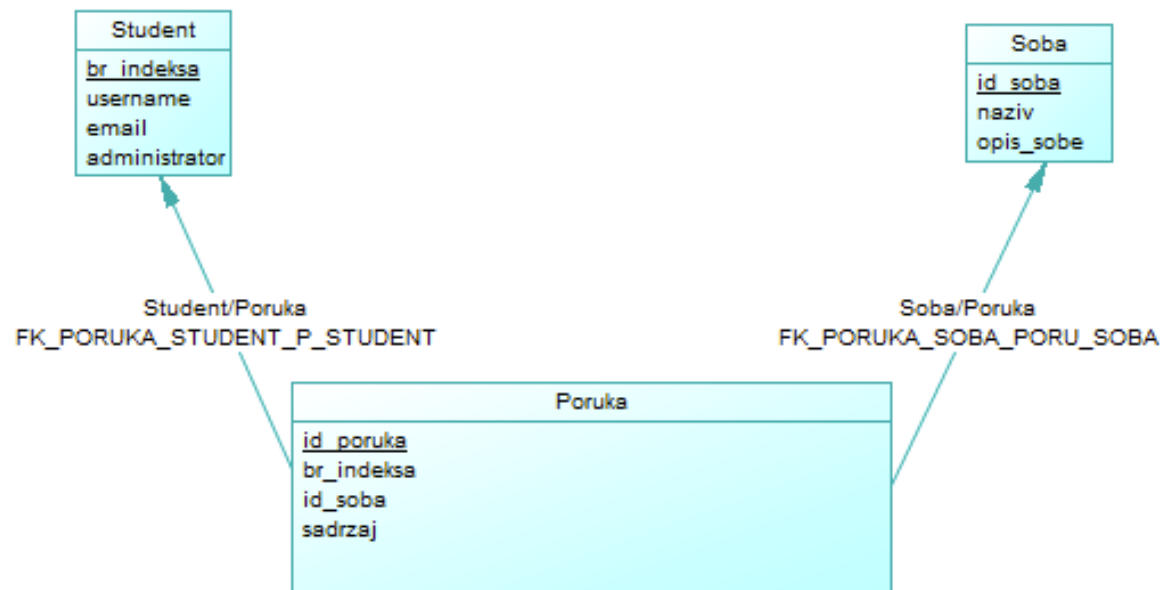


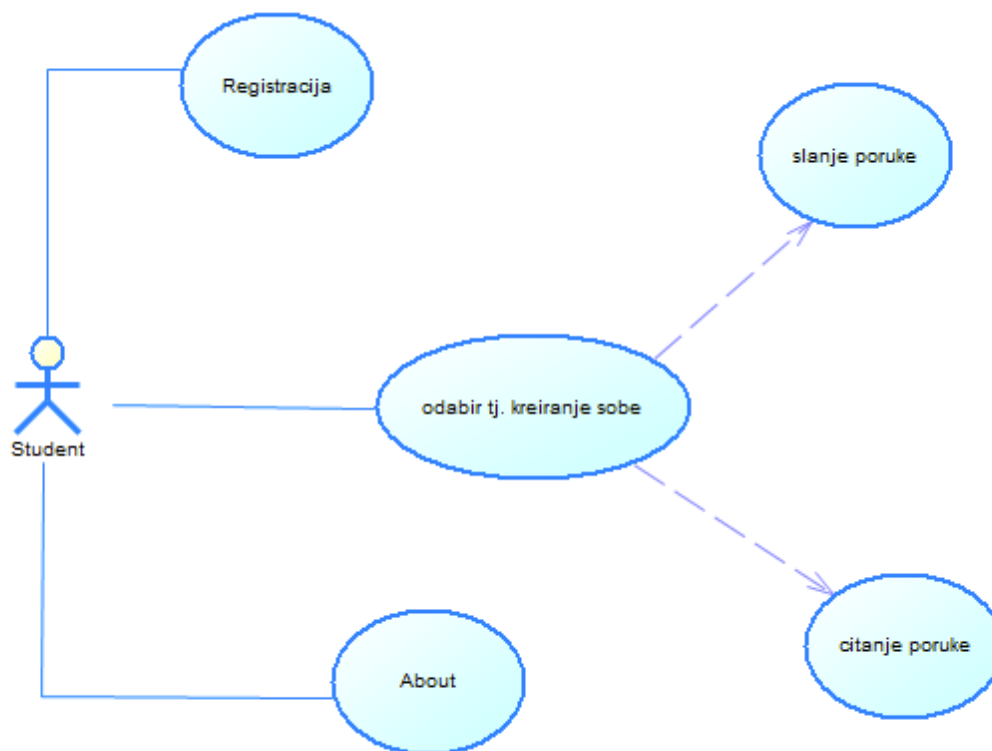
Tehničke specifikacije

Konceptualni model baze podataka



Fizički model baze podataka





Use case dijagram studenta

UC01: Registracija preko sajta (Učesnik: Student)

| Preduslov | Koraci | Šta je izvršeno |
|---|--|----------------------------------|
| Student mora biti na MetChat web aplikaciji | Student ulazi na stranu za registraciju—pocetna strana | Student je registrovan na sistem |
| Student mora da ima studentski email nalog. | Student unosi svoj studentski email u odgovorajuće polje | |

UC02: Kreiranje sobe (Učesnici: Student i Administrator)

| Preduslov | Koraci | Šta je izvršeno |
|-----------|--------|-----------------|
|-----------|--------|-----------------|

IT320 Savremene Tehnološke Platforme - Projektni Zadatak

| | | |
|---|---|---------------------------------|
| Učesnik mora biti registrovan na sistem | Učesnik klikce dugme za kreiranje nove sobe i daje naziv sobe | Učesnik je uspesno kreirao sobu |
|---|---|---------------------------------|

UC03: Brisanje sobe (Učesnici: Student i Administrator)

| Preduslov | Koraci | Šta je izvršeno |
|---|-------------------------------|---------------------------------------|
| Učesnik mora biti registrovan na sistem | Učesnik bira sobu za brisanje | Soba je obeležena i spremna za izmenu |
| Učesnik mora da je vec kreirao sobu ili da ima administratorske privilegije | Učesnik potvrđuje brisanje | Učesnik je uspesno izbrisao sobu |

UC04: Menjanje naziva sobe (Učesnici: Student i Administrator)

| Preduslov | Koraci | Šta je izvršeno |
|---|--------------------------------------|--|
| Učesnik mora biti registrovan na sistem | Učesnik bira sobu za menjanje naziva | Soba je obeležena |
| Učesnik mora da je vec kreirao sobu ili da ima administratorske privilegije | Učesnik upisuje novi naziv sobe | Učesnik je uspešno promenio naziv sobe |

UC05: Brisanje studenta (Učesnik: Administrator)

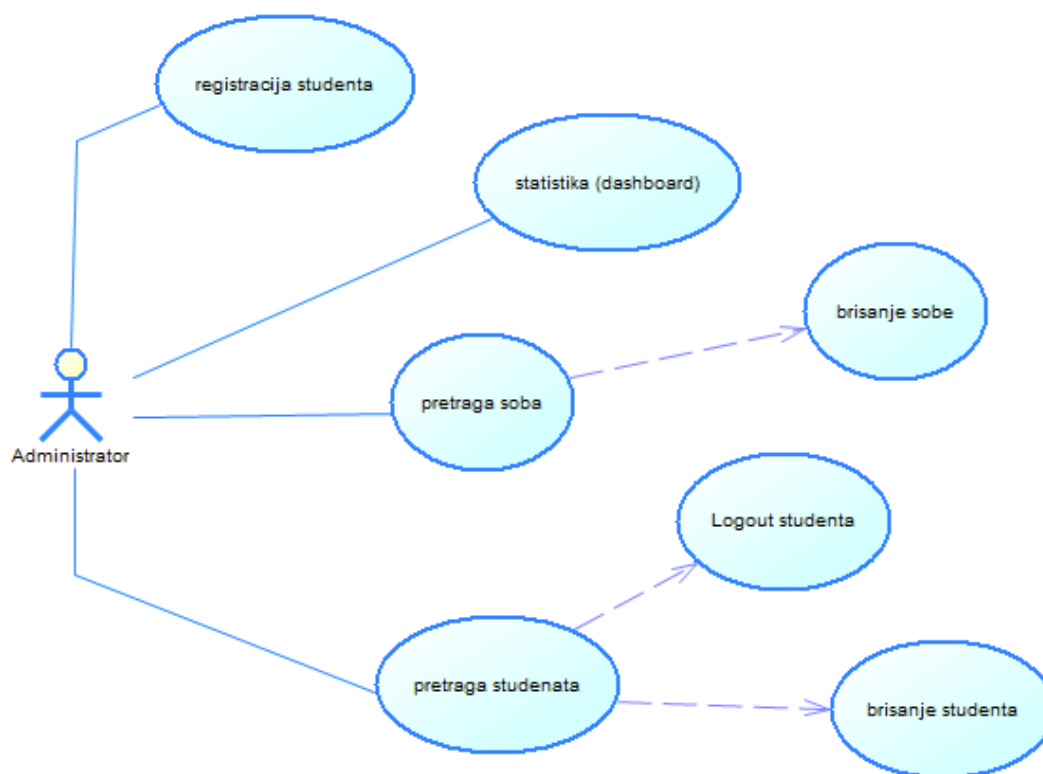
| Preduslov | Koraci | Šta je izvršeno |
|---|---|--|
| Administrator mora biti registrovan na sistem | Administrator bira studenta za brisanje | Administrator je uspešno izbrisao studenta |

UC06: Slanje poruka (Učesnici: Student i Administrator)

| Preduslov | Koraci | Šta je izvršeno |
|---|--|-----------------------------------|
| Učesnik mora biti registrovan na sistem | Učesnik bira sobu gde ce da posalje poruku | Učesnik je uspešno pristupio sobi |
| Učesnik mora da je vec pristupio sobi | Učesnik ispisuje novu poruku | Učesnik je uspešno poslao poruku |

UC07: Brisanje poruka (Učesnici: Student i Administrator)

| Preduslov | Koraci | Šta je izvršeno |
|---|---|------------------------------------|
| Učesnik mora biti registrovan na sistem | Učesnik bira sobu gde ce da pregleda poruke | Učesnik je uspešno pristupio sobi |
| Učesnik mora da je vec napisao poruku ili da ima adminsitratorske privilegije | Učesnik bira poruku koju želi da obriše | Učesnik je uspešno izbrisao poruku |



Use case dijagram administratora

| Preduslov | Koraci | Šta je izvršeno |
|--|--|--|
| Učesnik mora biti registrovan na sistem | Učesnik pretražuje korisnika koga želi da obriše | Osnovni podaci korisnika se prikazuje na ekranu |
| Učesnik mora da ima adminsitratorske privilegije | Administrator bira opciju za brisanje korisnika | Administrator je uspešno izbrisao korisnika sa sistema |

UC08: Brisanje korisnika (Učesnici: Administrator)

UC09: Unos korisnika (Učesnici: Administrator)

| Preduslov | Koraci | Šta je izvršeno |
|--|--|-------------------------------------|
| Učesnik mora biti registrovan na sistem | | |
| Učesnik mora da ima adminsitratorske privilegije | Administrator bira opciju za dodavanje korisnika i popunjava formu | Korisnik je uspešno dodat na sistem |

IT320 Savremene Tehnološke Platforme - Projektni Zadatak

UC10: Brisanje poruka (Učesnici: Student i Administrator)

| Preduslov | Koraci | Šta je izvršeno |
|--|---|----------------------------|
| Učesnik mora biti registrovan na sistem | Učesnik bira neku od svojih poruka koju želi da obriše | Poruka za uspešno obrisana |
| Učesnik mora da ima adminsitratorske privilegije | Administrator bira bilo koju poruku koju želi da obriše | Poruka za uspešno obrisana |

UC11: Logout korisnika (Učesnici: Student i Administrator)

| Preduslov | Koraci | Šta je izvršeno |
|--|--|------------------------------|
| Učesnik mora biti registrovan na sistem | Učesnik bira korisnika koga želi da izloguje | |
| Učesnik mora da ima adminsitratorske privilegije | Administrator bira opciju da se izloguje učesnik | Učesnik je uspešno izlogovan |

Tehnički opis funkcionalnosti

Administratorski deo

Izmene korisnika

Kod izmene korisnika, administratoru je prvo potrebno prikazati listu svih korisnika. Na ovoh strani potrebno je da postoji i pretraga korisnika. Sa ovog pogleda administrator treba da ima opciju da obriše, doda ili izmeni postojećeg korisnika kao i mogućnost da datog korisnika izloguje sa sistema.

Izmene soba

Kod izmene soba, administratoru je prvo potrebno prikazati listu svih soba. Na ovoj strani potrebno je da postoji i pretraga soba. Sa ovog pogleda administrator treba da ima opciju da obriše, doda ili izmeni postojeću soba.

Statistika

Na ovoj stranici administrator ima uvid broj registrovanih korisnika, broj kreiranih soba, broj korisnika u svakoj sobi kao i broj online korisnika. Svi brojevi i podaci trebaju biti i adekvatno grafički prikazani.

Korisnički deo

Registracija

Kada korisnik pristupi web aplikaciji MetChat, potrebno je da mu registracija bude na najviše jedan korak (Dugme registracija). Kada korisnik otvori opciju registracije treba mu dati prostu formu za registraciju. Nakon registracije korisnik treba da je automatski ulogovan na sistem.

Prijava

Prijava bi trebala da se nalazi na početnoj strani MetChat sistema. Korisnik se prijavljuje svojim mejlom i lozinkom koju je upisao prilikom registracije ili koja mu je dodeljena putem email-a.

Wireframe

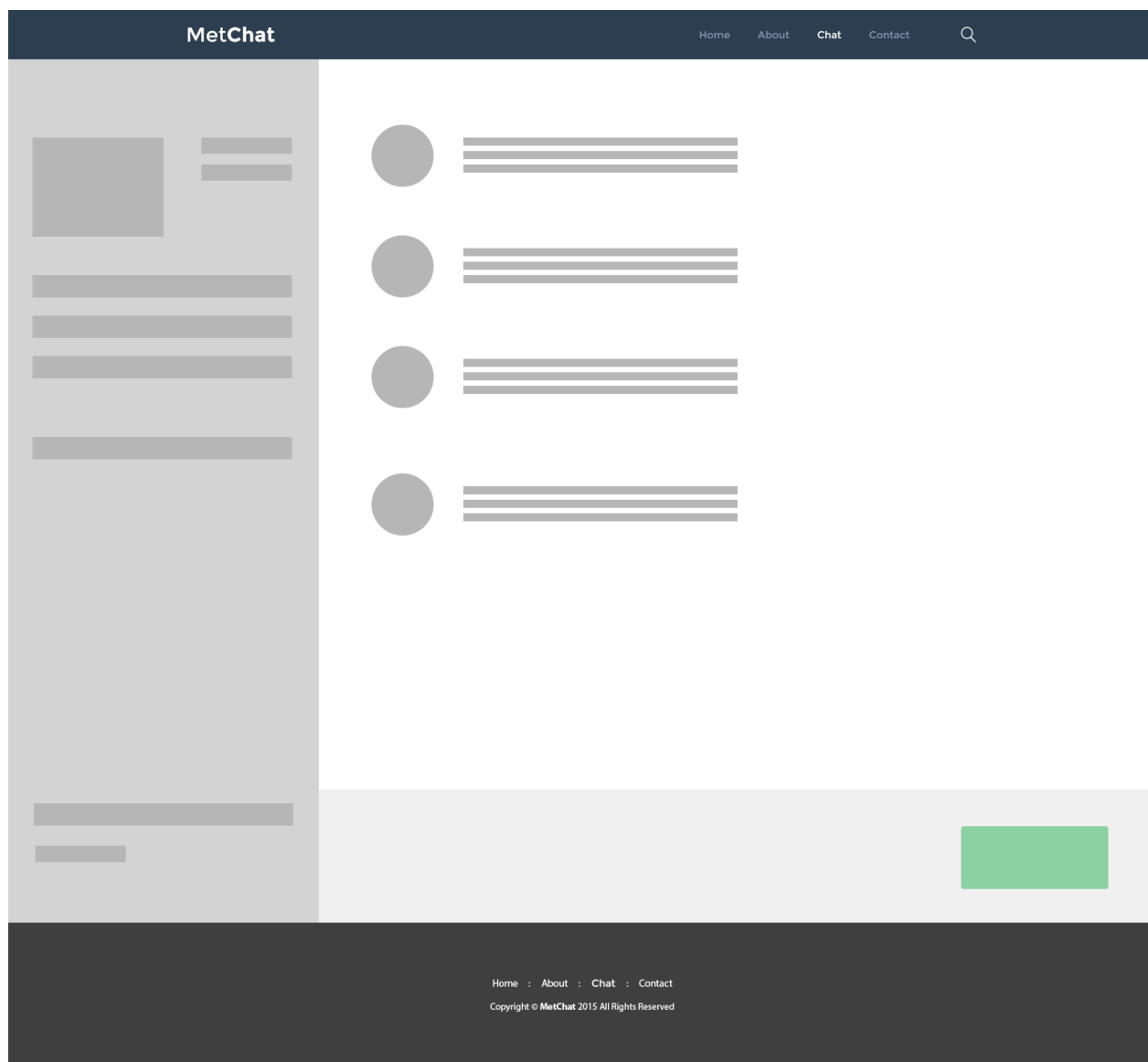
Mapa Sajta



Home stranica



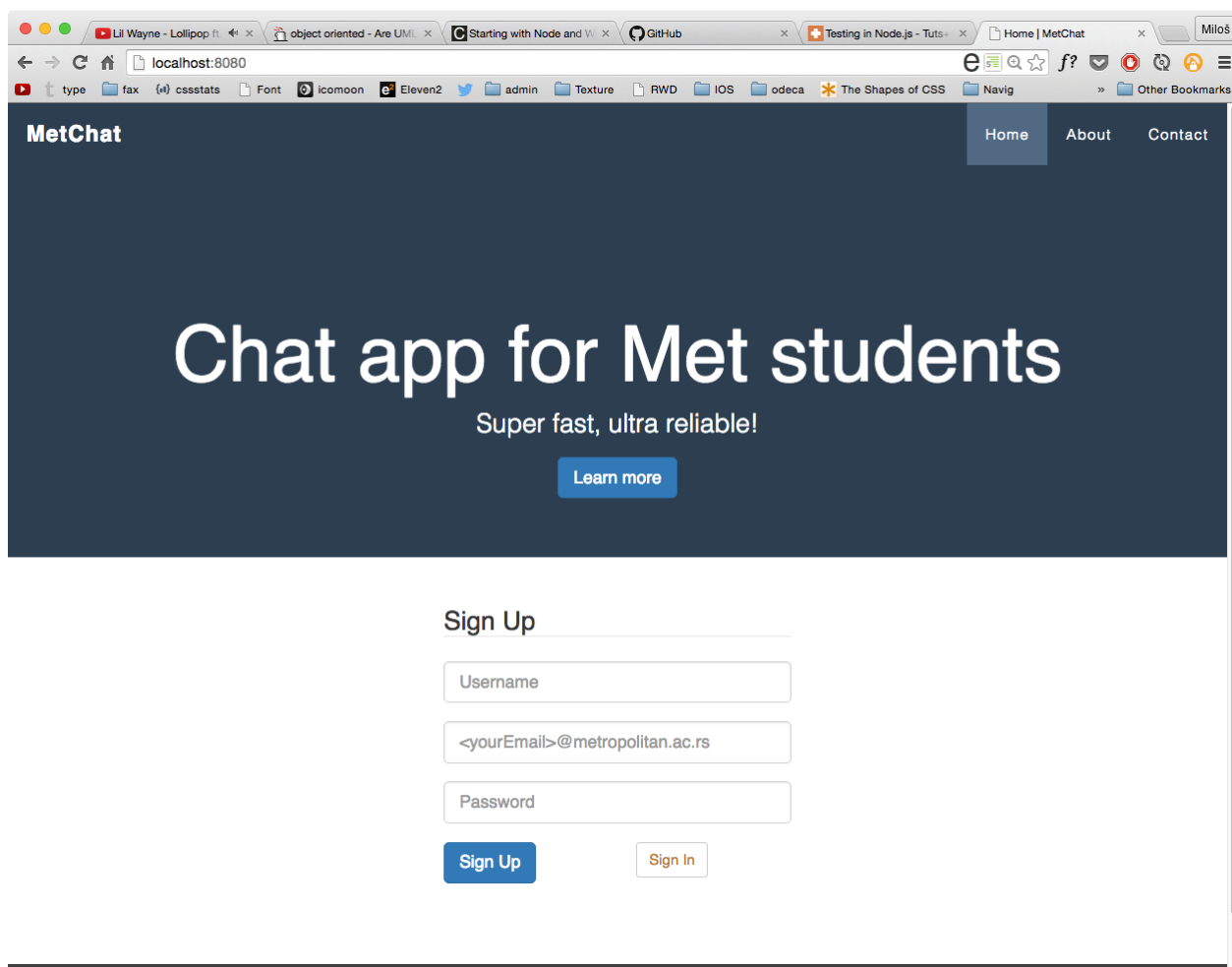
Chat stranica



Projektovanje Sistema - Detaljni projekat

MetChat aplikacija je izradjena koristeći Node.js platformu i Express framework. Node.js je platforma izrađena na JavaScript izvršnom okruženju Google Chrome-a za jednostavnu izradu brzih, skalabilnih mrežnih aplikacija. Koristi vođen-događajima (event-driven), neblokirajući U/I model koji ga čini laganim i efikasnim, savršenim za aplikacije koje rade u realnom vremenu i iziskuju veće količine prenosa podataka i koje se izvršavaju preko distribuiranih uređaja.

Aplikacija je namenjena studentima Metropolitan Univerziteta za razmenu poruka u realnom vremenu. U nastavku će biti opisane sve stranice na sajtu.

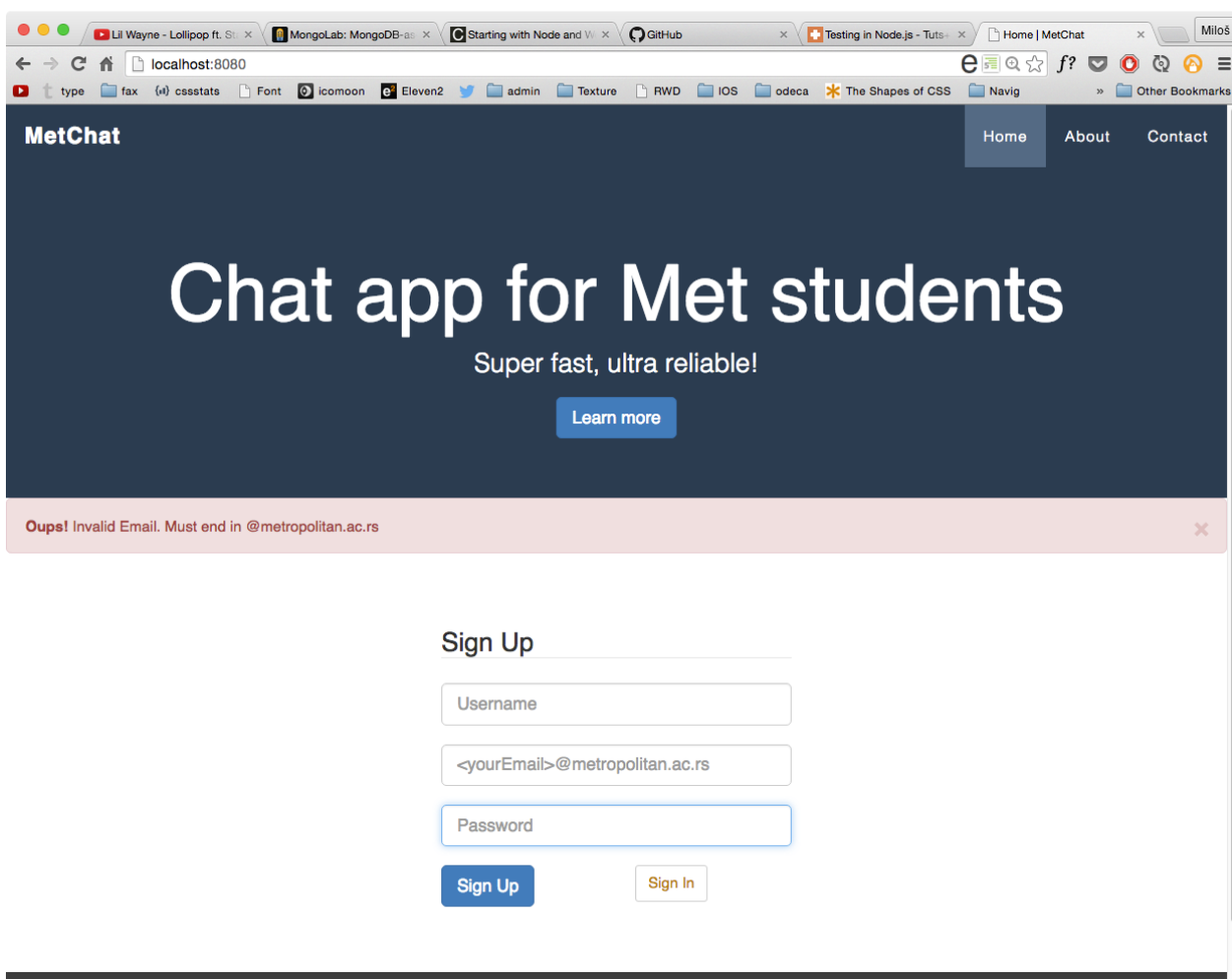


Slika 1: Početna stranica sa formama za registraciju i prijavu korisnika.

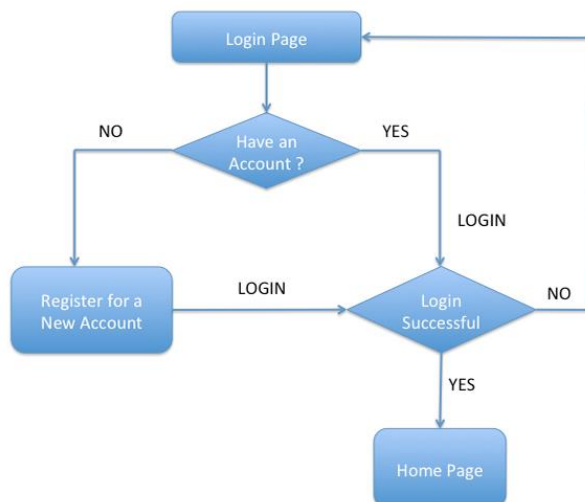
Početna stranica

Na početnoj stranici se nalaze linkovi ka ostalim stranicama sajta kome mogu pristupiti svi korisnici sajta. Te stranice su Home, About i Contact. Zatim se nalazi forma za registraciju i login korisnika. Kako bi se novi korisnik registrovao na sajt, mora da popuni formu svojim podacima (username, email i password). Korisnik mora da unese metropolitan email adresu inace nece moći da se registruje. Ukoliko korisnik ne unese ispravnu email adresu ili ako korisnik sa unetim username-om već postoji u bazi podataka, korisnik neće biti registrovan i prikazuje mu se greška. Ukoliko je uneo ispravne podatke u formu, novi nalog se kreira u bazi i korisnik se preusmerava na *chat* stranicu.

IT320 Savremene Tehnološke Platforme - Projektni Zadatak



Slika 2: Pogrešno unet email prilikom registracije



IT320 Savremene Tehnološke Platforme - Projektni Zadatak

Ukoliko korisnik ima već kreiran nalog, on onda mora da se prijavi na sajt kako bi mogao da pristupi sadržaju sajta samo za autorizovane korisnike. Korisnik popunjava formu za login tako što unosi svoj username i lozinku. Ukoliko ne unese ispravne podatke, ispisuje se greška. Ukoliko je korisnik uneo ispravne podatke u Login formu, on se preusmerava na *chat* stranicu i ima pristup sadržaju sajta samo za autorizovane korisnike.

Kod za Login je sličan kao i za registraciju

```
57      /* Handle Registration POST */
58      router.post('/signup', passport.authenticate('signup', {
59          successRedirect: '/chat',
60          failureRedirect: '/',
61          failureFlash : true
62      }));
63
```

Slika 3-1: Preusmeravanje korisnika na chat stranicu nakon uspešne registracije ili na root stranicu ukoliko je registracije neuspešna.

```
1  var LocalStrategy = require('passport-local').Strategy;
2  var User = require('../models/user');
3  var bcrypt = require('bcrypt-nodejs');
4
5  module.exports = function(passport){
6
7      passport.use('signup', new LocalStrategy({
8          passReqToCallback : true, // allows us to pass back the entire request to the callback
9      },
10     function(req, username, password, done) {
11
12         findOrCreateUser = function(){
13             // find a user in Mongo with provided username
14             User.findOne({ 'username' : username}, function(err, user) {
15                 // In case of any error, return using the done method
16                 if (err){
17                     console.log('Error in SignUp: '+err);
18                     return done(err);
19                 }
20                 // already exists
21                 if (user) {
22                     console.log('User already exists with username: '+ username);
23                     return done(null, false, req.flash('message', 'User Already Exists'));
24                 }
25                 // If email doesn't end in metropolitan.ac.rs
26                 if (!isValidEmail(req.param('email'))){
27                     console.log('Invalid Email');
28                     return done(null, false, req.flash('message', 'Invalid Email. Must end in @metropolitan.ac.rs')); //
29                     redirect back to login page
30                 }
31                 else {
32                     // if there is no user with that email
33                     // create the user
34                     var newUser = new User();
35
36                     // set the user's local credentials
```

Slika 3: Strategija za registraciju novih korisnika i pravljenje dokumenata u Mongo bazi podataka.

IT320 Savremene Tehnološke Platforme - Projektni Zadatak

Ovde treba obratiti pažnju na metod `findOne` koji kao parametar uzima `username` iz forme i upoređuje ga sa `username`-ima iz Mongo baze. Ukoliko ne nadje `username` u bazi, izbacuje grešku 'User Not found'. Ukoliko nadje, zatim proverava lozinku tj. upoređuje da li je lozinka iz forme ista kao i lozinka korisnika u bazi podataka. Ukoliko je i lozinka ista, funkcija done vraća korisnika.

Funkcija `isValidPassword` f-ja upoređuje hash lozinke iz baze sa lozinkom(plain-text) iz req objekta.

A screenshot of a 'Sign In' form. The form has a title 'Sign In' in the top left corner and a close button 'x' in the top right corner. Below the title, there are two input fields: 'Username' and 'Password'. Below the 'Password' field, there is a blue button labeled 'Login'. The form is enclosed in a thin black border.

Slika 4: Forma za prijavu korisnika

```
1 var LocalStrategy = require('passport-local').Strategy;
2 var User = require('../models/user');
3 var bcrypt = require('bcrypt-nodejs');
4
5 module.exports = function(passport){
6
7   passport.use('login', new LocalStrategy({
8     passReqToCallback : true
9   },
10    function(req, username, password, done) {
11      // check in mongo if a user with username exists or not
12      User.findOne({ 'username' : username },
13        function(err, user) {
14          // In case of any error, return using the done method
15          if (err)
16            return done(err);
17          // Username does not exist, log the error and redirect back
18          if (!user){
19            console.log('User Not Found with username ' + username);
20            return done(null, false, req.flash('message', 'User Not found.'));
21          }
22          // User exists but wrong password, log the error
23          if (!isValidPassword(user, password)){
24            console.log('Invalid Password');
25            return done(null, false, req.flash('message', 'Invalid Password')); // redirect back to login pa
26          }
27          // User and password both match, return user from done method
28          // which will be treated like success
29          console.log('sessionIdFromReq', req.sessionID);
30          // req.user = user;
31          return done(null, user);
32        }
33      );
34    }
35  })
36  );
```

Slika 4-1: Strategija za prijavu korisnika na sajt.

Ukoliko dodje do greške, ispisuje se poruka (req.flash.message) 'Wrong Password'.

```
extends layout
block content
  #wrap
    .jumbotron
      .container
        h1.text-center Chat app for Met students
        p.text-center Super fast, ultra reliable!
        .text-center.center-block
          a.btn.btn-primary.btn-primary(href='#', role='button') Learn more
    -if (message != '')
      #message.alert.alert-danger.flash.fade.in.alert-dismissible(role="alert")
        button.close(type='button', data-dismiss='alert', aria-label='Close')
        span(aria-hidden='true') ×
        strong Oops!
        span= message
    .container
      // h1= title
      // p Welcome to #{title}
    .col-md-4.col-md-offset-4
      form.form-horizontal#form-Reg(action='/signup', method='POST')
        fieldset
          // Form Name
          legend Sign Up
          // Text input
          .form-group
            // label.col-md-4.control-label(for='username')
            .col-md-12
              input#username.form-control.input-md(name='username', type='text', placeholder='Username', required)
            // span.help-block Type your Username
          // Email input
          .form-group
            // label.col-md-4.control-label(for='email')
            .col-md-12
```

Slika 4-2: Prikaz greske u index.jade datoteci ukoliko message promenljiva nije prazna.

Na slici 4.2 se može videti primer embedovanog js-a u ¹Jade datoteci.

Zatim na slici 4.3 je prikazano rutiranje (Express) tj. šta se dešava kada korisnik otvori index stranicu uz pomoc GET metode. Ako je req.user objekat definisan, to jest ako je korisnik ulogovan, render-uje (prikazuje) se index.jade fajl i toj datoteci se prosledjuju *title*, *message* i *useri* promenljive. Ukoliko korisnik nije ulogovan renderuje se istra stranica ali bez *message* promenljive.

```
/* GET Index page. */
router.get('/', function(req, res) {
  // Ako je sesija otvorena
  if (req.user) {
    res.render('index', {
      title: 'Home',
      message: req.flash('message'),
      useri: req.user.username
    });
  } else {
    res.render('index', {
      title: 'Home',
      message: req.flash('message')
    });
  }
});
```

Slika 4-3: Prikaz GET metod rute tj. način prikazivanja stranica.

About stranica

¹ Jade je template-ing engine za XML dokumente kao što je HTML.

About stranica je statična stranica na kojoj se nalazi tekst o funkcijama aplikacije.

About

MetChat web aplikacija služi za potrebe studente Metropolitan Univerziteta, predstavlja sistem za razmenu poruka u realnom vremenu. Sistem je dostupan na različitim uređajima (desktop, tablet, smartfon).

Klijentski Deo

Klijenti MetChat sistema su studenti koji žele da se dopisuju sa drugim studentima Metropolitan Univerziteta. Klijent se registruje preko sajta ili može da bude registrovan od strane administratora, gde će mu automatski stići automatski generisana šifra na met-ropolitan e-mail nalog.

Registracija

Kada student prvi put pristupa sistemu i ukoliko nije već registrovan od strane administratora, student prilikom registracije na početnoj stranici mora da unese sledeće podatke o sebi: korisničko ime, email (@metropolitan.ac.rs), i lozinku.

Prijava

Ukoliko je student registrovan na jedan od prethodno spomenutih načina, student se prijavljuje sa svojim username-om i lozinkom, koju je uneo prilikom registracije ili koja mu je dođeljena od strane administratora.

Dopisivanje

Kada se student prijavi, automatski se usmerava na drugu stranicu na kojoj se nalaze lista soba, lista online studenata, polje u kome se vidi prethodna konverzacija kao i polje za unos nove poruke.

Kreiranje Sobe

Prijavljeni korisnik ima mogućnost da kreira novu sobu (kanal) za komunikaciju sa ostalim studentima (soba može biti na primer šifra predmeta IT320).

Lista Korisnika

Student ima mogućnost da vidi ko je online od registrovanih korisnika u sobi.

Odjava sa Sajta

Student mora da ima opciju da se odjavi sa sajta.

Izmene Sobe

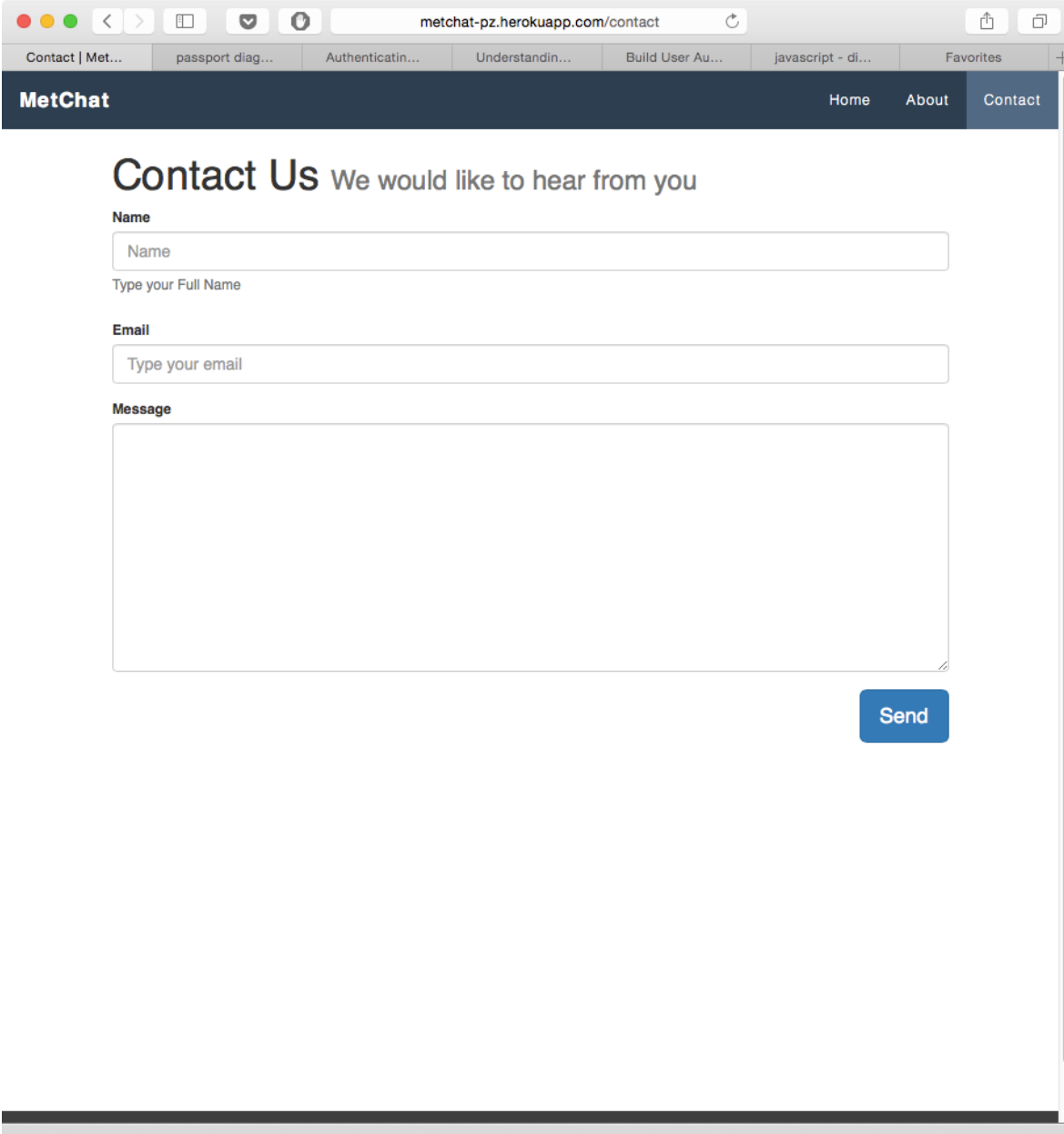
Kod izmene soba, prijavljenom korisniku je prvo potrebno prikazati listu svih soba. Na ovoj strani potrebno je da postoji i pretraga soba. Sa ovog pogleda administrator treba da ima opciju da izmeni postojeću sobu.

Copyright © MetChat 2017. All Rights Reserved.

Slika 4: Prikaz About stranice.

Contact Stranica

Na ovoj stranici se nalazi kontakt forma za slanje poruke administratoru sajta.



The screenshot shows a web browser window with the address bar displaying 'metchat-pz.herokuapp.com/contact'. The browser's tab bar shows several tabs, with 'Contact | Met...' being the active one. The website's header is dark blue with the 'MetChat' logo on the left and navigation links 'Home', 'About', and 'Contact' on the right. The 'Contact' link is highlighted. The main content area has a heading 'Contact Us' followed by the text 'We would like to hear from you'. Below this, there are three input fields: 'Name' (with placeholder text 'Name'), 'Email' (with placeholder text 'Type your email'), and 'Message' (a larger text area). A blue 'Send' button is positioned to the right of the message input field. The browser's address bar and tab bar are visible at the top, and a vertical scrollbar is on the right side of the page.

Slika 5: Prikaz Contact stranice.

Polja za unos su Name, Email i Message i sva moraju biti popunjena. Postoji front-end validacija forme za ispravan unet email kao i ako su polja prazna. Nakon klika dugmet "Send", pojavljuje se baner koji prikazuje da li je poruka uspesno poslata ili ne. Ako nije prikazuje grešku.

IT320 Savremene Tehnološke Platforme - Projektni Zadatak

Na slici 5-2 se može videti način konektovanja na server uz pomoć *nodemailer* modula. Nakon klika “Send” dugmeta aktivira se `router.post`. Transporter funkcija (objekat) služi za slanje mail-a i prima parametar koji je transportni mehanizam (SMTP konfiguracija). Nakon toga se poziva `sendMail` properti funkcija transporter objekta i prosledjuju mu se parametri `mailOptions` i callback f-ja. Na kraju se preusmerava na `users` stranicu.

Chat Stranica

Chat stranica je glavna stranica sajta. Njoj mogu pristupiti samo autorizovani korisnici.

```
router.post('/contact', function(req, res) {
  var transporter = nodemailer.createTransport({
    service: 'Gmail',
    auth: {
      user: 'milos.it320@gmail.com',
      pass: 'it320milos'
    }
  });

  // NB! No need to recreate the transporter object. You can use
  // the same transporter object for all e-mails

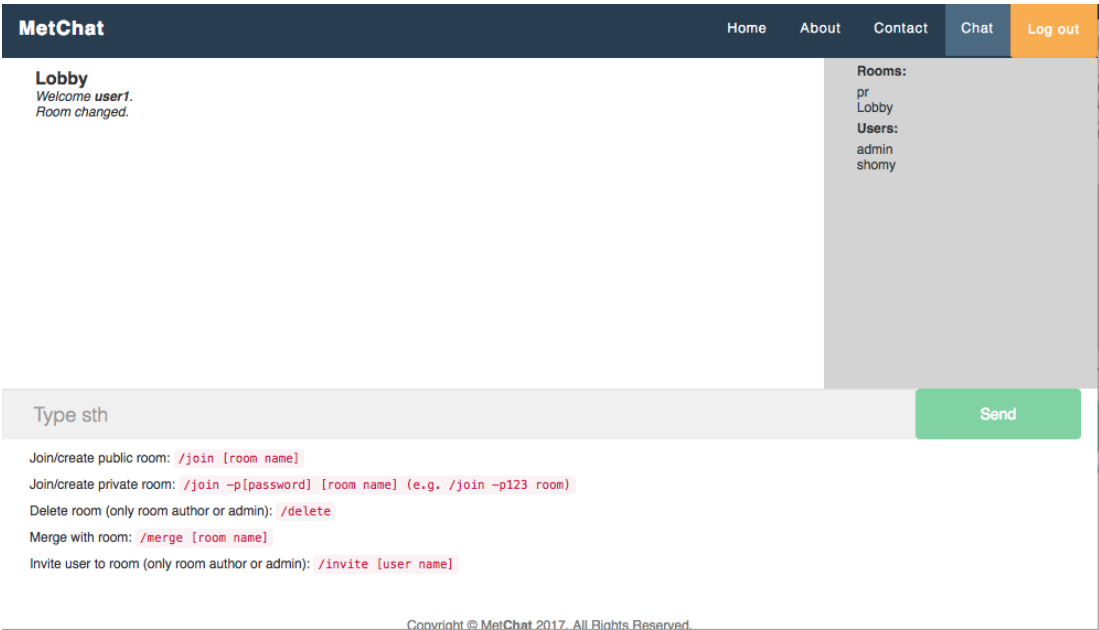
  // setup e-mail data with unicode symbols
  var mailOptions = {
    from: req.body.name + ' <' + req.body.email + '>', // sender address
    to: 'milos.menicanin.2119@metropolitan.ac.rs',
    subject: 'Hello 🍌', // Subject line
    text: req.body.message + '🍌', // plaintext body
    // html: '<b>Hello world 🍌</b>' // html body
  };

  // send mail with defined transport object
  transporter.sendMail(mailOptions, function(error, info){
    var msg;
    if(error){
      msg = 'Failed to send message- ';
      res.render('contact', {
        title: 'Contact',
        msg: msg,
        err:error
      });
      return console.log('Failed' + error);
    }
    console.log('Message sent: ' + info.response);
    msg = 'Message sent';
    res.render('contact', {
```

Slika 5-2: Prikaz konektovanja na mail server kao i slanje poruke.

Nakon autentikacije korisnika, njemu se prikazuje chat stranica, sa poljem za prikaz poruka, poljem za unos poruke kao i lista dostupnih soba. Takodje na dnu je prikazana komanda kako se kreira/menja soba. Pored navedenog u navigacionom meniju se pojavljuje dugme “Log Out” za prekidanje sesije kao i link ka stranici “/users” (ako je username admin). Klikom na sobu u listi soba se ulazi u sobu. Takodje, u donjem uglu su prikazne sve moguće komande.

IT320 Savremene Tehnološke Platforme - Projektni Zadatak

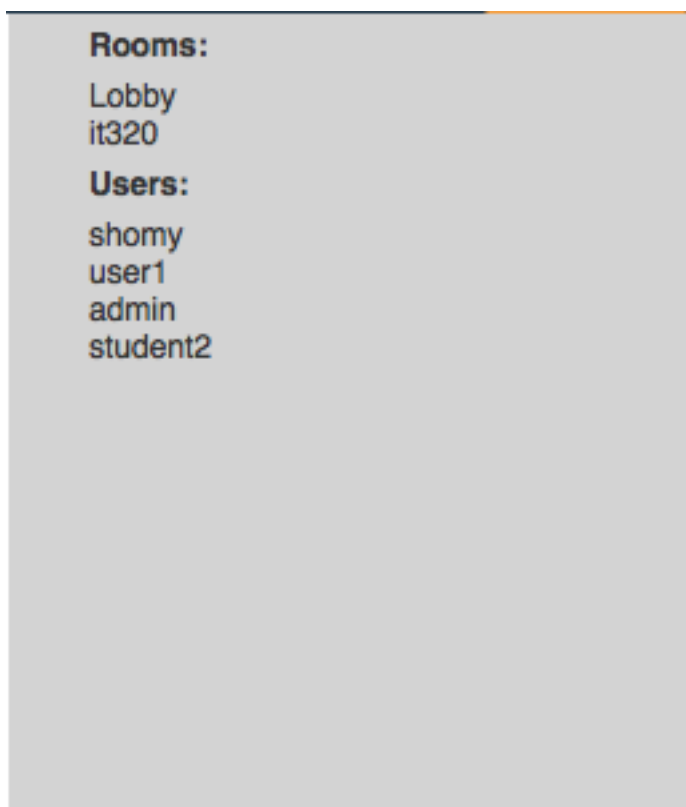


Slika 6: Prikaz Chat stranice.

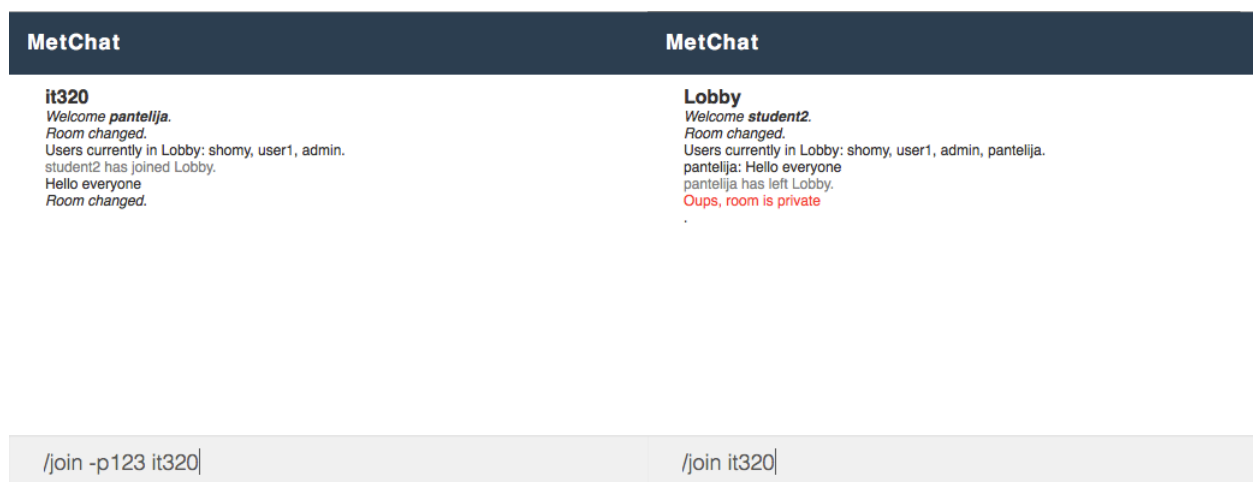
| MetChat | MetChat | MetChat | MetChat |
|--|--|--|--|
| Lobby Welcome <i>user1</i> . Room changed. Users currently in Lobby: shomy, admin. admin has joined Lobby. pantelija has joined Lobby. student2 has joined Lobby. pantelija: Hello everyone | Lobby Welcome <i>shomy</i> . Room changed. Users currently in Lobby: user1. admin has joined Lobby. user1 has joined Lobby. admin has joined Lobby. pantelija has joined Lobby. student2 has joined Lobby. pantelija: Hello everyone | Lobby Welcome <i>student2</i> . Room changed. Users currently in Lobby: shomy, user1, admin, pantelija pantelija: Hello everyone | Lobby Welcome <i>pantelija</i> . Room changed. Users currently in Lobby: shomy, user1, admin. student2 has joined Lobby. Hello everyone |
| Type sth | Type sth | Type sth | Type sth |
| Join/create public room: <code>/join [room name]</code> Join/create private room: <code>/join -p[password] [room name]</code> Delete room (only room author or admin): <code>/delete</code> Merge with room: <code>/merge [room name]</code> Invite user to room (only room author or admin): <code>/invite [user name]</code> | Join/create public room: <code>/join [room name]</code> Join/create private room: <code>/join -p[password] [room name]</code> Delete room (only room author or admin): <code>/delete</code> Merge with room: <code>/merge [room name]</code> Invite user to room (only room author or admin): <code>/invite [user name]</code> | Join/create public room: <code>/join [room name]</code> Join/create private room: <code>/join -p[password] [room name]</code> Delete room (only room author or admin): <code>/delete</code> Merge with room: <code>/merge [room name]</code> Invite user to room (only room author or admin): <code>/invite [user name]</code> | Join/create public room: <code>/join [room name]</code> Join/create private room: <code>/join -p[password] [room name]</code> (e.g. <code>/join -p123 room</code>) Delete room (only room author or admin): <code>/delete</code> Merge with room: <code>/merge [room name]</code> Invite user to room (only room author or admin): <code>/invite [user name]</code> |

Slika 6-1: Prikaz Chat stranice sa više korisnika i soba.

IT320 Savremene Tehnološke Platforme - Projektni Zadatak

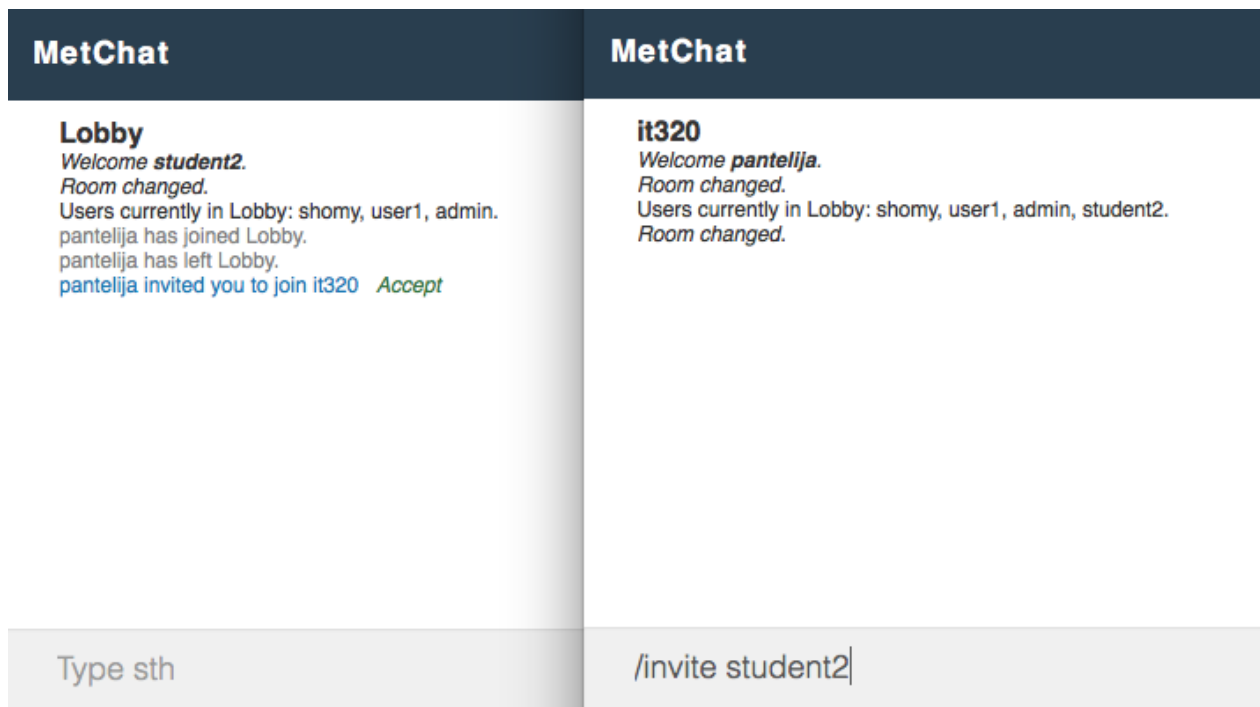


Slika 6-1-1: Lista kreiranih soba i online korisnika



Slika 6-1-2: pantelija je napravio privatnu sobu, student2 pokušava da udje bez lozinke

Korisnik može da napravi privatnu sobu (sobu zaštićenu lozinkom) i ima pravo da poziva druge korisnike u tu sobu ali i da obriše sobu.



Slika 6-1-3: Pantelija je autor sobe i pozvao je student2 da udje u sobu.

Autori soba i admini imaju mogućnost da pozovu druge korisnike u sobu. **Slika 6-1-3.** Student2 je primio obaveštenje da je pozvan u privatnu sobu. Ako klikne na Accept, preći će iz "Lobby" sobe u "it320" sobu.

Takodje, admin može da spoji sobe (osim "Lobby") tako što će izvršiti komandu ***"/merge [ime sobe]"***.

Autor i admin imaju pravo da obrišu sobu. Pri tom će svi korisnici iz sobe biti automatski prebačeni u "Lobby" sobu.

Za svaku od komandi, ukoliko nema ovlašćenja, biće mu prikazana notifikacija druge boje.

Kod

Izradjena je uz pomoć socket.io-a. Socket.io je node.js realtime framework server.

Glavni izazov na ovoj stranici je prosledjivanje passport sesiion id-a posto je chat

```
io.set('authorization', passportSocketIo.authorize({
  passport: require('passport'),
  cookieParser: cookieParser,
  key: 'connect.sid',
  secret: 'keyboard cat', // the session_secret to parse the cookie
  store: session,

  fail: function(data, message, critical, accept) {
    accept(null, false);
    console.log([data, message, critical, accept]);
  },
  success: function(data, accept) {
    console.log('successful authorization - chat_server.js, line: 36');
    accept(null, true);
  }
}));
```

Slika 6-2: Autorizacija (chat_server.js).

odvojen od ostatka sajta. To je rešeno uz pomoć passport.socket.io modula.

U suštini prilikom ranije prijave korisnika na sistem, kreira se kolačić sa session id-om i šifruje se sa tajnim ključem 'keyboard cat' (fajl: app.js).u chat_server.js fajlu na slici 6-2 se taj kolačić (ukoliko postoji) dešifruje sa istim tajnim ključem ('keyboard cat').

```
io.sockets.on('connection', function (socket) {
  var username = socket.handshake.user.username;
  nickNames[socket.id] = username;
  socket.emit('nameResult', {
    success: true,
    name: username
  });
  namesUsed.push(username);
  joinRoom(socket, 'Lobby');
  handleMessageBroadcasting(socket, nickNames);
  handleRoomJoining(socket);
  socket.on('rooms', function() {
    socket.emit('rooms', io.sockets.manager.rooms);
  });
  socket.on('users', function() {
    var allUsers = io.sockets.clients();
    var userList = [];
    if (allUsers.length > 1) {
      for (var index in allUsers) {
        var userSocketId = allUsers[index].id;
        if (userSocketId != socket.id) {
          userList.push(nickNames[userSocketId]);
        }
      }
    }
    socket.emit('users', userList);
  });
  handleClientDisconnection(socket, nickNames, namesUsed);
  handleRoomDeletion(socket);
  handleRoomMerge(socket);
  handleUserInvitation(socket);
});
```

Slika 6-3: Ulazak u sobu, setovanje promenljivi i prikaz informacija (chat_server.js).

```
151 function handleRoomJoining(socket) {  
152   socket.on('join', function(room) {  
153     // Join  
154     var currentRoomName = currentRoom[socket.id].name;  
155     if (allRooms[room.newRoom]) {  
156       var roomPassword = allRooms[room.newRoom].password // check for a new room password  
157       if (roomPassword) {  
158         if (room.password) {  
159           if (room.password !== roomPassword) {  
160             // send to sender-client only  
161             socket.emit('message', {  
162               text: 'Oops, wrong password!',  
163               color: '#f44336'  
164             });  
165             return;  
166           }  
167         } else {  
168           // send to sender-client only  
169           socket.emit('message', {  
170             text: 'Oops, room is private',  
171             color: '#f44336'  
172           });  
173           return;  
174         }  
175       }  
176     }  
177     socket.leave(currentRoomName);  
178     // sending to all clients in 'game' room(channel) except sender  
179     socket.broadcast.to(currentRoomName).emit('message', {  
180       text: nickNames[socket.id] + ' has left ' + currentRoomName + '.',  
181       color: 'grey'  
182     });  
183     joinRoom(socket, room.newRoom, room.password);  
184   });  
185 }
```

Slika 6-4: joinRoom funkcija (chat_server.js).

Nakon uspostavljene konekcije, definiše se kako će se upravljati sa korisnicima. *joinRoom(socket, 'Lobby')* postavlja korisnika u Lobby sobu kada se konektuje. *handleMessageBroadcasting(socket, nickNames)* upravlja porukama korisnika i kreiranjem tj. menjanjem soba/kanala. odmah zatim,ma zahtev se prikazuju aktivne sobe. Na kraju *handleClientDisconnection* je zadužen za brisanje logike nakon što se korisnik diskonektuje.

Za ulazak u sobe je zadužena joinRoom funkcija. Ta funkcija prikazana na slici 6.3, upravlja logikom oko ulaska i kreiranja chat sobe (privatne i javne) . To je moguće tako što se pozove join metod socket objekta. Potom aplikacija komunicira sa ulogovanim korisnikom ali i ostalim u istoj sobi.

Slanje Poruka

```

142
143 function handleMessageBroadcasting(socket) {
144   socket.on('message', function (message) {
145     socket.broadcast.to(message.room).emit('message', {
146       text: nickNames[socket.id] + ': ' + message.text
147     });
148   });
149 }

```

Slika 6-5: handleMessageBroadcasting funkcija (chat_server.js).

Na slici 6.5 je prikazan proces: korisnik emituje događaj koji ukazuje na sobu gde poruka treba da se pošalje i sam tekst poruke. Server potom preusmerava poruku svim ostalim korisnicima u sobi.

```

167 function handleRoomMerge(socket) {
168   socket.on('merge', function(room) {
169     var currentRoomName = currentRoom[socket.id].name;
170     console.log('nickNames[socket.id] ' + nickNames[socket.id]);
171     // console.log('roomAuthor[nickNames[socket.id]] ' + roomAuthor[nickNames[socket.id]].name);
172     // if (nickNames[socket.id])
173     if (nickNames[socket.id] == 'admin' && currentRoomName != 'Lobby') {
174       // if new room doesn't exists
175       if (!allRooms[room.newRoom]) {
176         socket.emit('message', {
177           text: "Oops, room to be merged with doesn't exists",
178           color: '#f44336'
179         });
180         return;
181       }
182       socket.leave(currentRoomName);
183       delete allRooms[currentRoomName];
184       joinRoom(socket, room.newRoom); // merge with
185
186       // redirect all clients from old room to new room
187       io.sockets.clients(currentRoomName).forEach(function(s) {
188         s.leave(currentRoomName);
189         joinRoom(s, room.newRoom);
190       });
191     } else {
192       socket.emit('message', {
193         text: "Sorry, you are not authorised to run that command!",
194         color: '#f44336'
195       });
196     }
197   });
198 }

```

Slika 6-6: handleRoomMerge

Na slici 6.6 je prikazana funkcija koja osluškuje događaj sa klijentske strane, tj. kada korisnik izvrši komandu /merge, emituje poruku koju ova metoda hvata i obradjuje. Ta poruka je ustvari objekat "room" koji ima jedan property—newRoom ili ime sobe u koju se spaja trenutna soba. Funkcija pre svega proverava da li korisnik ima administratorske privilegije i ako trenutna soba nije "Lobby", potom proverava da li uopšte postoji sobu u

koju se spaja. Ukoliko ne postoji emituje se poruka sa obaveštenjem korisniku koji je izvršio komandu. Ukoliko postoji soba, onda prvo izlazi admin (linija 182), brise se soba iz liste kreiranih soba, zatim admin ulazi u novu sobu. Na liniji 187 svi korisnici iz sobe izlaze iz sobe i prelaze u novu sobu.

```

200 ▾ function handleUserInvitation(socket) {
201 ▾   socket.on('inviteUser', function(user) {
202     console.log('handleUserInvitation called');
203     // if user is admin or author of room
204 ▾     if (nickNames[socket.id] == 'admin' || roomAuthor[nickNames[socket.id]] == currentRoom[socket.id]) {
205         var currentRoomName = currentRoom[socket.id].name;
206         var roomPassword = allRooms[currentRoomName].password;
207
208         console.log('roomPassword ' + roomPassword);
209         console.log('user.name: ' + user.name);
210
211         for (key in nickNames) {
212             // check if user exists
213 ▾             if (nickNames[key] == user.name) {
214                 console.log('user exists ' + key);
215                 console.log('user exists ' + nickNames[key]);
216 ▾                 io.sockets.socket(key).emit('message', {
217                     text: nickNames[socket.id] + ' invited you to join ' + currentRoomName,
218                     color: '#337ab7',
219                     password: roomPassword,
220                     notification: 'invite'
221                 });
222                 return;
223             } else {
224                 console.log('user doesnt exists');
225             }
226         }
227 ▾     } else {
228 ▾         socket.emit('message', {
229             text: "Sorry, you are not authorised to run that command!",
230             color: '#f44336'
231         });
232     }
233   });
234 }

```

Slika 6.8: handleUserInvitation funkcija

Ova funkcija prvo osluškuje i hvata poruku koju je emitovao korisnik koji je izvršio poruku. Funkcija hvata user objekat koji ima jedan property—ime korisnika. Kad uhvati objekat, prvo proverava da li je korisnik admin ili autor sobe, tj. da li je uopšte autorizovan da izvrši tu komandu. Ukoliko nije, obaveštava se porukom (linija 228). Ukoliko jeste, prvo se pribavlja trenutna soba, a zatim i trenutna sifra (ukoliko postoji). Zatim se proverava da li korisnik koji se poziva uopšte postoji (linija 213). Na liniji 211, key je ovde ustvari socket.id svakog online korisnika. Ukoliko postoji, emituje se objekat sa tekstom poruke, sifrom sobe i notification svojstvom. Na klijentskoj strani **Slika 6.9** se hvata objekat uzimaju neophodni podaci kako bi se napravila neophodna komanda (sa lozinkom za sobu ili bez) koja se prosledjuje metodi processCommand chatApp prototipu (linija 81).

```

52 ✓ socket.on('message', function (message) {
53     var newElement = $('<div></div>');
54
55     if (message.color) {
56         newElement.css('color', message.color);
57     } else {
58         // newElement.css('color', 'grey'); // default color for notification
59     }
60     // for invitation
61 ✓ if (message.notification) {
62     $('div#notification').removeAttr('id');
63     newElement.attr('id', 'notification');
64     var n = message.text.split(" ");
65     var room = n[0];
66     var acceptElement = newElement;
67     var command = '/join ' + room + ' ' + message.password;
68
69 ✓ if (message.password) {
70     var password = message.password;
71     command = '/join ' + '-p' + password + ' ' + room;
72 }
73 }
74
75 newElement.text(message.text);
76 $('#messages').append(newElement);
77 $('#notification').append(acceptElement);
78
79 ✓ $('#accept').one('click', function() {
80 ✓ // $('#accept').click(function() {
81     chatApp.processCommand(command);
82     $('div#notification').removeAttr('id');
83 });
84 });

```

Definition:
[node_modules/socket.io/lib/stores/redis.js:124](#)
 ge.text is the name of the room
 .text("Accept");

Slika 6.9: Priprihvatanje i obrada emitovane poruke iz funkcije handleUserInvitation

JavaScript Klijentska Strana

```

1  var Chat = function(socket) {
2    this.socket = socket;
3  };
4
5  Chat.prototype.sendMessage = function(room, text) {
6    var message = {
7      room: room,
8      text: text
9    };
10   this.socket.emit('message', message);
11 };
12
13 Chat.prototype.changeRoom = function(room) {
14   this.socket.emit('join', {
15     newRoom: room
16   });
17 };
18
19 Chat.prototype.processCommand = function(command) {
20   var words = command.split(' ');
21   var command = words[0]
22     .substring(1, words[0].length)
23     .toLowerCase();
24   var message = false;
25
26   switch(command) {
27     case 'join':
28       words.shift();
29       var room = words.join(' ');
30       this.changeRoom(room);
31       break;
32     case 'nick':
33       words.shift();
34       var name = words.join(' ');
35       this.socket.emit('nameAttempt', name);
36       break;
37     default:
38       message = 'Unrecognized command.';
39       break;
40   }
41
42   return message;
43 };

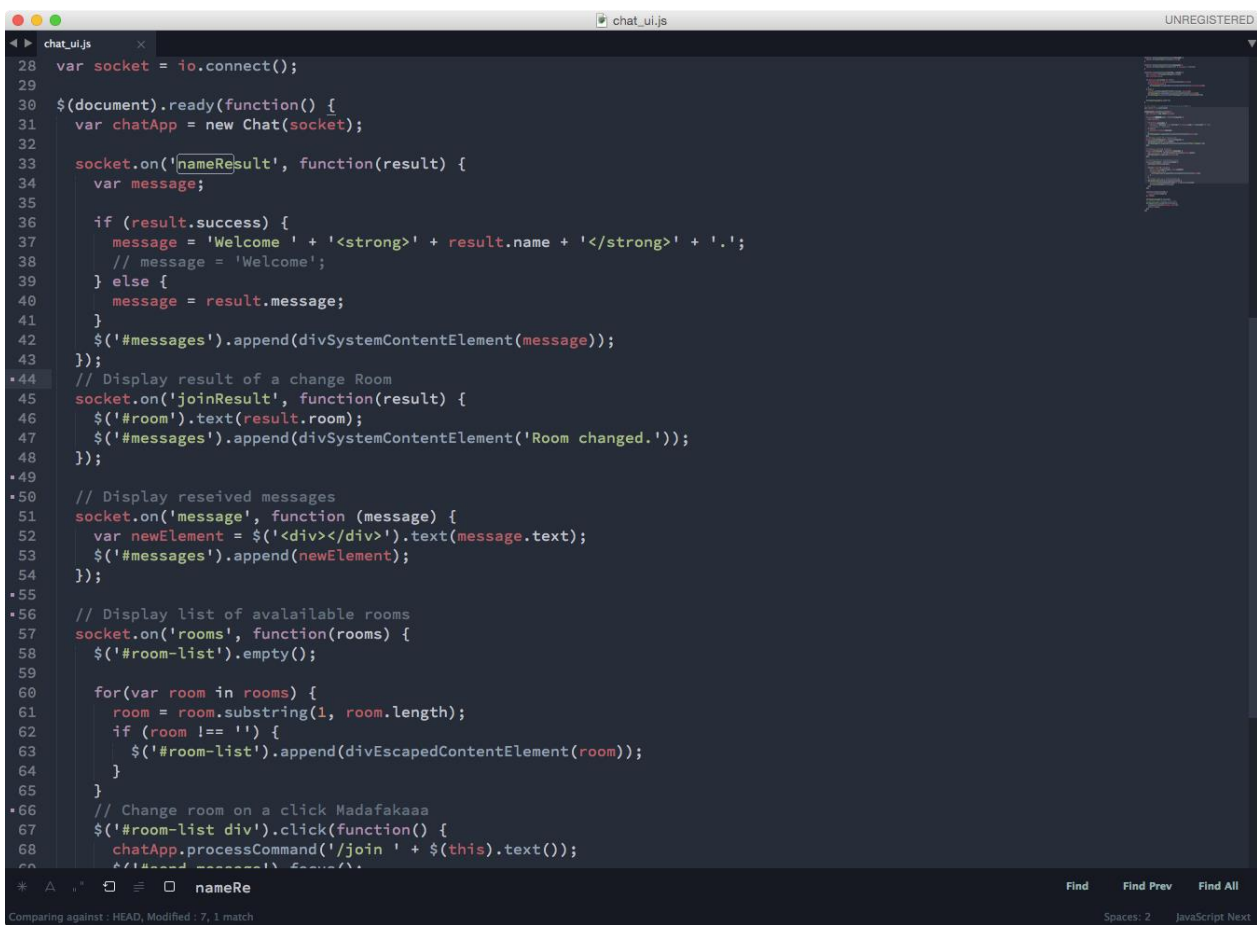
```

Slika 6-6: chat.js

Prvo je definisan prototip objekat koji će izvršiti sve chat komande, poslati poruke i zahteve za promenu sobe. Ovaj fajl je ekvivalent klase i prima jedan argument—instancu Socket.io objekta.

Sledeće na redu je implementacija logike koja je u direktnoj interakciji sa pretraživačem koristeći jQuery. Prva funkcionalnost je prikaz tekstualnih podataka. Pošto podaci koji dolaze iz browsera su *untrusted* podaci zato što zlonamerni korisnici mogu da unesu javascript kroz polje za unos implementiraćemo dve funkcije za prikaz podataka. *divEscapeContentElement* prikazuje nebezbedan tekst. Njegova uloga je da transformiše specijalne karaktere u html entitete. Druga funkcija *divSystemContentElement* prikazuje bezbedan sadržaj koji generiše sistem a ne korisnik.

IT320 Savremene Tehnološke Platforme - Projektni Zadatak



```
28 var socket = io.connect();
29
30 $(document).ready(function() {
31   var chatApp = new Chat(socket);
32
33   socket.on('nameResult', function(result) {
34     var message;
35
36     if (result.success) {
37       message = 'Welcome ' + '<strong>' + result.name + '</strong>' + ' .';
38       // message = 'Welcome';
39     } else {
40       message = result.message;
41     }
42     $('#messages').append(divSystemContentElement(message));
43   });
44   // Display result of a change Room
45   socket.on('joinResult', function(result) {
46     $('#room').text(result.room);
47     $('#messages').append(divSystemContentElement('Room changed.));
48   });
49
50   // Display received messages
51   socket.on('message', function (message) {
52     var newElement = $('<div></div>').text(message.text);
53     $('#messages').append(newElement);
54   });
55
56   // Display list of available rooms
57   socket.on('rooms', function(rooms) {
58     $('#room-list').empty();
59
60     for(var room in rooms) {
61       room = room.substring(1, room.length);
62       if (room !== '') {
63         $('#room-list').append(divEscapedContentElement(room));
64       }
65     }
66     // Change room on a click Madafakaaa
67     $('#room-list div').click(function() {
68       chatApp.processCommand('/join ' + $(this).text());
69     });
70   });
71 });
```

Slika 6-7: chat_ui.js

chat_ui.js je zadužen za osluškivanje i prikaz poruka sa servera.

Users Stranica

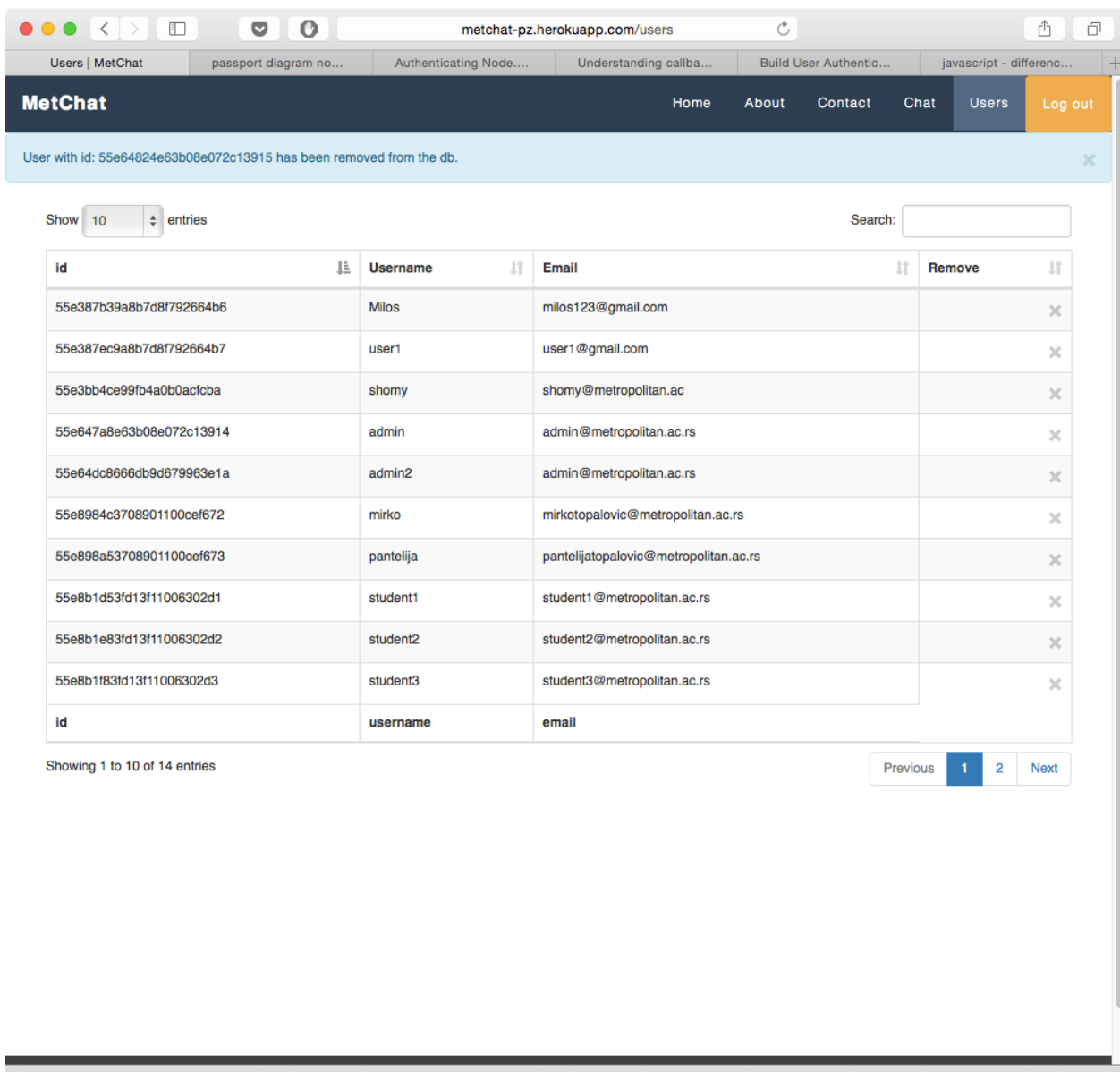
Ovoj stranici može pristupiti samo admin i na njoj se nalazi tabela sa registrovanim korisnicima. Admin ima mogućnost pretrage korisnika kao i da nekog korisnika ukloni sa sistema.

The screenshot shows a web browser window with the URL `metchat-pz.herokuapp.com/users`. The page has a dark blue header with the "MetChat" logo and navigation links: Home, About, Contact, Chat, Users (active), and Log out. Below the header, there is a "Show 10 entries" dropdown and a search bar. The main content is a table with 15 entries, displaying columns for id, Username, Email, and a Remove button. The table is sorted by id in ascending order. The bottom of the page shows pagination controls: "Showing 1 to 10 of 15 entries" and buttons for Previous, 1 (selected), 2, and Next.

| id | Username | Email | Remove |
|--------------------------|-----------|---------------------------------------|--------|
| 55e387b39a8b7d8f792664b6 | Milos | milos123@gmail.com | × |
| 55e387ec9a8b7d8f792664b7 | user1 | user1@gmail.com | × |
| 55e3bb4ce99fb4a0b0acfcba | shomy | shomy@metropolitan.ac | × |
| 55e647a8e63b08e072c13914 | admin | admin@metropolitan.ac.rs | × |
| 55e64824e63b08e072c13915 | admin1 | admin@metropolitan.ac.rs | × |
| 55e64dc8666db9d679963e1a | admin2 | admin@metropolitan.ac.rs | × |
| 55e8984c3708901100cef672 | mirko | mirkotopalovic@metropolitan.ac.rs | × |
| 55e898a53708901100cef673 | pantelija | pantelijatopalovic@metropolitan.ac.rs | × |
| 55e8b1d53fd13f11006302d1 | student1 | student1@metropolitan.ac.rs | × |
| 55e8b1e83fd13f11006302d2 | student2 | student2@metropolitan.ac.rs | × |

Slika 7: Users stranica sa registrovanim korisnicima.

IT320 Savremene Tehnološke Platforme - Projektni Zadatak



Slika 7-1: Prikaz obaveštenja da je korisnik uspešno uklonjen iz baze.

Kod

Logički deo (kontroler) je smešten u `routes/index.js` fajl a view u `views/users.jade`. Nakon pristupa `/users` stranici uz pomoc `find()` metode objekta `User` konektujemo se na Mongo bazu i povlačimo celu kolekciju `users`. Potom ih stavljamo u objekat `userMap` koji prosledjujemo view delu na dalju obradu prilikom renderovanja. Slika 7-2.

Brisanje Korisnika

U view delu, prilikom izlistavanja korisnika iz userMap objekta. svakom tr elementu se dodejuje id korisnika u kome se nalazi sam taj korisnik nalazi. Pošto u svakom redu (tr element) se nalazi dugme za brisanje korisnika, klikom na to dugme preusmerava se na /remove/id stranicu sa prosledjenim id-om korisnika kao url parametar. Nakon toga u url-u imamo id korisnika koga želimo da uklonimo.

```
158      /* Get users */
159      router.get('/users', function(req, res) {
160          User.find({}, function(err, users) {
161              var userMap = {};
162
163              users.forEach(function(user) {
164                  userMap[user._id] = user;
165              });
166              console.log('UserMap: ' + userMap);
167              // res.send(userMap);
168
169              res.render('users', {
170                  title: 'Users',
171                  useri: req.user.username,
172                  userMap: userMap,
173                  message: req.flash('message'),
174              });
175          });
176      });
177  }
```

Slika 7-1: Logika prikaza Users stranice u index.js datoteci.

Metoda za brisanje iz baze je findByIdAndRemove(metoda Mongoose modula za konektovanje na Mongo bazu). Toj metodi se prosledjuje parametar iz uri-a. Nakon uspešnog brisanja, poruka da je korisnik obrisani se smešta u flash sesiju kao message

promenljiva a korisnik se preusmerava nazad na /users/ stranici sa novom. To znači da

```
178     router.get('/remove/:id', function(req, res) {
179         uid= req.params.id.toString();
180         console.log('uid: ' + uid);
181         User.findByIdAndRemove({'_id': uid}, function(err, user) {
182             if (user) {
183                 console.log('User nadjen');
184             } else {
185                 console.log('user nie nadjen');
186             }
187         });
188         req.flash('message', 'User with id: ' + uid + ' has been removed from the db. ');
189         res.redirect('/users');
190     });
```

Slika 7-4: index.js datoteka prikaz brisanja korisnika iz baze na osnovu url parametra.

odmah nakon toga se izvršava router.get(/users) koji prosledjuje message promenljivu iz sesije view delu. To znači da sad u users.jade stranici možemo da pristupimo toj promenljivoj i to radimo tako sto je prikazujemo kao banner da je korisnik uspešno uklonjen iz baze.

```
1  extends layout
2
3  block content
4      #wrap
5          if(typeof message[0] != 'undefined')
6              #message.alert.alert-info.flash.fade.in.alert-dismissible(role="alert")
7                  button.close(type='button', data-dismiss='alert', aria-label='Close')
8                  span(aria-hidden='true') ×
9                  span= message
10         .container#users-cont
11             table#example.table.table-striped.table-bordered(cellspacing='0', width='100%')
12                 thead
13                     tr
14                         th id
15                         th Username
16                         th Email
17                         th Remove
18                 tfoot
19                     tr
20                         th id
21                         th username
22                         th email
23                 tbody
24                     each member in userMap
25                         tr(id= member._id)
26                             td= member._id
27                             td= member.username
28                             td= member.email
29                             td
29                                 a.close(href='/remove/' + member._id type='submit', aria-label='Close')
30                                 span(aria-hidden='true') ×
```

Slika 7-3: users.jade datoteka.

Log out dugme

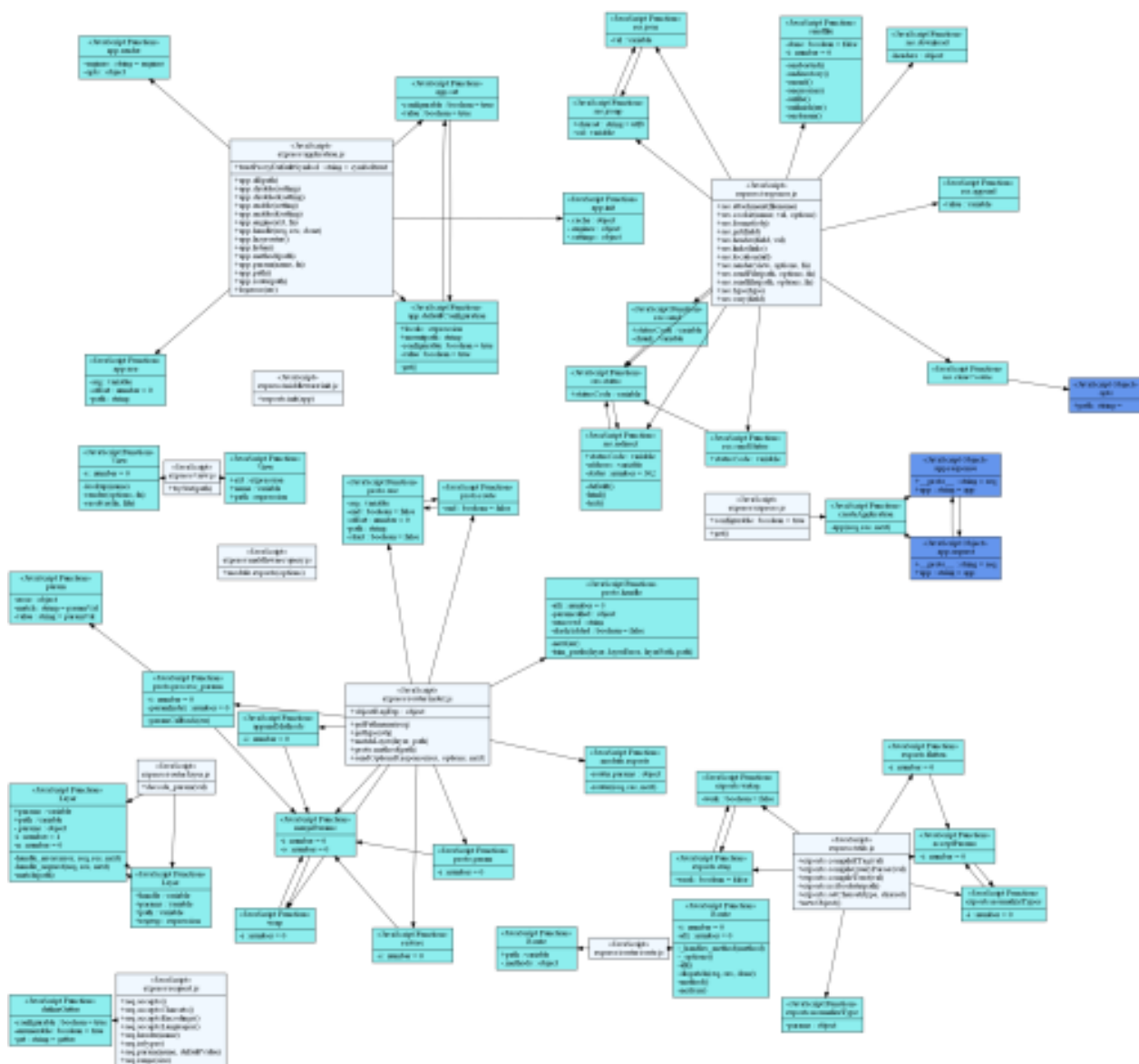
Pritiskom da logout, izvršava se get zahtev za /signout stranicu. Tom prilikom se poziva logout() metod nad request objektom i preusmerava se na root stranicu.

```
70      /* Get Logout */
71      router.get('/signout', function(req, res) {
72          req.logout();
73          // req.session.destroy();
74          res.redirect('/');
75      });
76
```

Slika 7-5: index.js raskidanje sesije.

Dijagram Klasa

IT320 Savremene Tehnološke Platforme - Projektni Zadatak



Slika 7-5: express.js dijagram

Baza Podataka

Za bazu podataka sam koristio NoSQL bazu MongoDB, on a čuva podatke kao JSON datoteke sa dinamičnim šemama. Moja baza ima dve kolekcije: users i sessions. U users kolekciji su smešteni registrovani korisnici sa poljima prikazanim na slici 7.6 a u sessions kolekciji sesije.

```
1 var mongoose = require('mongoose');
2
3 module.exports = mongoose.model('User',{
4   username: String,
5   email: String,
6   password: String,
7 });
```

Slika 7-6: User Šema

Testiranje

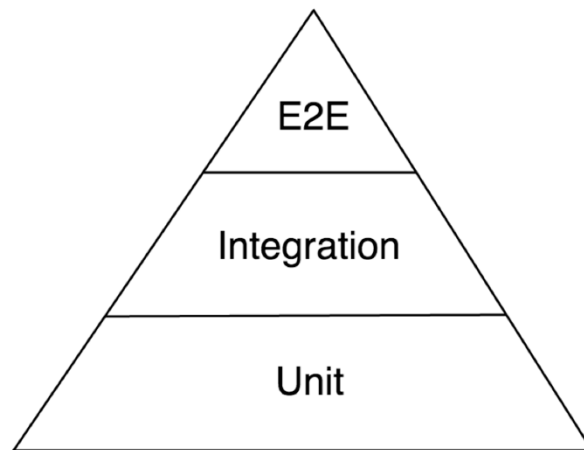
Testiranje predstavlja pokušaj da se pronađu greške u softveru koji je napravljen.

Softver je implementiran prema korisničkim zahtevima kojima se rešava neki realni problem ili se kreira neka korisna funkcionalnost koja predstavlja nešto što je potrebno krajnjim korisnicima. Kada se implementira, softver može u većoj ili manjoj meri da odgovara originalnim zahtevima prema kojima je i napravljen.

Svako ponašanje softvera koje se ne slaže sa originalnim zahtevima predstavlja grešku koju je potrebno identifikovati i otkloniti. U užem smislu, testiranje predstavlja upravo proveru da li je određeni softver u potpunosti implementiran prema originalnim korisničkim zahtevima. U širem smislu testiranje predstavlja sistem kontrole kvaliteta (QA – Quality Assurance) kojim se ne proverava samo softver već i sve njegove prateće komponente i karakteristike.

Testiranje softvera se najčešće deli na funkcionalno i nefunkcionalno testiranje, u zavisnosti od toga da li se kontrolišu samo krajnje funkcionalnosti aplikacije ili sam programerski kod. Postoji mnogo vrsta testiranja ali ovde ću navesti najčešće vrste testiranja koje se koriste za Node.js web aplikacije:

- Jedinično testiranje (eng. *Unit testing*)
- Integraciono testiranje (eng. *Integration testing*)
- Sistemsko testiranje (eng. *System testing*)
- Testiranje prihvatljivosti (eng. *Acceptance testing*)



Slika 8: Piramida testiranja

Jedinično testiranje

Jedinično testiranje (eng. *Unit testing*) osigurava da su svi pojedinačni moduli softverskog sistema testirani i da svaki od njih pojedinačno radi ispravno. Jedinično testiranje ipak ne garantuje da li će ovi moduli raditi u redu ukoliko se integrišu u siste— to garantuju integracioni testovi.

Za svaku atomsku jedinicu koda se pravi test koji testira tu istu jedinicu. Unit testiranjem prolazi se svaki i najmanji deo sistema, pa upravo zbog toga ima važnu ulogu prilikom osiguravanja kvaliteta razvijenog softvera.

Testiranje Node.js aplikacije se vrši uz pomoć **Mocha** alata za testiranje. Neophodno je da se napravi novi direktorijum u root direktorijumu aplikacije sa nazivom "test". Zatim se pokreće node server a potom se izvršava komanda "mocha -R spec".

Za početak, neophodno je locirati metodu koju želimo da testiramo. Na slici 9 se može videti metoda koja testira socket.io metodu koja registruje kada se klijent konektuje. Na slici 10 se testira registracija i prijava korisnika.

.


```
1  var should = require('should');
2  var io = require('socket.io-client');
3
4  var socketURL = 'http://127.0.0.1:5000';
5
6  var options = {
7    transports: ['websocket'],
8    'force new connection': true
9  };
10
11  var chatUser1 = { 'name': 'Milos' };
12
13  describe("Chat Server", function () {
14
15    /* Test 1 - A Single User */
16    it('Should broadcast new user once they connect', function (done) {
17      var client = io.connect(socketURL, options);
18
19      client.on('connect', function (data) {
20        client.emit('connection name', chatUser1);
21      });
22
23      client.on('new user', function (usersName) {
24        usersName.should.be.type('string');
25        usersName.should.equal(chatUser1.name + " has joined.");
26        client.disconnect();
27        done();
28      });
29    });
30  });
```

Slika 9: Testiranje notifikacije chat servera kada se klijent konektuje

```
Chat Server
  1) Should broadcast new user once they connect

0 passing (2s)
1 failing

1) Chat Server Should broadcast new user once they connect:
   Error: Timeout of 2000ms exceeded. For async tests and hooks, ensure "done()" is called;
   if returning a Promise, ensure it resolves.
```

Slika 9-1: Rezultat testiranja notifikacije chat servera kada se klijent konektuje (Test nije prošao)

```
14 var io = require('socket.io-client');
15 var socketURL = 'http://127.0.0.1:5000';
16 var chatUser1 = { 'name': 'Milos' };
17
18 request = require("supertest");
19 agent = request.agent(app)
20
21 describe('User', function () {
22   before(function (done) {
23     user = new User({
24       email: "petarpan1@metropolitan.ac.rs",
25       username: "petarpan1",
26       password: "123"
27     });
28     user.save(done)
29   });
30   describe('Login test', function () {
31     it('should redirect to /', function (done) {
32       agent
33         .post('/login')
34         .field('username', 'petarpan1')
35         .field('password', '123')
36         .expect('Location', '/')
37         .end(done)
38     })
39   })
40 })
```

Slika 10: Testiranje kreiranja naloga i prijave

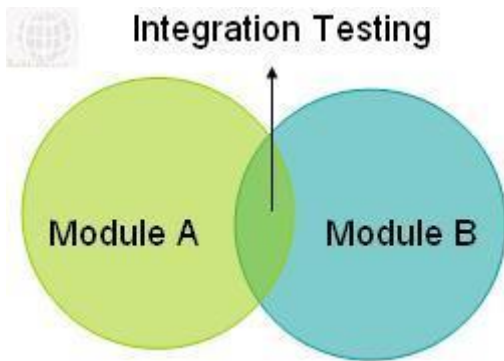
```
POST /login 302 16.285 ms - 23
✓ should redirect to / (1907ms)

1 passing (2s)
```

Slika 10-1: Rezultat jediničnog testiranja (Test je uspešno prošao)

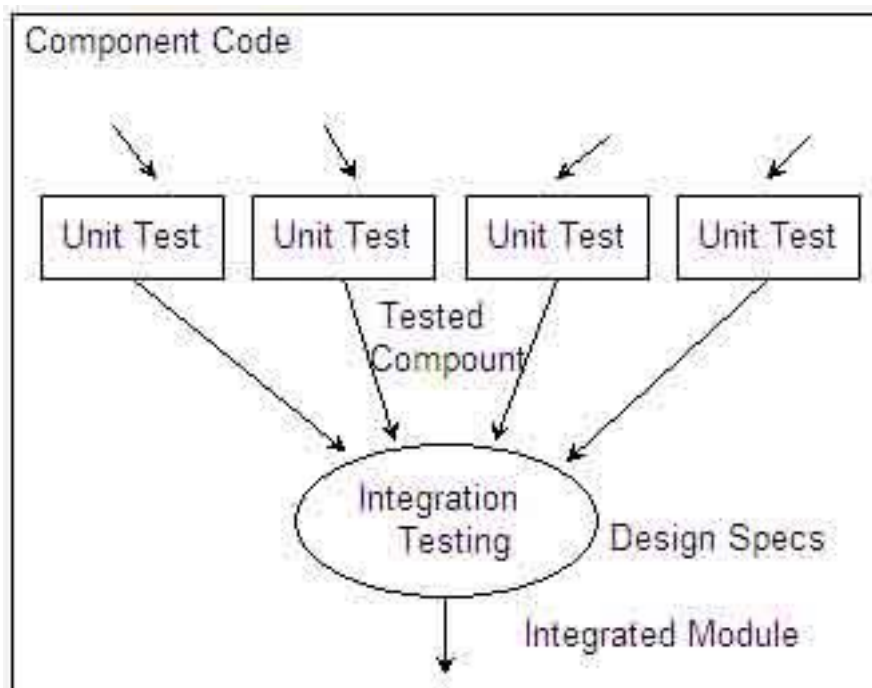
Integraciono testiranje

Integraciono testiranje (eng. Integration testing ili Integration and Testing) je faza u testiranju softvera u kojoj se pojedinačni moduli softverskog sistema kombinuju i testiraju kao grupa i tako se otkrivaju greške. Ovo testiranje se dešava pre sistemskog, a nakon jediničnog testiranja.



Slika 11: Ilustracija integracionog testiranja

Cilj integracionog testiranja je da se moduli, koji su jedinično testirani, integrišu, zatim da se pronađu greške, da se te greške uklone i da se izgradi celokupna struktura sistema, kao što je predviđeno dizajnom.



Slika 12: Nivoi integracionog testiranja

Da bi mogli da izvršimo integraciono testiranje kod Node.js web aplikacija, moramo da imamo sledeće Mocha okvir za testiranje kao i SuperTest i Chai biblioteke za testiranje.

Mocha je Javascript okvir za testiranje koji se radi u web čitaču i na Node.js-u. Jedna od njegovih glavnih mogućnosti je da učini asinhrono testiranje veoma jednostavno.

Chai je BDD / TDD assertion biblioteka za node i web čitač koji se lako integriše sa Mocha okvirom za testiranje.

SuperTest je biblioteka koja omogućava testiranje Node.js HTTP servera koristeći jednostavan API.

```
it('returns username if name param is a valid user', function(done) {
  users.list = ['test'];
  superagent.get('http://localhost:5000/user/test').end(function(err, res) {
    assert.ifError(err);
    assert.equal(res.status, status.OK);
    var result = JSON.parse(res.text);
    assert.deepEqual({ user: 'test' }, result);
    done();
  });
});
```

Slika 13: Metoda koja testira da li korisnik postoji

Kod na slici 13 demonstrira 2 ključne ideje API-level integracionog testiranja. Za početak, pošto je Node.js aplikacija u pitanju, kreira se Express server zatim se koristi superagent koji šalje HTTP zahteve ka serveru. Druga ključna ideja je ta pošto server i testovi dele iste procese, veoma je lako manipulirati serverom onako kako testovi zahtevu. Mogu da se manipulšu podaci, može da se isključi server mid-request, može i da se simuliraju maliciozni zahtevi.

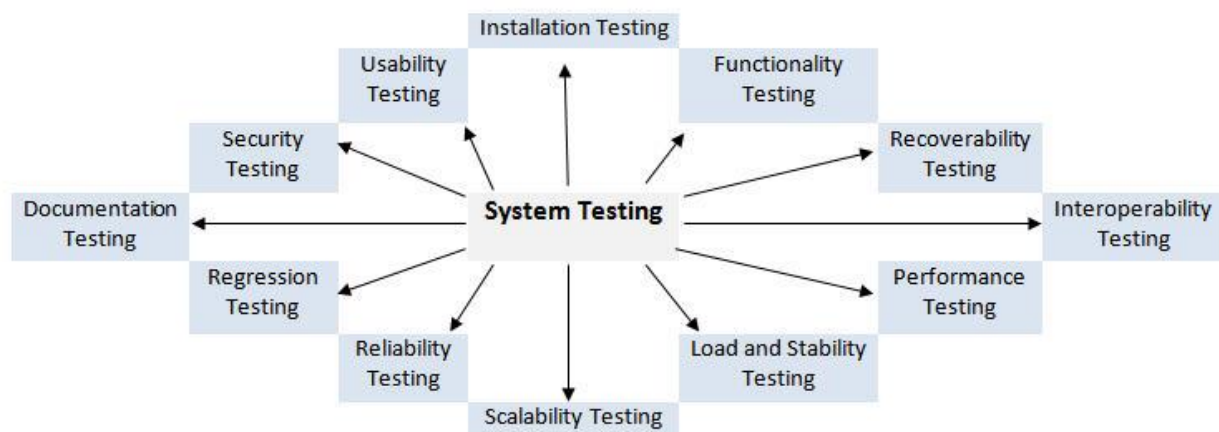
Integracioni testovi na nivou API-ja su veoma korisni za pronalaženje bagova u integraciji izmedju modula. Veoma su korisni za REST API TDD, ali i za generisanje dokumentacije.

Integraciono testiranje polazi od komponentata koji su prošle jedinično testiranje, grupiše ih u veće celine, primenjuje testove definisane u planu integracionog testiranja i daje kao izlaz integrisan sistem spreman za sistemsko testiranje.

Sistemsko testiranje

Sistemsko testiranje podrazumeva testiranje koje se sprovodi nad završenim, integrisanim sistemom kako bi se procenila usklađenost sistema za određenim zahtevima. Sistemske testiranje spada u okvir testiranja crne kutije i kao takvo, ne bi trebalo da zahteva nikakvo poznavanje unutrašnjeg dizajna koda ili logike.

Po pravilu, sistemske testiranje, kao svoj "ulaz" (eng. *input*) podrazumeva sve integrisane softverske komponente koje su prošle integraciono testiranje, ali isto tako i sam softver integrisan sa bilo kojim važećim hardverskim sistemom/sistemima.



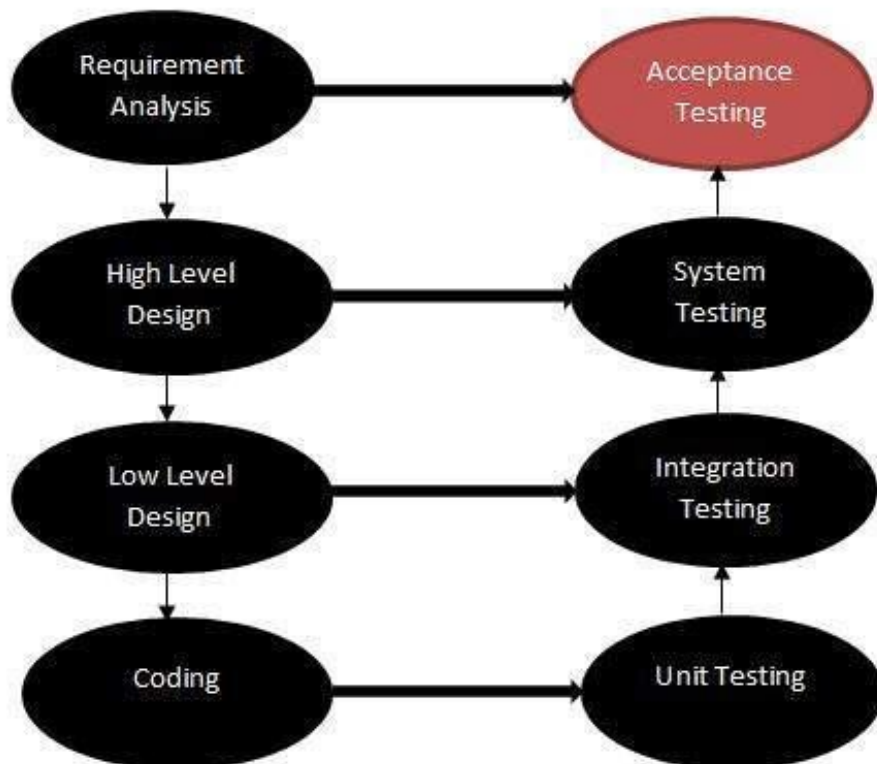
System Testing - © www.SoftwareTestingHelp.com

Slika 14: Sistemsko (E2E) testiranje

Da bi sistemsko testiranje dalo što bolji uvid u rad celokupnog sistema, od izuzetne je važnosti obezbediti što bolju metriku, odnosno prikupljanje relevantnih podataka o radu različitih delova sistema za vreme sprovođenja testova. Svi prikupljeni podaci se po završetku testiranja prosleđuju dalje na analizu kako bi se uporedili sa očekivanim, odnosno sa definisanim okvirima u kojima bi sistem trebao da radi. Ako su dobijeni rezultati ispod očekivanih, neophodno je pronaći delove koji predstavljaju „usko grlo“ i ispraviti ih na način da nemaju negativan uticaj na rad sistema. Ako su pak dobijeni rezultati iznad očekivanih, to je znak da sistem može da izvrši veće opterećenje od onog definisanog u zahtevima prilikom razmatranja i planiranja.

Test Prihvatljivosti

Test prihvatljivosti (eng. *acceptance testing*) je formatni opis ponašanja softverskog proizvoda, generalno izražen kao primer ili scenario korišćenja. Za takve primere ili scenarije predložen je niz različitih pristupa. U mnogim slučajevima cilj je da bi trebalo da bude moguće automatizirati izvršenje takvih testova pomoću softverskog alata, bilo ad-hoc-a za razvojni tim ili drugih.



Slika 15: Testiranje prihvatljivosti u životnom ciklusu testiranja

Slično jediničnom testu, test prihvatljivosti softvera generalno ima binarni rezultat— prolazak ili neuspeh. Nedostatak sugerše, iako ne dokazuje prisustvo defekta u proizvodu.

Timovi sazrevaju u svojoj praksi testiranja prihvatljivosti upotrebe kao glavnog oblika funkcionalne specifikacije i jedine formalne ekspresije poslovnih zahteva. Ostali timovi koriste testove prihvatanja kao dodatak dokumentaciji specifikacije koja sadrži slučajeve korišćenja ili više narativnog teksta.

Za testiranje prihvatljivosti kod Node.js aplikacije, koristi se Selenium aplikacija sa web drajverom za Node.js ili češće Concept.js JavaScript okvir.

Concept.js funkcionalnosti:

- Scenario Driven - omogućava pisanje testova prihvatljivosti iz ugla korisnika. Svaka komanda je opisana kao akcija korisnika prilikom posete sajtu.
- Backend Agnostic - Obezbeđuje API visokog nivoa koji se lako može izvršiti koristeći jednu od popularnih biblioteka za testiranje:
- Interactive Shell - Kontrolišite veb pregledač u realnom vremenu sa API-jem CodeceptJS-a.

-

Instalacija: "npm install -g codeceptjs".

```
Forms --
submit form successfully
• I am on page "/documentation"
• I fill field "Email", "hello@world.com"
• I fill field "Password", "123456"
• I check option "Active"
• I check option "Male"
• I click "Create User"
• I see "User is valid"
• I dont see in current url "/documentation"
✓ OK in 13352ms

OK 1 passed // 13s
```

Slika 16: Rezultat testiranja prihvatljivosti iz komandne linije

Puštanje u produkciju

Kako bi softver mogao da se pusti u produkciju, neophodno je da se obezbede svi tehnički uslovi za njegov neometani rad.

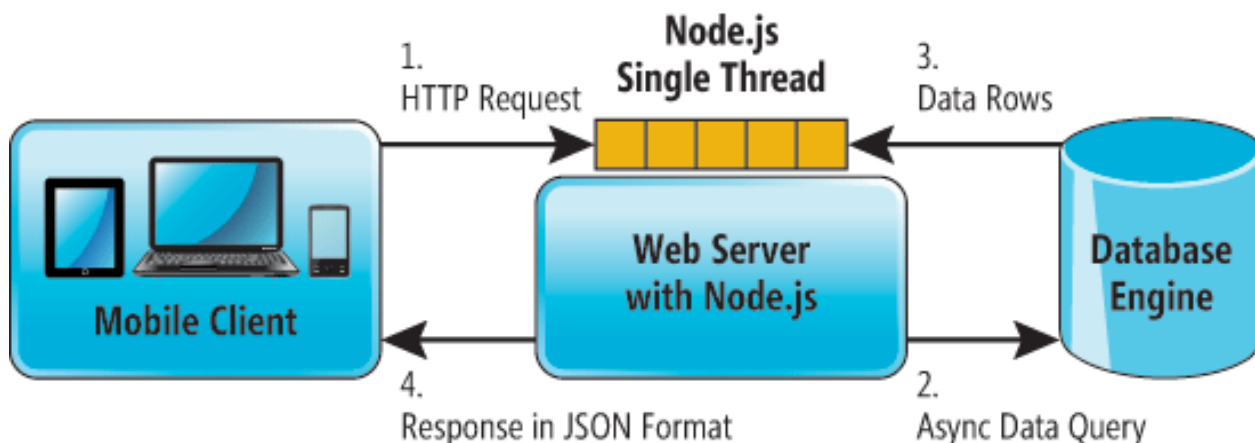
Pre nego što se sajt pusti u produkciju, mora se:

- Izabrati okruženje za hostovanje Express aplikacije
- Neophodno je napraviti nekoliko izmena u postavkama projekta
- Postaviti infrastrukturu za produkciju za serviranje veb stranice

Produkciono okruženje (eng. *production environment*) je okruženje koje pruža serverski računar u kome treba pokrenuti svoju veb stranicu za prave korisnike. Okruženje uključuje:

- Računarski hardver na kome će se veb stranica raditi
- Operativni sistem (npr. Linux ili Windows)
- Programski jezik i okvirne biblioteke na kojima je napisana veb aplikacija
- Infrastruktura veb servera, eventualno obuhvatajući veb server, obrnuti proksii, balanser opterećenja (eng. *load balancer*) itd.

- Baze podataka koje koristi veb stranica



Slika 17: Infrastruktura Node.js aplikacije

Server računar bi mogao biti lociran u našim prostorijama i povezan sa Internetom brzom vezom, ali je mnogo češće koristiti računar koji je hostovan "u oblaku". To zapravo znači da naš kod radi na nekom udaljenom računaru (ili možda "virtuelnom" računaru) u data centru naše hosting kompanije. Na udaljenom serveru se obično nudi nekoliko garantovanih nivoa računarskih resursa (npr. CPU, RAM, memorijska memorija itd.) i Internet povezivanje za određenu cenu.

Ova vrsta udaljeno dostupnog računarskog/mrežnog hardvera se naziva Infrastructure as a Service (IaaS). Mnogi IaaS proizvođači pružaju opcije za prethodno instaliranje određenog operativnog sistema, na koji moramo instalirati ostale komponente našeg proizvodnog okruženja. Ostali proizvođači nam takođe omogućavaju da izaberemo više okruženja koje su u potpunosti opremljene, možda uključujući kompletnu postavku Node.js-a.

Ostali hosting provajderi podržavaju Express kao deo platforme kao usluge (PaaS). Kada bi koristili ovu vrstu hostinga, ne bi morali da brinemo o većini našeg proizvodnog okruženja (serveri, balanseri opterećenja itd.) Jer se host platforma bavi time za nas. To čini primenu prilično jednostavno, jer se samo trebamo usredsrediti na svoju veb aplikaciju, a ne na bilo koju drugu serversku infrastrukturu.

Neki programeri biraju povećanu fleksibilnost koju pruža IaaS preko PaaS-a, dok će drugi ceniti smanjene troškove održavanja i lakše skaliranje PaaS-a. Kada započnemo, podešavanje naše veb lokacije na PaaS sistemu je mnogo lakše.

IT320 Savremene Tehnološke Platforme - Projektni Zadatak

Postoje mnogi hosting provajder koji pružaju podršku Node-u i Express-u. Jedan od najpoznatijih je **Heroku** koji omogućava besplatno (postoji limit resursa) PaaS okruženje dok AWS, Microsoft Azure nude besplatni kredit kada prvi put napravimo nalog.

Neke promene je nophodno napraviti prilikom puštanja aplikacije u produkciju:

- Setovanje `NODE_ENV = 'production'` - kako bi sakrili stack tragove.
- Smanjiti Log pozive - mogu da utiču negativno na performanse aplikacije prilikom velike aktivnosti korisnika
- Gzip/deflate kompresija za HTTP odgovore - web server bi trebao da kompresuje svoje HTTP odgovore koje šalje klijentu jer tako drastično smanju vreme koje je potrebno klijentu da dobije odgovor i učitava stranicu.
- Helmet middlewaree paket - omogućava da se zaštiti aplikacije od najčešćih napada tako što postavlja odgovarajuće HTTP header-e. Instalira se izvršavanjem komande `"npm install helmet --save"`

Zaključak

Node.js omogućava korišćenje jedinstvenog jezika između front-end i back-end-a. To znači da kompletnu aplikaciju možete realizovati pomoću jednog programskog jezika. Takodje, veoma je brz. Teži neblokirajućem i asinhronom načinu programiranja i u isto vreme može da obavlja više stvari odjednom. Baš zbog toga je idealna platforma za izradu realtime chat aplikacije. Uz pomog Express.js framework-a koje je nezaobilazan za izradu svake aplikacije