



# IZRADA VEB APLIKACIJE ZA POLAGANJE TESTOVAZNANJA

DEVELOPMENT OF WEB APPLICATION FOR  
TAKING EXAMS

## Diplomski rad

Student: *Dimitrije Radojević, br. ind. 09/0112*

Mentori: *Boško Nikolić, PhD*  
*Dražen Drašković, MSc*

Elektrotehnički fakultet, Univerzitet u Beogradu

5. oktobar 2016

*Autor želi da izrazi zahvalnost mentorima na razumevanju vannastavnih  
obaveza autora, i svu pomoć tokom izrade rada.*

# Sadržaj

<b>1 Uvod</b>	<b>3</b>
<b>2 Zahtevi za realizacijom sistema</b>	<b>4</b>
2.1 Korisnički zahtevi . . . . .	4
2.1.1 Prva grupa zahteva . . . . .	4
2.1.2 Druga grupa zahteva . . . . .	7
2.2 Opis korišćenih tehnologija . . . . .	10
2.2.1 Izbor jezika . . . . .	10
2.2.2 Bekend tehnologije . . . . .	11
2.2.3 Frontend tehnologije . . . . .	13
2.2.4 Alat za kompajliranje i pakovanje aplikacije . . . . .	13
<b>3 Opis rada sistema</b>	<b>15</b>
3.1 Opis rada sistema iz perspektive studenta . . . . .	15
3.1.1 Logovanje na sistem i registracija . . . . .	15
3.1.2 Stranica sa pregledom testova . . . . .	16
3.1.3 Stranica sa pitanjima . . . . .	16
3.1.4 Primer demonstracije: svodenje na KNF . . . . .	20
3.2 Opis rada sistema iz perspektive profesora . . . . .	21
3.2.1 Stranica za pretraživanje završenih testova . . . . .	21
3.2.2 Pregled i izmena ocene završenog testa . . . . .	22
3.2.3 Dodavanje novih testova . . . . .	23
<b>4 Realizacija sistema</b>	<b>27</b>
4.1 Uprošćavanje stabla parsiranja . . . . .	27
4.2 Prilagođivanje HTML stranica za neke statusne kodove . . . . .	30
4.3 SQL upit za dohvatanje završenih testova . . . . .	34
4.4 Raspakivanje Postgres nizova . . . . .	36
4.5 <i>Anti-forgery</i> token . . . . .	38
4.6 Moguće nadogradnje sistema . . . . .	40
<b>5 Zaključak</b>	<b>42</b>

# Glava 1

## Uvod

Cilj ovog rada jeste izrada veb aplikacije čija je namena da pomogne studen-tima pri savladavanju gradiva iz oblasti *formalne logike* koja se izučava na predmetu Ekspertski sistemi (IR4ES). Namera je da se ovo postigne kom-binacijom testova sa interaktivnom demonstracijom osnovnih koncepata iz ove tematike. Administrator bi mogao kontrolisati uspešnost studenata na testovima, i postavljati nove testove i demonstracije.

Tokom prethodnih godina, izrađene su dve aplikacije u istu svrhu: Java aplikacija i aplikacija za Android mobilne uređaje. Na žalost, ni jedna ni druga ne nude mogućnost lakog dodavanja novog sadržaja, kao ni mogućnost administracije. Zbog toga što se može koristiti na svim uređajima, veb aplikacija koja je predmet ovog rada ispravlja nedostatke i proširuje skup funkcionalnosti postojećih rešenja.

Ovaj dokument je podeljen na pet celina. U glavi 1, trenutnoj sekciji, se navodi kratak opis teme i struktura dokumenta. Potom sledi sekcija 2 sa zahtevima za realizaciju, koja sadrži korisničke zahteve, kao i opis korišćenih tehnologija. Nakon toga se nalazi glava 3 u kome je detaljno opisan rad sistema, kao i način upotrebe. U ovom delu nalazi se i poveći broj slika koje ilustruju aspekte rada sistema. U pretposlednjoj glavi 4 se nalazi selekcija interesantnih izazova na koje je autor naišao tokom izrade, kao i načini na koji su ovi problemi prevaziđeni (uz prateće delove izvornog koda). Napokon, u poslednjoj glavi 5 su izložena završna posmatranja i moguće nadogradnje sistema.

# Glava 2

## Zahtevi za realizacijom sistema

### 2.1 Korisnički zahtevi

Korisnički zahtevi se mogu podeliti u dve grupe. To su:

1. Zahtevi platforme generalne namene za polaganje, ocenjivanje i administraciju testova
2. Zahtevi sistema za parsiranje, transformisanje i prikazivanje logičkih izraza

#### 2.1.1 Prva grupa zahteva

Prva grupa zahteva se odnosi na samu platformu za polaganje testova.

##### Logovanje i registracija

Kako je sistem namenjen za upotrebu od strane studenata i administratora, neophodno je implementirati logovanje postojećih korisnika na sistem i registraciju novih korisnika. Pri tom je bitno razlikovati login standardnih korisnika i administratora. Osim toga što ove dve role koriste sistem na različit način, administrator ima neke privilegije koje nisu dostupne običnom korisniku - na primer, pregledanje rezultata testova. O ovome se mora voditi računa, tako da se onemogući pristup resursima za koje su potrebne administratorske privilegije.

Logovanje treba da se vrši preko studentskog email naloga i lozinke. Novi korisnici se registruju tako što unesu email adresu, na koju se potom šalje nasumično generisana lozinka. Lozinke se od klijenta ka serveru treba da se šalju u heširanom obliku, a u istom obliku treba da se pohranjuju u bazi podataka. Administratorski nalozi se automatski ubacuju u bazu prilikom konfiguracije sistema.

Nakon što se korisnik uloguje, preko kolačića u pregledaču treba da se pamti korisnička sesija, i time izbegava ponovno, nepotrebno logovanje na sistem nakon napuštanja stranice. Sesije treba da ima određeno vreme validnosti, nakon kojeg je neophodno da se korisnik ponovo uloguje.

Korisnik treba da ima izbor da se izloguje kada poželi, i time invalidira korisničku sesiju.

## Pregled testova

Nakon uspešnog logovanja, korisnik treba da vidi spisak svih testova koje trenutno može da polaže, testova čije je polaganje u toku, i testova koji su završeni, i to grupisanih po oblastima.

Testovi koji su završeni trebaju biti onemogućeni, tj. korisniku ne sme biti dozvoljeno da više puta polaže isti test. Za ovakve testove treba naznačiti i datum polaganja i uspešnost, tj. broj pitanja na koje je student tačno odgovorio u odnosu na ukupan broj pitanja na testu.

Za testove koji su u toku, tj. čije je polaganje korisnik započeo, ali nije završio, treba naznačiti trenutni progres, odnosno broj pitanja na koja je student dao odgovor naspram ukupnog broja pitanja na testu.

## Polaganje testova

Svaki test treba da se sastoji iz određenog broja pitanja, grupisanih po stranicama. Svaka stranica, uz pitanja, može opcionalno da sadrži i *demonstraciju*, koja može biti slika, animacija, jednačina, interaktivni element itd. Postoje tri tipa pitanja:

1. pitanja sa više ponuđenih, a jednim tačnim odgovorom - odgovarajuća komponenta je radio dugme,
2. pitanja sa više ponuđenih i više tačnih odgovora - odgovarajuća komponenta je polje koje se štiklira, i

3. pitanja sa odgovorom u slobodnom stilu - odgovarajuća komponenta je tekstualno polje.

Primer jedne stranice sa pitanjima dat je na slici 2.1.

**Opšta kultura 1**

Strana 3/7

Demonstracija



Zadaci

**1.** Koliko postoji kontinenata na zemaljskoj kugli?

jedan  
 pet  
 osam

**2.** Šta od navedenog NIJE satelit?

Jupiter  
 Demetra  
 Io  
 Kalisto  
 Orfej

**3.** Kako se zove glavni grad Estonije?

Talin

[Nazad](#) [Napred & Sačuvaj](#)

Slika 2.1: Korisnički interfejs stranice za polaganje testova

Korisnik može slobodno da se kreće kroz stranice sa pitanjima. Pri tome se čuvaju uneti odgovori na pitanja. Korisnik može preskočiti neka pitanja, ali mu se tada, pre predavanja testa, prikazuje upozorenje da na neka pitanja nije dat odgovor.

Nakon što korisnik odgovori na sva pitanja sa testa, na poslednjoj stranici treba da se pojavi dugme za predavanje testa. Kada korisnik klikne na ovo dugme, test treba da bude automatski ocenjen i trajno unet u bazu. Korisnika tada treba vratiti na stranicu za pregled testova. Time je proces polaganja testa završen.

Ukoliko korisnik u bilo kom trenutku prekine polaganje testa, potrebno je omogućiti nastavljanje polaganja prilikom sledećeg logovanja korisnika. Naravno, treba restaurirati dotadašnji progres polaganja testa.

## Administratorski interfejs

Za razliku od običnih korisnika, nakon što se administrator uloguje treba prikazati pregled ocenjenih testova studenata. Za svaki odrđeni test treba prikazati naziv testa, ime studenta, datum polaganja i uspešnost. Radi lakšeg prikazivanja rezultata, treba omogućiti filtriranje po email nalogu studenta, i po svakom određenom testu.

Administrator treba da ima mogućnost da pregleda svaku instancu završenog testa, tako što mu se prikaže isti interfejs kao za polaganje testa, ali bez mogućnosti menjanja odgovora. Međutim, administrator mora imati mogućnost promene automatske ocene tačnosti odgovora, tj. za svako pitanje administrator može pregaziti automatsku odluku sistema i oceniti odgovor kao tačan, tj. netačan. Ovo je izuzetno bitno za pitanja sa odgovorom u slobodnom stilu.

Takođe, administrator treba da ima mogućnost ubacivanja novih testova u sistem

### 2.1.2 Druga grupa zahteva

Druga grupa zahteva je vezana za oblast formalne logike, i odnosi se na unos, transformaciju i prikazivanje logičkih izraza.

Za potrebe demonstracije određenih algoritama, kao i za potrebe unosa odgovora na pitanja iz ove oblasti, neophodno je omogućiti unos stavova predikatske logike. Pod ovim se podrazumevaju **dobro formirane formule** (*well-formed formulae*, skraćeno *wff*, dalje u tekstu samo formule). Formule se sastoje od sledećih elemenata:

- **logičkih promenljivih**, koje se obeležavaju malim slovima abecede, na primer  $x$  ili  $var$ ,
- **predikata ili funkcija**, koji se navode kao reči abecede sa velikim početnim slovom, praćene proizvoljnim brojem logičkih promenljivih, i/ili drugih predikata ili funkcija kao argumentima, npr.  $Na(x, y)$ ,  $MANJE(a, b)$  ili  $PlusJedan(var)$ ,
- **logičkih operatora**, poimence:
  - binarnog operatora *konjukcije*  $\wedge$ ,
  - binarnog operatora *disjunkcije*  $\vee$ ,

- binarnog operatora *implikacije*  $\Rightarrow$ ,
- unarnog operatora *negacije*  $\neg$ ,
- **logičkih kvantifikatora**, tačnije:
  - *univerzalnog* kvantifikatora  $\forall$ , i
  - *egzistencijalnog* kvantifikatora  $\exists$

I pored bogatog skupa logičkih simbola, unos formula mora biti moguć preko standardnih znakova koji se unose preko tastature, ili preko ekrana mobilnog uređaja. Osim toga, neophodno je obezbediti i proveru da li je neki arbitraran unos zaista formula. Ova provera bi se idealno mogla izvršavati uporedno sa unosom, nakon svakog unešenog karaktera.

Nezavisno od načina unosa formula, treba implementirati i renderovanje formula, sa akcentom na čitljivosti. U idealnom slučaju, formule treba prikazati u LATEX formatu.

Kada se govori o transformaciji formula, misli se na svodenje na **konjuktivnu normalnu formu** (*conjuctive normal form*, skraćeno KNF). Pre definisanja konjuktivne normalne forme, zgodno je uvesti par definicija:

- **Atom** je najprostija dobro formirana formula koja se sastoji od logičke varijable ili predikata.
- **Literal** je atom ili njegova negacija.
- **Klauzula** je skup literalala međusobno povezanih disjunkcijama.

Sada se konjuktivna normalna forma može definisati kao *konjukcija klauzula*. Iz ovoga sledi da se u KN formi ne pojavljuju kvantifikatori niti operator implikacije, i da je logički operator negacije u KN formi spušten do atomskog nivoa. Značaj KN forme je u tome što je veoma pogodna za korišćenje od strane sistema za automatsko rezonovanje (zaključivanje).

Algoritam transformacije formule u KNF sastoji se od sledećih koraka:

1. Eliminisanje implikacija

$$x \Rightarrow y \equiv \neg x \vee y$$

2. Primena DeMorgan-ovih zakona i spuštanje negacija do atomskog nivoa

$$\begin{aligned}\neg(x \wedge y) &\equiv \neg x \vee \neg y \\ \neg(x \vee y) &\equiv \neg x \wedge \neg y \\ \neg(\neg x) &\equiv x \\ \neg \forall x (P(x)) &\equiv \exists x (\neg P(x)) \\ \neg \exists x (P(x)) &\equiv \forall x (\neg P(x))\end{aligned}$$

3. Zamena egzistencijalnih kvantifikatora arbitrarnim funkcijama. čiji su argumenti sve promenljive vezane univerzalnim kvantifikatorom na tom mestu u formuli

$$\forall x \exists y (P(x, y)) \rightarrow \forall x (P(x, \Phi(y)))$$

4. Preimenovanje varijabli tako da svakom kvantifikatoru odgovara unikatna promenljiva, na primer:

$$\forall x (P(x)) \wedge \forall x (Q(x)) \rightarrow \forall x (P(x)) \wedge \forall y (Q(y))$$

5. Premeštanje univerzalnih kvantifikatora na početak formule, bez promene redosleda, na prethodnom primeru:

$$\forall x (P(x)) \wedge \forall y (Q(y)) \rightarrow \forall x \forall y (P(x) \wedge Q(y))$$

6. Spuštanje disjunkcija do atomskog nivoa (osobina asocijativnosti)

$$(x \wedge y) \vee z \equiv (x \vee z) \wedge (y \vee z)$$

7. Eliminacija konjukcija razdvajanjem operanada na posebne formule

$$\begin{aligned}\forall x \forall y \forall z ((x \vee y) \wedge (y \vee a) \wedge (z \vee a)) &\rightarrow \forall x \forall y (x \vee y) \\ &\quad \forall y (y \vee a) \\ &\quad \forall z (z \vee a)\end{aligned}$$

8. Preimenovanje varijabli tako da ne postoje formule sa istim univerzalnim kvantifikatorima, na prethodnom primeru:

$$\begin{aligned}\forall x \forall y (x \vee y) \\ \forall w (w \vee a) \\ \forall z (z \vee a)\end{aligned}$$

9. Uklanjanje kvantifikatora, na prethodnom primeru:

$$\begin{array}{c} x \vee y \\ w \vee a \\ z \vee a \end{array}$$

Kao deo zahteva druge grupe, neophodno je implementirati automatsku konverziju unesene formule u KNF, sa mogućnošću prikaza rada algoritma po koracima.

## 2.2 Opis korišćenih tehnologija

### 2.2.1 Izbor jezika

Online platforma poput izrađene veb aplikacije se može realizovati u gotovo svim višim programskim jezicima današnjice. Prednost svakako imaju jezici koji:

- su na dovoljno visokom nivou apstrakcije - na primer, podrška za automatsko oslobođanje memorije je obavezna,
- koji se mogu izvršavati na velikom broju platformi uz minimalne izmene izvornog koda - pre svega, interpreterski jezici i jezici koji se izvršavaju na virtuelnim mašinama, tj. prevode se u bajtkod,
- dolaze sa kvalitetnom i robusnom standardnom bibliotekom, i za koje postoji veliki broj dostupnih biblioteka izrađenih od strane zajednice, i
- imaju detaljnu i preglednu dokumentaciju i podršku online zajednice.

Sa ovim ne pretarano striktnim ograničenjima na umu, prirodno se nameće veliki broj popularnih jezika: Java, C#, Python, Go, Rust itd.

Međutim, kako autor tokom dužine studija nije imao priliku da izučava još neku paradigmu osim objektno-orientisanog programiranja, a vođen idejom da jedan alat nikako ne može da bude pravo rešenje za sve probleme, odluka je pala na **Closure**. Closure je moderni dijalekat LISP-a, i samim tim je funkcionalan jezik. Izvorni kod Closure programa se prevodi na Java bajtkod, koji se izvršava na Java virtuelnoj mašini<sup>1</sup>. Zbog toga Closure, pored već

---

<sup>1</sup>Osim Java bajtkoda, Closure kod je moguće pokretati i na *Common Language Runtime* mašini i JavaScript mašini.

bogatog ekosistema raznoraznih biblioteka, ima pristup svim bibliotekama pisanim u Javi. Takođe, Clojure ima veoma aktivnu zajednicu, kao i veliki broj knjiga[5][7], priručnika, tutorijala i resursa za učenje.

Jedna prednost Clojure-a nad Javom, koju će autor vremenom naučiti da veoma ceni, jeste postojanje **REPL** (*read-eval-print-loop*) funkcionalnosti. Radi se o interaktivnoj konzoli u kojoj je moguće unositi i evaluirati delove Clojure koda, veoma slično Python interpretéruru. Uz činjenicu da su većina Clojure funkcija *čiste*<sup>2</sup>, REPL omogućava unikatan način razvijanja aplikacije, gde je funkcija osnovna celina razvoja i testiranja.

Izrađena veb aplikacija se sastoji od dva snažno spregnuta dela: *frontend* sistema i *bekend* sistema. Kao što im ime kaže, frontend deo je zadužen za prezentacioni sloj i obradu unosa, dok je bekend sistem zadužen za persistenciju i serviranje sadržaja, i veći deo poslovne logike.

### 2.2.2 Bekend tehnologije

**HTTP server** *Ring*[12] je apstrakcija HTTP servera koja preko veoma jednostavnog API-ja<sup>3</sup> omogućava korisniku da se fokusira na implementaciju hendlera (tj. poslovne logike), bez ulaženja u detalje HTTP protokola. API definiše HTTP zahtev i HTTP odgovor kao dve obične Clojure mape, a hendlere kao funkcije koje imaju jedan argument (mapu zahteva) i vraćaju odgovor kao mapu. Zbog toga što Ring predstavlja apstrakciju HTTP servera, Ring veb aplikacije mogu da se distribuiraju kao WAR aplikacije i pokreću unutar standardnih kontejnera za veb aplikacije (npr. Tomcat), ili da se pokrenu same pomoću integrisanog Jetty veb servera, što je za potrebe ove veb aplikacije više nego dovoljno.

**Rutiranje** Ring API je previše jednostavan za neke iole ozbiljnije zahteve, te je zbog toga nastao *Compojure*[4], koji je Clojure biblioteka za rutiranje. Jednostavno rečeno, ova biblioteka prosleđuje određene HTTP zahteve (GET, POST itd.) za URI putanjama (tj. resursima) hendlerima koji ih obrađuju. Tako je, na primer, moguće napraviti hendler koji vraća početnu stranicu ukoliko se ka serveru uputi HTTP GET zahtev za /index.html resursom. Međutim, POST zahtevi ka istom resursu će biti odbijeni sa statusom 403 **Forbidden**. Ovaj postupak uparivanja resursa i hendlera se naziva rutiranje.

---

<sup>2</sup>Čiste funkcije uvek vraćaju isti rezultat za iste argumente, i nemaju bočne efekte.

<sup>3</sup><https://github.com/ring-clojure/ring/blob/master/SPEC>

**Šabloni** Server generiše HTML stranice na upit pomoću mašine za šablone (*template engine*). U ovu svrhu je korišćena jednostavna Clojure biblioteka pod nazivom *Hiccup*[6]. Jedinstvena je po tome što nema statičke šablonske fajlove, već DOM stablo HTML dokumenta predstavlja preko Clojure struktura podataka, na primer [:div {:id "header"} [:h1 "Header"]].

**Perzistencija** Kako korisnički zahtevi nameću, svi podaci o testovima, polaganju testova, registrovanim korisnicima itd. moraju se trajno negde smestiti, tj. moraju biti perzistentni. Standardna rešenja za perzistenciju predstavljaju relacione baze podataka. Autor se odlučio za *PostgreSQL*[11], pre svega zbog postojećeg iskustva sa ovim RDBM sistemom.

**Objektno-relaciono mapiranje** ORM je strategija povezivanja stanja objekata u memoriji sa stanjem odgovarajućih objekata u relacionoj bazi. Iako su Clojure programima dostupne Java biblioteke koje predstavljaju industrijski standard (na primer, Hibernate), autor se odlučio za laganiju varijantu - za pristup bazi se koristi JDBC drajver za Postgres i mala Clojure biblioteka zvana *YeSQL*[13]. Ova biblioteka čita SQL skripte sa upitima, i za svaki upit pravi Clojure funkciju preko kojih je moguće proslediti imenovane parametre u upit, na sličan način na koji se to radi sa *NamedQuery* objektima u *Java Persistence Language*-u. Nije razmatrana upotreba ponovnog korišćenja konekcija ka bazi (*connection pooling*). Postoje biblioteke koje nude ovu funkcionalnost (npr. *C3P0*), i koje je lako integrisati ukoliko se ukaže potreba za time.

**Parsiranje logičkih jednačina** Mesto ručne izrade leksera i parsera za unošenje logičkih jednačina, autor se odlučio da iskoristi sjajnu Clojure biblioteku *Instaparse*[8] za generaciju parsera bezkontekstnih gramatika. Ova biblioteka generiše parser na osnovu specifikacije gramatike u EBNF obliku. Pri tome podržava levu i desnu rekurziju i regularne izraze. Izlaz parsera je stablo parsiranja u obliku standardnog Clojure vektora. Instaparse može da parsira i dvosmislene gramatike, tako što vraća sekvencu svih mogućih stabala parsiranja.

**Ostalo** Za logovanje se koriste već dobro poznata rešenja u vidu *Log4j* biblioteke i *SLF4j* fasade. Konfiguracija sistema vrši se preko EDN (*Extensible Data Notation*) datoteka i biblioteke zvane *Immuconf*<sup>4</sup>. EDN je zapravo

---

<sup>4</sup><https://github.com/levand/immuconf>

podskup Clojure jezika, i predstavlja standard za serijalizaciju podataka. *Cheshire*<sup>5</sup> se koristi za konverziju Clojure podataka u JSON format, i obratno. Konačno, *Postal*<sup>6</sup> biblioteka se koristi za slanje email poruka sa lozinkom korisnicima nakon uspešne registracije.

### 2.2.3 Frontend tehnologije

*AngularJS*[2] je korišćen kao *de facto* standard za frontend JavaScript framework, u kombinaciji sa *Bootstrap*[3] CSS stilovima. Angular je uglavnom korišćen za validaciju obrazaca i POST zahteve ka serveru u pozadini, dok Bootstrap obezbeđuje podrazumevanu temu ugodnu za oko, ikonice, i fluidnu konfiguraciju elemenata stranice, koja se automatski prilagođava tipu uređaja, bilo da je to desktop ili mobilni internet pregledač. HTML stranice su u skladu sa HTML5 standardom.

Od Bootstrap-a se koristi samo CSS stil, a ostatak funkcionalnosti obezbeđuju Angular direktive za Bootstrap[1]. Kako se ne koristi ni jQuery biblioteka (Angular dolazi sa svojom laganom alternativom, *jqLite*), ovim se značajno štedi na veličini JavaScript koda koji pregledač učitava, a ne gubi se na funkcionalnosti.

Za prikazivanje logičkih jednačina koristi se JS biblioteka *KaTeX*[9], koja omogućava efikasno renderovanje jednačina u L<sup>A</sup>T<sub>E</sub>X formatu.

### 2.2.4 Alat za kompajliranje i pakovanje aplikacije

Iako se za kompajliranje i pakovanje Clojure aplikacija može koristiti Maven, *Leiningen*[10] je alatka izbora za veliku većinu Clojure projekata. Popularan je zbog izuzetno lage konfiguracije - konfiguriše se preko jednog Clojure izvornog fajla u kome se specificira ime projekta i zavisnosti, koje se podrazumevano dovlače iz Maven centralnog repozitorijuma, kao i sa <http://clojars.org>. Preko `lein` komandne alatke moguće je: pokrenuti REPL (`lein repl`), napraviti izvršnu JAR arhivu, spakovanu sa svim zavisnostima (`lein uberjar`), pokrenuti server (`lein ring server-headless`) ili glavnu metodu projekta (`lein run`), kao i još mnogo toga.

Pomenuta *uberjar* arhiva veoma je pogodna za distribuciju na ciljnog serveru veb aplikacije. Kako su sve zavisnosti spakovane, jedino što je potrebno da se

---

<sup>5</sup><https://github.com/dakrone/cheshire>

<sup>6</sup><https://github.com/drewr/postal>

veb aplikacija pokrene jeste podešeno Java izvršno okruženje.

Međutim, kako veb aplikacija ima neke dodatne zahteve (Postgres server, napravljena baza, otvoren port u podešavanjima zaštitnog zida), napravljena je Vagrant skripta koja sve ove zahteve može da obezbedi tokom podešavanja VirtualBox virtuelne instance. Dovoljno je pokrenuti `vagrant up` da bi se dobilo funkcionalno, lokalno okruženje u kojoj veb aplikacija sluša na lokalnoj mreži na IP adresi `192.168.33.10:3000`. Instanca je zasnovana na CentOS 7 operativnom sistemu.

# Glava 3

## Opis rada sistema

U ovom poglavlju se detaljno izlaže opis rada sistema iz dve perspektive: perspektive studenta - korisnika sistema, i perspektive profesora - administratora sistema.

### 3.1 Opis rada sistema iz perspektive studenta

#### 3.1.1 Logovanje na sistem i registracija

Prva stranica koja se prikazuje kada korisnik uputi pretraživač na adresu servisa jeste login stranica (URI: /login), prikazana na slici 3.1. Na ovoj stranici korisnik unosi svoju email adresu i lozinku i nakon toga se loguje na sistem pritiskom na dugme **Prijava**. Prihvataju se samo email adrese koje se završavaju sa @etf.rs. Email adresa se dinamički proverava dok je korisnik unosi, i to tako da je dugme za prijavu onemogućeno dok se ne unese korektna adresa. Pokušaj prijave korisnika koji nije registrovan, ili koji se ulogovao sa pogrešnom lozinkom se prikazuju kao greška (slika 3.2).

Novi korisnici mogu da se registruju pritiskom na dugme **Registracija**. Ta akcija ih vodi na stranicu za registrovanje novih korisnika (URI: /register, slika 3.3), koja od korisnika traži da unese email adresu. Ista ograničenja vezana za email adresu se primenjuju i ovde. Pokušaj postojećeg korisnika da se ponovo registruje rezultuje greškom. Nakon što unese email, pritiskom na dugme **Izvrši** se korisnik registruje na sistem. Nakon uspešne registracije korisniku se prikazuje stranica sa slike 3.4. Tom prilikom se na unetu email adresu automatski šalje email sa nasumično generisanom lozinkom za tog

korisnika. Sa ovom lozinkom korisnik sada može da se uloguje na sistem preko stranice za logovanje.

### 3.1.2 Stranica sa pregledom testova

Nakon uspešne prijave na sistem, korisniku se prikazuje stranica sa pregledom svih testova trenutno u sistemu (URI: /user, slike 3.5 i 3.6). Ova stranica se prikazuje i kao početna stranica ukoliko je korisnik već ulogovan na sistem, a pregledač usmeri na root URL servisa. Testovi su grupisani po kategorijama, i prikazani unutar tabele u svakoj kategoriji. Zaglavljena tabele sadrže imena kategorija i predstavljaju interaktivne linkove, koji na klik proširuju svoj sadržaj, tj. telo tabele sa testovima.

Za svaki test se prikazuje ime testa, broj stranica sa pitanjima, i ukupan broj pitanja. Testovi koji se prikazuju na ovoj stranici se dele na:

- **Završene testove**, tj. testove koje je korisnik predao. Za ove testove se dodatno prikazuju datum i vreme kada je test započet, datum i vreme kada je test završen, i ocena, u vidu procenta tačnih odgovora u odnosu na ukupan broj pitanja. Boja pozadine ovih unosa je zelena.
- **Testove u toku**, tj. testove koje je korisnik započeo, ali nije završio. Za ove testove se dodatno prikazuju datum i vreme kada je test započet, procenat od ukupnog broja pitanja na koja je student dao odgovor, tj. progres, i link ka stranici sa testom. Boja pozadine ovih unosa je žuta.
- **Nezapočete testove**, tj. testove koje korisnik nikada nije polagao. Za ove testove se dodatno prikazuje link ka stranici sa testom i prazan progres. Boja pozadine ovih unosa je bela.

Nakon što korisnik klikne na dugme **Polaži**, prikazuje mu se prva stranica sa pitanjima odabranog testa. Kada korisnik prvi put otvoriti stranicu sa pitanjima nekog testa, smatra se da je otpočeo proces izrade tog testa, te će unos u tabeli testova za taj test od tada biti označen žutom pozadinom.

### 3.1.3 Stranica sa pitanjima

URI stranice sa pitanjima je sledećeg oblika:

/user/progress/<assignment-id>/<page-ord>

gde je **assignment-id** ID zadatka, a **page-ord** redni broj stranice sa pitanjima tog zadatka. Redni brojevi stranica počinju od broja 1.

Slika 3.1: *Levo*: inicijalni izgled login stranice, *desno*: login stranica tokom unosa email adrese

Slika 3.2: *Levo*: izgled login stranice nakon što neregistrovani korisnik pokuša da se uloguje, *desno*: izgled login stranice ukoliko korisnik unese pogrešnu lozinku

Slika 3.3: *Levo*: inicijalni izgled stranice za registrovanje, *desno*: stranica za registracije nakon što već registrovani korisnik pokuša da se registruje



Slika 3.4: Izgled stranice za registraciju nakon uspešne registracije

Stranica sa pitanjima sastoji se od tri dela (slika 3.7). Prvi deo je zaglavje stranice, na kome se nalazi ime testa, redni broj trenutne stranice sa pitanjima, ukupan broj stranica sa pitanjima u testu i progres popunjavanja testa, u procentima. Progres se računa na osnovu broja popunjenih pitanja, tako da korisnik može da primeti ukoliko je slučajno preskočio neko pitanje sa prethodnih stranica pre nego što preda test.

Drugi deo stranice sa pitanjima je deo sa demonstracijom. Ovaj opcioni deo se nalazi u panelu sa leve strane i sadrži proizvoljan sadržaj. Sadržaj se može specificirati za svaku stranicu sa pitanjima ponaosob, i unosi se direktno u bazu, zajedno sa ostalim informacijama vezanim za konkretni test. Sadržaj se navodi kao deo izvornog koda u Clojure-u, koji se interpretira i renderuje zajedno sa ostatkom HTML stranice, tako da je ovim putem moguće vršiti i složenije operacije nad sadržajem koji treba prikazati.

Treći deo stranice sa pitanjima su sama pitanja. Ona se nalaze u panelu sa desne strane. Na jednoj stranici sa pitanjima moguće je imati proizvoljan broj pitanja. Pitanja su navedena jedna ispod drugog, u rastućem redosledu rednog broja pitanja. Kada se stranica učita, odgovor na svako pitanje je nepotpuni. Kada korisnik jednom odgovori na pitanja sa jednim ili više ponuđenih odgovora, više nije u mogućnosti da poništi svoj odgovor i vrati pitanje u nepotpuno stanje. Nakon svih pitanja, na dnu panela se nalaze dva dugmeta: **Nazad** i **Dalje**. Prvo dugme služi da se korisnik vrati na prethodnu stranicu sa pitanjima, a potonje dugme služi da se ode na sledeću stranicu sa pitanjima. U oba slučaja se čuvaju odgovori koje je korisnik dao na trenutnoj stranici.

Kada se korisnik nađe na poslednjoj stranici sa pitanjima, na dnu stranice pojavljuje se dugme **Predaj**. Nakon pritiska na ovo dugme korisniku se prikazuje modalni dijalog koji upozorava na to da akcija predavanja testa ne može da se poništi, dajući korisniku još jednu šansu da proveri svoje odgovore (slika 3.8). Ukoliko korisnik u pomenutom dijalogu potvrди akciju predaje, test

Formlogic Početna

## Zadaci

<a href="#">Opšte znanje</a>	(3)
<a href="#">Ekspertske sistemi</a>	(3)

Slika 3.5: Inicijalni izgled stranice sa zadacima, sa izlistanim kategorijama (broj testova unutar kategorije je naznačen sa desne strane interaktivnog linka)

Formlogic Početna

## Zadaci

Opšte znanje					
Ime	Stranica	Pitanja	Započet	Progres	Ocena
Opšta kultura 1	2	5	Nije započet	<div style="width: 0%;"></div>	<a href="#">Polaži</a>
Opšta kultura 2 - Muzika	1	3	Nije započet	<div style="width: 0%;"></div>	<a href="#">Polaži</a>
Opšta kultura 3 - Nauka	0	0	Nije započet	<div style="width: 0%;"></div>	<a href="#">Polaži</a>

<a href="#">Ekspertske sistemi</a>	(3)
------------------------------------	-----

Formlogic Početna

## Zadaci

Opšte znanje					
Ime	Stranica	Pitanja	Započet	Progres	Ocena
Opšta kultura 1	2	5	21:13:54 27/09/2016	<div style="width: 20%; background-color: #007bff;"></div> 20%	<a href="#">Polaži</a>
Opšta kultura 2 - Muzika	1	3	21:14:42 27/09/2016	Završen u 21:15:11 27/09/2016	Položen (66%)
Opšta kultura 3 - Nauka	2	4	21:14:02 27/09/2016	Završen u 21:14:19 27/09/2016	Nije položen (25%)

<a href="#">Ekspertske sistemi</a>	(3)
------------------------------------	-----

Slika 3.6: *Gore*: stranica sa zadacima nakon što je sadržaj prve kategorije proširen, za korisnika koji nije polagao nijedan test, *dole*: stranica sa zadacima sa prvim testom u toku, položenim drugim testom i nepoloženim trećim testom

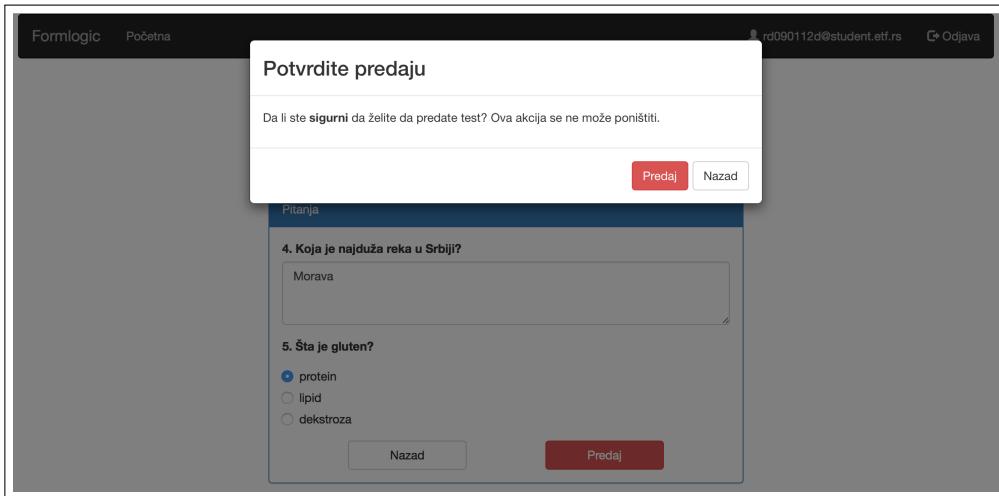
Slika 3.7: Izgled stranice sa pitanjima sa popunjениm odgovorima

se smatra završenim, a korisnik se vraća na stranicu sa pregledom testova. Od ovog trenutka korisnik više nije u mogućnosti da menja odgovore na pitanja ovog testa, niti da ga ponovo polaže.

Ukoliko korisnik prekine polaganje u bilo kom trenutku, na primer tako što zatvori prozor pregledača ili ode na neku drugu internet stranicu, može nastaviti sa polaganjem tako što će izabrati test čije je polaganje prekinuo sa stranice sa pregledom testova. Dotadašnji progres, tj. odgovori koje je korisnik dao pre nego što je prekinuo polaganje, će biti restaurirani.

### 3.1.4 Primer demonstracije: svođenje na KNF

Kao jedan primer onoga šta panel sa demonstracijom može da sadrži izrađena je demonstracija svođenja logičkog izraza u KNF formu (slika 3.9). Panel sadrži tekstualno polje u kome korisnik unosi formulu (za korišćenu sintaksu pogledati sekciju 4.1). Nakon unosa formula, pritiskom na dugme **Pošalji**, uneta formula se preko AJAX POST zahteva šalje serveru, koji je svodi na KNF i rezultat vraća u JSON obliku. Rezultat se, po koracima, renderuje kao L<sup>A</sup>T<sub>E</sub>X jednačina. Ukoliko sintaksa formule nije ispravna, korisniku se prikazuje greška, zajedno sa informacijom o problematičnom delu formule.



Slika 3.8: Izgled dijaloga za potvrdu predaje testa

Sadržaj celog panela se realizuje kao poziv Clojure funkcije `views/cnf-page`, unutar kolone sa sadržajem u odgovarajućoj tabeli (pogledati sekciju 3.2.3).

## 3.2 Opis rada sistema iz perspektive profesora

### 3.2.1 Stranica za pretraživanje završenih testova

Nakon uspešnog logovanja na sistem, administratorima se prikazuje stranica za pretraživanje i pregled odrađenih testova (URI: `/user`, slika 3.10). Ova stranica je podeljena na dva dela. Prvi deo se koristi za pretragu testova preko email naloga studenata, a drugi deo se koristi za pretragu testova po oblastima. Prelaz između ove dve stranice vrši se odabirom zališka pri vrhu strane.

Tokom unošenja email naloga, ili imena testa, prilikom pretrage, izbor se dinamički sužava, tako da je dovoljno otkucati samo mali broj karaktera kako bi se prikazao željeni izbor. Stavke koje odgovaraju upitu se prikazuju u padajućoj listi, a kroz prikazane stavke se može kretati strelicama gore i dole. Izabrana stavka iz liste se potvrđuje pritiskom na taster Enter, a može se odabrat i klikom. Nakon potvrđivanja izbora, server vraća sve odrađene testove (instance testova koje su predate, pogledati sekciju 3.1.3, i ti rezultati se prikazuju u tabelarnom obliku, ispod formulara za pretragu. Za svaki odrađen test prikazuje se ime testa, email nalog studenta, vreme početka i

Demonstracija

### Svođenje na KNF

Unesite formulu:

```
VA x { Cigla(x) => ((E y { Na(x, y) && ~Piramida(y))} && (~E y { Na(x,y) && Na(y,x) }) && (A y { ~Cigla(y) => ~Jednako(x,y))))}
```

**Pošalji**

0. Početna formula  
 $\forall x \{ Cigla(x) \Rightarrow \exists y \{ Na(x, y) \wedge \neg Piramida(y) \} \wedge \neg \exists y \{ Na(y, x) \} \wedge (\forall y \{ \neg Cigla(y) \Rightarrow \neg Jednako(x, y) \}) \}$
1. Eliminiranje implikacija  
 $\forall x \{ \neg Cigla(x) \vee \exists y \{ Na(x, y) \wedge \neg Piramida(y) \} \wedge \neg \exists y \{ Na(y, x) \} \}$
2. Spuštanje negacija do atomskog nivoa  
 $\forall x \{ \neg Cigla(x) \vee \exists y \{ Na(x, y) \wedge \neg Piramida(y) \} \wedge \forall y \{ \neg Na(y, x) \}$
3. Zamena egzistencijalnih kvant. f-jama  
 $\forall x \{ \neg Cigla(x) \vee Na(x, F21(x)) \wedge \neg Piramida(F21(x)) \wedge \forall y \{ \neg Na(y, x) \}$
4. Preimenovanje varijabli  
 $\forall x \{ \neg Cigla(x) \vee Na(x, F21(x)) \wedge \neg Piramida(F21(x)) \wedge \forall y \{ \neg Na(y, x) \}$
5. Premeštanje univerzalnih kvant. na početak  
 $\forall x \{ \forall y \{ \forall a \{ \neg Cigla(x) \vee Na(x, F21(x)) \wedge \neg Piramida(F21(x)) \wedge \forall z \{ \neg Na(z, x) \} \}$
6. Spuštanje disjunkcija do atomskog nivoa  
 $\forall x \{ \forall y \{ \forall a \{ \neg Cigla(x) \vee Na(x, F21(x)) \wedge \neg Cigla(x) \vee \neg Piramida(F21(x)) \wedge \forall z \{ \neg Na(z, x) \} \}$
7. Preimenovanje varijabli (ponovo)

Demonstracija

### Svođenje na KNF

Unesite formulu:

```
a && b => (c ||)
```

**Pošalji**

Malformatted formula! Parse error at line 1, column 16:  
`a && b => (c ||)`  
 ^  
 Expected one of:  
`"#" "[a-zA-Z][a-zA-Z]*"  
 "#S+*"  
 "#[A-Z][a-zA-Z]*"  
 "~-"  
 "("`

Slika 3.9: *Levo*: izgled panela sa demonstracijom algoritma svodenja na KNF, *desno*: izgled istog panela u slučaju greške u sintaksi formule

kraja izrade, ocena u obliku procenta pitanja na koja je student dao tačan odgovor, i, najzad, link za pregled testa (slika 3.11).

### 3.2.2 Pregled i izmena ocene završenog testa

Nakon što administrator odabere jedan od testova za pregled, prikazuje se stranica koja je identična stranici iz sekcije 3.1.3 (čak je i URI isti), sa nekoliko razlika (slika 3.12, uporediti sa slikom 3.7). Prva razlika je u tome što administratoru nije dozovljeno da menja odgovore koje je student dao - sve kontrole formulara su onemogućene. Ovim se garantuje da originalni odgovori ne mogu biti menjani nakon što student predaje test. Takođe, pri vrhu stranice se prikazuje email nalog studenta koji je predao test.

Još jedna razlika je u tome što se na ovoj stranici nakon svakog pitanja prikazuje automatski dodeljena *ocena* za to pitanje. Ocena je jedna obična Bulova vrednost koju sistem dodeljuje svakom odgovoru nakon što student predaje test, i označava da li se odgovor na pitanje prihvata kao tačan, ili

Slika 3.10: Dinamičko sužavanje izbora pretrage tokom kucanja, *gore*: primer pretrage preko email naloga studenata, *dole*: primer pretrage preko imena testa

odbacuje kao netačan odgovor. Administrator može da menja ovu vrednost za svako pitanje po svom nahođenju, i time utiče na konačnu ocenu testa, u vidu procenta uspešnosti. Kako je za pitanja sa jednim ili više ponuđenih odgovora automatsko ocenjivanje trivijalno, ova funkcionalnost se uglavnom koristi za pitanja na koje odgovor treba dati u slobodnoj formi, gde je mogućnost sistema da korektno oceni odgovor ograničena. Tada administrator može odlučiti da je student dao tačan odgovor tamo gde je sistem procenio da odgovor nije bio tačan, ili obratno.

Administrator se kreće kroz stranice sa pitanjima na isti način kao student, s tim što se na poslednjoj stranici mesto dugmeta **Predaj** nalazi dugme **Kraj**. Klik na ovo dugme administratora vodi na početnu stranicu. Bitno je napomenuti da se ovim ne menja zapisan datum predaje testa. Ovo predstavlja finalnu razliku u odnosu na funkcionalnost stranice kada joj pristupa student.

### 3.2.3 Dodavanje novih testova

Kako se dodavanje novih testova radi ubacivanjem novih redova u bazi, prvo je neophodno prikazati i objasniti entitete u bazi vezane za testove i relacije

The image contains two identical-looking tables from the Formlogic application, each titled "Pregled testova".

**Table Headers:**

Test	Započet	Završen	Ocena	Link
------	---------	---------	-------	------

**Table Data (Top Table):**

Opšta kultura 2 - Muzika	19:42:30 29/09/2016	19:42:34 29/09/2016	33%	Pregledaj
Opšta kultura 3 - Nauka	21:05:04 27/09/2016	19:42:26 29/09/2016	100%	Pregledaj
Opšta kultura 1	21:10:12 27/09/2016	19:42:16 29/09/2016	40%	Pregledaj

**Table Headers (Bottom Table):**

Student	Započet	Završen	Ocena	Link
---------	---------	---------	-------	------

**Table Data (Bottom Table):**

rk100092d@student.etf.rs	12:33:37 01/10/2016	12:33:49 01/10/2016	0%	Pregledaj
ac070324d@student.etf.rs	12:30:17 01/10/2016	12:30:32 01/10/2016	40%	Pregledaj
rd090112d@student.etf.rs	21:05:04 27/09/2016	19:42:26 29/09/2016	100%	Pregledaj

Slika 3.11: *Gore*: primer rezultata pretrage preko email nalogu studenata, *dole*: primer rezultata pretrage preko imena testa

između njih (slika 3.13).

**Assignment** je entitet na vrhu hijerarhije koji predstavlja jedan test. On sadrži ime i kategoriju testa. Primarni ključ se navodi prilikom ubacivanja u bazu.

**Task** je entitet koji predstavlja stranicu jednog testa. Ovaj entitet ima egzistencijalnu zavisnost od **Assignment** entiteta, koja se modeluje stranim ključem u vidu kolone **assignment\_id**. Sadrži redni broj unutar testa i opcionalni sadržaj panele sa demonstracijom. Primarni ključ se generiše automatski, inkrementiranjem sekvene.

**Question** predstavlja jedno pitanje u okviru stranice testa. Ima egzistencijalnu zavisnost od **Task** entiteta (kolona **task\_id**). Sadrži redni broj unutar stranice testa, tip pitanja, telo pitanja, ponuđene odgovore u vidu niza stringova, i niz celih brojeva koji označavaju indekse *tačnih* odgovora unutar niza ponuđenih odgovora. Primarni ključ se takođe generiše automatski, inkrementiranjem sekvene. Prilikom ubacivanja

Formlogic Početna ★ dimitrijer@etf.rs (administrator) Odjava

## Opšta kultura 3 - Nauka

Stranica 2 od 3 100%

Student: ac070324d@student.etf.rs

Demonstracija

Pitanja

2. Koji fizičar je otkrio zakon gravitacije?

Isak Njutn  
 Johannes Kepler  
 Tihoo Brahe

✓ Tačno ✗ Netačno

3. Za koja od ponuđenih naučnih dostignuća je dodeljena Nobelova nagrada?

Radijacija  
 Mikroskopsko pozadinsko zračenje  
 Antibiotici  
 Heliocentrični sistem  
 Struktura atoma

✓ Tačno ✗ Netačno

4. Brzina svjetlosti u kilometrima po sekundi iznosi:

200000

✓ Tačno ✗ Netačno

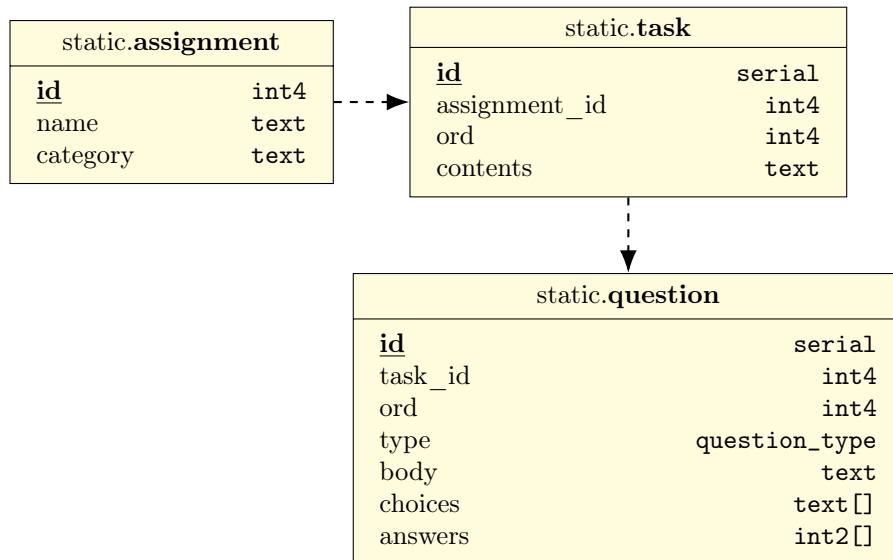
Nazad Dalje

Slika 3.12: Izgled stranice za pregled testa - izmena ocene za odgovor se vrši izborom jednog od dva radio dugmeta ispod svakog odgovora

reda se proverava da li su indeksi u koloni `answers` u validnom opsegu koji je definisan brojem elemenata u `choices` tabeli.

Prema ovome, dodavanje novih testova svodi se na ubacivanje novog reda koji predstavlja test u `assignment` tabelu, ubacivanje novog reda za svaku stranicu u testu u `task` tabelu, i ubacivanje svih pitanja po stranici u `question` tabelu. Takođe je bitno napomenuti da se sve ove tabele nalaze u `static` shemi, gde su odvojene sve tabele koje se ne menjaju tokom izvršavanja aplikacije. Primer ubacivanja testa sa slike 3.7 je priložen u listingu 3.1.

Kako bi automatsko ocenjivanje radilo i za odgovore u slobodnoj formi, za takva pitanja autor preporučuje da se u `choices` kolonu ubace tačni odgovori u što više mogućih formi, a da se u `answers` koloni navedu svi indeksi iz `choices` niza.



Slika 3.13: Entiteti koji opisuju testove i relacije među njima

```

-- Ubacuje se red za novi test.
INSERT INTO static.assignment(id, name, category)
VALUES (1, 'Opšta kultura 3 - Nauka', 'Opšte znanje');

-- Ubacuje se prva stranica.
INSERT INTO static.task(assignment_id, ord)
VALUES (1, 1);

-- Ubacuju se tri pitanja na prvoj stranici.
WITH task AS (SELECT max(id) AS id FROM static.task)
INSERT INTO static.question(task_id, ord, type, body, choices, answers)
VALUES
(task.id, 2, 'single', 'Koji fizičar je otkrio zakon gravitacije?', '{"Isak Njutn", "Johanes Kepler", "Tiho Brahe"}', '{0}::int2[]),
(task.id, 3, 'multiple', 'Za koja od ponuđenih naučnih dostignuća je dodeljena Nobelova nagrada?', '{"Radijacija", "Mikročestice", "Antibiotici", "Heliocentrični sistem", "Struktura atoma"}', '{0, 1, 2, 4}::int2[]),
(task.id, 4, 'fill', 'Brzina svetlosti u kilometrima po sekundi iznosi:', '{"300000"}', '{0}'::int2[]);

```

Listing 3.1: Primer dodavanja jednog testa sa jednom stranicom i tri pitanja

# Glava 4

## Realizacija sistema

U ovom poglavlju nalazi se par odabralih realizacija stavki koje su predstavljale veći izazov za implementaciju, uz način na koji su izazovi prevaziđeni i navedene isečke izvornog koda.

### 4.1 Uprošćavanje stabla parsiranja

Kao što je napomenuto, za parsiranje logičkih jednačina korišćen je parser bezkontekstne gramatike. U nastavku je izložena EBNF forma korišćene gramatike:

```
WellFormedFormula = QuantifiedFormula | Disjunction
QuantifiedFormula = SingleQuantifier QuantifiedFormula |
                     SingleQuantifier <'{'> QuantifiedFormula <'}'> |
                     SingleQuantifier <'{'> Disjunction <'}'>
<SingleQuantifier> = Quantifier | QuantifierNegation
Quantifier = FOREACH LITERAL | EXISTS LITERAL
QuantifierNegation = <'~'> Quantifier
Disjunction = Disjunction <'||'|> Conjunction | Conjunction
Conjunction = Conjunction <'&&'> Implication | Implication
Implication = Implication <'=>'> Term | Term
Negation = <'~'> Term
Term = <'('> WellFormedFormula <')'> | Negation | Predicate | LITERAL
Predicate = PRED <'('> Arguments <')'>
<Arguments> = Arguments <', '> Argument | Argument
Argument = LITERAL | Predicate
```

```

LITERAL = #'[a-zA-Z]+'
PRED = #'[A-Z][a-zA-Z]*'
FOREACH = <#'\\A'>
EXISTS = <#'\\E'>

```

Prvi deo gramatike čine neterminali, a drugi deo čine terminali, definisani regularnim izrazima. Elementi u uglastim zagradama se **ne pojavljuju** unutar stabla parsiranja, ali inače čine deo gramatike.

Parser kao rezultat parsiranja izbacuje stablo parsiranja u *Hiccup* formatu. Ovaj format sastoji se od ugnezđenih Clojure vektora. Svaki vektor obeležava čvor stabla. Prvi element svakog vektora jeste ključna reč koja obeležava terminal ili neterminal, a ostatak vektora su deca čvora, koja i sama mogu biti vektori. Radi pojašnjenja, daje se primer stabla parsiranja prostog logičkog izraza:

$$\forall x (\neg x \wedge y) \implies \text{Inv}(y, x) \quad (4.1)$$

U datoj gramatici, ovaj izraz se zapisuje sa `\A x {(\~x && y) => Inv(y, x)}`, a stablo parsiranja ovog izraza je:

```

[:WellFormedFormula
 [:QuantifiedFormula
  [:Quantifier [:FOREACH] [:LITERAL "x"]]
  [:Disjunction
   [:Conjunction
    [:Implication
     [:Implication
      [:Term
       [:WellFormedFormula
        [:Disjunction
         [:Conjunction
          [:Conjunction
           [:Implication [:Term [:Negation [:Term [:LITERAL "x"]]]]]]
           [:Implication [:Term [:LITERAL "y"]]]]]]]]
      [:Term
       [:Predicate
        [:PRED "Inv"]
        [:Argument [:LITERAL "y"]]
        [:Argument [:LITERAL "x"]]]]]]]]]]

```

Kao što se može primetiti, dubina ugnezđivanja za prost logički izraz je veoma velika. Ovo je posledica činjenice da se u gramatici neki neterminali mogu zameniti nekim drugim, hijerarhijski nižim neterminalnom. Na primer, startni

```

(defn- simplify-tree*
  "Simplifies hiccup format of instaparse parser tree for specified nonterm.
  This is done by removing superfluous levels of nesting from recursive
  elements. For example, [:Conjunction [:Implication [:Term]]] becomes just
  [:Term]."
[nonterm]
(fn [& children]
  (if (and (not (contains? excluded-nonterms nonterm))
            (= (count children) 1))
      ;; Return the only child.
      (first children)
      ;; Don't change the node.
      (into [nonterm] children)))

(defn simplify-tree
  "Applies simplify-tree* for all nonterms (except excluded)."
[tree]
(instaparse/transform
  ;; This makes a map of :Nonterminal -> (simplify-tree* :Nonterminal).
  (into {} (map #(assoc {} % (simplify-tree* %)) nonterms)) tree))

```

Listing 4.1: Funkcija za uprošćavanje stabla parsiranja

netermin `WellFormedFormula` može da se zameni sa `Disjunction`, a taj netermin može da se zameni sa `Conjunction`, koji može da se zameni sa `Implication`, te onda sa `Term` itd.

Kako bi se olakšala dalja obrada stabla, i izbegla nepotrebna redundantnost, implementirana je funkcija koja skraćuje dubinu stabla, tako što eliminiše čvorove koji imaju samo jedno dete, sa određenim izuzecima. Skidaju se čvorovi koji predstavljaju neterminale osim `Negation` i `QuantifierNegation`, koji po definiciji imaju samo jedno dete. U listingu 4.1 dat je izvorni kod ove funkcije.

Funkcija se oslanja na funkciju `transform` Instaparse biblioteke, koja prima mapu ključnih reči na funkcije. Ta funkcija potom prolazi kroz stablo, i menja čvorove sa rezultatima odgovarajućih funkcija. U ovom slučaju, za svaki netermin mapira se povratna vrednost funkcije `simplify-tree*`, sa tim neterminalom kao argumentom. `simplify-tree*` pravi funkciju koja vraća prvo dete čvora, ukoliko čvor ima samo jedno dete, ili vraća originalni čvor, ukoliko čvor ima više dece.

Bitno je napomenuti da, u skladu sa duhom funkcionalnog programiranja, ne postoji stanja drveta parsiranja, već svaka operacija (poziv funkcije) rezultuje novim stablom. Uprošćeno stablo parsiranja koje vraća funkcija `simplify-tree` za izraz 4.1 izgleda ovako:

```

[:QuantifiedFormula
 [:Quantifier [:FOREACH] [:LITERAL "x"]]]

```

```
(defn wff->cnf
  "Converts a well-formed formula in string form to
conjunctive-normal-form in tree form."
[formula]
(-> (logic-parser formula)
    simplify-tree
    transform-implications
    transform-negations
    transform-existential-quantifiers
    transform-universal-quantifiers
    pull-quantifiers-up
    descend-disjunctions
    split-on-conjunctions
    ;; From this point on we have a vector of formulas.
    rename-bound-vars
    remove-quantifiers))
```

Listing 4.2: Funkcija za svodenje na KNF formu

```
[:Implication
 [:Conjunction [:Negation [:LITERAL "x"]] [:LITERAL "y"]]
 [:Predicate [:PRED "Inv"] [:LITERAL "y"] [:LITERAL "x"]]]]
```

Dobijeno stablo je daleko prostije od početnog, i pogodno za dalju transformaciju. Zbog ovoga je prvi korak pri svodenju na KNF uprošćavanje stabla, što se može videti u telu funkcije `wff->cnf` (4.2). Ova funkcija svodi datu jednačinu na KNF formu i koristi *threading* Clojure makro da bi „provukla“ stablo kroz transformišuće funkcije.

## 4.2 Prilagođivanje HTML stranica za neke stusne kodove

Doslednom izgledu veb aplikacije umnogome doprinose prilagođene stranice u slučaju da resurs nije nađen (status 404) ili da se dogodila neka interna greška (status 500). U slučaju greške, takođe je korisno prikazati i potpis steka korisniku kako bi administratori imali više informacija za otklanjanje problema.

Jetty nudi podrazumevane stranice koje se vraćaju u ovakvim slučajevima, koje nisu u skladu sa izgledom izrađene veb aplikacije. Međutim, željeno ponašanje je moguće definisati preko Compojure ruta. Compojure rute se razrešavaju redom kojim su navedene. Ukoliko se nijedna ruta ne poklopi sa zahtevom, Compojure se obraća Jetty serveru, koji onda vraća 404 status sa podrazumevanom stranicom. Umetanjem posebne rute `compojure.route/not-found`,

```

;; routes.clj
(defroutes site-routes
  (GET "/" [session :session] (if (contains? session :user)
    (resp/found "/user/")
    (resp/found "/login")))
  (GET "/login" [] (views/login-page))
  (POST "/login" [email password :as r]
    (controllers/login email password r))
  (GET "/register" [] (views/register-page))
  (POST "/register" [email] (controllers/register email)))
;; wrap-routes will invoke wrapped handlers only if route matches.
;; wrap-routes #'user-routes handlers/wrap-user-session-check)
;; Custom 404 page.
(route/not-found views/not-found-page)

;; views.clj
(def not-found-page
  (page-template
    "404 - Stranica ne postoji"
    (well
      [:h1 {:class "text-warning"} "Stranica nije nađena!"]
      [:p "Tražena stranica ne postoji."]
      (button-link "/" "Početna stranica"))))


```

Listing 4.3: Implementacija prilagođene 404 stranice

koja prihvata sve zahteve, na kraj liste ruta, obezbeđeno je izvršavanje hendlera za te rute ukoliko se nijedna ruta pre nje ne poklopi sa zahtevom. Ova ruta implementira prost hendler koji vraća prilagođenu 404 stranicu. Kod je dat na listingu 4.3.

Situacija za stranicu sa greškom je malo komplikovanija, jer se ona pojavljuje ukoliko je došlo do neočekivane greške, na primer, neuhvaćenog izuzetka. Takođe, trik sa rutama ovde ne pomaže, jer se izuzeci koji su prošli van hendlera hvataju unutar Jetty koda. Ovde u pomoć uskače Compojure koncept koji se naziva *middleware*. Prosto rečeno, middleware je funkcija omotač koja prva dobija kontrolu toka, poziva obmotani hendler, a potom dobija nazad kontrolu toka kada se hendler izvrši. Ovakva funkcija ima mogućnost da promeni strukturu zahteva pre nego što se on prosledi hendleru, ili da promeni ono što je hendler vratio kao odgovor.

Vredno je napomenuti da se Compojure middleware funkcije mogu obmotavati jedna oko druge. Ovaj koncept jako podseća na dekoratore u OOP svetu.

Za potrebe prikazivanja prilagođene stranice u slučaju greške, napravljena je middleware funkcija `wrap-catch-exceptions` koja obmotava sve rute, a koja hvata sve tipove izuzetaka koji se dese u unutrašnjim hendlerima. Ukoliko se uhvati takav izuzetak, kao odgovor se vraća status 500, a kao telo se navodi string koji sadrži informacije o izuzetku, zajedno sa potpisom steka.

```

;; handlers.clj
(defn wrap-catch-exceptions
  "Middleware that catches all exceptions in business logic
  and returns a status 500 response with stack trace attached."
  [handler]
  (fn [req]
    (try (handler req)
          (catch Throwable e
            (do
              (log/error e "Unhandled exception in handler!")
              ;; We wrap exception and stack trace in response
              ;; body. This will get picked up by wrap-500.
              (resp/internal-server-error
                (str/join "\n" (conj (map str (.getStackTrace e))
                                      (.toString e)))))))

(defn wrap-500
  "Middleware that replaces all 500 responses with a custom page."
  [handler]
  (fn [req]
    (let [response-map (handler req)
          status (:status response-map)]
      (if (= 500 status)
          (-> (resp/internal-server-error
                  (views/internal-error-page (:body response-map)))
               (resp/content-type "text/html")
               (resp/charset "utf-8"))
          response-map)))))

;; views.clj
(defn internal-error-page
  "Renders a nice 500 page with error details."
  ([] (internal-error-page nil))
  ([details]
   (page-template
     "500 - Interna greška"
     (wide-well
       [:h1 {:class "text-danger"} "Interna greška!"]
       [:p "Nešto nije u redu, radimo na tome..."]
       (button-link "/" "Početna stranica"))
     (when details
       [:div {:class "im-centered-wide"}
        [:p "Detalji:"]
        [:pre (h details)]]))))

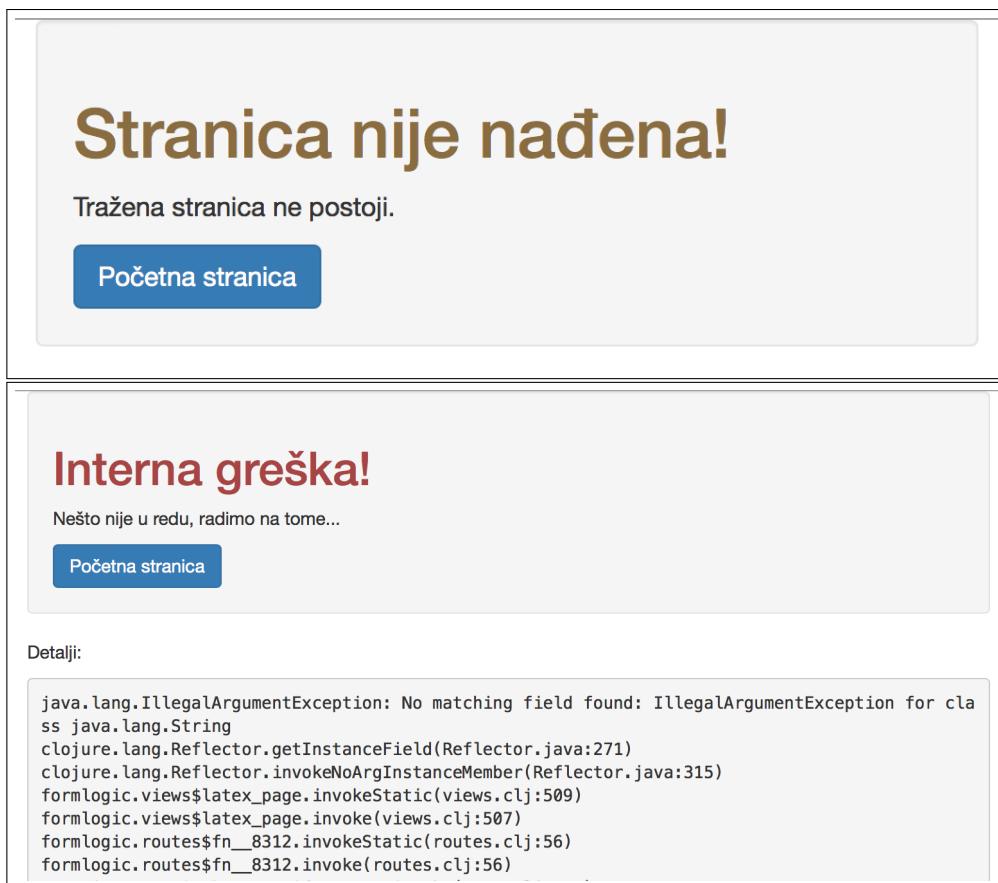
;; routes.clj
;; Main app route with all middleware wrapped.
(def app (-> #'site-routes
            wrap-json-response
            wrap-json-params
            (wrap-defaults site-defaults)
            handlers/wrap-catch-exceptions
            handlers/wrap-log-request
            handlers/wrap-500)))

```

Listing 4.4: Implementacija prilagođene 500 stranice

Pored ove funkcije postoji i `wrap-500` middleware, koji presreće sve odgovore sa statusnim kodom 500, telo menja punom HTML prilagođenom stranicom, i postavlja neophodna HTML zaglavlja. Izvorni kod je priložen u listingu 4.4, a izgled obe stranice se može videti na slici 4.1.

Autor je prvobitno napravio previd time što je hvatao samo izuzetke tipa `java.lang.Exception`, pa se ponekad desilo da „isplivaju” izuzeci koji nasleđuju klasu `Throwable`. Tada bi se prikazala standardna Jetty stranica za greške.



Slika 4.1: Izgled prilagođenih stranica za statusne kodove 404 (*gore*) i 500 (*dole*)

```

WITH
-- First CTE fetches completed assignment progresses for
-- specified assignment ID.
progress AS (SELECT * FROM assignment_progress ap
    WHERE ap.assignment_id = :id
    AND completed_at IS NOT NULL),
-- Second CTE fetches two rows for each assignment progress -
-- number of correct and number of wrong answers.
questions AS (SELECT p.id, correct, count(*) AS cnt
    FROM question_progress qp
    INNER JOIN progress p ON qp.assignment_progress_id = p.id
    GROUP BY p.id, qp.correct),
-- Third CTE sums up both rows for each assignment progress,
-- yielding total number of questions.
total_questions AS (SELECT id, sum(cnt) AS total FROM questions GROUP BY id)
SELECT p.id, p.started_at, p.completed_at, u.email,
    (100 * COALESCE(cnt::real, 0::real) / total::real)::int AS grade
FROM progress p
INNER JOIN public.user u ON p.user_id = u.id
LEFT OUTER JOIN questions q ON p.id = q.id AND q.correct
INNER JOIN total_questions tq ON p.id = tq.id
ORDER BY p.completed_at DESC;

```

Listing 4.5: SQL upit za dovlačenje završenih testova

## 4.3 SQL upit za dohvatanje završenih testova

Na početnoj administratorskoj stranici moguće je dohvatiti sve završene testove na osnovu email naloga studenta ili imena testa. Za svaki test treba sračunati i procenat uspešnosti.

Ova operacija uključuje nekoliko akcija:

- dohvatiti završene test na osnovu primarnog ključa studenta ili testa iz tabele `assignment_progress`,
- dohvatiti ukupan broj pitanja za svaki pronađeni test - ovo su statički podaci koji se nalaze u shemi `static`,
- dohvatiti ukupan broj pitanja na koja je student dao tačan odgovor iz tabele `question_progress`, i, konačno,
- izdvojiti i sračunati relevantne podatke i poslati ih nazad klijentu

Umesto pozivanja pojedinačnih upita za svaki od ovih koraka, autor se odlučio za pristup gde će svi podaci biti dohvaćeni odjednom, u svom finalnom obliku. Za ovaj podvig je korišćena funkcionalnost Postgres-a pod imenom *Common Table Expressions* (CTE). Ona omogućava pravljenje pod-upita unutar jednog upita, koje je moguće referencirati iz glavnog upita, i koristiti kao punopravne tabele. CTE izraze je takođe moguće referencirati i iz drugih CTE izraza.

Osim ovoga, podržavaju i rekurziju.

Implementirani upit dat je u listingu 4.5. Postoje tri CTE izraza koji dohvataju određene komponente neohodne za konstrukciju konačnih rezultata:

1. Prvi CTE dovlači gotove progrese (instance testa) za određeni test. Kriterijum završenog testa jeste da je kolona `completed_at` popunjena.
2. Drugi CTE za svaki red iz prvog dovlači progres za sva pitanja za tu instancu testa iz `question_progress` tabele. Ovaj upit ih nakon toga grupiše preko GROUP BY klauzule u dva reda po progresu, po jedan za svako stanje boolean kolone `correct`, i broji sve redove koji su na ovaj način grupisani. Rezultat se nalazi u `cnt` koloni. Na ovaj način su za svaku instancu testa prebrojani tačni i netačni odgovori. Drugi CTE ima ukupno  $2 \times N$  redova, gde je  $N$  broj redova u prvom CTE izrazu.
3. Treći CTE prosto sabira `cnt` kolonu za svaki progres iz drugog CTE izraza i rezultat stavlja u `total` kolonu. Ovo postiže tako što redove grupiše po ID koloni. Treći CTE ima  $N$  redova, po jedan za svaki red iz prvog CTE izraza.

Konačno, rezultat se dobija tako što se iz prvog CTE izraza pokupe ID progrusa, datum početka izrade, datum kraja izrade i email nalog studenta (ovo se zapravo dobija preko INNER JOIN-a sa `user` tabelom na koloni `user_id`). Poslednja kolona sadrži ocenu uspešnosti koja se sračunava kao količnik odgovarajuće kolone `cnt` iz drugog CTE izraza, za koju je `correct` tačan, i ukupnog broja pitanja iz trećeg CTE izraza, tj. odgovarajuće kolone `total`. Na kraju se rezultati sortiraju prema datumu predaje testa, i vraćaju klijentu u JSON formi.

Prva verzija upita je odgovarajući red iz drugog CTE izraza birala preko izraza WHERE `questions.correct = true` u glavnom upitu, a drugi CTE se stapaо sa prvim preko INNER JOIN-a. Međutim, na ovaj način su preskakani testovi na kojima studenti nisu odgovorili tačno ni na jedno pitanje. Rešenje je bilo da se koristi LEFT OUTER JOIN sa drugim CTE izrazom, uz dodatno ograničenje da se tabele stapanju samo za redove gde je `correct` kolona označena kao tačna. Ova vrsta stapanja će za testove gde nema nijedno tačno pitanje umetnuti NULL vrednosti na mesta gde fale kolone, a COALESCE unutar izraza za računanje uspešnosti će se pobrinuti da se takva vrednost pretvoriti u nulu u aritmetičkom izrazu.

## 4.4 Raspakivanje Postgres nizova

Atributi nekih entiteta u bazi su pohranjeni kao nizovi, na primer, kolone `choices` i `answers` unutar `static.question` tabele. Funkcije koje učitavaju takve entitete iz baze, generisane od strane YeSQL biblioteke na osnovu SQL skripti, vraćaju rezultate u vidu Clojure mapa. Međutim, nizovi unutar tih mapa se vraćaju kao instance `org.postgresql.jdbc4.Jdbc4Array` klase. Kao primer je naveden izlaz funkcije `db/find-questions-by-task-id`:

```
({:answers #<org.postgresql.jdbc4.Jdbc4Array@3a1205f7 {2}>,
 :body "Koje je boje nebo?",  

 :choices #<org.postgresql.jdbc4.Jdbc4Array@29ca0b4d  

 {zute,crvene,plave}>,  

 :id 1,  

 :ord 1,  

 :task_id 1,  

 :type "single"}  

{:answers #<org.postgresql.jdbc4.Jdbc4Array@242f86ce {0,1,4}>,  

 :body "Šta od navedenog NIJE satelit?",  

 :choices #<org.postgresql.jdbc4.Jdbc4Array@6e8b2491  

 {Jupiter,Demetra,Io,Kalisto,Orfej}>,  

 :id 2,  

 :ord 2,  

 :task_id 1,  

 :type "multiple"}  

{:answers #<org.postgresql.jdbc4.Jdbc4Array@441ca90a {0}>,  

 :body "Kako se zove glavni grad Estonije?",  

 :choices #<org.postgresql.jdbc4.Jdbc4Array@3df542a7 {Talin}>,  

 :id 3,  

 :ord 3,  

 :task_id 1,  

 :type "fill"})
```

Problem nastaje ukoliko se nad vrednostima polja koja predstavljaju pomenute instance pokuša pozvati metoda `getArray()`. Po JDBC dokumentaciji<sup>1</sup>, tada dolazi do materijalizacije elemenata niza, tj. tada se oni zapravo dovlače iz baze. Međutim, ukoliko se ova metoda pozove van JDBC transakcije, baca se izuzetak.

Rešenje ovog problema jeste da se vrednosti unutar JDBC niza raspakuju

---

<sup>1</sup><https://docs.oracle.com/javase/tutorial/jdbc/basics/array.html>

```

;; db.clj
(defn unwrap-arrays
  "Unwraps Jdbc4Array instances into arrays. Needs to be called
  within a transaction."
  [m]
  (into {} (for [[k v] m]
              [k (if (instance? org.postgresql.jdbc4.Jdbc4Array v)
                     (vec (.getArray v))
                     v)])))
  ;; When calling db functions, pass it as :row-fn
(db/find-questions-by-task-id {:task_id 1}
                               {:row-fn db/unwrap-arrays})

```

Listing 4.6: Funkcija za raspakivanje vrednosti JDBC nizova unutar transakcije

dok još traje transakcija. YeSQL ima mogućnost da se funkcijama koje predstavljaju upite prosledi funkcija za transformisanje redova. Ova funkcija se izvršava nad svakim dovučenim redom pre nego što se transakcija zatvori, i vraća rezultat obrade. Stoga je implementirana funkcija `db/unwrap-arrays`, priložena u listingu 4.6, koja prolazi kroz sve unose mape koja predstavlja jedan red iz tabele i raspakuje JDBC nizove, ukoliko nađe na njih.

Kada se pomenuta `db/find-questions-by-task-id` funkcija pozove sa funkcijom za transformisanje redova `db/unwrap-arrays`, na mestu JDBC nizova dobijaju se standardni Clojure vektori, unutar kojih su raspakovane vrednosti:

```

{:answers [2],
 :body "Koje je boje nebo?",
 :choices ["zute" "crvene" "plave"],
 :id 1,
 :ord 1,
 :task_id 1,
 :type "single"}
{:answers [0 1 4],
 :body "Šta od navedenog NIJE satelit?",
 :choices ["Jupiter" "Demetra" "Io" "Kalisto" "Orfej"],
 :id 2,
 :ord 2,
 :task_id 1,
 :type "multiple"}
{:answers [0],
 :body "Kako se zove glavni grad Estonije?",
 :choices ["Talin"],
```

```
:id 3,  
:ord 3,  
:task_id 1,  
:type "fill"})
```

## 4.5 *Anti-forgery* token

Kao deo standardnih *middleware* funkcija, Compojure obezbeđuje i zaštitu od *Cross-site Request Forgery* napada. Mete ovih napada su HTML obrasci koji izvršavaju POST zahteve ka serveru. Ukratko, napad se ogleda u tome da ulogovani korisnici mimo svoje volje izvrše neke akcije na serveru. Ovo se radi tako što se na stranom sajtu napravi obrazac čiji `action` atribut gađa resurs na sajtu mete. Nakon toga se metodama socijalnog inženjeringu (na primer, skraćeni link ka stranom sajtu napadač šalje ulogovanom korisniku preko čet poruke) ulogovani korisnik navede da izvrši `submit` akciju na stranom formularu. Kako je korisnik autentifikovan, server prihvata akciju kao validnu, te je izvršava.

Na primeru izrađene aplikacije, navedeni napad bi, na primer, primorao korisnika da preda test pre nego što je test gotov. Ili bi na silu započeo izradu nekog testa bez saglasnosti korisnika.

Zaštita od ovakvih napada se vrši tako što se na svim obrascima sajta generiše skriveno polje koje sadrži nasumice generisanu vrednost zvanu *anti-forgery token*. Ovaj token se prilikom `submit` akcije šalje sa ostalim parametrima obrasca. Server prilikom obrade zahteva proverava da li vrednost tokena koju je klijent poslao odgovorara vrednosti tokena generisanog za tu sesiju, i izvršava zahteva **samo ako** se vrednosti poklapaju.

Ovim se efektivno eliminiše pretnja od CSRF napada. Međutim, ovo takođe znači da će svi POST zahtevi ka serveru koji nemaju token biti odbijeni, pa čak i oni koji nisu rezultat `submit` akcije nekog formulara. Ovo je otežavajuća okolnost za izradu pristupnih resursa API-ja za, na primer, dovlačenje izrađenih testova na administratorskoj stranici, ili dovlačenja imena studenata čiji email sadrži unete karaktere (ovaj zahtev se koristi za `autocomplete` funkcionalnost na administratorskoj stranici).

Kao rešenje su korišćena dva pristupa: korišćenje GET zahteva tamo gde je moguće podatke poslati kao URL parametre, i generisanje CSRF tokena na stranicama koje nemaju formular, a koriste POST zahteve iza scene. Vrednost tokena se tada navodi u standardnom `X-CSRF-Token` zagлављу HTTP zahteva.

```

var myApp = angular.module('myApp', ['ui.bootstrap', 'ngSanitize']);

// Form controller.
myApp.controller('LatexFormController', ['$scope', '$http',
  function($scope, $http) {
    $scope.formula = "";
    // Find AF token on the page.
    $scope.antiForgeryToken =
      angular.element(document.querySelector('#__anti-forgery-token')).val();
    $scope.result = {};
    $scope.error = null;
    $scope.sendFormula = function() {
      formulaData = {
        formula : $scope.formula
      };
      $http({
        method: 'POST',
        url: '/latex',
        data: formulaData,
        headers: {
          "X-CSRF-Token" : $scope.antiForgeryToken
        }
      }).then(function successCallback(response) {
        $scope.result = response.data.result;
        $scope.error = null;
        kate.render(response.data.result,
                    document.getElementById("katexEquation"));
      }, function failedCallback(response, status) {
        $scope.error = response.data.error;
      });
    };
  }]);

```

Listing 4.7: JavaScript funkcija za slanje unete jednačine na server

U listingu 4.7 je naveden JavaScript kod koji demonstrira opisanu tehniku, a koristi se za slanje unetih jednačina na server.

## 4.6 Moguće nadogradnje sistema

Postoji značajan broj aspekata aplikacije koji bi se u budućnosti mogli unaprediti:

- Korisnički interfejs za unos novih testova. Jedini način dodavanja novih testova trenutno jeste preko SQL upita, što je dosta nezgodno jer zahteva pristup serveru i pristup bazi. Ovaj način dodavanja ispita je takođe podložan greškama.
- Vremensko ograničenje za izradu testa. Studentima je trenutno omogućeno da počnu sa izradom testa, i da sama izrada traje neodređeno dugo. Jedno od bitnijih unapređenja sistem jest da se postavi vremenski rok za polaganje testa, koji bi se prikazivao studentima dok odgovaraju na pitanja, i nakon koga bi se test automatski predao, tj. završio. Osim vremenskog intervala u kome je neophodno predati test, bilo bi zgodno definisati i vremenski period u kome je polaganje testa uopšte moguće, na primer za vreme ispitnih ili kolokvijumskih rokova.
- Administracija grupa studenata. Ovim bi se omogućilo raspoređivanje studenata po arbitarnim grupama, na primer, po predmetu. Neki testovi bi mogli da se polažu samo od strane studenata koji pripadaju određenoj grupi. Grupe bi mogle da označavaju i studentske godine, te bi student mogao da bude član i više različitih grupa. Studenti bi mogli da zahtevaju da budu primljeni u grupu, ili bi moglo da se implementira automatsko dodavanje studenata u neke grupe, na primer na osnovu broja indeksa iz email naloga. Još jedan benefit grupisanja studenata je lakše pregledanje testova.
- Paginacija za rezultate testova. Trenutno se svi testovi prikazuju na istoj stranici, čak iako se vrati veliki broj rezultata. Unapređenje predstavlja implementacija paginacije, sa recimo 50 rezultata po stranici.
- Podrška za *Secure Socket Layer* protokol. Korisnici se loguju na sistem tako što šalju email nalog i lozinku kao deo POST zahteva kao *urlencoded* parametri obrasca. Prostim prisluškivanjem mrežnog saobraćaja moguće je prikupiti sve kredencijale tokom procesa logovanja, i to iskoristiti za lažnu autentifikaciju. Na Ring serveru je relativno lako omogućiti OpenSSL podršku, tako da je sav saobraćaj između korisnika i sajta

enkriptovan. Time bi se kanal komunikacije sasvim dovoljno osigurao. Za ovu funkcionalnost bi trebalo obezbediti validan sertifikat izdat od strane priznatog autoriteta za sertifikate.

- Bolji heš algoritam za skladištenje lozinki. Ukoliko bi maliciozno lice dobilo pristup bazi, moglo bi sa lakoćom da izvrši *brute-force* napad, i da sa dovoljno velikim rečnikom uspe da dode do lozinki drugih korisnika, zajedno sa email nalozima. Trenutno se za skladištenje lozinki koristi MD5 heš vrednost lozinke. Time što bi se koristila bolja funkcija za heširanje, na primer SHA256, ili bcrypt, značajno bi se umanjio rizik od ovakve vrste napada. Takođe, treba imati u vidu da je ovakvu vrstu napada moguće izvršiti i bez upada u bazu, tako što bi se slao veliki broj login zahteva ka serveru. Ovaj rizik se može umanjiti ograničavanjem ukupnog broja zahteva u nekom vremenskom intervalu po sesiji.
- Opciono pamćenje sesije u kolačićima na određeno vreme. Ova funkcionalnost, poznatija kao „zapamti me”, omogućava korisnicima da se samo jednom uloguju na sistem, tj. da ne moraju da unose email i lozinku svaki put. Zbog bezbednosnih razloga je potrebno implementirati i vremensko ograničenje, tj. zastarivanje sesije. Trenutno se na sistem može ulogovati jednom, i ostati neodređeno dugo ulogovan, tj. do trenutka kada se korisnik ručno odjavи ili do kada se restartuje server. Ukoliko korisnik isključi opciju „zapamti me”, npr. na deljenim računarima, sistem bi automatski izlogovao korisnika čim zatvori stranicu u pregledaču.
- Skalarno ocenjivanje odgovora. Trenutno se odgovori mogu označiti kao tačni, ili kao netačni. Često se dešava da profesor proceni da je student delimično odgovorio na pitanje. Stoga je neophodno omogućiti profesoru da oceni odgovor sa određenim brojem poena. Ovim bi se takođe otvorila mogućnost definisanja broja poena po pitanju, tj. neka pitanja bi mogla da vrede manje, a neka više poena.
- Nove funkcionalnosti sistema za baratanje logičkim jednačinama. Ovde se misli na implementacije različitih strategija zaključivanja.

# Glava 5

## Zaključak

Kao konačni rezultat rada realizovana je veb aplikacija za polaganje testova od strane studenata i administraciju testova od strane profesora. Studentima se ovim omogućava polaganje testova u hodu, a profesorima lakše ocenjivanje i izrada novih testova.

Izrađeni sistem ima svega oko 2000 linija koda, od čega je 1500 linija Clojure izvornog koda, raspoređenih u 9 fajlova. Modularan je, tako da se lako nadograđuje. Lako se kompajlira, pakuje i distribuira, ili kao samostalna aplikacija ili kao kontejner za veb server. Ceo projekat je okačen na *GitHub*-u na adresi <https://github.com/dimitrijer/formlogic>. Izvorni kod je objavljen pod Eclipse javnom licensom, te predstavlja softver otvorenog koda.

Autor je veoma zadovoljan krajnjim rezultatom, imajući u vidu da mu je to prvi projekat pisan u Clojure jeziku. Funkcionalno programiranje nudi jedan sasvim drugačiji aspekt u odnosu na OOP paradigmu. Potrebno je neko vreme kako bi odviklo od pisanja sekvenčnog koda, sa kontrolnim strukturama kojima se kontroliše tok izvršavanja, sa eksplisitim iteriranjem, klasama, promenljivama, stanjem objekata itd. Autoru je najteže bilo da se liši koncepta perzistentnog stanja. Međutim, nakon nekog vremena počinju da se ukazuju neke prednosti funkcionalnog programiranja. Čiste funkcije postaju izuzetno moćan alat za enkapsulaciju i apstrakciju. Komponovanjem ovakvih funkcija uz pomoć elementarnih operacija `map`, `reduce`, `apply`, `comp`, `partial`, `threading` makroa itd. moguće je postići izuzetno kompleksne operacije, a opet zadržati čitljivost koda, i jasno odvojene odgovornosti na nivou funkcija. Clojure kolekcije (vektori, liste, rečnici, setovi) su izuzetno zgodne za rad. Iako svaka operacija nad kolekcijama vraća novu kolekciju, one su veoma performantne, zahvaljujući optimizacijama ispod haube, te se prilikom pisanja koda uopšte ne obraća pažnja na „skupljanje smeća”. Lenje sekvene, koje se

realizuju samo kada se pristupa elementima, omogućavaju apstraktne termine poput beskonačnih nizova, ili beskonačnih poziva funkcija. Rekurzija takođe postaje moćno oružje: Clojure ima podršku za repnu rekurziju pomoću `recur` operacije, pa nema bojazni od prekoračenja steka. Kako su sve vrednosti nepromenljive (*immutable*), to programer većinu vremena ne mora da razmišlja o mehanizmima sinhronizacije. Kada je neophodno obezbediti atomičnost nad dodelom vrednosti, postoje proste CAS (*compare and set*) primitive zvane atomi.

Autor želi da napomene da još uvek nije imao dovoljno prakse da bi stekao jasnu percepciju svih prednosti i mana ovakvog koncepta programiranja. Takođe, postoje koncepti u Clojure jeziku koje autor nije imao prilike da upotrebi: multimetode (dispečovanje na osnovu tipa argumenta), protokoli (slični interfejsima), rekordi (strukture podataka), makroi (veoma moćan koncept koji omogućava manipulisanje samim elementima jezika, tj. samo-modifikovanje koda), transakcije (implementacija *Software Transactional Memory* sistema svojstvena Clojure-u, koja obezbeđuje korektnost i konzistentnost podataka prilikom obrade od strane više niti) itd. Jedno je sigurno, a to je da se autor veoma raduje budućem radu u ovom programskom jeziku.

# Reference

- [1] *Angular directives for Bootstrap*. URL: <https://angular-ui.github.io/bootstrap/> (visited on 09/21/2016).
- [2] *AngularJS. Superheroic JavaScript MVW Framework*. URL: <https://angularjs.org> (visited on 09/21/2016).
- [3] *Bootstrap. The world's most popular mobile-first and responsive front-end framework*. URL: <http://getbootstrap.com> (visited on 09/21/2016).
- [4] *Compojure - A concise routing library for Ring/Clojure*. URL: <https://github.com/weavejester/compojure> (visited on 09/20/2016).
- [5] Chas Emerick, Brian Carper, and Christophe Grand. *Clojure Programming*. O'Reilly Media, Inc., 2012. ISBN: 1-4493-9470-1.
- [6] *Hiccup. Fast library for rendering HTML in Clojure*. URL: <https://github.com/weavejester/hiccup> (visited on 09/22/2016).
- [7] Daniel Higginbotham. *Clojure for the Brave and True. Learn the Ultimate Language and Become a Better Programmer*. No Starch Press, 2015. ISBN: 1-5932-7591-9.
- [8] *Instaparse. Context-free Grammer Parser Generator*. URL: <https://github.com/Engelberg/instaparse> (visited on 09/22/2016).
- [9] *KaTeX. Fast math typesetting for the web*. URL: <https://github.com/Khan/KaTeX> (visited on 09/21/2016).
- [10] *Leiningen. For automating Clojure projects without setting your hair on fire*. URL: <http://leinigen.org> (visited on 10/02/2016).
- [11] *PostgreSQL: The world's most advanced open source database*. URL: <https://www.postgresql.org> (visited on 09/20/2016).
- [12] *Ring. Clojure HTTP server abstraction*. URL: <https://github.com/ring-clojure/ring> (visited on 09/20/2016).
- [13] *YeSQL. A Clojure library for using SQL*. URL: <https://github.com/krisajenkins/yesql> (visited on 09/22/2016).