

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike

Filip Ožbolt

RAZVOJ WEB PLATFORME ZA UGOVARANJE

STRUČNE PRAKSE – BACKEND

Dokumentacija

Pula, Rujan, 2020. godine

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike

FILIP OŽBOLT

RAZVOJ WEB PLATFORME ZA UGOVARANJE

STRUČNE PRAKSE - BACKEND

Dokumentacija

Kolegij: Poslovni informacijski sustavi

Mentor: doc. dr. sc. Darko Etinger

Komentor: dr. sc. Nikola Tanković

Pula, Rujan, 2020. Godine

Sadržaj

Uvod.....	2
Funkcionalnosti	2
Kostur projekta	3
Struktura backenda.....	5
Baza podataka	5
Struktura korisničkog sučelja	5
Neautorizirani korisnik	6
Student.....	9
Poslodavac/partner	20
Admin.....	22
Struktura baze podataka (screenshots)	25
Reference:	28

Uvod

Vođeni idejom olakšavanja i optimiziranja ugovaranja stručne prakse na FIPU te poboljšanja iskustva kako za studente tako i za poslodavce, kolega Stjepan i ja smo odlučili napraviti web aplikaciju "Moja-praksa". Kolega je radio frontend dio, a ja pripadajući backend. Web aplikacija se sastoji od 4 sučelja ovisno o stupnju autorizacije pa tako imamo sučelje za: neautorizirane korisnike, studente, partnere odnosno poslodavce i admina. Na kraju možemo zaključiti da smo prilično zadovoljni s napravljenim funkcionalnostima, ali da bi u sljedećem poboljšanju svakako trebao biti neki chat između studenta i poslodavca te studenta i admina kako bi se i taj dio ugovaranja koji se obavlja preko maila(korisnicima bi samo automatski dolazile obavijesti na email da imaju poruku na "Moja-praksa" jer poslodavci sigurno neće posjećivati web aplikaciju na dnevnoj bazi) prebacio na "Moja-praksa" te bi se time cjelokupno ugovaranje prakse moglo obaviti putem naše aplikacije.

Funkcionalnosti

- Neautorizirani korisnici: Pregled dostupnih projekata i poslodavaca
- Studenti: Pregled dostupnih projekata i poslodavaca, Upute za obavljanje prakse, odabir projekata, uvid u rezultat dodjele projekata, uvid u vlastiti i tuđi tijekom obavljanja prakse(student ne vidi tuđa imena nego samo JMBAG), popunjavanje prijavnice, preuzimanje templatea dnevnika i predaja popunjenog dnevnika, uvid u vlastiti profil i izmjena podataka, brisanje korisničkog računa ili promjena lozinke
- Poslodavci/partneri: Pregled dostupnih projekata i poslodavaca, dodavanje projekata, mijenjanje postojećih projekata, dodavanje slika i logotipa na vlastiti profil i projekte, izmjena vlastitih podataka, brisanje korisničkog računa ili projekata
- Admin: Pregled dostupnih projekata i poslodavaca, alokacija studenata na određeni projekt, popis/pregled studenata te njihovih prijavnica i dnevnika, tablica studenata koja pruža brži uvid u status i obaveze studenata, dodavanje partnera i njihovih projekata, dodavanje uputa i templatea za dnevnik, brisanje/mijenjanje partnera i projekata(samo onih koje je admin i napravio)

Kostur projekta

Za izradu projekta odnosno backenda web aplikacije sam koristio editor Visual Studio Code, a sam backend pokreće platforma Node.js, dok osnovu backenda aplikacije čine njegovi npm paketi nodemon, babel i express.js. Kao bazu podataka smo koristili cloud servis MongoDB.

Prvo sam naredbom „npm init“ inicijalizirao package.json u direktorij koji sam napravio za backend. Spomenuti file je neizostavan dio svake Node aplikacije jer se u njemu, između ostalog, nalazi zapis dependency-a i skripta za pokretanje aplikacije.

Na slici 1 vidimo popis ostalih paketa koje sam koristio u sklopu izrade backenda, svih ćemo se postepeno dotaći kada ćemo doći do dijela koda gdje su nam metode iz tih paketa koristile i olakšale funkcioniranje aplikacije.

```
15  "dependencies": {
16    "bcrypt": "^5.0.0",
17    "cors": "^2.8.5",
18    "dotenv": "^8.2.0",
19    "express": "^4.17.1",
20    "jsonwebtoken": "^8.5.1",
21    "mongodb": "^3.5.9"
22  },
23  "devDependencies": {
24    "@babel/core": "^7.10.4",
25    "@babel/node": "^7.10.4",
26    "@babel/preset-env": "^7.10.4",
27    "nodemon": "^2.0.4"
28  }
```

Slika 1 Package.json - dependencies

Pojasnimo malo taj „kostur“ naše aplikacije odnosno čemu služe osnovni paketi ovog projekta

Express.js -> open-source framework za node.js. Najvažnije što omogućava je „hendlanje“ HTTP upita

Babel -> Pretvara novije ES6 sintakse u pripadajuće „starije“ sintakse i time omogućava da svaki browser može pokrenuti našu aplikaciju i „razumijeti“ sav kod.

Nodemon -> Služi tome da se prilikom izmjene filea automatski resetira aplikacija

```

6   "scripts": {
7     "serve": "nodemon --exec babel-node src/index.js",
8     "test": "echo \"Error: no test specified\" && exit 1",
9     "server": "node src/index.js --exec babel-node",
10    "heroku-postbuild": "npm install",
11    "start": "babel-node src/index.js"
12  },

```

Slika 2 Package.json - scripts

Važnost Nodemona i Babela možemo vidjeti i na slici 2 koja prikazuje skriptu za pokretanje naše aplikacije. Vidimo da su Nodemon, a zatim Babel oni koji pokreću našu aplikaciju. Točnije, definirali smo da se upisivanjem naredbe „npm run serve“ pokreću ti paketi koji učitaju naš glavni file indexs.js i „dižu“ aplikaciju.

Tako je napravljen kostur aplikacije, ali ovo se još ne može zvati web aplikacija. Za to trebamo napraviti nešto čemu se može pristupiti preko web browsera (barem lokalno). Za to sam kreirao index.js file koji će biti glavni file i preko kojeg će dolaziti i odlaziti svi upiti s frontenda.

Da bi upiti funkcionirali trebamo instancirati express.js aplikaciju i pridodati joj port. Često se za backend port koristi port 3000, ali mi smo je podesili da bude dinamička i da se učitava iz environment variable kako ne bi hardcodiranje izazvalo problema sa deployanjem na Heroku koji sam dodjeljuje portove koji nisu fiksni.

```

1  import dotenv from 'dotenv'
2  dotenv.config();
3
4  import express from 'express';
5  import bodyParser from 'body-parser';
6  import routes from './routes';
7  import cors from 'cors'
8  import auth from './auth.js'
9
10
11  const app = express()
12  const port = process.env.PORT;
13
14  app.use(cors())
15
16  app.use(bodyParser.json({limit: '50mb', extended: true}))
17  app.use(bodyParser.urlencoded({limit: '50mb', extended: true}))

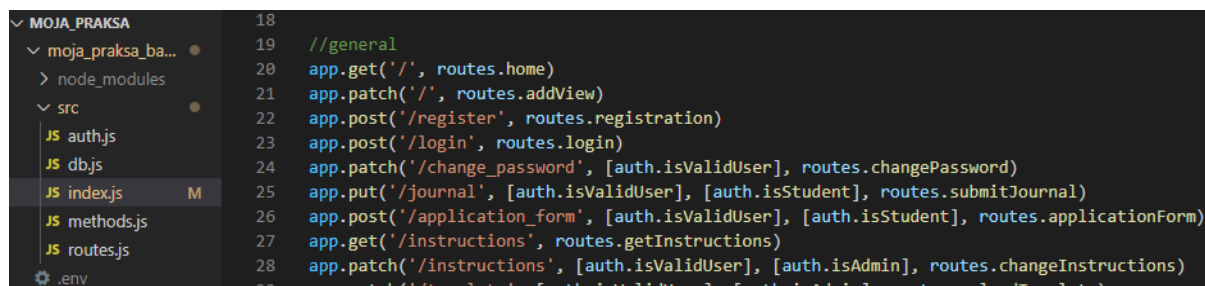
```

Slika 3 Index.js - pokretanje aplikacije

Aplikacija je zatim pripremljena i čeka upite na backend pomoću metode listen()

Za spomenuti je još sa slike 3 da sam npm paket cors koristio za omogućavanje nesmetane komunikacije između frontenda i backenda, a, iako express.js ima u sebi parser, koristio sam bodyParser jer nam je trebao parser koji može obraditi veće fileove od ovih koje express.js prihvaća.

Struktura backenda



Slika 4 Struktura backenda i neke od ruta na index.js

Backend se sastoji od pet fileova koje možemo vidjeti na slici 4. Uz spomenuto, u index.js se nalaze HTTP rute odnosno requesti dok se njihovi handleri nalaze u routes.js. Svrha filea methods.js je odvojiti funkcije od ruta i handlera radi dekompozicije koda i preglednosti, a možda najbitniji razlog zašto nisu pisane u handlerima je smanjenje redundancije koda.

Baza podataka

Rekli smo da svi upiti s frontenda dolaze na index.js, no treba spomenuti i komunikaciju backenda s bazom podataka. Naravno, svakoj web aplikaciji treba neka vrsta trajne pohrane podataka. Kao našeg database providera koristili smo MongoDB u cloudu. Razlog tome je što nam je lakše raditi i više smo upoznati s JSON-om i Javascript objektima nego s radom preko neke SQL baze. Pohrana i dohvaćanje iz Mongo kolekcija i dokumenata nam je iz tih razloga puno jednostavnija.

Struktura korisničkog sučelja

Moja-praksa se sastoji od 4 sučelja ovisno o stupnju autorizacije pa tako imamo sučelje za: neautorizirane korisnike, studente, poslodavce i admina. Pošto su u uvodu već navedene funkcionalnosti svakog tipa korisnika, neću ih nabrajati nego ću proći kroz backend dio tih funkcionalnosti i objasniti ga.

Neautorizirani korisnik

```
629   async getProjects (req, res) {
630
631     let query = req.query
632     let atributi = ["company", "technologies", "location", "project_description"]
633
634     let result = await methods.search(query, atributi, 'projects')
635
636     res.json(result)
637   },
638
639
640
641   async getPartners (req, res) {
642
643     let query = req.query
644     let atributi = ["company", "about_us"]
645
646     let result = await methods.search(query, atributi, 'partners')
647
648     res.json(result)
649   },
```

Slika 5 routes.js – getProjects i getPartners

Prva od rijetkih stvari kojima neregistrirani korisnici imaju pristup jest razgledavanje projekata i partnera. Handleri za rute imaju hardcodane attribute po kojima se može pretraživati projekt ili partner. Funkcionira tako da ako je query prazan odnosno ako s frontenda ne dolaze nikakvi podaci za pretragu, funkcije vraćaju sve projekte ili partnere, a ako se u query nalazi neki podatak, query podaci se zajedno s atributima proslijeđuju u metodu na slici 6. Doduše, mogli smo kod partnera staviti više atributa za pretragu. Na primjer, po tehnologijama koje koriste.


```

144 search : async (query, atributi, collectionName) =>{
145     let db = await connect()
146
147     let selekcija = {}
148
149     if(query._any || collectionName === 'users'){
150         let pretraga = query._any
151
152         if (collectionName === 'users'){
153             pretraga = pretraga + ' Student'
154         }
155
156         let terms = pretraga.split(' ')
157         if (!query._any) terms.shift()
158         console.log('terms:',terms)
159
160         selekcija = {
161             $and: []
162         }
163
164
165         terms.forEach((term) => {
166             let or = {
167                 $or: []
168             };
169
170             atributi.forEach(atribut => {
171                 or.$or.push({ [atribut]: new RegExp(term, "i") });
172             })
173
174             selekcija.$and.push(or);
175         });
176
177     }
178
179 }
180
181 let cursor = await db.collection(collectionName).find(selekcija).sort({company: 1})
182
183 let results = await cursor.toArray()
184
185 results.forEach(doc => {
186     doc.id = doc._id
187     delete doc._id
188
189     if (collectionName == 'users') delete doc.password
190 })
191 return results

```

Slika 6 methods.js - search

Ono što nisam spomenuo je da pretraživanje studenata (dostupno kod admin interfeceja) ima poseban tretman. Naime, ako se pretražuju studenti, automatski se hardcoda i traženi pojam „student“ koji se nalazi u atributu `account_type`. Funkcionalnost ostaje ista jer se mogu dobiti svi studenti ili neki određeni pojmom pretrage. Bolji način za to izvesti bi bio pomoću `$and` i `$or` operatora u `forEach` petlji i kompozitnih upita, ali nisam uspio pa sam se odlučio za ovaj način.

Kompozitni upit u principu radi tako da za sve pojmove koji korisnik pretražuje, svaki pojam pretraži u svakom hardcodanom atributu i tako u svakom dokumentu određene kolekcije. Na kraju remapiramo `id` i brišemo heshiran `password` iz dokumenata studenata ako smo njih pretraživali

Ono što jedino neautorizirani korisnik još ima pristup su rute pojedinih projekata i partnera gdje korisnici mogu naći detaljnije informacije o istom. Funkcije se sastoje od jednostavnih upita i kod je praktički isti.

```
384   async getOnePartner (req,res) {
385
386     let id = req.params.id
387     let db = await connect()
388
389     try{
390       let result = await db.collection("partners").findOne({_id: ObjectId(id)})
391
392       result.id = result._id
393       delete result._id
394
395       if (!result.id) throw new Error('id is undefined')
396
397       res.json(result)
398     }
399     catch(e){
400       res.json({error: e.message})
401     }
402
403   },
```

Slika 7 routes.js – getOnePartner

```
334   async getOneProject (req,res) {
335     let id = req.params.id
336
337     let db = await connect()
338
339     try{
340       let result = await db.collection("projects").findOne({_id: ObjectId(id)})
341
342       result.id = result._id
343       delete result._id
344
345       res.json(result)
346     }
347     catch(e){
348       if (id == null) res.json({error: 'id is undefined'})
349
350       else res.json({error: e.message})
351     }
352   },
```

Slika 8 routes.js – getOneProjects

Kada korisnik pristupi ruti „app.get('/partners/:id', routes.getOnePartner)“ poziva se i ruta „app.get('/partner_projects/:id', routes.getPartnerProjects)“ jer se na frontendu u PartnerInfo.vue nalazi i popis odnosno kartice projekata koje su u vlasništvu poslodavca na čijem smo profilu(slika 9).

```

355   async getPartnerProjects (req,res) {
356
357       let partnerID = req.params.id
358       let db = await connect()
359
360       let cursor= await db.collection("projects").find({partnerID: ObjectId(partnerID)})
361       let results = await cursor.toArray()
362
363       res.send(results)
364
365   },

```

Slika 9 routes.js – getPartnerProjects

Student

Student se mora, kao i svi korisnici koji žele biti autorizirani za ono što žele, prvo registrirati i zatim ulogirati pa ću backend dio postupka ovih procesa prikazati ovom prilikom. Nebi bilo dobro da baš svaki posjetitelj stranice može jednostavno na registraciju i registrira se kao student ili poslodavac. Zbog toga smo se odlučili da uz registraciju, korisnik mora priložiti entry code odnosno pristupni kod. Kod može biti usmeno predan studentima ili poslodavcima od strane profesora. Naravno da kod može završiti u pogrešnim rukama i da će se nedobrodošli korisnici ipak uspjeti probiti, ako ne s kodom onda bez njega jer tko želi naći će način, a ova aplikacija nema ionako nikakvih sigurnosnih dodataka. No, i ovime se znatno smanjuje mogućnost da se neki nasumični posjetitelj odluči iz dosade registrirati. Daljnji stup obrane bi bio admin koji može obrisati nepoželjne korisnike.

```

444   async registration (req, res) {
445       let newUser = req.body.new_user;
446       let entryCode = req.body.registrationCode
447
448       try {
449           if (entryCode !== process.env.entry_code) throw new Error("Wrong entry code")
450
451           let user = await auth.register(newUser);
452           let result
453
454           //dodavanje korisnika automatski u partnere čim se registrira
455           if (newUser.account_type == 'Poslodavac') result = await methods.addPartner(user)
456
457           res.json({status: `user with id ${result} added`})
458
459       } catch (e) {
460           res.status(500).json({
461               error: e.message,
462           });
463       }
464   },

```

Slika 10 routes.js - registration

Traženi pristupni kod koji je korisnik upisao dolazi s frontenda i uspoređuje se s našim stringom pohranjenim u environment varijablama. One su pohranjene lokalno u procesima pripadajuće aplikacije i „stvaraju se“ prilikom pokretanja iste. Više o prednostima environment varijabla nešto kasnije.

Handler za registraciju nam osim za prosljeđivanje potrebnih podataka za registraciju u auth.register, služi , kako i piše u komentaru koda, za automatsko dodavanje novostvorenog korisnika u kolekciju partnera. Na slici 11 vidimo detaljnije kako ide taj postupak.

```
28  async function register(userData){
29
30      for (const [key, value] of Object.entries(userData)) {
31          if(!value){
32              res.json({status: 'Missing data'})
33              return
34          }
35      }
36
37      let db = await connect()
38
39      let partner = {}
40
41      let user = {
42          email: userData.email,
43          password: await bcrypt.hash(userData.password, 8),
44          date_created: Date.now(),
45      }
46
47      if(userData.account_type == 'Admin') user.account_type = userData.account_type
48
49      if(!user.account_type){
50          /* kod registracije automatski sortiramo podatke u "user" i "partner" ...
51          ...kako bi kada kreiramo usera, automatski kreirali i partnera(ako je user partner)*/
52          if(userData.jmbag){
53              user.account_type = 'Student',
54              user.jmbag = userData.jmbag,
55              user.name = userData.name,
56              user.surname = userData.surname,
57              user.technology = userData.technology
58              user.year= userData.year
59              user.journalID = false
60          } else{
61              user.account_type = 'Poslodavac',
62              partner.company = userData.name,
63              partner.technology= userData.technology,
64              partner.adress = userData.adress,
65              partner.about_us = userData.about_us,
66              partner.date_created = Date.now(),
67              partner.contact_email = userData.contact_email,
68              partner.telephone_number = userData.telephone_number,
69              partner.img_url = 'https://images.unsplash.com/photo-1493119508027-2b584f234d6c',
70              partner.account_type = 'Poslodavac'
71          }
72      }
73  }
```

Slika 11 auth.js register 1.dio

Proces registracije se može nastaviti izvoditi dalje od početne provjere samo ako svaki atribut sadrži vrijednost. Ako je taj test prošao, svakom tipu autoriziranog korisnika(student, poslodavac, admin), se u objektu 'user' pridodaju atributi koji su zajednički svakoj vrsti korisnika. Naravno, admin se ne može registrirati kao običan korisnik, taj proces ću objasniti kasnije, no ovaj dio im je isti.

Šifru kako je i običaj nismo spremali na bazu u izvornom obliku. Mislim da nije potrebno objašnjavati zašto. Tu nam priskače u pomoć bcrypt paket koji nam između ostalog, pomoću metode hesh(), heshira lozinku sa sha-512 algoritmom odnosno ulaz pretvara u izlaz koji je nerazumljiv i probojan jedino brute-force napadom kojemu bi trebalo par desetljeća da probije neku srednje kompliciranu lozinku, a i do par stoljeća za složeniju lozinku¹. Prvi parametar hash() funkcije je lozinka u čistom obliku, drugi parametar označava broj random transformacija koje će se izvesti nad našom lozinkom uz to što će se lozinka heshirati. To omogućava da isti hash bude na različitim stranicama različito transformiran, pa se ne može upasti svugdje korisniku koji svugdje koristi tu istu lozinku.

Za jedinstvenost korisničkim imena odnosno u našem slučaju bi bilo ispravnije reći emaila smo indexirali na mongodb bazi atribut email i smjestili je izvan scopea funkcije da se ne poziva svaki put nego samo pri pokretanju aplikacije(slika 28), smatramo da su na FIPU svi mailovi jedinstveni te da možemo email imati kao jedinstveno korisničko ime bez stvaranja problema.

Dalje se, ako je korisnik admin, preskače cijeli novi niz remapiranja koji se tiče studenta i poslodavca. Što se tiče studenata i poslodavaca, atributi od oba dva koja se tiču podataka korisničkih podataka spremaju se u objekt 'user' i insertaju se na bazu u kolekciju 'user', dok se podaci usko vezani uz partnera spremaju u pripadajući objekt te se šalju nazad na registration u routes.js gdje se poziva metoda addPartner iz methods.js te ona poziva metodu pushData koja dodaje partnera u bazu. Atributi su prilično „straightforward“, možda treba jedino reći da se slika hardcoda čisto iz razloga da partner ima neke slike koje će se prikazivati preko Vue komponente partner_card.vue dok ne uploada svoje. Kraj procesa registracije prikazan je na slici 12(dolje) koji se sastoji od klasičnog insertanja u bazu

¹ How long would it take to crack your password?, July 1, 2019, Team SpyCloud, <https://spycloud.com/how-long-would-it-take-to-crack-your-password/>

```

74
75   try{
76     let insertResult = await db.collection('users').insertOne(user);
77
78     if(insertResult && insertResult.insertedId){
79       delete user.password
80       partner.userID = ObjectID(insertResult.insertedId)
81       return partner
82     }
83   }
84   catch(e){
85     if (e.name == "MongoError" && e.code == 11000){
86       throw new Error("User already exists")
87     }
88   }
89
90
91 }

```

Slika 12 auth.js -register 2.dio

Jednom kada korisnik ima stvoren račun, može se prijaviti u aplikaciju i autentificirati se. Autentifikacija funkcionira na principu da traži na bazi korisnika s pripadajućim emailom, i ako ga nađe uspoređuje unesenu lozinku s onom heshiranom na bazi (slika 13), dok se nakon toga autorizacija prije pristupa određenim rutama provjerava pomoću middleware funkcija koje nam omogućava express.js. Odlučili smo se za način s jwt autentifikacijom jer je korištenje tokena za tu funkcionalnost moderan način koji danas koristi većina stranica, a i radili smo ga na kolegiju „Web aplikacije“ pa smo upoznati s njim. Dobar je sa korisnikove perspektive odnosno za user experience jer se ne treba logirati svaki put kada „posjeti“ neku web stranicu(zamislamo da korisnik 10 puta na dan posjeti neku stranicu i svaki puta se treba ulogirati) i dobar je s sigurnosne perspektive za obje strane, kako korisnika tako i ljudi odgovornih za sigurnost aplikacije jer ne šaljemo svaki put podatke pa nam presretnu promet Dakle, kada se jednom ulogiramo dobijemo token(pravo pristupa) na određenih x dana i toliko dana se ne treba ulogiravati. Iznimka je kada se korisnik sam „izlogira“ tj. odjavi. Onda se „gubi“ token, a pri sljedećoj prijavi dobiva novi.

```

449   async login(req,res) {
450     let user = req.body
451
452     try{
453       let result = await auth.authenticateUser(user.email, user.password)
454       res.json(result)
455     }
456     catch(e){
457       res.status(403).json({error: e.message})
458     }
459   },

```

Slika 13 routes.js - login

```

97 //referenca: prof. Tanković
98 v async authenticateUser(email,password){
99     let db = await connect()
100     let user = await db.collection("users").findOne({email : email})
101
102     // provjerava da li je "čista lozinka" ista kao izvedeni hesh u bazi izveden te lozinke
103 v if(user && user.password && (await bcrypt.compare(password, user.password))){
104     //šifra za potpisivanje korisnika(kriptografski potpis) vežemo je uz naš backend(JWT_SECRET), s tom šifrom potpisujemo...
105     //...tokene svih korisnika, kad nam korisnik vraća token provjeravamo da li je on potpisan s našom šifrom
106     delete user.password
107 v let token = jwt.sign(user, process.env.JWT_SECRET, {
108         algorithm: "HS512",
109         expiresIn: "1 week"
110     })
111     user.token = token
112
113     return user

```

Slika 14 auth.js - authenticateUser

Vratimo se na uspoređivanje lozinke. Metoda `compare()` iz `bcrypta` je ta koja nam olakšava i omogućava provjeru da li se čisti password koji dolazi s frontenda podudara s onim heshiranim u bazi i vraća `true/false`. Ako je sve lozinka dobra, `jwt` metodom `sign()` stvaramo token tako da kod kriptografskog potpisivanja pružimo šifru koja se nalazi u environment varijabli i s njom potpisujemo tokene svih korisnika. U tokenu se drži sve ono što nam je potrebno da kroz rad aplikacije dobijemo podatke o tom korisniku. I našem slučaju: `email`, `account_type`, `date created`, i `id`. Na slici(15) se vidi kako nam middleware može koristiti i kako dohvaćamo te spomenute podatke.

```

121 //referenca: prof. Tanković
122 async isValidUser(req,res, next){
123     try{
124         let authorization = req.headers.authorization.split(' ')
125         let type = authorization[0]
126         let token = authorization[1]
127
128         if (type != 'Bearer'){
129             res.status(401).send()
130             return false;
131         }
132         else {
133             //podaci spremljeni u jwt se mogu koristiti na bilo kojem mjestu -> npr da se zna ko šalje upit
134             req.jwt = jwt.verify(token, process.env.JWT_SECRET)
135             return next()
136         }
137     }
138     catch(e){
139         return res.status(401).send()
140     }
141 },
142
143 },
144
145 },

```

Slika 15 auth.js - isValidUser

IsValidUser smo stavili na svaku rutu za koju je potrebna bilo kakva autorizacija. Funkcija dohvaća token koji se nalazi u headeru² HTTP requesta pod pod objektom 'authorization' koji spremamo u varijablu i razdvajamo na dva dijela pa nam ostaju token i tip autorizacije koje ćemo provjeriti. Prije nego se preko .env varijable provjeri da li je token ispravan, treba provjeriti da li je uopće tip autorizacije ispravan. Ako je token autorizacija prošla, verify() nam vraća podatke o korisniku pa ih kroz „req.jwt“ možemo koristiti u svakoj ruti koja sadrži ovaj middleware. Razlog zašto „JWT_SECRET“ spremamo u environment set varijabli je da ju držimo izvan koda. Tome nam služi npm paket dotenv. On omogućava da napravimo jednu dot.env datoteku koja ne ide na Github i da učitavamo u kod iz tih varijabli prilikom pokretanja aplikacije.

```
189   async isAdmin(req, res, next){
190       let accountType = req.jwt.account_type
191
192       try{
193           if (accountType === 'Admin' ) return next()
194           /*za rute na kojima je isAdmin middleware prisutan, autoriziran je samo admin...
195           ... ali iznimka je ruta getStudents za putanju /TableOfStudents kojoj ima pristup i student*/
196           else if(accountType === 'Student' && req.route.path == '/students' && req.route.methods.get == true ) return next()
197
198           else {
199               res.status(401).send()
200               return false
201           }
202       }
203       catch(e){
204           return res.status(401).send()
205       }
206   },
```

Slika 16 auth.js - isAdmin

Još imamo nekoliko middlewarea koji se pokreću nakon isValidUser, a koji će se pokrenuti ovisi o tome koji tip korisnika je autoriziran za koju rutu. Pošto su rute isAdmin, isStudent i isPartner praktički iste, dosta je prikazati jednu (slika 16).

Najvažnije funkcionalnosti za studenta su odabir projekata, rezultati dodjele, popunjavanje prijavnice i preuzimanje templatea te predaja dnevnika prakse. Prikazat i objasniti ću neke od njih.

² JSON Web Token (JWT) explained, Dec 13, 2018, <https://flaviocopes.com/jwt/>


```

482   async submitChosenProjects (req, res) {
483     let data = req.body
484     let db = await connect()
485
486     // struktura na bazi : {first_priority: [id1, id2, id2], second_priority:[...], third_priority:[...]}
487     let selectedBy = {
488       first_priority : [],
489       second_priority : [],
490       third_priority : []
491     }
492     //destrukcija strukture
493     let entries = Object.entries(selectedBy);
494
495     try{
496       for(let [index, [key, value]] of entries.entries()){
497
498         let projectID = data.selection[index]
499         key = 'selected_by.' + key
500
501         await db.collection('projects').updateOne( { _id: ObjectID(projectID) }, { $addToSet: { [key] : data.user } })
502       }
503     } catch(e){
504       res.json({error: e.message, status: "Error during submitting chosen projects"})
505     }
506   }
507
508   res.json('success')
509 },

```

Slika 17 routes.js - submitChosenProjects

Kada student odabire projekte, oni se spremaju u listu na frontendu, a kada se konačno odluči poslati svoja tri odabira, oni se spremaju na bazu u pripadajuće dokumente u kolekciji 'projects'. Prvotno sam napravio da se odabrani i alocirani projekti spremaju u dokument određenog studenta u kolekciji 'users' što je nekako logičniji odabir, no kasnije smo odlučili da se ovako više isplati zbog uklapanja s ostalim funkcionalostima i CRUD operacijama tih ostalih funkcionalnosti. Kada student odabere tri projekta, njihovi id-jevi dolaze na backend poredani u listi – prvi id predstavlja prvi prioritet i tako dalje. Prema tom redosljedu u petlji u objekt s poljima spremamo id-jeve studenata odnosno userID-jeve u pripadajuće polje. Kada admin dodijeli studentu projekt, studentov id se sprema u atribut `allocated_to` te se tako dohvaća studentu onaj projekt koji mu je alociran pomoću handlera za rutu `getApprovedProjects` koji neću prikazivati jer se sastoji samo od jednostavnog mongo upita. Kriterij po kojem će se dodjeljivati praksa nismo mi napravili, nego ćemo to prepustiti adminu odnosno profesoru.

```

264   async applicationForm (req, res) {
265
266       let formData = {
267         id : req.body.userID,
268         application : req.body.form,
269         updateDoc : true
270       }
271
272       let obj = req.route.methods
273       formData.method = Object.keys(obj).toString()
274
275
276       let db = await connect()
277
278       let appExists = await db.collection('users').find( { _id: ObjectID(formData.id) , application: { $exists: true} } )
279
280       try{
281         if (appExists == true) throw new Error("Error accured during inserting")
282
283         let result = await methods.changeInfo(formData, 'users')
284
285         res.send(`${result} at inserting application.`)
286       }
287       catch(e){
288         res.status(500).json({ error: e.message});
289       }
290     },
291   },
292 }

```

Slika 18 routes.js – applicationForm

Nakon što je studentu dodijeljen projekt i student dogovori datum početka prakse i ostale uvjete, student mora popuniti prijavnicu i poslati ju adminu odnosno profesoru. Student može samo jednom poslati prijavnicu. Još treba primjetiti atribut updateDoc u objektu formData koji se postavlja na true i šalje se zajedno s objektom kao prvi parametar u funkciju changeInfo.

```

71   changeInfo : async (data, collectionName) => {
72
73       let filteredData = methods.filterData(data)
74       let result, id
75
76       if (!filteredData._id){
77         id = filteredData.id
78         delete filteredData.id
79       }else{
80         id = filteredData._id
81         delete filteredData._id
82       }
83
84       let db = await connect();
85
86       try {
87         if (filteredData.updateDoc=== true && filteredData.method == 'put') {
88           delete filteredData.updateDoc
89           delete filteredData.method
90           result = await db.collection(collectionName).replaceOne( { _id: ObjectID(id) }, filteredData);
91         }
92
93         else if (filteredData.updateDoc=== true && filteredData.method == 'patch') {
94           delete filteredData.updateDoc
95           delete filteredData.method
96           result = await db.collection(collectionName).updateOne( { _id: ObjectID(id) }, { $set: filteredData, });
97         }
98
99         else result = await db.collection(collectionName).deleteOne( { _id: ObjectID(id) } )
100       }
101
102       catch(e){
103         console.log(e)
104       }
105
106       if (result.modifiedCount == 1 || result.deletedCount == 1) return 'success'
107       else return 'fail'
108     },
109 }

```

Slika 19 methods.js - changeInfo

Tu funkciju sam napravio kada sam uvidio da se kod svakog update requesta prema bazi ponavlja jedan prilično sličan dio koda i da nema smisla taj dio koda imati u svakoj ruti zasebno kada imamo toliko update operacija prema bazi. ChangeInfo može obraditi put, patch, i delete requeste, a svaki se aktivira ovisno o metodi iz koje dolazi request i o tome da li je atribut updateDoc postavljen na true ili false. Vidimo da se na početku spomenute funkcije poziva funkcija filterData. Ona se brine da se na bazu ne spremaju nikakve prazne vrijednosti.

```
10  filterData : (data) => {
11      /*pošto se sve mandatory vrijednosti provjeravaju da nisu undefined...
12      ... na frontendu, ova funkcija dodaje false vrijednosti non mandatory atributima*/
13      for (const [key, value] of Object.entries(data)) {
14          if(!value && key !== 'views'){
15              data[key] = false
16          }
17      }
18      return data
19  },
20  },
21  }
```

Slika 20 methods.js - filterData

Što se tiče getJournalTemplate tj. slanja predložka dnevnika prakse s baze na frontend, to je običan query pa ga nećemo objašnjavati, ali treba reći da se predložak sprema u kolekciju 'content' gdje se nalaze svi takvi podaci koji nemaju pripadnost određenom entitetu nego su generalni za sve korisnike. Tako u kolekciji 'content' za sada imam samo jedan dokument gdje uz predložak nalazi i objekt koji sadrži upute za obavljanje prakse pod atributom 'instructions'. Za predaju dnevnika prakse odabral smo način da student može više puta predati dnevnik, a ako ga preda nakon roka, to će biti jasno vidljivo jer kod inserta u bazu pohranjuje i vrijeme predaje(slika 21).

```

296   async submitJournal (req,res) {
297     let data = {
298       userID : ObjectID(req.body.user_id),
299       journal : req.body.journal,
300       upload_date : Date.now()
301     }
302
303     let db = await connect()
304     let journal = await db.collection('users').findOne({_id: ObjectID(data.userID)})
305     let newJournal
306
307     try{
308       /* ako želimo da korisnik ne može više puta uploadati dnevnik */
309       //let checkUser = await db.collection('users').findOne({_id : ObjectID(data.userID)})
310       //if (checkUser.journalID != false) throw new Error("Error accured during inserting")
311
312       if (journal){
313         newJournal = await db.collection('journals').replaceOne({_id: ObjectID(journal._id), data})
314       }
315       else {
316         newJournal = await db.collection('journals').insertOne(data)
317       }
318
319       //ubacuje jos journalID u usera da budu dvostruko povezani
320       try {
321         let user = {
322           id : ObjectID(data.userID),
323           journalID : journal.insertedId,
324           updateDoc : true
325         }
326
327         let obj = req.route.methods
328         user.method = Object.keys(obj).toString()
329
330         await methods.changeInfo(user, 'users')
331
332         res.json(true)
333         //res.json({message: 'upload successful'})
334       }
335
336       catch(e){
337         res.send('Error accured during connecting journal ID with user')
338         return false;
339       }
340
341     }

```

Slika 21 routes.js -submitJournal

Za pohranu dnevnika smo napravili posebnu kolekciju 'journals' zbog toga što svaki od dokumenata odnosno dnevnika ima i po par megabajta, pa bi bilo užasno ne efikasno povlačiti svaki dnevnik za svakog usera svaki puta kada trebamo nešto iz kolekcije 'users'. Nadalje, nismo znali što će biti optimalnije za daljnje operacije, staviti id dnevnika u dokument studenta u kolekciju 'users' ili staviti id studenta u kolekciju 'journals' stoga smo napravili oboje pa se kasnije lošiji način samo izbrisati.

Student ima također uobičajene mogućnosti profila kao što su promjena vlastitih podataka, promjena lozinke i brisanje računa.

```

61   async changeUserInfo (req, res) {
62     let userInfo = req.body
63     let obj = req.route.methods
64     userInfo.method = Object.keys(obj).toString()
65     let validated = false
66     let response
67
68     // kod brisanja računa se provjerava autentičnost korisnika
69     if (userInfo.updateDoc == false) validated = await methods.checkPassword(userInfo)
70
71     // ako je prošla provjera lozinke ili ako se radi samo o updejtju profila
72     if (validated || userInfo.updateDoc == true) response = await methods.changeInfo(userInfo, 'users')
73
74
75     res.send(response)
76   },

```

Slika 22 routes.js -changeUserInfo

Promjena korisničkih podataka i brisanje računa se opet opet odvijaju preko metode changeInfo, no prije toga se, ako je riječ o brisanju, kroz metodu checkPassword koju korisnik mora unijeti prije brisanja računa. Tu nam opet u pomoć priskače još jedna metoda s bcrypta – compare() koja nam olakšava uspoređivanje čiste, transparente lozinke s onom heshiranom u bazi. Proces promjene lozinke je isti, samo što se još na bazi updejta atribut password pa ga ne treba dodatno prikazivati.

```

204   checkPassword : async (userData) =>{
205
206     let id, user
207
208     if (userData._id == null){
209       id = userData.id
210     }else{
211       id = userData._id
212     }
213
214     let db = await connect()
215
216     // kod partnera id dolazi iz kolekcije "partners" pa ga treba prepisati s vrijednošću userID
217     if (userData.account_type == 'Poslodavac' || userData.account_type == 'Admin') id = userData.userID
218
219     user = await db.collection("users").findOne({_id : ObjectId(id)})
220
221     try{
222       if (user && user.password && (await bcrypt.compare(userData.password, user.password || userData.password == true))){
223         return true
224       }
225       else{
226         return false
227       }
228     }
229     catch(e){
230       return false
231     }
232
233   }

```

Slika 23 methods.js - checkPassword

Poslodavac/partner

Najvažnije funkcionalnosti poslodavca su: dodavanje, izmjena i brisanje projekta.

```
626   async addProject (req, res) {
627
628     let project = req.body
629
630     /* pušteno ovako u slučaju da se imena atributa razlikuju pa je lakše promijeniti, ali za sada ne treba
631     let projectData = req.body
632     let project = await methods.mapAttributes(projectData)
633     */
634
635     project.views = 0
636     project.date_created = Date.now()
637     project.img_url = "https://images.unsplash.com/photo-1504610926078-a1611febcbad3?ixlib=rb-1.2.1&ixid=eyJh
638
639     //brisanje atributa koji su prazni kod inicijalizacije projekta da shodno tome ne aktivira validateData
640     if (!project.selected_by) delete project.selected_by
641
642
643     try{
644       //insert projekta
645       let result = await methods.pushData(project, 'projects')
646
647       //dohvati partnera i ako partner ima uploadane slike, prenesi ih na projekt
648       let db = await connect()
649       let partner = await db.collection("partners").findOne({_id: ObjectID(project.partnerID)})
650
651       if(partner.headers || partner.logo){
652
653         let addedProject = {
654           id : result,
655           updateDoc : true,
656           method : 'patch'
657         }
658         if (partner.headers) addedProject.headers = partner.headers
659         if (partner.logo) addedProject.logo = partner.logo
660
661         await methods.changeInfo(addedProject, 'projects')
662       }
663
664
665       res.send(`project with id ${result} added.`)
666
667     }
668     catch(e){
669       res.status(500).json({ error: e.message});
670     }
671   },
```

Slika 24 routes.js - addProject

Prije sam koristio funkciju za remapiranje iz methods.js, ali sada nam ne treba pošto su nazivi svih atributa usklađeni. Uz regularne podatke koji dolaze iz frontenda, poslodavcu još pridodajemo brojač posjeta, datum kreiranja i hardcodamo neku univerzalnu sliku projekta(desktop računalo) dok ne promijeni u neku po vlastitom izboru. A zatim se zove funkcija pushData (slika 25) koja dalje hendla insertanje projekta. Ne želimo da stari projekti koje je napravio partner imaju pripadajuće slike poslodavca, a novi projekt da nema, zato dohvaćamo partnera koji je vlasnik projekta i ako on ima sliku, postavljamo ih i u novopečeni projekt, a id novog projekta nam je vratila funkcija PushData.

```

25  ✓ pushData : async (data, collectionName) => {
26      let filteredData = methods.filterData(data)
27
28      let db = await connect()
29
30      try{
31
32          //projektu pridodajemo partnerID radi lakšeg mapiranja i rada s podacima
33      ✓  if(collectionName === 'projects' ) {
34          filteredData.partnerID = ObjectID(filteredData.partnerID)
35          delete filteredData.userID
36      }
37
38
39      let insertResult = await db.collection(collectionName).insertOne(filteredData);
40      let id = insertResult.insertedId
41
42
43      if(id) return id
44      else throw new Error("Error accrued during inserting project or partner")
45
46      }
47
48
49  ✓  catch(e){
50      console.error(e.name + ': ' + e.message)
51      }
52  },

```

Slika 25 methods.js pushData

```

518  async changePartnerInfo (req, res) {
519
520      let partnerInfo = req.body
521      let id
522
523      if (partnerInfo._id == null){
524          id = partnerInfo.id
525          delete partnerInfo.id
526      }else{
527          id = partnerInfo._id
528          delete partnerInfo._id
529      }
530
531      delete partnerInfo._id;
532      partnerInfo.id = req.params.id;
533      partnerInfo.userID = ObjectID(partnerInfo.userID)
534
535      let obj = req.route.methods
536      partnerInfo.method = Object.keys(obj).toString()
537
538      let partnerTemp, response, result
539
540      let db = await connect()
541
542      //provjera lozinke u slučaju da je zahtjev brisanje partnera
543      let validated = false
544      if (partnerInfo.updateDoc == false) validated = await methods.checkPassword(partnerInfo)
545
546
547      // dohvacanje partnera kako bi preko userID-a obrisali i usera ako je API metoda delete + brisanje
548      if (!partnerInfo.updateDoc) { // ili req.route.methods == 'DELETE'
549          partnerTemp = await db.collection("partners").findOne({_id: ObjectID(id)})
550          partnerTemp.updateDoc = false
551
552          try {
553              await db.collection("projects").deleteMany( { partnerID : ObjectID(partnerTemp._id) } );
554          } catch (e) {
555              console.log (e);
556          }

```

Slika 26 routes.js - changePartnerInfo 1.dio

changePartnerInfo je malo složenija, i ne baš spretno napisana funkcija. Ona updatea podatke partnera odnosno podatke poslodavca, ako je riječ o izmjeni logotipa ili header slika, ona tu izmjenu radi na svim partnerovim projektima. Uz to, ako je riječ o brisanju računa te ako je prošla provjera lozinke, briše podatke o partneru, a zatim isto tako i korisnički račun od tog poslodavca i sve poslodavčeve projekte. Brisanje pojedinog projekta se odvija preko funkcije changeProjectInfo tako da updateDoc atribut postavimo na false. Promjena korisničkih podataka se ne odvija preko funkcije changeUserInfo kao kod studenata nego također preko funkcije na slikama 26 i 27.

```
559 // prenesi headere i logo partnera na njegove projekte ako ih ima te ako se radi o updeajtu
560 if (partnerInfo.headers || partnerInfo.logo && partnerInfo.updateDoc == true) {
561     try {
562         await db.collection("projects").updateMany({partnerID : ObjectID(partnerInfo.id)}, {$set: {
563             headers: partnerInfo.headers,
564             logo: partnerInfo.logo}
565         })
566     }
567     catch(e) {res.send('Error accured during updating project headers')}
568 }
569
570 //hardcodamo opet defaultnu sliku za svaki slučaj ako partner nema nikakvog logotipa
571 if (partnerInfo.image_url) partnerInfo.image_url = "https://images.unsplash.com/photo-1493119508027-2b584f234d6c?ixlib=rb-1.2
572
573
574 // ako je prosla provjera lozinke ili ako se radi samo o updeajtu profila, onda se poziva metoda za promjenu podataka na bazi
575 if (validated || partnerInfo.updateDoc == true) {
576     partnerInfo.id = id
577     response = await methods.changeInfo(partnerInfo, 'partners')
578 }
579 else return false
580
581
582 // briše se i user ako je user izbrisao svoj partner profil
583 if(response == 'success' && partnerTemp && partnerTemp.created_by_admin != true){
584     partnerTemp._id = partnerTemp.userID
585     result = await methods.changeInfo(partnerTemp, 'users')
586 }
587 else result = response
588
589
590 res.send(result)
591 },
```

Slika 27 routes.js - changePartnerInfo 2.dio

Admin

Prije nego idemo u funkcionalnosti, moramo objasniti kako se uopće stvara korisnički račun s admin privilegijama. Za to nam je trebao način izrade koji je automatski, koji ne prosljeđuje čistu lozinku i prije svega koji se ne odvija klasično na frontendu. Dosjetio sam se omotane asinkrone funkcije s auth.js koja se poziva svaki put kada se spajamo na bazu i koja nam je prvotno služila da se korisnikov email indexira. A što se tiče lozinke, spremljena je u environment varijablu te se samo od tamo učitava u atribut 'password'. Nakon što imamo sve podatke, pozivamo istu funkciju – register kao i za ostale korisnike. Ako pri pokretanju baze već postoji račun tipa admin, ne

stvra se novi. Kroz rad aplikacije admina od ostalih korisnika razlikujemo preko atributa `account_type` (slika 28).

```
8  (async () => {
9    let db = await connect();
10   let admin = await db.collection("users").findOne({account_type : 'Admin'})
11
12   db.collection('users').createIndex({ email: 1 }, { unique: true });
13
14   if(!admin){
15
16     let adminData = {
17       email: 'admin@admin',
18       password: process.env.ADMIN_PASSWORD,
19       date_created: Date.now(),
20       account_type: 'Admin'
21     }
22     register(adminData)
23     console.log("Admin created")
24   }
25 })();
```

Slika 28 auth.js - omotana asinkrona funkcija

Kod admina su funkcionalnosti koje ću objasniti: popis/pregled studenata te njihovih prijavnica i dnevnika, dodavanje partnera (samo onih koje je admin i napravio), dodavanje uputa i na kraju, dodavanje templatea za dnevnik. Brisanje/mijenjanje partnera su isti postupci i metode kao i za korisničko sučelje partnera i već smo ih spomenuli, alokacija studenata na određeni projekt se samo updatea atribut `allocated_to` preko funkcije `changeProjectInfo`, a tablica studenata obrađuje podatke na samo na frontendu.

Admin ima na frontendu view `Students.vue` gdje mu se preko kartica prikazuju studenti, za to su mu potrebne rute s backenda: `app.get('/students')` o kojem smo pričali za dohvaćanje studenata uz `search` funkciju. S te rute se dohvaća i prijavnica koja se sprema u dokument korisnika pod atribut `application_form`, a ruta za dohvaćanje dnevnika je `app.get('/journal/:id')`. Sastoji se od običnog mongo query-a koji traži po id-u studenta pripadajući dnevnik prakse.

```

679     async createPartner (req, res) {
680
681         let partnerData = req.body
682         partnerData.userID = ObjectID(partnerData.userID)
683
684         //hardcodano za pocetak dok partner ne uploada svoje headere ili logo
685         partnerData.img_url = "https://images.unsplash.com/photo-1504610926078-
686
687         //za raspoznavanje koji partneri su se sami kreirali, a koji ne
688         partnerData.created_by_admin = true
689         partnerData.account_type = 'Admin'
690
691         try{
692             let partnerID = await methods.pushData(partnerData, 'partners')
693
694             res.send(`partner with id ${partnerID} added.`)
695         }
696         catch(e){
697             res.status(500).json({ error: e.message});
698         }
699     },
700

```

Slika 29 routes.js – createPartner

Kreiranje partnera sam mogao smjestiti u funkciju addPartner i dodati neke dodatne uvjete koje vidimo ovdje npr. 'created_by_admin, ali insert i update metode u methods.js su već prenatrpane uvjetima što nije optimalno za izvođenje, a ni pregledno. Uz to, jedna sitna promjena može izazvati raspad sistema na puno dijelova. Created_by_admin označava da je on napravio partnera ili projekt i samo one koje je on napravio može izmjenjivati ili brisati.

```

166     async changeInstructions(req, res) {
167
168         let db = await connect();
169         let content = await db.collection('content').findOne();
170
171         let data = {
172             instructions : req.body
173         }
174
175         let obj = req.route.methods
176         data.method = Object.keys(obj).toString()
177
178
179         try{
180             let result
181
182             // ako ne postoji dokument u kolekciji 'content' insertaj instrukcije, inace ih updejta
183             if (!content){
184                 delete data.method
185                 result = await methods.pushData(data, 'content')
186             } else{
187                 data.id = content._id,
188                 data.updateDoc = true
189                 result = await methods.changeInfo(data, 'content')
190             }
191
192             res.send('success at changing instructions.')
193         }
194         catch(e){
195             res.status(500).json({ error: e.message});
196         }
197     },
198
199

```

Slika 30 routes.js - changeInstructions

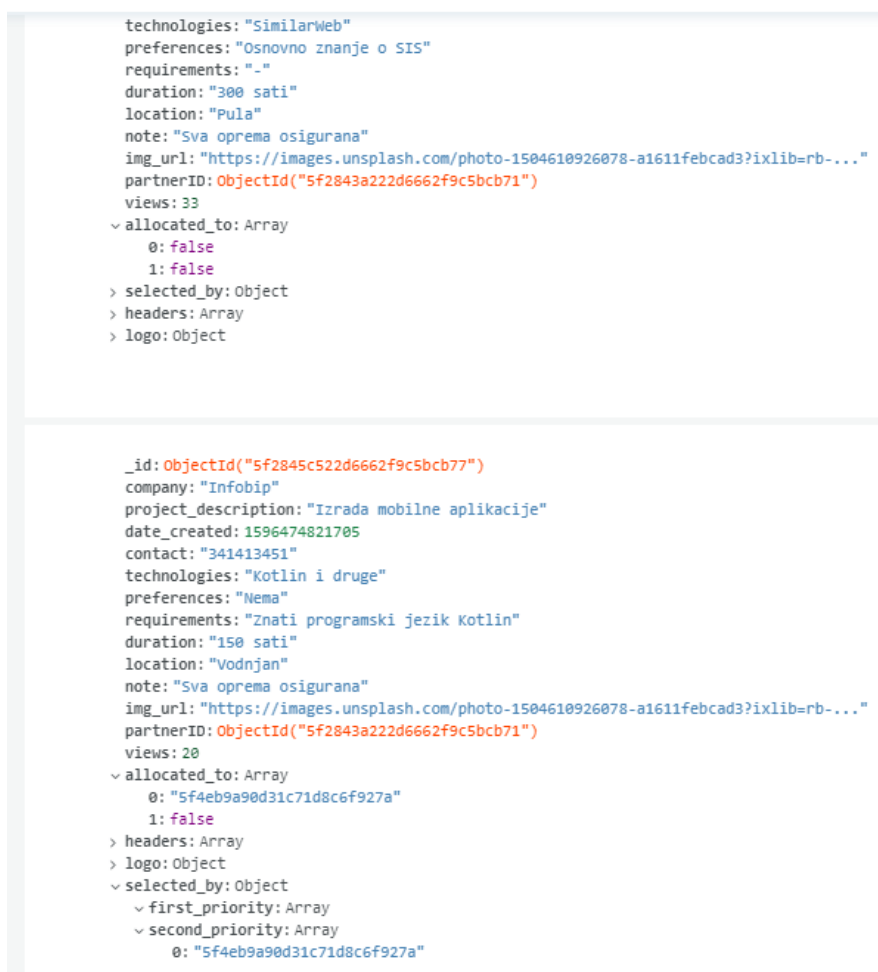
Instrukcije su pohranjene u jednom jedinom dokumentu u kolekciji 'content' jer sam odlučio sve neke takve generalne podatke pohraniti u jednu kolekciju. Ovisno o tome da li već postoji neki dokument u toj kolekciji, instrukcije se insertaju ili updateuju kroz

pushData odnosno changelInfo. Pošto su instrukcije samo jedan atribut objekata, a metoda je patch, ovaj handler vrši dodavanje, brisanje i mijenjanje instrukcija. Za upload templetea je funkcija ista, a file se pohranjuje kao base64 encoding.

Ovime sam obuhvatio sve bitnije funkcije i funkcionalnosti aplikacije. Cijeli kod je dostupan na Github profilu „fozbolt“.

Struktura baze podataka (screenshots)

Iako je jako mala, prilično je apstraktno pričati o bazi bez da je čitatelj vidi pa ću ovdje priložiti neke screenshots da se vidi kako smo rasporedili podatke.

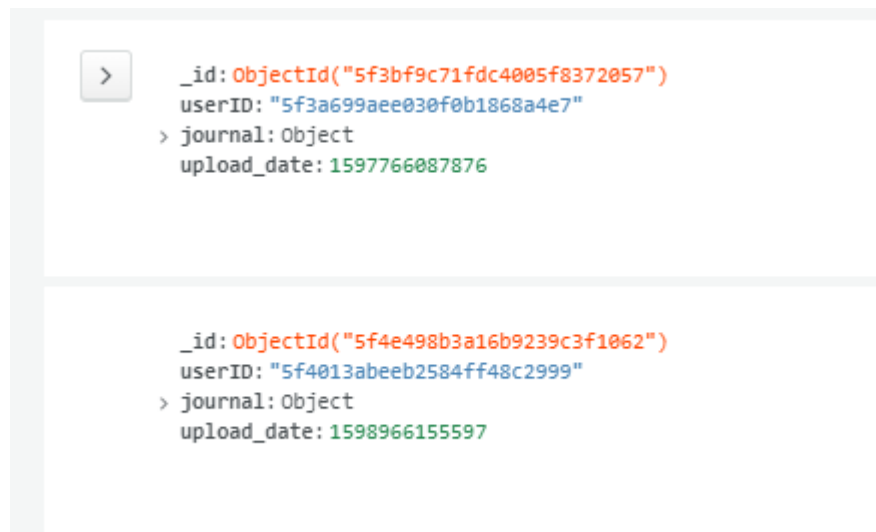


Slika 31 moja_praksa.projects

```
_id: ObjectId("5f3ed6c14f91b45bf0e40b51")
email: "miro@mail.com"
password: "$2b$08$876Qo4zrTaq.bCSFtFBdWuqFxl0uClmVTJysnoUPL.nf3w897AMp0"
date_created: 1597953729821
account_type: "Student"
jmbag: "34252"
name: "Miro"
surname: "Gavran"
technology: "java"
year: "3"
journalID: false
```

```
_id: ObjectId("5f4c0ff838b1882980fb5488")
email: "cenosco@mail.com"
password: "$2b$08$x5t8ZryLk5Vbdswo/1hQpOddX1VZGig9./SGee5WgTUV1/p.uqyLS"
date_created: 1598820344648
account_type: "Poslodavac"
```

Slika 34 moja_praksa.users - student i poslodavac



```
> { "_id": ObjectId("5f3bf9c71fdc4005f8372057"),
  "userID": "5f3a699aee030f0b1868a4e7",
  "journal": Object,
  "upload_date": 1597766087876 }

{ "_id": ObjectId("5f4e498b3a16b9239c3f1062"),
  "userID": "5f4013abeeb2584ff48c2999",
  "journal": Object,
  "upload_date": 1598966155597 }
```

Slika 35 moja_praksa.journals

Reference

Skripte i video materijali(<https://bit.ly/3gYtAkZ>), doc. dr. sc. Nikola Tanković

How long would it take to crack your password?, July 1, 2019, Team SpyCloud, <https://spycloud.com/how-long-would-it-take-to-crack-your-password/>

JSON Web Token (JWT) explained, Dec 13, 2018, <https://flaviocopes.com/jwt/>

Slika 1 – dependencies :

<https://www.npmjs.com/package/bcrypt>

<https://www.npmjs.com/package/dotenv>,

<https://www.npmjs.com/package/cors>,

<https://www.npmjs.com/package/mongodb>,

<https://www.npmjs.com/package/jsonwebtoken>,

<https://www.npmjs.com/package/express>,

<https://babeljs.io>