Šolski center Novo mesto

Srednja elektro šola in tehniška gimnazija

Šegova ulica 112

8000 Novo mesto

**Aplikacija za upravljanje z delovnimi nalogi**

(Izdelek in zagovor)

Avtor: Luka Žohar, r4.A

Mentor: Simon Vovko, dipl. inž. rač.

Kanižarica, februar 2019**Povzetek in ključne besede**

V tem projektnem delu sem izdelal aplikacijo, ki deluje na 'full stack' tehnologijah. Te so:

1. **Front-end tehnologije**:
   1. Angular,
2. **Back-end tehnologije**:
   1. Node.js,
   2. Express.js,
   3. Mongoose.
3. **Podatkovna baza**:
   1. MongoDB.

V teoretičnem delu opisal njihovo zgodovino, namen in uporabo, v praktičnem delu pa še predstavil na kakšen način sem jih uporabil v svoji aplikaciji.

**Ključne besede:** spletna aplikacija, web app delovni nalog, angular, node.js, express.js, mongoose, mongodb, javascript, typescript, html, css, podatkovna baza, database.

Kazalo

Vsebina

[1 Uvod 1](#_Toc9801160)

[2 Jedro naloge 2](#_Toc9801161)

[2.1 Teoretični del 2](#_Toc9801162)

[2.1.1 Angular 3](#_Toc9801163)

[2.1.2 Node.js 3](#_Toc9801164)

[2.1.3 Express.js 4](#_Toc9801165)

[2.1.4 MongoDB in NOSQL 5](#_Toc9801166)

[2.1.5 Mongoose 6](#_Toc9801167)

[2.1.6 Ostale NPM knjižnice 6](#_Toc9801168)

[2.2 Praktični del 7](#_Toc9801169)

[2.2.1 Instalacija MongoDB 7](#_Toc9801170)

[2.2.2 Instalacija Node.js 7](#_Toc9801171)

[2.2.3 Instalacija Angular-ja 7](#_Toc9801172)

[2.2.4 Postavitev projekta 7](#_Toc9801173)

[2.2.5 Registracija 8](#_Toc9801174)

[2.2.6 Prijava 11](#_Toc9801175)

[2.2.7 Upravljanje s računom 13](#_Toc9801176)

[2.2.8 Dodajanje in upravljanje s delovnim nalogom 14](#_Toc9801177)

[2.2.9 Postavitev spletnega strežnika 23](#_Toc9801178)

[2.2.10 Instalacija potrebnih storitev 24](#_Toc9801179)

[2.2.11 Postavitev spletne aplikacije 25](#_Toc9801180)

[3 Zaključek 26](#_Toc9801181)

[4 Sklici 27](#_Toc9801182)

**Slike :**

[Slika 1: MEAN skupek Vir: (http://www.techcluesblog.com/wp-content/uploads/2018/10/maxresdefault-1.jpg) 2](#_Toc9709782)

[Slika 2: Forma za registracijo 9](#_Toc9709783)

[Slika 3: Forma za login 11](#_Toc9709784)

[Slika 4: Oblika dodajanja podlage 15](#_Toc9709785)

[Slika 5: Oblika urejanja podlage 17](#_Toc9709786)

[Slika 6: Oblika izdelovanja nove podlage 19](#_Toc9709787)

[Slika 7: Statusi is prioriteta 20](#_Toc9709788)

[Slika 8: Oblika naloga 21](#_Toc9709789)

[Slika 9: Nastavitve pri ustarjanju droplet-a 24](#_Toc9709790)

**Skripte :**

[Skripta 1: Vstop v MongoShell 7](#_Toc9709791)

[Skripta 2: Ukaz za preverjanje instalacije Node-a 7](#_Toc9709792)

[Skripta 3: Ukaz za preverjanje NPM-a 7](#_Toc9709793)

[Skripta 4: Ukaz za instalacijo Angular CLI-ja 7](#_Toc9709794)

[Skripta 5: Ukaz za instalacijo potrebnih NPM paketov 7](#_Toc9709795)

[Skripta 6: Ukaz za ustvarjanje Angular projekta z CLI-jem 8](#_Toc9709796)

[Skripta 7: Skripta za ustvarjanje forma s zahtevami 9](#_Toc9709797)

[Skripta 8: Skripta za preverjanje zasedenosti uporabniškega imena 10](#_Toc9709798)

[Skripta 9: Telo zahtevka za registracijo 10](#_Toc9709799)

[Skripta 10: Skripta za dodajanje uporabnika na strežniku 11](#_Toc9709800)

[Skripta 11: Telo odgovora registracije uporabnika 11](#_Toc9709801)

[Skripta 12: Skripta za autentikacijo uporabnika 12](#_Toc9709802)

[Skripta 13: Telo odgovora za registracijo 13](#_Toc9709803)

[Skripta 14: Skripta za shranjevanje podatkov po autentikaciji 13](#_Toc9709804)

[Skripta 15: Dodajanje nove podlage 16](#_Toc9709805)

[Skripta 16: Posodabljanje polage na strežniku 18](#_Toc9709806)

[Skripta 17: Spreminjanje statusa 20](#_Toc9709807)

[Skripta 18: Skripta za sortiranje podlag po statusu padajoče 1/2 22](#_Toc9709808)

[Skripta 19: Skripta za sortiranje podlag po statusu padajoče 2/2 23](#_Toc9709809)

**Seznam uporabljenih kratic**

|  |  |  |
| --- | --- | --- |
| **Kratica** | **Angleško** | **slovensko** |
| ASP.NET | Active Server Pages | Aktivne strežniške strani |
| MSSQL | MicrosoftSQL | MicrosoftSQL |
| ER | Entity-relational | Entitetno-relacijski |
| HTML | Hyper Text Markup Language | Markantni jezik |
| CSS | Cascading Style Sheets | kaskadne stilske podloge |
| MVC | Model View Controller | Model pogled kontroler |
| API | Application programming interface | Aplikacijsko programerski pogled |
| SQL | Structured Query Language | Strukturiran poizvedovalni jezik |
| NPM | Node packet manager | Upravljalnik Node paketkov |

# Uvod

Ker imamo doma podjetje, ki se ukvarja z vezenjem na tekstil, sem bil v naše majhno podjetje vključen že pred leti. Do sedaj smo imeli vedno delovne naloge le v fizični obliki, ki so se lahko hitro izgubili, poškodovali, ni bilo neke stalnice glede tega, če je delovni nalog narejen ali ne,… Zato sem se v začetku 4. letnika srednje šole odločil, da bom naredil aplikacijo za upravljanje delovnih nalogov v elektronski obliki.

Glavni namen te aplikacije je lažje organiziranje podjetij, ki se ukvarjajo s storitvami.

**Primer uporabe**: šef pekarne dobi naročilo za 100 krofov, 40 marmeladnih in 60 čokoladnih, 30 slanikov ter 15 burekov. Peku pove, da naj speče toliko izdelkov, on pa pozabi, koliko krofov je čokoladnih in koliko marmeladnih, saj jih vsak dan dobi par deset naročil. Če pride do napake in se pozabi količina naročenih krofov, mora šef ponovno poklicati naročnika ter se ponovno dogovoriti za naročilo, ker pomeni da je izgubil čas ter nekoliko zaupanja naročnika.

Z mojo rešitvijo se ta problem odpravi, saj imamo trajno shranjene podatke o količini in vrsti izdelkov.

# Jedro naloge

## Teoretični del

Ko sem začel delati aplikacijo, sem na temo razvoja spletnih aplikacij spoznal, a ne dovolj, da bi znal samostojno narediti aplikacijo težavnosti, ki sem si jo zastavil.

Za razvijanje aplikacije sem najprej za 'front-end' uporabil Angular, 'back-end' pa ASP.NET ogrodje in MSSQL podatkovno bazo ter Entity Framework za upravljanje z podatkovno bazo. Naredil sem celoten ER model podatkovne baze, ki mi ga je uspelo narediti še kar dobro, a ko sem začel bazo dejansko ustvarjati ter poskušal s podatki upravljati preko Entity Frameworka, sem ugotovil, da mi za to tehnologijo primanjkuje znanja na tem področju in sem vedel, da bom potreboval za razvoj potreboval nekaj drugega in sem začel iskati alternativo.

Našel sem jo, ko sem na platformi YouTube našel video, ki je pokazal kako uporabljati Node.js s ogrodjem Express.js ter NOSQL podatkovno bazo MongoDB. Tehnologije so mi bile zelo všeč, saj so bile preproste za uporabo, a nič slabše od prejšnjih, le da je bilo potrebne manj kode, zato sem se odločil, da jih bom uporabil.

Slika, ki vsebuje besede zunanje

Opis je samodejno ustvarjen

Slika : MEAN skupek Vir: (<http://www.techcluesblog.com/wp-content/uploads/2018/10/maxresdefault-1.jpg>)

Glavne tehnologije, ki sem jih uporabil :

* Angular
* Node.js
* Express
* MongoDB
* Mongoose
* Različni NPM paketi
* HTTP protokol

### Angular

Ko se podamo v vode spletnih aplikacij, po navadi začnemo z osnovami spletnih strani, tj. HTML, CSS, nekaj osnovnih funkcionalnosti jezika JavaScript, a ko želimo izdelovati kompleksnejše aplikacije, se v njih lahko hitro 'izgubimo'. Zato se je dobro naučiti ogrodje (*ang.* *framework*) ali knjižnice, ki nam olajšajo in pohitrijo delo. Na voljo jih je veliko, od osnovnega jQuery-a, do kompleksnejših, kot so React, Vue, Meteor in Angular. Med temi sem se odločil za Angular, saj so mi ga priporočili ter mi je všeč njegova arhitektura.

**Zgodovina:** Ogrodje Angular obstaja od leta 2010, ko so ga razvili v podjetju Google. Prva verzija se je imenovala AngularJS, saj je bil programski jezik, ki je bil uporabljen v njem JavaScript. To je trajalo do leta 2014, ko je podjetje ogrodje zelo spremenilo. Odločilo se je, da bodo v njem uporabili programski jezik TypeScript, ki je programski jezik s statičnimi tipi, veliko funkcionalnosti so premaknili v module,… Od takrat posodobitve prihajajo na približno pol leta z novostmi, ki ga še poboljšajo. Trenutna verzija je Angular 7.

Deluje na principu MVC (model view controller) modela, v katerem je v tem primeru view HTML stran, model podatki in controller funcionalnost spletne strani, ki upravlja s podatki (TypeScript).

### Node.js

Če želimo izdelati spletne aplikacije, ki imajo funkcionalnost shranjevanja, posodabljanja, dodajanja in brisanja podatkov oziramo so dinamične, potrebujemo aplikacijo, ki deluje na strežniku in lahko upravlja s podatkovnimi bazami in datotekami. To dosežemo z 'back-end' tehnologijami, ki so tega zmožne. Te tehnologije so na primer PHP, C# in ostale. Na 'back-endu' lahko z podatki upravljamo s obdelovanjem le-teh (na primer računanje veljavnosti vnesene forme), zapisovanjem v tabelo,… Front-end je povezan preko API-ja, ki deluje preko HTTP protokola.

Node.js je spletni strežnik, ki izvaja JavaScript kodo zunaj brskalnika. JavaScript se primarno uporablja za skripte v brskalniku, a se s pomočjo Node.js tehnologije lahko uporablja tudi na strežniku. Čeprav obstaja šele od leta 2009, v njem prednosti vidi veliko velikih podjetij, kot so PayPal, Microsoft, Amazon, Walmart in ostali. S Node.js strežnikom po navadi uporabljamo NPM (node packet manager), ki upravlja s paketi, ki jih dobimo z zbirke paketov. Te paketi nam delo močno skrajšajo, saj zmanjšajo čas, porabljen za razvoj enakovrednih funkcij. Na primer lahko uporabimo knjižnico generate-Password, ki nam generira naključno geslo. Če tega paketa nebi uporabili, bi morali napisati veliko kode, ki bi nam vzela veliko časa ter še mogoče nebi bila zanesljiva.

V njem lahko pišemo v vseh jezikih, ki se prevedejo v JavaScript (TypeScript, Dart, CoffeScript,…).

### Express.js

Programiranje v Node.js-u lahko s povečevanjem velikosti in težavnosti lahko postane zamudno in manj pregledno, posledično za dodajanje funkcionalnosti projekta porabimo več časa in v primeru, da projekt zapustimo in pride za nami nekdo, ki kodo napisano v samem Node-u se bo verjetno težko ali sploh ne znašel. Zato so po objavi Node-a začeli razvijati ogrodja, ki nam delo zelo pohitrijo in polepšajo. Poznamo jih več, najbolj popularni so pa Koa2, Sails, Nest, Meteor in veliko ostalih, daleč najpopularnejši pa je Express.js, ki sem ga uporabil v tem projektu.

Express.js je minimalistično ogrodje, ki ga uporabljamo z Node-om. Najbolj koristen je za ustvarjanje REST API-jev. Prva verzija je bila izdana maja 2010. Najbolj uporabljen v kombinaciji z MEAN (Mongo Express Angular Node) in MERN (MongoDB Express React Node) skupkom. Pohitri nam veliko zadev s Node-om, na primer za zagon strežnika s Node-om potrebujemo več vrstic kode kot s Express-om.

### MongoDB in NOSQL

Ko izdelujemo spletne aplikacije, ki shranjujejo podatke, prej ali slej pridemo to točke, ko želimo podatke shranjevati v podatkovni bazi, saj tako podatke shranimo trajno. Vrst podatkovnih baz poznamo več, najbolj popularne so entitetno-relacijske oziroma baze, ki za upravljanje z njimi potrebujejo jezik SQL (MYSQL, MSSQL, PostgreSQL, MariaDB,…) ter NOSQL podatkovne baze.

**NOSQL podatkovne baze**

Take vrste baz delujejo na sistemu dokumentov. Razlike med NOSQL in SQL bazami lahko razložimo na praktičnem primeru oblike shranjenih podatkov:

**Shranjevanje podatkov v SQL bazah**

SQL baze si lahko predstavljamo kot da imamo več predalov, v katerih shranjujemo podatke eni osebi. V tem primeru je en predal ena tabela. Podatki, shranjeni v teh predalih so napisani v eni tabeli, podatki med predali so pa povezani z indeksi.

**Shranjevanje podatkov v NOSQL bazah**

Shranjevanje podatkov v NOSQL bazah lahko na praktičnem primeru predstavimo tako, da imamo za vsako osebo (dokument) svoj predal, v katerega se shranjujejo vsi podatki o njem. Ta način se razlikuje od SQL-a v tem, da ne rabimo po določen podatek iskati v drug predal, kar je bolj pregledno. Najboljši primer je dokumentiranje pacientov pri zdravniku, kjer ima vsak svojo mapo, v kateri so shranjeni vsi podatki o tej osebi, njegovi pregledi, operacije in ostalo.

**MongoDB**

MongoDB je NOSQL podatkovna baza, ki deluje na sistemu shranjevanja dokumentov. Je med najpopularnejšimi podatkovnimi bazami, saj je bil na platformi StackOverflow četrta najbolj uporabljena med vsemi. Že od začetka je brezplačna. Razvilo in objavilo ga je podjetje MongoDB Inc. februarja 2009. Podatki so shranjeni v formatu, podobnem JSON-u, kar omogoča fleksibilnost sheme dokumenta. Model dokumenta se lahko 'mapira' kot objekt, zato je delo s podatki lažje. Za upravljanje z njo lahko uporabljamo konzolo, v katere vpisujemo poizvedbe, MongoDB Compass, ki je grafična namizna aplikacija, priročna če imamo veliko dokumentov, saj si podatke lažje prikažemo grafično. MongoDB Atlas deluje kot podatkovna baza v oblaku, podobno kot Microsoftov Azure.

### Mongoose

Mongoose.js ali bolj uporabljeno Mongoose je orodje za objektno modeliranje, ki ga uporabljamo za modeliranje sheme za interakcijo s MongoDB podatkovno bazo.

### Ostale NPM knjižnice

1. **materialize-css**

Materialize je ogrodje za enostavno grafično izdelovanje spletnih strani. Deluje na Material osnovi.

1. **bcryptjs**

bcryptjs je knjižnica, ki sem jo uporabil za kodiranje gesel ter odkodiranje le teh za preverjanje veljavnosti vnesenega gesla.

1. **jsonwebtoken**

Knjižnica, ki se uporablja za preverjanje pristnosti uporabnikov. Knjižnica generira kodirano besedilo, v katerem so podatki o uporabniku in o načinu kodiranja besedila.

1. **passport in passport-jwt**

Passport je knjižnica za avtorizacijo uporabnika, ki ima več načinov avtorizacije. V mojem primeru sem uporabil avtorizacijo z passport-jwt, ki preveri veljavnost token-a, poslanega v glavi http zahtevka.

1. **moment**

Moment je knjižnica, ki sem jo uporabil za generiranje datumov ter za njihovo formatiranje.

## Praktični del

### Instalacija MongoDB

S uradne spletne strani MongoDB naprej prenesemo .msi instalacijsko datoteko, ki jo nato namestimo. V konzoli se premaknemo v naslednjo mapo in zaženemo ukaz 'mongo' da se nam odpre MongoDB shell, v katerem lahko pišemo ukaze za upravljanje s bazo.

C:\Program Files\MongoDB\Server\4.0\bin>mongo

Skripta : Vstop v MongoShell

### Instalacija Node.js

Node.js instalacijsko datoteko prenesemo z uradne strani in jo namestimo. Hkrati se nam namesti tudi NPM. Če je vse nameščeno preverimo z naslednjima ukazoma v konzoli.

C:\Users\lukaz>node -v

Skripta : Ukaz za preverjanje instalacije Node-a

C:\Users\lukaz>npm -v

Skripta : Ukaz za preverjanje NPM-a

### Instalacija Angular-ja

Za instalacijo uporabimo naslednji ukaz, ki nam namesti Angularjev CLI.

C:\Users\lukaz>npm install @angular/cli

Skripta : Ukaz za instalacijo Angular CLI-ja

### Postavitev projekta

Za projekt moramo prenesti vse NPM knjižnice, ki jih bomo potrebovali. To naredimo z naslednjim ukazom :

C:\Users\lukaz> npm install bycryptjs body-parser cors express moment passport passport-jwt

Skripta : Ukaz za instalacijo potrebnih NPM paketov

Da ustvarimo nov Angular projekt, izvedemo naslednji ukaz :

C:\Users\lukaz> ng new workSheet

Skripta : Ukaz za ustvarjanje Angular projekta z CLI-jem

Ta nam ustvari novo mapo, kjer so preneseni NPM module, potrebni za njegovo delovanje. Ustvari tudi strukturo map in datotek, katere bomo uporabljali za programiranje. Po tem postopku začnemo z programiranjem.

### Shema podatkovne baze

Za ustvarjanje sheme sem uporabil orodje Mongoose. Vsakemu polju sem določil podatkovi tip, če je obvezno in nadomestno vrednost (default).

const UserSchema = Schema({

profile: {

username: { type: String, require: true },

email: { type: String, require: true },

firstName: { type: String, default: '' },

lastName: { type: String, default: '' },

created: { type: String, default: moment().format('YYYY-MM-DDTHH:mm:ss') },

modified: { type: String, default: moment().format('YYYY-MM-DDTHH:mm:ss') },

password: { type: String, require: true }

},

sheets: [

{

templateTitle: { type: String },

title: { type: String },

description: { type: String, default: '' },

sheetCreated: { type: String, default: moment().format('YYYY-MM-DDTHH:mm:ss') },

sheetModified: { type: String, default: moment().format('YYYY-MM-DDTHH:mm:ss') },

status: { type: String, default: 'Not Started Yet' },

statusModified: { type: String, default: moment().format('YYYY-MM-DDTHH:mm:ss') },

priority: { type: String, default: 'Low' },

priorityModified: { type: String, default: moment().format('YYYY-MM-DDTHH:mm:ss') },

items: { type: Array }

}

],

templates: [

{

title: { type: String },

description: { type: String },

items: { type: Array },

created: { type: String, default: moment().format('YYYY-MM-DDTHH:mm:ss') },

modified: { type: String, default: moment().format('YYYY-MM-DDTHH:mm:ss') }

}

]

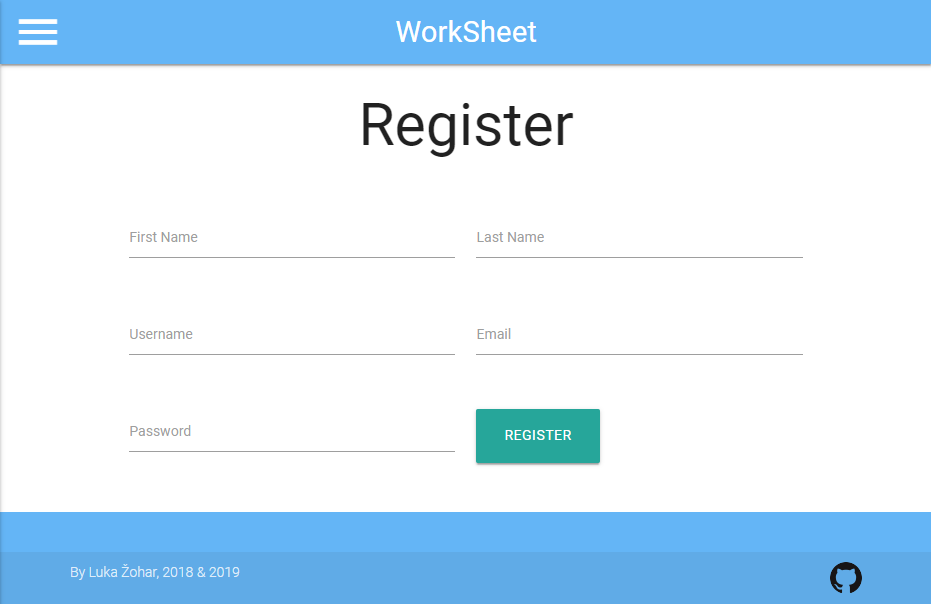
});

Skripta : Shema podatkovne baze

### Registracija

Ker je aplikacija narejena za trajno shranjuje podatke sem moral narediti registracijo, ki na podatkovni bazi naredi prostor za shranjevanje podatkov o uporabniku, saj je brez registracije nemogoče imeti organizirano ter varno aplikacijo. Pri registraciji je potrebno vpisati uporabniško ime, elektronski naslov, geslo, ime in priimek. Zahteve za veljavnost forme so:

1. Ime in priimek :
   1. Minimalna dolžina je 2 znaka,
   2. Veljavni znaki so male in velike črke od a do ž,
   3. Najdaljša dolžina je 100 znakov.
2. Uporabniško ime :
   1. Minimalna dolžina so 3, najdaljša pa 100 znakov
   2. Veljavno znaki so male in velike črke, številke od 0 do 9 ter znaka \_ in –
   3. Če je uporabniško ime na voljo preverim s HTTP poizvedbo na strežnik, ki odgovori s true ali false. Glede na odgovor se nastavi veljavnost uporabniškega imena ter se uporabnika obvesti
3. Elektronski naslov
   1. Veljavnost naslova se preveri z reggex preverjanjem
   2. Če je na voljo se preveri enako kot pri up. imenu, le da preveri veljavnost elektronski naslov
4. Geslo
   1. Minimalna dolžina je 8 znakov
   2. Mora vsebovati majhno in veliko črko ter številko



Slika : Forma za registracijo

// First name can contain small and big letters from a-Ž

this.firstName = new FormControl('', [

Validators.minLength(2),

Validators.maxLength(100),

Validators.pattern('[a-žA-Ž ]\*'),

Validators.required

]);

// Same as with first name

this.lastName = new FormControl('', [

Validators.minLength(2),

Validators.maxLength(100),

Validators.pattern('[a-žA-Ž ]\*'),

Validators.required

]);

// Username has to contain small and big letter and number

// Availability of username is being checked asynchronously

this.username = new FormControl('', [

Validators.required,

Validators.minLength(3),

Validators.maxLength(100),

Validators.pattern('[[a-žA-Ž0-9\_-]\*')

], [

this.userValidator.validateUsernameAvailability()

]

);

// Email has to contain @ sign and atleast one dot after @

// Availabity of email is being checked asynchronously

this.email = new FormControl('', [

Validators.required,

Validators.email

], [

this.userValidator.validateEmailAvailability()

]);

// Password has to contain one small and big letter and number and has to be atleast 8 charecters long

this.password = new FormControl('', [

Validators.required,

Validators.minLength(8),

Validators.pattern(/^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).\*$/)

]);

}

Skripta : Skripta za ustvarjanje forme s zahtevami

Ker sem hotel narediti aplikacijo čim bolj realno časovno, sem up. imenu in el. pošti dodal preverjanje, ki na vsako sekundo pošlje zahtevek na strežnik o tem, če je ime ali email, ki ga je uporabnik vpisal še na voljo. Koda izgleda tako :

URL = 'http://localhost:3000/api/users';

// Calls username availability on backend with username in parameters

private checkForUsernameAvailable(username: string): Observable<ISuccessMsgResponse> {

return this.http.get<ISuccessMsgResponse>(`${this.URL}/usernameAvailability?username=${username}`, {

headers: new HttpHeaders({

'Authorization': this.token.getToken()

})

});

}

// Calls function to call server for availability

public checkForUsernameAvailability(username: string): boolean {

let isAvailable: boolean;

this.checkForUsernameAvailable(username).subscribe(

// If res.success is true, username is available

// If res.success is false, username is taken

(res: ISuccessMsgResponse) => {

if (res.success) {

isAvailable = true;

} else {

isAvailable = false;

}

},

err => {

console.error(err);

}

);

return isAvailable;

}

Skripta : Skripta za preverjanje zasedenosti uporabniškega imena

**Postopek registracije :**

1. Uporabnik vnaša podatke, pri tem brskalnik sproti preverja veljavnost podatkov. Zaradi varnosti gesla se celotna forma ne pošlje na strežnik preden se preveri veljavnost uporabniškega imena ali elektronskega naslova, saj je geslo nezavarovano pri prenosu,
2. Ko je forma veljavna uporabnik klikne na gumb za registracijo. Brskalnik pošlje POST zahtevek na strežnik, ki v bazo vpiše novega uporabnika s podanimi podatki in geslom, ki se ga zakodira s 'saltom' 10. stopnje in hashom,

**Telo prošnje** :

{

"username": "johndoe",

"email": "john.doe@gmail.com",

"firstName": "John",

"lastName": "Doe",

"password": "MyPass1234"

}

Skripta : Telo zahtevka za registracijo

**Funkcija za dodajanje uporabnika :**

module.exports.addUser = function (newUser, callback) {

// Generates salt rounded, 10 rounded

bcrypt.genSalt(10, (err, salt) => {

if (err) throw err;

// Hashes password with salt for more security

bcrypt.hash(newUser.profile.password, salt, (err, hash) => {

if(err) throw err;

newUser.profile.password = hash;

newUser.save();

callback(null, newUser);

})

})

};

Skripta : Skripta za dodajanje uporabnika na strežniku

1. Strežnik odgovori s sporočilom ter indeksom novo ustvarjenega uporabnika.

{

"success": true,

"msg": "Successfully added user, 5ce950f06259e92f14912186"

}

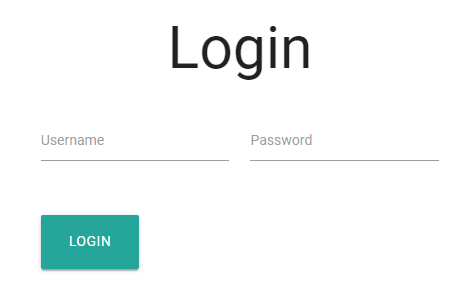
Skripta : Telo odgovora registracije uporabnika

### Prijava

Da uporabnik dobi že podane podatke se mora prijaviti. Za to potrebuje veljavno uporabniško ime ter geslo. Če uporabniško ime ali geslo ni pravilno, se uporabnika o tem obvesti. Če je vse pravilno dobi žeton (JWT Token), s katerim lahko dostopa do funkcij oziroma poti, ki so za uporabnike brez njega nedostopni, kot so na primer dodajanje nove podlage ali delovnega lista, saj bi brez preverjanje pristnosti lahko vsak upravljal s podatki uporabnika.

**Postopek prijave :**

1. Uporabnik vpiše uporabniško ime ali elektronski naslov ter geslo



Slika : Forma za login

1. Brskalnik pošlje POST zahtevek z up. imenom in geslom na strežnik
2. Strežnik preveri, če obstaja podano uporabniško ime, če da, preveri še če se gesla ujemata, če se ne poda odgovor, da so podani napačni

// Authenticate

router.post('/authenticate', (req, res, next) => {

const username = req.body.username;

const password = req.body.password;

// It finds all users with given username

User.getUserByUsername(username, (err, user) => {

if(err) {

console.error(err);

return res.json({ success: false, msg: 'Server error' }).status(500);

}

// Returns 400 Client error if user doesn't exist

if(!user) {

return res.json({

success: false,

msg: "User doesn't exist"

}).status(400);

}

// If it exists, it compares given and hashed password in found user

User.comparePasswords(password, user.profile.password, (err, isMatch) => {

if(err) {

console.error(err);

return res.json({ success: false, msg: 'Server error' }).status(500);

}

if (isMatch) {

// If password are matched, it created token with user data that expires in 1209600 seconds (2 weeks)

const token = jwt.sign(user.toJSON(), config.secret, {

expiresIn: 1209600 // 2 weeks

});

// Deletes password hash for safety

var profile = user.profile;

profile.password = undefined;

// Responds with status 200 OK with data of user and created token

return res.json({

\_id: user.\_id,

success: true,

token: 'JWT ' + token,

userData: {

profile: profile,

sheets: user.sheets,

templates: user.templates

}

}).status(200)

} else {

// If password is incorrect, it responds with 400 Client error and message

return res.json({

success: false,

msg: 'Wrong password'

}).status(400);

}

})

})

});

Skripta : Skripta za autentikacijo uporabnika

1. Če sta podatka pravilna, strežnik odgovori s sporočilom, novo ustvarjenim žetonom, ki je veljaven 14 dni ter vsemi podatki o uporabniku, razen gesla zaradi varnostnih razlogov.

{

"\_id": "5ce950f06259e92f14912186",

"success": true,

"token": "JWT token\_string",

"userData": {

"profile": {

"firstName": "John",

"lastName": "Doe",

"created": "2019-05-25T16:27:22",7

"modified": "2019-05-25T16:27:22",

"username": "johndoe",

"email": "john.doe@gmail.com"

},

"sheets": [],

"templates":[]

}

}

Skripta : Telo odgovora za registracijo

1. Brskalnik shrani žeton in podatke ter uporabnika pošlje na začetno stran, kjer lahko uporablja aplikacijo.

loginUser() {

if (this.loginForm.valid) {

// If login form is valid. it calls auth service for logging user

this.authService.loginUser(this.loginForm.value).subscribe(

(data: ILoginResponse) => {

if (data.success === true) {

// If user credentials are correct, it saves data in local storage

this.saveData(data.token, data.userData, data.\_id);

this.toast.success('Successfull login');

// Navigates to workSheet route

this.router.navigate(['/worksheets']);

} else {

// If user credentials are invalid, it informs user

this.toast.warning('Invalid credentials');

}

},

err => {

console.log(err);

this.toast.error('Error logging in');

}

);

} else {

this.toast.warning('Invalid credentials');

}

}

Skripta : Skripta za shranjevanje podatkov po autentikaciji

### Upravljanje s računom

Z računom lahko upravljamo tako, da urejamo podatke uporabnika ter nastavitve, kot so uporabniško ime, elektronski naslov, ime,... Pri urejanju se preverja, če je spremenjeno uporabniško ime ali elektronska pošta veljavno. Funkcija za preverjanje je enaka kot pri registraciji.

### Dodajanje in upravljanje s delovnim nalogom

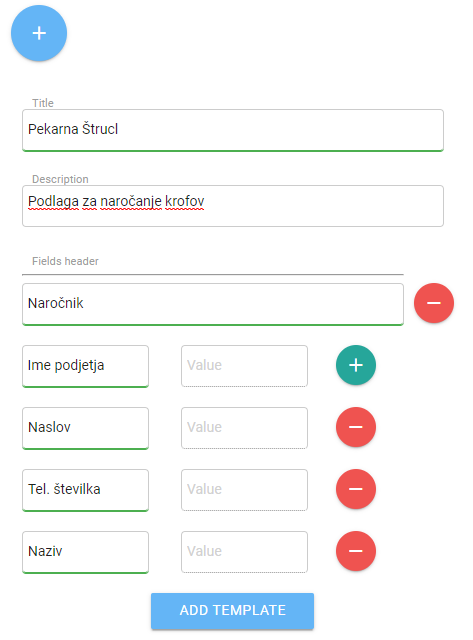
Potek dodajanja delovnega naloga sem razdelil na 2 dela, in sicer dodajanje podlage, ki je osnova za dodajanje delovnega naloga in dodajanje delovnega naloga.

#### Podlaga

Podlaga je prostor, kjer oblikujemo format delovnega naloga. Po navadi mora podjetnik za izdelavo podlage uporabljati programe, ki so lahko zahtevni za osebe, ki se v njih ne znajdejo, kot so na primer In Design podjetja Adobe, v tem projektu pa sem naredil preprosto oblikovanje le-tega. Sicer ni tako fleksibilen, je pa za večino več kot zadovoljiv.

##### Dodajanje nove podlage

Novo podlago ustvarimo tako, gremo na zavihek `add-template`. Vpišemo naslov, po želji tudi opis in začnemo dodajati elemente. Elementi ki so na voljo so: naslov, skupina vrstic, tabela, lista, v prihodnosti pa še ostale. Tako lahko naredimo element, kjer bomo vpisovali podatke o naročniku (podjetje, naslov, elektronski naslov, telefonska številka,…) Če je forma podlage veljavna, dodamo novo s klikom na gumb. Na strežniku se podlaga doda v bazo s trenutnim časom.



Slika : Oblika dodajanja podlage

**template.js**

module.exports.addTemplate = (newTemplate, userID, callback) => {

// Pushes template to array of templates on databse

User.findByIdAndUpdate(userID, {$push: {"templates": newTemplate}}, { new: true }, (err, newUser) => {

if(err) throw err;

callback(null, newUser.templates[newUser.templates.length - 1]);

});

}

**templates.js**

router.post(passport.authenticate('jwt', {session: false}), (req, res) => {

let template = req.body;

let userID = req.user.\_id;

// Calls function for adding template

Template.addTemplate(template, userID, (err, template) => {

if(err) {

console.error(err);

return res.json({ success: false, msg: 'Server error' }).status(500);

}

// Returns with newly created template data with status 201 Created

return res.json({

success: true,

msg: template.title + ' created',

data: template

}).status(201);

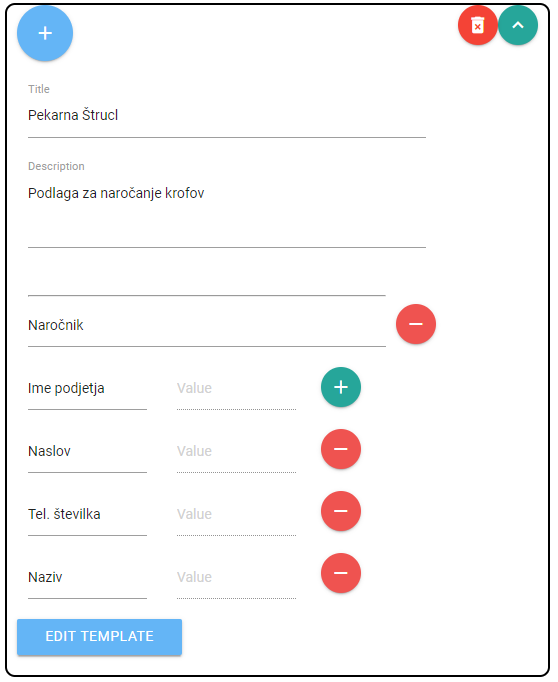
});

})

Skripta : Dodajanje nove podlage

##### Urejanje podlag

Že dodano podlago je možno urejati tako, da odstranjujemo prejšnje elemente, dodajamo nove ali jih urejamo. Na strežniku se spremeni čas zadnjega urejanje podlage glede na trenutni strežniški čas.



Slika : Oblika urejanja podlage

**templates.js**

// Updates sheet

// 1. Calls function for updating template with updated template

// 2. Responding depending of modification status

router.put(passport.authenticate('jwt', {session: false}), (req, res) => {

// Sets modified to current time

req.body.modified = moment().format('YYYY-MM-DDTHH:mm:ss')

Template.updateTemplate(req.user.\_id, req.body.\_id, req.body, (err, modifiedStatus) => {

if(err) {

console.error(err);

return res.json({ success: false, msg: 'Server error' }).status(500);

}

const templateTitle = req.body.title;

// If n is 0, template doesn't exist. Returns status 400

if(modifiedStatus.n === 0) {

return res.json({

success: false,

msg: templateTitle + " doesn't exist"

}).status(400);

}

// If template is found and is modified OR is found and not modified, it returns

// status 200

if(( modifiedStatus.n === 1 && modifiedStatus.nModified === 1 ) || ( modifiedStatus.n === 1 && modifiedStatus.nModified === 0 )) {

return res.json({

success: true,

msg: templateTitle + ' updated'

}).status(200);

}

})

})

// Updating found template and returning status of modification

**template.js**

module.exports.updateTemplate = (userId, templateId, editedTemplate, callback) => {

// Finds template with templateID in parameters and updates it in databse

User.updateOne({\_id: ObjectId(userId), "templates.\_id": ObjectId(templateId)}, {$set: {"templates.$": editedTemplate}}, {new: true}, (err, modifiedStatus) => {

if(err) throw err;

callback(null, modifiedStatus);

});

}

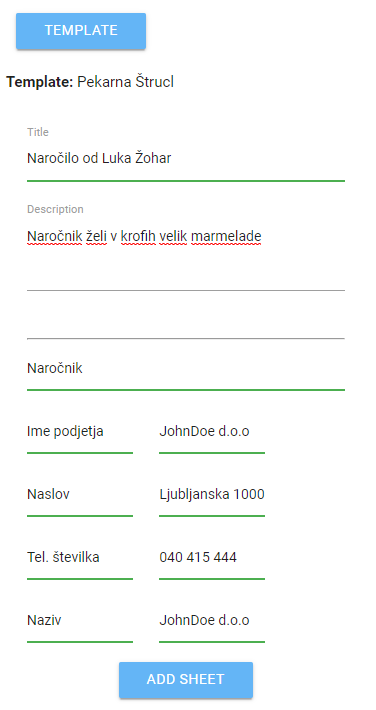
Skripta : Posodabljanje polage na strežniku

#### Delovni nalog

Za delovni nalog moramo imeti že v naprej pripravljeno podlago, v katero bomo vnašali podatke, ki bodo bili za vsako naročilo drugačni.

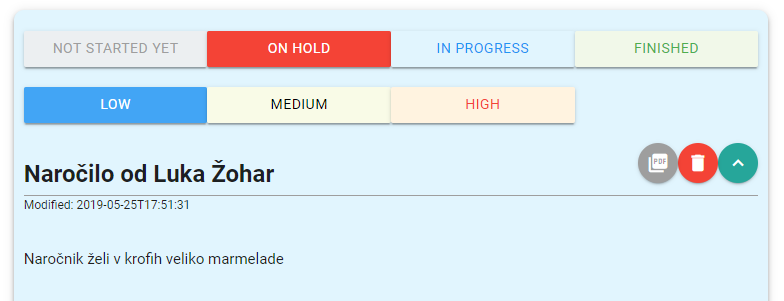
##### Dodajanje delovnega naloga

Delovni nalog dodamo tako, da izberemo v naprej narejeno podlago. Podlaga je potrebna, saj brez nje nebi imeli formata za vpisovanje podatkov. Najprej vpišemo naslov in opis, nato pa v elemente vpisujemo vrednosti.



Slika : Oblika izdelovanja nove podlage

Po dodajanju naloga mu lahko določimo status in prioriteto. Status pove, v kateri fazi je naročilo. Prioriteta pove, kakšna je nujnost naročila.



Slika : Statusi is prioriteta

**sheets.js**

// Request for changing status of sheet

// 1. Calling function for setting status of sheet, passing sheetID and new status

// 2. Checking modification status and returning depending to that status

router.put('/setStatus', passport.authenticate('jwt', {session: false}), (req, res) => {

const userID = req.user.\_id;

const sheetID = req.body.sheetID;

const newStatus = req.body.status;

// Function for setting status

Sheet.setStatus(userID, sheetID, newStatus, (err, modifiedStatus) => {

if(err) throw err;

// If sheets is found, modifiedStatus.n is 1. If modified.nModified is 1, sheet was also

// modified. Both are success because if modifiedStatus.n is 1, sheet was found but status is

// same as before

if(( modifiedStatus.n === 1 && modifiedStatus.nModified === 1 ) || ( modifiedStatus.n === 1 && modifiedStatus.nModified === 0 )) {

// Responds with success, message and time of change

return res.json({

success: true,

msg: 'Status changed to ' + newStatus,

data: moment().format('YYYY-MM-DDTHH:mm:ss')

}).status(200);

} else return res.json({

success: false,

msg: 'Error updating sheet status to ' + newStatus

}).status(500);

})

})

**sheet.js**

module.exports.setStatus = function(userID, sheetID, status, callback) {

// In database, it changes status, statusModified date and sheetModified date to current date

User.updateOne({$set: {

"sheets.$.status": status,

"sheets.$.statusModified": moment().format('YYYY-MM-DDTHH:mm:ss'),

"sheets.$.sheetModified": moment().format('YYYY-MM-DDTHH:mm:ss')

}})

.where({\_id: ObjectId(userID)})

.where({"sheets.\_id": ObjectId(sheetID)})

.exec({ new: true}, (err, modifiedStatus) => {

if(err) throw err;

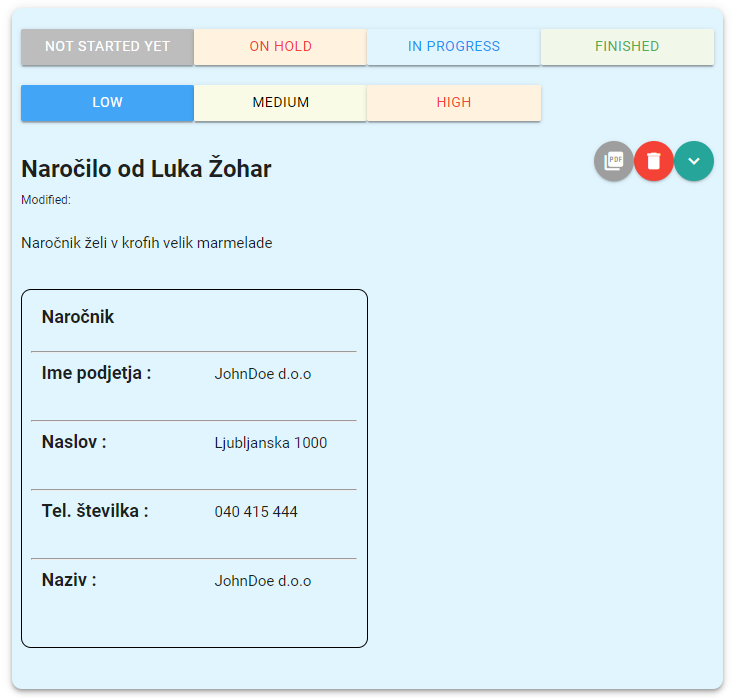
// Returns status of modification (is sheet found? and is sheet modified?)

callback(null, modifiedStatus);

});

}

Skripta : Spreminjanje statusa



Slika : Oblika naloga

##### Urejanje delovnega naloga

Že obstoječi nalog lahko urejamo tako, da spreminjamo le vrednosti, saj če bi spreminjali elemente, bi spremenili podlago. Če je urejen nalog veljaven, ga posodobimo s klikom. Na strežniku se posodobijo podatki o nalogi ter datum zadnjega urejanja.

#### Sortiranje nalogov

Že dodane delovne naloge pregledujemo, saj si jih moramo med naročilom večkrat ogledati. Razvrstiti jih je možno po datumu, imenu in pomembnosti. Podlage in naloge razširimo s klikom na gumb. Podlage in naloge lahko brišemo.

**sortSheet.js**

// 1. Sorting sheets by priority

// 2. Sorting each priority group by date ascending

// 3. Returning combined sheets

module.exports.orderSheetsByPriorityAscending = function(sheets, limit, page, callback) {

// 3 arrays for every group of priority types

let sortedSheetsHigh = [];

let sortedSheetsMedium = [];

let sortedSheetsLow = [];

// Loops through array of all sheets

sheets.forEach(sheet => {

switch(sheet.priority) {

// If priority if high, it pushes to array of sortedSheetsHigh

case 'High' : {

sortedSheetsHigh.push(sheet);

break;

};

// If priority if medium, it pushes to array of sortedSheetsMedium

case 'Medium' : {

sortedSheetsMedium.push(sheet);

break;

};

// If priority if low, it pushes to array of sortedSheetsLow

case 'Low' : {

sortedSheetsLow.push(sheet);

break;

};

}

});

// Orders sheets by group by date - ascending

exports.orderSheetsByDateAscending(sortedSheetsHigh, 0, 0, (err, ordered) => {

if(err) throw err;

sortedSheetsHigh = ordered;

});

exports.orderSheetsByDateAscending(sortedSheetsMedium, 0, 0, (err, ordered) => {

if(err) throw err;

sortedSheetsMedium = ordered;

});

exports.orderSheetsByDateAscending(sortedSheetsLow, 0, 0, (err, ordered) => {

if(err) throw err;

sortedSheetsLow = ordered;

});

// Combines all sorted arrays, from low priority to high

let allSheetsSorted = sortedSheetsLow.concat(sortedSheetsMedium, sortedSheetsHigh);

callback(null, allSheetsSorted);

}

Skripta : Skripta za sortiranje podlag po statusu padajoče 1/2

**sheet.js**

// 1. Finding user

// 2. Calling function by type user requested and passing in his sheets, limit and page

// 3. Returning ordered sheets

module.exports.getSortedSheets = function(userID, sortType, order, limit, page, callback) {

// Finds all user

User.findOne({\_id: ObjectId(userID)})

.exec((err, user) => {

if(err) throw err;

if(!user) {

callback(null, undefined);

} else {

// Checks for sort type, and executes that type ascending or descending

switch(sortType) {

case 'priority': {

if(order == 'ascending') {

sortModule.orderSheetsByPriorityAscending(user.sheets, limit, page, (err, orderedSheet) => {

if(err) throw err;

// Returns ordered sheets by priority ascending

callback(null, orderedSheet);

});

} else {

sortModule.orderSheetsByPriorityDescending(user.sheets, limit, page, (err, orderedSheet) => {

if(err) throw err;

callback(null, orderedSheet);

});

}

};

}

}

})

}

**sheets.js**

// Request for sorting with parameters: type, limit, page and order

// 1. Getting all sort parameters

// 2. Calling function to get sorted sheets

// 3. Returning sorted sheets

router.get('/sort', passport.authenticate('jwt', {session: false}), (req, res) => {

const userID = req.user.\_id;

const type = req.query.type;

const limit = () => {

// If limit in query is undefined, it sets limit to default 16 elements per page

if(req.query.limit) { return req.query.limit; }

else { return 16; }

}

// Similar as limit

const page = () => {

if(req.query.page) { return req.query.page; }

else { return 1; }

}

const order = () => {

if(req.query.order) { return req.query.order; }

else { return 'descending'; }

}

// In sheet.js

Sheet.getSortedSheets(userID, type, order(), limit(), page(), (err, sheets) => {

if(err) throw err;

if(!sheets) {

return res.json({

success: false,

msg: 'Failed to sort sheets by ' + type,

}).status(400);

} else {

// If sorting succedded, it responds with succes, message and ordered sheets

return res.json({

success: true,

msg: 'Found and returned sheets, sorted by ' + type + ' ' + order(),

data: sheets

}).status(200);

}

});

})

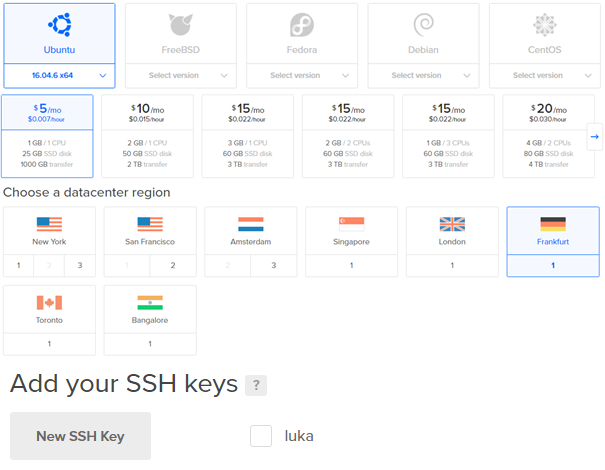
Skripta : Skripta za sortiranje podlag po prioriteti padajoče 2/2

### Postavitev spletnega strežnika

V tem poglavju bom opisal postopke, ki sem jih uporabil za vzpostavitev strežnika.

#### Kreiranje VPS-a na platformi DigitalOcean

Na platformi DigitalOcean, ki ponuja strežniško gostovanje, sem ustvaril 'droplet', ki omogoča ustvarjanje VPS-jev (Virtual Private Server). V nastavitvah sem dodal javni SSH ključ za povezavo preko konzole, nastavil lokacijo strežnika in velikost strojne opreme, ki jo bom zakupil, ter operacijski sistem (Ubuntu).



Slika : Nastavitve pri ustarjanju droplet-a

### Instalacija potrebnih storitev

Na novo ustvarjen VPS sem se povezal preko SSH ključa in instaliral potrebne storitve.

**Posodobitev apt-get:** sudo apt update

**Ukaz za instalacijo MongoDB-ja:** sudo apt install -y mongodb

Za preverjanje delovanja baze: sudo systemctl status mongodb

Za dostop do baze sem najprej vnesel ukaz: export LC\_ALL=C

Nato sem do nje preko konzole dostopal s ukazom: mongo

**Ukaz za instalacijo NodeJS:** sudo apt install nodejs

Preverjanje, če je instalacija uspela: nodejs -v

**Ukaz za instalacijo NPM:** sudo apt install npm

Preverjanje, če je instalacija uspela: npm -v

### Postavitev spletne aplikacije

Najprej sem kloniral repozitorij iz platforme GitHub.

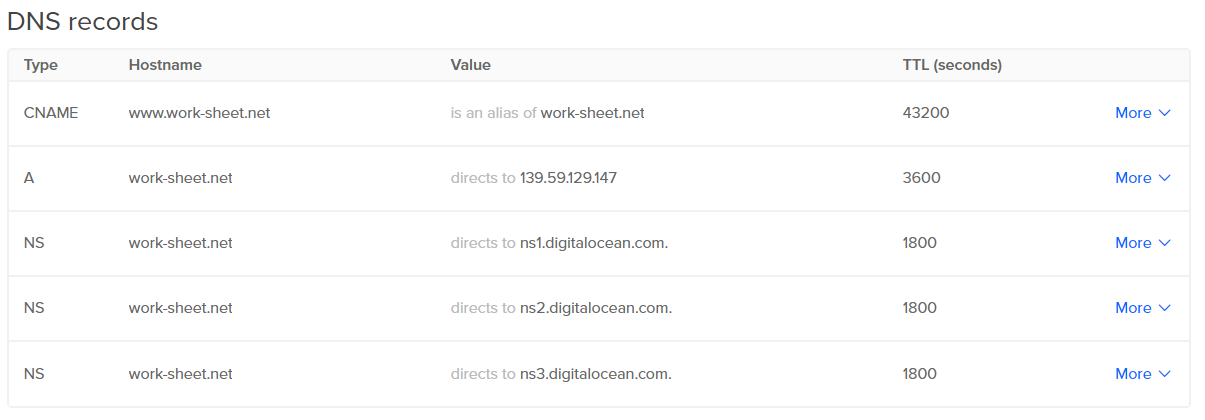
**Ukaz:** git clone <https://github.com/lukazohar/workSheet.git>

Premaknil sem se v preneseno mapo in s pomočjo nano urejevalnika kode spremenil vrata, skozi katera bo tekel promet na 80 (default http). Nato sem instaliral vse NPM pakete s ukazom `npm install`. Dodal sem še paket pm2, ki omogoča delovanje node strežnika brez ukaza node ali nodemon, tako da aplikacija deluje kot 'service'.

**Ukaz**: npm install pm2 -g

**Zagon**: pm2 start app.js

Po teh postopkih je aplikacija delovala. A ker bi uporabnik moral vpisati IP naslov za dostop do spletne strani, sem še zakiš domeno work-sheet.net in uredil DNS zapise.



Da sem lahko prihranil čas z ročnim spreminjanjem portov v kodi, sem izvedel naslednji ukaz : sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 3000

Ta ukaz vse zahtevke, ki pridejo na port 80 (default http port) preusmeri na port 3000, na katerem se nahaja aplikacija. Tako ne rabim vsakič, ko hočem na strežnik naložiti posodobljeno aplikacijo ročno spremeniti port, na katerem deluje.

# Zaključek

Ta projekt mi je dal ogromno izkušenj na področju načrtovanja in izdelave spletne aplikacije, saj sem se naučil postopkov za njeno izdelavo.

Idejo aplikacije mi je zelo všeč in se mi zdi zelo praktična, ima tudi potencial za zagonsko podjetje. V prihodnosti načrtujem dodati veliko več funkcionalnosti, kot so analiza dela, predlogi za izdelavo podlage in delovnega naloga, dodajanje časovnikov k nalogu za pregled nad porabljenim časom za storite, izvoz dokumenta v .pdf obliko, ki je primerna za tiskanje,..

Največja zahvala gre spletni skupnosti programerjev, ki na različnih platforma objavljajo učne ure in odgovore na vprašanja. Brez njih si dandanes programiranja sploh ne moremo predstavljati

# Sklici

*bycrypt*. (2019). Pridobljeno iz bycrypt: https://www.npmjs.com/package/bcryptjs

Google. (2. 5 2019). *Angular*. Pridobljeno iz Angular: https://angular.io/docs

*JWT*. (2019). Pridobljeno iz JWT: https://jwt.io/

*Materialize*. (2019). Pridobljeno iz Materialize: https://materializecss.com/

*Moment.js*. (2019). Pridobljeno iz Moment.js: https://momentjs.com/

MongoDB Inc. (10. 5 2019). *MongoDB*. Pridobljeno iz MongoDB documentation: https://docs.mongodb.com/

*Mongoose*. (2019). Pridobljeno iz Mongoose: https://mongoosejs.com/

Neznan. (10. 5 2019). *Express.js*. Pridobljeno iz Express.js: https://expressjs.com/

*Node.js*. (2019). Pridobljeno iz Node.js: https://nodejs.org/en/

*Passport.js*. (2019). Pridobljeno iz Passport.js: http://www.passportjs.org/

*Stack Overflow*. (2019). Pridobljeno iz Stack Overflow: https://stackoverflow.com/

*Wikipedia*. (19. 4 2019). Pridobljeno iz Angular (web framework): https://en.wikipedia.org/wiki/Angular\_(web\_framework)

Spletna stran se nahaja na domeni: [www.work-sheet.net](http://www.work-sheet.net)

Projekt se nahaja na GitHub-u : <https://github.com/lukazohar/workSheet>