Šolski center Novo mesto

Srednja elektro šola in tehniška gimnazija

Šegova ulica 112

8000 Novo mesto

**Aplikacija za upravljanje s delovnimi nalogi**

(Izdelek in zagovor)

Avtor: Luka Žohar, r4.A

Mentor: Simon Vovko, diplomirani naziv računalništva

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. **Podatkovne baze**:
   1. MongoDB.

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

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

Kazalo

Vsebina

[1 Uvod 1-1](#_Toc8341067)

[2 Jedro naloge 2-2](#_Toc8341068)

[2.1 Teoretični del 2-2](#_Toc8341069)

[2.1.1 Angular 2-4](#_Toc8341070)

[2.1.2 Node.js 2-5](#_Toc8341071)

[2.1.3 Express.js 2-6](#_Toc8341072)

[2.1.4 MongoDB in NOSQL 2-7](#_Toc8341073)

[2.1.5 Mongoose 2-8](#_Toc8341074)

[2.1.6 Ostale NPM knjižnice 2-9](#_Toc8341075)

[2.2 Praktični del 2-10](#_Toc8341076)

[2.2.1 Instalacija MongoDB 2-10](#_Toc8341077)

[2.2.2 Instalacija Node.js 2-10](#_Toc8341078)

[2.2.3 Instalacija Angular-ja in Express-a 2-10](#_Toc8341079)

[2.2.4 Registracija 2-10](#_Toc8341080)

[2.2.5 Prijava 2-12](#_Toc8341081)

[2.2.6 Upravljanje s računom 2-13](#_Toc8341082)

[2.2.7 Dodajanje in upravljanje s delovnim nalogom 2-13](#_Toc8341083)

[3 Zaključek 3-15](#_Toc8341084)

[4 Sklici 4-16](#_Toc8341085)

[5 Priloge 5-17](#_Toc8341086)

[Slika 1: MEAN skupek 2-2](#_Toc8341087)

[Slika 2: Ukaz za zagon MongoDB konzole 2-10](#_Toc8341088)

[Slika 3: Ukaza za preverjanje instalacije Node-a in NPM-a 2-10](#_Toc8341089)

[Slika 4: Ukaz za instalacijo Angular-ja 2-10](#_Toc8341090)

[Slika 5: Ukaz za instalacijo express-a 2-10](#_Toc8341091)

**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 na naročilo pozabi, saj jih vsak dan dobi par deset. Če pride do napake in se pozabi koliko krofov je bilo naročenih, 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, se 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 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

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 olajša in pohitri delo. Na voljo jih je veliko, od osnovnega jQuery-a, do kompleksnejših, kot so React, Vue, Meteor in Angular. Med temi sem se osebno odločil za Angular, saj so mi ga priporočili ter mi je všeč njegovo 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 le 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.

Angular 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 po navadi 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 razvoje 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 se sploh ne bo 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 je uporabljen v kombinaciji s 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 oni osebi. V tem primeru je en predal ena tabela. Podatki, shranjeni v teh predalih so napisani veni tabeli, podatki med predali so pa povezani s indeksi.

**Shranjevanje podatkov v NOSQL bazah**

Shranjevanje podatkov 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 podobno kot Bootstrap, ki je sicer bolj razširjen.

1. **generate-password**

To knjižnico sem uporabil zato, da sem pri registraciji generiral naključna varna gesla, če bi uporabnik hotel naključno generirano geslo.

1. **bcryptjs**

bcryptjs je knjižnica, ki sem jo uporabil za kodiranje gesel ter od kodiranje 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 uporabniki 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 s passport-jwt, ki preveri veljavnost tokena, poslanega v HTTP zahtevku.

1. **moment**

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

## Praktični del

### Instalacija MongoDB

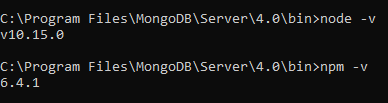
S uradne spletne strani MongoDB naprej prenesemo .msi instalacijsko datoteko, ki jo nato namestimo. V konzoli se premaknemo v naslednjo mapo in zaženemo ukaz 'mongo' da se nam odpre MongoDB shell, v katerem lahko pišemo ukaze za upravljanje s bazo.



Slika : Ukaz za zagon MongoDB konzole

### 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.



Slika : Ukaza za preverjanje instalacije Node-a in NPM-a

### Instalacija Angular-ja in Express-a

Za njuno instalacijo uporabimo NPM s naslednjima ukazoma.



Slika : Ukaz za instalacijo Angular-ja



Slika : Ukaz za instalacijo express-a

### 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

**Postopek registracije :**

1. Uporabnik vnaša podatke, pri tem brskalnik sproti preverja veljavnost podatkov. Zaradi varnosti gesla se celotna forma ne pošlje na strežnik ko 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,
3. 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,
4. Strežnik odgovori s sporočilom ter indeksom novo ustvarjenega uporabnika.

{

"username": "johndoe",

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

"firstName": "John",

"lastName": "Doe",

"password": "MyPass1234"

}

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

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

if (err) throw err;

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

if(err) throw err;

newUser.profile.password = hash;

newUser.save();

callback(null, newUser);

})

})

};

|  |
| --- |
|  |

### Prijava

Da uporabnik dobi že podane podatke se mora prijaviti. Za to potrebuje veljavno uporabniško ime ali elektronski naslov 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
2. Brskalnik pošlje POST zahtevek na strežnik
3. Strežnik preveri, če obstaja podano uporabniško ime ali elektronski naslov, če da preveri še če se gesla ujemata, če se ne odgovori, da so podani podatki napačni
4. Če sta podatka pravilna, strežnik odgovori s sporočilom, novo ustvarjenim žetonom, ki je veljaven 14 dni ter vsemi podatki o uporabniku, razen gesla.
5. Brskalnik shrani žeton in podatke ter uporabnika pošlje na začetno stran, kjer lahko uporablja aplikacijo.

### 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. 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 vsebina 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.

##### 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 čas.

#### 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.

##### 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.

#### Ogledovanje podlag in nalogov

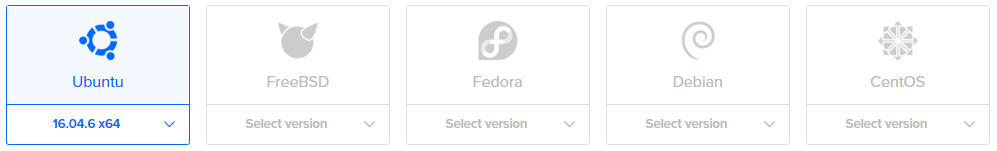
Že dodane delovne naloge pregledujemo, saj si jih moramo med naročilom večkrat ogledati. Razvrstiti jih je možno po datumu, imenu in pomembnosti. Podlage in naloge razširimo s klikom na gumb. Podlage in naloge lahko brišemo. Prvič izbrisani se hranijo v košu, saj v primeru, da je bil nalog ali podlaga izbrisana po pomoti, nevračljiva, v tem sistemu pa jih lahko vrnemo ali izbrišemo dokončno.

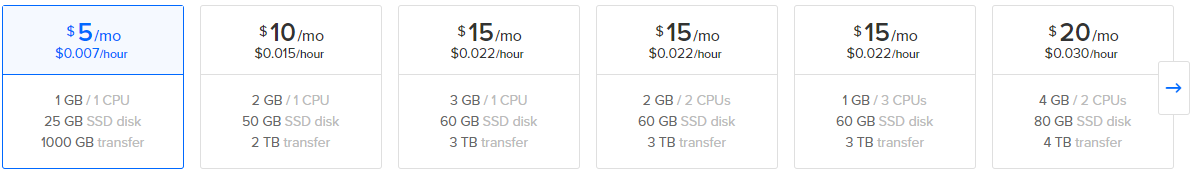
### Postavitev spletnega strežnika

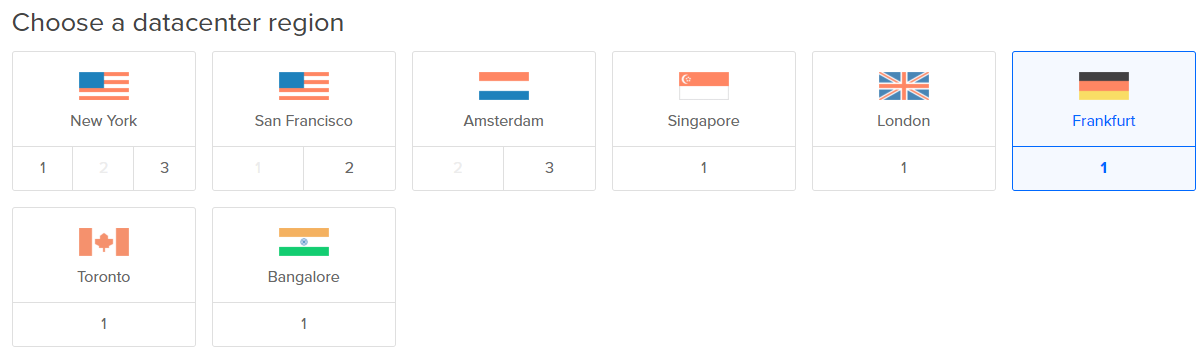
V tem poglavju bom opisal postopke, ki sem jih uporabil za vzpostavitev strežnika.

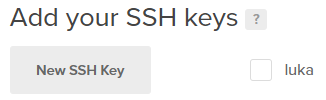
#### Kreiranje 'dropleta' na platformi DigitalOcean

Na platformi DigitalOcean, ki ponuja strežniško gostovanje, sem ustvaril 'droplet', ki je VPS (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).









### 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

# Zaključek

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

Idejo aplikacije mi je zelo všeč in se mi zdi zelo praktično, 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

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

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

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