MEVN Full stack CRUD

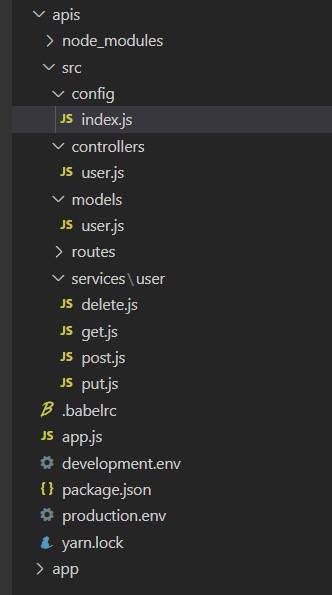
MEVN stack

Mevn(sto znači MongoDB, Express.js, VueJS, Node.js) stack je open-source JavaScript software stack koji služi za pravljenje dinamičkih web aplikacija. Njegove komponente služe za dizajn i frontenda i backenda kao i razvijanja funkcionalnosti web aplikacije.

1. MongoDB: vrsta open source programa za baze podataka koje nemaju striktnu strukturu baze(suprotno od tabelarnih SQL baza) u putem se koji čuvaju JSON dokumenti(kojima možemo dodati opcinalne skeme kako bi eksplicitno modelirali i mapirali podatke u databazi) i time ubrzava rad aplikacije i smanjuje kompleksnost koda.
2. Express.js: back end framework za Node.js I koji se nalazi u okviru njegovog servera. On služi za pravljenje single page, multipage i hibridnih web aplikacija.
3. VueJS: open source front end JavaScript framework koji se koristi za pravljenje UI-a(user interfejsa ili korisničkog interfejsa).
4. Node.js: open source back end JavaScript runtime environment(program koji radi istovremeno I na korisničkom računaru kao I na serveru) namenjen za izvršavanje JavaScript koda izvan web browsera I koji je napravljen za skalabilne network aplikacije.



APIS folder(MEVN backend)



**Package.json**

Ovaj fajl se nalazi u rootu JavaScript/Node projekta(ili u ovom slučaju root je apis folder pošto posebno pokrećemo frontend i backend) i sadrži svu potrebnu metadatu koja je potrebna za njegov rad.

Sadrži opis projekta i opisuje kako se izvršava aplikacija. Koji je privi fajl ili scriptai se učitava kao i sve dependecie i devDependecije zajedno sa njihovim vertijama.

Dependencies su svi eksterni paketi koje smo instalirali u projekat putem nmp(u nasem slucaju yarn CLI) i koje su povučene u nas node\_modules folders. Ovo nam omogućava da lako dodajemo nove dependencije u nas projekat i putem komande **npm install** da povučemo sve njih u slučaju da nam je neka od njih potrebna.

devDependencies su paketi slični dependeci-jama ali oni se samo koriste tokom razvijanja aplikacije i ne koriste se tokom njenog pokretanja. **npm install --save-dev <package>**

1. CORS(*Cross-Origin Resource Sharing*) se koristi za stavljanje restrikciju pristupa resursa koje web server ima putem HTTP upita, ali može i da da pristup podacima van domašaja brauzera.
2. Dotenv je paket koji učitava sve varijable iz .env fajla i pretvara ga u process.env koji koristi za konfiguraciju projekta koji učitava. Time mi ne učitavamo direktno senzitivne podatke i sakrivamo ga od korisnika.
3. Express nam omogućava korišćenje Express frameworka u projektu.
4. Mongoose je Object Data Modeling (ODM) biblioteka koja prevodi objekte iz koda u objekte u MongoDB
5. Nodemon služi da kada dodje do promena u kodu automatski se refresh-uje strana u browser-u.
6. @babel/core je deo babela
7. @babel/node je deo babela
8. @babel/preset-env je deo babela

{

  "name": "apis",

  "version": "1.0.0",

  "main": "app.js",

  "scripts": {

    "start": "nodemon --exec babel-node app.js"

  },

  "license": "MIT",

  "dependencies": {

    "cors": "^2.8.5",

    "dotenv": "^16.0.1",

    "express": "^4.18.1",

    "mongoose": "^6.4.3",

    "nodemon": "^2.0.19"

  },

  "devDependencies": {

    "@babel/core": "^7.18.6",

    "@babel/node": "^7.18.6",

    "@babel/preset-env": "^7.18.6"

  }

}

**.babelrc**

{

    "presets":[

        "@babel/preset-env"

    ]

}

Babel se koristi za konvertovanje ECMAScript koda u JavaScript koda zbog toga što je on idalje u upotrebi kod starijih verzija nekih browser-a.

**App.js**

import express from 'express'

import cors from 'cors'

import './src/config'

import User from './src/routes/user'

require('dotenv').config({path: `./${process.env.NODE\_ENV}.env`})

const app = express()

app.use(express.json())

app.use(express.urlencoded({extended:true}))

app.use(express.static('public'))

app.use(cors())

app.use('/users', User)

app.listen(process.env.PORT)

Import funkcija se koristi za dodavanje eksportovanog koda(funkcije ili moduli).

Express.js: back end framework za Node.js I koji se nalazi u okviru njegovog servera. On služi za pravljenje single page, multipage i hibridnih web aplikacija

CORS(Cross-Origin Resource Sharing) se koristi za stavljanje restrikciju pristupa resursa koje web server ima putem HTTP upita, ali može i da da pristup podacima van domašaja brauzera.

Dotenv je paket koji učitava sve varijable iz .env fajla i pretvara ga u process.env koji koristi za konfiguraciju projekta koji učitava. Time mi ne učitavamo direktno senzitivne podatke i sakrivamo ga od korisnika.

Express.json() služi za konvertovanje json podataka poslatih putem zahteva

Express.urlencoded služi za prasovanje zahteva koji pristižu sa urlencoded payloads.

Process.env.PORT znači da koristi port u aplikaciji iz procesa.

**Config/index.js**

import mongoose from 'mongoose'

mongoose.connect('mongodb://localhost:27017/test')

Mongoose paket se koristi za povezivanje koda sa MongoDb.

Mongoose.connect je funkcija za povezivanje baze sa aplikacijom.

**Controllres/user.js**

import \* as getUserService from '../services/user/get'

import \* as deleteUserService from '../services/user/delete'

import \* as postUserService from '../services/user/post'

import \* as putUserService from '../services/user/put'

export const fetchUsers = async (req, res) => {

    try{

        res.status(200).json((await getUserService.all()))

    }catch(e){

        res.status(500).json(e)

    }

}

export const fetchUser = async (req, res) => {

    try{

        res.status(200).json((await getUserService.byId(req.params.id)))

    }catch(e){

        res.status(500).json(e)

    }

}

export const createUser = async (req, res) => {

    try{

        const{

            name,

            email,

            password

        } = req.body

        res.status(201).json((await postUserService.create({

            name,

            email,

            password

        })))

    }catch(e){

        res.status(500).json(e)

    }

}

export const updateUser = async (req, res) => {

    try{

        const{

            name,

            email,

            password

        } = req.body

        res.status(200).json((await putUserService.byId(req.params.id, {

            name,

            email,

            password

        })))

    }catch(e){

        res.status(500).json(e)

    }

}

export const deleteUser = async (req, res) => {

    try{

        res.status(200).json((await deleteUserService.byId(req.params.id)))

    }catch(e){

        res.status(500).json(e)

    }

}

Async se koristi zajedno sa await funkcijom i koristi se za izvršavanje koda samo u slučaju da kod ispunjava odredjene uslove.(slično try/catch funkciji)

Req odnosi se na ono sto dobijemo kao zahtev kad koristimo ovu funkciju, a res odnosi se na obavezan odgovor koju funkcija daje.

Try i catch služe za trazenje greske i omogućavanju radu koda i ispisivanju sta je kroz statusni kod u slučaju da postoji.

Kada koristimo json response dobijamo uz odgovor statusni kod HTTP-a.

1) Informacioni statusi (100 – 199)

2) Statusi za uspeh (200–299)

3) Redirekcija (300–399)

4) Statusi klijentskih grešaka(400–499)

5) [Statusi](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#server_error_responses) serverskih grešaka (500–599)

200 Ok znači da je uspeo zahtev.

201 znači da je zahtev uspeo i da je kreirano u bazi nešto novo.

400 server ne može ili neće da obradi zahtev zbog klijentskog error-a(greške u sintaksi zahteva npr).

401 klijent mora proći kroz autentifikaciju da bi dobio odgovor.

403 klijent nema pristup sadržaju i server odbija da mu da odgovor. Razlika u odnosu na 401 je to što je identitet klijent poznat serveru.

404 server ne može da pronađe zahtevani resurs. U brauzeru to znači da URL nije prepoznat. U API-ju to može isto da znači da je krajnja tačka to jest endpoint ispravna ali resurs ne postoji. Ovaj odgovor kod je najpoznatiji zbog njegovog čestog  pojavljivanja na webu.

405 metoda nije dozvoljena

408 Ovaj odgovor se šalje na ne aktivnu konekciju od strane pojedinih servera, čak I bez prethodnih zahteva od klijenta.

500 server je naišao na grešku koju ne zna kako da reši.

501  zahtev nije podržan od strane servera i ne može biti hendlovan.

502 Ovaj error odgovor znači da je server dobio neispravan odgovor.

503 Server nije spreman da hendluje zahtev. Uobičajeni slučajevi su da je server pao zbog održavanja ili da je “prenatrpan”.

504  Ovaj error odgovor je dat kada server ima ulogu prolaza I ne može da preuzme odgovor na vreme.

505 HTTP verzija korišćena u zahtevu nije podržana od strane servera.

**Models/user.js**

import mongoose from "mongoose"

export const User = mongoose.model('user', new mongoose.Schema({

    name : String,

    email : String,

    password : String

}))

Mongoose.model služi za kontrolu nad bazom I vršenje upita u kodu.

Mongoose.Schema služi definiše strukturu kolekcije u bazi I tipove podataka.

U ovom slučaju na slici iznad je tip podataka String što znači tekstualni tip.

**Routes/user.js**

import {Router} from 'express'

import \* as userController from '../controllers/user'

const User = Router()

User.get('/', userController.fetchUsers)

User.get('/:id', userController.fetchUser)

User.post('/', userController.createUser)

User.put('/:id', userController.updateUser)

User.delete('/:id', userController.deleteUser)

export default User

Kada koristimo vitičaste zagrade to znači da importujem samo onaj deo sa tim nazivom, a kada koristimo \* to znači da importujemo sve što je dostupno za export iz tog fajla.

Router() sadrži funkcije ta get, post, put i delete I ima dva parametra. Prvi parametar se odnosi nastavak url adrese posle endpointa(tj. Posle .com/), a drugi se odnosi na koj kontroler i koju njegovu funkciju poziva za izvršavanje.

**Services/user**

import { User } from "../../models/user"

export const byId = async (id) => User.findByIdAndDelete(id)

import { User } from "../../models/user"

export const all = async () => User.find({})

export const byId = async (id) => User.findById(id)

import { User } from "../../models/user"

export const create = async (user) => User.create(user)

import { User } from "../../models/user"

export const byId = async (id, user) => User.findByIdAndUpdate(id, user)

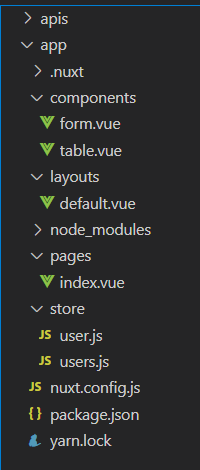
find() je funkcija koja služi za nalaženje svih objekata koji ispunjavaju uslov u parametru.

findById() je funkcija koja u bazi nalazi i vraća sve objekte koji imaju traženi id.

Create() je funkcija koja pravi novu instancu objekta u bazi.

FindByIdAndDelete() je funkcija koja traži odredjeni objekat sa tim id-jem I brise ga iz baze podataka.

APP folder(MEVN frontend)



**Package.json**

{

  "name": "app",

  "version": "1.0.0",

  "main": "index.js",

  "scripts": {

    "start": "nuxt"

  },

  "license": "MIT",

  "dependencies": {

    "@nuxtjs/axios": "^5.13.6",

    "@nuxtjs/vuetify": "^1.12.3",

    "nuxt": "^2.15.8"

  }

}

Nuxt je besplatna JavaScript biblioteka koja je bazirana da React.js-u. Ona služi za prikaz web komponenti iz Vue.js-a I prikazuje njegov rad sa JavaScript kodom.

Nuxtjs/axios služi za nameštanje URL adrese za klijentsku i serversku stranu, kao i za nameštanje tokena za autentifikaciju.

Nuxtjs/vuetify daje još alata pri pravljenju Vue.js fajlova.

**Nuxt.config.json**

export default{

    buildModules : [

        "@nuxtjs/vuetify"

    ],

    modules : [

        "@nuxtjs/axios"

    ],

    components: true

}

BuildModules se koriste samo takom pravljenja aplikacije I ponašaju se ko devDependency.

**Layouts/default.vue**

<template>

  <div>

    <v-app>

        <v-main>

            <v-app-bar>

                <v-app-bar-title>

                    Home

                </v-app-bar-title>

            </v-app-bar>

            <v-container justify="center" align="center">

                <Nuxt />

            </v-container>

        </v-main>

    </v-app>

  </div>

</template>

<script>

export default {

}

</script>

<style>

</style>

<v-app> I <v-main> su obavezni da se ubace u kod.

<Nuxt /> u ovom slučaju učitava iz pages foldera template.

**Pages/index.vue**

<template>

  <v-row>

    <v-col>

        <Form/>

    </v-col>

    <v-col>

        <Table/>

    </v-col>

  </v-row>

</template>

<script>

export default {

}

</script>

<style>

</style>

<v-row> služi za dodavanje reda.

<v-col> služi za dodavanje kolone.

<Form> i <Table/> učitavaju kod iz components folder-a.

**Store/users.vue**

export const state = () => ({

    data: []

})

export const mutations = {

    storeData: (state, data)=>{

        state.data = data

    }

}

U primeru iznad se čuvaju podaci o korisnicima u nizu.

Mutacije se služe za da se preuzimaju statičke vrednosti I menjaju praveći nove instance. One se sastoje iz prethodnog stanja(state) I neke akcije(action) koja izaziva promenu i pravi novo stanje.

**Store/user.vue**

export const state = () => ({

    id: "",

    name: "",

    email: "",

    password: ""

})

export const mutations = {

    setId: (state, data)=>{

        state.id = data

    },

    setName: (state, data)=>{

        state.name = data

    },

    setEmail: (state, data)=>{

        state.email = data

    },

    setPassword: (state, data)=>{

        state.password = data

    }

}

U ovom slučaju se čuva korisnik.

Const je ključna reč koja definiše da vrednost varijable nije promenljiva I ne moze se manipulisati u daljem kodu.

Varijable u state se koriste za čuvanje podataka I njihov prikaz na stranici.

**Components/table.vue**

<template>

    <div>

        <br/>

        <h2>List of Users</h2>

        <v-data-table :headers="headers" :items="users">

            <template v-slot:[`item.edit`]="{item}">

                <v-btn color="success" @click="editItem(item.\_id)">Edit</v-btn>

            </template>

            <template v-slot:[`item.delete`]="{item}">

                <v-btn color="danger" @click="deleteItem(item.\_id)">Delete</v-btn>

            </template>

        </v-data-table>

    </div>

</template>

<script>

export default {

    data(){

        return {

            headers: [

                {text: 'Id', value: '\_id'},

                {text: 'Name', value: 'name'},

                {text: 'Email', value: 'email'},

                {text: 'Password', value: 'password'},

                {text: 'Edit', value: 'edit'},

                {text: 'Delete', value: 'delete'},

            ]

        }

    },

    computed: {

        users(){

            return this.$store.state.users.data

        }

    },

    async fetch(){

        this.$store.commit(

            "users/storeData",

            (await this.$axios.get("http://localhost:4000/users")).data

        )

    },

    methods: {

        async deleteItem(id){

            await this.$axios.delete("http://localhost:4000/users/" + id)

            this.$store.commit(

                "users/storeData",

                (await this.$axios.get("http://localhost:4000/users")).data

            )

        },

        async editItem(user){

            this.$store.commit("user/setId", user.\_id);

            this.$store.commit("user/setName", user.name);

            this.$store.commit("user/setEmail", user.email);

            this.$store.commit("user/setPassword", user.password);

        }

    }

}

</script>

U kodu iznad iznad imamo headers u koji je ubačen headers niz objekata iz data u skripti. Imamo i items koji sadrži korisnike koji se čuvaju u stejtu u store-u.

Ovo v-slot item.edit da je jednako {item} se piše da bi se koristio taj korisnik i menjali njegovi podaci.

@click editItem sačuva podatke u state i onda u formi sa leve strane se pojave ti podaci o korisniku.

@click deleteItem briše korisnika iz baze podataka.

U data imamo headers u kojem imamo objekte sa tekstom i vrednosti korisnika.

U computed se stavljaju podaci koji se keširaju i kada se refrešuje stranica te funkcije se ne izvršavaju iznova.

**Components/form.vue**

<template>

    <div>

        <br/>

        <h2>Add User</h2>

        <v-form>

            <v-text-field v-if="id" v-model="id" label="Id"></v-text-field>

            <v-text-field v-model="name" label="Name"></v-text-field>

            <v-text-field v-model="email" label="Email"></v-text-field>

            <v-text-field v-model="password" label="Password"></v-text-field>

            <v-btn @click="submit({id,name,email,password})">{{ id ? "Edit" : "Submit" }}</v-btn>

        </v-form>

    </div>

</template>

<script>

export default {

    computed: {

        id: {

            get(){

                return this.$store.state.user.id

            },

            set(value){

                this.$store.commit("user/setId", value);

            }

        },

        name: {

            get(){

                return this.$store.state.user.name

            },

            set(value){

                this.$store.commit("user/setName", value);

            }

        },

        email: {

            get(){

                return this.$store.state.user.email

            },

            set(value){

                this.$store.commit("user/setEmail", value);

            }

        },

        password: {

            get(){

                return this.$store.state.user.password

            },

            set(value){

                this.$store.commit("user/setPassword", value);

            }

        }

    },

    methods: {

        async submit(user){

            if(user.id){

                await this.$axios.put("http://localhost:4000/users/" + user.id, {

                    name: user.name,

                    email: user.email,

                    password: user.password

                });

            } else {

                await this.$axios.post("http://localhost:4000/users/",{

                    name: user.name,

                    email: user.email,

                    password: user.password

                });

            }

            this.resetUser({id : 0, name : "", email : "", password : ""});

            this.$store.commit(

                "users/storeData",

                (await this.$axios.get("http://localhost:4000/users")).data

            )

        },

        resetUser(user){

            this.$store.commit("user/setId", user.id)

            this.$store.commit("user/setName", user.name)

            this.$store.commit("user/setEmail", user.email)

            this.$store.commit("user/setPassword", user.password)

        }

    }

}

</script>

U primeru iznad kada postoji id u stejtu onda za dugme piše edit, a kada ne postoji piše Submit.

V-model pravi vezu varijabla sa stejtom tako da u tim v-text-field-ovima imamo vrednosti iz stejta.

U ovom slučaju funkcije iz computed nisu iskorišćene u kodu.

Axios kontaktira sa bazom i vrši izmenu podataka sa put ako id postoji.

Ako id ne postoji onda se pravi novi korisnik.

Kada se to izvrši onda se resetuje stejt na početno stanje.