# Menghubungkan Vue.js dengan API Node.js

Pada pertemuan kali ini, kita akan menghubungkan tampilan frontend yang telah dibuat pada pertemuan sebelumnya menggunakan Vue.js dengan API yang dibuat menggunakan Node.js.

Buatlah project baru pada VSCode dan jalankan perintah npm init. Pada direktori project tersebut, buatlah direktori-direktori dan file sebagai berikut:

* ./api
* ./api/middlewares
* ./api/routes
* ./api/validators
* ./controllers
* ./models
* index.js

Kemudian pada package.json, tambahkan package-package berikut terlebih dahulu, kemudian jalankan perintah npm install.

Text

Description automatically generated

Penjelasan dari beberapa package di atas yang belum pernah dibahas pada pertemuan sebelumnya antara lain:

* @hapi/joi dan celebrate untuk validasi data input
* cors agar API yang dibuat dapat diakses dari project lain

Selanjutnya, isilah potongan kode berikut pada index.js.

const express = require('express')

const bodyParser = require('body-parser')

const cors = require('cors')

const mongoose = require('mongoose')

const app = express()

// enable cross origin resource sharing to all origins by default

app.use(cors())

// middleware that transforms the raw string of req.body into json

app.use(bodyParser.json())

// connect to mongoose database

mongoose.connect(

'<URL STRING MONGODB>',

{ useNewUrlParser: true, useUnifiedTopology: true }

)

// check db connection

const db = mongoose.connection

db.once('open', () => {

console.log("Successfully connected to MongoDB using Mongoose!")

})

// define URL routes

const users = require('./api/routes/users')

app.use('/users', users)

// error handling

app.use((*err*, *req*, *res*, *next*) => {

res.status(err.status || 500)

res.json({

errors: {

message: err.message

}

})

})

// listen to port

app.listen(9999)

console.log('Server runs at port 9999')

Jangan lupa mengisi <URL STRING MONGODB> di atas dengan informasi dari MongoDB kalian masing-masing.

Dalam memproses sebuah input dari frontend, kita perlu memvalidasi data yang dikirimkan agar sesuai dengan ketentuan yang diinginkan. Pada contoh ini, kita akan membuat API yang menerima data user dan kemudian menyimpannya ke database. Seperti pada contoh yang dibuat pada pertemuan minggu lalu di Vue.js, data user yang disimpan antara lain nama lengkap (fullname), email, username, dan password.

Misalkan email yang dikirim harus menggunakan format email yang benar. Bisa juga kita membuat aturan agar password harus minimal 6 karakter. Aturan-aturan input ini dapat divalidasi secara otomatis menggunakan package Joi dan celebrate.

Tambahkan kode program di bawah ini pada ./api/validators/users.js.

const Joi = require('@hapi/joi')

module.exports = {

createUser: {

body: {

fullname: Joi.string().max(200).required(),

email: Joi.string().email().required(),

username: Joi.string().min(3).max(32).required(),

password: Joi.string().min(6).max(32).required()

}

},

updateUser: {

body: {

fullname: Joi.string().max(200).required(),

email: Joi.string().email().required(),

username: Joi.string().min(3).max(32).required(),

password: Joi.string().min(6).max(32).required()

}

}

}

Pada contoh rule di atas, kita membuat 2 aturan untuk proses create dan update user.

Selanjutnya, kita akan membuat model untuk menyimpan informasi user ke database. Tambahkan kode program di bawah ini pada ./models/users.js.

const mongoose = require('mongoose')

const users = mongoose.model('users', new mongoose.Schema({

username: String,

fullname: String,

email: String,

password: String

}))

module.exports = users

Langkah selanjutnya adalah membuat file ./controllers/users.js yang menjembatani penyimpanan data ke database.

const User = require('../models/users')

async function findAll () {

return await User.find({})

}

async function findOne (*id*) {

return await User.findOne({ \_id: id })

}

async function create (*fullname*, *email*, *username*, *password*) {

return await User.create({

fullname: fullname,

email: email,

username: username,

password: password

})

}

async function update (*id*, *fullname*, *email*, *username*, *password*) {

return await User.updateOne({

\_id: id

}, {

fullname: fullname,

email: email,

username: username,

password: password

})

}

async function remove (*id*) {

return await User.remove({ \_id: id })

}

module.exports = {

findAll,

findOne,

create,

update,

remove

}

Langkah terakhir adalah menambahkan kode program pada ./api/routes/users.js untuk meng-*handle* setiap URL yang diakses untuk mengubah database users.

const express = require('express');

const { celebrate } = require('celebrate')

const UserController = require('../../controllers/users')

const UserValidator = require('../validators/users');

const router = express.Router()

// find all users

router.get('/',

async (*req*, *res*) => {

const users = await UserController.findAll()

return res.json(users)

}

)

// get user detail

router.get('/:id',

async (*req*, *res*) => {

const id = req.params.id

const user = await UserController.findOne(id)

return res.json(user)

}

)

// create new user

router.post('/',

celebrate(UserValidator.createUser),

async (*req*, *res*) => {

await UserController.create(

req.body.fullname,

req.body.email,

req.body.username,

req.body.password

)

return res.json({ status: 'OK' })

}

)

// update user

router.put('/:id',

async (*req*, *res*) => {

await UserController.update(

req.params.id,

req.body.fullname,

req.body.email,

req.body.username,

req.body.password

)

return res.json({ status: 'OK' })

}

)

// delete user

router.delete('/:id',

async (*req*, *res*) => {

await UserController.remove(req.params.id)

return res.json({ status: 'OK' })

}

)

module.exports = router

Setelah kode program untuk backend API di atas selesai, kalian bisa mencoba terlebih dahulu untuk mengecek apakah masih ada kesalahan dengan menggunakan aplikasi seperti Postman.

Cobalah untuk melakukan request ke:

* <http://localhost:9999/users> menggunakan method GET untuk mendapatkan daftar semua user di database.
* [http://localhost:9999/users/<id](http://localhost:9999/users/%3cid)> menggunakan method GET dan <id> merupakan ID dari user untuk mendapatkan detail dari seorang user dengan ID tersebut.
* <http://localhost:9999/users> menggunakan method POST untuk menyimpan data user baru. Tambahkan juga data pada body yang terdiri dari fullname, email, username, dan password.
* [http://localhost:9999/users/<id](http://localhost:9999/users/%3cid)> menggunakan method PUT untuk mengubah data user dengan ID tertentu. Tambahkan juga data pada body yang terdiri dari fullname, email, username, dan password.
* [http://localhost:9999/users/<id](http://localhost:9999/users/%3cid)> menggunakan method DELETE untuk menghapus user dengan ID tersebut.

Graphical user interface

Description automatically generated

Untuk menyambungkan Vue.js dengan backend API yang baru saja dibuat, sekarang kita berpindah ke project Vue.js yang telah dibuat pada pertemuan minggu lalu.

Instalasi package yang dinamakan axios yang berguna untuk melakukan request ke API.

$ npm install --save axios

Pertama-tama, ubahlah ./src/components/User.vue untuk menambahkan kode untuk mendapatkan daftar user pada database dan menampilkannya ke table. Pada file tersebut juga tambahkan kode untuk memanggil API delete.

<template>

<section class="section">

<div class="container is-max-desktop">

<h1 class="title">Users</h1>

<hr>

<div class="buttons">

<b-button class="is-primary" tag="router-link" to="/user-add">

Add New User

</b-button>

</div>

<b-table :data="users" :loading="isLoading">

<b-table-column label="No" field="no" v-slot="props" width="50">

{{ props.index + 1 }}

</b-table-column>

<b-table-column label="Full Name" field="fullname" v-slot="props"

sortable>

{{ props.row.fullname }}

</b-table-column>

<b-table-column label="Email" field="email" v-slot="props"

sortable>

{{ props.row.email }}

</b-table-column>

<b-table-column label="Username" field="username" v-slot="props"

sortable>

{{ props.row.username }}

</b-table-column>

<b-table-column custom-key="actions" class="is-actions-cell"

v-slot="props">

<div class="buttons is-right">

<b-button class="is-secondary" tag="router-link" :to="{ name: 'UserDetail', params: { id: props.row.\_id }}">Edit</b-button>

<b-button class="is-danger" @click="trashModal(props.row.\_id, props.row.fullname)">Delete</b-button>

</div>

</b-table-column>

</b-table>

</div>

</section>

</template>

<script>

import axios from 'axios'

export default {

data () {

return {

users: [],

isLoading: false

}

},

mounted () {

this.isLoading = true

this.loadData()

},

methods: {

loadData () {

axios.get('http://localhost:9999/users')

.then((*r*) => {

if (r.data) {

this.users = r.data

}

})

.catch((*e*) => {

this.$buefy.snackbar.open({

message: 'Error!',

type: 'is-danger',

position: 'is-top'

})

})

.finally(() => {

this.isLoading = false

})

},

trashModal (*id*, *name*) {

this.$buefy.dialog.confirm({

title: 'Delete Confirmation',

message: `Are you sure want to delete <b>${name}</b>?`,

confirmText: 'Yes',

cancelText: 'Cancel',

type: 'is-danger',

onConfirm: () => {

axios.delete(`http://localhost:9999/users/${id}`)

.then((*r*) => {

this.loadData()

this.$buefy.snackbar.open({

message: 'Success!',

type: 'is-success',

position: 'is-top'

})

})

.catch((*e*) => {

this.$buefy.snackbar.open({

message: 'Error!',

type: 'is-danger',

position: 'is-top'

})

})

}

})

}

}

}

Untuk melakukan request ke API dengan method GET, maka gunakan perintah sebagai berikut:

axios.get(url)

.then((r) => {

// do something when the request is a success

})

.catch((e) => {

// do something when the request returns an error

})

.finally(() => {

// do something when all is done

})

Begitu juga dengan delete, gunakan perintah axios.delete(url).

Kemudian, untuk menambahkan user, ubahlah kode ./src/components/UserAdd.vue menjadi seperti di bawah ini:

<template>

<section class="section">

<div class="container is-max-desktop">

<h1 class="title">Add New User</h1>

<hr>

<section>

<b-field label="Full Name">

<b-input v-model="user.fullname" required />

</b-field>

<b-field label="Email">

<b-input v-model="user.email" type="email" maxlength="255" required />

</b-field>

<b-field label="Username">

<b-input v-model="user.username" maxlength="30" required />

</b-field>

<b-field label="Password">

<b-input v-model="user.password" type="password" maxlength="30" required />

</b-field>

<b-button type="is-success" @click="saveUser" expanded>Save</b-button>

<br>

<router-link to="/user">Cancel</router-link>

</section>

</div>

</section>

</template>

<script>

import axios from 'axios'

export default {

data () {

return {

user: {

fullname: '',

email: '',

username: '',

password: ''

}

}

},

methods: {

saveUser () {

axios.post('http://localhost:9999/users', {

fullname: this.user.fullname,

email: this.user.email,

username: this.user.username,

password: this.user.password

})

.then((*r*) => {

this.$buefy.snackbar.open({

message: 'Success!',

type: 'is-success',

position: 'is-top'

})

this.$router.push('/user')

})

.catch((*e*) => {

this.$buefy.snackbar.open({

message: e.response.data.errors.message,

type: 'is-danger',

position: 'is-top'

})

})

}

}

}

</script>

Perbedaan utama kode di atas dengan User.vue adalah pada pemanggilan API untuk menyimpan user menggunakan method POST, yaitu axios.post(url, data).

Untuk edit user pada ./src/components/UserDetail.vue diubah menjadi seperti di bawah ini:

<template>

<section class="section">

<div class="container is-max-desktop">

<h1 class="title">Edit User</h1>

<hr>

<section>

<b-field label="Full Name">

<b-input v-model="user.fullname" />

</b-field>

<b-field label="Email">

<b-input v-model="user.email" type="email" maxlength="255" />

</b-field>

<b-field label="Username">

<b-input v-model="user.username" maxlength="30" />

</b-field>

<b-field label="Password">

<b-input v-model="user.password" type="password" maxlength="30" />

</b-field>

<b-button type="is-success" @click="saveUser" expanded>Save</b-button>

<br>

<router-link to="/user">Cancel</router-link>

</section>

</div>

</section>

</template>

<script>

import axios from 'axios'

export default {

data () {

return {

user: {

fullname: '',

email: '',

username: '',

password: ''

}

}

},

mounted () {

axios.get(`http://localhost:9999/users/${this.$route.params.id}`)

.then((*r*) => {

if (r.data) {

this.user.fullname = r.data.fullname

this.user.email = r.data.email

this.user.username = r.data.username

}

})

},

methods: {

saveUser () {

axios.put(`http://localhost:9999/users/${this.$route.params.id}`, {

fullname: this.user.fullname,

email: this.user.email,

username: this.user.username,

password: this.user.password

})

.then((*r*) => {

this.$buefy.snackbar.open({

message: 'Success!',

type: 'is-success',

position: 'is-top'

})

this.$router.push('/user')

})

.catch((*e*) => {

this.$buefy.snackbar.open({

message: e.response.data.errors.message,

type: 'is-danger',

position: 'is-top'

})

})

}

}

}

</script>

Pada saat halaman UserDetail.vue ini dibuka, maka kode program pada mounted() akan dijalankan untuk mendapatkan informasi detail dari seorang user. Kemudian, untuk mengubah data melalui API dilakukan menggunakan pemanggilan axios.put(url, data).

## Deploy ke Heroku

Pada bagian ini akan dijelaskan singkat mengenai langkah-langkah *deployment* project API di atas ke Heroku.

Mula-mula, lakukan instalasi Heroku CLI dengan mengikuti langkah-langkah pada halaman web berikut <https://devcenter.heroku.com/articles/heroku-cli>.

Pada package.json, tambahkan kode untuk menyatakan versi Node.js yang digunakan.

"engines": {

"node": "12.x"

}

Pada index.js, tambahkan kode berikut ini pada bagian penentuan PORT aplikasi anda.

const PORT = process.env.PORT || 3000;

app.listen(PORT)

Jika nanti pada saat menjalankan program pada Heroku dan anda menemukan error R10 karena port, cobalah melihat dokumentasi berikut <https://help.heroku.com/P1AVPANS/why-is-my-node-js-app-crashing-with-an-r10-error>.

Pastikan project anda telah ada di dalam local repository git (telah dilakukan add dan commit). Pada terminal, jalankan perintah di bawah ini untuk login ke akun Heroku anda.

$ heroku login

Kemudian buatlah project ini menjadi project Heroku dengan menjalankan perintah

$ heroku create

Jalankan perintah ini untuk melakukan deploy ke Heroku.

$ git push heroku master

Program anda telah di-*deploy* ke Heroku. Alamat URL untuk mengakses API anda di Heroku akan ditampilkan di layar.

Untuk menghubungkan frontend yang telah dibuat dengan Vue.js di atas, ubahlah pemanggilan axios ke <http://localhost:9999> menjadi alamat URL Heroku.