Kopienas mājas lapa

MERN aplikācijas izstrāde

Pasākumu finansē: Eiropas Jūrlietu un zivsaimniecības fonds, projekts "Viedo ciemu attīstība piekrastes teritorijās". Nr. 19-00-F043.0443-000001.
VRG "Partnerība laukiem un jūrai",sadarbībā ar vadošo partneri VRG "Liepājas rajona partnerība"

NACIONĀLAIS ATTĪSTĪBAS PLĀNS 2020

EIROPAS SAVIENĪBA
Eiropas Jūrlietu un zivsaimniecības fonds

Atbalsta Zemkopības ministrija un Lauku atbalsta dienests

Mēs zinam

- MERN aplikācija - kas tas ir
- Visual studio Code un izmantotās bibilotēkas
- Klients, Serveris un bibliotēku sagatave
- Pirmā mūsu klienta aplikācija
- Kā izveidot savu DB
- Kā pieslēgt mūsu viedoto aplikāciju DB

Iesākām pirmo Full Stack aplikāciju, ko radam paši

Šodien uzzināsim

- API izveide kas darbojas ar mūsu Mongo DB
- API testi uz testa rīkiem:
    - Pārlūka
    - Postman

Solis tuvāk. Pirmā Full Stack aplikācija, ko radam paši

MERN aplikācija

MERN nozīmē - MongoDB, Express, React, Node - tās ir 4 atslēgas tehnoloģijas, kas nodrošina pilnu izstrādes vidi:

- MongoDB - dokumentu datubāze
- Express(.js) - Node.js WEB framework
- React(.js) - klienta puses JavaScript framework
- Node(.js) - pamata framework JavaScript web serverim

Ko mēs tehniski gribam uzprogrammēt?

CRUD Nozīme:

CRUD ir abreviatūra no programmēšanas pasaules, kas apzīmē 4 funkcijas. Tās nepieciešamas lai izveidot pamata lietas jebkurai applikācijai.
C - create - izveidot ierakstu
R - read - nolasīt ierakstu no datu glabātuves
U - update - atjaunot/labot ierakstu datu glabātuvē,
D - delete - dzēst ierakstu no datu glabātuves.

Kā tas izskatīsies?

Ziņojumu dēlis

Daina
7 days ago

#vietejais #svaigs #jauks

26565812

Tas ir mans ziņojums

LIKE 3 DELETE

Jolanta
7 days ago

#zemenes #vasara #2021

3425322

Šodien pie manis ir zemenes

LIKE 1 DELETE

Jauns ziņojums

Autors: Daina

Kontakti

Ziņojums

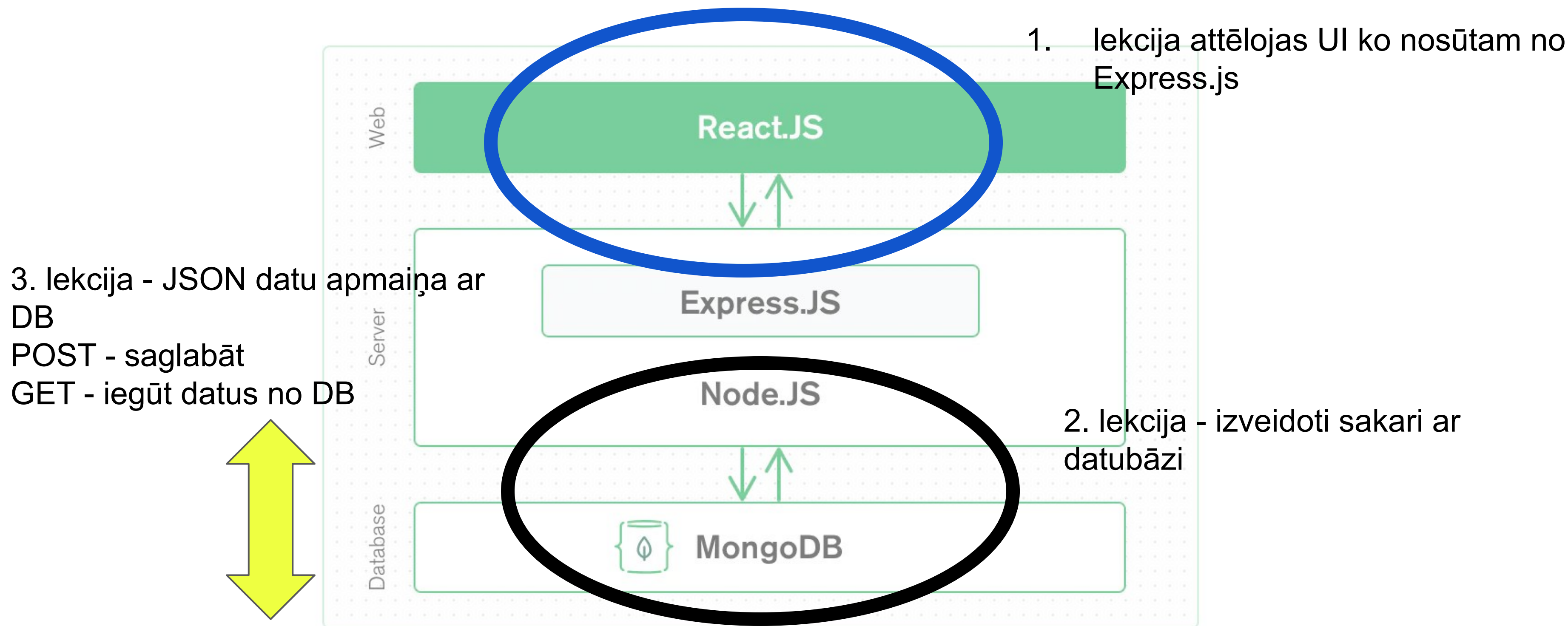
Hashtagi

Choose File No file chosen

SAGLABĀT

NOTĪRĪT

Arhitektūra?



1. Iekcija attēlojas UI ko nosūtam no Express.js

2. Iekcija - izveidoti sakari ar datubāzi

3. Iekcija - JSON datu apmaiņa ar DB
POST - saglabāt
GET - iegūt datus no DB

Vides sagatavošana

Ir jābūt izietām iepriekšējām divām lekcijām, kuru laikā sagatvojām aplikācijas pamatu un testējām pieslēgumu.

DB runā ar Backend JSON formātā

server

1. izveidojam direktoriju controllers un tur failu posts.js
te mes liksim visas detaļas (biznesa loģiku) backend-am, lai routes/posts.js būti tīrāks un vieglāk lasāms, kad mums pievienosies jauni ceļi

2. controllers/posts.js definējam funkcija

```
export const getPosts = (req,res) => {
res.send('DUMMY WORKS!');}
```

3. importējam controllers/post.js failā routes/posts.js

DB runā ar Backend JSON formātā

server

3. importējam controllers/post.js failā routes/posts.js funkciju

```
 import {getPosts} from ‘../controllers/posts.js’;

//shis ir jālabo
router.get(‘/’, getPosts);
```

// Tagad pievienojot arvien jaunus ceļus, fails kurš definē ceļus būs krietni tīrāks un vieglāk uztverams

DB runā ar Backend JSON formātā

server

4. tagad veidosim modeli kā dati glabāsies datubāzē. izveidojam models direktoriju, kurā definējam failu postMessage.js

5. definējam postMessages.js failā DB vajadzīgās funkcijas izmantojot mongoose bibliotēkas

```
import mongoose from ‘mongoose’;
```

6. definējam shēmu, kā saglabāt datus

```
const postSchema = mongoose.Schema({
})
```

DB runā ar Backend JSON formātā

server

7. ar shēmu mēs pasakam kādā formātā glabāsies dati, mūsu gadījumā

```
const postSchema = mongoose.Schema({
title: String,
message: String,
creator: String,
tags: [String],
selectedFile: String,
likeCount: {type: Number, default: 0},
createdAt: {type: Date, default: new Date()}
})

//definējam to kā modeli
const postMessage = mongoose.model('PostMessage', postSchema);

// un exportējam modeli
export default PostMessage;
```

DB runā ar Backend JSON formātā

server

8. tagad izmantojot definēto modeli mēs varēsim darbināt komandas, kas meklē, labo, dzēš un izveidot ierakstus

9. Tagad mums ir datu modelis un mēs varam pievienot jaunus ceļus, kas darbojas ar datiem (routes)

10. server/routes/post.js fails
 router.get('/',getPosts);
router.post('/', createPost);

11. createPost mēs importējam no controllers/posts.js
import {getPosts, createPost} …..

DB runā ar Backend JSON formātā

server

8. ejam uz controllers/posts.js un definējam jauno funkciju

```
export const createPost = (req, res)  =>
{ res.send(‘Post creation’);
}
```

Protams ka redzam ka šobrīd abas funkcijas kas definētas controllers/posts.js neko nedara. Tagad vajadzētu definēt reālu loģiku, kas izveido ierakstu DB un kas salasa visus DB ierakstus.

DB runā ar Backend JSON formātā

server

9. importējam mūsu shēmas datu modeli PostMessage

```
import PostMessage from ‘../models/postMessage.js’;
```

10. Katrai funkcijai ir jāpievieno try un catch bloks.

```
try {
//te būs kods, kas izpildīsies ja viss sanāks
}
catch (error) {
//te būs kods, kas izpildīsies ja būs kļūda
}
```

DB runā ar Backend JSON formātā

server

11. controllers/posts.js

```
export const getPosts = async (req, res) => {
try {
//te būs kods, kas izpildīsies ja viss sanāks, funkcijas ir asinhrona
        const postMessages = await PostMessage.find();

        res.status(200).json(postMessages);
}
catch (error) {
//te būs kods, kas izpildīsies ja būs kļūda
        res.status(404).json({message: error.message});
}
}
```

DB runā ar Backend JSON formātā

server

11. Tagad to var nočekot uz mūsu servera localhost:5000/posts, kur parādās tukšs masīvs

[]

12. Tagad pēc līdzīga principa definēsim funckiju, kas pievieno datus DB createPost

DB runā ar Backend JSON formātā

server

13. uzlabojam savu createPost funkciju

```
export const createPost = async (req, res) => {
        const post = req.body;
        const newPost = new PostMessage(post);
try {
//te būs kods, kas izpildīsies ja viss sanāks, funkcijas ir asinhrona
        await newPost.save();
        res.status(201).json(newPost);
}
catch (error) {
//te būs kods, kas izpildīsies ja būs kļūda
        res.status(409).json({message: error.message});
}
}
```

//https://www.restapitutorial.com/httpstatuscodes.html - te vairāk info par atbildes kodiem

Ko tālāk

4. lekcija - vizualizācija

UI

1. lekcija attēlojas UI ko nosūtam no Express.js

Web

React.JS

Server

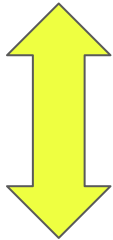
Express.JS

Node.JS

Database

MongoDB

3. lekcija - JSON datu apmaiņa ar DB
POST - saglabāt
GET - iegūt datus no DB



izveidoti sakari ar datubāzi
mums ir pirmais dummy API

Apmācības nodrošina NVO “Piekrastes Konvents”. Vairāk info - info@piekrasteskonvents.lv