Routing:

<https://expressjs.com/en/guide/routing.html>

<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes>

Skapa en routes mapp och skapa admin.js och products.js fil:

I admin.js filen :

const express = require('express');

const router = express.Router();

router.get('/add-product', (req, res, next) => {

res.send("adding data... ");

});

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

console.log(req.body);

res.redirect('/');

});

module.exports = router;

I product.js filen:

const express = require('express');

const router = express.Router();

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

res.send('<h1>Hej hej</h1>');

});

module.exports = router;

OBS: vi har exporterat router

Och i app.js:

const express = require('express');

const app = express();

const adminRoutes = require('./routes/admin');

const productRoutes = require('./routes/product');

app.use("/admin" , adminRoutes);

app.use(productRoutes);

const port = process.env.PORT || 3004;

app.listen(port, () => console.log(`Listening on port ${port}...`));

Övningar:

* Flytta data till en egen fil. till ex. flytta product listan från main.js fil till product.js fil och lägg filen in i model folder.
* Från din tidigare övningar med get, post osv :

Flytta alla dina router i olika moduler till ex. productRouter , adminRouter

* importera och anropa dem(filerna/modulerna) i main.js/app.js/index.js så att den funkar som förut.
* Skapa en 404 route så att det kan visa Page not found meddelandet när använder skriver fel route.

----------------------------------------

Upprepning: callback, promises, async , await.

async sync programmering:

html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

<script src="app.js" defer></script>

</head>

<body>

<div class="main">

<input type="text" id="id1"><input type="text" id="id2">

<button>

click me

</button>

</div>

</body>

</html>

app.js:

//den kör sync

const input1 = document.querySelector("#id1").value;

// den kör sync

const input2 = document.querySelector("#id2").value;

// den byggs upp men kör inte tills man anropar den

function callback(){

console.log(' Show me the value ', input1 )

}

//den kör async

document.querySelector("button").addEventListener("click", callback)

// den kör sync

const mainDiv =document.querySelector(".main")

//den kör sync

console.log(mainDiv)

Lets block the code for a moment :

let summa =0;

for(let i= 0; i<2000000000; i++){

summa += 1;

}

console.log(summa)

//den kör sync

const input1 = document.querySelector("#id1").value;

// den kör sync

const input2 = document.querySelector("#id2").value;

// den byggs upp men kör inte tills man anropar den

function callback(){

console.log(' Show me value ', input1 )

}

//den kör async

document.querySelector("button").addEventListener("click", callback)

// den kör sync

const mainDiv =document.querySelector(".main")

//den kör sync

console.log(mainDiv)

Hur kommer ihåg js click events??

Hur kan vi konvertera den till “non-blocking”

function doSum() {let summa =0;

for(let i= 0; i<2000000000; i++){

summa += 1;

}

console.log(summa)

}

setTimeout(doSum, 3000)

//den kör sync

const input1 = document.querySelector("#id1").value;

// den kör sync

const input2 = document.querySelector("#id2").value;

// den byggs upp men kör inte tills man anropar den

function callback(){

console.log(' Show me value ', input1 )

}

//den kör async

document.querySelector("button").addEventListener("click", callback)

// den kör sync

const mainDiv =document.querySelector(".main")

//den kör sync

console.log(mainDiv)

Varför setTimeout kunde göra den “non-blocking”?

Function Stack :

function one(){

console.log("hej hej ")

}

function two(){

one()

}

function three(){

two()

}

three()

Vad kommer att köras först?

Event Loops:

Öppna länken :

[Testa event loops , callstack i js](http://latentflip.com/loupe/?code=JC5vbignYnV0dG9uJywgJ2NsaWNrJywgZnVuY3Rpb24gb25DbGljaygpIHsKICAgIHNldFRpbWVvdXQoZnVuY3Rpb24gdGltZXIoKSB7CiAgICAgICAgY29uc29sZS5sb2coJ1lvdSBjbGlja2VkIHRoZSBidXR0b24hJyk7ICAgIAogICAgfSwgMjAwMCk7Cn0pOwoKY29uc29sZS5sb2coIkhpISIpOwoKc2V0VGltZW91dChmdW5jdGlvbiB0aW1lb3V0KCkgewogICAgY29uc29sZS5sb2coIkNsaWNrIHRoZSBidXR0b24hIik7Cn0sIDUwMDApOwoKY29uc29sZS5sb2coIldlbGNvbWUgdG8gbG91cGUuIik7!!!PGJ1dHRvbj5DbGljayBtZSE8L2J1dHRvbj4%3D)

Läs mer :

Event loop: <https://www.youtube.com/watch?v=cCOL7MC4Pl0>

Async and await ex:

--------------------------------------------------------------------------------------------------------------------------

Initial kodbas:

const data = [

{id:1, json:"data1"},

{id:2 , json:"data2"}

]

function fetchData(){

setTimeout(()=>{

let res= "";

data.forEach((e)=>{

res +=e.json + " ";

} )

console.log(res)

}

, 2000)

}

fetchData()

Underlektionskod:

const data = [

{id:1, json:"data1"},

{id:2 , json:"data2"}

]

function fetchData(){

let resultat= "Data 0";

setTimeout(()=>{

data.forEach((e)=>{

resultat +=e.json + " ";

console.log("In i loopen" , resultat)

} )

console.log("efter loppen" ,resultat)

}

, 2000)

console.log("efter timeout funktionen" ,resultat)

}

fetchData();

Make a callback :

const data = [

{id:1, json:"data1"},

{id:2 , json:"data2"}

]

function fetchData(){

setTimeout(()=>{

let res= "";

data.forEach((e)=>{

res +=e.json + " ";

} )

console.log(res)

}

, 2000)

}

function postData(newData, callback){

setTimeout(()=>{

data.push(newData);

callback()

}, 2000)

}

//fetchData()

const skapaData = {id:3, json:"data3"}

postData( skapaData , fetchData);

Make it Promise :

const data = [

{id:1, json:"data1"},

{id:2 , json:"data2"}

]

function fetchData(){

setTimeout(()=>{

let res= "";

data.forEach((e)=>{

res +=e.json + " ";

} )

console.log(res)

}

, 2000)

}

function postData(newData){

return new Promise((resolve, reject)=>{

setTimeout(()=>{

data.push(newData);

//after pushing data promise resolve the data

const error=false;

!error? resolve():reject()

}, 2000)

})

}

//fetchData()

const skapaData = {id:3, json:"data3"}

postData(skapaData)

use and catch the promise:

Samma kod som ovan plus:

//skapa data sedan anropa fetchData function. om det finns error fånga den

postData(skapaData).then(fetchData).catch((err)=>{console.log(err)})

Promise ex:

const data = [];

function postData(newData) {

//data.push(newData);

return new Promise((resolve, reject) => {

if (data.length == 1) {

resolve("Jag fick data av new Data");

}

reject("Jag fick inte nån data");

})

}

postData()

.then((message) => console.log(message))

.catch((errMsg) => console.log(errMsg))

true/false:

dataArr.push(" ")

1

!dataArr.push(" ")

false

!!dataArr.push(" ")

true

dataArr.push(" ")

4

!dataArr.push(" ")

false

!!dataArr.push(" ")

true

Samma kod ex: till async await:

const data = [

{id:1, json:"data1"},

{id:2 , json:"data2"}

]

function fetchData(){

setTimeout(()=>{

let res= "";

data.forEach((e)=>{

res +=e.json + " ";

} )

console.log(res)

}

, 2000)

}

async function postData(newData){

await setTimeout(()=> data.push(newData)

, 2000)

}

//fetchData()

const skapaData = {id:3, json:"data3"}

postData(skapaData).then(fetchData).catch((err)=>{console.log(err)})

Extra:

Promise.all(): för att undvika promise helvete

const data = [

{id:1, json:"data1"},

{id:2 , json:"data2"}

]

function fetchData(){

setTimeout(()=>{

let res= "";

data.forEach((e)=>{

res +=e.json + " ";

} )

console.log(res)

}

, 2000)

}

async function postData(newData){

await setTimeout(()=> data.push(newData)

, 2000)

}

//fetchData()

const skapaData = {id:3, json:"data3"}

postData(skapaData).then(fetchData).catch((err)=>{console.log(err)})

const promise1= Promise.resolve("hello world");

const someData = 10;

const promise3 = new Promise((resolve,reject)=>{

setTimeout(resolve, 3000, "some data")

})

Promise.all([promise1,promise3,someData]).then((promisedData)=>{

console.log(promisedData)

})

Fake jsondata : <https://jsonplaceholder.typicode.com/>

\*\* Lägg en till promise

const promise4 =

fetch('https://jsonplaceholder.typicode.com/todos/1')

.then(response => response.json())

Promise.all([promise1,promise3,someData, promise4]).then((promisedData)=>{

console.log(promisedData)

})

en till ex:

function fetchDb(){

console.log("some important data")

}

/\* function fetchDb2(){

console.log("using data from db1 to db2 ")

} \*/

async function doAsync(){

await setTimeout(fetchDb, 2000)

console.log("completed");

}

doAsync();

Fånga error:

function fetchDb(){

console.log("some important data")

}

/\* function fetchDb2(){

console.log("using data from db1 to db2 ")

} \*/

async function doAsync(){

try{

await setTimeout(fetchDb, 2000)

console.log("completed");

}

catch(err){

console.log(err)

}

}

doAsync();

Database integration :

<https://expressjs.com/en/guide/database-integration.html>

MongoDB vs vanligt SQL database:



Vilken database ska man välja?

SQL database/RDMS :

|  |
| --- |
| Relational database management system.  Object/Entity har relationer mellan varandra.  one to one  one to many  many to many |

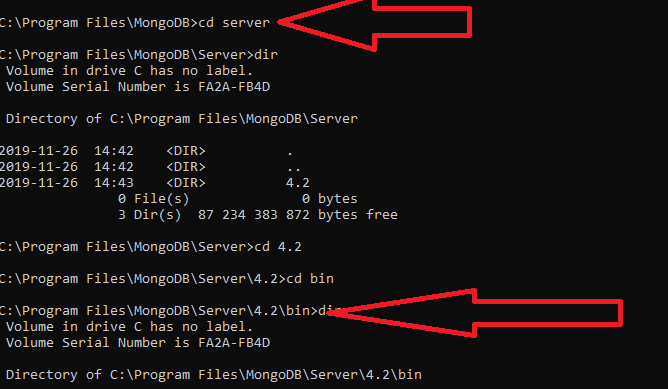
SQL vs NOSQL :

|  |
| --- |
| SQL : Structured Query language  ex. Select \* from product  //välja alla produkter från produkt tabellen  NoSql : Not only SQL.  Enkel att skala applikation.  db.students.find() |

Arbeta med mongodb:

ladda ner och installera mongodb : <https://docs.mongodb.com/manual/administration/install-community/>

Spara mongo db i c mappen (inte måste).



Gå in till din mongodb mappen tills du når till bin mappen och kör :

mongod // för att köra mongodb server

Mongo i terminalen // för att kunna öppna mongo shell

För mac användare : <https://treehouse.github.io/installation-guides/mac/mongo-mac.html>

**mongo** kommandon för att öppna mongo shell och **mongod** kommandon för att öppna kopplingen för klient.

Testa nu i mongo shell :

Show dbs // för att visa vilka databaser du har

Use db // för att skapa / använda nån databas

> show collections // för att visa collections ( Mongo sparar data i collection och RDMS(mysql) sparas data i tabell)

Om du inte har nån collections då visas ingenting i Kommandon.

> db.createCollection("students") //för att skapa collections.

{ "ok" : 1 }

db.students.insert({First\_name: "rakib", lastname: "Ahmed"});// för att lagra data i collections.

> db.students.find() // för att läsa /hämta data från collections.

{ "\_id" : ObjectId("58c94fd205aa924d9acb309b"), "First\_name" : "rakib", "lastname" : "Ahmed" }

// objectID är unik nyckel för alla objekten i collections .

db.students.find().pretty() // för att visa lite fint.

{

"\_id" : ObjectId("58c94fd205aa924d9acb309b"),

"First\_name" : "rakib",

"lastname" : "Ahmed"

}

db.students.insert([{First\_name: "rakib", lastname: "Ahmed"}, {first\_name:"Håkan", last\_name:"Lundberg"}]); // du kan lagra flera data samtidigt med en array [ { } ] obs. javascript array syntax.

db.students.insert([{First\_name: "rakib", lastname: "Ahmed"}, {first\_name:"Håkan", last\_name:"Lundberg"}, {first\_name: ”Sam”, last\_name:”Svenson”, Grade:”E”]); // objekten kan ha olika typer av data .