

# **Backend Development**

## **Lab 2**

<b>1. Laboration goals</b>	3
<b>2. Pre - requirements</b>	3
<b>3. Instructions</b>	3
3.1 Live server and npm install cors	3
3.2 Rest API example server	5
3.3 Rest API example client fetch	6
3.4	7
<b>4. Task</b>	7
<b>5. Requirements</b>	8

# 1. Laboration goals

The goal of this lab is to be able to connect the front end to the backend using a rest api and fetch. This way the front end should be able to send instructions to the backend and handle errors. In this lab you will also learn about cors issues and how to handle json objects. Also you will learn how to use a live server while developing a connection between back end and front end.

## 2. Pre - requirements

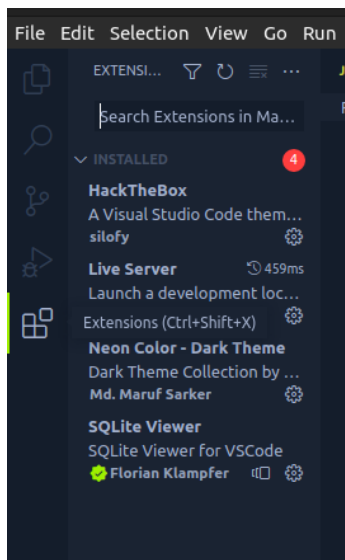
Lab 1

## 3. Instructions

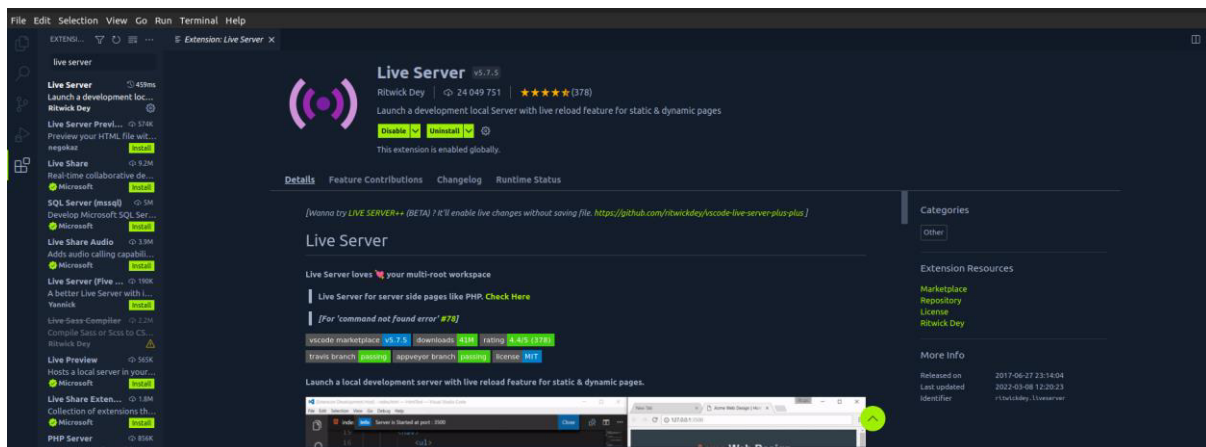
Two new things are needed for this lab. Live server and cors. It is preferable if you use the code from the first lab.

### 3.1 Live server and npm install cors

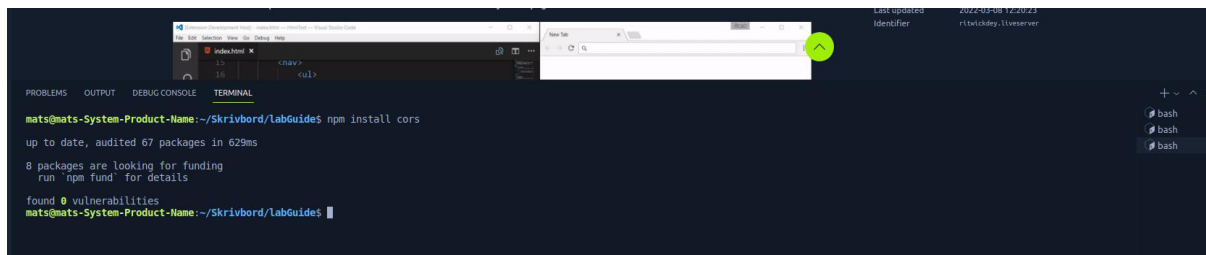
To download live Server go to visual studio code and press extensions.



Search for a live server and download it.



Like the previous lab we need to download a package using npm. Open a terminal in visual studio code and make sure you are on the right path. Then enter npm install cors.



## 3.2 Rest API example server

Create a server.js file. In the file enter the following.

```
// IMPORT ALL NECESSARY LIBRARIES AND TOOLS
const express = require('express')
const app = express()
const cors = require('cors')

app.use(cors({
  origin: "*",
}))

// LISTENING ON PORT
app.listen(5000, () => {
  console.log("Server listening on port: " + 5000);
})

app.get('/dataParam', function (req, res) {
  sendBack = {
    message: "This is lab2",
    name: "Hkr"
  }
  res.json(sendBack)
  res.end();
})
```

Just like the previous lab we are first implementing an express app. This time we are using something called cors as well. Cors is something that is used to prevent xss exploitation. Without cors you will not be able to send requests to the rest api server. Read more at <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

After that we need to tell the server what ip number we are allowing. In the above server code we are saying that we are allowing every ip using the \* symbol.

App listen code should be familiar from the previous lab. Then we have the app.get function. This is a route that we talked about in lab 1. Rest api uses routes to create, gather, delete and update data. Here are some code examples <https://docs.amplify.aws/lib/restapi/getting-started/q/platform/js/>

### 3.3 Rest API example client fetch

```
async function getdata() {
  try {
    var result = fetch('http://localhost:5000/dataParam', {
      method: 'GET',
      headers: { 'content-type': 'application/json' }
    });
    var rest = result.json();
    return(rest)
  } catch (error) {
    console.log(error)
  }
}
```

Here is the method used to get data on the client side. Everything is in a try catch to handle errors. Then there is a fetch call to the url and route of the server. In the fetch method we specify a message, the message contains headers and methods. Then we await the message and return it. Important note is that a post request for instance can contain a body with parameters.

```
// GET THE VALUES FROM DATABASE BASED ON THE LOGGED IN USER AND DATE
get.addEventListener('click', event => {
  let data = getdata()
  const value = Promise.resolve(data);

  value.then(text =>{
    console.log(text)
    document.getElementById("outPut").value += text.name + " " +
text.message
  }).catch(err => {
    console.log(err);
  })
});
```

Here the code is used inside an eventlistener. Since the functions are asynchronous, we make them wait for the data to be collected with a promise. You can learn more about promises here. [https://medium.com/@armando\\_amador/how-to-make-http-requests-using-fetch-api-and-promises-b0ca7370a444](https://medium.com/@armando_amador/how-to-make-http-requests-using-fetch-api-and-promises-b0ca7370a444)

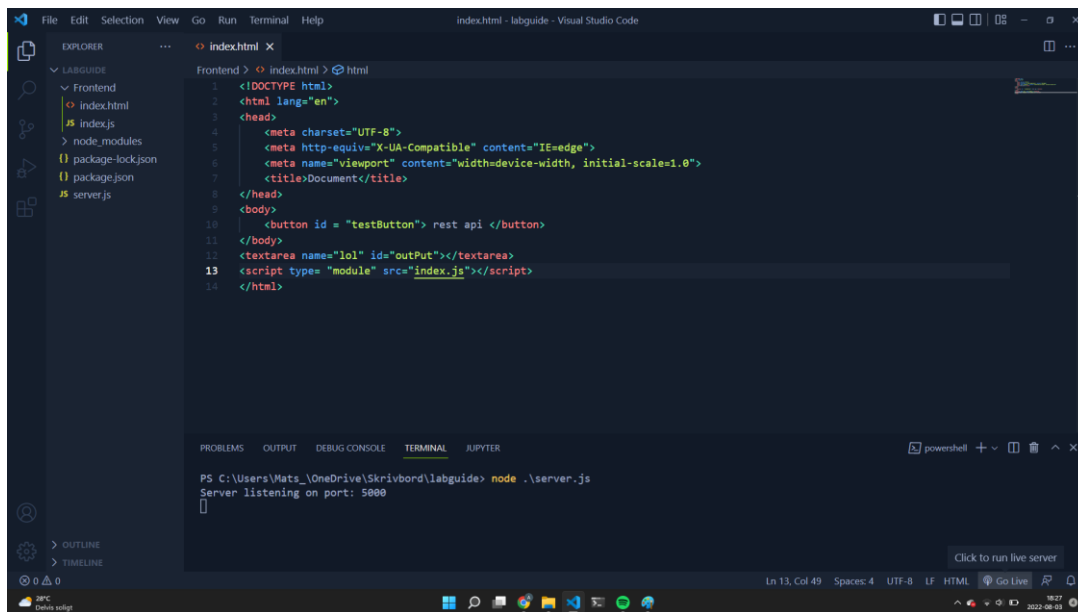
## 3.4 Try the code

Open the labguide folder and open a new terminal. Run the command `node server.js`



```
PS C:\Users\Mats_\OneDrive\Skrivbord\labguide> node .\server.js
Server listening on port: 5000
```

Open the `index.html` and start live server on the bottom right.



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8 </head>
9 <body>
10   <button id = "testButton"> rest api </button>
11 </body>
12 <textarea name="lol" id="outPut"></textarea>
13 <script type= "module" src="index.js"></script>
14 </html>
```

```
PS C:\Users\Mats_\OneDrive\Skrivbord\labguide> node .\server.js
Server listening on port: 5000
```

Now in the webbrowser press the rest api button and see what happens.

## 4. Task

The library has now made a simple html page for adding and getting books. They want you to make the backend functionality. They should be able to get the books, remove books and add a book in the database from lab 1. Everything in the front end has been coded. You only need to fix the fetch for the client side and the rest api on the backend.

## 5. Requirements

You should only code in the client.js and server.js files. The html file should already be connected to the client.js file. So no need to worry about that.

### Route 1

- Get with path “/getData”
- should get data in the same way as lab one.

### Route 2

- Delete with path “deleteData”
- body in request with params.
- should call a function in database.js named deleteData that deletes book

### Route 3

- put with path “addData”
- body in request with params.
- should call a function in database.js named addData that adds book