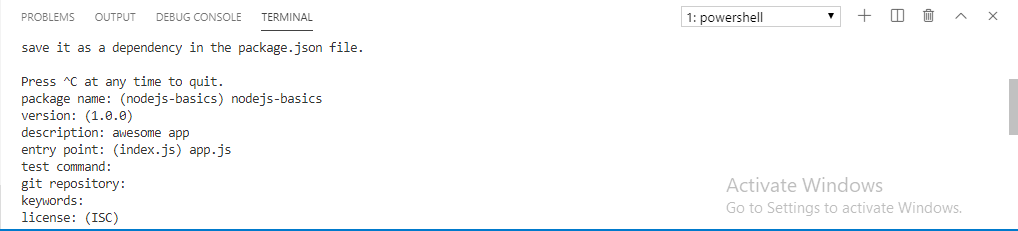
**Node JS CNQ**

**STEP1 –**

Npm init 🡪 fill few steps in command line 🡪 package.json file will be created



Import Export Node JS

1. Single Export

Method 1

Test.js

function sum(a, b) {

  return a + b;

}

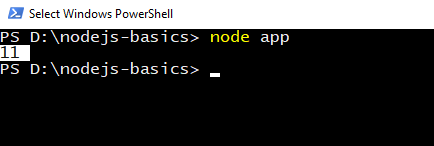
module.exports = sum;

app.js

const getsum = require("./test");

console.log(getsum(5, 6));

o/p



Method 2 (test.js)

module.exports = function sum(a, b) { // writing export at the time of defining

  return a + b;

};

Method 3 (test.js)

module.exports = function(a, b) { // Anonymous function

  return a + b;

};

Method 4 (test.js)

module.exports = (a, b) => {

  return a + b;

};

1. Multiple Export

* Here we write objects of function

Test.js

module.exports = {

  sum: (a, b) => {      // function 1

    return a + b;

  },

  mul: (a, b) => {      // function 2

    return a \* b;

  }

};

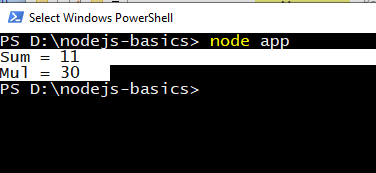
App.js

const getsum = require("./test");

console.log("Sum = " + getsum.sum(5, 6));

console.log("Mul = " + getsum.mul(5, 6));

O/P



* We have seen creating custom modules
* Other than this we have built-in modules (like http) and 3rd party modules
* Till now we were seeing in terminal. To see in we browser we will use http

Creating Server (Core Node Way)

App.js

const http = require("http"); // built in module

const path = require("path"); // built in module

http.createServer((req, res) => { // create server

    res.write("Hello World");

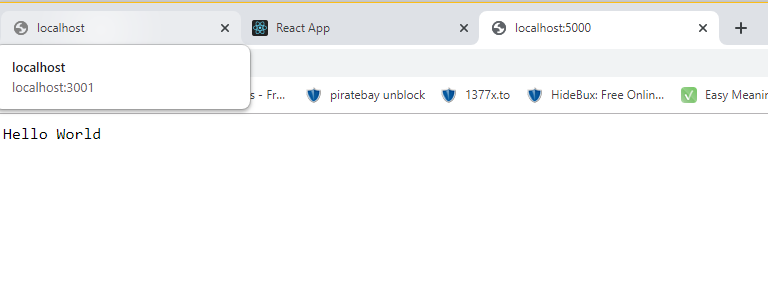
    res.end();

}).listen(5000);

console.log("Server is running in port 5000");

Terminal - node app

O/P:



Creating Server (Express Way)

Step 1 – Install express – npm install express

App.js

const express = require("express");

const app = express();

const port = 5000;

app.get("/home", (req, res) => {

  res.send("Hello World");

});

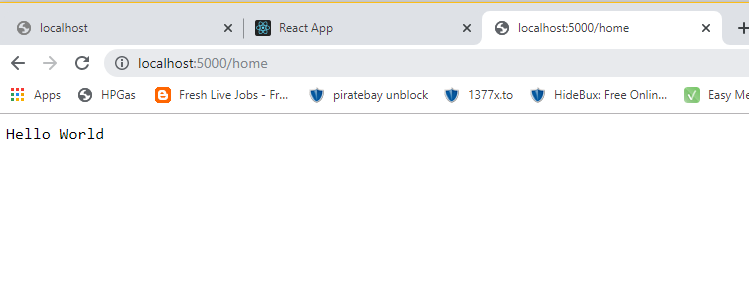
app.listen(port, () => {

  console.log("Server is running in port" + port);

});

Terminal – node app.js

O/P



Remove CORS

* Whenever client wants to interact with server this issue comes.
* Base domain of client (say React JS or plain html) is not same as server (node / express server)
* CORS is applied on server end (i.e. domain client is accessing)

Soln

In node install cors package – npm install cors

App.js

const express = require("express");

const cors = require("cors");

const app = express();

app.use(cors()); // Use this after the variable declaration

Client page want to interact with server

* Here we will see client-server communication
* Here say in app.js a route is created named home which returns object

App.js

const express = require("express");

const cors = require("cors");

const app = express();

app.use(cors()); // Use this after the variable declaration

const port = 5000;

app.get("/", (req, res) => {

  res.send("Hello World");

});

app.get("/home", (req, res) => {

  res.send({ name: "Amir Mustafa", age: 27, job: "Software engineer" });

});

app.get("/about", (req, res) => {

  res.send("this is about page");

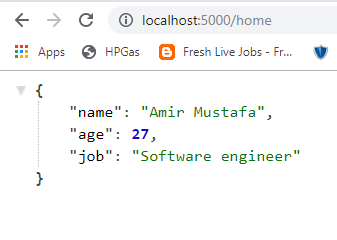
});

app.listen(port, () => {

  console.log("Server is running in port" + port);

});

Hitting route in browser



Index.html (this page is anywhere in your computer

* A button is created which fetches data from home route

<html>

  <head>

    <title>Client</title>

  </head>

  <body>

    <h1>Fetch API</h1>

    <button id="mybtn">fetch it</button>

  </body>

</html>

<script>

  document.getElementById("mybtn").onclick = () => {

    const url = "http://localhost:5000/home"; // server url (i.e. node route)

    fetch(url) // fetching like regular API

      .then(res => res.json())

      .then(res => {

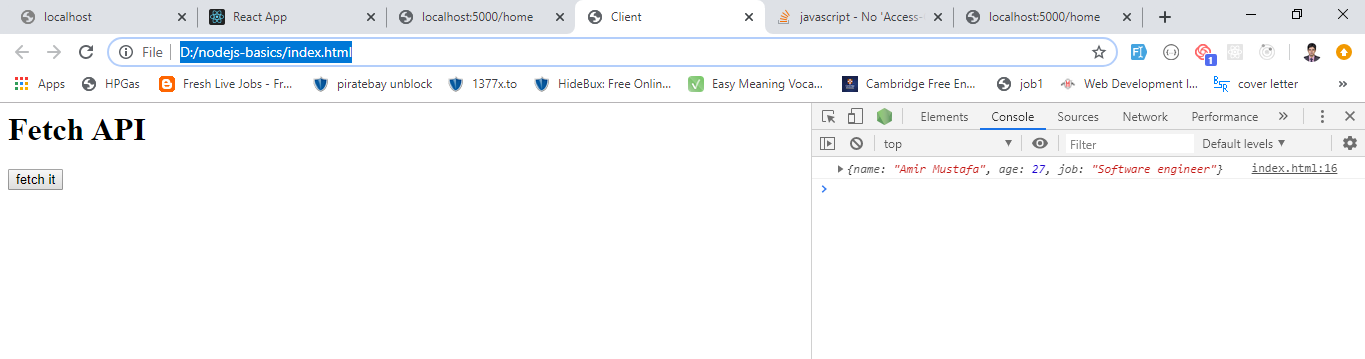
        console.log(res);

      });

  };

</script>

O/P



Send file instead of text

App.js

const express = require("express");

const path = require("path");

const cors = require("cors");

const app = express();

app.use(cors()); // Use this after the variable declaration

const port = 5000;

app.get("/", (req, res) => {

  res.sendFile(path.join(\_\_dirname + "/index.html")); //node base path till \_\_dirname

// res.sendFile(\_\_dirname + "/index.html"); // without path module

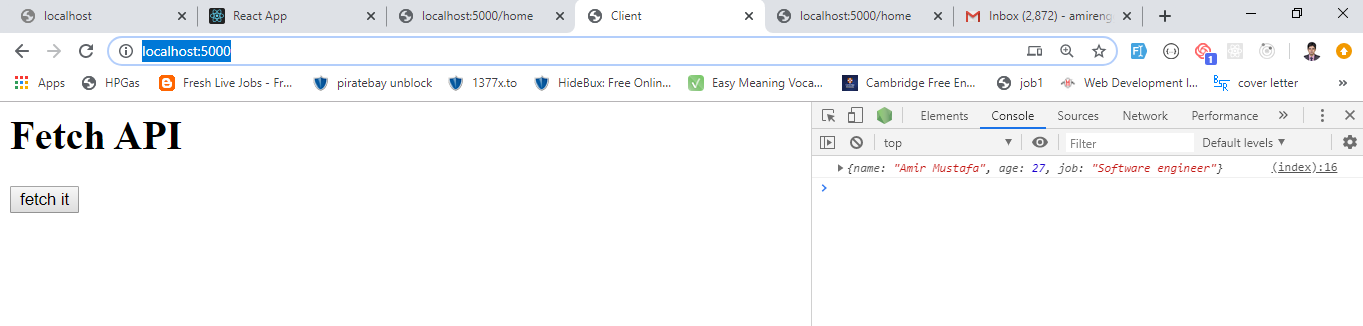
});

app.listen(port, () => {

  console.log("Server is running in port" + port);

});

O/P – Everything is same but we have opened from localhost (i.e. server)



\_\_

1. Suppose we write internal CSS in index.html

Index.html

<html>

  <head>

    <title>Client</title>

    <style>

        body {

            background: red;

        }

    </style>

  </head>

  <body>

    <h1>Fetch API</h1>

    <button id="mybtn">fetch it</button>

  </body>

</html>

<script>

  document.getElementById("mybtn").onclick = () => {

    const url = "http://localhost:5000/home";

    fetch(url)

      .then(res => res.json())

      .then(res => {

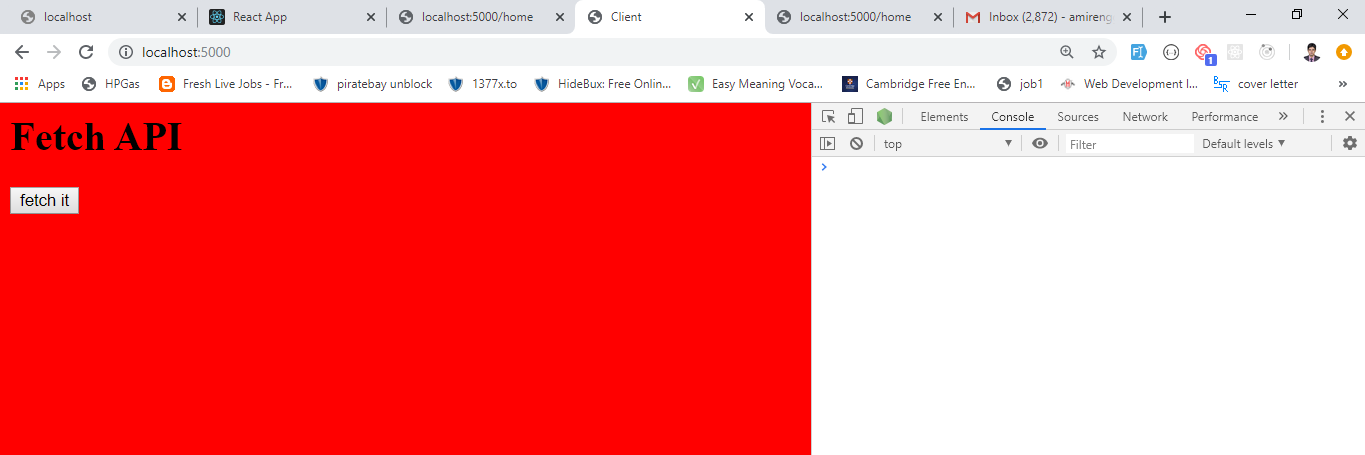
        console.log(res);

      });

  };

</script>

O/P:



1. Now suppose we create the external style.css file and link to file we get error

Style.css

body {

  background: red;

}

Index.html

<html>

  <head>

    <title>Client</title>

    <link rel="stylesheet" href="style.css" />

  </head>

  <body>

    <h1>Fetch API</h1>

    <button id="mybtn">fetch it</button>

  </body>

</html>

<script>

  document.getElementById("mybtn").onclick = () => {

    const url = "http://localhost:5000/home";

    fetch(url)

      .then(res => res.json())

      .then(res => {

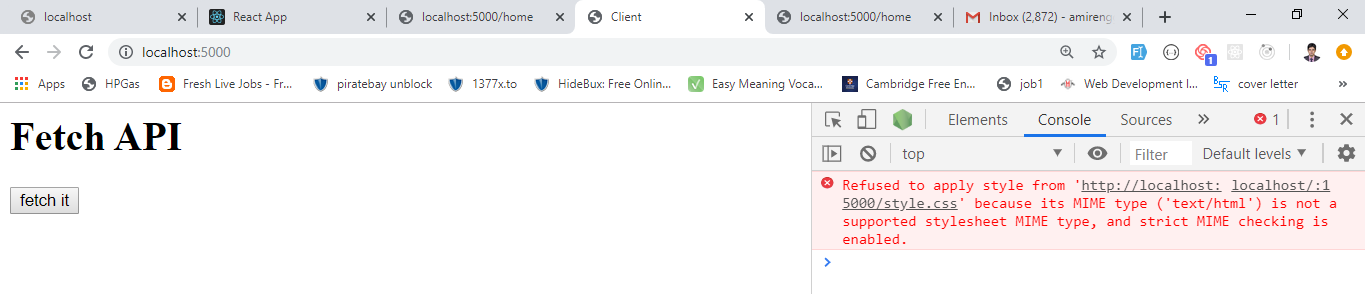
        console.log(res);

      });

  };

</script>

O/P:



* So when we are trying to link external file be it css or js server is rejecting it. Actually we have to tell Node JS to explicitly serve these files.
* These files are called Static files

**Way Server (i.e. Node ) understand static file**

const express = require("express");

const path = require("path");

const cors = require("cors");

const app = express();

app.use(cors()); // Use this after the variable declaration

const port = 5000;

app.get("/", (req, res) => {

  res.sendFile(\_\_dirname + "/index.html");

});

app.use(express.static("public")); // whatever is written inside public is accepted by node i.e. style.css or some.js which have fetch method

index.html

<html>

  <head>

    <title>Client</title>

    <link rel="stylesheet" href="/style.css" /> // file inside public

  </head>

  <body>

    <h1>Fetch API</h1>

    <button id="mybtn">fetch it</button>

  </body>

</html>

<script>

  document.getElementById("mybtn").onclick = () => {

    const url = "http://localhost:5000/home";

    fetch(url)

      .then(res => res.json())

      .then(res => {

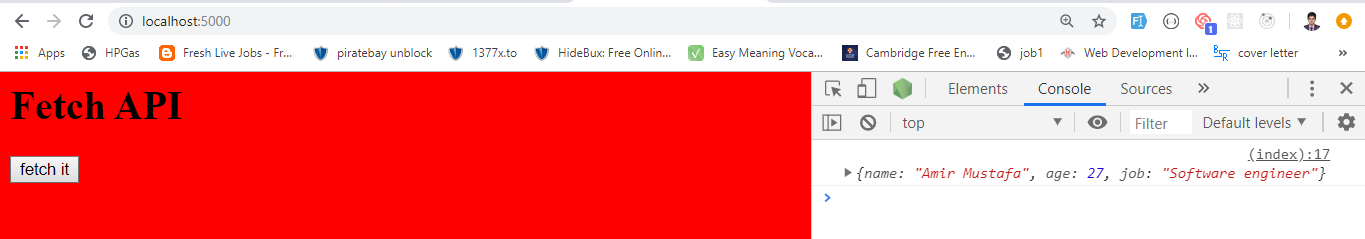
        console.log(res);

      });

  };

</script>

O/P – Css works now (express server accepts style.css inside static folder)



Organizing routes

Routes.js

module.exports = app => { // this will receive app as an argument

  app.get("/", (req, res) => {

    res.sendFile(\_\_dirname + "/index.html");

  });

  app.get("/about", (req, res) => {

    res.send("this is about page");

  });

  app.get("/home", (req, res) => {

    res.send({ name: "Amir Mustafa", age: 27, job: "Software engineer" });

  });

};

App.js

const express = require("express");

const path = require("path");

const cors = require("cors");

const app = express();

app.use(cors()); // Use this after the variable declaration

const port = 5000;

// serving static files

app.use(express.static("public"));

// import routes

require("./routes")(app); // passing app as argument

app.listen(port, () => {

  console.log("Server is running in port" + port);

});

Nodemon

* Install this package – npm install nodemon
* Now instead of node app run nodemon app. This will auto run without killing server every time

EJS

* This is the view engine provided by node. – npm install ejs
* For using view two steps needs to be done

Step1 – Create view/home.ejs

Step2:

App.js

const express = require("express");

const path = require("path");

const cors = require("cors");

const app = express();

app.use(cors()); // Use this after the variable declaration

const port = 5000;

// serving static files

app.use(express.static("public"));

// this line must be written to set default view type - ejs , inside route file used - res.render()

app.set("view engine", "ejs"); // written before routes import

// import routes

require("./routes")(app);

app.listen(port, () => {

  console.log("Server is running in port" + port);

});

Routes.js

module.exports = app => {

  app.get("/", (req, res) => {

    res.render("home");

  });

  app.get("/about", (req, res) => {

    res.send("this is about page");

  });

  app.get("/home", (req, res) => {

    res.send({ name: "Amir Mustafa", age: 27, job: "Software engineer" });

  });

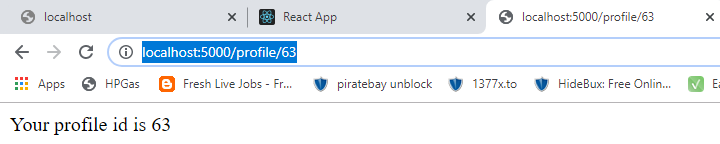
};

Way to pass dynamic id in routes:

app.get("/profile/:id", (req, res) => {

    res.send(`Your profile id is ${req.params.id}`);

});



Node JS Dynamic Webpage (EJS)

* Node JS is itself very powerful that it may not require any client end framework/library like React JS or Angular or Vue JS or plain index.html.
* Itself have dynamc templating engine (i.e. view Engine like EJS)
* There are two famous view engine – EJS and handlebar.

EJS

* We have already seen to render from route to view (EJS) we use res.render
* Now to pass data, we will pass in object format

We will pass data from route to EJS

Routes.js

module.exports = app => {

  const data = {

    // data to pass in EJS View

    name: "Aamir",

    status: "programmer"

  };

  app.get("/", (req, res) => {

    res.render("home", { data: data });

  });

  app.get("/about", (req, res) => {

    res.send("this is about page");

  });

  app.get("/profile/:id", (req, res) => {

    res.send(`Your profile id is ${req.params.id}`);

  });

};

Views/home.ejs

<html>

  <head>

    <title>Client</title>

    <link rel="stylesheet" href="/style.css" />

  </head>

  <body>

    <h1>This is my home page.</h1>

    <h2>

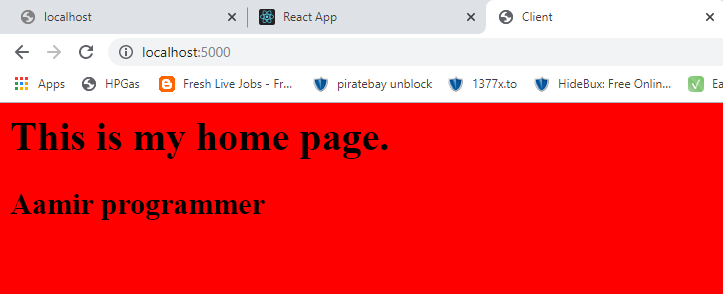
      <%= data.name%> <%= data.status%>

    </h2>

  </body>

</html>

O/P



For more EJS syntax see official site - <https://ejs.co/>

Partails

* Partials are the pieces of code like navigation etc. It’s just like component of React but in node style

STEP1 – inside views, create folder partials

We will use materialize for navbar

<https://materializecss.com/navbar.html>

Views/partials/nav.ejs – here we will write navbar snippet

<nav>

  <div class="nav-wrapper">

    <a href="#" class="brand-logo">Logo</a>

    <ul id="nav-mobile" class="right hide-on-med-and-down">

      <li><a href="sass.html">Sass</a></li>

      <li><a href="badges.html">Components</a></li>

      <li><a href="collapsible.html">JavaScript</a></li>

    </ul>

  </div>

</nav>

Views/home.ejs

<html>

  <head>

    <title>Client</title>

    <link rel="stylesheet" href="/style.css" />

    <link

      rel="stylesheet"

      href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css"

    />

  </head>

  <body>

    <%- include('partials/nav'); %> <!-- included navbar partials here -- >

    <h1>This is my home page.</h1>

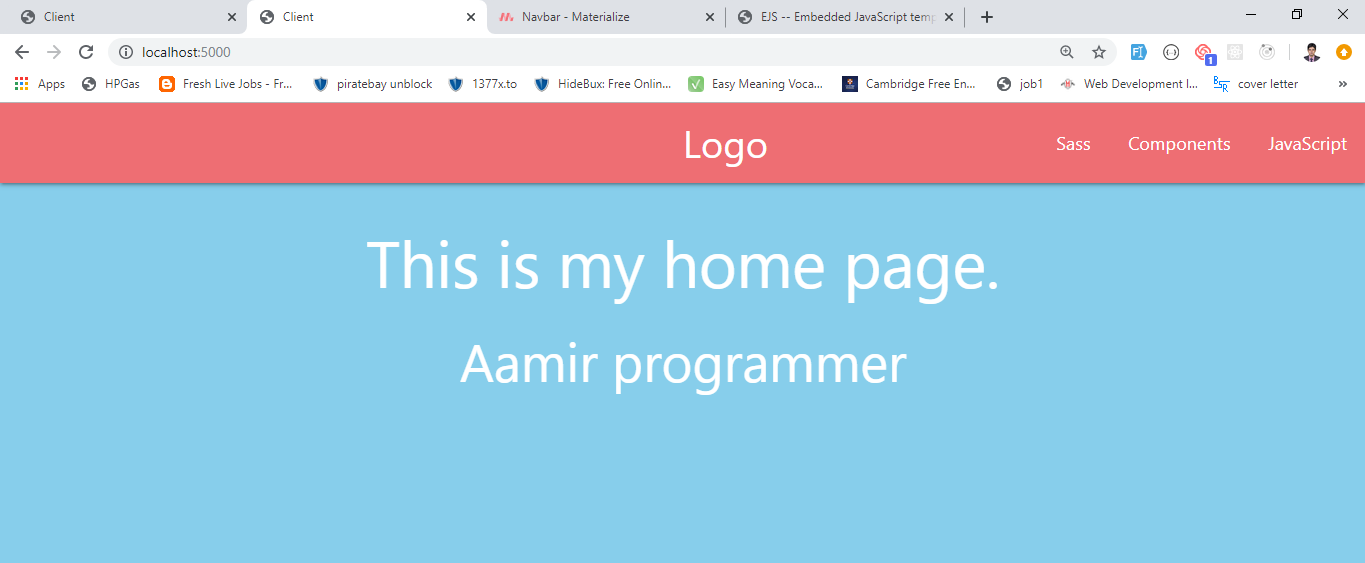
    <h2>

      <%= data.name%> <%= data.status%>

    </h2>

  </body>

</html>



If render dynamic data to ejs page

app.get("/profile/:id", (req, res) => {

    data = {

      name: req.params.id

    };

    res.render("home", { data: data });

    // res.send(`Your profile id is ${req.params.id}`);

  });

Sending Client data to Server using fetch POST methods (This we will implement in next topic)

1. **Without form**

Index.html (Client)

<html>

  <head>

    <title>Client</title>

  </head>

  <body>

    <h1>Fetch API</h1>

<button type="button" id="mybtn">Submit without form</button>

  </body>

</html>

<script>

  // data

  var data = {

    name: "Aamir",

    status: "Programmer"

  };

  document.getElementById("mybtn").onclick = () => {

    const url = "http://localhost:5000/sent-data"; // Node/express server url

    // using POST in fetch request

    fetch(url, { // second parameter is used when we want to use POST req

      method: "post",

      body: JSON.stringify(data), // sending data

      headers: {

        "Content-Type": "application/json"

      }

    })

      .then(res => res.json())

      .then(res => {

        console.log(res);

      });

  };

</script>

App.js (node server)

const express = require("express");

const path = require("path");

const cors = require("cors");

const bodyParser = require("body-parser");

const app = express();

app.use(cors()); // Use this after the variable declaration

app.use(bodyParser.json());

app.use(

  bodyParser.urlencoded({

    extended: false

  })

);

const port = 5000;

// serving static files

app.use(express.static("public"));

// this line must be written to set default view type - ejs , inside route file used - res.render()

app.set("view engine", "ejs"); // written before routes import

// import routes

require("./routes")(app);

app.listen(port, () => {

  console.log("Server is running in port" + port);

});

Routes.js

module.exports = app => {

  let data = ["code", "sleep", "workout", "eat"];

  app.get("/", (req, res) => {

    res.render("home", { wish: data });

  });

  app.get("/about", (req, res) => {

    res.render("about");

    // res.send("this is about page");

  });

  app.post("/sent-data", (req, res) => {

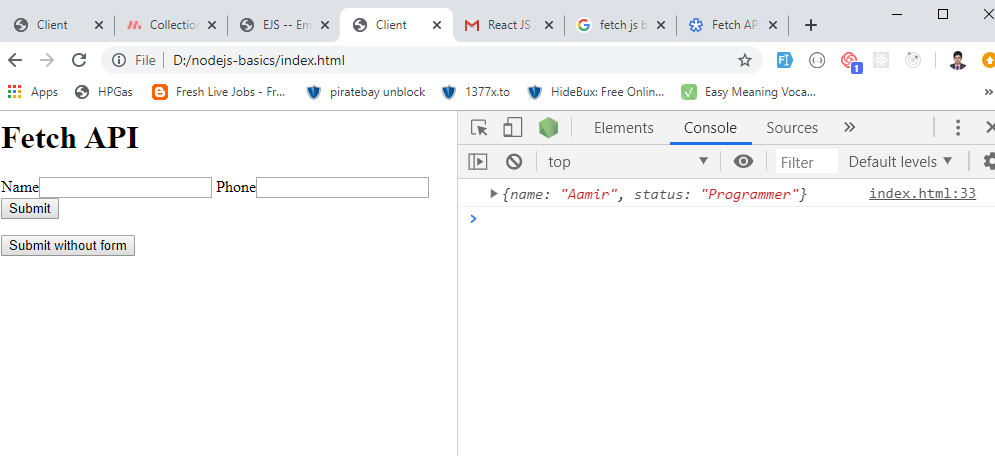
    console.log(req.body);

    res.send(JSON.stringify(req.body)); // to receive response from body(in client side we were sending JSON.stringify(json – receives here))

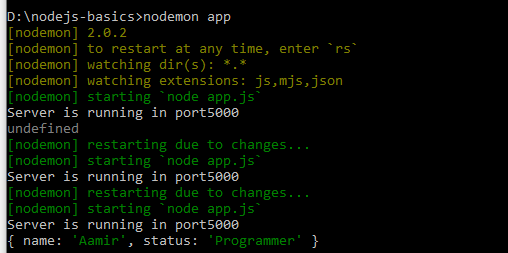
  });

};

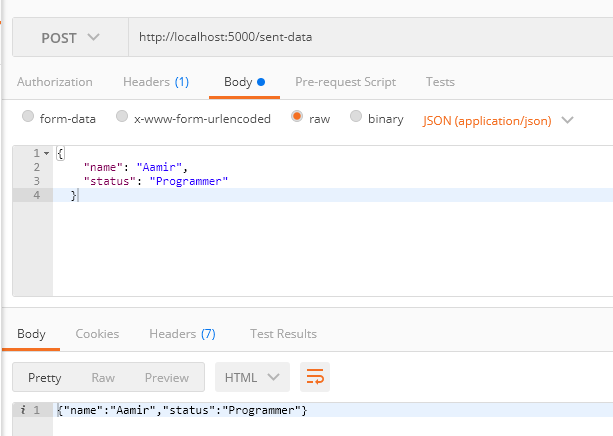
O/P



This response is coming from the server i.e. response of the fetch



Postman



1. With Form

Index.html (client page)

<html>

  <head>

    <title>Client</title>

  </head>

  <body>

    <h1>Fetch API</h1>

    <form id="myForm">

      <label>Name</label><input type="text" name="name" /> <label>Phone</label

      ><input type="text" name="phone" />

      <button type="submit" id="submit">Submit</button>

    </form>

  </body>

</html>

<script>

  document.getElementById("myForm").onsubmit = e => {

    e.preventDefault(); // prevent default refresh

    const url = "http://localhost:5000/sent-data"; // node server URL

    let usp = new URLSearchParams();                // initialized empty

    for (const pair of new FormData(e.target)) {    // e.target contains form itself

      // console.log(pair);     // this gives array of elements - i.e. one key val pair for name and another for phone

      usp.append(pair[0], pair[1]);                 // pair[0] = element key i.e. name or phone, pair[1] = element val i.e. val given in i/p box

    }

    fetch(url, {

      method: "post",

      body: usp                                     // data (don't stringify in form)

    })

      .then(res => res.json())

      .then(res => {

        console.log(res);

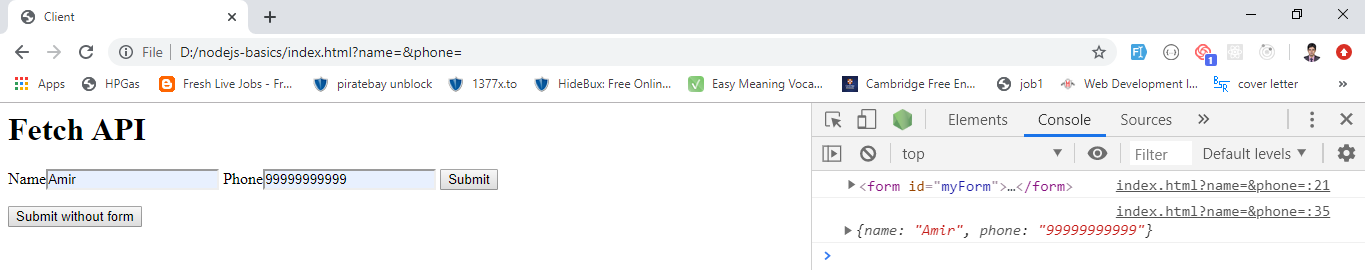
      });

  };

</script>

App.js (node server) – same as above

O/P – So data from the form goes from form i/p to the node server and data is seen as response



TRICK: Before handling post request in Node/Express, you must install body-parser package

npm install body-parser

app.post("/sent-data", (req, res) => {

    console.log(req.body);

    res.send(JSON.stringify(req.body));

});

CRUD OPERATION (Without Database)

* From here we will see CRUD operation We will create some data in route and pass to view page(i.e. home.ejs)

1. Read Operation

Routes.js

module.exports = app => {

  let data = ["code", "sleep"];     // data to send

  app.get("/", (req, res) => {      // Read Operation

    res.render("home", { wish: data }); // passing data to home.ejs view page

  });

  app.get("/about", (req, res) => {

    res.render("about");

    // res.send("this is about page");

  });

};

Home.ejs

<html>

  <head>

    <title>Client</title>

    <link rel="stylesheet" href="/style.css" />

    <link

      rel="stylesheet"

      href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css"

    />

  </head>

  <body>

    <%- include('partials/nav'); %>

    <h2>This is my home page.</h2>

<div class="collection">

        <ul> // Reading Sent Data from routes page

            <% wish.forEach((item, index) => { %>

                <li id="<%= index %>>"><%= item %></li>

            <% }) %>

        </ul>

    </div>

    </h3>

  </body>

</html>

1. Create:

* For create operation we will write a form and send data by post method of fetch
* We have discussed sending data by post method one topic above

Step1 – Create a form and use script to send data to node server(routes.js)

Home.ejs

<html>

  <head>

    <title>Client</title>

    <link rel="stylesheet" href="/style.css" />

    <link

      rel="stylesheet"

      href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css"

    />

  </head>

  <body>

    <%- include('partials/nav'); %>

    <h2>This is my home page.</h2>

    <form id="myForm">

            <input type="text" name="item" />

            <button type="submit" class="waves-effect waves-light btn">Add</button>

    </form>

    <div class="collection">

        <ul>

            <% wish.forEach((item, index) => { %>

                <li id="<%= index %>>"><%= item %></li>

            <% }) %>

        </ul>

    </div>

    </h3>

  </body>

</html>

<script>

    document.getElementById('myForm').onsubmit = (e) => {

        e.preventDefault();

        url = 'http://localhost:5000/sent-data'; // Node Server Post route URL

        usp = new URLSearchParams();

        for(const path of new FormData(e.target)) {

            usp.append(path[0], path[1]);

        }

        fetch(url, {

            method: 'post',

            body: usp

        })

        .then(res => res.json())

        .then(res => {

            console.log(res); // Here we get response from post route

            location.reload();

        })

    }

</script>

Routes.js

module.exports = app => {

  let data = ["code", "sleep"];     // This data we will read in home.ejs

  app.get("/", (req, res) => {      // Read Operation

    res.render("home", { wish: data }); // passing data to home.ejs view page

  });

  app.get("/about", (req, res) => {

    res.render("about");

    // res.send("this is about page");

  });

  app.post("/sent-data", (req, res) => {

    console.log(req.body.item);

    data.push(req.body.item); // pushing data to the new array

    res.send(JSON.stringify(req.body.item)); // writing this is must to receive a response in client's end

  });

};

1. Delete

* For this delete method in fetch (Client side) will be used and app.delete (node server side)in route is used

Home.ejs

<html>

  <head>

    <title>Client</title>

    <link rel="stylesheet" href="/style.css" />

    <link

      rel="stylesheet"

      href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css"

    />

  </head>

  <body>

    <%- include('partials/nav'); %>

    <h2>This is my home page.</h2>

    <form id="myForm">

            <input type="text" name="item" />

            <button type="submit">Add</button>

    </form>

    <div class="collection">

        <ul>

            <% wish.forEach(item => { %>

                <li onclick="deleteMe(this)"><%= item %></li>

            <% }) %>

        </ul>

    </div>

    </h3>

  </body>

</html>

<script>

    function deleteMe(item) {

        // console.log(item.innerText) // get the text

        const target = "http://localhost:5000/remove/"+item.innerHTML;

        console.log(target);

        fetch("http://localhost:5000/remove/"+item.innerHTML, {

            method: 'delete'

        })

        .then(res => res.json())

        .then(res => {

            console.log(res);

            location.reload();

        });

    }

    document.getElementById('myForm').onsubmit = (e) => {

        e.preventDefault();

        url = 'http://localhost:5000/sent-data';

        usp = new URLSearchParams();

        for(const path of new FormData(e.target)) {

            usp.append(path[0], path[1]);

        }

        fetch(url, {

            method: 'post',

            body: usp

        })

        .then(res => res.json())

        .then(res => {

            console.log(res);

            location.reload();

        })

    }

</script>

Routes.js (node)

module.exports = app => {

  let data = ["code", "sleep"]; // This data we will read in home.ejs

  // get routes - Read Operation

  app.get("/", (req, res) => {

    res.render("home", { wish: data }); // passing data to home.ejs view page

  });

  app.get("/about", (req, res) => {

    res.render("about");

    // res.send("this is about page");

  });

  // post routes - Create Operation

  app.post("/sent-data", (req, res) => {

    console.log(req.body.item);

    data.push(req.body.item); // pushing data to the new array

    res.send(JSON.stringify(req.body.item)); // writing this is must to receive a response in client's end

  });

  // delete routes  - Delete Operation

  app.delete("/remove/:id", (req, res) => {

    // we are reassigning array - to remove currently selected data

    data = data.map(item => {

      if (item !== req.params.id) {

        return item;

      }

    });

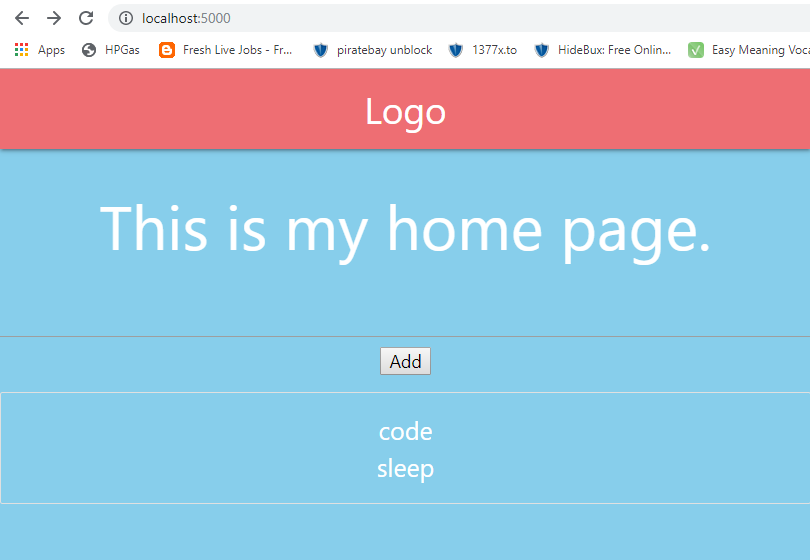
    console.log(data);

    res.send(data);

  });

};

Video - <https://www.loom.com/share/238dbf2d629e410ea9cbe9457448486c>



* Now we create script.js inside public folder and push all js codes there
* In home.ejs

<script src="/script.js"></script>

Mongo DB Atlas

Terminology

**Cluster** = Workspace (like AWS, Azure or local where db will setup)

**Database**

**Collection** = table

**Documents** = rows (i.e. number of entries)

* STEP1 – Create the cluster and get the mongo url

Email – [amirengg15@gmail.com](mailto:amirengg15@gmail.com)

Pass – 1101297141@amir

Steps to get Atlas Url - Video

Creating Cluster - <https://www.loom.com/share/adee9881e4874f48a472b0ab57747db5>

Other steps - <https://www.loom.com/share/a8642a8ac6bd40399352ff8c041560f8>

* STEP2 - In project create a config folder and keys.js. Paste your key there

Config/keys.js

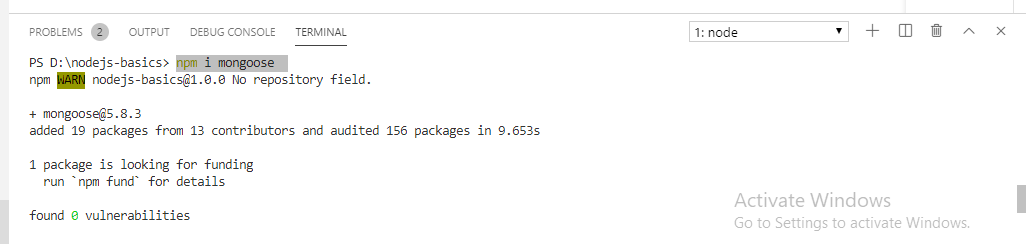
module.exports = {

    const mongodb = "mongodb+srv://amirengg15:123456@cluster0-jnxve.mongodb.net/test?retryWrites=true&w=majority";

}

* STEP3 – npm install mongoose

This package will make easier to communicate with mongodb database



* STEP4 – Now we will create collection/model (or table file). There can be more than one collection. Create a model folder

model/wish.js –

* This is basically a collection file
* A project can have more than one collection

const mongoose = require("mongoose");

const Schema = mongoose.Schema;

const WishSchema = Schema({// schema is the blue print of collection or table

  wish: String

});

// wishes is collection/table name in mongo db

module.exports = mongoose.model("wishes", WishSchema);

Routes.js

const express = require("express");

const app = express();

let data = ["code", "sleep"]; // This data we will read in home.ejs

const mongoose = require("mongoose");

const { mongourl } = require("./config/keys");

const Wish = require("./model/wish");

mongoose.connect(mongourl);  // connect to mongoDB

//mongoose.connect(mongourl, { useNewUrlParser: true, useUnifiedTopology: true }); // write this if shows some deprecated error

module.exports = app => {

  // get routes - Read Operation

  app.get("/", (req, res) => {

    res.render("home", { wish: data }); // passing data to home.ejs view page

  });

// post routes - Create Operation

  app.post("/sent-data", (req, res) => {

    // Save in Mongo DB: Start

    const Item = new Wish({ // Wish is model name imported above

      wish: req.body.item

    });

    Item.save().then(data => {

      console.log(`${req.body.item} saved in mongo db`);

res.send({ post: "data inserted" }); // response just for reloading

    });

    // Earlier we were pushing into data array

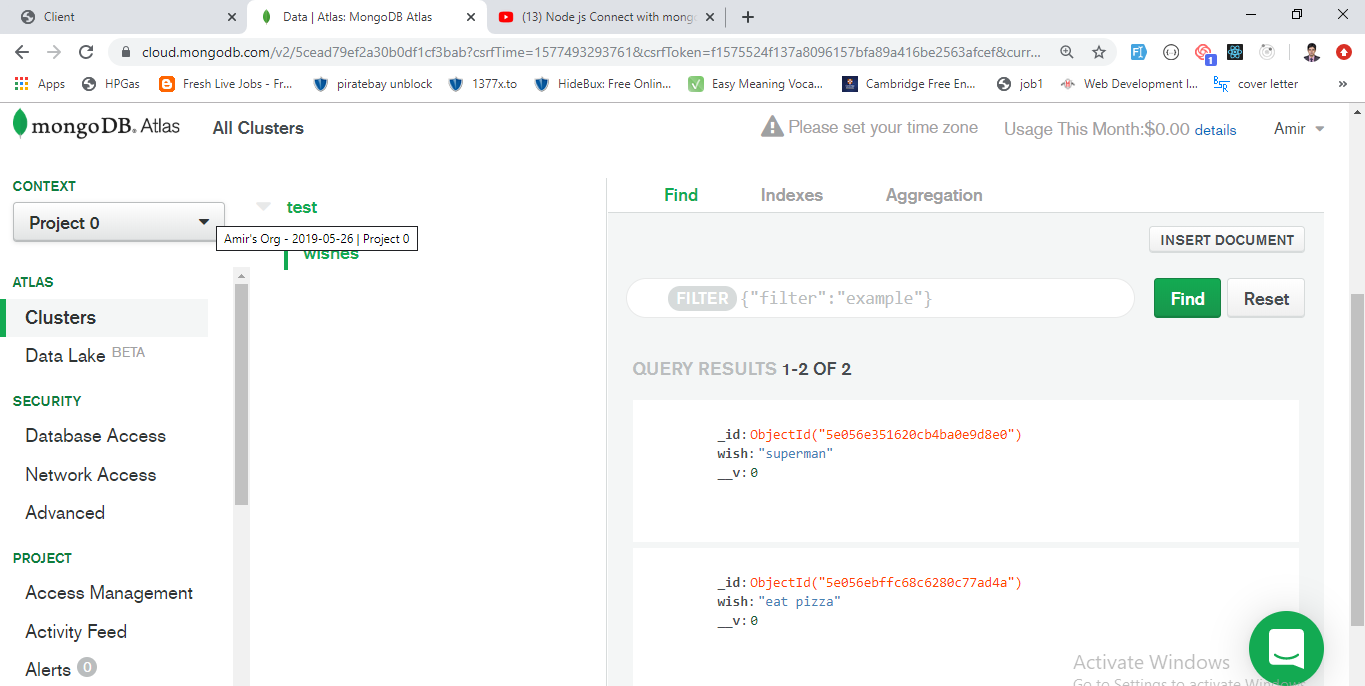
    // console.log(req.body.item);

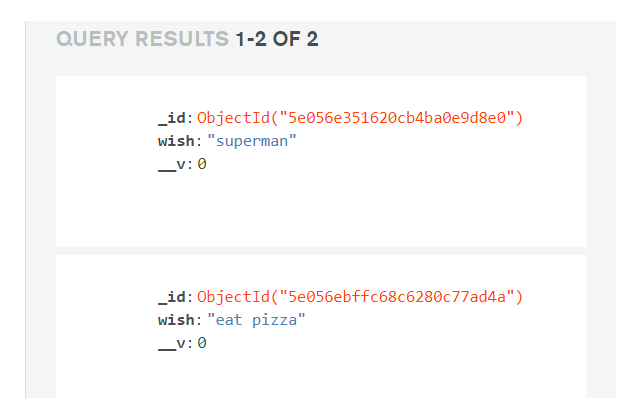
    // data.push(req.body.item);

    // res.send(JSON.stringify(req.body.item));

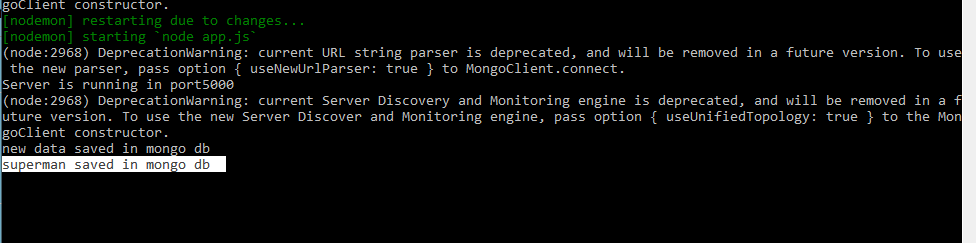
  });

};





\_id = unique key like primary key of SQL db



* Whatever we write in callback function of mongoose save can be seen in terminal
* We can also write catch with then

// mongo save query

Item.save()

      .then(data => {

        console.log(`${req.body.item} saved in mongo db`);

      })

      .catch(err => {

        throw err;

      });

Fetch from Mongo DB

* For fetch we will use .find() method. This will fetch data from database

Routes.js

const express = require("express");

const app = express();

// let data = ["code", "sleep"]; // This data we will read in home.ejs

const mongoose = require("mongoose");

const { mongourl } = require("./config/keys");

const Wish = require("./model/wish");

mongoose.connect(mongourl, { useNewUrlParser: true, useUnifiedTopology: true }); // connect to mongoDB

module.exports = app => {

  // get routes - Read Operation

  app.get("/", (req, res) => {

    Wish.find({}).then(data => { // empty object is for all data

      console.log(data);

res.send({ post: "data inserted" }); // response just for reloading

      res.render("home", { wish: data }); // passing data to home.ejs view page

    });

  });

};

Home.ejs

<html>

  <head>

    <title>Client</title>

    <link rel="stylesheet" href="/style.css" />

    <link

      rel="stylesheet"

      href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css"

    />

  </head>

  <body>

    <%- include('partials/nav'); %>

    <h2>This is my home page.</h2>

    <form id="myForm">

            <input type="text" name="item" />

            <button type="submit">Add</button>

    </form>

    <div class="collection">

        <ul>

            <% wish.forEach(item => { %>

                <li><span onclick="deleteMe(this)"><%= item.wish %></span></li>

            <% }) %>

        </ul>

    </div>

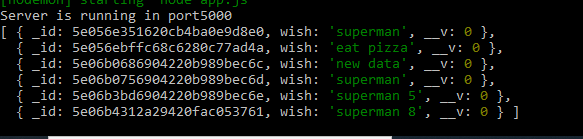
    </h3>

  </body>

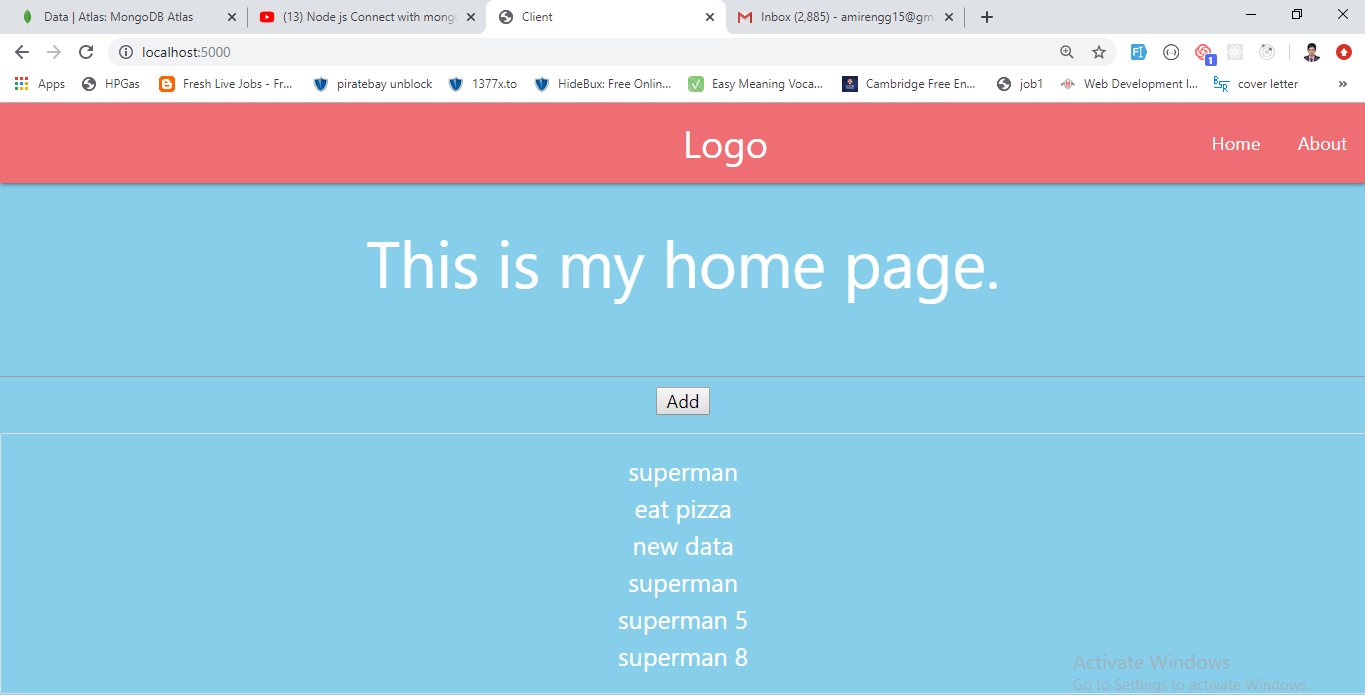
</html>

<script src="/script.js"></script>

Terminal



Browser:



Delete Operation

// delete routes  - Delete Operation

  app.delete("/remove/:id", (req, res) => {

    Wish.findOneAndRemove({ wish: req.params.id }).then(() => {

      console.log(`${req.params.id} is successfully deleted`);

      res.send({ post: "deleted" }); // sending response just for reloading

    });

    /\* data = data.map(item => {

      if (item !== req.params.id) {

        return item;

      }

    });

    console.log(data);

    res.send(data); \*/

  });

TRICKS MONGODB

TRICK 1:

* To insert in database

const mongoose = require("mongoose");

const { mongourl } = require("./config/keys");

const Wish = require("./model/wish");

mongoose.connect(mongourl);  // connect to mongoDB

app.post("/sent-data", (req, res) => {

    const Item = new Wish({

      wish: req.body.item

    });

    Item.save()

      .then(data => {

        console.log(`${req.body.item} saved in mongo db`);

        res.send({ post: "data inserted" }); // sending response just for reloading

      })

      .catch(err => {

        throw err;

      });

});

TRICK2 – Read

app.get("/", (req, res) => {

    Wish.find({}).then(data => {

      // empty object means fetch all data

      console.log(data);

      res.render("home", { wish: data }); // passing data to home.ejs view page

    });

  });

TRICK3 – Delete

// delete routes - Delete Operation

  app.delete("/remove/:id", (req, res) => {

    Wish.findOneAndRemove({ wish: req.params.id }).then(() => {

      console.log(`${req.params.id} is successfully deleted`);

      res.send({ post: "deleted" }); // sending response just for reloading

    });

});

Error with mongodb export/import

* Now there is this one error comes with models

You cannot export and require like we have done throws some

Step1 - model/wish.js

replace

module.exports = mongoose.model("wishes", WishSchema); // wishes is collection name

with

mongoose.model("wishes", WishSchema); // wishes is collection name

Step2 – app.js

Add - require('./models/wish');

Step3 – routes.js

Remove

const Wish = require("./model/wish");

Add

const Wish = mongoose.model("wishes");// wishes is schema name or table name

Deployment of project in Heroku

STEP1 - In heroku port is set so we will grab it by environment variable

App.js

const port = process.env.PORT || 5000;

STEP2 – We type node app or nodemon appin terminal. But how will heroku know about.

Therefore in package.json

{

  "name": "wish-list",

  "version": "1.0.0",

  "description": "awesome app",

  "main": "app.js",

  "scripts": {

    "start": "node app"

  },

  "author": "Amir Mustafa",

  "license": "ISC",

  "dependencies": {

    "cors": "^2.8.5",

    "ejs": "^3.0.1",

    "express": "^4.17.1",

    "mongoose": "^5.8.3"

  }

}

STEP3 – Create account on heroku get your username and password. It is free

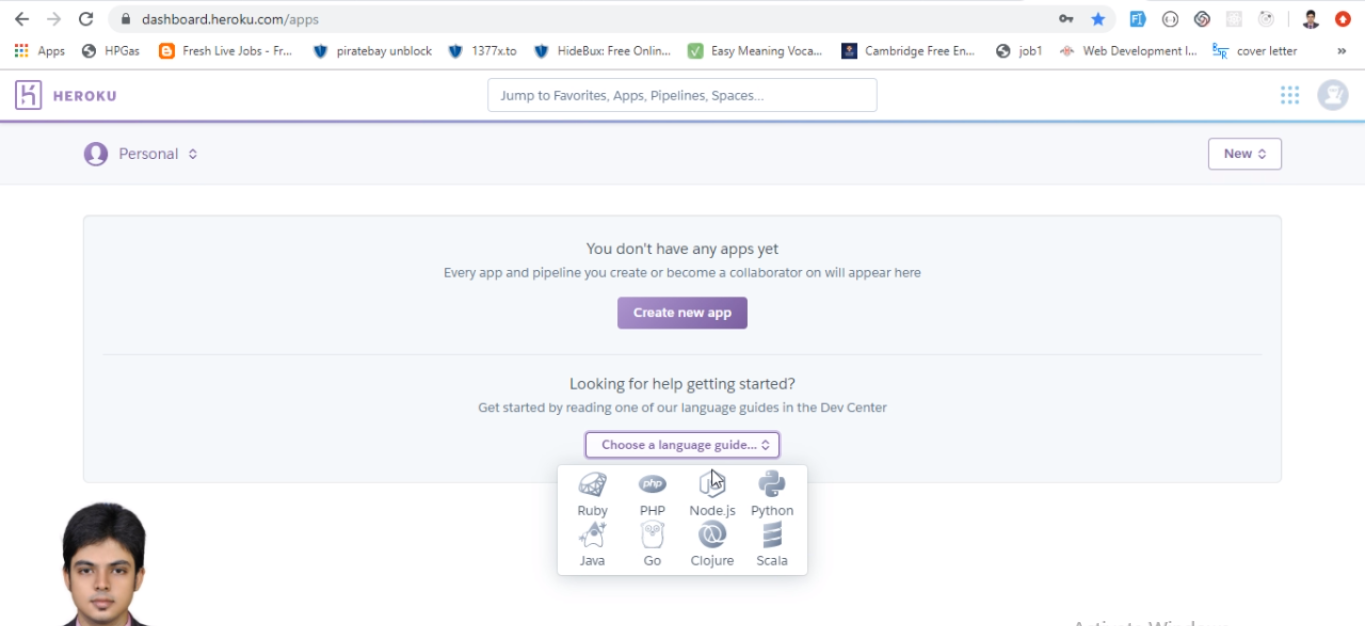
User – [amirengg15@gmail.com](mailto:amirengg15@gmail.com), password – 11\*\*\*\*\*\*41@\*\*ir

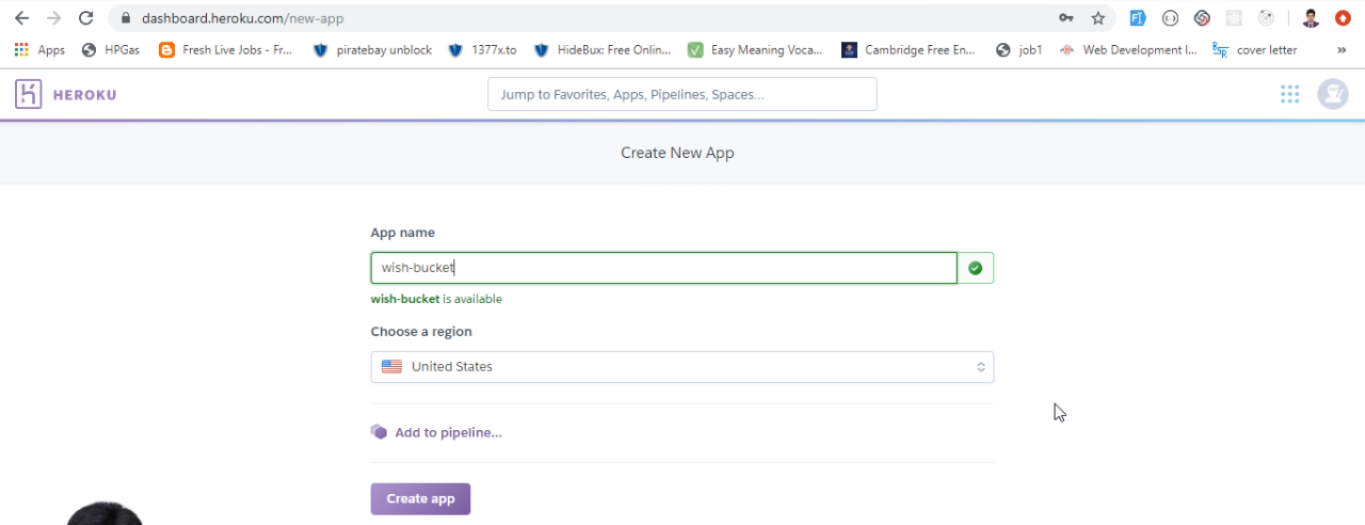
Heroku full video:

<https://www.loom.com/share/079b8f654c264d19926a0310e56fdc5c>

<https://www.loom.com/share/3c7f555a8eca4031bf83ba507cf1b3a2>

Once we enter into video we first create the app name like wish bucket





Once you create app, the next page we get is lists of command we have to execute.

<https://dashboard.heroku.com/apps/wish-bucket/deploy/heroku-git>

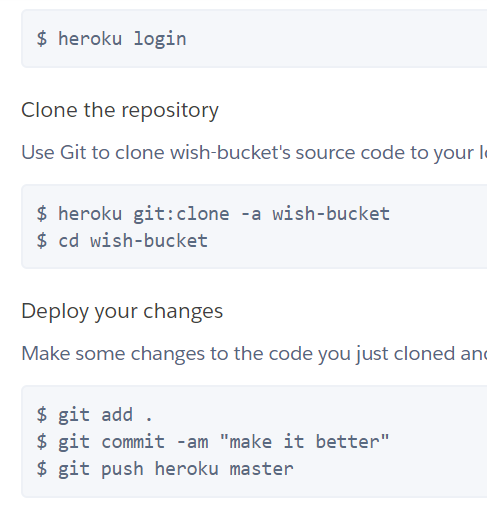
STEP 4 – Install Heroku on local system. Once you install heroku in local system check

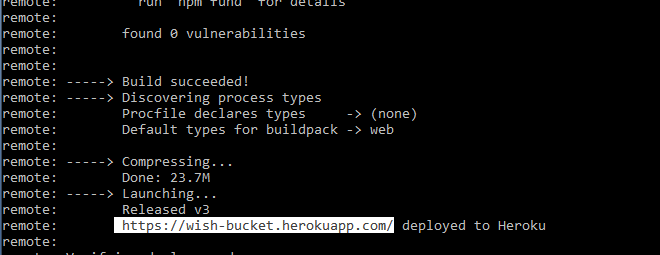
heroku –v // gets you version

heroku login // browser opens asks for credentials

now run the command given in above link

In the end you get the website link





* In production page(prod.js) we have given

module.exports = {

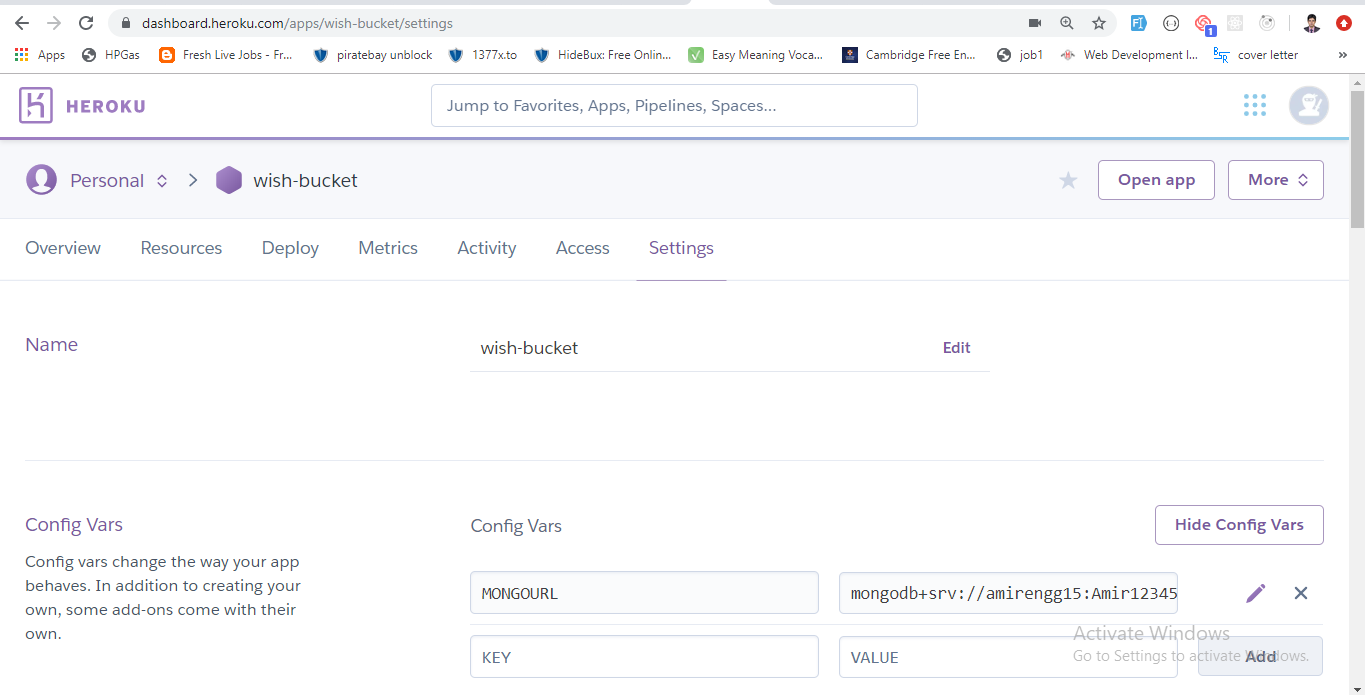
  mongourl: process.env.MONGOURL // the url we will fetch from heroku

};

So we will mongo db url there

Login to heroku 🡪 Settings 🡪 Reveal Config Vars

Add Key as MONGOURL and value as db url



Heroku Project - <https://wish-bucket.herokuapp.com/>

Video of project- <https://www.loom.com/share/fecc880e539a4f49a57c239f0650ad50>

MERN Stack TRICKS

* Now instead of creating external index.html or Node’s template engine i.e. EJS, we will connect React JS with Node and of course we are using Mongo DB Atlas as above
* No code from server side is changed as discussed above just we will sent request from React JS

TRICK 1 – **React JS inside Node JS**

Create a client folder inside Node JS project. Install/paste your React JS project here

TRICK 2 – **Start both the server**

i.e.

path – project(Node Project)

nodemon app

path – project/client (React Project)

npm start

* Now go to home component (React client/src/components/Home.js)

A simple form is created and we are sending to node server using fetch post method

import React, { Component } from "react";

class Home extends Component {

  state = {

    text: ""

  };

  handleSubmit = e => {

    e.preventDefault();

    // const url = "http://localhost:5000/sent-data"; // Node JS Server route (check route.js)

    let usp = new URLSearchParams();

    console.log(e.target);

    for (const pair of new FormData(e.target)) {

      usp.append(pair[0], pair[1]);

    }

    // proxy is written in Client's package js which target's node server port

    fetch("/sent-data", {

      method: "post",

      body: usp

    })

      .then(res => res.json())

      .then(res => {

        console.log(res);

      });

  };

  render() {

    return (

      <div className="row">

        <h4>React Form</h4>

        <form onSubmit={e => this.handleSubmit(e)} className="col s12">

          <input type="text" name="item" />

          <button type="submit" className="waves-effect waves-light btn">

            Add

          </button>

        </form>

      </div>

    );

  }

}

export default Home;

TRICK – Now there is two ways to connect to node js server, I mean the url

1. 1st way write url of server in a variable

// Node JS Server route (check route.js)

const url = "http://localhost:5000/sent-data";

fetch(url, {

      method: "post",

      body: usp

    })

      .then(res => res.json())

      .then(res => {

        console.log(res);

      });

  };

In the Node JS app.js CORS is installed and used to allow external requests

App.js

const express = require("express");

const path = require("path");

const cors = require("cors");

const bodyParser = require("body-parser");

require("./model/wish");

const app = express();

app.use(cors()); // Use this after the variable declaration

app.use(bodyParser.json());

app.use(

  bodyParser.urlencoded({

    extended: false

  })

);

Route.js (Same as previous – Not Changed)

module.exports = app => {

app.post("/sent-data", (req, res) => {

   // Save in Mongo DB Atlas

const Item = new Wish({

      wish: req.body.item // this we are getting using body parser installed

    });

    Item.save()

      .then(data => {

        console.log(`${req.body.item} saved in mongo db`);

        res.send({ post: `${req.body.item} - saved in mongo db` }); // sending response just for reloading

      })

      .catch(err => {

        throw err;

      });

});

});

1. In client’s package.json (i.e. React JS) We will write a proxy to node server url

Client/package.json

{

  "name": "react-wish-bucket",

  "version": "0.1.0",

  "private": true,

  "dependencies": {

    "@testing-library/jest-dom": "^4.2.4",

    "@testing-library/react": "^9.4.0",

    "@testing-library/user-event": "^7.2.1",

    "react": "^16.12.0",

    "react-dom": "^16.12.0",

    "react-router-dom": "^5.1.2",

    "react-scripts": "3.3.0"

  },

  "scripts": {

    "start": "react-scripts start",

    "build": "react-scripts build",

    "test": "react-scripts test",

    "eject": "react-scripts eject"

  },

  "eslintConfig": {

    "extends": "react-app"

  },

  "browserslist": {

    "production": [

      ">0.2%",

      "not dead",

      "not op\_mini all"

    ],

    "development": [

      "last 1 chrome version",

      "last 1 firefox version",

      "last 1 safari version"

    ]

  },

  "proxy": <http://localhost:5000> // Node Server url

}

* Restart React JS server
* Now in fetch command we will write /sent-data (node route) **instead of passing node direct url**

Suppose

Node JS – <http://localhost:5000>

React JS – <http://localhost:3000>

Without writing proxy in client’s package.json

Will go to <http://localhost:3000/sent-data> (**wrong port** – This is **react js port**)

After writing proxy, will go to <http://localhost:5000/sent-data> (Right port - i.e. Node JS Port where sent-data route is written)

Client/component/Home.js

import React, { Component } from "react";

class Home extends Component {

  state = {

    text: ""

  };

  handleSubmit = e => {

    e.preventDefault();

    // const url = "http://localhost:5000/sent-data"; // Node JS Server route (check route.js)

    let usp = new URLSearchParams();

    console.log(e.target);

    for (const pair of new FormData(e.target)) {

      usp.append(pair[0], pair[1]);

    }

    // proxy is written in Client's package js which target's node

    fetch("/sent-data", {

      method: "post",

      body: usp

    })

      .then(res => res.json())

      .then(res => {

        console.log(res);

      });

  };

  render() {

    return (

      <div className="row">

        <h4>React Form</h4>

        <form onSubmit={e => this.handleSubmit(e)} className="col s12">

          <input type="text" name="item" />

          <button type="submit" className="waves-effect waves-light btn">

            Add

          </button>

        </form>

      </div>

    );

  }

}

export default Home;

RUN multiple command in one terminal

* We can run both React and Node server in one terminal.
* Install package – concurrently – npm install concurrently

Package.json

"scripts": {

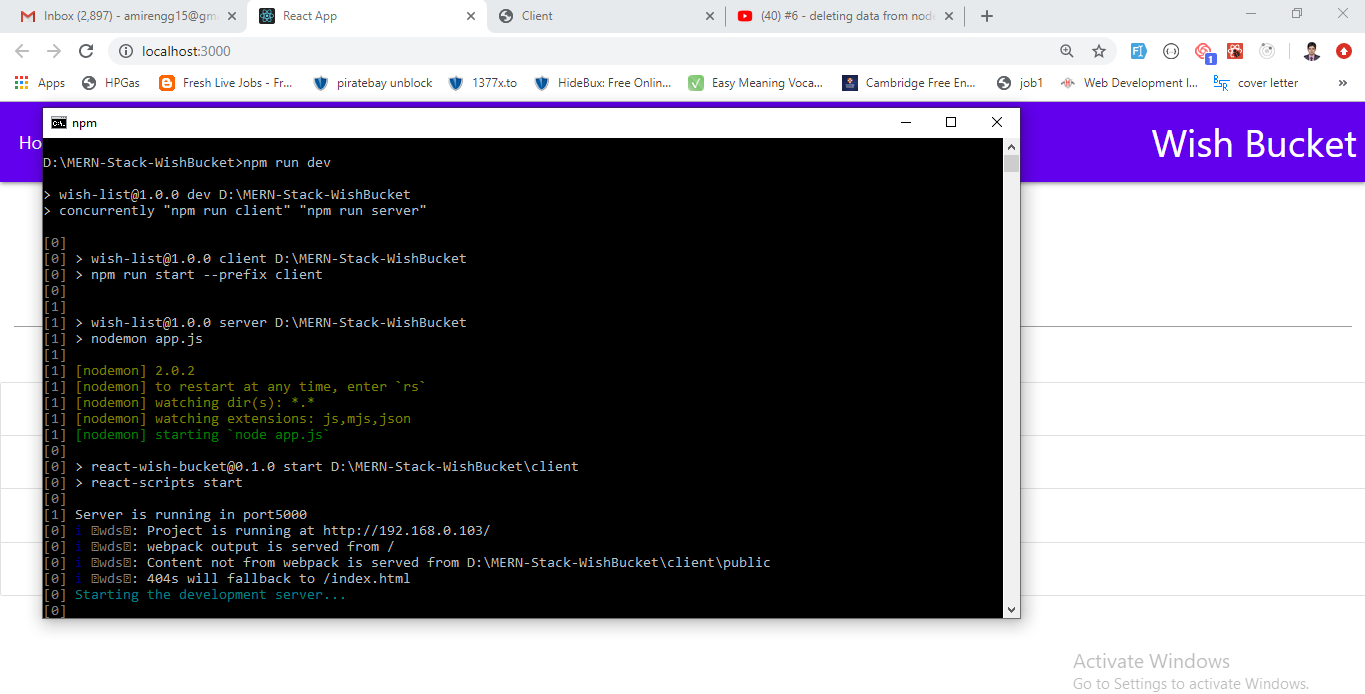
    "start": "node app",

    "client": "npm run start --prefix client", // Execute inside client folder

    "server": "nodemon app.js",

    "dev": "concurrently \"npm run client\" \"npm run server\""

  },



Deleting on Click

* On clicking the anchor list in react that element should delete

import React, { Component } from "react";

class Home extends Component {

  state = {

    text: "",

    mywishes: [{ \_id: 1, wish: "loading" }]

  };

  handleDelete = id => { // client i.e. react sending to node delete route

    fetch("/remove/" + id, { method: "delete" })

      .then(res => res.json())

      .then(res => { // from db data is removed + new array removing deleted id

        const newwishes = this.state.mywishes.filter(item => {

          return item.\_id !== id; // return id which is not equal to passed id

        });

        console.log(newwishes);

        this.setState({ mywishes: newwishes });

      });

  };

  handleSubmit = e => {

    e.preventDefault();

    // const url = "http://localhost:5000/sent-data"; // Node JS Server route (check route.js)

    let usp = new URLSearchParams();

    console.log(e.target);

    for (const pair of new FormData(e.target)) {

      usp.append(pair[0], pair[1]);

    }

    // proxy is written in Client's package js which target's node

    fetch("/sent-data", {

      method: "post",

      body: usp

    })

      .then(res => res.json())

      .then(res => {

        this.setState({

          mywishes: [...this.state.mywishes, res]

        });

      });

  };

  componentDidMount() {

    // for inserting

    fetch("/data")

      .then(res => res.json())

      .then(res => {

        this.setState({

          mywishes: res

        });

      });

  }

  render() {

    const list = this.state.mywishes.map(item => {

      return (

        <a

          className="collection-item btn-link"

          key={item.\_id}

          onClick={() => this.handleDelete(item.\_id)}

        >

          {item.wish}

        </a>

      );

    });

    return (

      <div className="row">

        <h4>React Form</h4>

        <form onSubmit={e => this.handleSubmit(e)} className="col s12">

          <input type="text" name="item" />

          <button type="submit" className="waves-effect waves-light btn">

            Add

          </button>

        </form>

        <div className="collection">{list}</div>

      </div>

    );

  }

}

export default Home;

routes.js (node server)

// delete routes  - Delete Operation

  app.delete("/remove/:id", (req, res) => {

    res.send({ data: req.params.id });

// removing from mongo DB

    Wish.findOneAndRemove({ \_id: req.params.id }).then(() => {

      console.log(`${req.params.id} is successfully deleted`);

      res.send({ post: "deleted" }); // sending response just for reloading

    });

  });

Changes in Production Level – MERN Stack

* In the development mode we were running two servers i.e. react server (port 3000) and node js server separately (port 5000) separately
* Once react js coding is done run – npm run build
* Now we will tell node to run pick up index.html from client’s build folder
* NOTE if you rewrite code in react, run again npm run build
* Kill react server, just run nodemon start in node. Now react will be served from node js
* Now in Node app.js. Earlier we were using ejs as default template engine (comment that) and server

App.js (node)

const express = require("express");

const cors = require("cors");

const bodyParser = require("body-parser");

…

…

// this line must be written to set default view type - ejs , inside route file used - res.render()

// app.set("view engine", "ejs"); // COMMENTED EJS

// import routes

require("./routes")(app);

// in case of production

app.use(express.static("client/build"));

const path = require("path");

app.get("\*", (req, res) => {

  res.sendFile(path.resolve(\_\_dirname, "client", "build", "index.html"));

});

app.listen(port, () => {

  console.log("Server is running in port" + port);

});

And in routes.js

Comment route taking to ejs

module.exports = app => {

  // get routes - Read Operation

  /\* app.get("/", (req, res) => {

    Wish.find({}).then(data => {

      // empty object means fetch all data

      console.log(data);

      res.render("home", { wish: data }); // rendering EJS

    });

  }); \*/

  app.get("/data", (req, res) => { // so any other route will take to react

    Wish.find({}).then(data => {

      // empty object means fetch all data

      res.send(data); // passing data to home.ejs view page

    });

  });

TRICKS – Two ways

1. When we push code to server we push only build folder of client + node project
2. In the server we push the code and run build command from server itself like heroku (not recommended way but most developers do)

Package.json (method 2)

"scripts": {

    "start": "node app",

    "client": "npm run start --prefix client",

    "server": "nodemon app.js",

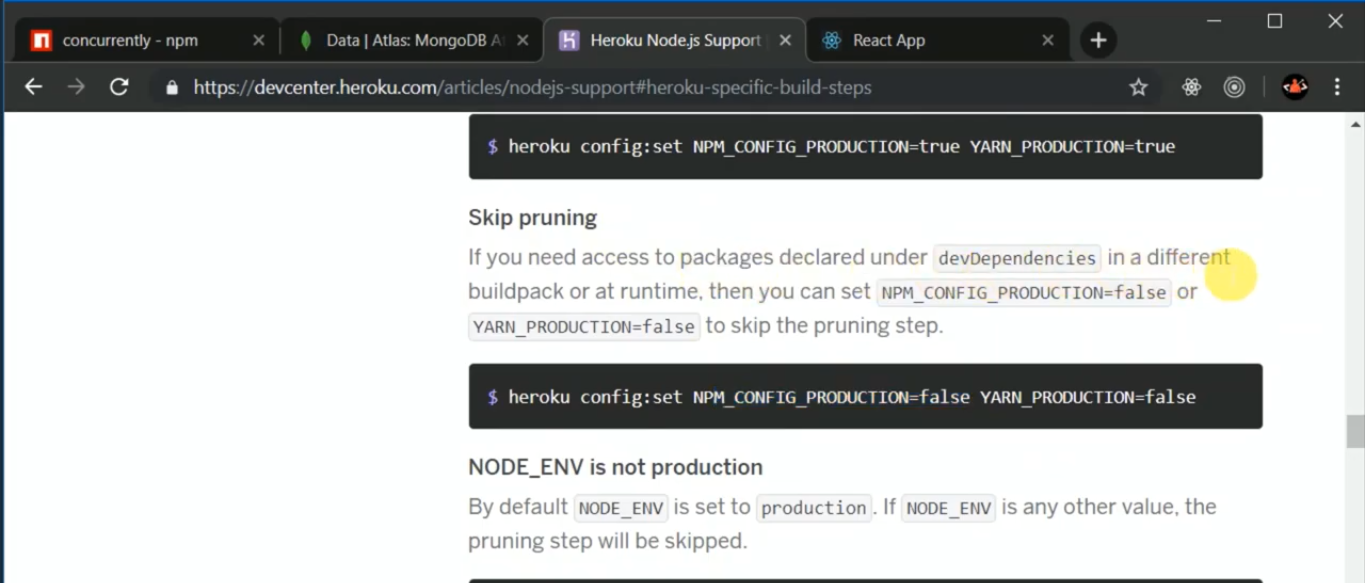
    "dev": "concurrently \"npm run client\" \"npm run server\"",

"heroku-postbuild": "NPM\_CONFIG\_PRODUCTION=false npm install --prefix client && npm run build --prefix client"

  },

<https://devcenter.heroku.com/articles/nodejs-support#build-behavior> (Follow this link)

heroku config:set NPM\_CONFIG\_PRODUCTION=true YARN\_PRODUCTION=true



Implementing Redux in MERN Stack Wish Bucket

STEP 1 – Download wish bucket from github

Npm install (project) – node dependencies

Npm install (project/client) – react dependencies

//Npm run build (project/client) – react production

STEP2 – Install redux packages

Npm install redux react-redux redux-thunk

STEP3 – run our concurrently package –

Npm run dev- this runs both our react server (port 3000) + node server (port 5000) - check package.json

Open - <http://localhost:3000/>

TRICK - Instead of keeping data in state in a component we will now keep data in redux’s reducer and access data from there

Concept:

Earlier in home component we were fetching data from state of home component. Now we will fetch from reducer i.e. redux. No more state in home component

We will cut paste state to reducer page

STEP1 – Connection with reducer/redux.

Install 3 packages:

npm install redux react-redux redux-thunk

client/index.js (React’s index js)

import React from "react";

import ReactDOM from "react-dom";

import { Provider } from "react-redux";

import thunk from "redux-thunk";

import { createStore, applyMiddleware } from "redux";

import { composeWithDevTools } from "redux-devtools-extension"; // chrome’s debugging tool

import reducer from "./myreducer/reducer"; // here all data is kept

import "./index.css";

import App from "./App";

import \* as serviceWorker from "./serviceWorker";

const mystore = createStore(

  reducer,

  composeWithDevTools(applyMiddleware(thunk))

);

ReactDOM.render(

  <Provider store={mystore}>

    <App />

  </Provider>,

  document.getElementById("root")

);

// If you want your app to work offline and load faster, you can change

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();

// Traverse Steps: call function (component)🡪 mapDispatchToProps (home component)(compoenent) 🡪 action creator(myactions/action.js) 🡪 reducer.js.

Client/src/Home.js (Home component)

import React, { Component } from "react";

import { connect } from "react-redux";

import { handleInputAction, fetchWishAction } from "../myactions/action";

class Home extends Component {

  // state = {

  //   text: "",

  //   mywishes: [{ \_id: 1, wish: "loading" }] // some dummy data prev set

  // };

  handleDelete = id => {

    fetch("/remove/" + id, { method: "delete" }) // React JS delete request

      .then(res => res.json())

      .then(res => {

        const newwishes = this.props.mywishes.filter(item => {

          return item.\_id !== id; // return id which is not equal to passed id

        });

        /\* const newwishes = this.state.mywishes.filter(item => {

          return item.\_id !== id; // return id which is not equal to passed id

        }); \*/

        console.log(newwishes);

        this.setState({ mywishes: newwishes });

      });

  };

  handleSubmit = e => {

    e.preventDefault();

    // const url = "http://localhost:5000/sent-data"; // Node JS Server route (check route.js)

    let usp = new URLSearchParams();

    console.log(e.target);

    for (const pair of new FormData(e.target)) {

      usp.append(pair[0], pair[1]);

    }

    // proxy is written in Client's package js which target's node

    // now this will target nodejs (i.e. localhost:5000) instead of localhost:3000 (i.e. React JS)

    fetch("/sent-data", {

      method: "post",

      body: usp

    })

      .then(res => res.json())

      .then(res => {

        this.setState({

          mywishes: [...this.state.mywishes, res]

        });

      });

  };

  componentDidMount() {

    this.props.fetchWish(); // calling this in mapDispatchToProps --> function --> action creator

    // for inserting

    // written in action creator for redux - function written in mapDispatchToProps

    /\* fetch("/data") // this hit's node js data route)

          .then(res => res.json())

          .then(res => {

            this.setState({

              mywishes: res

            });

          }); \*/

  }

  render() { // replaced state with props

    const list = this.props.mywishes.map(item => {

      return (

        <a

          className="collection-item"

          key={item.\_id}

          onClick={() => this.handleDelete(item.\_id)}

        >

          {item.wish}

        </a>

      );

    });

    return (

      <div className="row">

        <h4>React Form</h4>

        <form onSubmit={e => this.handleSubmit(e)} className="col s12">

          <input

            type="text"

            name="item"

            value={this.props.text}

            onChange={e => this.props.handleInput(e.target.value)}

          />

          {/\* <input              // without redux

            type="text"

            name="item"

            value={this.state.text}

            onChange={e => this.setState({ text: e.target.value })}

          />\*/}

          <button type="submit" className="waves-effect waves-light btn">

            Add

          </button>

        </form>

        <div className="collection">{list}</div>

      </div>

    );

  }

}

// getting data from reducer store

const mapStateToProps = state => {

  return {

    text: state.text,

    mywishes: state.mywishes // some dummy data prev set

  };

};

// sending this data to action creator for reducer's type and payload

const mapDispatchToProps = dispatch => {

  return {

    handleInput: input => {

      dispatch(handleInputAction(input));

    },

    fetchWish: () => {

      dispatch(fetchWishAction());

    }

  };

};

export default connect(mapStateToProps, mapDispatchToProps)(Home);

Action creator (client/src/myactions/action.js)

// for update i/p

export const handleInputAction = input => {

  return {

    type: "UPDATE\_INPUT",

    payload: input

  };

};

// For fetching data from componentDidMount --> mapDispatchToProps --> actioncreator --> reducers

export const fetchWishAction = () => {

  // We are fetching data asynchrously

  return dispatch => {

    fetch("/data") // this hit's node js data route)

      .then(res => res.json())

      .then(res => {

        /\* this.setState({  // earlier without redux we were updating state -

          mywishes: res

        }); \*/

        dispatch({ type: "GET\_WISH", payload: res });

      });

  };

};

client/myreducer/reducer.js

const iState = {    // state data of home component is now here

  text: "",

  mywishes: [{ \_id: 1, wish: "loading" }] // some dummy data prev set

};

const reducer = (state = iState, action) => {

  switch (action.type) {

    case "UPDATE\_INPUT": // updating React's i/p case (check action creator page)

      return {

        ...state,

        text: action.payload

      };

    case "GET\_WISH": // fetching this from Component Did Mount(home component)🡪 mapDispatchToProps (home component) 🡪 action creator(myactions/action.js) 🡪 reducer.js. Earlier we were fetching directly in componentDidMount()

      return {

        ...state,

        mywishes: action.payload

      };

    default:

      return state;

  }

  //   if (action.type === "UPDATE\_INPUT") {

  //     return {

  //       ...state,

  //       text: action.payload

  //     };

  //   }

  // return state;

};

export default reducer;

TRICK TO Convert any function into Redux.

* Earlier we had state in component we updated the state using setState component

Now data is in redux (i.e. reducer) so we will update by dispatch(action.payload)

* So if you have any property i.e. state property earlier is now in reducer. So in mapStateToProps

// getting data from reducer store

const mapStateToProps = state => {

  return {

    text: state.text, // RHS = data from reducer, LHS= data we will use as props in this component

    mywishes: state.mywishes // some dummy data prev set

  };

};

* If you have any functions/methods

Say componentDidMount or formSubmit

We will call function name

Eg.

old

<form onSubmit={e => this.handleSubmit(e)} className="col s12">

new

<form onSubmit={e => this.props.handleSubmit(e)} className="col s12">

Instead of writing function directly we will write in mapDispatchToProps(home component) 🡪 action creator 🡪 reducer

import { handleInputAction, fetchWishAction, handleSubmitAction

} from "../myactions/action";

// sending this data to action creator for reducer's type and payload

const mapDispatchToProps = dispatch => {

  return {

    handleInput: input => {

      dispatch(handleInputAction(input));

    },

    fetchWish: () => {

      dispatch(fetchWishAction());

    },

    handleSubmit: e => { // functions are written here

      dispatch(handleSubmitAction(e));

    }

  };

};

Action

export const handleSubmitAction = e => {

  return dispatch => {

    e.preventDefault();

// Pasted code of handleSubmit here

let usp = new URLSearchParams();

    console.log(e.target);

    for (const pair of new FormData(e.target)) {

      usp.append(pair[0], pair[1]);

    }

    // proxy is written in Client's package js which target's node

    // now this will target nodejs (i.e. localhost:5000) instead of localhost:3000 (i.e. React JS)

    fetch("/sent-data", {

      method: "post",

      body: usp

    })

      .then(res => res.json())

      .then(res => {

        // Earlier we were updating state --> Now let reducer update it's data

        /\* this.setState({

          mywishes: [...this.state.mywishes, res]

        }); \*/

// Instead of updating state we have written type and payload to reducer

        dispatch({ type: "ADD\_WISH", payload: res });

      });

  };

};

Reducer.js

const iState = {

  // state data of home component is now here

  text: "",

  mywishes: [{ \_id: 1, wish: "loading" }] // some dummy data prev set

};

const reducer = (state = iState, action) => {

  switch (action.type) {

    case "UPDATE\_INPUT": // updating React's i/p case (check action creator page)

      return {

        ...state,

        text: action.payload

      };

    case "GET\_WISH": // fetching this from Component Did Mount

      return {

        ...state,

        mywishes: action.payload

      };

    case "ADD\_WISH": // fetching this from Component Did Mount

      return {

        ...state,

        mywishes: [...state.mywishes, action.payload]

      };

    default:

      return state;

  }

};

export default reducer;

Delete using redux

Traverse steps:

**Component 🡪 action creator 🡪 reducer (this updates data in redux data)**

Home.js (Component)

import React, { Component } from "react";

import { connect } from "react-redux";

import {

  handleInputAction,

  fetchWishAction,

  handleSubmitAction,

  handleDeleteAction

} from "../myactions/action";

class Home extends Component {

  componentDidMount() {

    this.props.fetchWish(); // calling this in mapDispatchToProps --> function --> action creator

  }

  render() {

    const list = this.props.mywishes.map(item => {

      return (

        <a

          className="collection-item"

          key={item.\_id}

          onClick={() => this.props.handleDelete(item.\_id)}

        >

          {item.wish}

        </a>

      );

    });

    return (

      <div className="row">

        <h4>React Form</h4>

        <form onSubmit={e => this.props.handleSubmit(e)} className="col s12">

          <input

            type="text"

            name="item"

            value={this.props.text}

            onChange={e => this.props.handleInput(e.target.value)}

          />

          {/\* <input              // without redux

            type="text"

            name="item"

            value={this.state.text}

            onChange={e => this.setState({ text: e.target.value })}

          />\*/}

          <button type="submit" className="waves-effect waves-light btn">

            Add

          </button>

        </form>

        <div className="collection">{list}</div>

      </div>

    );

  }

}

// getting data from reducer store

const mapStateToProps = state => {

  return {

    text: state.text,

    mywishes: state.mywishes // some dummy data prev set

  };

};

// sending this data to action creator for reducer's type and payload

const mapDispatchToProps = dispatch => {

  return {

    handleInput: input => {

      dispatch(handleInputAction(input));

    },

    fetchWish: () => {

      dispatch(fetchWishAction());

    },

    handleSubmit: e => {

      dispatch(handleSubmitAction(e));

    },

    handleDelete: id => {

      dispatch(handleDeleteAction(id));

    }

  };

};

export default connect(mapStateToProps, mapDispatchToProps)(Home);

client/myactions/action.js (Action Creator)

// for update i/p

export const handleInputAction = input => {

  return {

    type: "UPDATE\_INPUT",

    payload: input

  };

};

// For fetching data from componentDidMount --> mapDispatchToProps --> actioncreator --> reducers

export const fetchWishAction = () => {

  // We are fetching data asynchrously

  return dispatch => {

    fetch("/data") // this hit's node js data route)

      .then(res => res.json())

      .then(res => {

        /\* this.setState({  // earlier without redux we were updating state -

          mywishes: res

        }); \*/

        dispatch({ type: "GET\_WISH", payload: res });

      });

  };

};

// For posting data in server

export const handleSubmitAction = e => {

  return dispatch => {

    e.preventDefault();

    // const url = "http://localhost:5000/sent-data"; // Node JS Server route (check route.js)

    let usp = new URLSearchParams();

    console.log(e.target);

    for (const pair of new FormData(e.target)) {

      usp.append(pair[0], pair[1]);

    }

    // proxy is written in Client's package js which target's node

    // now this will target nodejs (i.e. localhost:5000) instead of localhost:3000 (i.e. React JS)

    fetch("/sent-data", {

      method: "post",

      body: usp

    })

      .then(res => res.json())

      .then(res => {

        // Earlier we were updating state --> Now let reducer update it's data

        /\* this.setState({

          mywishes: [...this.state.mywishes, res]

        }); \*/

        dispatch({ type: "ADD\_WISH", payload: res });

      });

  };

};

export const handleDeleteAction = id => {

  return dispatch => {

    fetch("/remove/" + id, { method: "delete" }) // React JS delete request

      .then(res => res.json())

      .then(res => {

        /\* const newwishes = this.props.mywishes.filter(item => {

          return item.\_id !== id; // return id which is not equal to passed id

        }); \*/

        /\* const newwishes = this.state.mywishes.filter(item => {

          return item.\_id !== id; // return id which is not equal to passed id

        }); \*/

        // this.setState({ mywishes: newwishes }); // earlier we were updating state

        dispatch({ type: "REMOVE\_WISH", payload: res });

      });

  };

};

Client/myreducers/reducers.js

const iState = {

  // state data of home component is now here

  text: "",

  mywishes: [{ \_id: 1, wish: "loading" }] // some dummy data prev set

};

const reducer = (state = iState, action) => {

  switch (action.type) {

    case "UPDATE\_INPUT": // updating React's i/p case (check action creator page)

      return {

        ...state,

        text: action.payload

      };

    case "GET\_WISH": // fetching this from Component Did Mount

      return {

        ...state,

        mywishes: action.payload

      };

    case "ADD\_WISH": // fetching this from Component Did Mount

      return {

        ...state,

        mywishes: [...state.mywishes, action.payload]

      };

    case "DELETE\_WISH": // fetching this from Component Did Mount

      const newwishes = this.props.mywishes.filter(item => {

        return item.\_id !== action.payload.id; // return id which is not equal to passed id

      });

      return {

        ...state,

        mywishes: [newwishes]

      };

    default:

      return state;

  }

};

export default reducer;

TRICK – We have state access in reducer page. So pass every logic of state in reducer page.