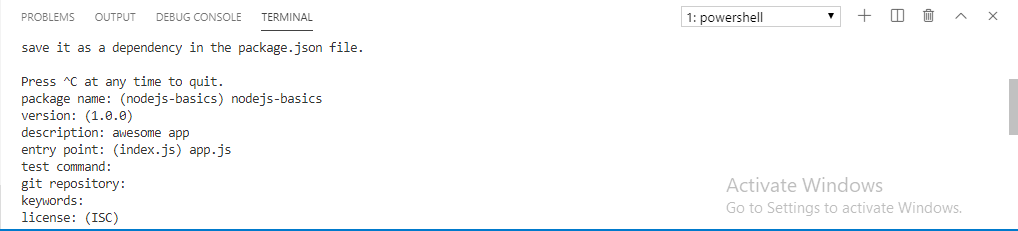
**Node JS CNQ**

**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;

}

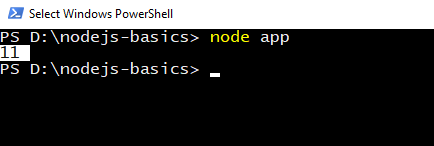
module.exports = sum;

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;

  }

};

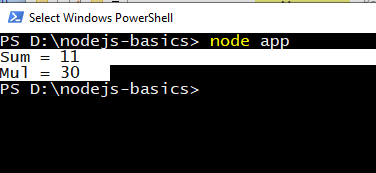
App.js

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");

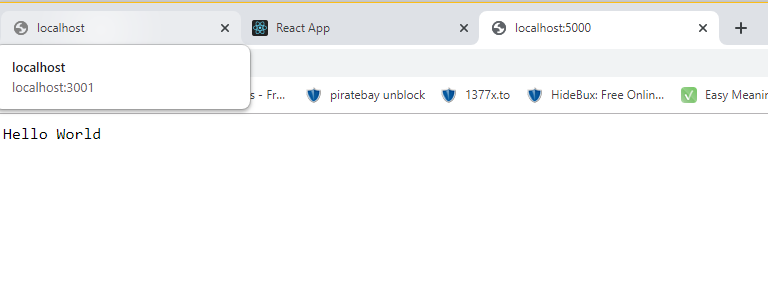
    res.end();

}).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");

});

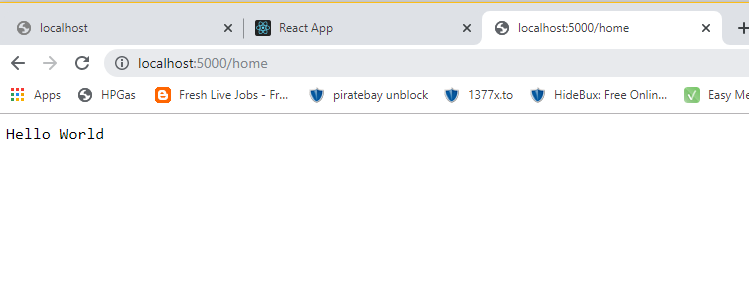
app.listen(port, () => {

  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");

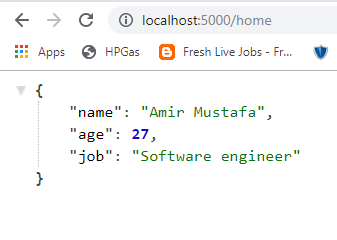
});

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 => {

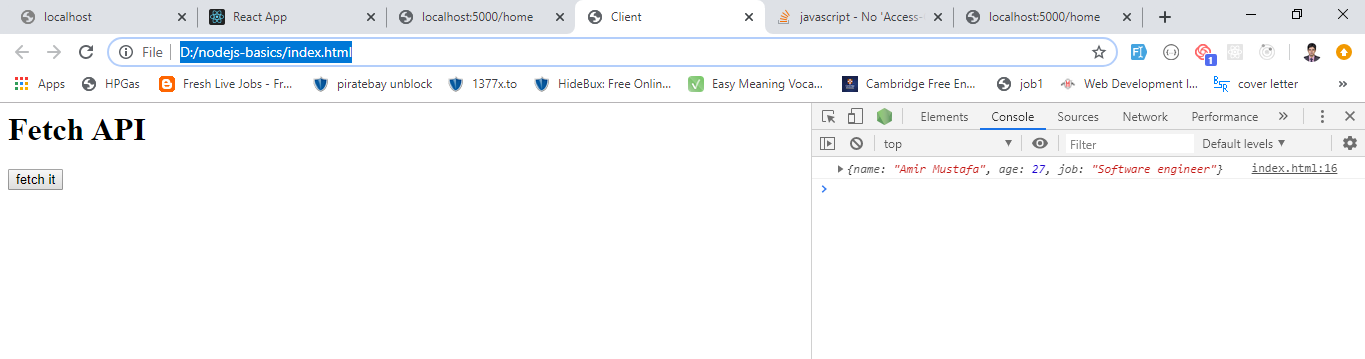
        console.log(res);

      });

  };

</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

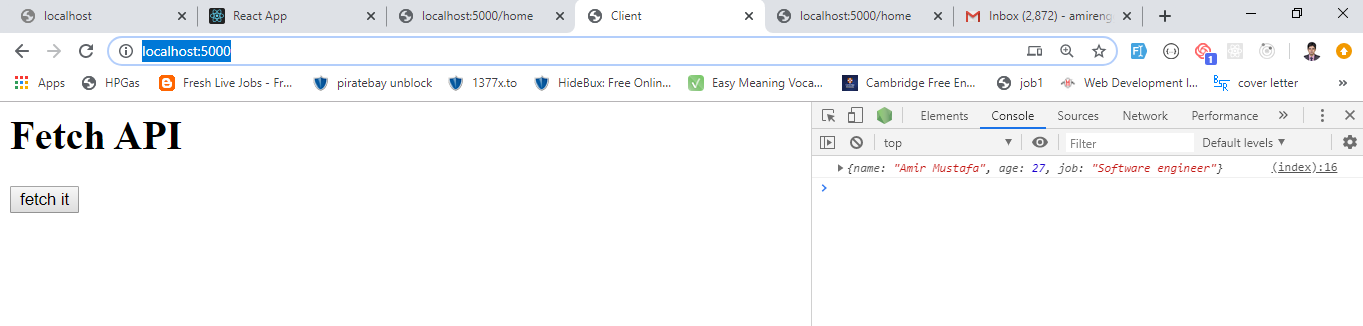
});

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 => {

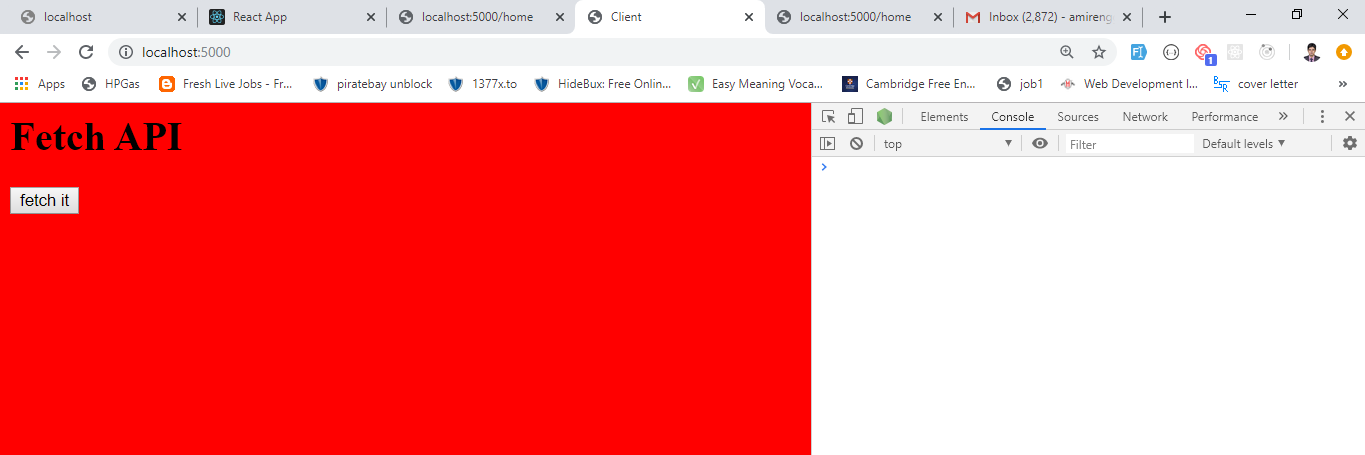
        console.log(res);

      });

  };

</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 => {

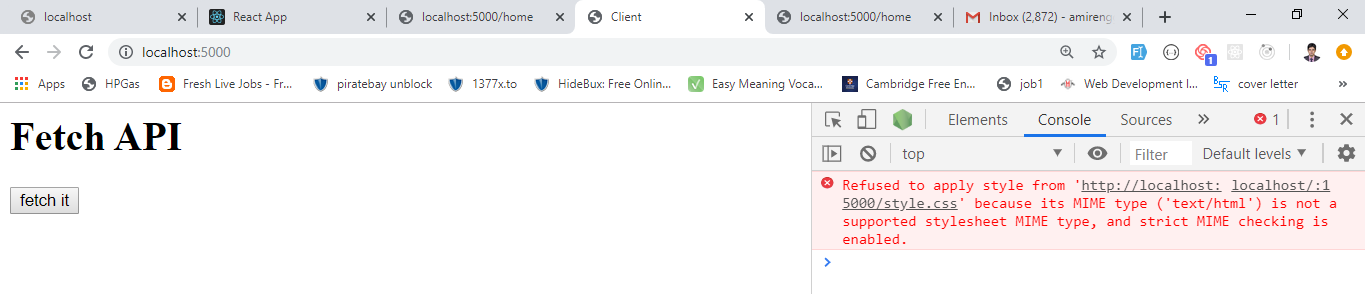
        console.log(res);

      });

  };

</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 => {

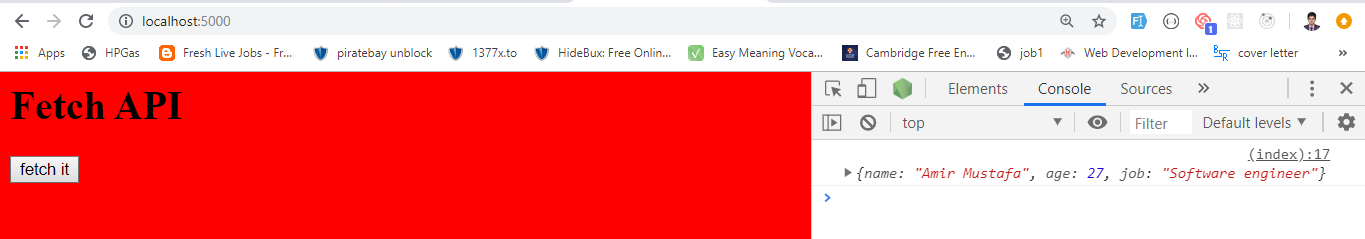
        console.log(res);

      });

  };

</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" });

  });

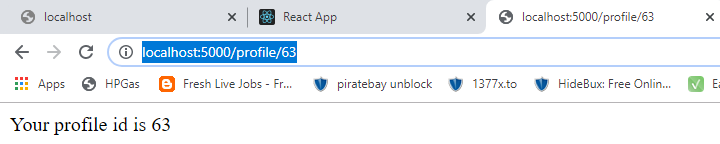
};

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>

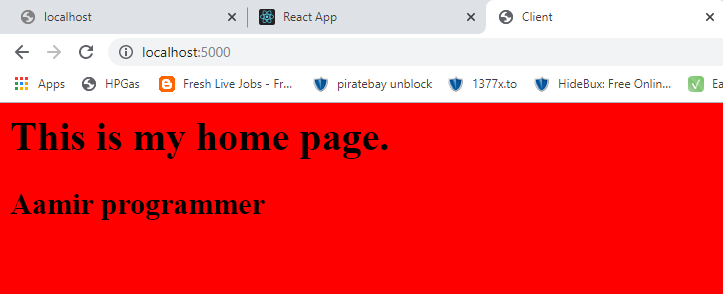
      <%= data.name%> <%= data.status%>

    </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>

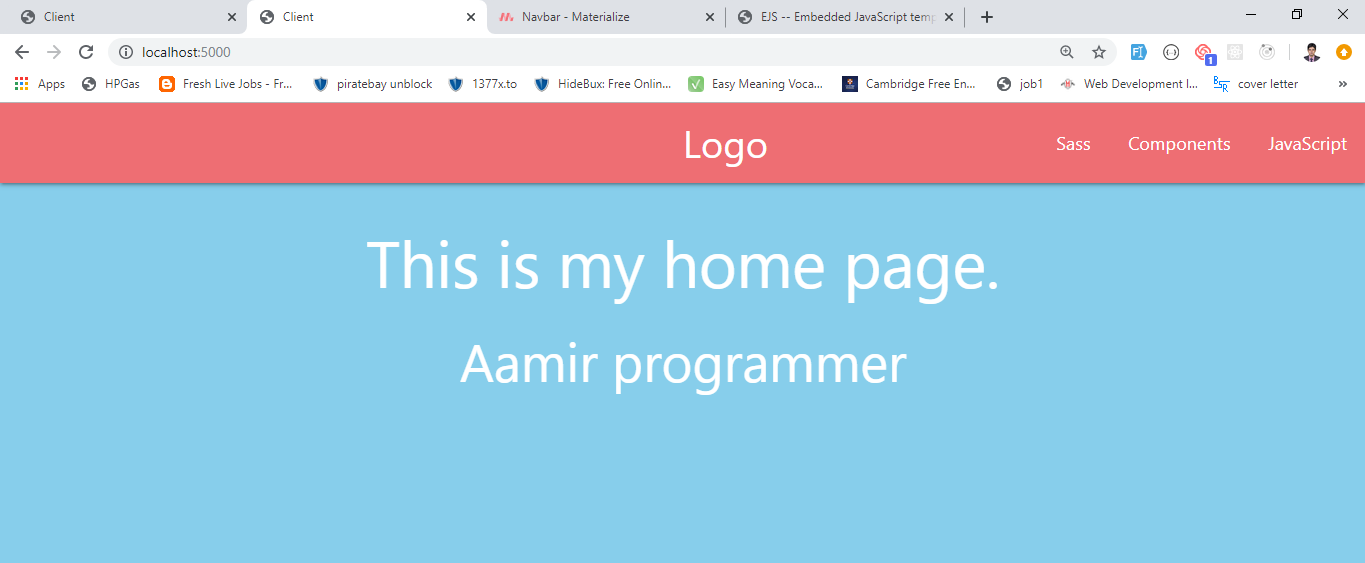
    <h2>

      <%= 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) => {

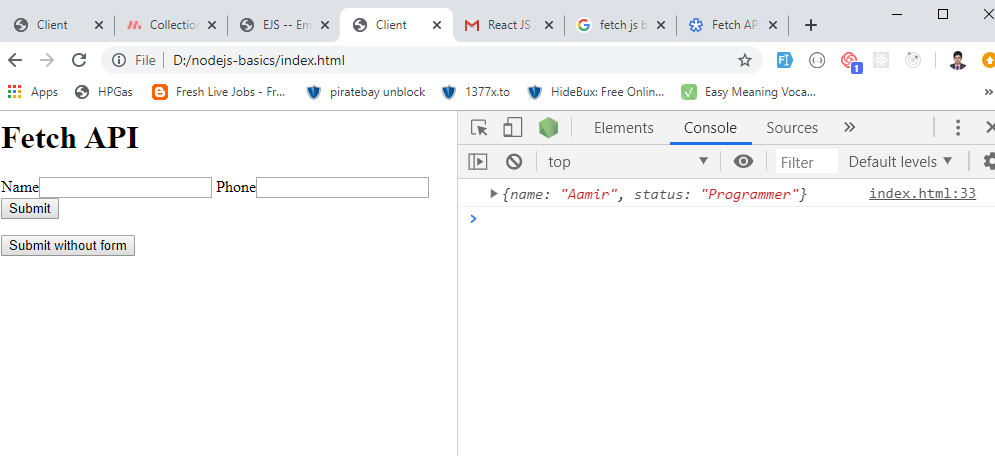
    console.log(req.body);

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

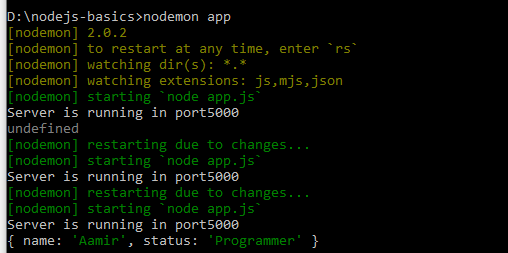
  });

};

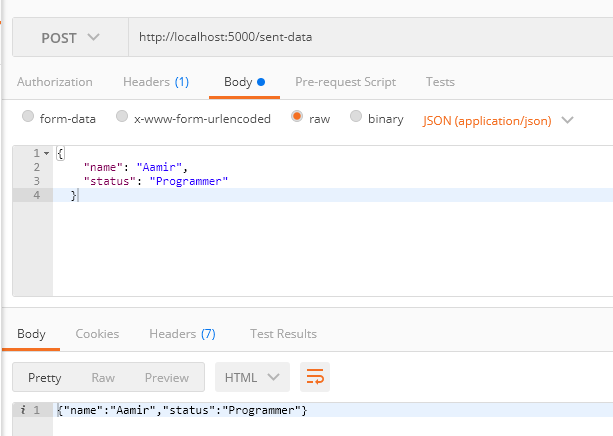
O/P



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);

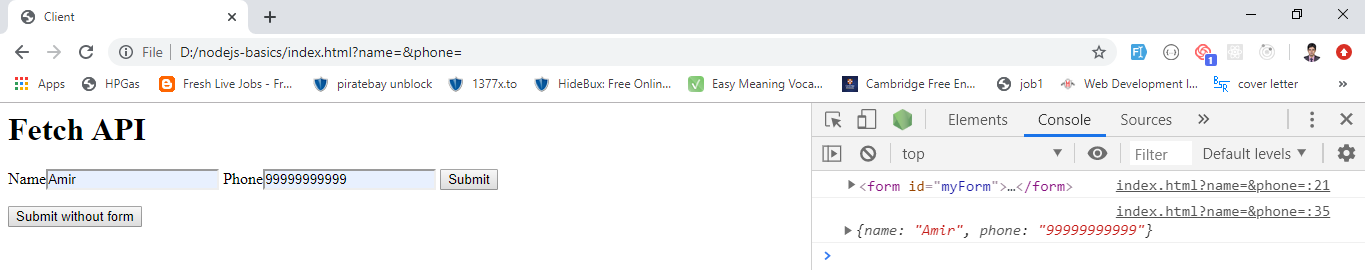
      });

  };

</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;

      }

    });

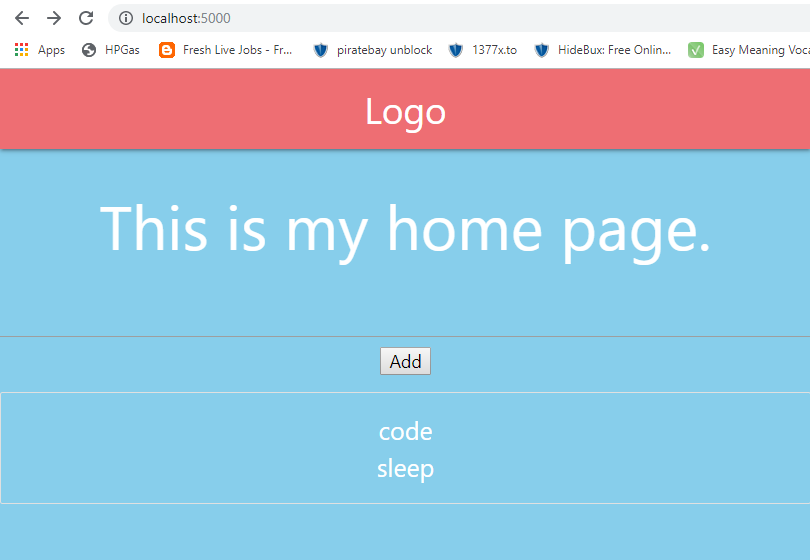
    console.log(data);

    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

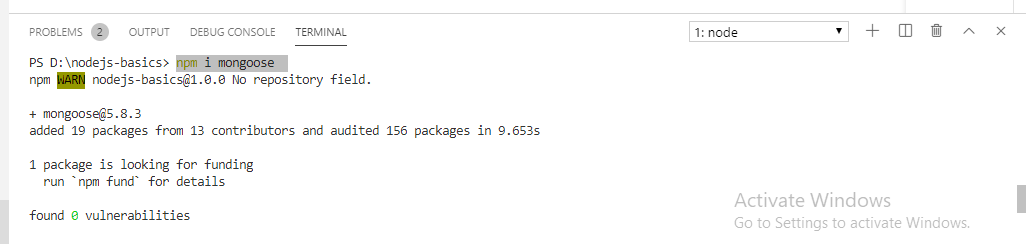
module.exports = {

    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

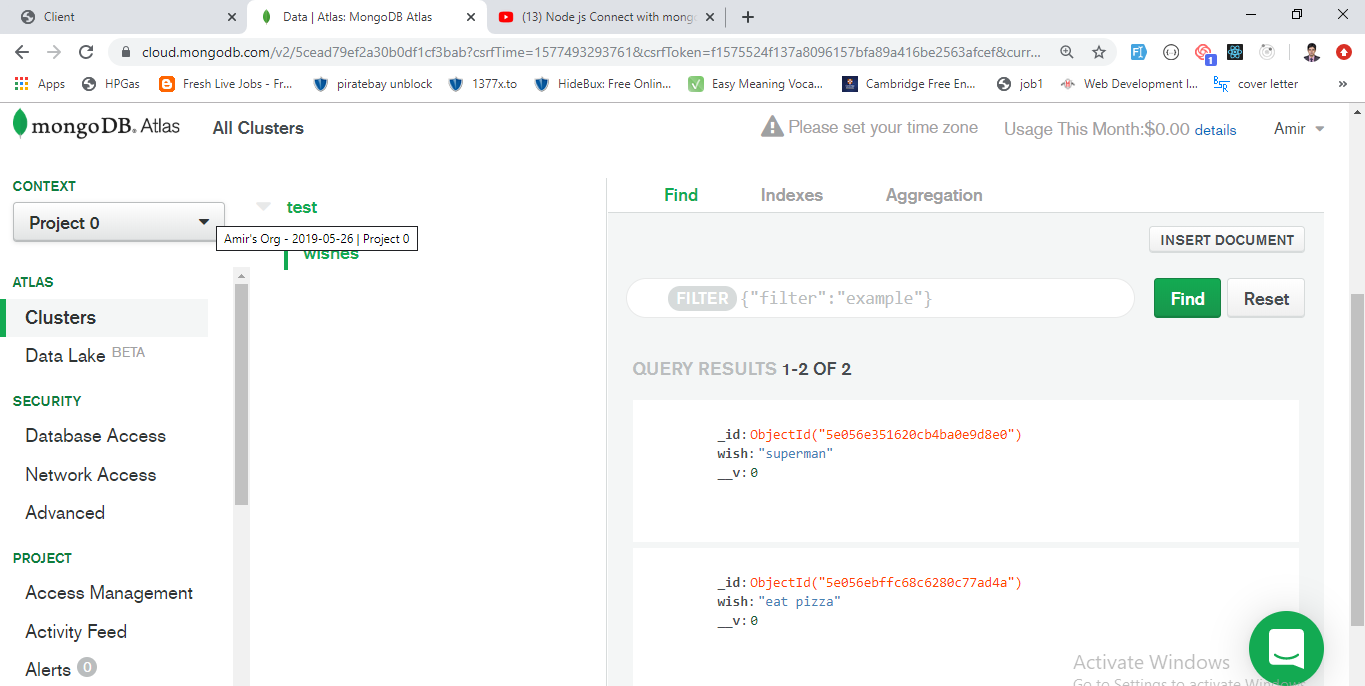
    // console.log(req.body.item);

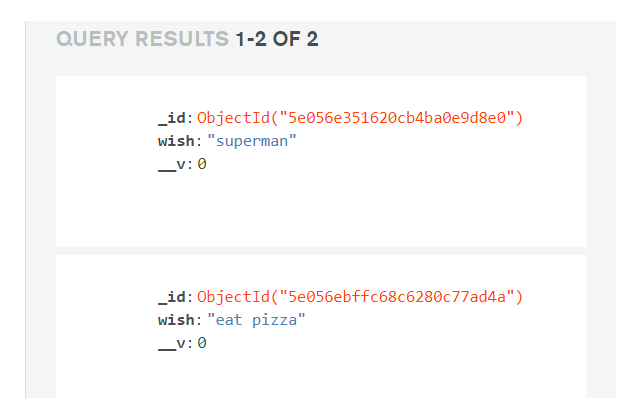
    // data.push(req.body.item);

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

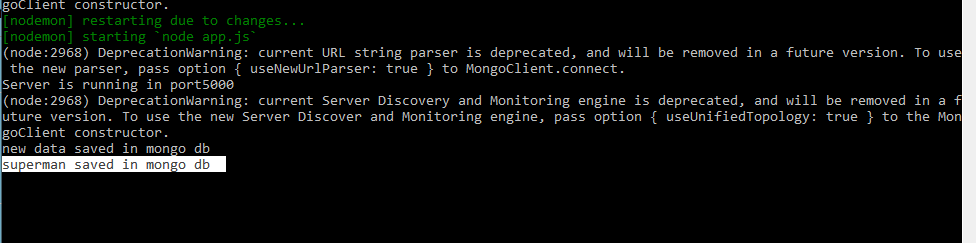
  });

};





\_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>

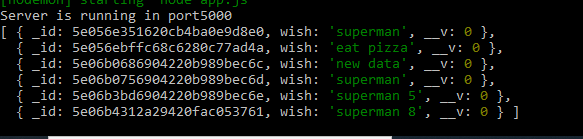
    </h3>

  </body>

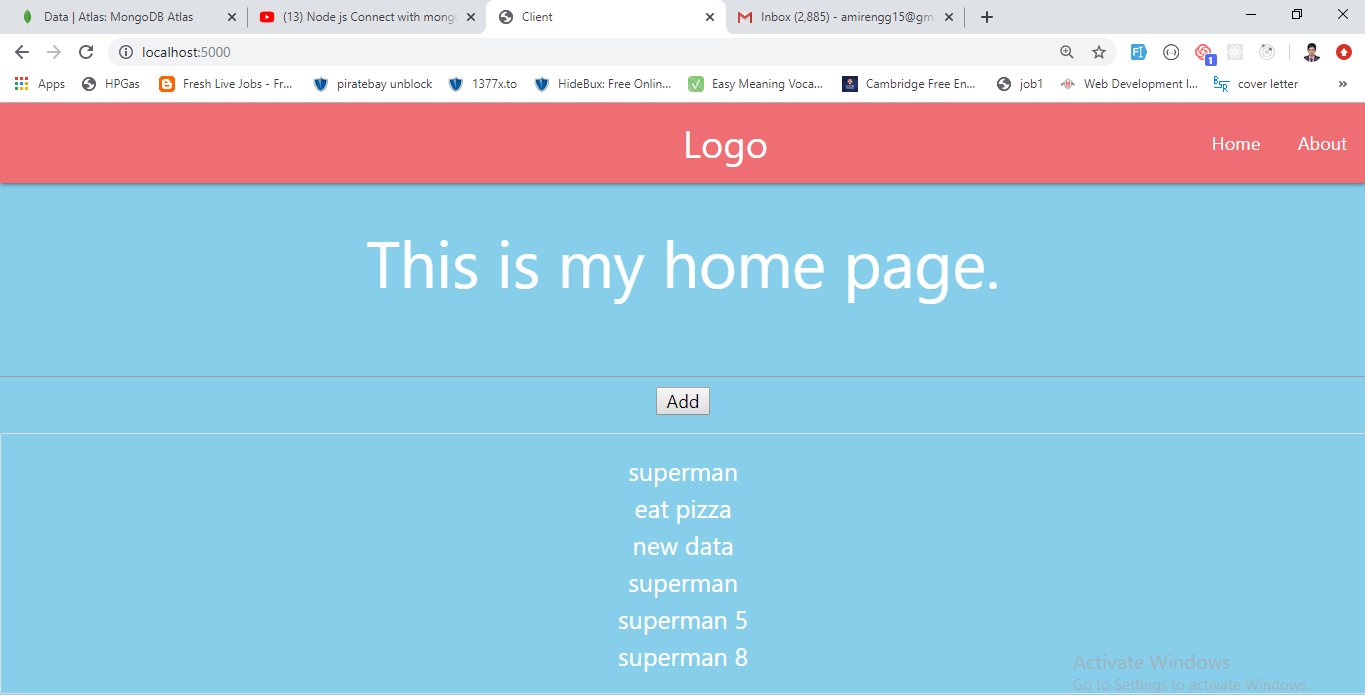
</html>

<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

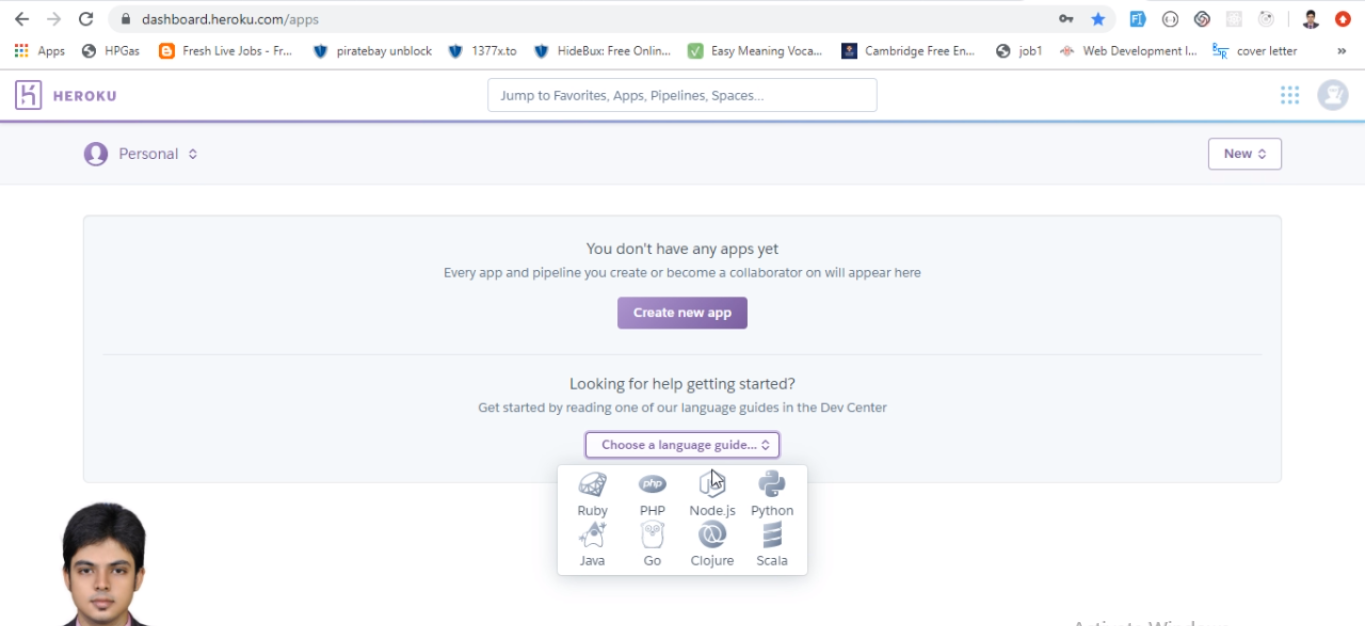
User – [amirengg15@gmail.com](mailto:amirengg15@gmail.com), password – 11\*\*\*\*\*\*41@\*\*ir

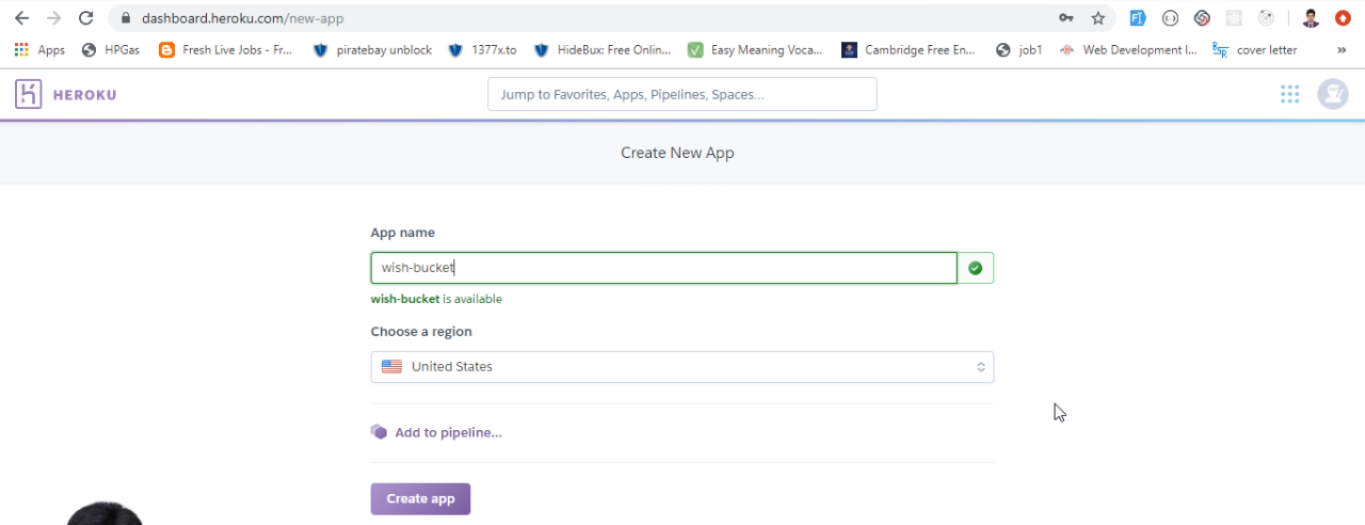
Heroku full video:

<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>

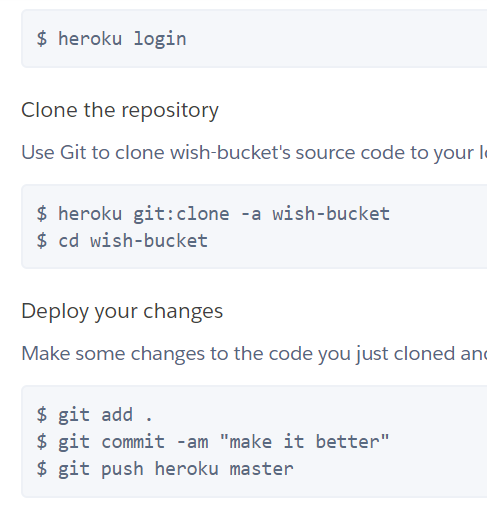
STEP 4 – Install Heroku on local system. Once you install heroku in local system check

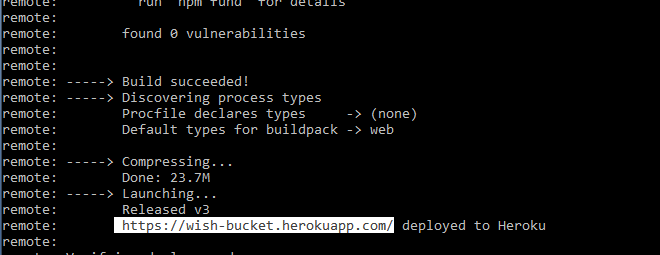
heroku –v // gets you version

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 = {

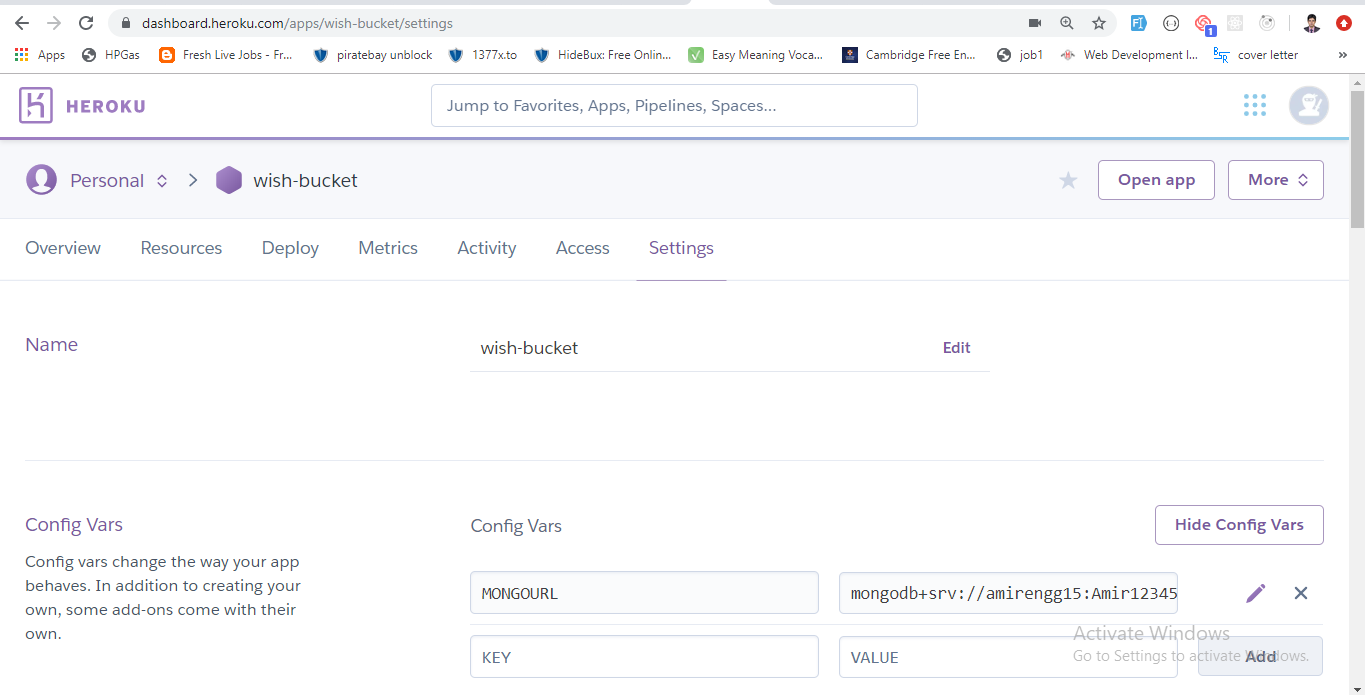
  mongourl: process.env.MONGOURL // the url we will fetch from heroku

};

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>