Hands-on Lab - Express Server(50 min)

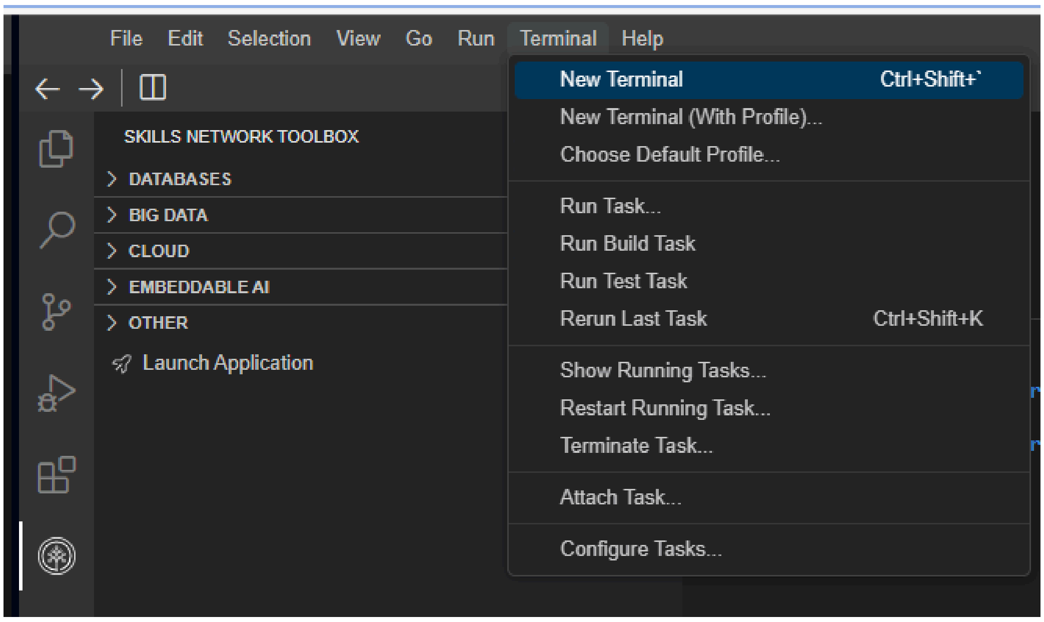
Skills Network

Objective for Exercise:

- Create express server and run it
- Work on Middlewares with Express server
- Use middleware and JWT for authentication
- Render a static HTML page through express server

Set-up : Clone lab files

1. Open a terminal window by using the menu in the editor: Terminal > New Terminal.



2. Change to your project folder.

```
cd /home/project
```

3. Check if you have the folder **lkpho-Cloud-applications-with-Node.js-and-React**

```
ls /home/project
```

If you do, you can skip to step 5.

4. Clone the git repository that contains the artifacts needed for this lab, if it doesn't already exist.

```
git clone https://github.com/ibm-developer-skills-network/lkpho-Cloud-applications-with-Node.js-and-React.git
```

5. Change to the directory for this lab.

```
cd lkpho-Cloud-applications-with-Node.js-and-React/CD220Labs/expressjs/
```

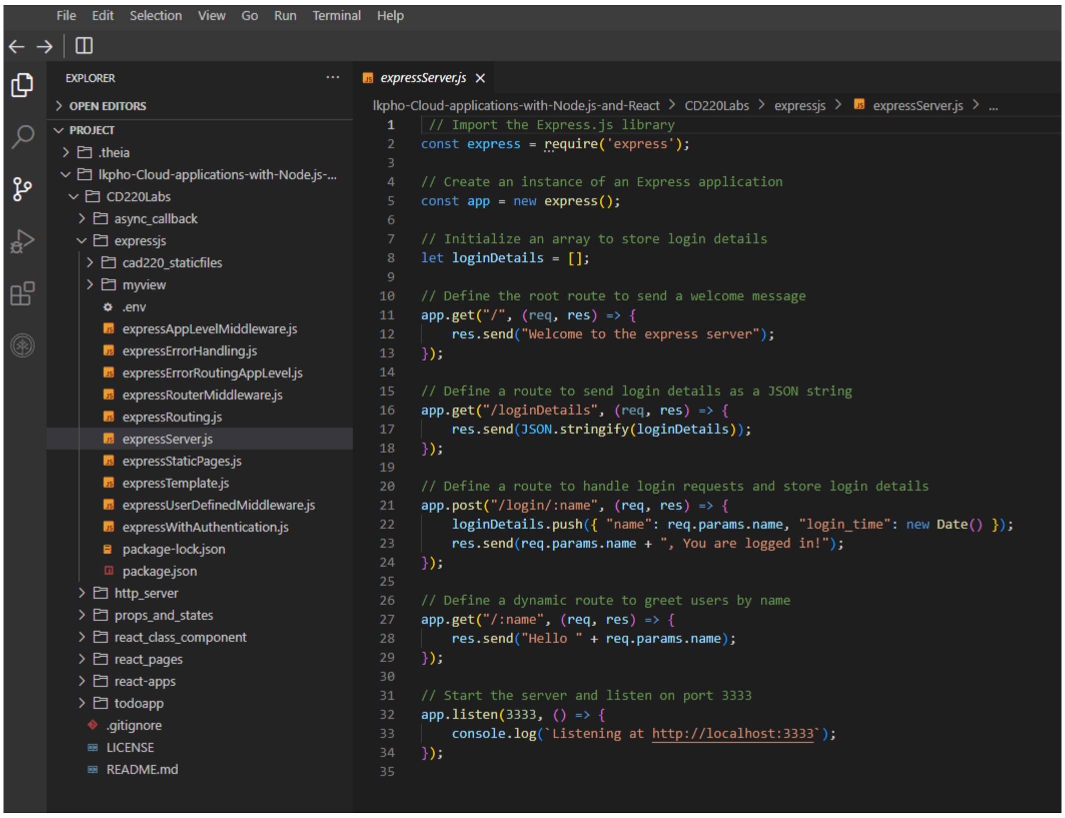
6. List the contents of this directory to see the artifacts for this lab.

```
ls
```

You must have a few exercise files that you will be running in the exercises.

View code, run server and connect to server through curl/browser

1. In the files explorer view expressServer.js. It would appear like this.



```
// Import the Express.js library
const express = require('express');

// Create an instance of an Express application
const app = new express();

// Initialize an array to store login details
let loginDetails = [];

// Define the root route to send a welcome message
app.get("/", (req, res) => {
    res.send("Welcome to the express server");
});

// Define a route to send login details as a JSON string
app.get("/loginDetails", (req, res) => {
    res.send(JSON.stringify(loginDetails));
});

// Define a route to handle login requests and store login details
app.post("/login/:name", (req, res) => {
    loginDetails.push({ "name": req.params.name, "login_time": new Date() });
    res.send(req.params.name + ", You are logged in!");
});

// Define a dynamic route to greet users by name
app.get("/:name", (req, res) => {
    res.send("Hello " + req.params.name);
});

// Start the server and listen on port 3333
app.listen(3333, () => {
    console.log(`Listening at http://localhost:3333`);
});
```

▼ You can also click here to view the code

```
 // Import the Express.js library
const express = require('express');
// Create an instance of an Express application
const app = new express();
// Initialize an array to store login details
let loginDetails = [];
// Define the root route to send a welcome message
app.get("/", (req, res) => {
    res.send("Welcome to the express server");
});
// Define a route to send login details as a JSON string
app.get("/loginDetails", (req, res) => {
    res.send(JSON.stringify(loginDetails));
});
// Define a route to handle login requests and store login details
app.post("/login/:name", (req, res) => {
    loginDetails.push({ "name": req.params.name, "login_time": new Date() });
    res.send(req.params.name + ", You are logged in!");
});
// Define a dynamic route to greet users by name
app.get("/:name", (req, res) => {
    res.send("Hello " + req.params.name);
});
// Start the server and listen on port 3333
app.listen(3333, () => {
    console.log(`Listening at http://localhost:3333`);
});
```

Here is an explanation of the code in it:

- `const express = require('express');` imports the Express.js library.
- `const app = new express();` creates an instance of an Express application.
- `let loginDetails = [];` initializes an array to store login details.
- `app.get("/", (req, res) => { res.send("Welcome to the express server"); });` defines the root route to send a welcome message.
- `app.get("/loginDetails", (req, res) => { res.send(JSON.stringify(loginDetails)); });` defines a route to send login details as a JSON string.
- `app.post("/login/:name", (req, res) => { loginDetails.push({ "name": req.params.name, "login_time": new Date() }); res.send(req.params.name + ", You are logged in!"); });` defines a route to handle login requests and store login details.
- `app.get("/:name", (req, res) => { res.send("Hello " + req.params.name); });` defines a dynamic route to greet users by name.
- `app.listen(3333, () => { console.log(Listening at http://localhost:3333); });` starts the server and listens on port 3333.

This is a simple express Server which listens at port 3333, with 4 end points.

```
/
/loginDetails
/login/:name - POST
/:name
```

2. In the terminal window, run the following command which will ensure that the express package is installed.

```
npm install --save express
```

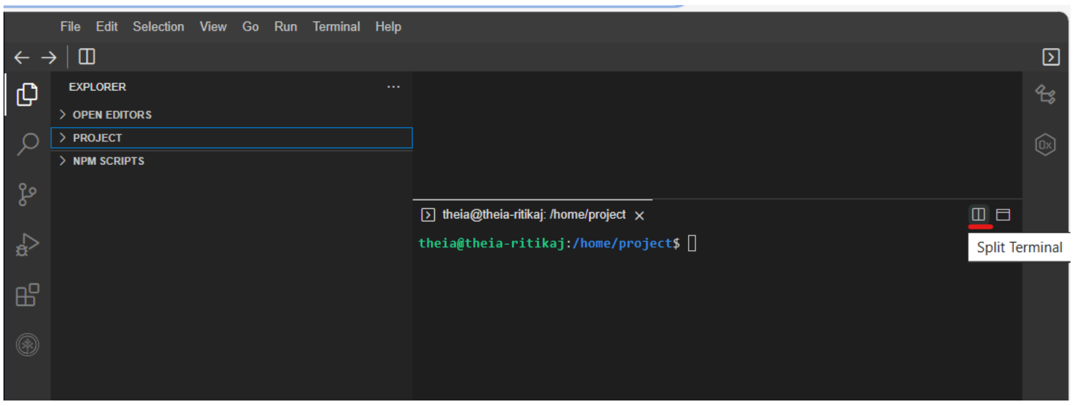
3. In the terminal window run the server with the following command.

```
node expressServer.js
```

You should see output similar to this.

```
Listening at http://localhost:3333
```

4. Click on “Split Terminal” to divide the terminal, as depicted in the image below.



```
theia@theia-ritikaj:/home/project$
```

5. In the second terminal window, use the `curl` command to ping the application.

```
curl localhost:3333
```

You should see output similar to this.

```
Welcome to the express server
```

This indicates that your app is up and running.

6. Try the other end points with the curl commands in the same terminal.

/login/:name

```
curl -X POST http://localhost:3333/login/Jason
```

You should see output similar to this.

```
Jason, You are logged in!
```

/:name

```
curl http://localhost:3333/Jason
```

You should see output similar to this.

```
Hello Jason
```

/loginDetails

```
curl http://localhost:3333/loginDetails
```

You should see output similar to this.

```
[{"name":"Jason","login_time":"2020-11-20T06:06:56.047Z"}]
```

7. To stop the server, go to the main command window and press Ctrl+c to stop the server.

Task 1 : Add your own end point

*Note - This is non-graded

Create a list with the names of the month. Add an end point in the code `/fetchMonth/:num` which will fetch a particular month from a list and return it to user. If the number is invalid, it should return appropriate error message.

▼ Click here, if you need help to do the task

```
# Define an array containing the names of the months
const months = ["January","February","March","April","May","June","July","August","September","October","November","December"];
# Define a route to fetch the month name based on a given number
app.get("/fetchMonth/:num", (req, res) => {
    # Parse the number from the request parameters
    let num = parseInt(req.params.num);
    # Check if the number is a valid month number
    if(num < 1 || num > 12) {
        # Send an error message if the number is not valid
        res.send("Not a valid month number");
    } else {
        # Send the corresponding month name if the number is valid
        res.send(months[num - 1]);
    }
});
```

Using Middleware

1. On the file explorer view the code expressAppLevelMiddleware.js

▶ You can click here to view the code

This server uses middleware for authentication. If the password is not pwd123 it will not allow the user to login. This server has just one end point and it takes password as query parameter.

2. Run the server.

```
node expressAppLevelMiddleware.js
```

You should see output which says `Listening at http://localhost:3333`

3. In the second terminal window, use the `curl` command to ping the application.

   ```
   curl localhost:3333/home
   ```

You should see output which say `This user cannot login`.

4. Execute curl command passing the password parameter

   ```
   curl http://localhost:3333/home?password=pwd123
   ```

You should see output which say `Hello World!`.

This is because the server has a middleware which filters each request to the server to see what the password is and allows to proceed only when the password is `pwd123`.

5. To stop the server, go to the main command window and press Ctrl+c to stop the server.

Express server with Authentication

In this exercise you will learn how to build in authentication layer in your express server inorder to make the server secure. You will be using the postman tool for this lab.

1. On the file explorer view the code expressWithAuthentication.js

▼ You can click here to view the code

```
 // Importing required modules: Express.js, JSON Web Token (JWT), and Express session
const express = require('express');
const jwt = require('jsonwebtoken');
const session = require('express-session');
let users = [];
// Function to check if the user exists
const doesExist = (username) => {
  let userswithsamename = users.filter((user) => {
    return user.username === username;
  });
  return userswithsamename.length > 0;
};
// Function to check if the user is authenticated
const authenticatedUser = (username, password) => {
  let validusers = users.filter((user) => {
    return user.username === username && user.password === password;
  });
  return validusers.length > 0;
};
const app = express();
app.use(express.json()); // Middleware to parse JSON request bodies
app.use(session({ secret: "fingerpint" })); // Middleware to handle sessions
// Middleware to authenticate users using JWT
app.use("/auth", function auth(req, res, next) {
  if (req.session.authorization) { // Get the authorization object stored in the session
    token = req.session.authorization['accessToken']; // Retrieve the token from authorization object
    jwt.verify(token, "access", (err, user) => { // Use JWT to verify token
      if (!err) {
        req.user = user;
        next();
      } else {
        return res.status(403).json({ message: "User not authenticated" });
      }
    });
  } else {
    return res.status(403).json({ message: "User not logged in" });
  }
});
// Route to handle user login
app.post("/login", (req, res) => {
  const username = req.body.username;
  const password = req.body.password;
  if (!username || !password) {
    return res.status(404).json({ message: "Error logging in" });
  }
  if (authenticatedUser(username, password)) {
    let accessToken = jwt.sign({
      data: password
    }, 'access', { expiresIn: 60 * 60 });
    req.session.authorization = {
      accessToken, username
    };
    return res.status(200).send("User successfully logged in");
  } else {
    return res.status(208).json({ message: "Invalid Login. Check username and password" });
  }
});
// Route to handle user registration
app.post("/register", (req, res) => {
  const username = req.body.username;
  const password = req.body.password;
  if (username && password) {
    if (!doesExist(username)) {
      users.push({ "username": username, "password": password });
      return res.status(200).json({ message: "User successfully registered. Now you can login" });
    } else {
      return res.status(404).json({ message: "User already exists!" });
    }
  }
  return res.status(404).json({ message: "Unable to register user." });
});
// Main endpoint to be accessed by authenticated users
app.get("/auth/get_message", (req, res) => {
  return res.status(200).json({ message: "Hello, You are an authenticated user. Congratulations!" });
});
const PORT = 5000; // Define the port number
app.listen(PORT, () => console.log("Server is running")); // Start the server and listen on the specified port
```

Explanation:

- `Modules and Middleware:` Import necessary modules like Express, JWT, and Express session. Use middleware like express.json() for parsing JSON bodies and session() for managing sessions.
- `User Functions:` Define functions to check if a user exists (doesExist) and if a user is authenticated (authenticatedUser).
- `Routes:` Create routes for user authentication (/auth), login (/login), registration (/register), and a main endpoint for authenticated users (/auth/get_message).
- `Middleware for Authentication:` Use a custom middleware (auth) to authenticate users using JWT and session management.
- `Server Setup:` Set up the Express app, define the port (PORT), and start the server to listen on the specified port.

2. To run this application, as you may notice we use two new packages that you have not used before. Run the following command to install `jsonwebtoken` and `express-session`.

```
npm install --save express-session jsonwebtoken
```

3. In this code you have one end-point, **/auth/get_message** which is allowed only for authenticated users. Run the server and try to access the end point, firstly. It should throw an error.

```
node expressWithAuthentication.js
```

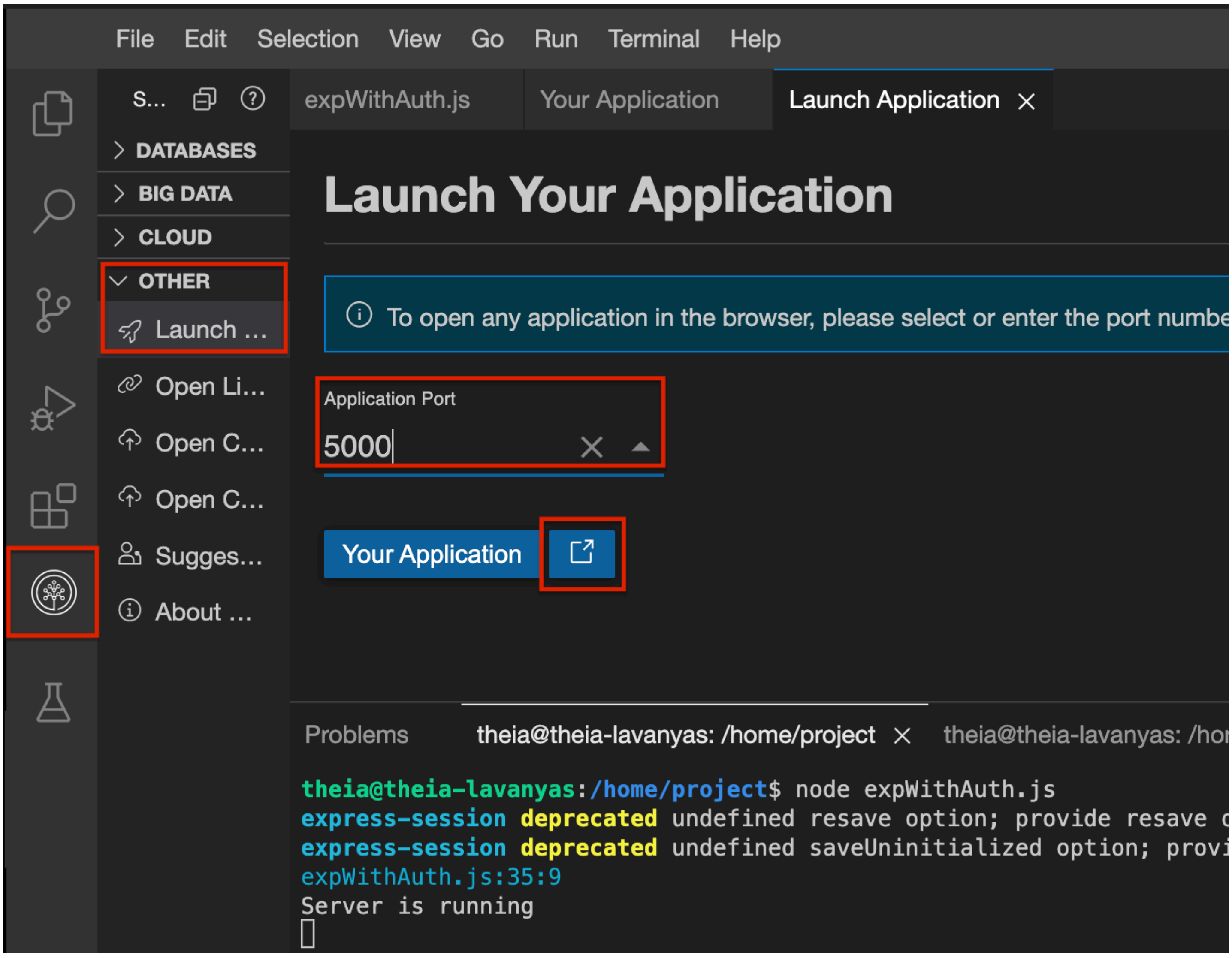
You should see an output which says `Listening at http://localhost:5000`.

4. In the second terminal window, use the following `curl` command.

```
curl localhost:5000/auth/get_message
```

You should see an output which says `{"message":"User not logged in"}`.

5. You have to register a user with a username and password and login with that username and password to be able to access the end-point. Click on the **Skills Network Toolbox** icon, choose **Others** and click **Launch Application**. Enter the port number `5000` and open the URL. It will open in a new browser window. Copy the url. Go to https://www.postman.com/. You may have to sign in if this is your first time using postman.



Launch Your Application

To open any application in the browser, please select or enter the port numbe

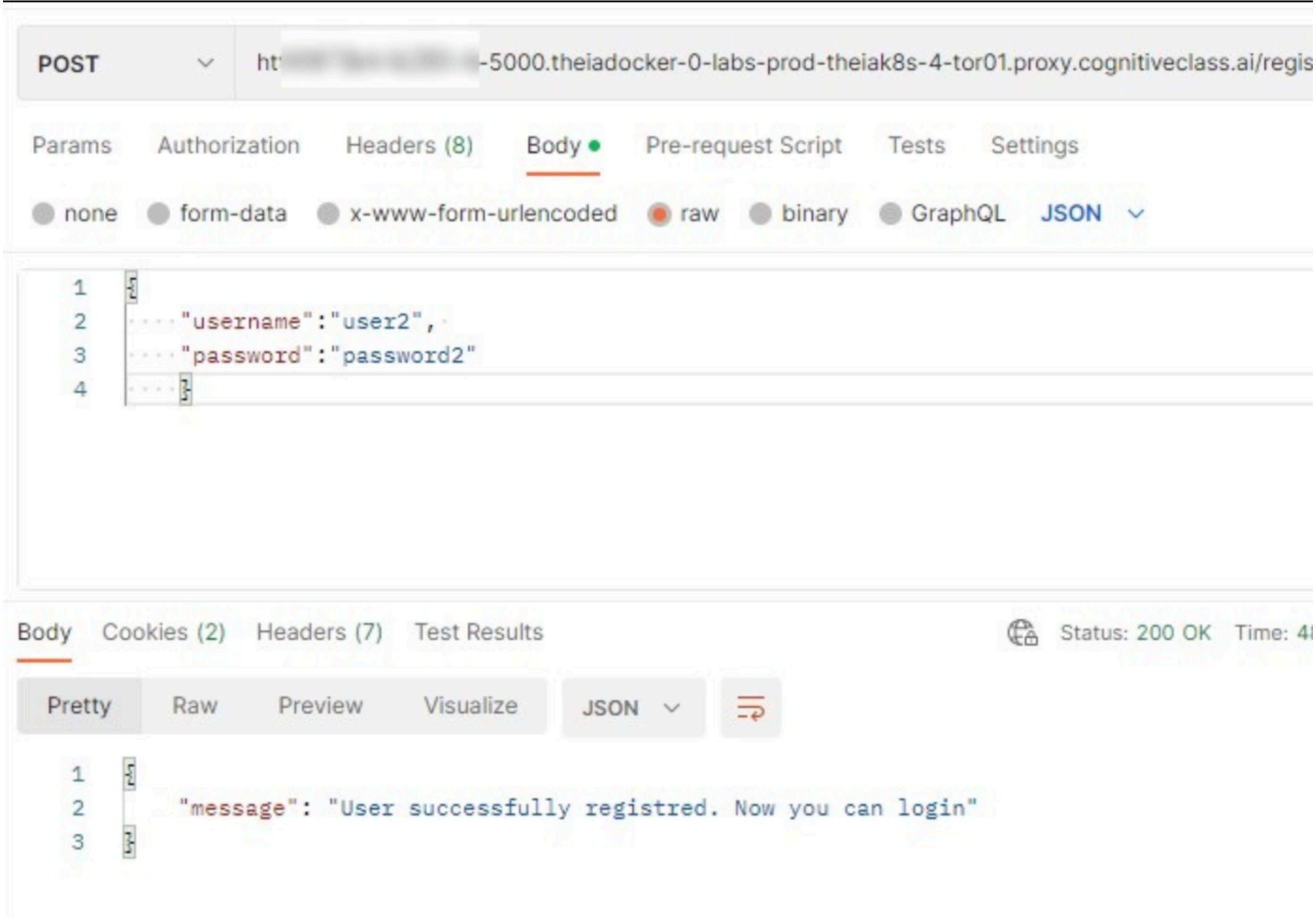
Application Port: 5000

```
theia@theia-lavanyas:/home/project$ node expWithAuth.js
express-session deprecated undefined resave option; provide resave c
express-session deprecated undefined saveUninitialized option; provi
expWithAuth.js:35:9
Server is running
```

6. In the postman, enter the URL that you copied and suffix it with **/register**.
7. Select 'Body' >> 'raw' >> 'JSON' and pass the parameters.

```
{"username":"user2", "password":"password2"}
```

Note: "user2" & "password2" are used for reference. You can use any username & password.



POST ht[redacted]-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/regis

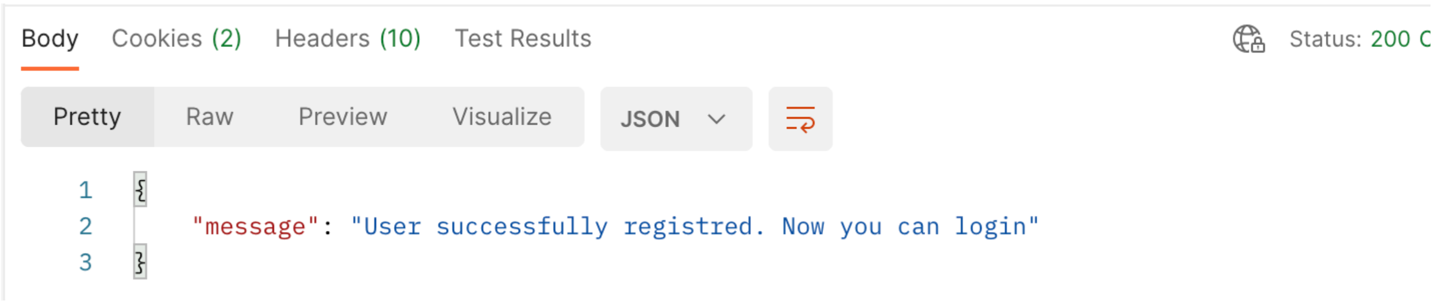
Body: raw, JSON

```
{
    "username":"user2",
    "password":"password2"
    }
```

Status: 200 OK Time: 4

```
{
    "message": "User successfully registred. Now you can login"
}
```

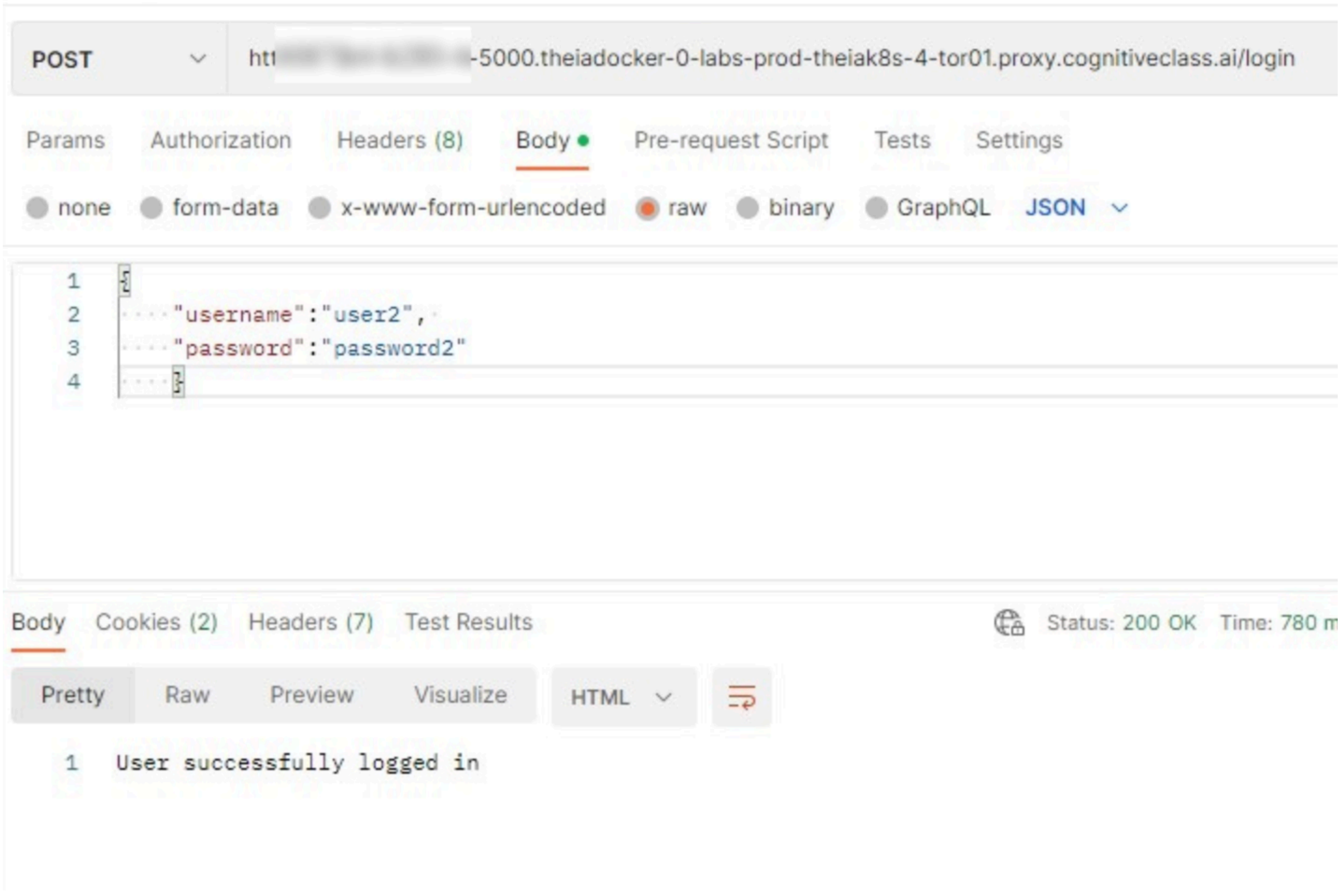
8. Set the query type to **POST** and click **send**. You will see a confirmation saying that the user has been registered.



Status: 200 C

```
{
    "message": "User successfully registred. Now you can login"
}
```

9. Use the same copied url now to login with the suffix **/login**. The parameters to be passed remain the same. This is also a **POST** request. Click **send**. You will see a message confirming that you are logged in, as seen below.



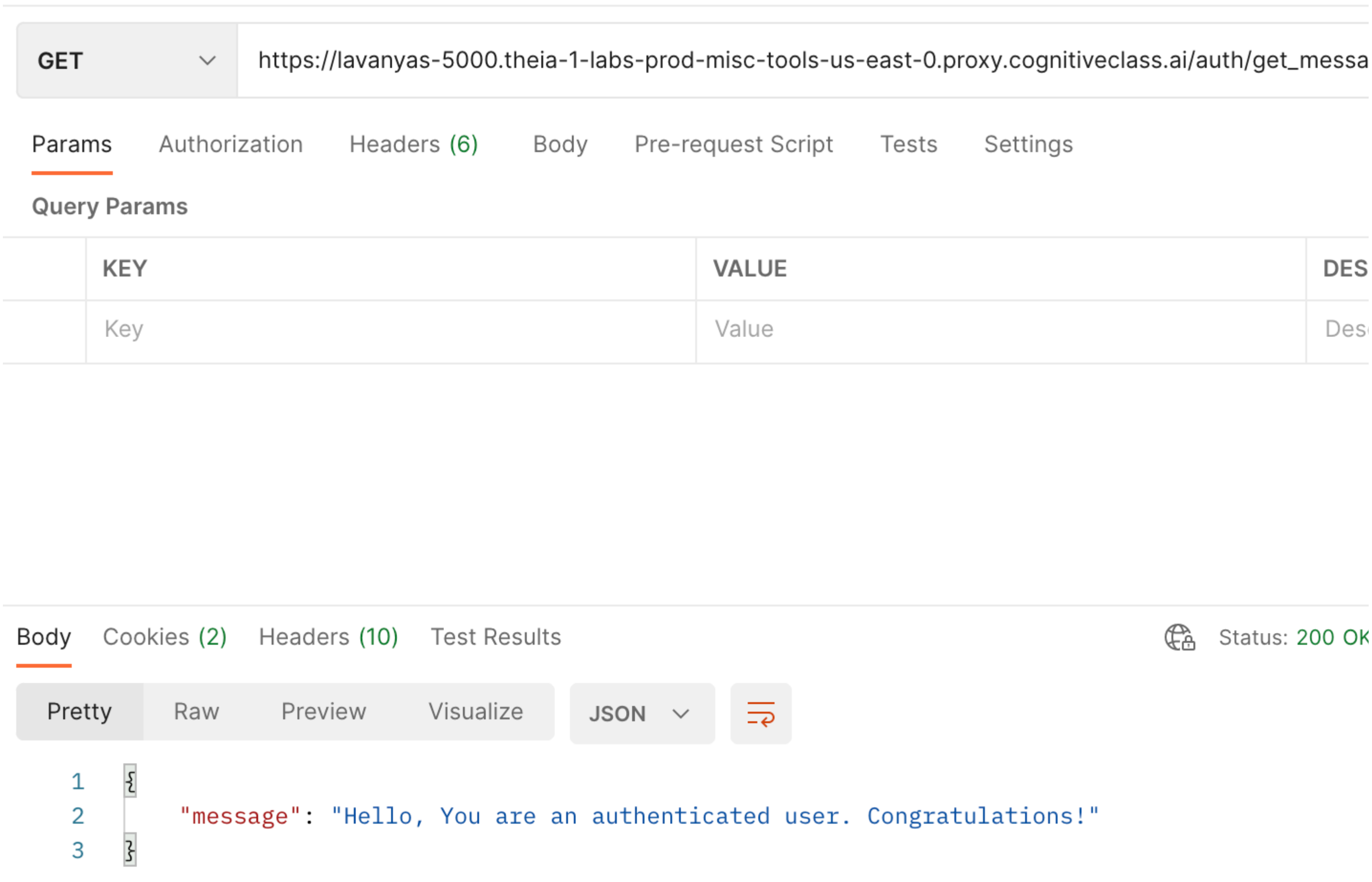
htt[redacted]-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/login

```
{
    "username":"user2",
    "password":"password2"
}
```

Status: 200 OK Time: 780 m

```
User successfully logged in
```

10. Now you can access the **/auth/get_message** end point and it will return the message.



https://lavanyas-5000.theia-1-labs-prod-misc-tools-us-east-0.proxy.cognitiveclass.ai/auth/get_messa

Query Params

KEY	VALUE	DES
Key	Value	Des

Status: 200 OK

```
{
    "message": "Hello, You are an authenticated user. Congratulations!"
}
```

Routing

As the names suggests, you can route the API requests to different handlers. Usualy the handlers are logically divided on the basis of the objects they deal with.

1. On the file explorer view the code expressRouting.js

▼ You can click here to view the code

```
 // Import the Express.js library
const express = require('express');
// Create an instance of an Express application
const app = new express();
// Create routers for users and items
let userRouter = express.Router();
let itemRouter = express.Router();
// Middleware for user router to log query time
userRouter.use(function (req, res, next) {
    console.log('User query Time:', Date());
    next();
});
// Route to handle user requests with ID parameter
userRouter.get('/:id', function (req, res, next) {
    res.send("User " + req.params.id + " last successful login " + Date());
});
// Middleware for item router to log query time
itemRouter.use(function (req, res, next) {
    console.log('Item query Time:', Date());
    next();
});
// Route to handle item requests with ID parameter
itemRouter.get('/:id', function (req, res, next) {
    res.send("Item " + req.params.id + " last enquiry " + Date());
});
// Mount the routers to specific paths
app.use('/user', userRouter);
app.use('/item', itemRouter);
// Start the server and listen on port 3333
app.listen(3333, () => {
    console.log(`Listening at http://localhost:3333`);
});
```

Explanation:

- `Express and Routers:` Import Express and create an instance of the Express application. Create routers for handling user and item routes separately using express.Router().
- `Middleware for Routers:` Add middleware to the user and item routers to log query times using console.log('User query Time:', Date()); and console.log('Item query Time:', Date());.
- `Route Handlers:` Define route handlers for user and item routes with ID parameters. These handlers send responses with formatted messages including the ID and the current date.
- `Mount Routers:` Mount the user router to the /user path and the item router to the /item path using app.use('/user', userRouter); and app.use('/item', itemRouter);.
- `Server Setup:` Start the server and listen on port 3333 using app.listen(3333, () => { console.log(Listening at http://localhost:3333`); });`.

This server branches and the requests based on the end points and uses routers to handle them. All the /user endpoints are handled by userRouter and /item endpoints are handled by itemRouter.

```
 /user/:id
 /item/:id
```

```
node expressRouting.js
```

You should see output which says `Listening at http://localhost:3333`.

2. In the second terminal window, use the following `curl` command.

```
curl localhost:3333/item/1
```

You should see output which says `Item 1 last enquiry Fri Nov 20 2020 15:17:46 GMT+0530 (India Standard Time)`.

3. In the second terminal window, use the following `curl` command.

```
curl localhost:3333/user/1
```

You should see output which says `User 1 last successful login Fri Nov 20 2020 15:19:52 GMT+0530 (India Standard Time)`.

4. To stop the server, go to the main command window and press Ctrl+c to stop the server.

Rendering Static Pages

1. On the file explorer view the code expressStaticPages.js

▼ You can click here to view the code in expressStaticPages.js

```
 // Import the Express.js library
const express = require('express');
// Create an instance of an Express application
```

```
const app = new express();
// Serve static files from the 'cad220_staticfiles' directory
app.use(express.static('cad220_staticfiles'));
// Start the server and listen on port 3333
app.listen(3333, () => {
    console.log(`Listening at http://localhost:3333`);
});
```

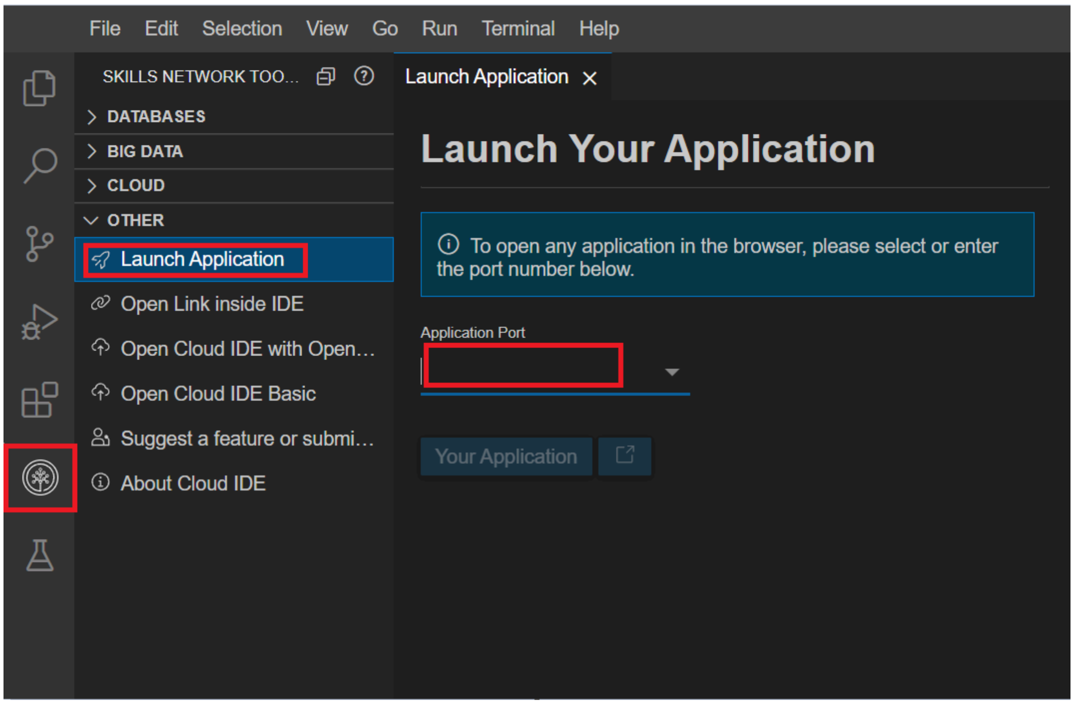
This server, as you see doesn't have any end points. But it has a middleware which sets the directory for static files. So any file that is in the `cad220_staticfiles` directory will be accessible. The folder contains the HTML page that would be rendered.

2. Run the server using the following command.

```
node expressStaticPages.js
```

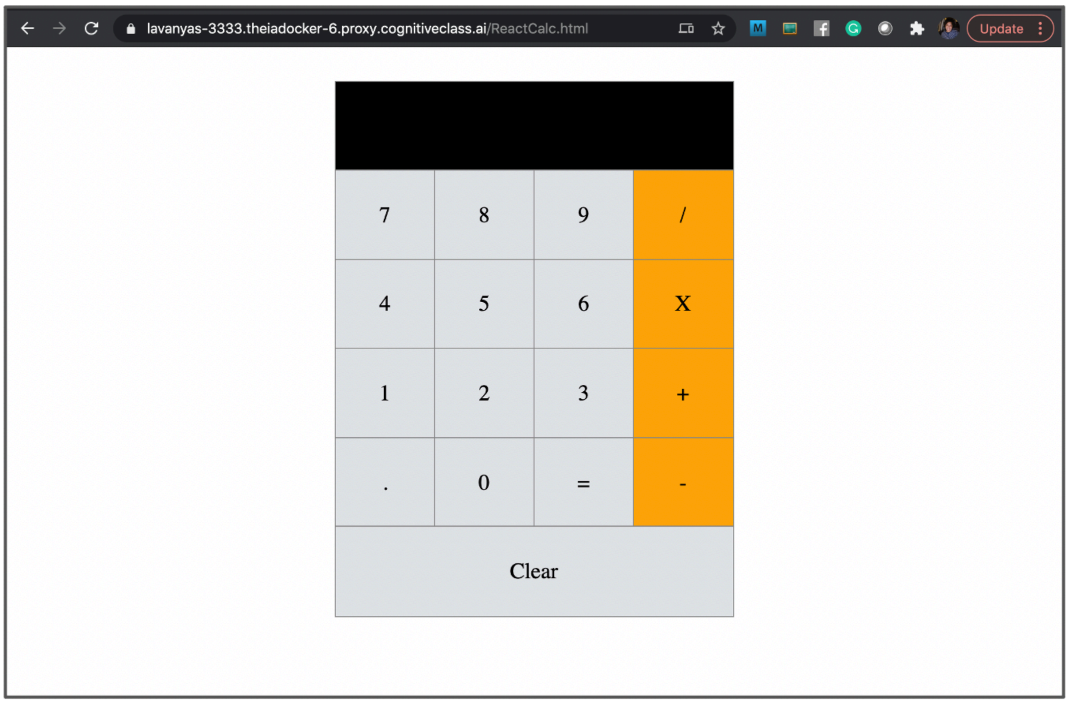
You should see output which says `Listening at http://localhost:3333`.

3. Click on the **Skills Network** button on the left, it will open the “Skills Network Toolbox”. Then click the **Other** then **Launch Application**. From there you should be able to enter the port 3333 and launch.



4. Add `/ReactCalc.html` to the url in the address bar.

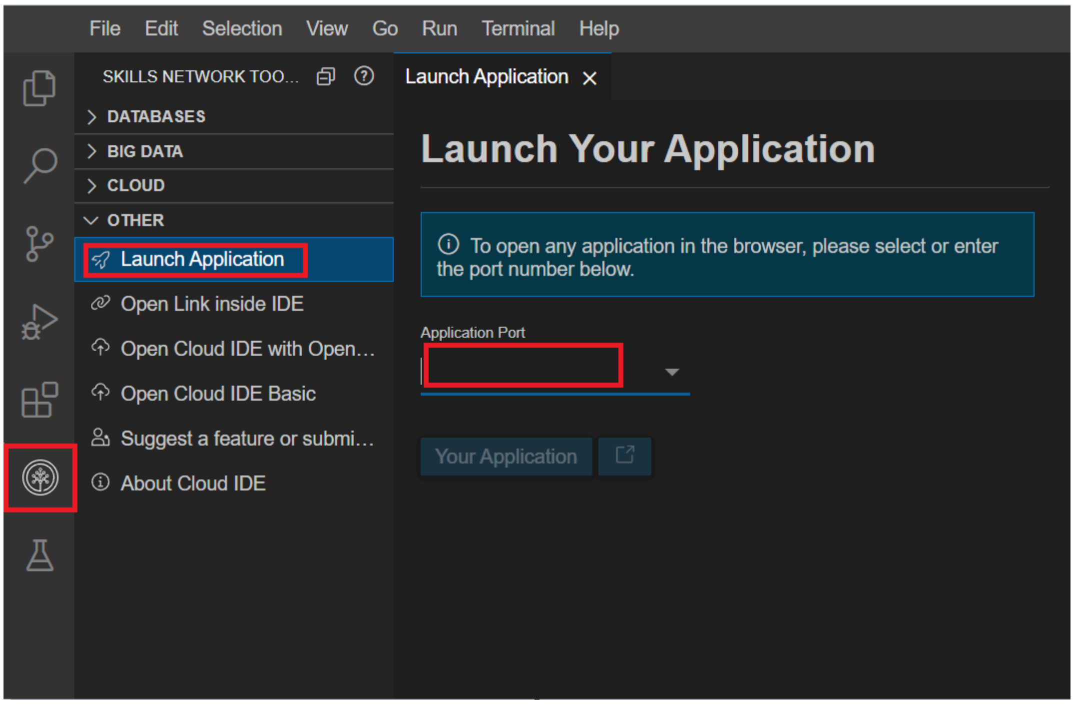
You will see th page rendered as below.


lavanyas-3333.theiadocker-6.proxy.cognitiveclass.ai/ReactCalc.html
Update
7
8
9
/
4
5
6
X
1
2
3
+
.
0
=
-
Clear

Task: Add your own static file

*Note - This is non-graded

Add a static file, an image or html file, to the directory `cad220_staticfiles` and try to access it through the `/<filename>` on the browser launched by clicking on the **Skills Network** button on the left, it will open the "Skills Network Toolbox". Then click the **Other** then **Launch Application**. From there you should be able to enter the port and launch.



Launch Your Application

To open any application in the browser, please select or enter the port number below.

Create an express server from scratch with nodemon

1. Go to the `/home/project` directory.

```
cd /home/project
```

2. Create a directory named `myexpressapp` and change to that directory

```
mkdir myexpressapp
cd myexpressapp
```

3. Now run `npm init`.

This will init the api directory to serve as a web application. Follow the prompts on the screen to complete the intialization.

- The package name by default is the name of the current folder (myexpressapp in this case). You can specify a different name if you want.
- Next it asks you for the version you want to set. The default is 1.0.0.
- It then prompts for a description where you can give a short description of what the api intends to do. You can leave it blank.
- Next we specify the entry point into the API, which by default is index.js.
- When it prompts for the author, you can give your name or leave it blank.
- License by default is ISC (Internet Systems Consortium) which means it is a permissive license that lets people do anything with your code with proper attribution and without warranty.
- It will generate the contents for your package.json, a file that keeps track of all the packages your server application needs, and asks you to check if the details are OK. Once you confirm, the details are all written on to the package.json.

4. Now run the following command to install express.

```
npm install express --save
```

`--save` option ensures that the package.json is updated.

5. Now run `touch index.js` command. You will see that this file is created in the file explorer. You can use the IDE to write the code you want inside from what you have learnt in the previous exercises.

▼ Sample code has been given here

```
 // Import the Express.js library
const express = require('express');
// Create an instance of an Express application
const app = new express();
// Define the port number
const port = 8080;
// Route to handle requests to the root path "/"
app.get("/", (req, res) => {
    return res.send("Hello World!");
});
// Start the server and listen on the specified port
let server = app.listen(port, () => {
    console.log("Listening at http://localhost:" + port);
});
```

6. Make changes in `package.json` to start the server with npm start. Include `"start"` under scripts.

```
{
  "name": "myexpressapp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^X.X.X"
  }
}
```

7. From the command prompt, you can now run `npm start` to run the server.

8. Now you can include other end points or make changes to the server as needed. But it can be very frustrating to stop and start the server everytime you make changes. There is a package that comes handy in this case. The package is called nodemon. Every time you make changes in the server API, it will automatically restart the server. Let's install that in the same directory where we created our index.js. We will install and store it as a dev dependency with the –save-dev option because we want to use this only when we are running the server locally in our development environment.

```
npm install --save-dev nodemon
```

9. Once nodemon is installed, we will make changes to package.json to make use of this and re-start the script when there are changes. We will include the `"start" : "nodemon index.js"` in the scripts section of our package.json. With the changes, the package.json will look like this.

```
{
  "name": "myexpressapp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start" : "nodemon index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^X.X.X"
  },
  "devDependencies": {
    "nodemon": "^X.X.X"
  }
}
```

At the command prompt now run `npm start` to start the web server.
Now make some change or add another endpoint returns and see if the server is restarting and changes are reflecting without having to explicitly restart. Magic!

Congratulations! You have completed the lab for express JS.

Summary

Now that you have have learnt how create and run an express server and how to use middleware, templates and routing, we will go further learn how to create clients to connect to the servers from.

Author(s)

Lavanya

© IBM Corporation. All rights reserved.