Hackathon application

TODO:

UML for front end u/x:

**Overview Of the application**

\* Full stack web application

\*Node, Express, Mongo DB

\*User Authentication with Passport JS

\*Admin page for registered users

\*Sample store where users can save items for later purchases (i.e. a “cart”)

**Server-Side Web Application**

* Traditional CRUD application
* Create, Read, Update, Delete
* Additional: forget password flow
* Automated email sending (Mailgun)

**Setting up the environment**

**Install Chocolatey for Individual Use:**

1. First, ensure you are using an [***administrative shell***](https://www.howtogeek.com/194041/how-to-open-the-command-prompt-as-administrator-in-windows-8.1/) - you can also install as a non-admin; check out [Non-Administrative Installation](https://docs.chocolatey.org/en-us/choco/setup#non-administrative-install).

Install with powershell.exe.

**NOTE**

Now run the following command in PowerShell:

Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

choco install express

choco install mongodb

choco install nodejs

choco install python

Addition Requirements: cd into the project to install

npm install -g nodemon

npm install hjs –save

npm i bootstrap@5.3.0-alpha2

**Initializing the application: NODEJS**

1. Cd into the folder of the repository
2. Run npm init
3. Check the package.json

**Initializing the application: Express**

1. Cd into the folder of the repository
2. rpm install express --save
3. Check the for node\_modules

**Configure server.js file**

1. Open the repository in Visual Studio
2. Create a name named server.js
3. Use the following code in file server.js
4. const express = require('express'); /\* import express \*/
5. const app = express(); /\* instanciate the express app \*/
6. app.get('/', (req, res,next) =>
7. {
8. res.send('this is the home route');
9. })
10. app.listen(5000);
11. console.log('App running on http://localhost:5000');
12. In package.json, change line 5 to
13. "main": "server.js",
14. cd Into the repository and run nodemon
15. Check if the application is ‘App running on http://localhost:’
    1. To turn off the server, use ctrl+c

**Create a directory to store the routes (pages)**

1. Create a new directory by running the command mkdir routes
2. Cd in routes and create a file named home.js and cd back to the previous directory with the command cd ..
3. Add to the following to home.js
4. const express = require('express');
5. const router = express.Router(); /\*handles routing functanlity, get, post, etc \*/
6. //        path, request, response, and next
7. router.get('/',(req    , res     ,    next)   => {
8. res.send('This is form the home router!')
9. })
10. // export the file  into server.js and use the router instead of hard coding it
11. module.exports = router;
12. Remove the following code from server.js
13. app.get('/', (req, res,next) =>
14. {
15. res.send('this is the home route');
16. })

5. The server.js file should look like this

const express = require('express') /\* import express \*/

const home = require('./routes/home');

const app = express() /\* instanciate the express app \*/

app.use('/', home) /\* this route is responsable for all gets posts  deletes etc\*/

app.listen(5000);

console.log('App running on http://localhost:5000');

6. Re-run the server and check for This is form the home router!

**App Configuration: Hogan templating engine**

1. Add the following code to server.js to Import the path module
2. const path = require(); /\*Default module for node js \*/

2. Create a views directory in the project, cd into views

3. Create a file named home.hjs template (basically a html document)

4. Add the following code to server.js

app.set('views', path.join(\_\_dirname,'views'))/\*Set the app to use the views dir\*/

app.set('view engine','hjs') /\*Set the app to use the hjs file\*/

* 1. The server.js file should like this now

1. const express = require('express') /\* import express \*/
2. const path = require(‘path’); /\*Default module for node js \*/
3. const home = require('./routes/home');
5. const app = express() /\* instanciate the express app \*/
6. app.set('views', path.join(\_\_dirname,'views'))/\*Set the app to use the views dir\*/
7. app.set('view engine','hjs') /\*Set the app to use the hjs file\*/
8. app.use(express.static(path.join(\_\_dirname, 'public')))/\*Find all static assets inside the public dir\*/ /\*We will add our code with the other instance of the app.use function.\*/
9. app.use('/', home) /\* this route is responsable for all gets posts  deletes etc\*/
10. app.listen(5000);
11. console.log('App running on http://localhost:5000');

5. modify the following code to home.js

//        path, request, response, and next

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

    res.render('home', null)    /\* automatically  look up the views directory

    for the home.hjs template\*/

})

**App Configuration: home.hjs**

1. Open the home.hjs
2. Add the following template code
3. <html>
4. <head></head>
5. <body>
6. <h1>
7. This is a test of server side rendering with hjs
8. </h1>
9. </body>
10. </html>

**Registration Form**

1. Goto <https://www.bootstrapcdn.com/> and find the latest CSS link ex. <https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css>
2. Add bootstrap to the home.hjs file
   1. The home.hjs file should look like the code snippet below
3. <html>
4. <head>
5. <link rel="stylesheet" type="text/css" href="https://cdn.jsdelivr.net/
6. npm/bootstrap@5.2.3/dist/css/bootstrap.min.css">
7. </head>
8. <body>
9. <h1>
10. This is a test of server side rendering with hjs
11. </h1>
12. </body>

</html>

1. Add a navigation bar. for Ex navigation bar <https://getbootstrap.com/docs/4.0/components/navbar/>
2. The implementation should look like the following

a. <html>

    <head>

        <link rel="stylesheet" type="text/css" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css">

    </head>

    <body>

            <nav class="navbar navbar-expand-lg navbar-light bg-light">

    <a class="navbar-brand" href="#">Navbar</a>

    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">

        <span class="navbar-toggler-icon"></span>

    </button>

        <div class="collapse navbar-collapse" id="navbarNavAltMarkup">

        <div class="navbar-nav">

        <a class="nav-item nav-link active" href="#">Home <span class="sr-only">(current)</span></a>

        <a class="nav-item nav-link" href="#">Features</a>

        <a class="nav-item nav-link" href="#">Pricing</a>

        <a class="nav-item nav-link disabled" href="#">Disabled</a>

        </div>

    </div>

</nav>

        <h1>

            This is a test of server side rendering with hjs

        </h1>

    </body>

</html>

1. Move the <h1> tag into a new div container class, this allows us to use the bootstrap column and grid structure
   1. The code should look like the following
2. </nav>
3. <div class="container">
4. <h1>
5. This is a test of server side rendering with hjs
6. </h1>
7. </div>
9. </body>
10. </html>

6.Create the user registration form

a. The code below adds 3 bootstrap forms for user registration inputs

<div class="container">

    <h1>

        This is a test of server side rendering with hjs</h1>

        <form action="/register" method="post">

            <input type="text" name = "email" placeholder="Email" /><br/>

            <input type="password" name = "password" placeholder="Password" /><br/>

            <input type="submit" value="Sign Up" />

        </form>

</div>

    </body>

</html>

7. Create the registration endpoint for the POST

a. Create a new file named register.js in the route folder

b. Copy and paste the code from the home.js file into the register.js file.

c. modify the router method call to post and record the the data payload in the response (res).json.

The file register.js should look as follows:

const express = require('express');

const router = express.Router(); /\*handles routing functanlity, get, post, etc \*/

//        path, request, response, and next

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

       res.json({

        data: req.body

       })

})

// export the file  into server.js and use the router instead of hard coding it

module.exports = router;

7. Add the registration route to the server.js file

a. The code should look like the following

const register = require('./routes/register')

b. Add a call to the use method to set the register path

app.use('/', home) /\* this route is responsable for all gets posts  deletes etc\*/

app.use('/register', register)

8. Add the parse form data middleware in the server.js file

a. The server.js should appear as follows:

const express = require('express') /\* import express \*/

const path = require('path') /\*Default module for node js \*/

const home = require('./routes/home')

const register = require('./routes/register')

const app = express() /\* instanciate the express app \*/

app.set('views', path.join(\_\_dirname,'views'))/\*Set the app to use the views dir\*/

app.set('view engine','hjs') /\*Set the app to use the hjs file\*/

app.use(express.json())/\*parse are form as json\*/

app.use(express.urlencoded({extended: false})) /\*enables receiving form data\*/

app.use(express.static(path.join(\_\_dirname, 'public')))/\*Find all static assets inside the public dir\*/ /\*We will add our code with the other instance of the app.use function.\*/

app.use('/', home) /\* this route is responsable for all gets posts  deletes etc\*/

app.use('/register', register)

app.listen(5000);

console.log('App running on http://localhost:5000');

**Setting up The Database**