Session

Today's main agenda is to host your NodeJS application. Whatever we have been developing is inside our system and external world has no access to it. we were developing all the applications and were able to access it in our browser. Today's agenda is how to host so that our web application is applicable to the entire world through the internet.

We will use something called as **Heroku** to get this done. Before hosting, we will add few features to it and then we will upload it. First, we will add a link saying SignUp. At signUp page, we will Sign UP and at login page we will be able to Login. Once the signUp is done, we will re-direct to the login page. In the home.hbs page, we have created a login, so we will be posting this route to login. Now when they click on the Sign Up form, it needs to directed to the Sign up page.

```
<div class="container">
  <h1 class="text-center display-5">Login</h1>
  <div class="d-flex flex-column justify-content-center align-items-cent
er">
    <form action="/login" method="POST">
      <div class="mb-3">
        <label for="emailId" class="form-label">Email address</label>
        <input type="email" name="email" value="johnDoe@jasper.info" cla
ss="form-control" id="emailId">
      </div>
      <div class="mb-3">
          <label for="password" class="form-label">Password</label>
          <input type="password" name="password" value="123abc" class="f
orm-control" id="password">
      </div>
      <button type="submit" class="btn btn-primary">Login</button>
    </form>
    <a href="/signUp">Don't have an account? Click Here to signUp</a>
  </div>

</div>
```

We will create a signUp.hbs file and provide route as **/signUp**. We will add two input options of name, email and password. We will add an <a> tag in the end for user's who have already have account.

```
<div class="container">
```

```
  <h1 class="text-center display-5">SignUp</h1>
  <div class="d-flex flex-column justify-content-center align-items-cent
er">
    <form action="/signUp" method="POST">
      <div class="mb-3">
        <label for="name" class="form-label">Name</label>
        <input type="text" name="name" class="form-control" id="name">
      </div>
      <div class="mb-3">
        <label for="emailId" class="form-label">Email address</label>
        <input type="email" name="email" class="form-control" id="emailI
d">
      </div>
      <div class="mb-3">
        <label for="password" class="form-label">Password</label>
        <input type="password" name="password" class="form-control" id="
password">
      </div>
      <button type="submit" class="btn btn-primary">Sign Up</button>
    </form>
    <a href="/">Already have an account? Click Here to Login</a>
  </div>

</div>
```

We will add one more field for name as well. Since signUp is not being served through any route right now, we have to create a route for signUp. The following are the routes we are going to use for today.

```
app.get('/', (req, res) => {})

app.get('/signUp', (req, res) => {})

app.post('/signUp', (req, res) => {})

app.get('/profile', (req, res) => {})

app.post('/login', (req, res) => {})

app.listen(3000, () => console.log('Server Started'))
```

The data is getting through the server. We need to create an object and push it to the user array. We will add name, email and password to the signUp route (POST). The logic we can take the email and password from post route itself.

To see if everything is working fine, we will do,

```
app.post('/signUp', (req, res => {
    const {name, email, password} = req.body

    res.send(`${ name } ${ email } ${ password }`)
})
```

We are getting the details on the frontend.

```
app.post('/signUp', (req, res => {
    const {name, email, password} = req.body

    res.send(`${ name } ${ email } ${ password }`)

    const newUser = {
        name: name,
        email: email,
        password: password
    }
})
```

We have to push it to the new user, so to do that

```
app.post('/signUp', (req, res => {
    const {name, email, password} = req.body

    res.send(`${ name } ${ email } ${ password }`)

    const newUser = {
        name: name,
        email: email,
        password: password
    }

    user.push(newUser)

    req.session.isLoggedIn = true
    req.session.user = newUser

    res.redirect('/profile')
})
```

Now when we hit on submit with all the inputs filled, everything works perfectly and we are getting redirected to the profile page. Even in the memory store session has been created for the user.

Since we have logged-in the server remembers it and is allowing us to go to profile page. So, let us check this in another browser. We will not be able to login as session is not maintained in the new browser.

We will not touch the server. We will restart, delete the cookie and signup again. Now the user has been added to the particular array. In the edge browser the session is not available. So, once we signup the it will create an account and we can login with it.

The next step is to add one logout button and then we'll upload or hoist it.

As we go along it keeps adding the details to the server for now. These details can be appended into a file as well. Now, the user has been created, in this profile, we are directly getting redirected to the profile page. Even if we want to go to login page and then go to profile page, server is remembering it. So, we need to test this in another browser. So, in another browser, we cannot go to profile page directly without logging in first.

Since the server has be reset, let us login again and the data is being added to the terminal. We can add a logout option as well and then we'll upload it or host it. We can improve the styling as well. We will go to profile.hbs and will add,

In the server the details would be present even though the cookie is lost. We will create another route. We will not put GET route, or else with some URL it can get logged out. So, we will use POST method.

```
app.post('/logout', (req, res) => {
    // it will destroy the session for the user
    req.session.destroy(() => {
        res.redirect('/')
    })

}})
```

We can even have an error checker, to see if the logout was successful. If we click on logout, it will still not work, because we have not yet connected this button to a function in the profile.hbs file.

```
<div class="container">
  <h1 class="display-4"> Name: {{name}}</h1>
  <h2 class="lead">Email: {{email}}</h2>
  <div>
    <form action='/logout' method="POST">
      <button>Log Out</button>
    </form>
  </div>
</div>
```

Now we are able to logout and once we have logged out, we cannot go back to the profile page by changing the URL.

We will not use GET for better practice, because if someone puts a URL in the address bar it would get logged out.

Although we can set the cookie from the server, but there is not direct of way of deleting the cookie from the server. One way is we can say,

`req.session.cookie.maxAge = 0` and it will delete the cookie.

Second way is by using;

`res.clearCookie()` ➔ internally it would make the maxAge or time = 0

```
app.post('/logout', (req, res) => {
    // it will destroy the session for the user

    req.session.destroy(() => {

        res.clearCookie('connect.sid')
        res.redirect('/')
    })

})
```

Hoisting ➔ which is publicly accessible.

There are many platforms where we can hoist our websites and one of them is Heroku. We can create an account in Heroku.

Heroku is something we call as Paas. It is a system in which we deploy our source code and Heroku will run your code and it will make it available to the world wide web, so that anyone can access your code. Right now, we just want to send the code to Heroku platform and then Heroku will start serving it in their server. Heroku actually connects to git repository. It internally creates a git repository and then it will clone from it, install all the dependency and will then will run it. we can do this in GitHub, but only for frontend. We cannot create our own server in GitHub pages.

It needs a GitHub repository to work properly. We have created one GitHub repo for day4 project. We'll just commit it to the GitHub.

We will do git init so it will initialize a git.

The computer has created a server for you.

Once we create the application, we have some over view, resources and application. We have something called as Heroku Login. We will use Heroku CLI. We have to download and install it. once that is done, we'll be able to type in commands of Heroku or else, we can just type in

```
npm i -g Heroku
```

hreoku depends on express, express-handlebars and express-session. Once the project is ready, we will commit it. now, we will say in terminal,

```
heroku login
```

And it will ask you press any key to open the browser tab. In the terminal it shows

Logging in…done

Logged in abc@jasper.com

All we have to say, **`heroku git:remote -a attainu-app`**

When we go to GitHub repository, in the code we can see the URL created in the Heroku platform and all we need to do is push the code. Using this command, we need to push the command. Now that this particular part is done, here it says, deployer application.

```
git push heroku main
```

This command basically tells the local git to push the entire code into the Heroku remote and on the branch master.

The code is running successfully, but there are few more changes that ned to be made. So, we need to make few changes in the package.json file. So, in package.json,

```
"engines": {
  "node": "^14"
},
```

When we upload the code to Heroku, it gets crashed because the port that we have given is 3000. So, the application is trying to connect to 3000 on Heroku, but actually Heroku is trying another port. So, for this, in the index.js file, we will wirte,

```
app.listen(process.env.PORT || 3000, () => {console.log('Server Started'
)})
```

Now, we will commit it and will run the same command again. As soon as we upload it, it will trigger a deploy. It will install everything again. In Heroku platform we can see all the details that we see in the terminal.