CREATING STUDENT INFORMATION DETAILS

Project Description: The scope of **STUDENT INFORMATON DETAILS** is to develop a web application that will store students information like Name, faculty number,department, etc. It will allow administrator to login and see details of all students, add new student detail and delete any student profile from this web application.

Introduction : Here in this project we will discuss about ,how to create this **STUDENT MANAGEMENT SYSTEM** web application on AWS CLOUD 9.

Pre Requisites:

- AWS cloud 9
- Understanding of terminal commands on Cloud 9.
- HTML5,JavaScript,NodeJS,MongoDB,REST API,JSON, Embedded JavaScript, Passport JS ,Express
- Understanding of HTTP methods like GET,POST,DELETE.

Advantages of this System:

- Details of any student can be stored in our database
- Delete any student detail can be done
- Better management of student detail can be dong

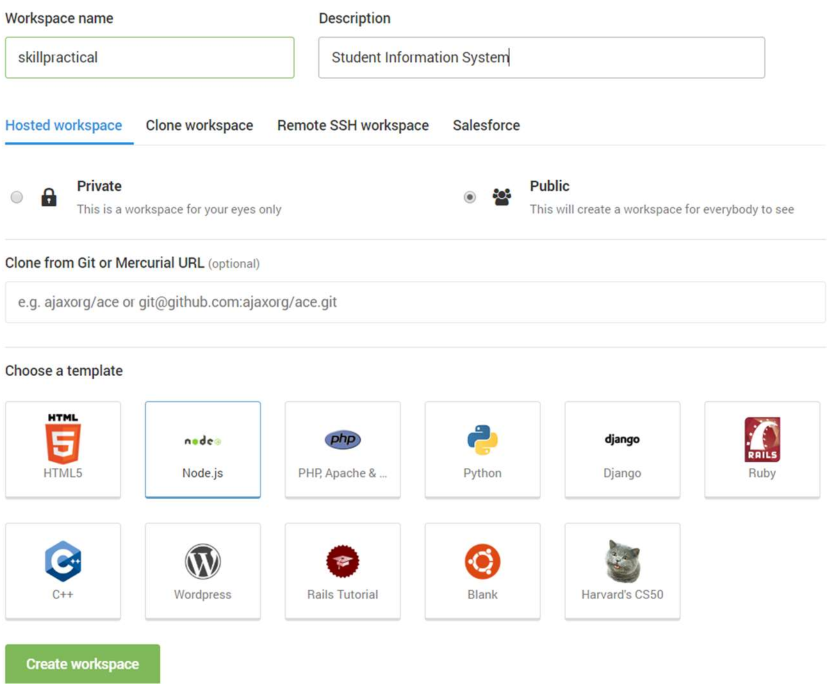
Application of this System:

- Can be used college, library, result processing unit, Club like literary club, debating club and many more
- In company for storing detail of interns

Steps for creating Student Management System:

Module 1:Creating a Node.js Project on cloud 9

- It all starts with creating a new NodeJS project. Choose an environment, for this project create a account on cloud 9.Cloud 9 is free for students.
- After this create a new workspace for NodeJS project:


Workspace name: skillpractical

Description: Student Information System

Hosted workspace | Clone workspace | Remote SSH workspace | Salesforce

Private — This is a workspace for your eyes only
[✓ Public — This will create a workspace for everybody to see]

Clone from Git or Mercurial URL (optional)

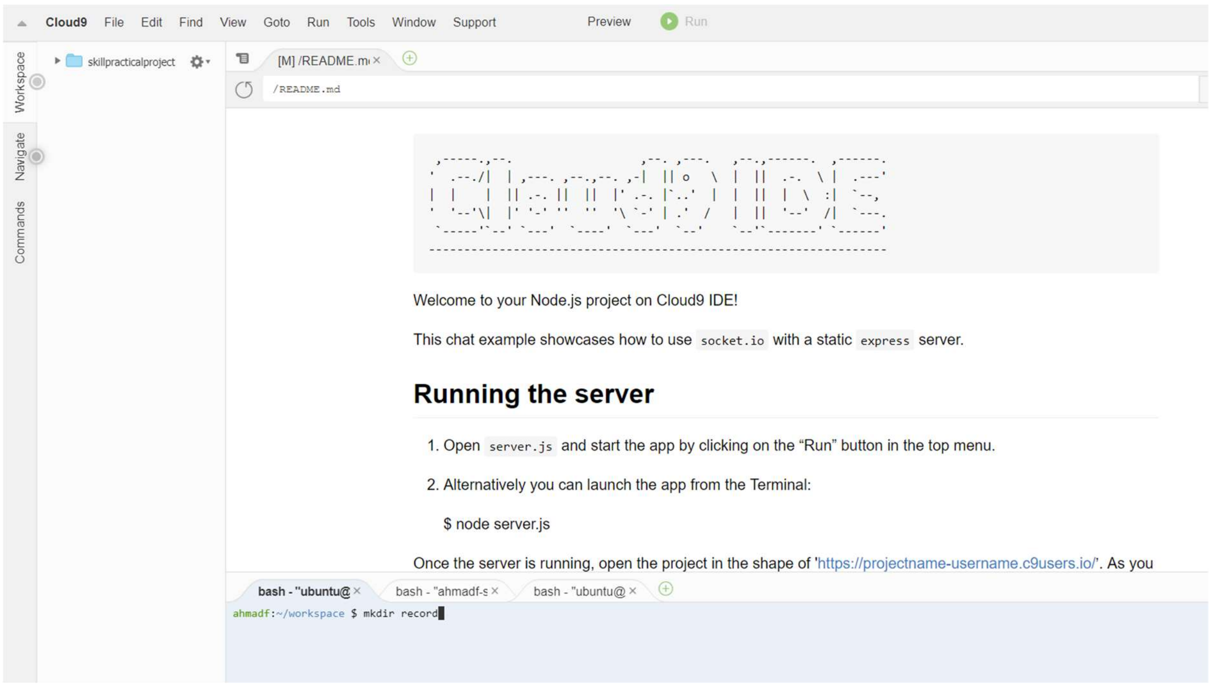
e.g. ajaxorg/ace or git@github.com:ajaxorg/ace.git

Choose a template

Click create workspace.

- Make a record directory by typing following commands:

  mkdir record



Welcome to your Node.js project on Cloud9 IDE!

This chat example showcases how to use `socket.io` with a static `express` server.

Running the server

1. Open `server.js` and start the app by clicking on the "Run" button in the top menu.
2. Alternatively you can launch the app from the Terminal:

   $ node server.js

Once the server is running, open the project in the shape of 'https://projectname-username.c9users.io/'. As you

```
ahmadf:~/workspace $ mkdir record
```

Summary: In above module we have discussed about how to create account on cloud 9 and creating a Node.JS project.

Test:

- Which of the following is correct about Node.js application? d
  a) Network Applications
  b) Distributed Systems
  c) Web applications
  d) All of the above
- Node.js is JavaScript run time environment that executes JavaScript code outside of a browser. B
  a) False
  b) True
- Which framework most commonly used in node.js? b
  a) Django
  b) Express
  c) Cake PHP
  d) Laravel

Resources:

- Creating a free account on AWS cloud 9: https://www.youtube.com/watch?v=AzqzVT7nGaw
- Node.js tutorial: https://www.w3schools.com/nodejs/
- Tutorial on Command Line of cloud 9: https://www.davidbaumgold.com/tutorials/command-line/
- Tutorial of command line: https://www.davidbaumgold.com/tutorials/command-line/
- Html tutorial: http://webdev.slides.com/coltsteele/deck-7-50#/

Module 2:Setting up MongoDB and Authentication

Step 1:Run database -For this open a new terminal, do following points-

- For a setting up Mongo DB database ,Steps are as follows

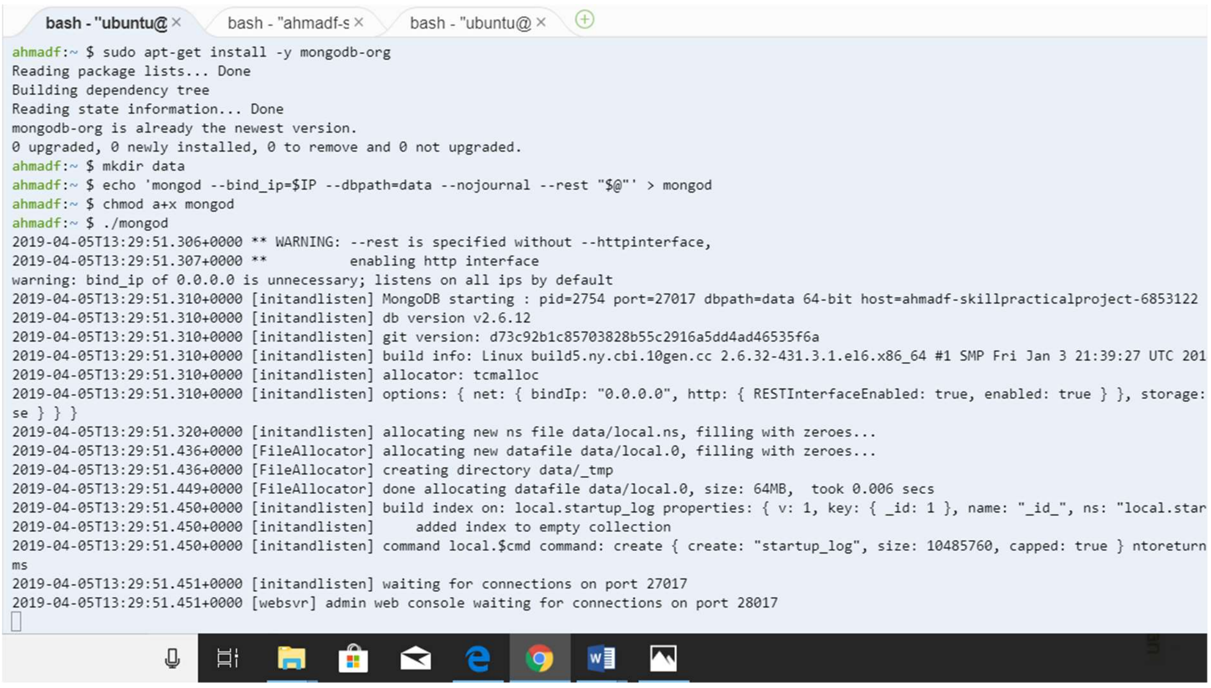
```
1-Firstly install Mongo DB, type command on command line
        sudo apt-get install -y mongodb-org
```

2-create a data directory, type command

```
mkdir data
```

```
3-type below  2 commands
1st-   $ echo 'mongod --bind_ip=$IP --dbpath=data --nojournal --rest "$@"' > mongod
2nd-   $ chmod a+x mongod
```

```
4-run your database
        ./mongod
```

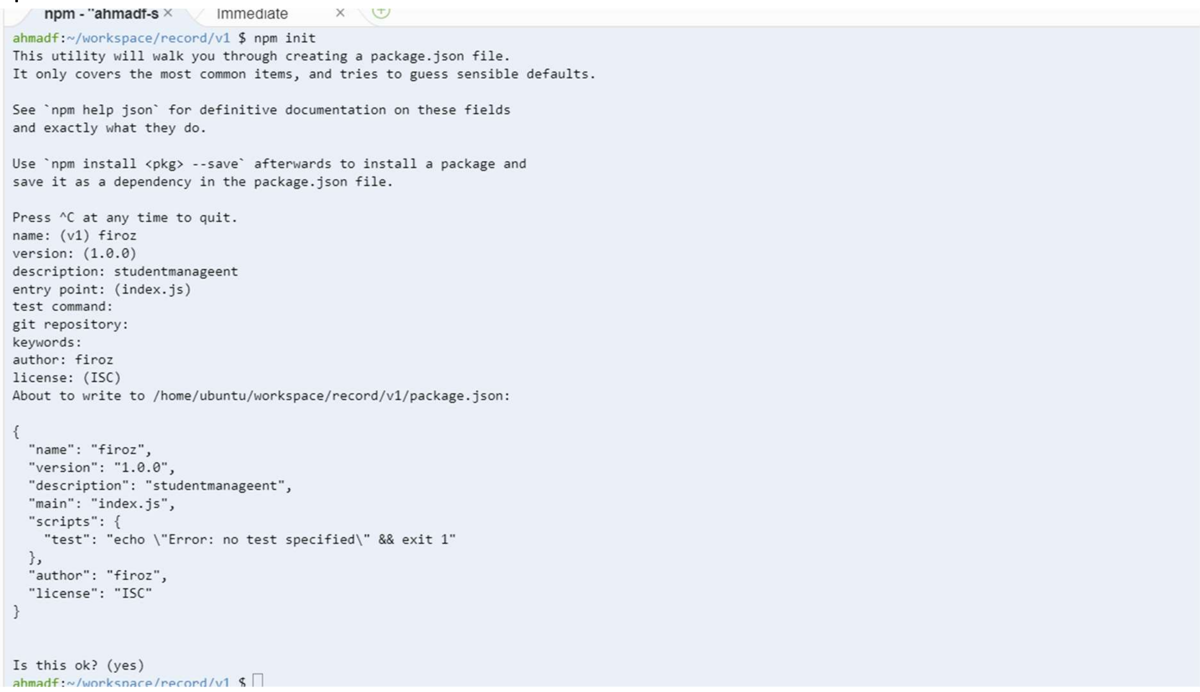


```
ahmadf:~ $ sudo apt-get install -y mongodb-org
Reading package lists... Done
Building dependency tree
Reading state information... Done
mongodb-org is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
ahmadf:~ $ mkdir data
ahmadf:~ $ echo 'mongod --bind_ip=$IP --dbpath=data --nojournal --rest "$@"' > mongod
ahmadf:~ $ chmod a+x mongod
ahmadf:~ $ ./mongod
2019-04-05T13:29:51.306+0000 ** WARNING: --rest is specified without --httpinterface,
2019-04-05T13:29:51.307+0000 **          enabling http interface
warning: bind_ip of 0.0.0.0 is unnecessary; listens on all ips by default
2019-04-05T13:29:51.310+0000 [initandlisten] MongoDB starting : pid=2754 port=27017 dbpath=data 64-bit host=ahmadf-skillpracticalproject-6853122
2019-04-05T13:29:51.310+0000 [initandlisten] db version v2.6.12
2019-04-05T13:29:51.310+0000 [initandlisten] git version: d73c92b1c85703828b55c2916a5dd4ad46535f6a
2019-04-05T13:29:51.310+0000 [initandlisten] build info: Linux build5.ny.cbi.10gen.cc 2.6.32-431.3.1.el6.x86_64 #1 SMP Fri Jan 3 21:39:27 UTC 201
2019-04-05T13:29:51.310+0000 [initandlisten] allocator: tcmalloc
2019-04-05T13:29:51.310+0000 [initandlisten] options: { net: { bindIp: "0.0.0.0", http: { RESTInterfaceEnabled: true, enabled: true } }, storage:
se } } }
2019-04-05T13:29:51.320+0000 [initandlisten] allocating new ns file data/local.ns, filling with zeroes...
2019-04-05T13:29:51.436+0000 [FileAllocator] allocating new datafile data/local.0, filling with zeroes...
2019-04-05T13:29:51.436+0000 [FileAllocator] creating directory data/_tmp
2019-04-05T13:29:51.449+0000 [FileAllocator] done allocating datafile data/local.0, size: 64MB,  took 0.006 secs
2019-04-05T13:29:51.450+0000 [initandlisten] build index on: local.startup_log properties: { v: 1, key: { _id: 1 }, name: "_id_", ns: "local.star
2019-04-05T13:29:51.450+0000 [initandlisten]     added index to empty collection
2019-04-05T13:29:51.450+0000 [initandlisten] command local.$cmd command: create { create: "startup_log", size: 10485760, capped: true } ntoreturn
ms
2019-04-05T13:29:51.451+0000 [initandlisten] waiting for connections on port 27017
2019-04-05T13:29:51.451+0000 [websvr] admin web console waiting for connections on port 28017
```

`Now my database is running and it will store data given to it.`

Step 2:Include all required package on your package.json file

- run following commands for that:
  - npm init



```
ahmadf:~/workspace/record/v1 $ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (v1) firoz
version: (1.0.0)
description: studentmanageent
entry point: (index.js)
test command:
git repository:
keywords:
author: firoz
license: (ISC)
About to write to /home/ubuntu/workspace/record/v1/package.json:

{
  "name": "firoz",
  "version": "1.0.0",
  "description": "studentmanageent",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "firoz",
  "license": "ISC"
}


Is this ok? (yes)
ahmadf:~/workspace/record/v1 $
```

Above process are-

“name” :”(myproject)”, //press enter

“ version”: “1.0.0” , //press enter

“description”: “studentmanagement” ,   //press enter

“main” :”index.js”,   //leave it and press enter

“scripts” : {

```
                "test" :" echo \"Error: no test specified\" && exit 1"
            },                              //leave it and press enter
    "license":"ISC"            //press and enter
Is this ok? (yes)
```

Explanation: This will create a package.json file (screenshot of package.json file is taken below). Package.json contains all the files needed for a module and modules are the JavaScript libraries that can be included in Node project according to the requirement of the project. It does not contain some modules like mongoose ,body parser, passport. So we have to explicitly type below commands for including all modules .

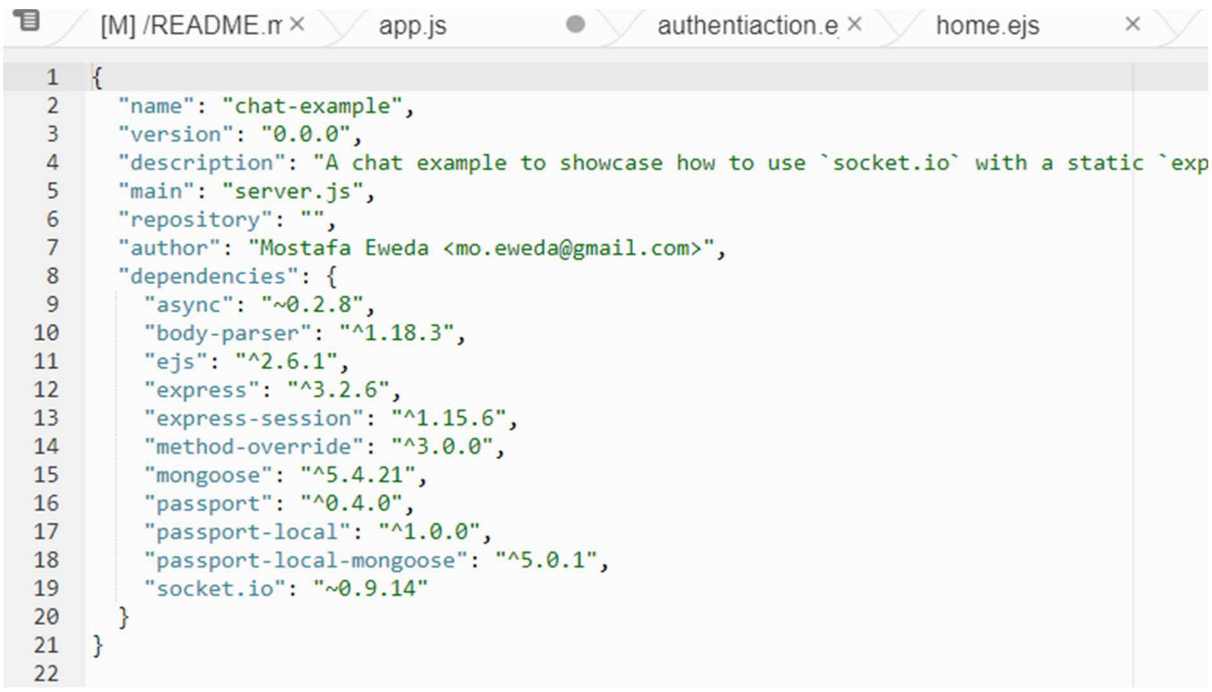
- npm install express mongoose –save

  Explanation: This command will install the express and mongoose module into our package.json. Both module as installed as a dependency in package.json.
- npm install passport passport -local –save

  Explanation: Passport makes it a breeze to provide authentication for our users.
- npm install passport -local-mongoose –savesave

  Explanation: In our application, in website authentication for users via username and password. So this authentication is done by passport local module .
- npm install body-parser express-session –save

  Explanation: Parse incoming request bodies in a middleware before your handlers , available under the req.bodyproperty.



```
{
  "name": "chat-example",
  "version": "0.0.0",
  "description": "A chat example to showcase how to use `socket.io` with a static `exp
  "main": "server.js",
  "repository": "",
  "author": "Mostafa Eweda <mo.eweda@gmail.com>",
  "dependencies": {
    "async": "~0.2.8",
    "body-parser": "^1.18.3",
    "ejs": "^2.6.1",
    "express": "^3.2.6",
    "express-session": "^1.15.6",
    "method-override": "^3.0.0",
    "mongoose": "^5.4.21",
    "passport": "^0.4.0",
    "passport-local": "^1.0.0",
    "passport-local-mongoose": "^5.0.1",
    "socket.io": "~0.9.14"
  }
}
```

Included all files in package.json

Step 3: Create a app.js file and open it:

For creating app.js type following command on terminal:

- On first Terminal,

  Go into record directory, by following command

  cd record

after this make a JS file

touch app.js

Now require all package in your app.js file and connection it to database:

Coding our app.js file:

After this ,copy and paste this code below

1. *var express=require("express"),*
2. *mongoose=require("mongoose"),*
3. *passport=require("passport"),*
4. *methodOverride=require("method-override"),*
5. *bodyParser=require("body-parser"),*
6. *LocalStrategy=require("passport-local"),*
7. *passportLocalMongoose=require("passport-local-mongoose");*
8. **mongoose.connect('mongodb://localhost/student_record', {useNewUrlParser: true});**

Code Explanation

- **Line1:** require express framework
- **Line2:** require mongoose for schema-based solution to model our application data
- **Line 3:** require passport for authentication which is a authentication middleware for Node.js
- **Line 4:** To use a header to override the method, specify the header name as a string argument to the methodOverride function.
- **Line 5:** body-parser extract the entire body portion of an incoming request stream and exposes it on req.body.
- **Line 6**:This module lets you authenticate using a username and password in your Node.js applications. By plugging into Passport ,local authentication can be easily and unobtrusively Integrated into any applications or framework that supports Connect-style middleware, including Express.
- **Line 7:** use passport-local-mongoose to authenticate a user after submitting a POST on the /register form.
- **Line 8:** When you call `mongoose.connect`, it will set up a connection with the database(connects student_record to databse).

Summary:

In above, we have discussed how to create create our database on basis of some command. We also installed all packages like express,bodyparser,etc.

Test:

- To install Node.js express module which statement is correct? a
  a- $ npm install express
  b- $ node install express
  c- $ install express
  d- None of above
- What npm stands for? a
  a- Node package manager
  b- Node project manager
  c- New project manager
  d- New package manager

Resources:

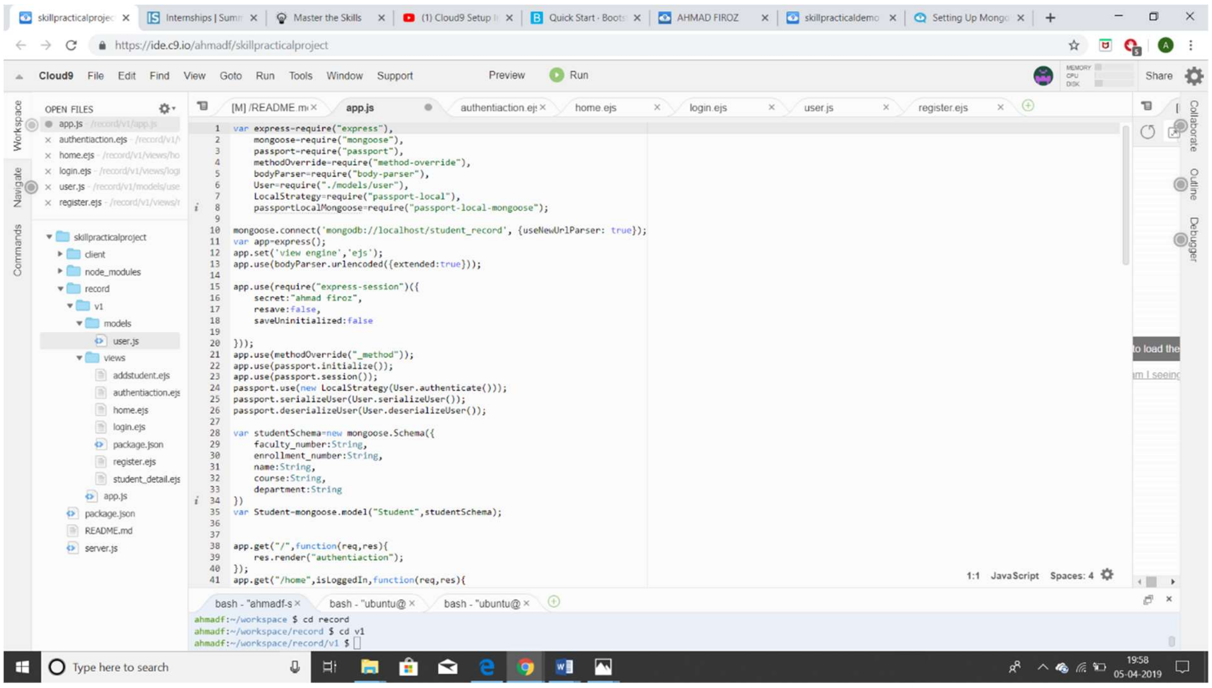
- For overriding: https://www.npmjs.com/package/method-override
- Connecting our new database:
  https://stackoverflow.com/questions/20360531/mongoose-connection
- Passport.js Authentiaction official website:
  http://www.passportjs.org/packages/passport-local/

Module 3:Creating authentication(login/Sign Up)

Step 1: Authentication in app.js and create required files:

- Make make a new directory v1
  ```
  mkdir v1
  ```
- Go into this directory
  ```
  cd v1
  ```
- Create two directory more inside v1
  1-mkdir views
  2- mkdir models
- Create following files in directory views
  ```
  touch authentication.ejs
  touch login.ejs
  touch register.ejs        //signup page
  ```
- Create following files in directory models
  ```
  touch user.js             //for authentication criteria
  ```

Now open your app.js file previously created and copy following codes:



```
var express=require("express"),
    mongoose=require("mongoose"),
    passport=require("passport"),
    methodOverride=require("method-override"),
    bodyParser=require("body-parser"),
    User=require("./models/user"),
    LocalStrategy=require("passport-local"),
    passportLocalMongoose=require("passport-local-mongoose");

mongoose.connect('mongodb://localhost/student_record', {useNewUrlParser: true});
var app=express();
app.set('view engine','ejs');
app.use(bodyParser.urlencoded({extended:true}));

app.use(require("express-session")({
    secret:"ahmad firoz",
    resave:false,
    saveUninitialized:false

}));
app.use(methodOverride("_method"));
app.use(passport.initialize());
app.use(passport.session());
passport.use(new LocalStrategy(User.authenticate()));
passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());

var studentSchema=new mongoose.Schema({
    faculty_number:String,
    enrollment_number:String,
    name:String,
    course:String,
    department:String
})
var Student=mongoose.model("Student",studentSchema);


app.get("/",function(req,res){
    res.render("authentiaction");
});
app.get("/home",isLoggedIn,function(req,res){
```

```
ahmadf:~/workspace $ cd record
ahmadf:~/workspace/record $ cd v1
ahmadf:~/workspace/record/v1 $
```

Codes in app.js file

Code the app.js file:

Line1: app.use(passport.initialize());

Line 2: app.use(passport.session());

Line 3: passport.use(new LocalStrategy(User.authenticate()));

Line 4: passport.serializeUser(User.serializeUser());

Line 5: passport.deserializeUser(User.deserializeUser());

Line 6: app.get("/",function(req,res){

Line 7: res.render("authentiaction");});

Line 8: app.post("/register",function(req,res){

LINE 9: req.body.username

Line 10: req.body.password

Line11 :
User·register(newUser({username:req·body·username}),req·body·password,functi on(err,user){if(err)

Line 12: *{console·log(err);return res·render('register');}*

Line 13: *passport·authenticate("local")(req,res,function(){*

Line 14: *res·redirect("/home");});});});*

Line 15: *app·listen(process·env·PORT,process·env·IP,function(){*

Line 16: *console·log("Sever Started···");})*

Code Explanation:

Line 1: It initializes the authentication module.

Line 2: It acts as a middleware to alter the required object and change the 'user' value that is currently session id into true deserialized user object.

Line 3: LocalStrategy expects to find credentials in parameters named username and password.

Line 4-5: In this, when a user is authenticated, Passport will save the user's _id property to the session as req.session.passport.user. Later on when the user object is needed, Passport will use the _id property to grab the user object from the database.

Line 6-7:app.get("/") to handle get request of "/".It will send user to for authentication.

Line 8:app.post("/register") to handle post request of "/register".

Line 9-10: By using req.body.username ,it will take its username for saving in our database. Similarly for password.

Line 11: It will take its username and password and save it our database.

Line 12: If there is any error, It will again send user to sign up page.

Line 13-14: Now authentication will occur for matching username and password. If user is authenticated, he will go to home page for seeing or adding more student detail.

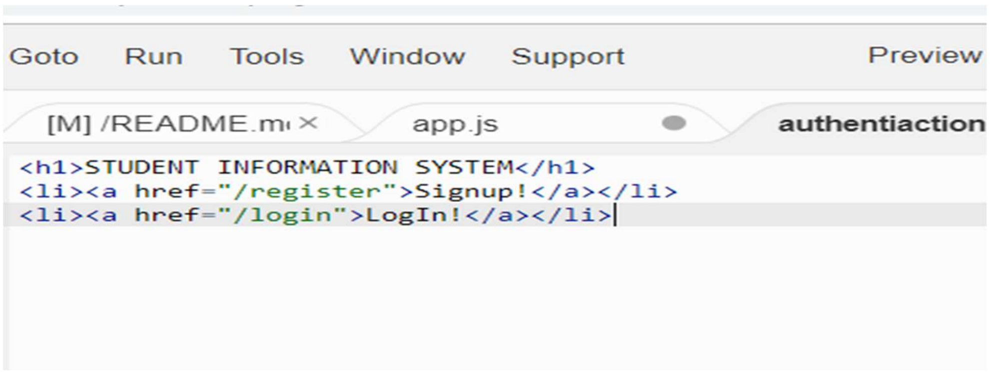
Line 15-16: `app.listen`, that makes your server be able to accept a parameter from the environment what port to listen on.

Remark: By creating more routing like for login, register for get request you can render page for login.ejs and register.ejs in the same format of Line 6-7.

Step 2: creating authentication page : This page will create a authentication page for authentication by login/signup.

Codes in authentication.ejs file: copy below code in your authentication ejs file-

1. <h1>STUDENT INFORMATION SYSTEM</h1>
2. <li><a href="/register">Signup!</a></li>
3. <li><a href="/login">LogIn!</a></li>



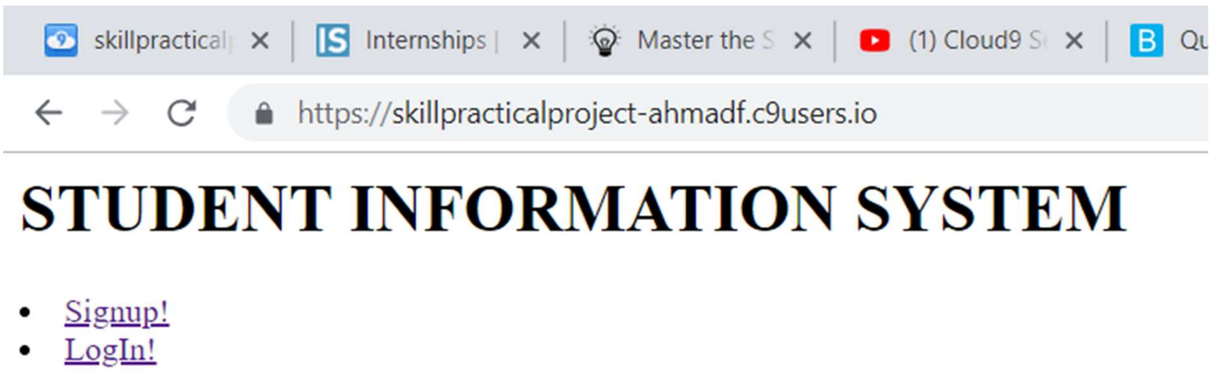
```
<h1>STUDENT INFORMATION SYSTEM</h1>
<li><a href="/register">Signup!</a></li>
<li><a href="/login">LogIn!</a></li>
```

Codes in authentication.ejs file

Code explanation of authentication.ejs file:

Line 1: h1 tag displaying texy STUDENT INFORMATION SYSTEM
LINE 2: on clicking this page ,you wil give to register(Sign Up) page.
LINE 3: this is link for LogIn page.

authentication page



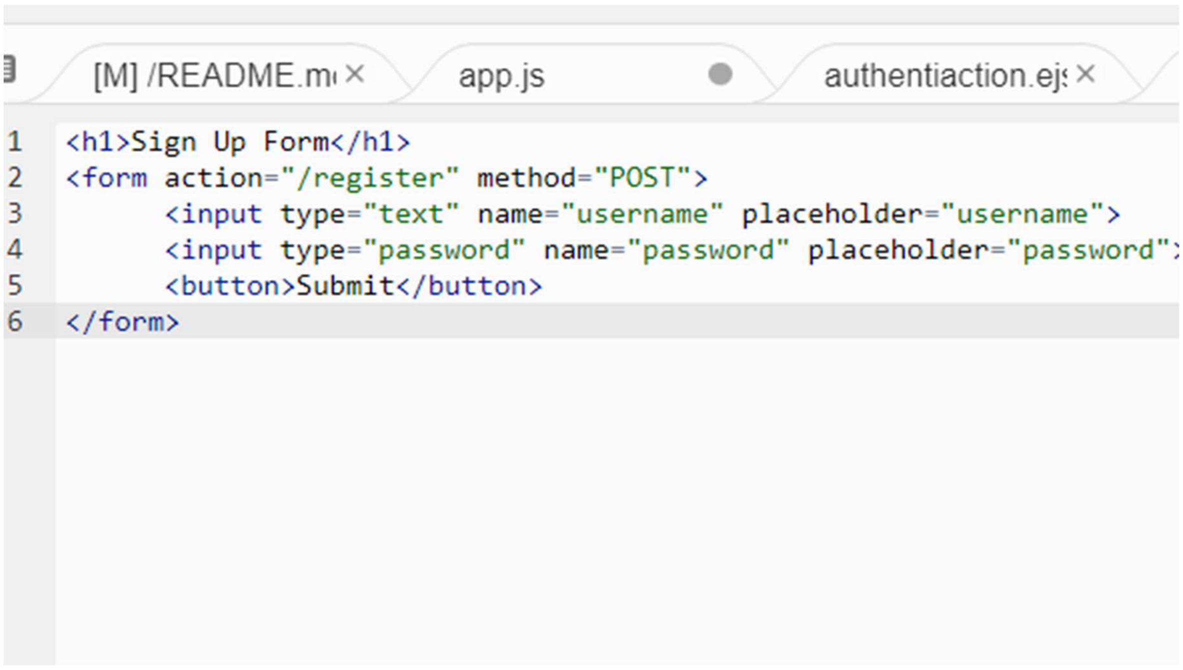
STUDENT INFORMATION SYSTEM

- Signup!
- LogIn!

Step 3: creating sign up page:

Codes in register.ejs file: copy below code and paste it to your register.ejs page.

```
1.   <h1>Sign Up Form</h1>
2.   <form action="/register" method="POST">
3.   <input type="text" name="username" placeholder="username">
4,   <input type="password" name="password" placeholder="password">
5.   <button>Submit</button>
6.   </form>
```



```
1 <h1>Sign Up Form</h1>
2 <form action="/register" method="POST">
3       <input type="text" name="username" placeholder="username">
4       <input type="password" name="password" placeholder="password"
5       <button>Submit</button>
6 </form>
```

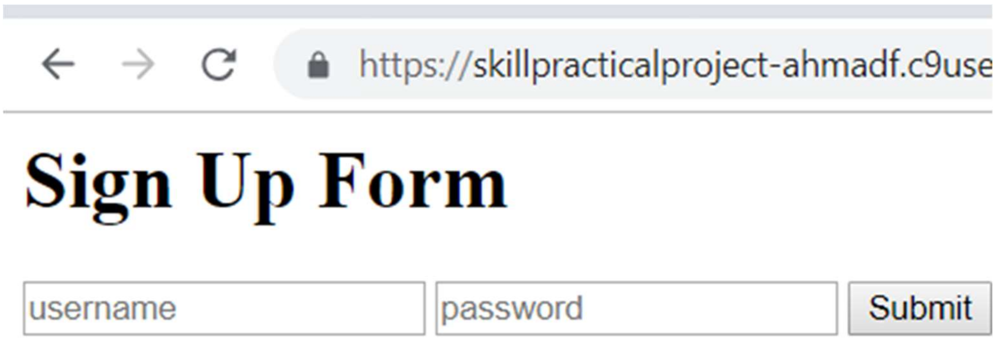
register.ejs file

Code Explanation of register.ejs file

Line1:displaying “Sign Up Form” text

Line 2-6: In this form tag,we take two inputs username ,password. By taking two inputs we store its username and password in our database for future LogIn.

Register/Sign Up Page below

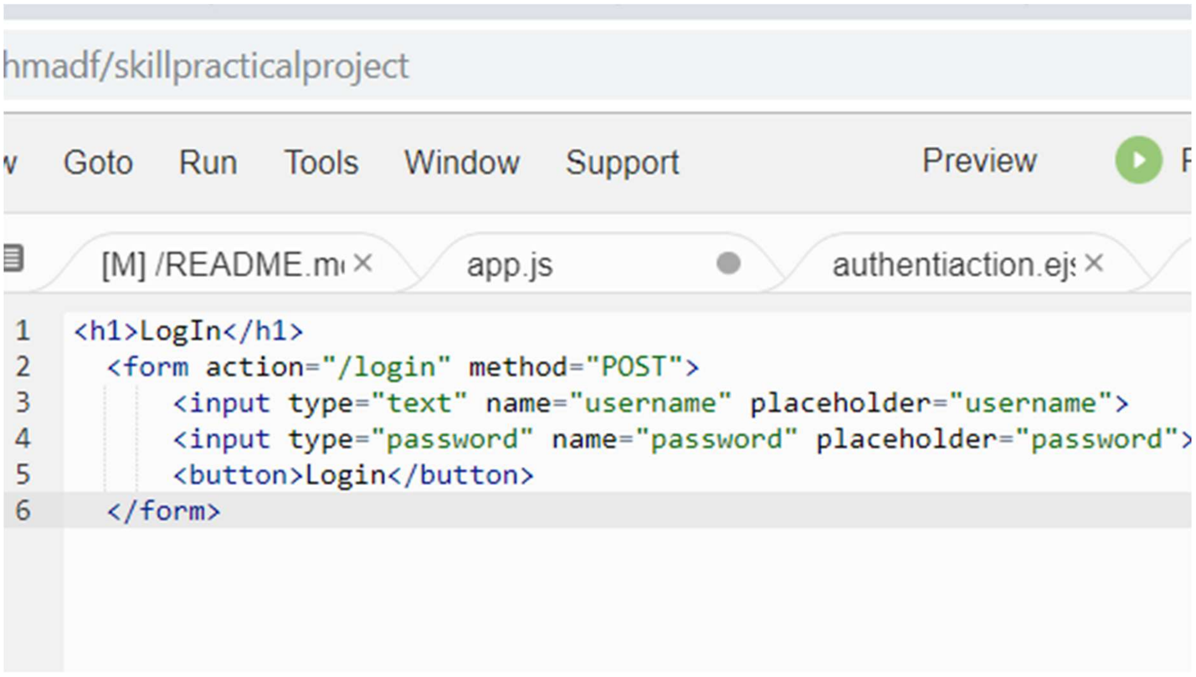


Sign Up Form

Step 4:creating login page:

Codes in login.ejs file: : copy below code and paste it to your login.ejs page.

```
1.  <h1>LogIn</h1>
2.   <form action="/login" method="POST">
3.          <input type="text" name="username" placeholder="username">
4.         <input type="password" name="password" placeholder="password">
5.    <button>Login</button>
6.   </form>
```



```
<h1>LogIn</h1>
  <form action="/login" method="POST">
      <input type="text" name="username" placeholder="username">
      <input type="password" name="password" placeholder="password">
      <button>Login</button>
  </form>
```

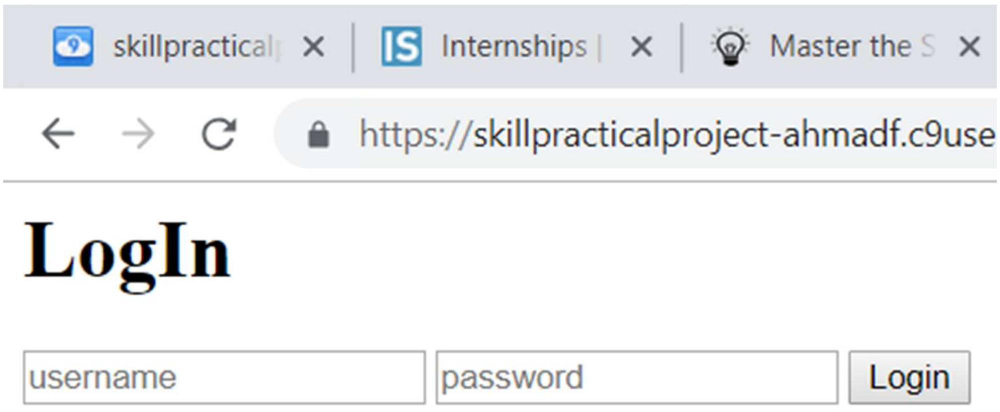
login.ejs file

Code explanation of screenshot of above login.ejs file:

Line 1: displaying "LogIn" text.

Line 2-6: In login page we take two inputs username and password, authenticate with any username ,password in our database ,if it is present in database then app.post it to home.ejs page.

Log in page



Summary: In this module, we have done coding of our app.js file ,authentication page ,sign up page and log in page in backend of our website. We have added functionalities of storing administrator username and password and after successful sign up we give access to
Administrator to see student detail in his institute, add more student.

Test:

- What tag is used used to link from one page to another? c

  a)- src

  b)- img

  c)- href

  d)-h1

- HTML web pages can be read and rendered by____? c

  a- Compiler

  b- Server

  c- Web browser

d- Interpreter

- HTML tags are surrounded by which type of brackets**?** d

a- Curly

b- Round

c- Squart

d- Angle

Resources:

For creating mongoose : https://mongoosejs.com/

For authentication: http://www.passportjs.org/

Tutorial of authenctiaction: https://medium.com/@johnnyszeto/node-js-user-authentication-with-passport-local-strategy-37605fd99715

Notes about npm: https://www.youtube.com/channel/UCqo2YWBtmFSWhuUk4WEyfGg

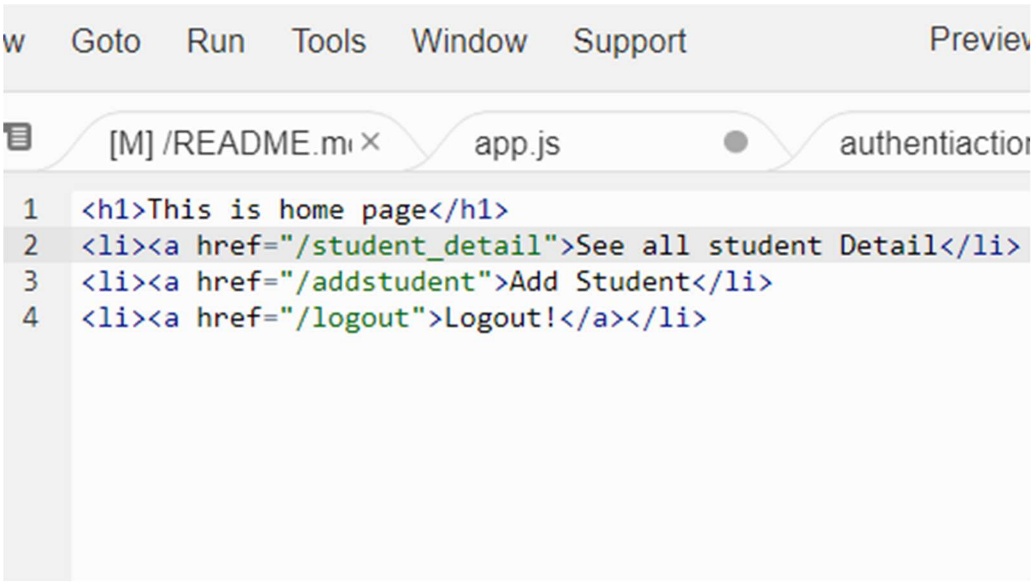
Module 4:Creating home page, new student_record, displaying it, deleting any student record

Step 1:Creating home page:

After successful registering/login we get this page:

Code in home.ejs: copy below code and paste it to your home.ejs page.

```
1.  <h1>This is home page</h1>
2.  <li><a href="/student_detail">See all student Detail</li>
3.  <li><a href="/addstudent">Add Student</li>
4.  <li><a href="/logout">Logout!</a></li>
```



```
<h1>This is home page</h1>
<li><a href="/student_detail">See all student Detail</li>
<li><a href="/addstudent">Add Student</li>
<li><a href="/logout">Logout!</a></li>
```

Codes in home.ejs

Code Explaination of above screenshot of home.ejs file:

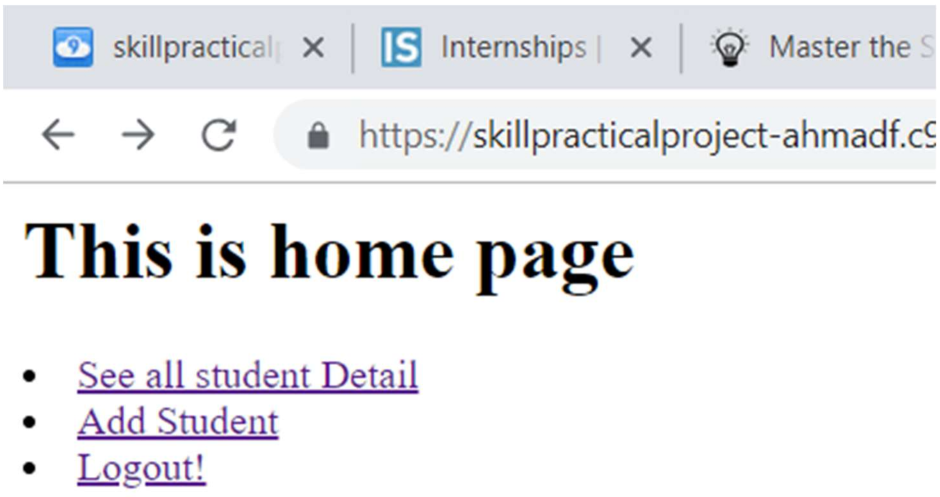
Line 1: text displaying "This is home page"

Line 2-4: provide three option of

a)- adding new student detail

b)-display all student detail present in database

c)- logout from this page.



This is home page

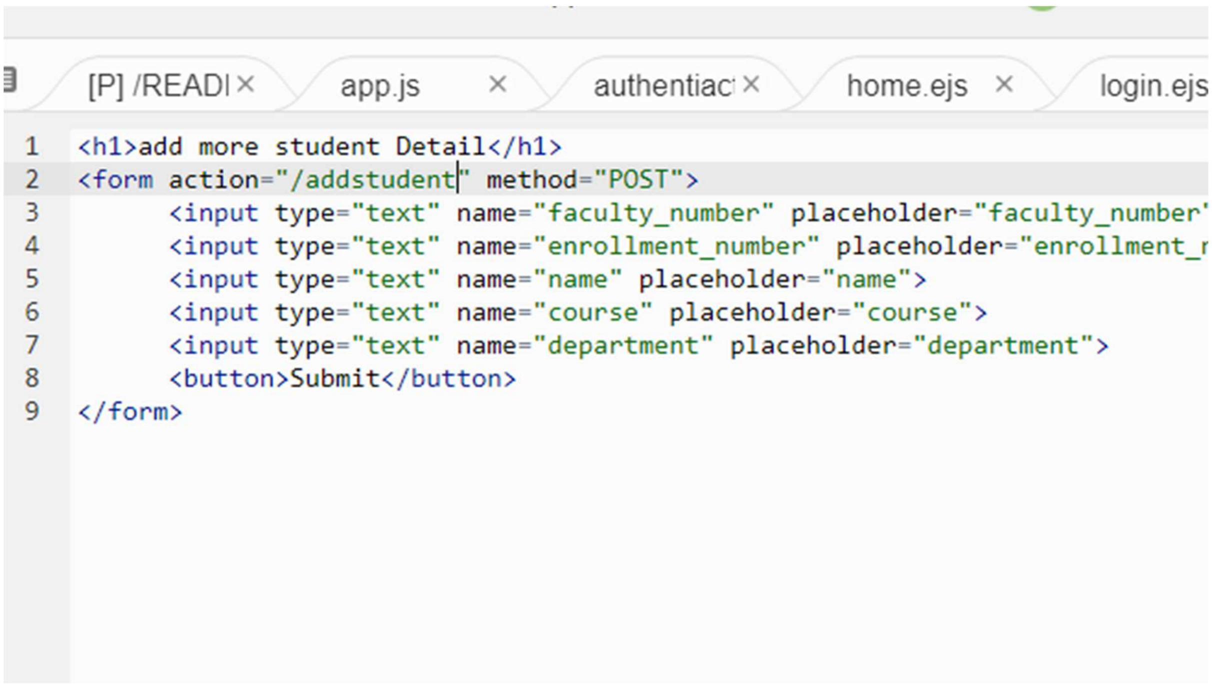
- See all student Detail
- Add Student
- Logout!

Step2-creating addstudent.ejs

This file takes detail of any student from administrator like it his name, faculty number ,etc.

Codes in addstudent.ejs file: copy and paste below code in addstudent.ejs file.

```
1. <h1>add more student Detail</h1>
2.  <form action=“/addstudent” method=“POST”>
3.    <input type=“text” name=“faculty_number” placeholder=“faculty_number”>
4.     <input type=“text” name=“enrollment_number” placeholder=“enrollment_number”>
5.    <input type=“text” name=“name” placeholder=“name”>
6.     <input type=“text” name=“course” placeholder=“course”>
7.     <input type=“text” name=“department” placeholder=“department”>
8.      <button>Submit</button>
9.   </form>
```



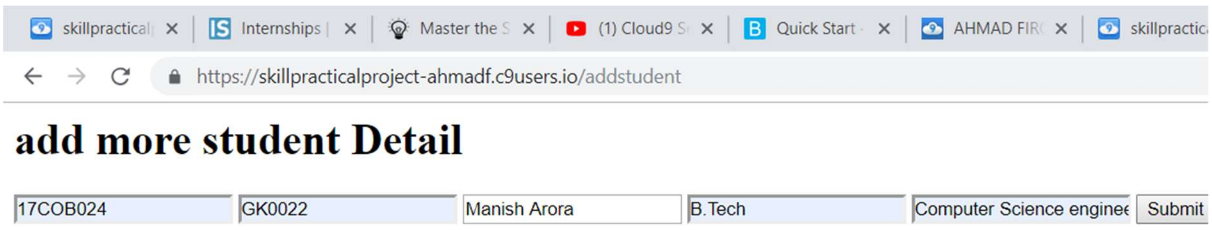
```
1  <h1>add more student Detail</h1>
2  <form action="/addstudent" method="POST">
3        <input type="text" name="faculty_number" placeholder="faculty_number'
4        <input type="text" name="enrollment_number" placeholder="enrollment_r
5        <input type="text" name="name" placeholder="name">
6        <input type="text" name="course" placeholder="course">
7        <input type="text" name="department" placeholder="department">
8        <button>Submit</button>
9  </form>
```

addstudent.ejs file

Code explanation of above screenshot Addstudent.ejs file:

Line 1:displaying text “add more student Detail”

Line 2-9: A form tag taking student detail faculty number, enrolment number, name, course, department. After clicking Submit, student detail will be added to our database.



https://skillpracticalproject-ahmadf.c9users.io/addstudent

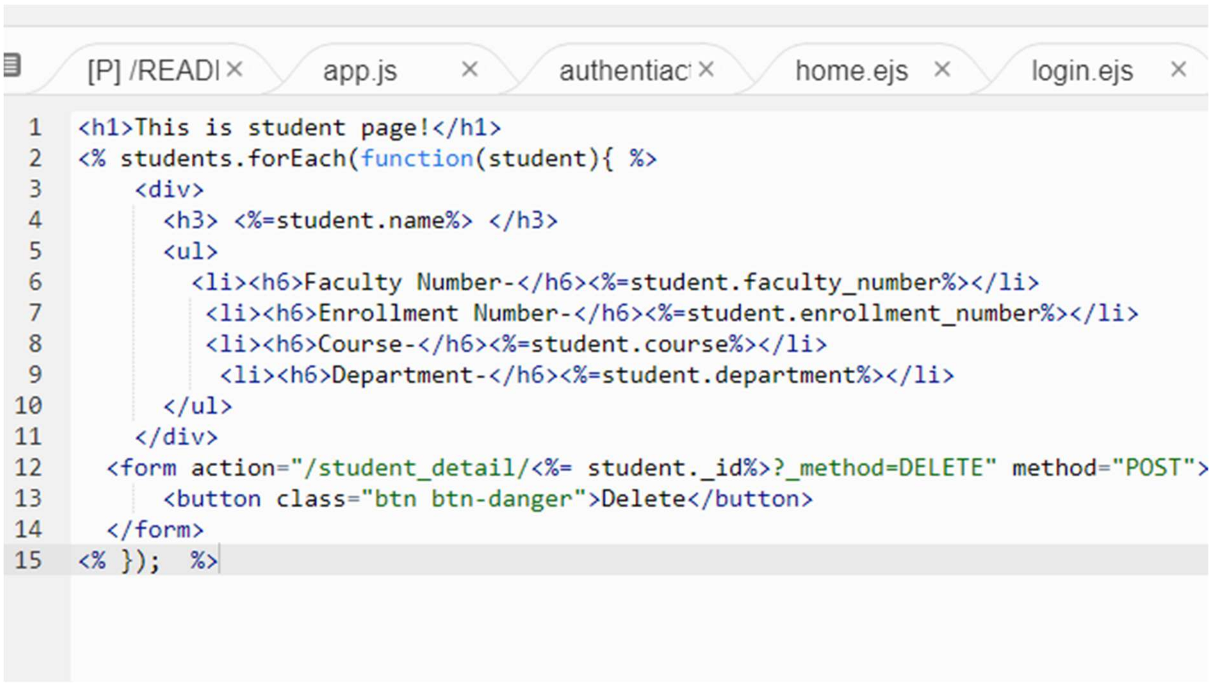
add more student Detail

17COB024 | GK0022 | Manish Arora | B.Tech | Computer Science enginee | Submit

Step 3: Creating student_detail.ejs: This file will display all student detail stored in our database.

Codes in student_detail.ejs file: copy and paste below code in student.detail.ejs file-

```
<h1>This is student page!</h1>
<% students.forEach(function(student){ %>
  <div>
    <h3> <%=student.name%> </h3>
     <ul>
        <li><h6>Faculty Number-</h6><%=student.faculty_number%></li>
        <li><h6>Enrollment Number-</h6><%=student.enrollment_number%></li>
        <li><h6>Course-</h6><%=student.course%></li>
         <li><h6>Department-</h6><%=student.department%></li>
      </ul>
  </div>
<form action="/student_detail/<%= student._id%>?_method=DELETE" method="POST">
  <button class="btn btn-danger">Delete</button>
</form>
<% }); %>
```



```
1  <h1>This is student page!</h1>
2  <% students.forEach(function(student){ %>
3      <div>
4        <h3> <%=student.name%> </h3>
5        <ul>
6          <li><h6>Faculty Number-</h6><%=student.faculty_number%></li>
7           <li><h6>Enrollment Number-</h6><%=student.enrollment_number%></li>
8           <li><h6>Course-</h6><%=student.course%></li>
9            <li><h6>Department-</h6><%=student.department%></li>
10       </ul>
11     </div>
12   <form action="/student_detail/<%= student._id%>?_method=DELETE" method="POST">
13       <button class="btn btn-danger">Delete</button>
14   </form>
15 <% }); %>
```

student_detail.ejs file

Code explanation of above screenshot student_detail.ejs file:

Line1: displaying text “This is student page”

Line 2: executing a loop **for** printing all student detail in our database.

Line 3-11: displaying each student detail

Student name in dark (in h3 tag)

Faculty number

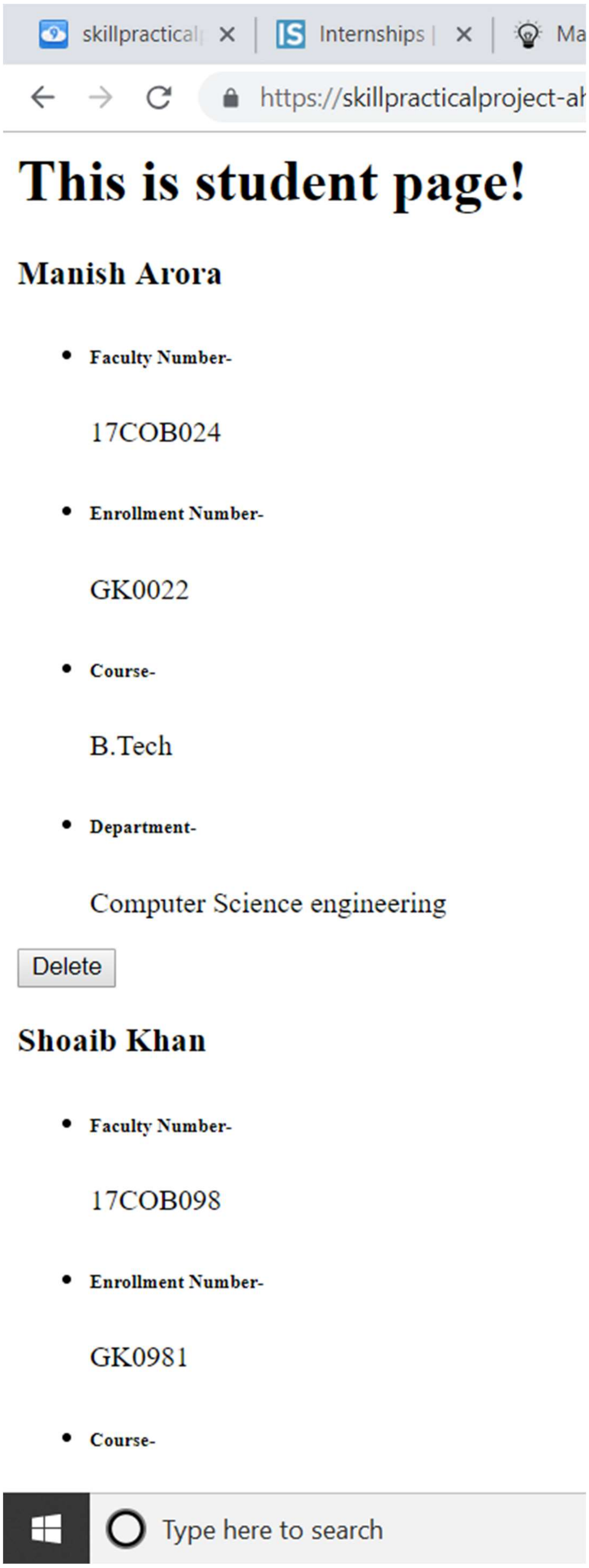
Enrolment number

Course

Department

Line 12-14: This code is explained in step 4.

Displaying all student detail below



This is student page!

Manish Arora

- **Faculty Number-**

  17COB024

- **Enrollment Number-**

  GK0022

- **Course-**

  B.Tech

- **Department-**

  Computer Science engineering

Delete

Shoaib Khan

- **Faculty Number-**

  17COB098

- **Enrollment Number-**

  GK0981

- **Course-**

Step 4: Deleting student record of any student-

In the student_detail .ejs file line 12-14 ,by clicking "Delete" button, student record of any student can be deleted.

Summary: Here In this module we have added functionalities of adding new student detail, displaying all student detail and delete any student record.

Test:

- Node.js is ________Language. a
  - a- server side
  - b- client side
  - c- framework
  - d- all of the above
- Is node js multithreaded? b
  - a- True
  - b- False
- Which function is used to include modules in Node Js? b
  - a- Include()
  - b- require()
  - c- attach()
  - d- import()

Resources:

- Information of RESTful Blog: https://softwareengineering.stackexchange.com/questions/114156/why-are-there-are-no-put-and-delete-methods-on-html-forms
- Node js tutorial: https://www.udemy.com/the-web-developer-bootcamp/
- Express web framework: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs

Summary:

In this project we learnt how to run server for our dynamic website in cloud 9.We have just made a prototype for STUDENT MANAGEMENT SYSTEM. We have learned server side language node.js ,authentication.js for signup/login. We have seen how our website taking data from external source(administrator) and storing into our database(creating a dynamic web page).

For all codes in detail visit my Github account: All codes of Student Management System is done on my account on Cloud 9.For Code visit my account and link is given as follows-

https://github.com/Ahmadmf/Student-Management-System

It is In the record/v1 directory.