

PROJECT REPORT

JOB SEARCH APP

Mentor : Dr Subhash Chandra Pandey

Aditi(BE/15086/17) CSE
Salvi Verma(BE/15048/17) CSE

CONTENTS

- INTRODUCTION
- SOFTWARE REQUIREMENT
- HARDWARE REQUIREMENT
- TECHNOLOGIES USED
 - WHY ANGULAR ?
 - WHY SPRING BOOT ?
 - WHY SQL ?
- FLOW CHART
- SCREENSHOTS
- REFERENCES
- THANK YOU NOTE

Introduction

- This webapp has some filters for searching for any job.
- Some of the filters include Job Profile, company name(eg : Software developer, Data engineer).
- It is open to everyone..
- This application is made available to wide base of job seekers
- This app encapsulates the security of credentials of the registered users.
- Anonymous contribution for the vacancy of post in a specific company

Software requirement

- JAVA setup
- Angular Setup
- Editor : IntelliJ (Backend)
- Editor : Visual Studio Code (Frontend)

Hardware requirement

- High level laptop
- 8+ gb ram
- 4+ cores

Technologies Used

- Programming language : JAVA
- Angular is used for frontend.
- Spring Boot is used for backend.
- Queries are written using MySQL.
- InMemory h2 database has been used.

Why Angular ?

Angular is an open-source front-end framework developed by Google for creating dynamic, modern web apps.

Feature of angular:

- TypeScript : Angular applications are built using TypeScript language, a superscript for JavaScript, which ensures higher security as it supports types (primitives, interfaces, etc.). It helps catch and eliminate errors early when writing the code or performing maintenance tasks.
- POJO : In angular data models are Plain old javascript object which increase code readability and reusability.
- Easy Testing : Angular.js modules has the application parts, which are easy to manipulate.
- Modular Structure : Angular organizes code into *buckets*, whether it is components, directives, pipes, or services. Modules make application functionality organization easy, segregating it into features and reusable chunks. Modules also allow for lazy loading, which paves way for application feature loading in the background or on-demand.
- Code Consistency : Consistent coding has several benefits, such as it makes sites easier to use and enables the use of templates or pre-defined code snippets.

- Simplified Unit-Testing: It organizes code into buckets whether it is components or directives e.t.c, all these being independent, makes the testing hassle free.
- Reusability :The component-based structure of Angular makes the components highly reusable across the app.
- Improved Readability : Consistency in coding makes reading the code a piece of cake for new developers on an ongoing project, which adds to their productivity.
- Ease of Maintenance : Decoupled components are replaceable with better implementations.

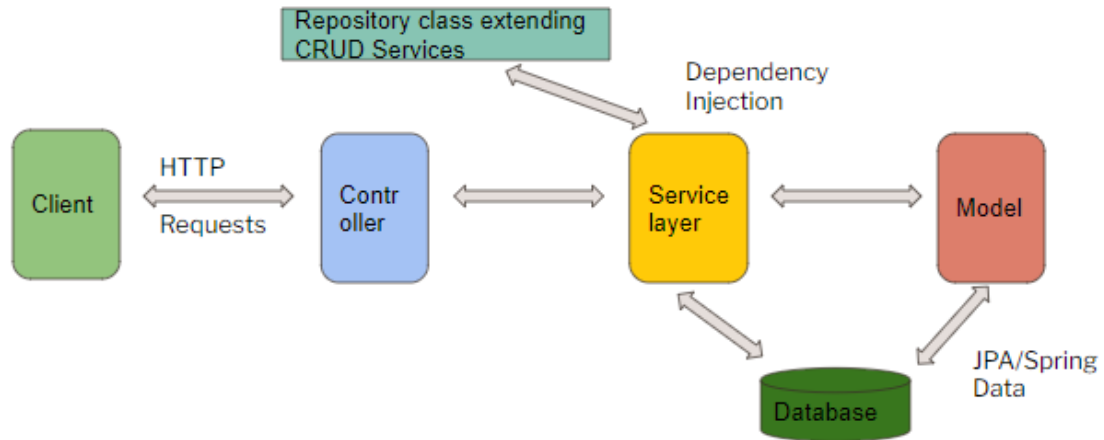
Why Spring Boot ?

Spring is a very popular Java-based framework for building web and enterprise applications.

Features of Spring Boot:

- Easy dependency Management: one class is dependent on the other class for its execution , this is called dependency. Springboot will on its own create the object of the independent class dynamically at the run time and inject it into the dependent class.
- Auto Configuration: Spring Boot auto-configure feature of a bean automatically into the project.
- Embedded Servlet Container Support: Embedding servlet container has advantages such as simple startup, eliminating mismatched libraries or drivers and easy testing.
- Easy to use
- powerful database transaction management capabilities

SPRING BOOT FLOW ARCHITECTURE



Why SQL ?

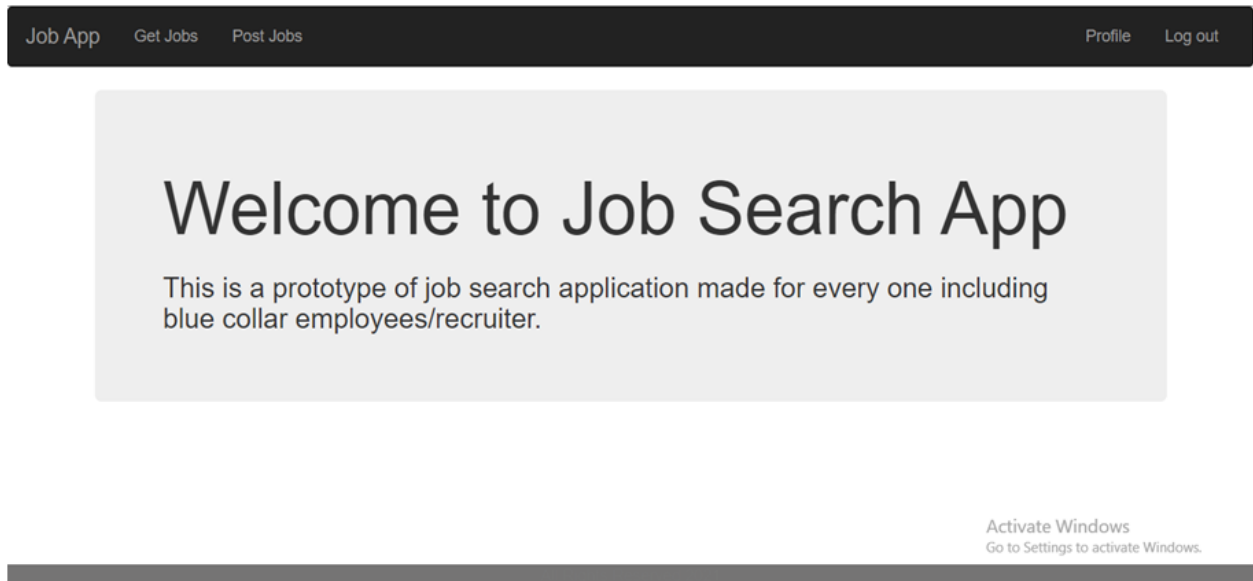
SQL stands for Structured Query Language. It is used for storing and managing data in relational database management systems (RDMS). It is a standard language for the Relational Database System. It enables a user to create, read, update and delete relational databases and tables.

Flow chart



Application Screenshots

This is about us page of our webapp. It is the first page which will open. We can then login or signup.



This is the signup page. It contains various fields and there are filters for some fields. Eg : first name, email address or password can't be empty.

[Job App](#) [About](#) [Log In](#) [Sign up](#)

First Name

Second Name

Email Address

Phone no

Password

Sign Up

Activate Windows
Go to Settings to activate Windows.

This is the login page. When username and password are entered, it will check whether password is correct for corresponding username or not by making an api call. If the password is not correct or the username doesn't exist, this screen will show a message "Incorrect username or password" otherwise it will redirect to the about us page.

[Job App](#) [About](#) [Log In](#) [Sign up](#)

Username

Password

Log In

Activate Windows
Go to Settings to activate Windows.

Username

Password

Log In

Wrong username or password

Activate Windows
Go to Settings to activate Windows.

This is the logout page.

Logged Out

Activate Windows
Go to Settings to activate Windows.

This is the user profile page. When profile option is clicked from the top menu, then the user profile is fetched from the backend.

[Job App](#) [Get Jobs](#) [Post Jobs](#) [Profile](#) [Log out](#)

First Name	Aditi
Second Name	.
Emailid	aditigupta6226@gmail.com
Phone no	77390

Activate Windows
Go to Settings to activate Windows.

This is the “post job” page. Company name , job profile are necessary fields. When the post button is clicked it will show the message “Added successfully”. If anything goes wrong it will show the message “Error”.

[Job App](#) [Get Jobs](#) [Post Jobs](#) [Profile](#) [Log out](#)

Company Name	<input type="text" value="Amazon"/>
Job Profile	<input type="text" value="Data Engineer"/>
Location	<input type="text" value="Banglore"/>
Link	<input type="text" value="https://www.amazon.jobs/en"/>
Salary	<input type="text" value="1300000"/>
Number of Employees	<input type="text" value="50"/>

Post

Added Successfully

Activate Windows
Go to Settings to activate Windows.

This is “getjob” page. There are two dropdowns for Job profile and company. We can apply either of two filters, both or none.

If neither of two filters are applied and search is clicked, then it will show all jobs.

[Job App](#)[Get Jobs](#)[Post Jobs](#)[Profile](#)[Log out](#)

Job Profile

Company Name

Search

Profile	Company	location	Salary	Number of Employees	Link
Software Developer	Amazon	Banglore	1500000	10	Open
Data Engineer	Amazon	Banglore	1300000	50	Open
Software Developer	Infosys	Noida	500000	80	Open

Activate Windows
Go to Settings to activate Windows.

[Job App](#)[Get Jobs](#)[Post Jobs](#)[Profile](#)[Log out](#)

Data Engineer

Company Name

Search

Profile	Company	location	Salary	Number of Employees	Link
Data Engineer	Amazon	Banglore	1300000	50	Open

Activate Windows
Go to Settings to activate Windows.

Software Developer

Company Name

Search

Profile	Company	location	Salary	Number of Employees	Link
Software Developer	Amazon	Banglore	1500000	10	Open
Software Developer	Infosys	Noida	500000	80	Open

Activate Windows
Go to Settings to activate Windows.

Job Profile

Amazon

Search

Profile	Company	location	Salary	Number of Employees	Link
Software Developer	Amazon	Banglore	1500000	10	Open
Data Engineer	Amazon	Banglore	1300000	50	Open

Activate Windows

Software Developer

Amazon

Search

Profile	Company	location	Salary	Number of Employees	Link
Software Developer	Amazon	Banglore	1500000	10	Open

Activate Windows
Go to Settings to activate Windows.

Software Developer

Amazon

Search

Profile	Company	location		Number of Employees	Link
Software Developer	Amazon	Banglore	1500000	10	Open

Activate Windows
Go to Settings to activate Windows.

Software Developer

Amazon

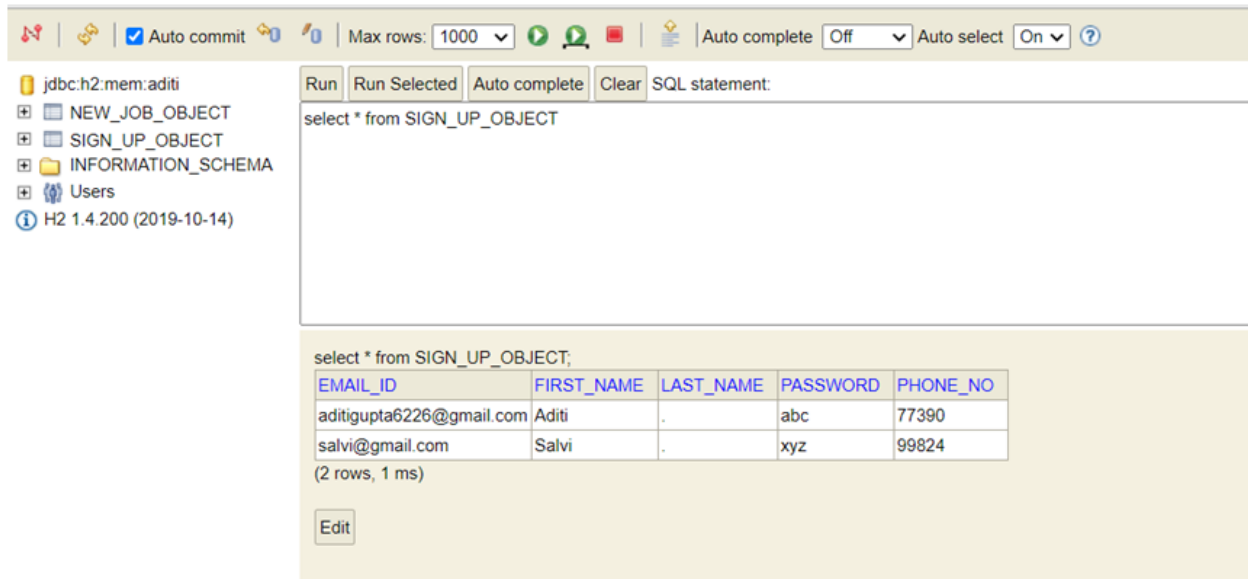
Search

Profile	Company		Salary	Number of Employees	Link
Software Developer	Amazon	Banglore	1500000	10	Open

Activate Windows
Go to Settings to activate Windows.

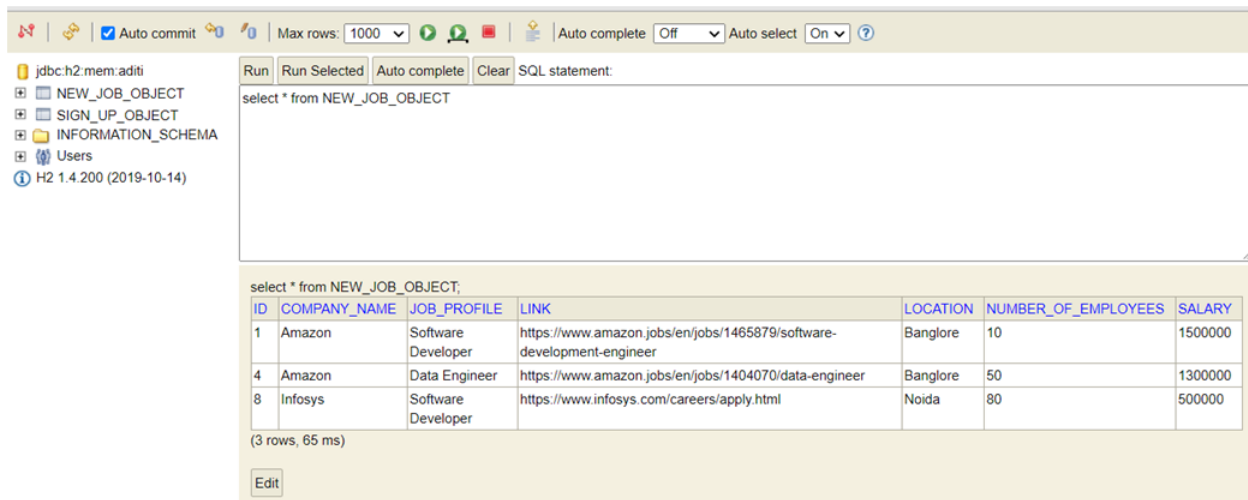
This is h2 console.

There are two tables NewJobObject and SignUpObject.



The screenshot shows the H2 console interface. On the left, the database structure is listed: jdbc:h2:mem:aditi, NEW_JOB_OBJECT, SIGN_UP_OBJECT, INFORMATION_SCHEMA, Users, and H2 1.4.200 (2019-10-14). The main area displays a SQL statement: `select * from SIGN_UP_OBJECT`. Below the statement, the results are shown in a table with 5 columns: EMAIL_ID, FIRST_NAME, LAST_NAME, PASSWORD, and PHONE_NO. The results are: aditigupta6226@gmail.com, Aditi, ., abc, 77390; and salvi@gmail.com, Salvi, ., xyz, 99824. The status bar indicates (2 rows, 1 ms).

EMAIL_ID	FIRST_NAME	LAST_NAME	PASSWORD	PHONE_NO
aditigupta6226@gmail.com	Aditi	.	abc	77390
salvi@gmail.com	Salvi	.	xyz	99824



The screenshot shows the H2 console interface. On the left, the database structure is listed: jdbc:h2:mem:aditi, NEW_JOB_OBJECT, SIGN_UP_OBJECT, INFORMATION_SCHEMA, Users, and H2 1.4.200 (2019-10-14). The main area displays a SQL statement: `select * from NEW_JOB_OBJECT`. Below the statement, the results are shown in a table with 6 columns: ID, COMPANY_NAME, JOB_PROFILE, LINK, LOCATION, NUMBER_OF_EMPLOYEES, and SALARY. The results are: 1, Amazon, Software Developer, https://www.amazon.jobs/en/jobs/1465879/software-development-engineer, Bangalore, 10, 1500000; 4, Amazon, Data Engineer, https://www.amazon.jobs/en/jobs/1404070/data-engineer, Bangalore, 50, 1300000; and 8, Infosys, Software Developer, https://www.infosys.com/careers/apply.html, Noida, 80, 500000. The status bar indicates (3 rows, 65 ms).

ID	COMPANY_NAME	JOB_PROFILE	LINK	LOCATION	NUMBER_OF_EMPLOYEES	SALARY
1	Amazon	Software Developer	https://www.amazon.jobs/en/jobs/1465879/software-development-engineer	Banglore	10	1500000
4	Amazon	Data Engineer	https://www.amazon.jobs/en/jobs/1404070/data-engineer	Banglore	50	1300000
8	Infosys	Software Developer	https://www.infosys.com/careers/apply.html	Noida	80	500000

Frontend Screenshots:

- **GetJob page**

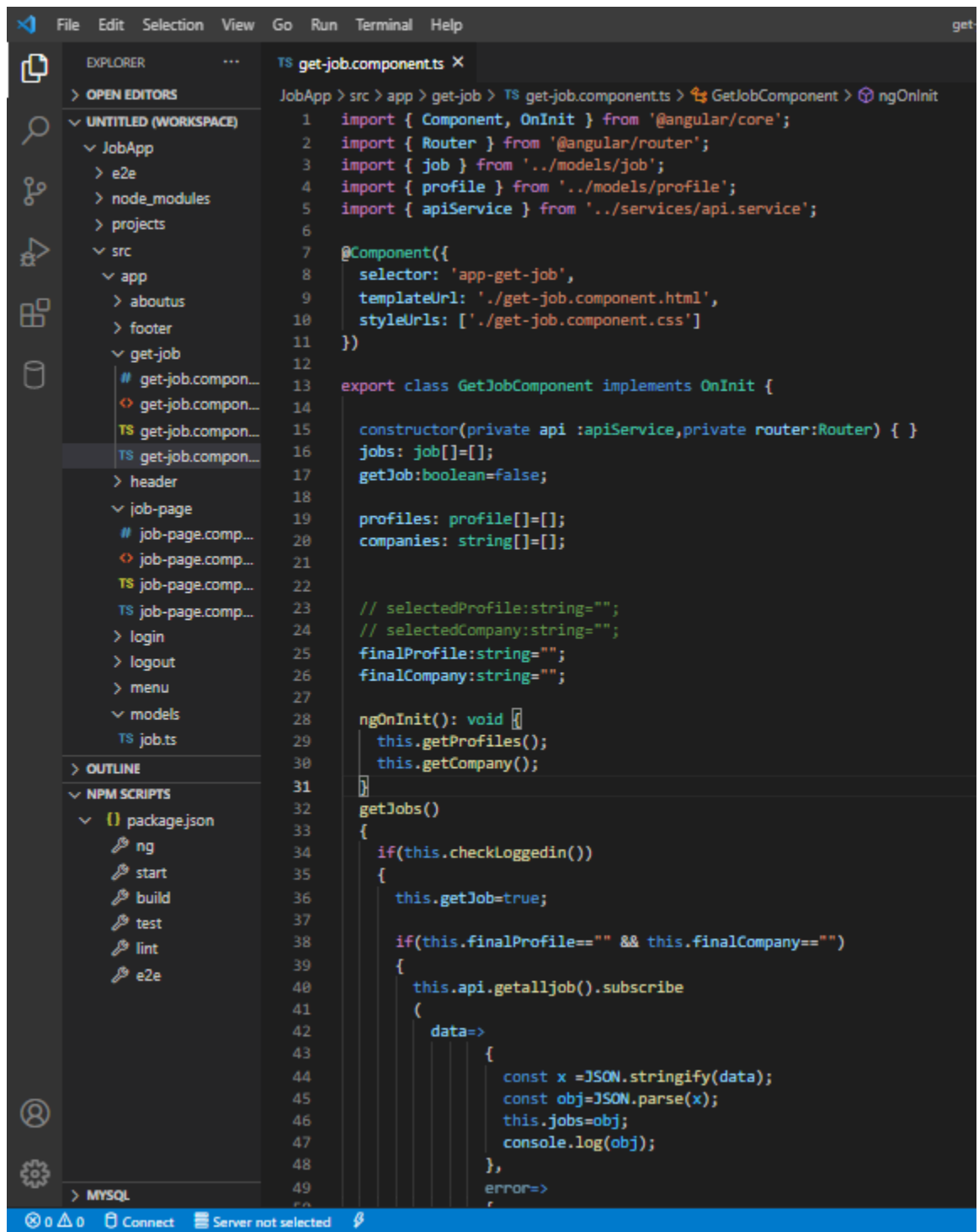
HTML

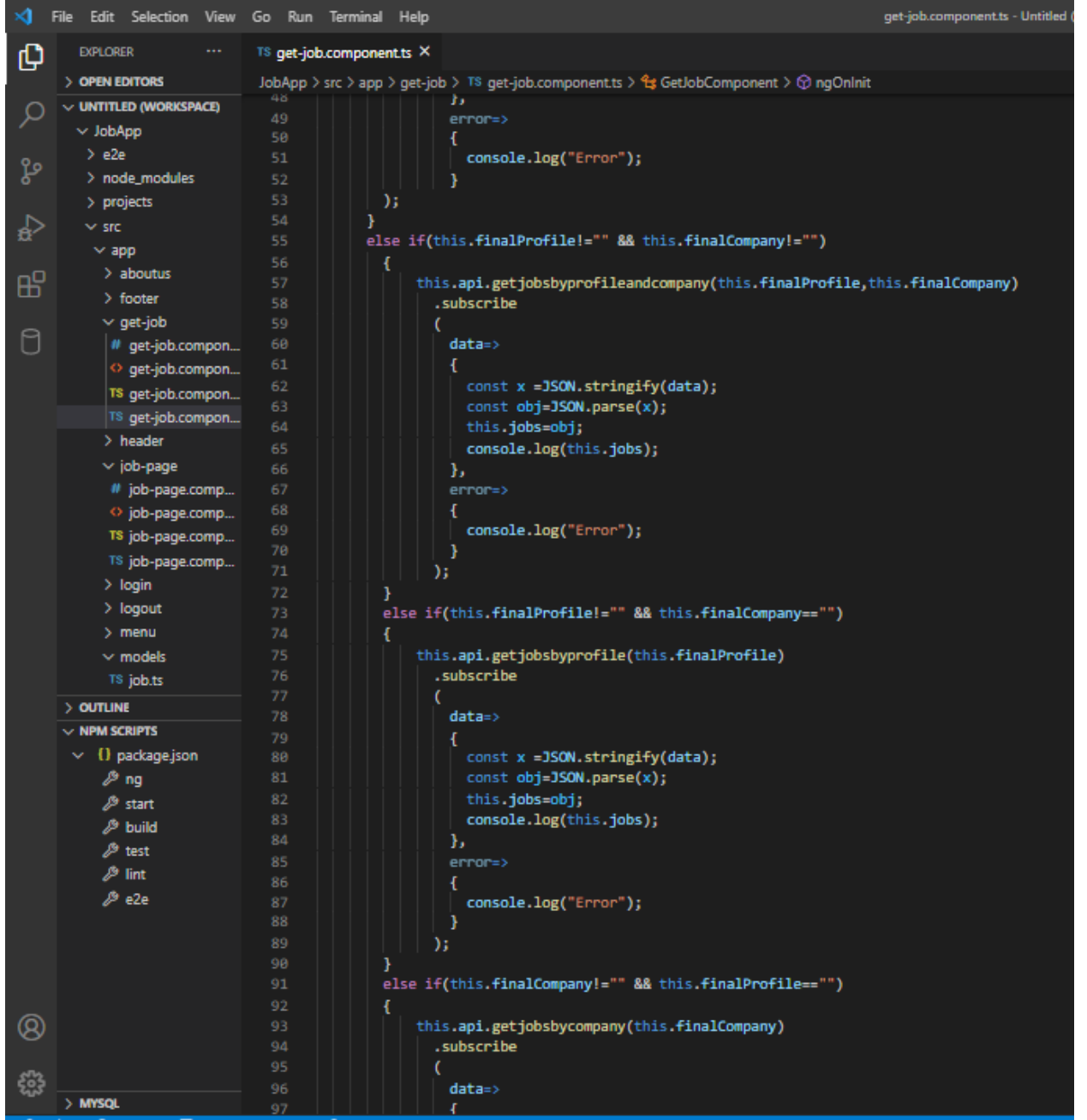
```
1 <div class="block">
2
3   <div class="box">
4     <select [(ngModel)]="finalProfile" class="dropdown">
5       <option value="" disabled selected>Job Profile</option>
6       <option *ngFor="let string of profiles" [value]="string">
7         {{string}}
8       </option>
9     </select>
10
11     <select [(ngModel)]="finalCompany" class="dropdown">
12       <option value="" disabled selected>Company Name</option>
13       <option *ngFor="let string of companies" [value]="string">
14         {{string}}
15       </option>
16     </select>
17
18     <button class="btn btn-primary button" label="Get" icon="pi pi-check" iconPos="left" (click)="getJobs()">
19       Search </button>
20   </div>
21
22   <p-table [value]="jobs" *ngIf="getJob">
23     <ng-template pTemplate="header">
24       <tr>
25         <th class="header">Profile</th>
26         <th class="header">Company</th>
27         <th class="header">Location</th>
28         <th class="header">Salary</th>
29         <th class="header">Number of Employees</th>
30         <th class="header">Link</th>
31       </tr>
32     </ng-template>
33     <ng-template pTemplate="body" let-jobs>
34       <tr>
35         <td>{{jobs.jobProfile}}</td>
36         <td>{{jobs.companyName}}</td>
37         <td>{{jobs.location}}</td>
38         <td>{{jobs.salary}}</td>
39         <td>{{jobs.numberEmployees}}</td>
40         <td><a href="{{jobs.link}}">Open</a></td>
41       </tr>
42     </ng-template>
43   </p-table>
44 </div>
```

CSS

```
1 .block {
2   margin-top: 50px;
3 }
4
5 .dropdown {
6   width: 150px;
7   height: 34px;
8   margin-right: 20px;
9   margin-left: 30px;
10  margin-bottom: 20px;
11 }
12
13 .box {
14   margin-left: 180px;
15 }
16
17 .button {
18   margin-left: 25px;
19   width: 100px;
20 }
```

Component

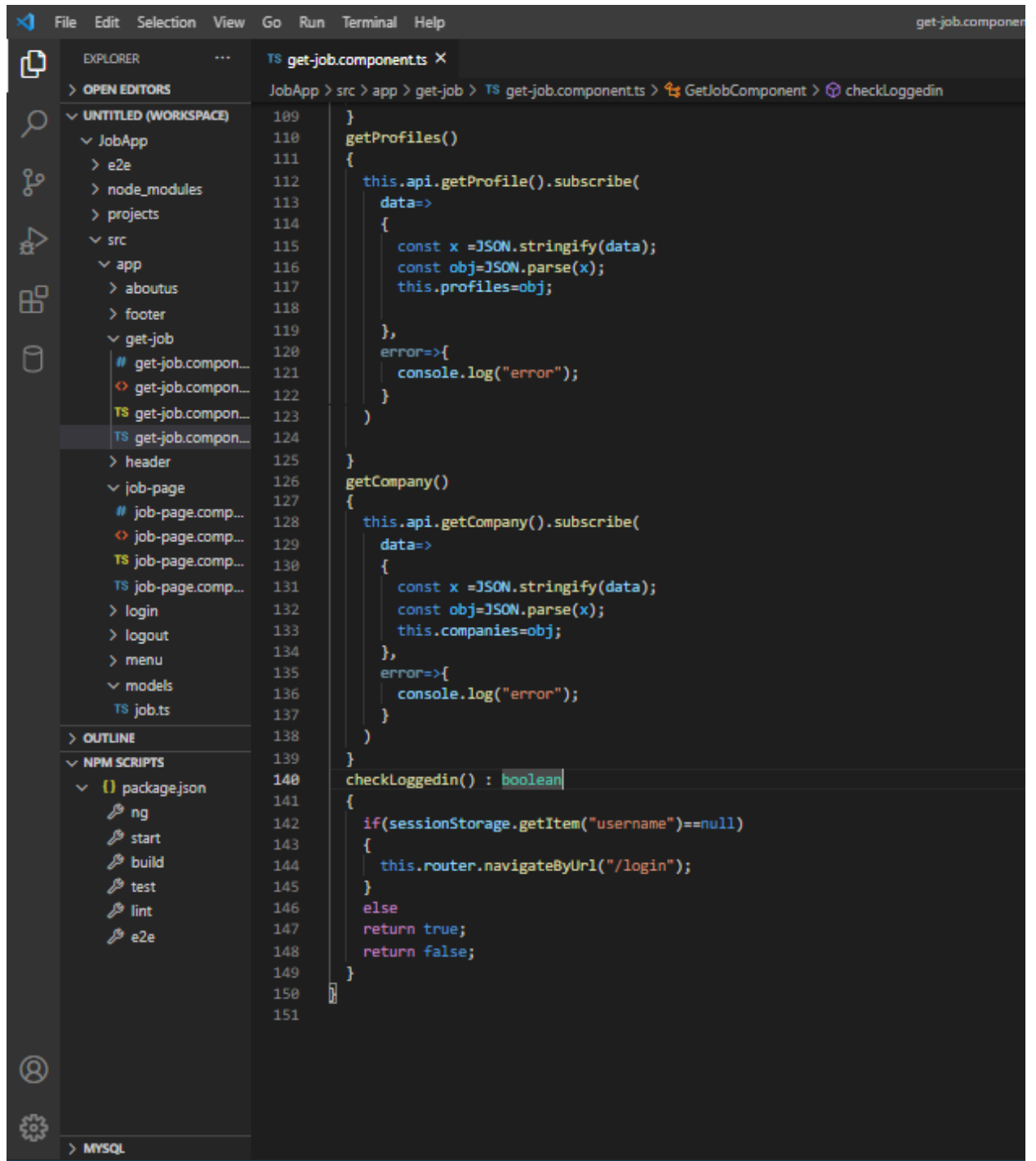




VS Code Explorer and Editor view showing a TypeScript file named `get-job.component.ts` in the `src/app/get-job` directory. The Explorer sidebar on the left shows the project structure, including `JobApp`, `src`, `app`, `get-job`, and `models`. The Editor view on the right displays the code for `get-job.component.ts`, which includes methods like `data=>`, `error=>`, `getProfiles()`, `getCompany()`, and `checkLoggedIn()`. The code uses `JSON.stringify` and `JSON.parse` for data handling and `console.log` for debugging. The `checkLoggedIn()` method checks for a session item in `localStorage` and navigates to the login page if it is null.

```
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
```

```
data=>
{
  const x =JSON.stringify(data);
  const obj=JSON.parse(x);
  this.jobs=obj;
},
error=>
{
  console.log("Error");
}
);
}
}
getProfiles()
{
  this.api.getProfile().subscribe(
    data=>
    {
      const x =JSON.stringify(data);
      const obj=JSON.parse(x);
      this.profiles=obj;
    },
    error=>{
      console.log("error");
    }
  )
}
getCompany()
{
  this.api.getCompany().subscribe(
    data=>
    {
      const x =JSON.stringify(data);
      const obj=JSON.parse(x);
      this.companies=obj;
    },
    error=>{
      console.log("error");
    }
  )
}
checkLoggedIn() : boolean
{
  if(sessionStorage.getItem("username")==null)
  {
    this.router.navigateByUrl("/login");
  }
}
```



Service layer(API call)

```

18 }
19 getalljob()
20 {
21     return this.http.get("http://localhost:8080/getalljobs");
22 }
23 getjobsbyprofile(profile :string)
24 {
25     return this.http.get('http://localhost:8080/getjobsbyprofile/${profile}');
26 }
27 getjobsbyprofileandcompany(profile :string,company:string)
28 {
29     return this.http.get('http://localhost:8080/getjobs/${profile}/${company}');
30 }
31 getjobsbycompany(company :string)
32 {
33     return this.http.get('http://localhost:8080/getjobsbycompany/${company}');
34 }

```

- **Post Job**

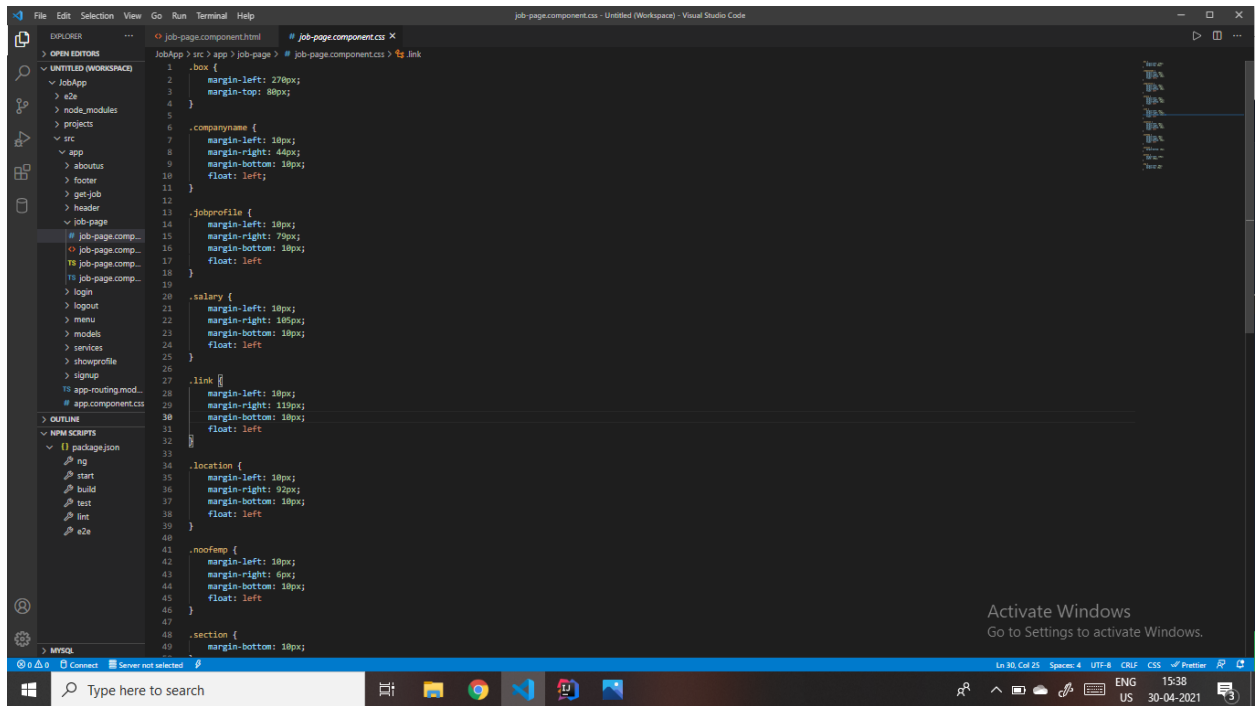
HTML

```

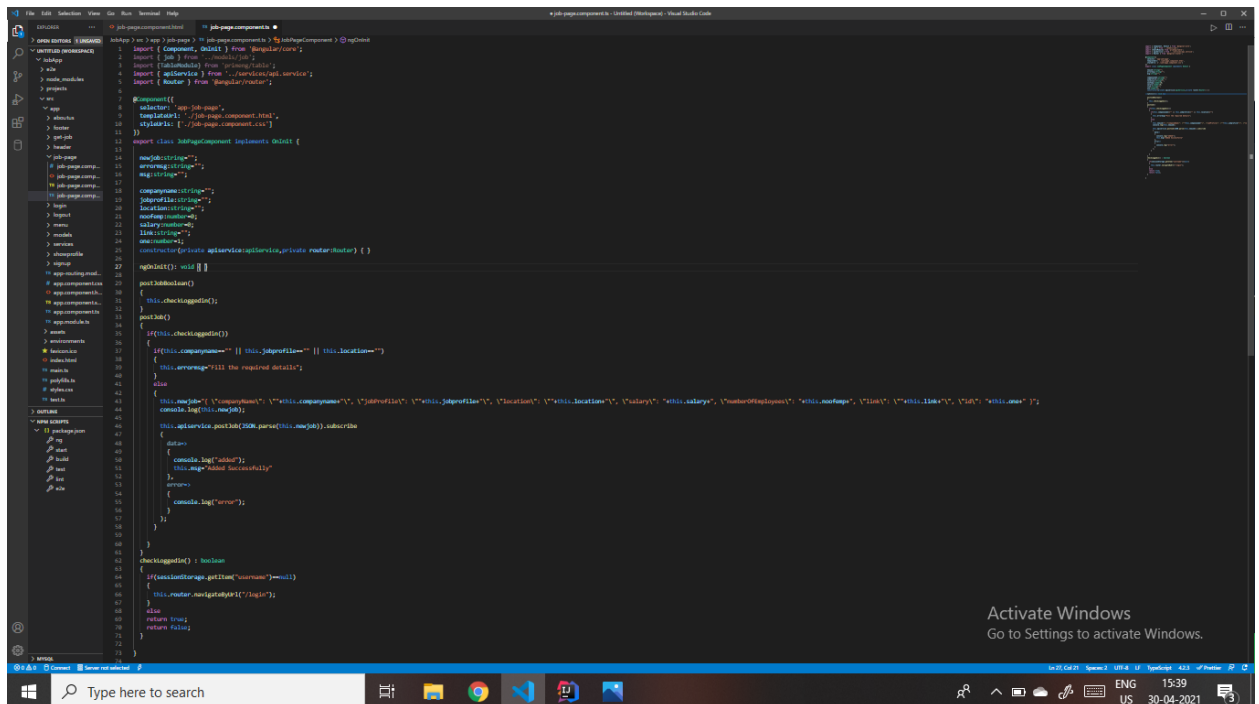
job-page.component.html X
JobApp > src > app > job-page > job-page.component.html > div.box > div.section > div
1 <div class="box">
2   <div class="section">
3     <div class="companyname">Company Name</div>
4     <div><input type="text" [(ngModel)]="companyname" pInputText /></div>
5   </div>
6   <div class="section">
7     <div class="jobprofile">Job Profile</div>
8     <div><input type="text" [(ngModel)]="jobprofile" pInputText /></div>
9   </div>
10  <div class="section">
11    <div class="location">Location</div>
12    <div><input type="text" [(ngModel)]="location" pInputText /></div>
13  </div>
14  <div class="section">
15    <div class="link">Link</div>
16    <div><input type="text" [(ngModel)]="link" pInputText /></div>
17  </div>
18  <div class="section">
19    <div class="salary">Salary</div>
20    <div><input type="text" [(ngModel)]="salary" pInputText /></div>
21  </div>
22  <div class="section">
23    <div class="noofemp">Number of Employees</div>
24    <div><input type="text" [(ngModel)]="noofemp" pInputText /></div>
25  </div>
26  <div>
27    <button class="btn btn-primary button" label="Post" icon="pi pi-check" iconPos="left" (click)="postJob()">
28      Post
29    </button>
30  </div>
31  <div class="msg">{{msg}}</div>
32  <div class="msg">{{msg}}</div>
33  <div class="msg">{{msg}}</div>
34 </div>

```

CSS



Component



Service(API Call)

```

    postJob(s:string)
    {
        return this.http.post("http://localhost:8080/postjobs",s,{responseType : 'text'});
    }
}

getProfile()

```

- Login

HTML

The screenshot shows the Visual Studio Code editor with a workspace titled "login.component.html - Untitled (Workspace)". The Explorer sidebar on the left shows a project structure with folders like "job-page", "login", "logout", "menu", "models", "services", and "showprofile". The "login" folder is expanded, showing files like "login.component.html" and "login.component.ts". The main editor area displays the HTML content of "login.component.html".

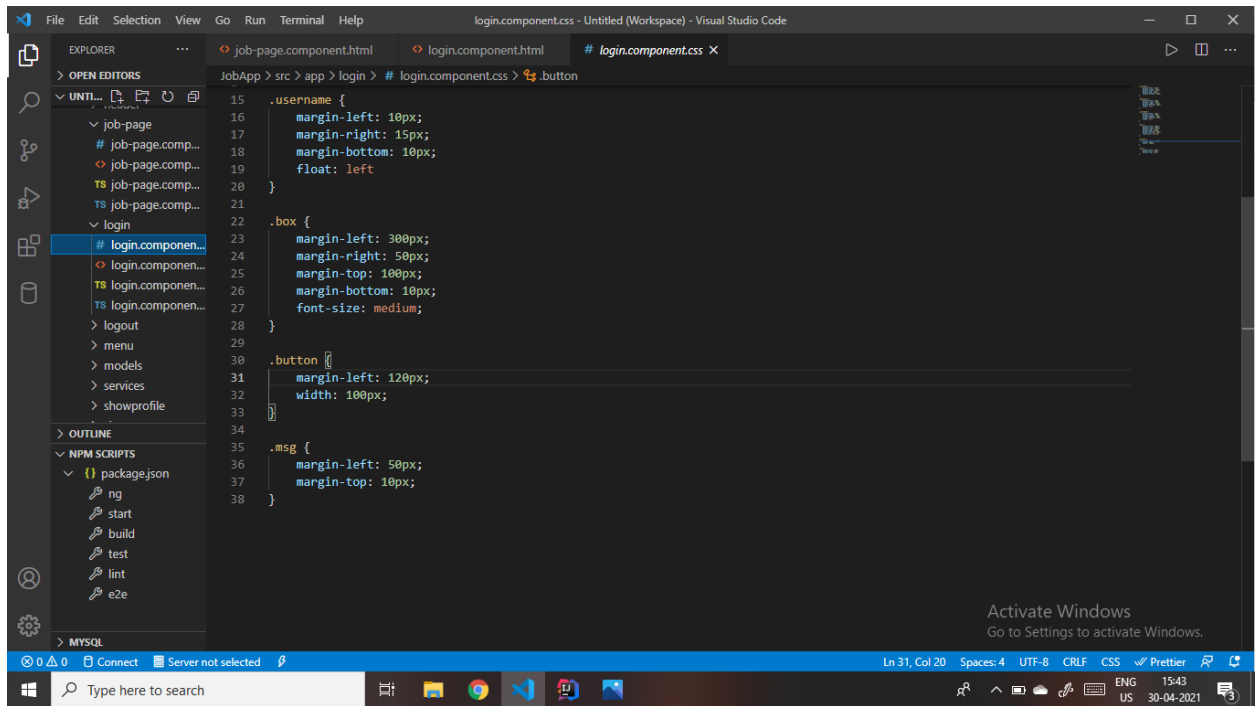
```

1 <div class="box">
2   <div class="section">
3     <div class="username">Username</div>
4     <div><input type="email" [(ngModel)]="myusername" pInputText /></div>
5   </div>
6   <div class="section">
7     <div class="password">Password</div>
8     <div><input type="password" [(ngModel)]="mypassword" pInputText /></div>
9   </div>
10
11   <div>
12     <button class="btn btn-primary button" label="log In" icon="pi pi-check" iconPos="left" (click)="onClick()">
13       Log In
14     </button>
15   </div>
16   <div class="msg">{{msg}}</div>
17 </div>

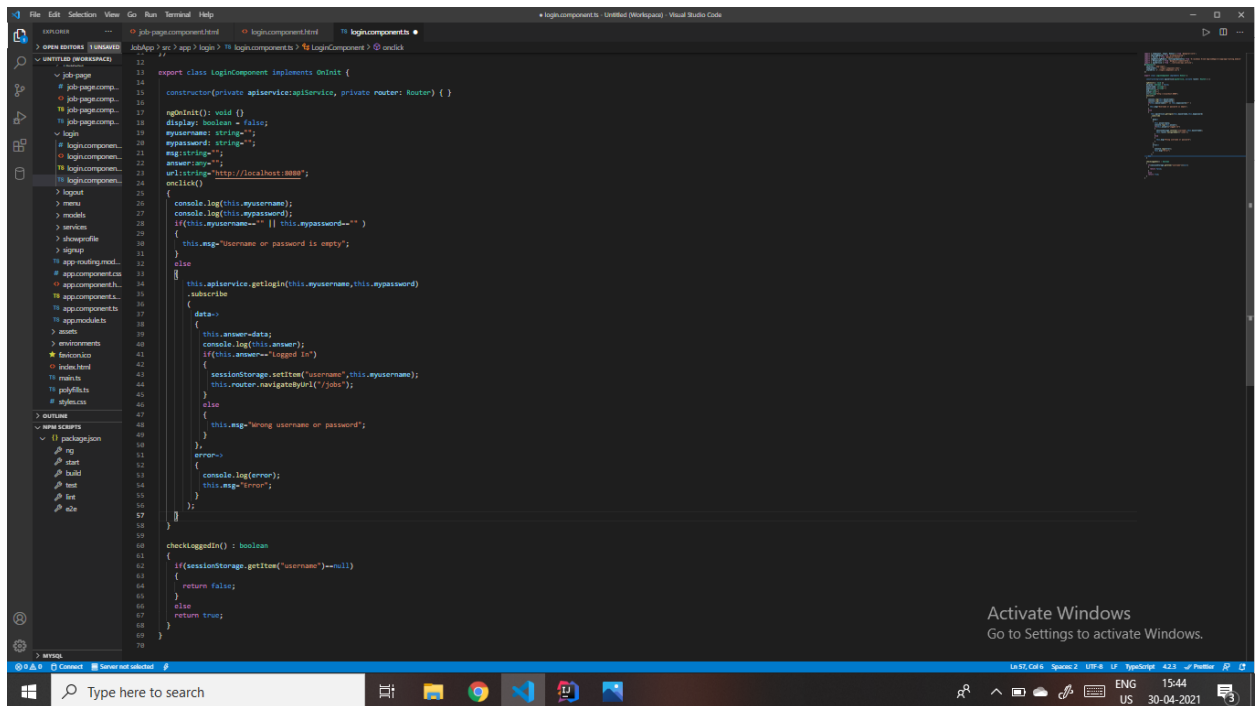
```

The status bar at the bottom indicates the current line and column (Ln 17, Col 7), the number of spaces (4), the encoding (UTF-8), the line ending (CRLF), the language (HTML), and the formatter (Prettier). The Windows taskbar is visible at the bottom with various icons and the system clock showing 15:43 on 30-04-2021.

CSS



Component



Service(API Call)

```

    }
    getlogin(username:string,password:string) : Observable<any>
    {
        return this.http.get(`http://localhost:8080/login/${username}/${password}`, {responseType: 'text'});
    }
}

```

- **Signup**

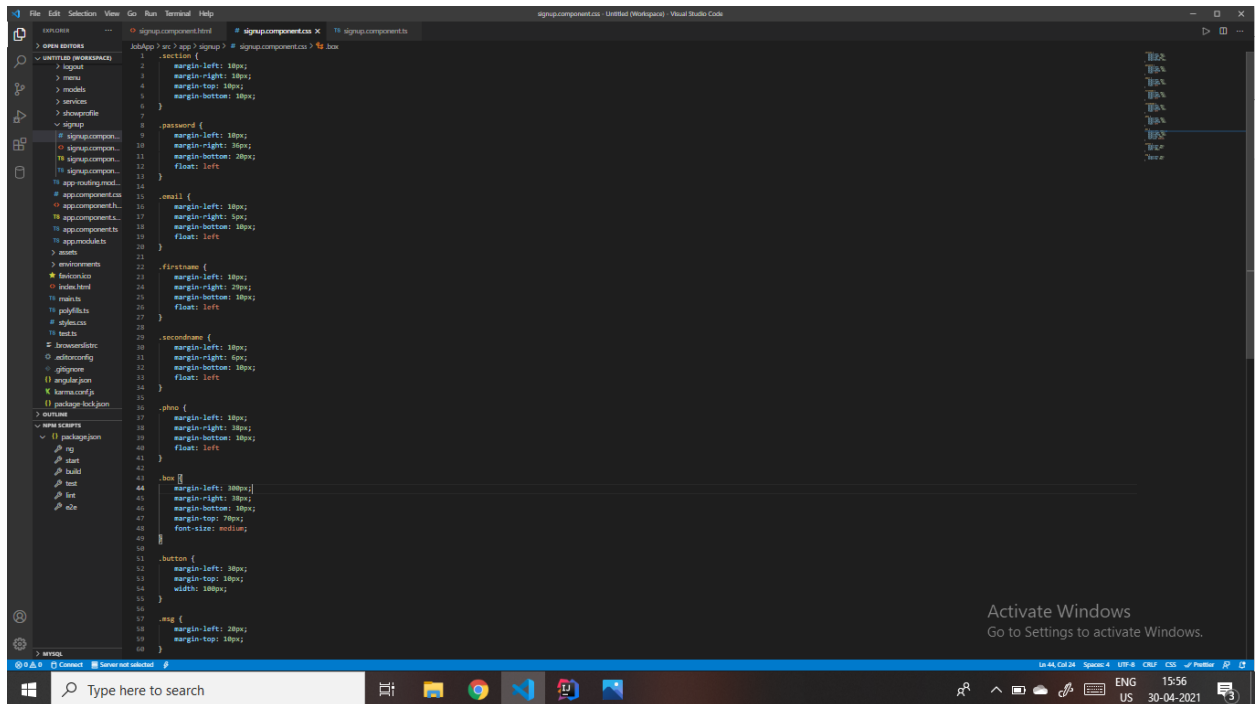
HTML

```

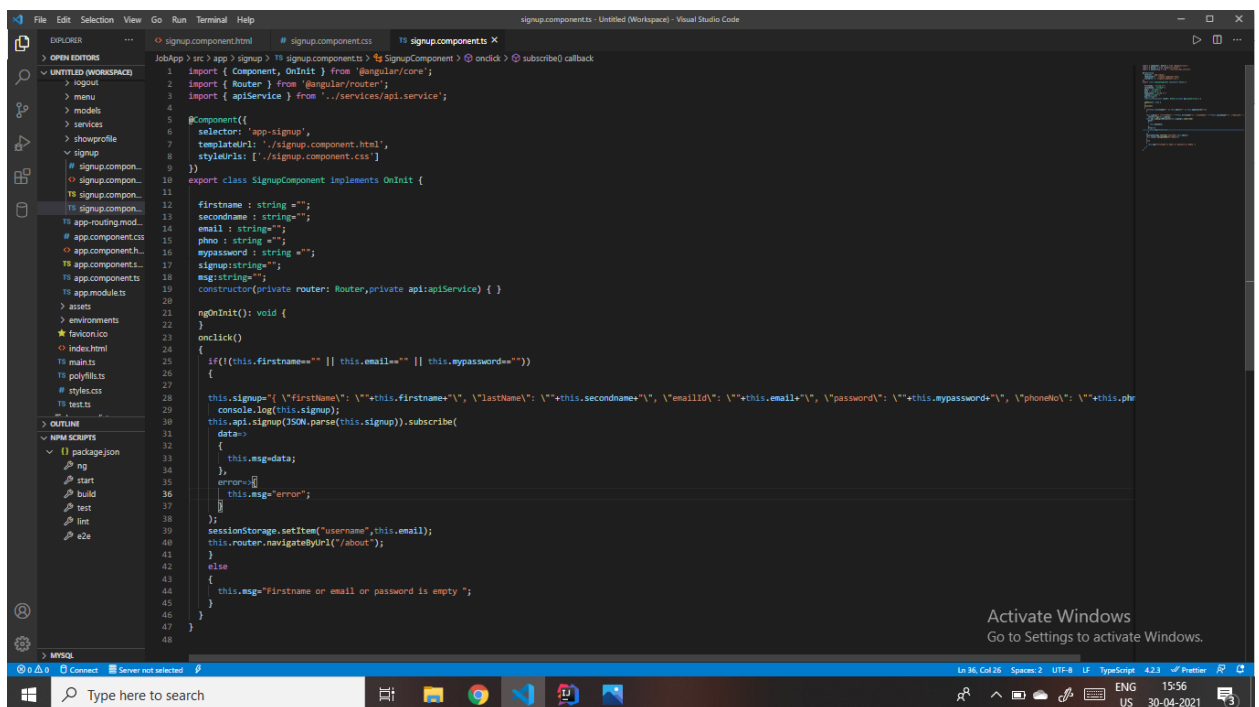
1 <div class="box">
2   <div class="section">
3     <div class="firstname">First Name</div>
4     <div><input type="text" [(ngModel)]="firstname" pInputText /></div>
5   </div>
6   <div class="section">
7     <div class="secondname">Second Name</div>
8     <div><input type="text" [(ngModel)]="secondname" pInputText /></div>
9   </div>
10  <div class="section">
11    <div class="email">Email Address</div>
12    <div><input type="email" [(ngModel)]="email" pInputText /></div>
13  </div>
14  <div class="section">
15    <div class="phno">Phone no</div>
16    <div><input type="text" [(ngModel)]="phno" pInputText /></div>
17  </div>
18  <div class="section">
19    <div class="password">Password</div>
20    <div><input type="text" [(ngModel)]="mypassword" pInputText /></div>
21  </div>
22  <div>
23    <button class="btn btn-primary button" label="Sign up" icon="pi pi-check" iconPos="left" (click)="onlick()">
24      Sign Up
25    </button>
26  </div>
27  <div class="msg">{{msg}}</div>
28
29 </div>
30
31
32 <!-- <div class="p-field p-grid">
33   <label for="firstname3" class="p-col-fixed" style="width:100px">Firstname</label>

```

CSS



Component



Service(API Call)


```

    }
    signup(s:string)
    {
        return this.http.post("http://localhost:8080/signup",s,{responseType:'text'});
    }
}

```

- Profile Dropdown Service (API Call)

```

38     }
39     getProfile()
40     {
41         return this.http.get("http://localhost:8080/getprofiles");
42     }
}

```

- Company DropDown Service (API Call)

```

getCompany()
{
    return this.http.get("http://localhost:8080/getcompanies")
}
getUserProfile()

```

- User Profile Service (API Call)

```

}
getUserProfile()
{
    return this.http.get("http://localhost:8080/getuserprofile/${sessionStorage.getItem("username")}");
}

```

Backend Screenshots:

- Controller

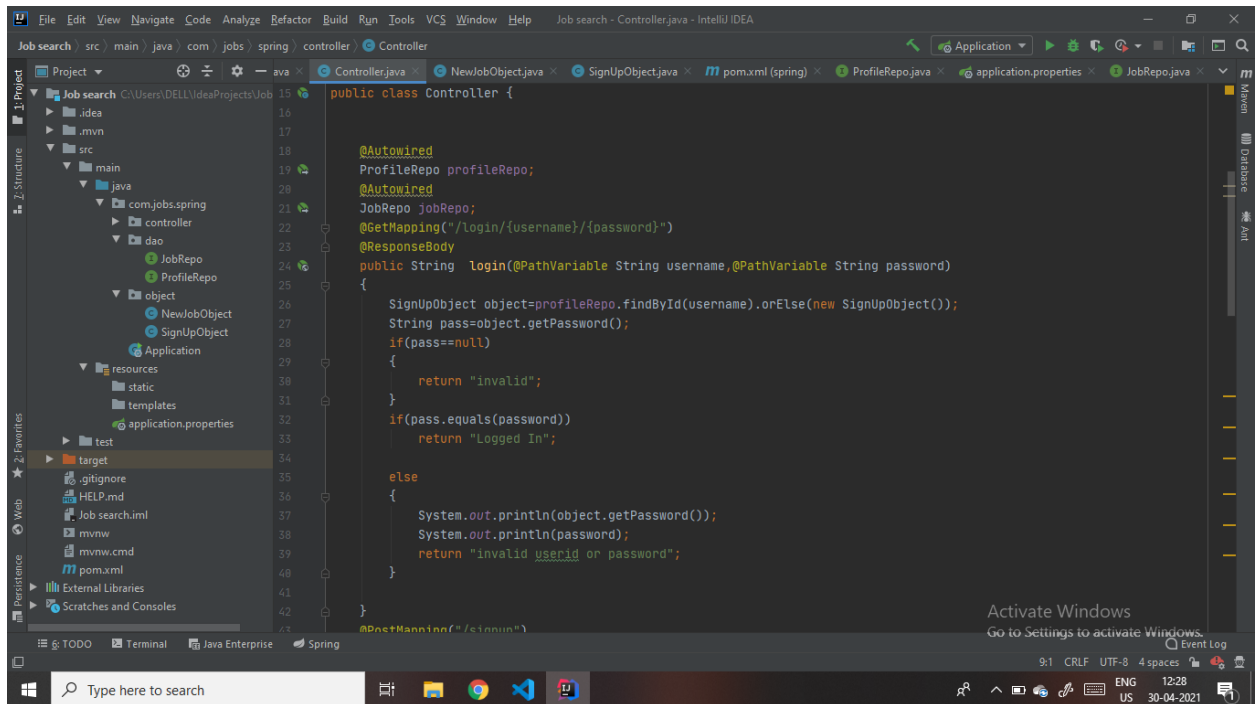
1. Sign Up

```

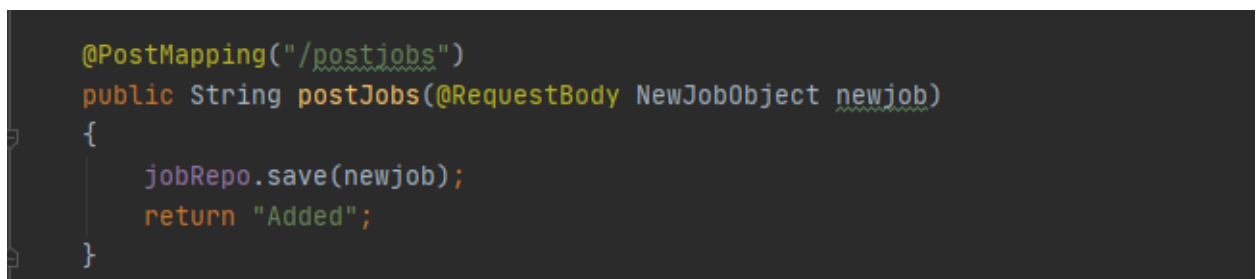
}
@PostMapping("/signup")
public String signup(@RequestBody SignUpObject signupobject)
{
    profileRepo.save(signupobject);
    return "success";
}

```

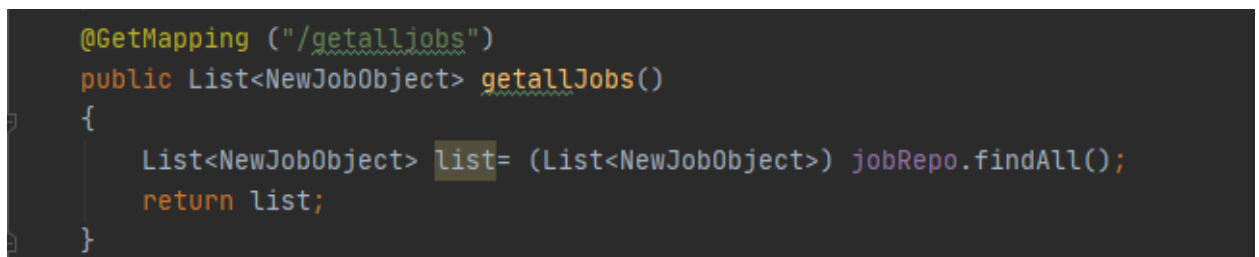
2. Login



3. Post Job



4. Get all the jobs



5. Get all the jobs by profile as filter

```

@GetMapping("/getjobsbyprofile/{profile}")
public List<NewJobObject> getJobsByProfile(@PathVariable String profile)
{
    List<NewJobObject> list=jobRepo.findByJobProfile(profile);
    return list;
}

```

6. Get all the jobs by company name as filter

```

@GetMapping("/getjobsbycompany/{company}")
public List<NewJobObject> getJobsByCompany(@PathVariable String company)
{
    List<NewJobObject> list=jobRepo.findByCompany(company);
    return list;
}

```

7. Get all the jobs by profile and company name as filter

```

@GetMapping("/getjobs/{profile}/{company}")
public List<NewJobObject> getJobsByProfileandCompany(@PathVariable String profile, @PathVariable String company)
{
    List<NewJobObject> list=jobRepo.findByJobProfileandCompany(profile,company);
    return list;
}

```

8. Get all the jobs of different profile present in backend for dropdown

```

@GetMapping("/getprofiles")
public List<String> getProfile()
{
    List<String> list=jobRepo.findallprofiles();
    return list;
}

```

9. Get all the companies present in backend for dropdown

```

    @GetMapping("/getcompanies")
    public List<String> getCompanies()
    {
        List<String> list=jobRepo.findallcompanies();
        return list;
    }

```

10. Get the user profile

```

    @GetMapping("/getuserprofile/{username}")
    public Optional<SignUpObject> getuserprofile(@PathVariable String username)
    {
        return profileRepo.findById(username);
    }

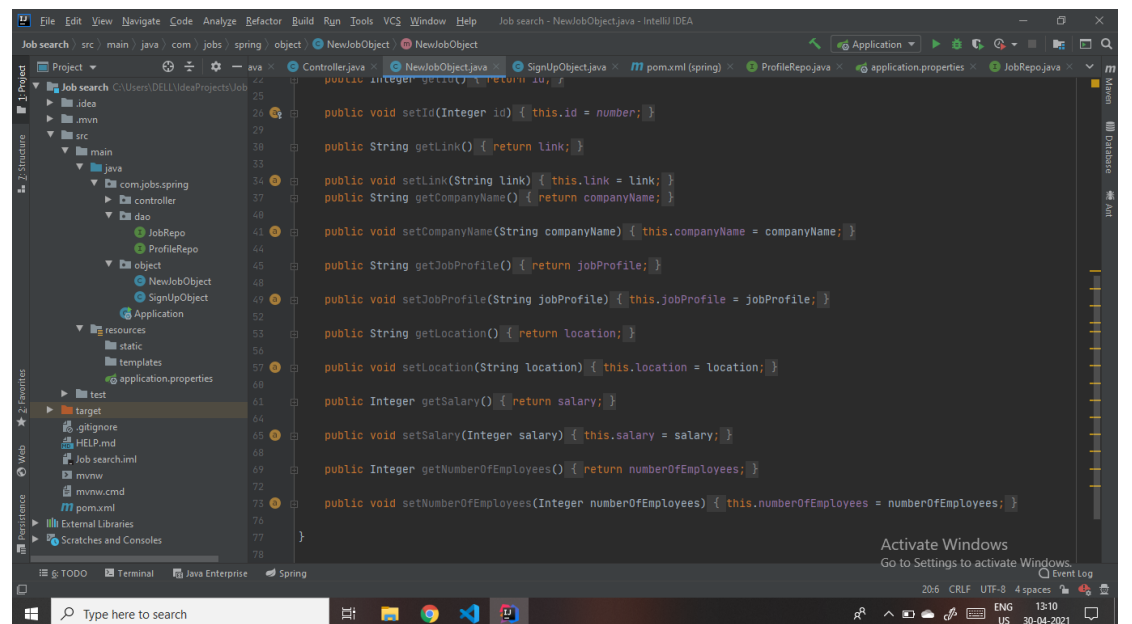
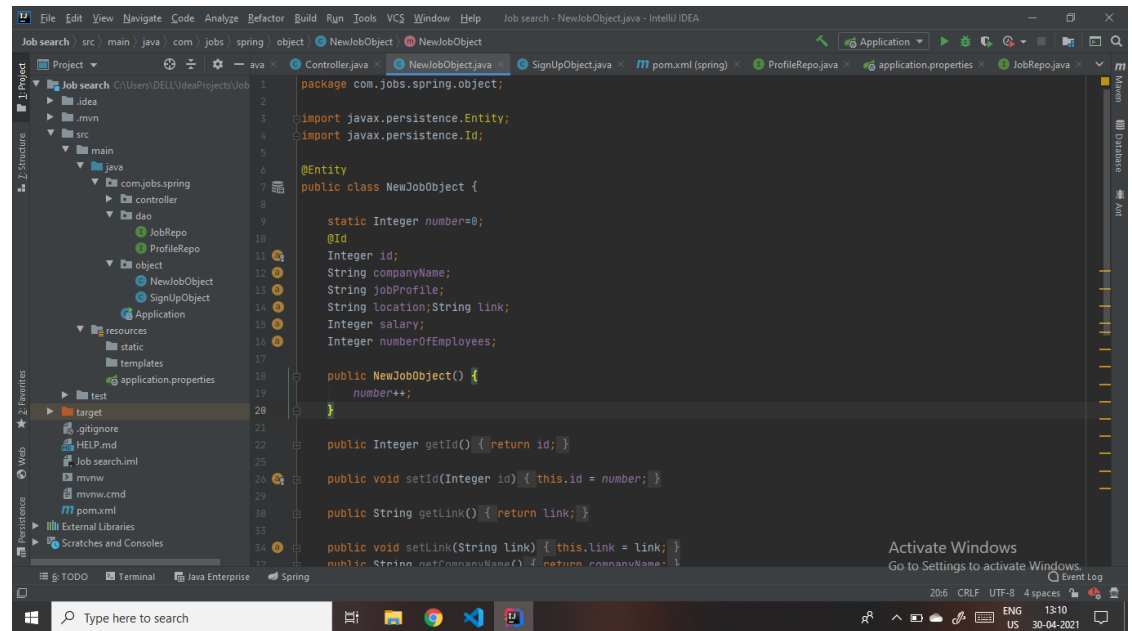
```

- **Object class**

1. NewJobObject: When a new job is added an object of this NewJobObject class will be created.

Fields of this class:

- Id : this is primary key.
- Company Name
- Number of Employees
- Salary
- Field
- Location



2.SignUpObject: When a new user sign up an object of this SignUpObject class will be created.

Fields of this class:

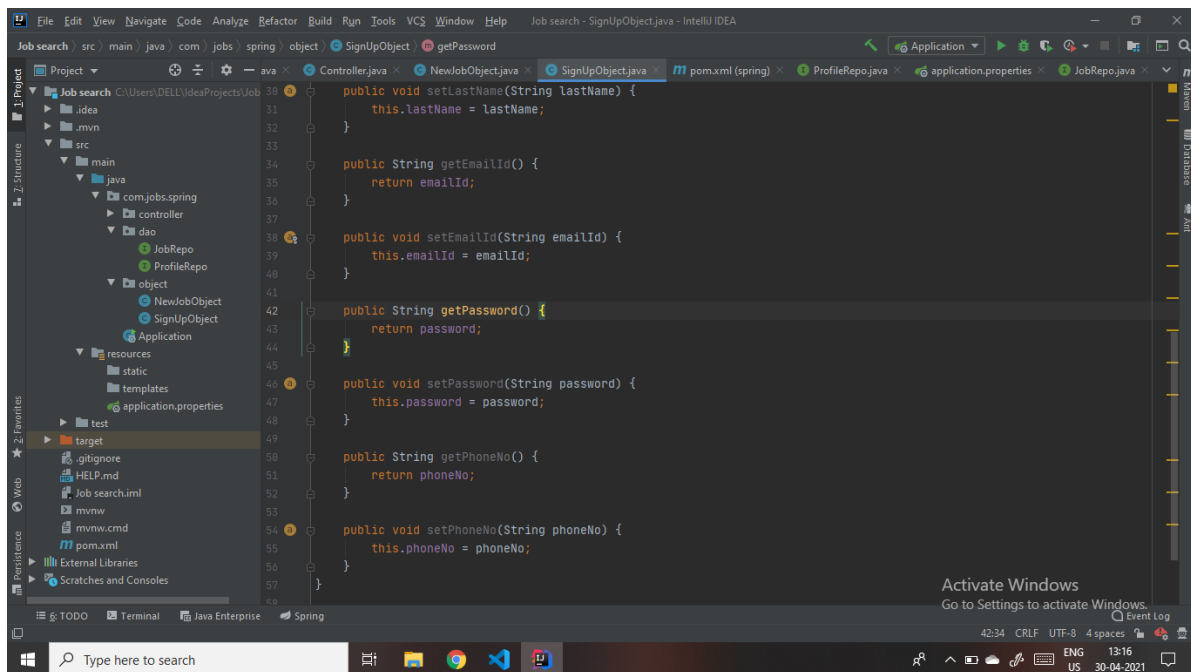
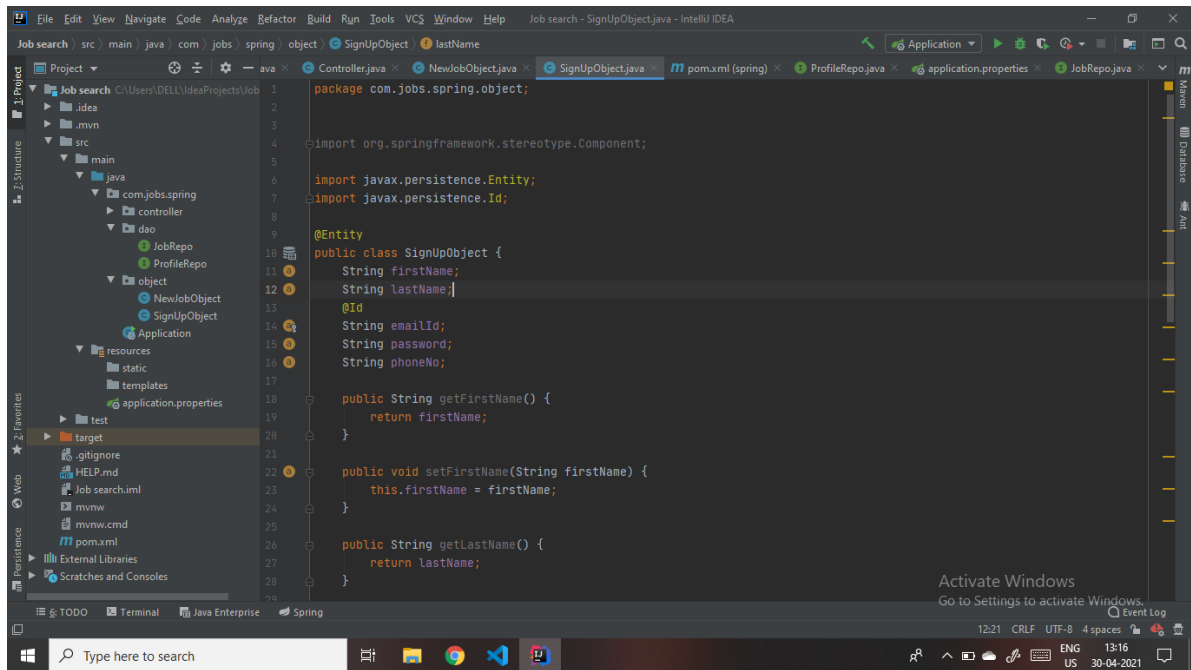
-FirstName

-LastName

-Email id : This is primary key

-Password

-Phone number



Reference

1. <https://angel.co/>
2. <https://www.firstnaukri.com/>
3. <https://www.primefaces.org/primeng/>
4. <https://www.udemy.com/course/full-stack-application-development-with-spring-boot-and-angular/>
5. <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>

Conclusion

This webapp will help everyone to get a job easily and conveniently.

As any person can add job, we will have a large number of jobs from all sectors easily.

Thank you