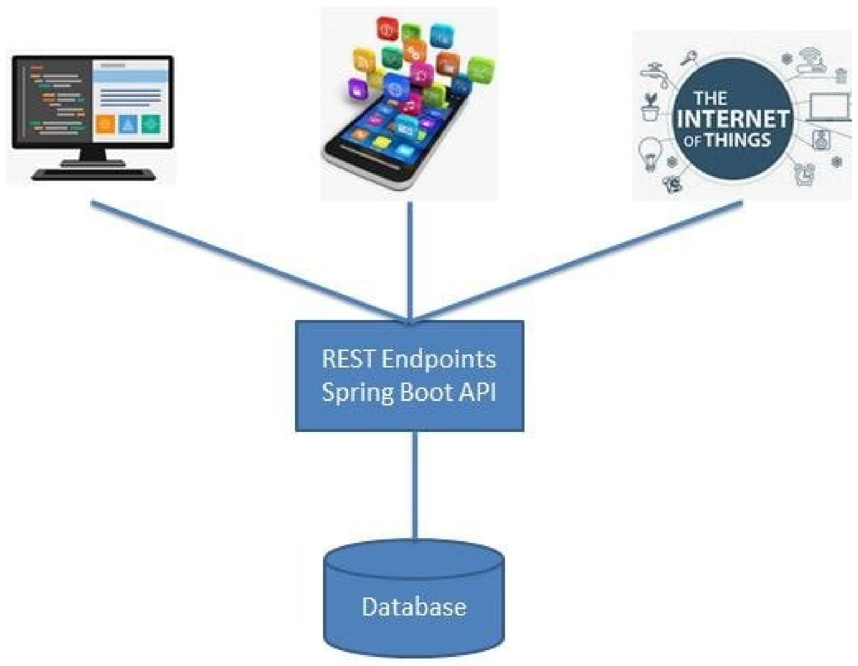
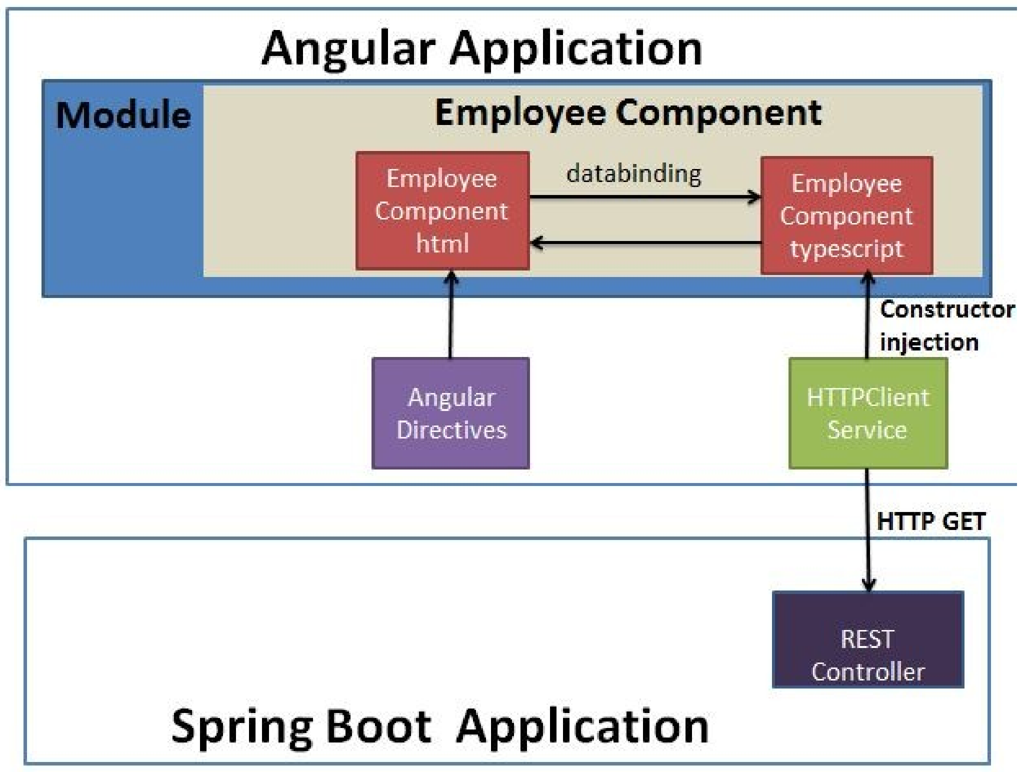
Spring Angular Security Integration step by step

Angular spring boot CRUD





spring boot Rest

step 1: choose dependencies: devTools, actutator, jpa, mysql, web, security

step 2: paste in application.properties file

```
server.servlet.context-path=/empapp
server.port=8080

spring.datasource.url=jdbc:mysql://localhost:3306/kr_jdbc?useSSL=false
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=create
logging.level.org.springframework.web: DEBUG
logging.level.org.hibernate: ERROR
spring.jpa.show-sql=true
```

step 3: create domain

```
@Entity
@Table(name="emp_table")
public class Employee {
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int id;
	private String name;
	private int age;
}


@Repository
public interface EmpRepo extends CrudRepository<Employee, Integer>{

}
```

step 4: create service

```
public interface EmployeeService {
	public List<Employee> getAll();
	public Employee getEmployeeById(int id);
	public Employee save(Employee emp);
	public Employee delete(int empId);
	public Employee update(int empId, Employee emp);
}



@Service
@Transactional
public class EmployeeServiceImpl implements EmployeeService {

	@Autowired
	private EmpRepo empRepo;

	@Override
	public List<Employee> getAll() {
		return (List<Employee>) empRepo.findAll();
	}

	@Override
	public Employee getEmployeeById(int id) {
		return empRepo.findById(id).orElseThrow(EmployeeNotFoundException:: new);
	}

	@Override
	public Employee save(Employee emp) {
		return empRepo.save(emp);
	}

	@Override
	public Employee delete(int empId) {
		Employee employeeToDelete=getEmployeeById(empId);
		empRepo.delete(employeeToDelete);
		return employeeToDelete;
	}

	@Override
	public Employee update(int empId, Employee emp) {
		Employee employeeToUpdate=getEmployeeById(empId);
		employeeToUpdate.setName(emp.getName());
		employeeToUpdate.setAge(emp.getAge());
		return empRepo.save(employeeToUpdate);
	}
}



public class EmployeeNotFoundException extends RuntimeException{
}
```

step 5: create rest controller

```
@RestController
public class EmpRestController {

	@Autowired
	private EmployeeService employeeService;


	@GetMapping(path="employee", produces=MediaType.APPLICATION_JSON_VALUE)
	public List<Employee> allEmployees(){
		return employeeService.getAll();
	}




@GetMapping(path="employee/{id}",produces=MediaType.APPLICATION_JSON_VALUE )
	public Employee getEmployeeById(@PathVariable(name="id")int id){
		return employeeService.getEmployeeById(id);
	}

	@PostMapping(path="employee",produces=MediaType.APPLICATION_JSON_VALUE,
				consumes=MediaType.APPLICATION_JSON_VALUE )
	public Employee addEmployee(@RequestBody Employee employee){
		return employeeService.save(employee);
	}


@PutMapping(path="employee/{id}",produces=MediaType.APPLICATION_JSON_VALUE,
				consumes=MediaType.APPLICATION_JSON_VALUE )
	public Employee updateEmployee(@PathVariable(name="id") int id,   @RequestBody
Employee emp){
		return  employeeService.update(id, emp);

	}



@DeleteMapping(path="employee/{id}",produces=MediaType.APPLICATION_JSON_VALUE)
	public Employee deleteEmplloyee(@PathVariable(name="id") int id){
		return employeeService.delete(id);
	}
}
```

```
@SpringBootApplication
public class DemoApplication implements CommandLineRunner{

	@Autowired
	private EmployeeService empService;

	public static void main(String[] args) {
		SpringApplication.run(DemoApplication.class, args);
	}

	@Override
	public void run(String... args) throws Exception {
		System.out.println("rec are saved...");
		empService.save(new Employee("raj", 33));
		empService.save(new Employee("ekta", 30));
		empService.save(new Employee("gunika", 10));
		empService.save(new Employee("keshav", 5));
	}

}
```

Enable core, create filter to enable cors

```
@CrossOrigin(origins = "http://localhost:4200")
```

or

```
@Component
public class CORSFilter implements Filter {

    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
            throws IOException, ServletException {
        HttpServletResponse response = (HttpServletResponse) res;
        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Access-Control-Allow-Methods", "POST, GET, PUT, OPTIONS,
DELETE");
        response.setHeader("Access-Control-Max-Age", "3600");
        response.setHeader("Access-Control-Allow-Headers", "X-requested-with, Content-Type");
        chain.doFilter(req, res);
    }
    public void init(FilterConfig filterConfig) {
    }

    public void destroy() {
    }

}
```

Spring boot angular integration:

Angular crud application:
----

1. configure bootstrap:

install bootstrap:

```
sudo npm install bootstrap --save
```

refer link to :style.css

```
@import "~bootstrap/dist/css/bootstrap.min.css"
```

check if working:

```
  <h1 class="text-center">{{ title }} app is running!</h1>

<router-outlet></router-outlet>
```

alternative ways:
----

angular.json:

```
"styles": [
              "./node_modules/bootstrap/dist/css/bootstrap-grid.css",
              "src/styles.css"
            ],
```

angular.cli.json

```
npm install bootstrap@4.0.0-alpha.6 --save

 "../node_modules/bootstrap/dist/css/bootstrap.min.css"

 "../node_modules/jquery/dist/jquery.min.js",
"../node_modules/bootstrap/dist/js/bootstrap.min.js"
```

Generate employee class:

```
ng g class employee
```

```
export class Employee {
    id: number;
    name: string;
    age: number
}
```

create employee component:

```
ng g c employee
```

Now display all employee :

```
<table class = "table table-striped">
    <thead>
        <tr>
            <th> First Name</th>
            <th> Last Name </th>
            <th> Email Id</th>
            <th> Actions </th>
        </tr>
    </thead>
    <tbody>
        <tr *ngFor = "let employee of employees" >
            <td> {{ employee.id }} </td>
            <td> {{ employee.name }} </td>
            <td> {{ employee.age }} </td>
            <td>
                <button (click) = "updateEmployee(employee.id)" class = "btn btn-info">
Update</button>
                <button (click) = "deleteEmployee(employee.id)" class = "btn btn-danger" style="margin-
left: 10px"> Delete</button>
```

```
                <button (click) = "employeeDetails(employee.id)" class = "btn btn-info" style="margin-
left: 10px"> View</button>
            </td>
        </tr>
    </tbody>
```

Now populate hard coded employees:

```
export class EmployeeComponent implements OnInit {

  employees: Employee[];

  constructor() { }

  ngOnInit() {
    this.employees=[
      {
        "id":1,
        "name":"rajeev",
        "age":40
      },
      {
        "id":2,
        "name":"ekta",
        "age":40
      },
      {
        "id":3,
        "name":"gunika",
        "age":15
      }
    ];
  }

}
```

put table in proper position:

```
<div class="text-center">
  <app-employee></app-employee>
</div>
```

create service to fetch with rest endpoint:

```
 ng g s employee
```

Now we need to register HttpClientModule in app.module.ts

```
import { HttpClientModule } from '@angular/common/http';

imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
```

install rxjs:

```
npm install rxjs@6 rxjs-compat@6 -–save
```

employee service:
----

http://localhost:8080/empapp/api/employee

```
import { Injectable } from '@angular/core';

import { from, Observable } from 'rxjs';
import {HttpClient} from '@angular/common/http'
import { Employee } from './employee';

@Injectable({
  providedIn: 'root'
})
export class EmployeeService {
  private baseURL="http://localhost:8080/empapp/employee";
  constructor(private httpClient: HttpClient) { }

  getEmployeesList(): Observable<Employee[]>{
    return this.httpClient.get<Employee[]>(`${this.baseURL}`);
  }
}
```

employee component:

```
import { Component, OnInit } from '@angular/core';
import { Employee } from '../employee';
import { EmployeeService } from '../employee.service';

@Component({
  selector: 'app-employee',
  templateUrl: './employee.component.html',
  styleUrls: ['./employee.component.css']
})
export class EmployeeComponent implements OnInit {

  employees: Employee[];

  constructor(private employeeService: EmployeeService) { }

  ngOnInit() {
    this.getEmployees();
  }
  private getEmployees(){
    this.employeeService.getEmployeesList().subscribe(data=>{
            this.employees=data;
    });
  }

}
```

configure route in app-routing-module.ts

```
const routes: Routes = [
  {path:'employees', component: EmployeeComponent},
  {path:'', redirectTo:'employees', pathMatch:'full'}
];
```

use route in app.component.html

```
<div class="text-center">
  <router-outlet></router-outlet>
</div>
```

Routing and Navigation:
----
add to app.component.html:

```
<nav class="navbar navbar-expand-sm bg-primary navbar-dark">
  <ul class = "navbar-nav">
      <li class = "nav-item">
          <a routerLink="employees" routerLinkActive="active" class="nav-link" >Employee List</a>
      </li>
  </ul>
</nav>


<div class="text-center">
  <router-outlet></router-outlet>
</div>

<footer class = "footer">
  <div class = "container">
      <span>All Rights Reserved 2020 @HCL training</span>
  </div>
</footer>
```

add to style.css

```
@import "~bootstrap/dist/css/bootstrap.min.css";

.footer {
    position: absolute;
    bottom: 0;
    width:100%;
    height: 40px;
    background-color: blue;
    text-align: center;
    color: white;
}
```

Creating add employee ui:

```
 ng g c create-employee
```

now create rounte for create employee:

```
const routes: Routes = [
  {path:'employees', component: EmployeeComponent},
  {path:'create-employee', component: CreateEmployeeComponent},
  {path:'', redirectTo:'employees', pathMatch:'full'}
];
```

update app.component.html

```
<li class = "nav-item">
          <a routerLink="create-employee" routerLinkActive="active" class="nav-link" >Add
Employee</a>
</li>
```

create an empty object of emp to attacehd with form:

```
export class CreateEmployeeComponent implements OnInit {

  employee: Employee = new Employee();
  constructor() { }

  ngOnInit() {
  }

 onSubmit(){
    console.log(this.employee);
  }
}
```

crete the form:

```
<div class="col-md-6 offset-md-3">
    <h3> Create Employee </h3>
    <form (ngSubmit) = "onSubmit()">

        <div class="form-group">
            <label> Name</label>
            <input type="text" class ="form-control" id = "name"
                [(ngModel)] = "employee.name" name = "name">
        </div>

        <div class="form-group">
            <label> Age </label>
            <input type="text" class ="form-control" id = "age"
                [(ngModel)] = "employee.age" name = "age">
        </div>
        <button class = "btn btn-success" type ="submit">Submit</button>

    </form>
    </div>
```

Note:

=> Error:Can't bind to 'ngModel' since it isn't a known property of 'input'. (

=> Before using ngModel directive in a two way data binding,
we must import the FormModule and add it to the NgModules import list

=> import forms modules into app.module.ts

```
import { FormsModule } from '@angular/forms';
```

and put to imported list:

```
imports: [
 BrowserModule,
FormsModule
]
```

Now add createEmployee to EmployeeService:

```
export class EmployeeService {

  private baseURL = "http://localhost:8080/api/v1/employees";

  constructor(private httpClient: HttpClient) { }


  createEmployee(employee: Employee): Observable<Object>{
    return this.httpClient.post(`${this.baseURL}`, employee);
  }

}
```

Now code CreateEmployeeComponent:

```
import { Component, OnInit } from '@angular/core';

import { Router } from '@angular/router';
import { Employee } from '../employee';
import { EmployeeService } from '../employee.service';

@Component({
  selector: 'app-create-employee',
  templateUrl: './create-employee.component.html',
  styleUrls: ['./create-employee.component.css']
})


export class CreateEmployeeComponent implements OnInit {

  employee: Employee = new Employee();
  constructor(private employeeService: EmployeeService,private router: Router) { }

  ngOnInit(): void {
  }

  saveEmployee(){
    this.employeeService.createEmployee(this.employee).subscribe( data =>{
      console.log(data);
      this.goToEmployeeList();
    },
    error => console.log(error));
  }

  goToEmployeeList(){
    this.router.navigate(['/employees']);
  }

  onSubmit(){
    console.log(this.employee);
    this.saveEmployee();
  }
}
```

Now update employee:

```
ng g c update-employee
```

now create rounte for update employee:

```
----------------------------------
const routes: Routes = [
  {path:'employees', component: EmployeeComponent},
  {path:'create-employee', component: CreateEmployeeComponent},
  {path:'update-employee/:id', component: UpdateEmployeeComponent},
  {path:'', redirectTo:'employees', pathMatch:'full'}


];
```

add service for update employee:

```
export class EmployeeService {

  private baseURL = "http://localhost:8080/api/v1/employees";

  constructor(private httpClient: HttpClient) { }

 //.......

  updateEmployee(id: number, employee: Employee): Observable<Object>{
    return this.httpClient.put(`${this.baseURL}/${id}`, employee);
  }
 //......
}
```

now add update link and update mtthod :EmployeeComponent

```
import { Component, OnInit } from '@angular/core';
import { Employee } from '../employee';
import { EmployeeService } from '../employee.service';
import { Router } from '@angular/router';
@Component({
  selector: 'app-employee',
  templateUrl: './employee.component.html',
  styleUrls: ['./employee.component.css']
})
export class EmployeeComponent implements OnInit {

  employees: Employee[];

  constructor(private employeeService: EmployeeService, private  router: Router ) { }

  ngOnInit() {
    this.getEmployees();
  }
  private getEmployees(){
    this.employeeService.getEmployeesList().subscribe(data=>{
       this.employees=data;
    });
  }

  updateEmployee(id: number){
    console.log(`-----------`)
    this.router.navigate(['update-employee', id]);
  }

}
```

now update UpdateEmployeeComponent so that record populate automatically:

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { Employee } from '../employee';
import { EmployeeService } from '../employee.service';

@Component({
  selector: 'app-update-employee',
  templateUrl: './update-employee.component.html',
  styleUrls: ['./update-employee.component.css']
})
export class UpdateEmployeeComponent implements OnInit {
  id: number;
  employee: Employee = new Employee();
  constructor(private employeeService: EmployeeService,private route: ActivatedRoute) { }

  ngOnInit(): void {
    this.id=this.route.snapshot.params['id'];
   this.employeeService.getEmployeeById(this.id).subscribe(data=>{
    this.employee=data;
    }, error=>console.log(error))
  }


}
```

Now code for update and route back to showing all records:

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { Employee } from '../employee';
import { EmployeeService } from '../employee.service';

@Component({
  selector: 'app-update-employee',
  templateUrl: './update-employee.component.html',
  styleUrls: ['./update-employee.component.css']
})

export class UpdateEmployeeComponent implements OnInit {
  id: number;
  employee: Employee = new Employee();
  constructor(private employeeService: EmployeeService,
    private route: ActivatedRoute, private router: Router) { }

  ngOnInit(): void {
    this.id=this.route.snapshot.params['id'];
   this.employeeService.getEmployeeById(this.id).subscribe(data=>{
    this.employee=data;
    }, error=>console.log(error))
  }

  onSubmit(){
    this.employeeService.updateEmployee(this.id, this.employee)
    .subscribe(data=> {
        this.goToEmployeeList();
    }, error=> console.log(error))
  }
goToEmployeeList(){
    this.router.navigate(['/employees']);
  }
}
```

Delete employee:
----

add service for delete Employee:

```
export class EmployeeService {

 //...........
  deleteEmployee(id: number): Observable<Object>{
    return this.httpClient.delete(`${this.baseURL}/${id}`);
  }
}
```

update EmployeeComponent for deletion:

```
export class EmployeeComponent implements OnInit {

  //....


  deleteEmployee(id: number){
    this.employeeService.deleteEmployee(id).subscribe(data=>{
      this.getEmployees();
      console.log(data);
    })
  }


  employeeDetails(id: number){
    this.router.navigate(['employee-details', id]);
  }
}
```

Creating employee details components:
----
step : create employee-details componenet

```
ng g c employee-details
```

step 2: register employee-details in routes:

```
const routes: Routes = [
  {path:'employees', component: EmployeeComponent},
  {path:'create-employee', component: CreateEmployeeComponent},
  {path:'update-employee/:id', component: UpdateEmployeeComponent},
  {path:'employee-details/:id', component: EmployeeDetailsComponent},
  {path:'', redirectTo:'employees', pathMatch:'full'}


];
```

EmployeeDetailsComponent code:

```
import { ActivatedRoute, Router } from '@angular/router';
import { Employee } from '../employee';
import { EmployeeService } from '../employee.service';

export class EmployeeDetailsComponent implements OnInit {

  id: number
  employee: Employee=new Employee();

  constructor(private route: ActivatedRoute,
     private employeService: EmployeeService) { }

  ngOnInit(): void {
    this.id = this.route.snapshot.params['id'];

    this.employee = new Employee();
    this.employeService.getEmployeeById(this.id).subscribe( data => {
      this.employee = data;
    });
  }
}
```

employee-detail html page:

```
<h3> View Employee Details</h3>
<div>
    <div>
        <label> <b> First Name: </b></label> {{employee.id}}
    </div>
    <div>
        <label> <b> Last Name: </b></label> {{employee.name}}
    </div>
    <div>
        <label> <b> Email Id: </b></label> {{employee.age}}
    </div>
</div>
```

update EmployeeComponent for deletion:

```
export class EmployeeComponent implements OnInit {

  //....


  employeeDetails(id: number){
    this.router.navigate(['employee-details', id]);
  }
}
```

Employee service:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http'
import { Observable } from 'rxjs';
import { Employee } from './employee';

@Injectable({
  providedIn: 'root'
})
export class EmployeeService {

  private baseURL = "http://localhost:8080/api/v1/employees";

  constructor(private httpClient: HttpClient) { }

  getEmployeesList(): Observable<Employee[]>{
    return this.httpClient.get<Employee[]>(`${this.baseURL}`);
  }

  createEmployee(employee: Employee): Observable<Object>{
    return this.httpClient.post(`${this.baseURL}`, employee);
  }

  getEmployeeById(id: number): Observable<Employee>{
    return this.httpClient.get<Employee>(`${this.baseURL}/${id}`);
  }

  updateEmployee(id: number, employee: Employee): Observable<Object>{
    return this.httpClient.put(`${this.baseURL}/${id}`, employee);
  }

  deleteEmployee(id: number): Observable<Object>{
    return this.httpClient.delete(`${this.baseURL}/${id}`);
  }
}
```

Angular Spring Boot Application Login

step 1:Create authentication service:

=> Create a new authentication service where we check if the user name and password is correct then set it in session storage.

=> Using sessionStorage properties we can save key/value pairs in a web browser.
The sessionStorage object stores data for only one session .
So the data gets deleted if the browser is closed

authenticate() Authenticate the username and password
isUserLoggedIn() -checks the session storage if user name exists. If it does then return true
logout()- This method clears the session storage of user name

```
ng g s authentication
```

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class AuthenticationService {

  constructor() { }

  authenticate(username, password) {
    if (username === "raj" && password === "raj123") {
      sessionStorage.setItem('username', username)
      return true;
    } else {
      return false;
    }
  }

  isUserLoggedIn() {
    let user = sessionStorage.getItem('username')
    console.log(!(user === null))
    return !(user === null)
  }

  logOut() {
    sessionStorage.removeItem('username')
  }
}
```

step 2: create a login component:

```
------------------------------
ng g c login



import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { AuthenticationService } from '../authentication.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})

export class LoginComponent implements OnInit {

  username = 'raj'
  password = ''
  invalidLogin = false

  constructor(private router: Router,
    private loginservice: AuthenticationService) { }

  ngOnInit() {
  }

  checkLogin() {
    if (this.loginservice.authenticate(this.username, this.password)
    ) {
      this.router.navigate([''])
      this.invalidLogin = false
    } else
      this.invalidLogin = true
  }
}


from:
--------

<div class="container">
  <div>
    User Name : <input type="text" name="username" [(ngModel)]="username">
    Password : <input type="password" name="password" [(ngModel)]="password">
  </div>
  <button (click)=checkLogin() class="btn btn-success">
    Login
  </button>
</div>
```

Add the login path to the routing module.

```
import { LoginComponent } from './login/login.component';

const routes: Routes = [

  { path: 'login', component: LoginComponent },
];
```

Create a Logout Component

```
ng g c logout


import { Component, OnInit } from '@angular/core';
import { AuthenticationService } from '../authentication.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-logout',
  templateUrl: './logout.component.html',
  styleUrls: ['./logout.component.css']
})
export class LogoutComponent implements OnInit {

  constructor(
    private authentocationService: AuthenticationService,
    private router: Router) {

  }

  ngOnInit() {
    this.authentocationService.logOut();
    this.router.navigate(['login']);
  }

}
```

add logout path to rountes:

```
import { LogoutComponent } from './logout/logout.component';


const routes: Routes = [
  { path: 'logout', component: LogoutComponent },
];
```

Modify existing header component to add login , logout menu options
----

We need to inject AuthenticationService to

```
import { Component } from '@angular/core';
import { AuthenticationService } from './authentication.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit{
  constructor(private loginService:AuthenticationService) { }

  title = 'empclient';

 ngOnInit() {

  }
}
```

```
 <nav class="navbar navbar-expand-sm bg-primary navbar-dark">
  <ul class = "navbar-nav">
      <li class = "nav-item">
          <a *ngIf="loginService.isUserLoggedIn()"  routerLink="employees"
routerLinkActive="active" class="nav-link" >Employee List</a>
      </li>
      <li class = "nav-item">
        <a *ngIf="loginService.isUserLoggedIn()" routerLink="create-employee"
routerLinkActive="active" class="nav-link" >Add Employee</a>
      </li>
      <li class = "nav-item">
        <a *ngIf="!loginService.isUserLoggedIn()"  routerLink="login" routerLinkActive="active"
class="nav-link" >login</a>
      </li>
      <li class = "nav-item">
```

```
        <a *ngIf="loginService.isUserLoggedIn()" routerLink="logout" routerLinkActive="active"
class="nav-link" >logout</a>
      </li>
  </ul>
</nav>
```

Need of auth service:Create the AuthGaurd Service

=> But what will happen if the user directly tries to access a page without login.
For example if a user directly navigates to localhost:4200 He will be able to view the page.

=> So we should first check if the user is logged in and only then allow the user to view the page. We achive this using the CanActivate interface.

```
ng generate service authGaurd
```

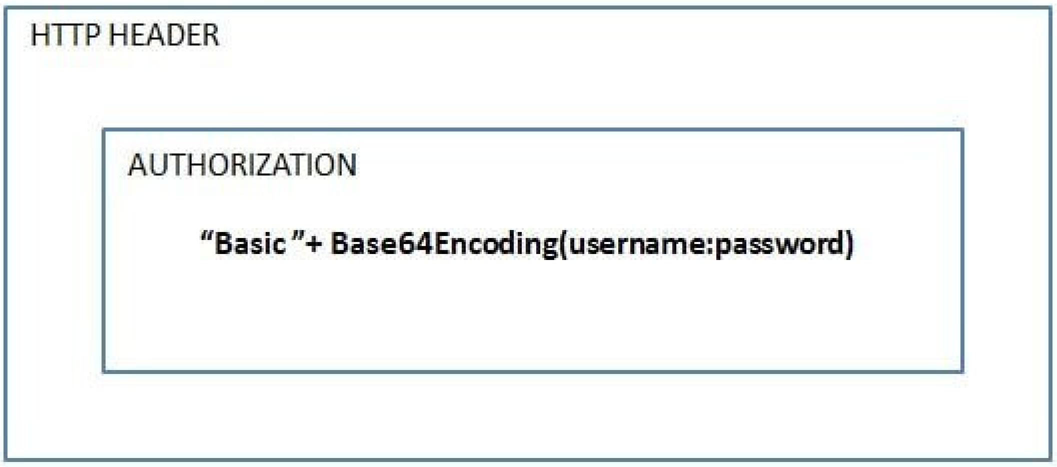
```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router } from
'@angular/router';
import { AuthenticationService } from './authentication.service';
@Injectable({
  providedIn: 'root'
})
export class AuthGaurdService {

  constructor(private router: Router,
    private authService: AuthenticationService) { }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    if (this.authService.isUserLoggedIn())
      return true;

    this.router.navigate(['login']);
    return false;

  }
}
```

Modify the app.routing.ts to activate route only if the user is logged in using the above AuthGaurdService.

```
import { AuthGaurdService } from './auth-gaurd.service';




const routes: Routes = [
  {path:'employees', component: EmployeeComponent, canActivate:[AuthGaurdService]},
  {path:'create-employee', component: CreateEmployeeComponent, canActivate:
[AuthGaurdService]},
  {path:'update-employee/:id', component: UpdateEmployeeComponent, canActivate:
[AuthGaurdService]},
  {path:'employee-details/:id', component: EmployeeDetailsComponent,canActivate:
[AuthGaurdService] },
  { path: 'login', component: LoginComponent },
  { path: 'logout', component: LogoutComponent ,canActivate:[AuthGaurdService]},
  {path:'', redirectTo:'employees', pathMatch:'full'}
];
```

<u>Angular + Spring Boot Basic Authentication</u>

Authorization header which has the word Basic followed by a space and base 64 encoded string username:password

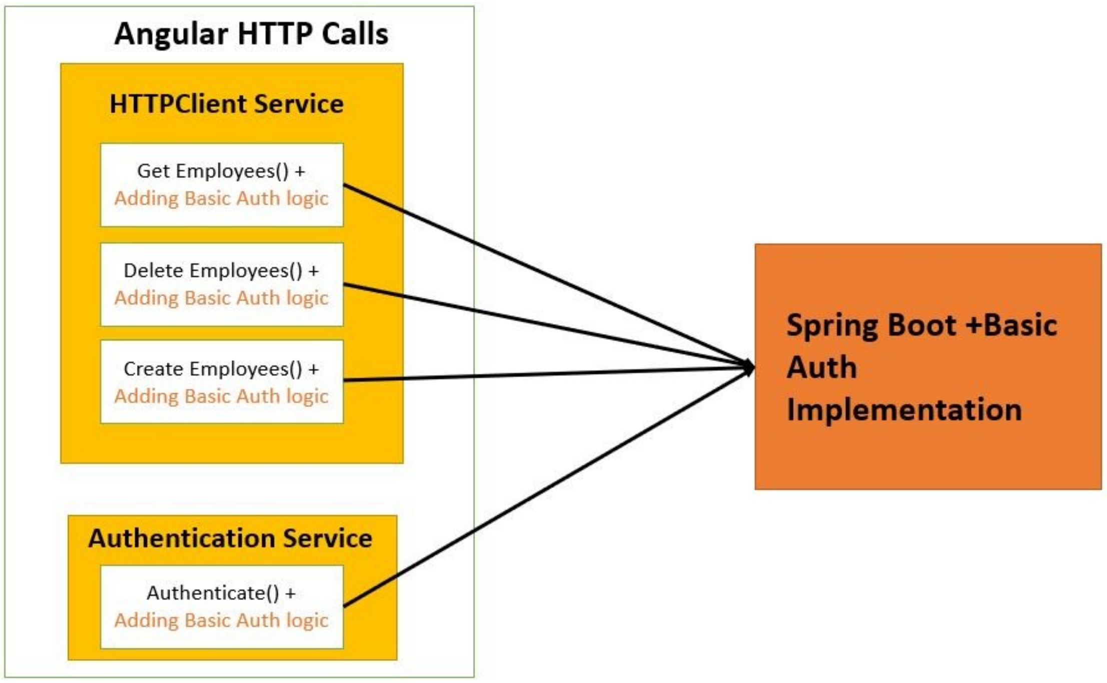


```
@CrossOrigin(origins = "http://localhost:4200")
```

spring boot update:
----
security dependency

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
	@Override
	protected void configure(HttpSecurity http) throws Exception {
		http.csrf().disable().

				authorizeRequests().antMatchers(HttpMethod.OPTIONS,
"/**").permitAll().anyRequest().authenticated()
				.and().httpBasic();
	}

	@Autowired
	public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

	auth.inMemoryAuthentication().withUser("raj").password("{noop}raj123").roles("USER");
	}
}
```

```
public class AuthResponse {
	private String status;

	public AuthResponse(String status) {
		this.status = status;
	}
	public String getStatus() {
		return status;
	}

	public void setStatus(String status) {
		this.status = status;
	}

}


@GetMapping(produces = "application/json")
@RequestMapping({ "/validateLogin" })
public AuthResponse validateLogin() {
		return new AuthResponse("User successfully authenticated");
}
```


Angular HTTP Calls
HTTPClient Service
Get Employees() + Adding Basic Auth logic
Delete Employees() + Adding Basic Auth logic
Create Employees() + Adding Basic Auth logic
Authentication Service
Authenticate() + Adding Basic Auth logic
Spring Boot +Basic Auth Implementation

Implement changes for Basic Authentication on the Angular side
----
In the EmployeeService we will be adding the basic auth header before making the REST calls.

```
import { Injectable } from '@angular/core';

import { from, Observable } from 'rxjs';
import {HttpClient, HttpHeaders} from '@angular/common/http'
import { Employee } from './employee';

@Injectable({
  providedIn: 'root'
})
export class EmployeeService {

  private baseURL="http://localhost:8080/empapp/employee";
  constructor(private httpClient: HttpClient) { }

  getEmployeesList(): Observable<Employee[]>{
    let username='raj'
    let password='raj123'
    const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' + password) });
    return this.httpClient.get<Employee[]>(`${this.baseURL}`,{headers});
  }

  createEmployee(employee: Employee): Observable<Object>{
    let username='raj'
    let password='raj123'
    const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' + password) });
    return this.httpClient.post(`${this.baseURL}`, employee,{headers});
  }
  updateEmployee(id: number, employee: Employee): Observable<Object>{
    let username='raj'
    let password='raj123'
    const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' + password) });
    return this.httpClient.put(`${this.baseURL}/${id}`, employee,{headers});
  }

  getEmployeeById(id: number): Observable<Employee>{
    let username='raj'
    let password='raj123'
    const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' + password) });
    return this.httpClient.get<Employee>(`${this.baseURL}/${id}`,{headers});
  }

  deleteEmployee(id: number): Observable<Object>{
    let username='raj'
    let password='raj123'
    const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' + password) });
    return this.httpClient.delete(`${this.baseURL}/${id}`,{headers});
  }
}
```

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';


@Injectable({
  providedIn: 'root'
})
export class HttpClientService {
  constructor(
    private httpClient:HttpClient
  ) {
     }

     getEmployees()
  {
    let username='raj'
    let password='raj123'

    const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' + password) });
    return this.httpClient.get<Employee[]>('http://localhost:8080/employees',{headers});
  }


  public deleteEmployee(employee) {
    let username='raj'
    let password='raj123'

    const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' + password) });
    return this.httpClient.delete<Employee>("http://localhost:8080/employees" + "/"+
employee.empId,{headers});
  }

  public createEmployee(employee) {
    let username='raj'
    let password='raj123'

    const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' + password) });
    return this.httpClient.post<Employee>("http://localhost:8080/employees", employee,{headers});
  }

}
```

Previously the authentication.service.ts used to check the credentials against hardcoded values. Now we will make a REST call using Basic Auth header. Only if the User object is returned will be login be successful.

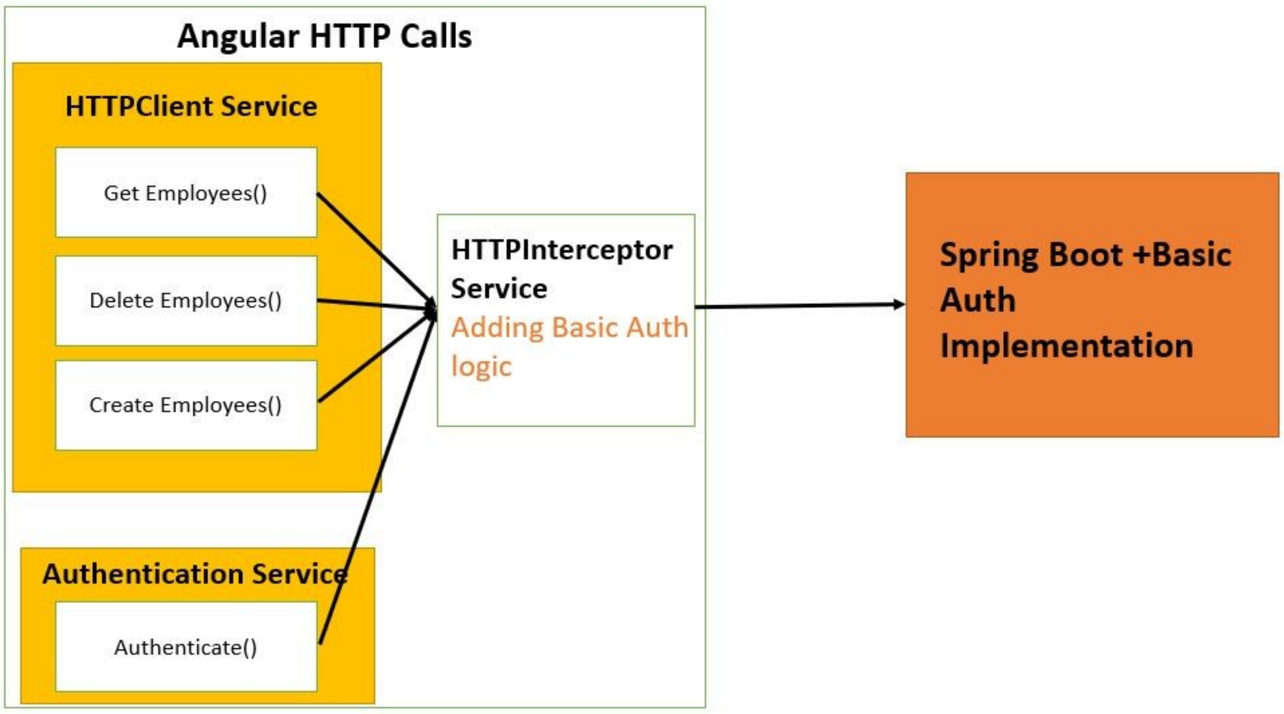
```
import { Injectable } from '@angular/core';
import { HttpClientService } from './http-client.service';
import { HttpClient,HttpHeaders } from '@angular/common/http';
import { map } from 'rxjs/operators';

export class AuthResponse{
  constructor(public status:string) {}
}

@Injectable({
  providedIn: 'root'
})
export class AuthenticationService {
  constructor(private httpClient:HttpClient) {  }

  authenticate(username, password) {
    const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' + password) });
    return this.httpClient.get<AuthResponse>('http://localhost:8080/empapp/validateLogin',
{headers}).pipe(
     map(
       userData => {
        sessionStorage.setItem('username',username);
        return userData;
       }
     )

    );
  }

  isUserLoggedIn() {
    let user = sessionStorage.getItem('username')
    console.log(!(user === null))
    return !(user === null)
  }

  logOut() {
    sessionStorage.removeItem('username')
  }
}
```

In the login.service.ts we check if the valid user is returned by the authentication service. If yes then login is successful and the user is forwarded to the employee page.

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { AuthenticationService } from '../service/authentication.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

  username = ''
  password = ''
  invalidLogin = false

  constructor(private router: Router,
    private loginservice: AuthenticationService) { }

  ngOnInit() {
  }

  checkLogin() {
    (this.loginservice.authenticate(this.username, this.password).subscribe(
      data => {
        this.router.navigate([''])
        this.invalidLogin = false
      },
      error => {
        this.invalidLogin = true

      }
    )
    );

  }

}
```

Angular Spring Boot Basic Auth Using HTTPInterceptor

We had seen we had to duplicate the code for adding Basic Auth Headers to the HTTPRequest before making HTTP calls.
In the authentication.service.ts if the authentication for the user entered username and password is successful, we will be saving the basicAuth string which we are adding the Authorization Header for basic Authenication in the session.



```
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { map } from 'rxjs/operators';
export class AuthResponse{
  constructor(public status:string) {}
}
@Injectable({
  providedIn: 'root'
})
export class AuthenticationService {
  constructor(private httpClient:HttpClient) {  }

  authenticate(username, password) {
    const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' + password) });
    return this.httpClient.get<AuthResponse>('http://localhost:8080/empapp/validateLogin',
{headers}).pipe(
      map(
        userData => {
          sessionStorage.setItem('username', username);
          let authString = 'Basic ' + btoa(username + ':' + password);
          sessionStorage.setItem('basicauth', authString);
          return userData;
        }
      )

    );
  }
  isUserLoggedIn() {
    let user = sessionStorage.getItem('username')
    console.log(!(user === null))
    return !(user === null)
  }

  logOut() {
    sessionStorage.removeItem('username')
  }
}
```

HttpInterceptor service:

Next we will be creating a new HttpInterceptor service called BasicAuthInterceptor Service. This service will check if the session has valid username and basicAuth String, then it will update the headers of all outgoing HTTP requests. We implement the interceptor by extending the HttpInterceptor.

```
ng g service BasicAuthHtppInterceptor
```

```
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler } from '@angular/common/http';
import { AuthenticationService } from './authentication.service';

@Injectable({
  providedIn: 'root'
})
export class BasicAuthHtppInterceptorService implements HttpInterceptor {

  constructor() { }

  intercept(req: HttpRequest<any>, next: HttpHandler) {

    if (sessionStorage.getItem('username') && sessionStorage.getItem('basicauth')) {
      req = req.clone({
        setHeaders: {
          Authorization: sessionStorage.getItem('basicauth')
        }
      })
    }

    return next.handle(req);

  }
}
```

Now we will register the created HTTPInterceptor using the app.module.ts by updating it in the provider section.

```
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { BasicAuthHtppInterceptorService } from './basic-auth-htpp-interceptor.service';



providers: [
    {
      provide:HTTP_INTERCEPTORS, useClass:BasicAuthHtppInterceptorService, multi:true
    }
    ]
```

Finally we will remove the hardcoded basic auth logic from the Http client service.

```
import { Injectable } from '@angular/core';

import { from, Observable } from 'rxjs';
import {HttpClient, HttpHeaders} from '@angular/common/http'
import { Employee } from './employee';

@Injectable({
  providedIn: 'root'
})
export class EmployeeService {

  private baseURL="http://localhost:8080/empapp/employee";
  constructor(private httpClient: HttpClient) { }

  getEmployeesList(): Observable<Employee[]>{
    // let username='raj'
    // let password='raj123'
    // const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' +
password) });
    // return this.httpClient.get<Employee[]>(`${this.baseURL}`,{headers});
    return this.httpClient.get<Employee[]>(`${this.baseURL}`);
  }

  createEmployee(employee: Employee): Observable<Object>{
    // let username='raj'
    // let password='raj123'
    // const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' +
password) });
```

```
   // return this.httpClient.post(`${this.baseURL}`, employee,{headers});
   return this.httpClient.post(`${this.baseURL}`, employee);
  }
  updateEmployee(id: number, employee: Employee): Observable<Object>{
    // let username='raj'
    // let password='raj123'
    // const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' +
password) });
    // return this.httpClient.put(`${this.baseURL}/${id}`, employee,{headers});

    return this.httpClient.put(`${this.baseURL}/${id}`, employee);
  }

  getEmployeeById(id: number): Observable<Employee>{
    // let username='raj'
    // let password='raj123'
    // const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' +
password) });
    //return this.httpClient.get<Employee>(`${this.baseURL}/${id}`,{headers});
    return this.httpClient.get<Employee>(`${this.baseURL}/${id}`);
  }

  deleteEmployee(id: number): Observable<Object>{
    // let username='raj'
    // let password='raj123'
    // const headers = new HttpHeaders({ Authorization: 'Basic ' + btoa(username + ':' +
password) });
    // return this.httpClient.delete(`${this.baseURL}/${id}`,{headers});
    return this.httpClient.delete(`${this.baseURL}/${id}`);
  }
}
```