Spring Security

1- Add dependency into pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

2- Restart server

3 - Generate user service into frontend: ng g s user
4 - Add login and signup functions
5 - logout function into service:

```
logout() {
    sessionStorage.removeItem('authUser');
  }
```

6 - header

```
  isUserLoggedIn:boolean = false;
```

```
    this.isUserLoggedIn = this.userService.isUserLoggedIn();
```

```
<li><a *ngIf='!isUserLoggedIn' routerLink="login"
class="nav-link">Login</a></li>
            <li><a *ngIf='isUserLoggedIn' routerLink="Logout"
class="nav-link">Logout</a></li>
```

7 - Auth guard
ng g s route-guard

```
constructor(private userService:UserService) { }
```

```
  canActivate ( route: ActivatedRouteSnapshot,  state:
RouterStateSnapshot){

    if (this.userService.isUserLoggedIn()){
      return true;
    }
    return false;
  }
```

```
  {path:'admin', component:AdminComponent,
canActivate:[RouteGuardService]},
```

```
constructor(
```

```
    private userService:UserService,
    private router:Router) { }
  canActivate ( route: ActivatedRouteSnapshot,  state:
RouterStateSnapshot){

    if (this.userService.isUserLoggedIn()){
      return true;
    }
    this.router.navigate(['login']);

    return false;
  }
```

8 - configure application.propoerties

spring.security.user.name= crococoder
spring.security.user.password= crococoder

Visit the page : http://localhost:8080/login

9 - adding headers to ignore this error

No 'Access-Control-Allow-Origin' header is present on the requested resource.

10-

Response to preflight request doesn't pass access control check: It does not have HTTP ok status.

Request Method: OPTIONS

Status Code: 401

CSRF ?

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute **unwanted actions** on a web application in which they're currently authenticated. With a little help of social engineering(such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack canforce the user to perform state changing requests like transferring funds, changing their email address ...

est une vulnérabilité de sécurité Web qui permet à un attaquant d'inciter les utilisateurs à effectuer des actions qu'ils n'ont pas l'intention d'effectuer. Il permet à un attaquant de contourner en partie la même stratégie d'origine, qui est conçue pour empêcher différents sites Web de s'interférer les uns avec les autres.

```
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityC
onfigurerAdapter;

@Configuration
@EnableWebSecurity
public class SpringSecurityConfigurationBasicAuth extends
WebSecurityConfigurerAdapter{

	@Override
	protected void configure(HttpSecurity http) throws Exception {
		http
		.csrf().disable()
		.authorizeRequests()
		.antMatchers(HttpMethod.OPTIONS,"/**").permitAll()
				.anyRequest().authenticated()
				.and()
			//.formLogin().and()
			.httpBasic();
	}
}
```

11 - To evite writing code into all web services:

ng g s http-interceptor

```
export class HttpInterceptorBasicAuthService implements HttpInterceptor
{



  constructor() { }

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

    let username = 'crococoder';

    let password = 'a';

    // btoa to convert string to base 64
```

```
    let basicAuthHeader = 'Basic ' +
window.btoa(username+':'+password);

    let request = req.clone({

      setHeaders: {

        Authorization: basicAuthHeader

      }

    });

    return next.handle(request);

  }

}
```

12 - Configure http-interceptor in app.module

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

13 - Create AuthBean

```
@CrossOrigin(origins = "http://localhost:4200")
@RestController
@RequestMapping("api/users")
public class BasicAuthControlelr {

        @GetMapping("/basic-auth")
        public AuthBean hello() {
                return new AuthBean("You are autheticated");
        }
}
```

14 - Create AuthBean

```
public class AuthBean {

        private String message;

        public AuthBean() {
```

```
        super();
    }


    public AuthBean(String message) {
        super();
        this.message = message;
    }


    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }


}
```

15 - ng g s basic-auth

```
login(email, pwd) {
    let basicAuthHeader = 'Basic ' + window.btoa(email+':'+pwd);
    let header = new HttpHeaders({
      Authorization: basicAuthHeader
    });

    return this.httpClient.get(this.userUrl+'/basic-auth',
{headers:header})

  }
```

16 - edit login function call into login component

```
login() {
    this.basicAuth.login(this.user.email, this.user.pwd).subscribe(
      (data)=> {
        console.log('here data',data);
        this.router.navigate(['admin']);
        this.invalidLogin = false;
      } ,
      error => {
```

```
        console.log('error', error);
        this.invalidLogin = true;

      }
    )
  }
```

That works into all cases, comment provider into app.module.ts

16- Edit basic-auth service

```
 login(email, pwd) {
    let basicAuthHeader = 'Basic ' + window.btoa(email + ':' + pwd);
    let header = new HttpHeaders({
      Authorization: basicAuthHeader
    });

    return this.httpClient.get(this.userUrl + '/basic-auth', { headers:
header }).pipe(
      map(
        data => {
          sessionStorage.setItem('authUser', email);
          sessionStorage.setItem('token', basicAuthHeader);
          return data;
        }
      )
    )

  }

  getAuthUser(){
    return sessionStorage.getItem('authUser');
  }

  getToken(){
    if (this.getAuthUser()) {
      return sessionStorage.getItem('token');
    }
  }

  isUserLoggedIn() {
    let user = sessionStorage.getItem('authUser');
    return !(user === null);
  }
```

```
  logout() {
    sessionStorage.removeItem('authUser');
    sessionStorage.removeItem('token');
  }
```

17- edit interceptor service

```
constructor(private basicAuth:BasciAuthService) { }
  intercept(request: HttpRequest<any>, next: HttpHandler){

    let basicAuthHeader = this.basicAuth.getToken();
    if (basicAuthHeader) {
       request = request.clone({
        setHeaders: {
          Authorization: basicAuthHeader
        }
      });
    }
    return next.handle(request);
  }
```

18 - http://jwt.io/

JWT :

What happened was we wanted a common token standard, which is called Jott. Json Web token. The awesome thing about Jott is that it can contain user authorisations and a number
of other details.

There is a standard by which it is defined, and there is a part where you can extend and add custom information

An open industry standard method for representing claims between two parties talking to each
other. And you don't want to know what are the claims what are the authorizations that a specific user has.

Now you might be wondering what is present in the JWT?what is present in the jar? And let's look at it
right now.

So what you are seeing on the left hand side, is the encoded final token, and on the right hand side
what is present is the information which is present in that specific token, and it contains a header
a payload and a signature part. The things which are present in the header are typically the

algorithm.
What are you using for the hashing.
So if you look at here, there are a number of them to represent. In the example that we'll build we'll
be using HS512.
And you would see that this header contains the algorithm HS512, and it contains a type, it's a standard
jott,
So it's JWT.
The payload contains a few predefined elements, that is most of these are not really mandatory.
The subject is Who are we talking about.
The name is the name of the person.
You can have additional things.
This admin to indicate that this guy is an admin.
The 80 here indicates the creation time of the token. In addition to this, you can add custom defined
stuff as well.
You might add what are the authorizations that a specific user has.
The last part of the JWT is the verify signature. It contains the base 64 encoded payload. The
last.
And also one of the most important part of it is your 512 bit secret.
This is kind of what secret string is provided in the JWT.
They can have a secret string, which should not be revealed to anyone and that makes sure that when you
decoded it, those guys who know the secret would be able to decode it properly. And you can also base
64 encoding on the secret key as well.
So what you're seeing in here are the most important part of the job.
Typically what we would do in web applications, is for example application, management application. What
we would do is when user logs in with a user ID password, we would send a request to the server and we
would get JWT token back, and for all the subsequent requests we would using Jott, or the JWT token
and we will be positioning it as part of the header. There would be an expiration time also on the jar
tokens and when the expiration date is coming near, you have to send the request to something called
refresh token request.

Adding User and Role Models
Adding Repositories
Adding services and implementation
Adding UserController and Login Controller

Adding Login and signup into user-jwt-service

Adding AuthenticationResponse Model
Adding Filters package
Adding Security
Adding Util

Adding jwtt depedance into pom.xml

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

Adding permitted-url into application.properties