```
================
Spring Security
=================
```

1) Authentication (verifying credentials)

2) Authorization (can this user access specific functionality)

-> Security is very important for every web application

-> To protect our application & application data we need to implement security logic

-> Spring Security concept we can use to secure our web applications / REST APIs

-> To secure our spring boot application we need to add below starter in pom.xml file

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

Note: When we add this dependency in pom.xml file then by default our application will be secured with basic authentication. It will generate random password to access our application.

Note: Generated Random Password will be printed on console.

-> We need to use below credentials to access our application

```
Username : user

Password : <copy the pwd from console>
```

-> When we access our application url in browser then it will display "Login Form" to authenticate our request.

-> To access secured REST API from postman, we need to set Auth values in POSTMAN to send the request

```
Auth : Basic Auth
Username : user
Password : <copy-from-console>
```

```
=====================================================
How to override Spring Security Default Credentials
=====================================================
```

-> To override Default credentials we can configre security credentials in application.properties file or application.yml file like below

```
spring.security.user.name=ashokit
spring.security.user.password=ashokit@123
```

-> After configuring credentials like above, we need to give above credentials to access our application / api.

```
====================================
How to secure specific URL Patterns
====================================
```

-> When we add 'security-starter' in pom.xml then it will apply security filter for all the HTTP methods of our application.

-> But in reality we need to secure only few methods not all methods in our application.

```
                ##For Example##

                                / login-page --> security not required

                                / transfer ---> security required

                                / balance ---> security required

                                / about-us ---> security not required

                                / contact-us ---> security not required


-> In order to achieve above requirement we need to Customize Security Configuration in our
project like below


@Configuration
@EnableWebSecurity
public class SecurityConfigurer {

        @Bean
        public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
                http.authorizeHttpRequests((authorize) -> authorize
                                                .requestMatchers("/contact", "/swagger-
ui.html").permitAll()
                                                .anyRequest().authenticated()
                        )
                                .httpBasic(withDefaults())
                                .formLogin(withDefaults());
                return http.build();
        }
}


========================================
Spring Security In-Memory Authentication
========================================

-> In Memory Authentication means storing user credentials in the program for Authentication
Purpose.

-> This is not recommended for production.

        @Bean
        public InMemoryUserDetailsManager inMemoryUsers() {

                UserDetails ashokUser = User.withDefaultPasswordEncoder()
                                                                        .username("ashok")
                                                                        .password("ashok")
                                                                        .authorities("ADMIN")
                                                                        .build();


                UserDetails johnUser = User.withDefaultPasswordEncoder()
                                                                        .username("john")
                                                                        .password("john")
                                                                        .authorities("USER")
                                                                        .build();

                return new InMemoryUserDetailsManager(ashokUser, johnUser);

        }


        @Bean
        public SecurityFilterChain securityConfig(HttpSecurity http) throws Exception {
```

```
                http.authorizeHttpRequests( (req) -> req
                                .antMatchers("/admin").hasRole(ADMIN)
                                .antMatchers("/user").hasAnyRole("ADMIN","USER")
                                .antMatchers("/").permitAll()
                                .anyRequest().authenticated()
                ).formLogin();

                return http.build();
        }

}


=======================================================
How to work with UserDetailsService in Spring Security
=======================================================

=> UserDetailsService is a predefined interface which contains loadUserByUsername(String name)
method.

=> This is used to load User record for Authentication purpose in Spring Security.

=> We can implement UserDetailsService interface and we can write the logic to retrieve User
record based on given username for Authentication purpose.

=> If we give UserDetailsService object to AuthenticationProvider then AuthManager will call this
method for every login request.


=============================================
Login and Registration using Spring Security
=============================================

## 1) Create Boot app with required dependencies ##

                        a) web-starter
                        b) data-jpa-starter
                        c) mysql
                        d) security-starter
                        e) devtools

## 2) Configure Data Source properties in yml file ##

## 2) Create Entity class & Repository interface ##

@Repository
public interface CustomerRepo extends CrudRepository<Customer, Integer> {

        public Customer findByUname(String cuname);

}

## 3) Create UserDetailsService class ##

@Service
public class MyUserDetailsService implements UserDetailsService {

        @Autowired
        private CustomerRepo crepo;

        @Override
        public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
                Customer c = crepo.findByUname(username);
                return new User(c.getUname(), c.getPwd(), Collections.emptyList());
        }
}


## 4) Create Security Config Class ##
```

```
@Configuration
@EnableWebSecurity
public class AppSecurityConfig {

	@Autowired
	private MyUserDetailsService userDtlsSvc;

	@Bean
	public PasswordEncoder pwdEncoder() {
		return new BCryptPasswordEncoder();
	}

	@Bean
    public AuthenticationProvider authenticationProvider(){
        DaoAuthenticationProvider authenticationProvider=
                        new DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(userDtlsSvc);
        authenticationProvider.setPasswordEncoder(pwdEncoder());
        return authenticationProvider;
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws
Exception {
        return config.getAuthenticationManager();
    }

    @Bean
	public SecurityFilterChain securityConfig(HttpSecurity http) throws Exception {
		return http.csrf().disable()
		.authorizeHttpRequests()
		.requestMatchers("/register", "/login").permitAll()
		.and()
		.build();
	}
}
```

5) Create RestController with required methods

```
@RestController
public class CustomerRestController {

	@Autowired
	private CustomerRepo crepo;

	@Autowired
	private PasswordEncoder pwdEncoder;

	@Autowired
	private AuthenticationManager authManager;

	@PostMapping("/login")
	public ResponseEntity<String> loginCheck(@RequestBody Customer c) {

		UsernamePasswordAuthenticationToken token =
                        new UsernamePasswordAuthenticationToken(c.getUname(), c.getPwd());

		try {
			Authentication authenticate = authManager.authenticate(token);

			if (authenticate.isAuthenticated()) {
				return new ResponseEntity<>("Welcome To Ashok IT", HttpStatus.OK);
			}

		} catch (Exception e) {
			//logger
		}
```

```
                return new ResponseEntity<String>("Invalid Credentials", HttpStatus.BAD_REQUEST);
        }

        @PostMapping("/register")
        public String registerCustomer(@RequestBody Customer customer) {

                // duplicate check

                String encodedPwd = pwdEncoder.encode(customer.getPwd());
                customer.setPwd(encodedPwd);

                crepo.save(customer);

                return "User registered";
        }

}


## 6) Run the application and test it



##############
OAuth 2.0
##############

###  1) Create Spring Boot application with below dependencies

                a) web-starter
                b) security-starter
                c) oauth-client


### 2) Create OAuth app in Github.com

        (Login --> Profile -> Settings --> Developer Settings --> OAuth Apps --> Create App -->
Copy Client ID & Client Secret)


### 3) Configure GitHub OAuth App client id & client secret in application.yml file like below


spring:
  security:
    oauth2:
      client:
        registration:
          github:
            clientId:
            clientSecret:


### 4) Create Rest Controller with method

@RestController
public class WelcomeRestController {

        @GetMapping("/")
        public String welcome() {
                return "Welcome to Ashok IT";
        }
}

#### 5) Run the application and test it.


Assignment : Spring Boot with oAuth using google account. Get username also from google and
display that in response.
```

```
########################
Spring Boot with JWT
########################

-> JWT stands for JSON Web Tokens

-> JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely
between two parties.

-> JWT official Website : https://jwt.io/

-> Below is the sample JWT Token

token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF
0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

-> JWT contains below 3 parts

                1) Header
                2) Payload
                3) Signature

Note: JWT 3 parts will be seperated by using dot(.)


## Git Hub Repo : https://github.com/ashokitschool/SpringBoot_JWT_App.git



1) JWT Token generation (JwtService.java)

                - generateToken(String uname)
                - validateToken(String uname)

2) JWT Token validation Filter (AppFilter.java) - OncePerRequest

                - check Authorization header presence
                - retrieve bearer token from header
                - validate token
                - if token is valid, update security context to process req

3) Customize SecurityFilterChain

                - permit /api/register & /api/login urls
                - authenticate any other request
                - set security context as stateless

============================
Authorization Token Format
============================
Key = Authorization
Value = Bearer <token>


================================
Microservices with JWT Security
================================

=> Auth-Service contains functionality for user registration and user login with MySQL DB.

=> If user login successfully then auth-service will generate JWT token and will send it as
response to user.

=> API-Gateway contains logic to validate the token using Filter.

Note: In API-Gateway we have added routings for our microservices along with Filter.
```

=> When we access any microservice url through api-gateway then api-gateway will execute filter to validate the token. If token is valid then only api-gateway will route the request to particular microservice. If token is invalid then api-gateway will throw Exception.

Git Hub Repo : https://github.com/ashokitschool/Microservices_Security.git