Sequence

1. pom.xml
2. application.properties
3. com.p1.model
4. services
5. WebSecurityConfig
6. Request
7. Response

| **Methods** | **Urls** | **Actions** |
| --- | --- | --- |
| POST | /api/auth/signup | signup new account |
| POST | /api/auth/signin | login an account |
| GET | /api/test/all | retrieve public content |
| GET | /api/test/user | access User’s content |
| GET | /api/test/mod | access Moderator’s content |
| GET | /api/test/admin | access Admin’s content |

Spring Security will manage cors, csrf, session, rules for protected resources, authentication & authorization along with exception handler.

Diagram explanation

* **Spring Security**
* – WebSecurityConfig is the crux of our security implementation. It configures cors, csrf, session management, rules for protected resources. We can also extend and customize the default configuration that contains the elements below.  
  (WebSecurityConfigurerAdapter is deprecated from Spring 2.7.0, you can check the source code for update. More details at:  
  [WebSecurityConfigurerAdapter Deprecated in Spring Boot](https://www.bezkoder.com/websecurityconfigureradapter-deprecated-spring-boot/))
* – [UserDetailsService](https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/" \l "tech-userdetailsservice) interface has a method to load User by username and returns a UserDetails object that Spring Security can use for authentication and validation.
* – UserDetails contains necessary information (such as: username, password, authorities) to build an Authentication object.
* – [UsernamePasswordAuthenticationToken](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/authentication/UsernamePasswordAuthenticationToken.html) gets {username, password} from login Request, AuthenticationManager will use it to authenticate a login account.
* – [AuthenticationManager](https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/" \l "core-services-authentication-manager) has a DaoAuthenticationProvider (with help of UserDetailsService & PasswordEncoder) to validate UsernamePasswordAuthenticationToken object. If successful, AuthenticationManager returns a fully populated Authentication object (including granted authorities).
* – [OncePerRequestFilter](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/filter/OncePerRequestFilter.html) makes a single execution for each request to our API. It provides a doFilterInternal() method that we will implement parsing & validating JWT, loading User details (using UserDetailsService), checking Authorizaion (using UsernamePasswordAuthenticationToken).
* – [AuthenticationEntryPoint](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/AuthenticationEntryPoint.html) will catch authentication error.
* **Repository** contains UserRepository & RoleRepository to work with Database, will be imported into **Controller**.
* **Controller** receives and handles request after it was filtered by OncePerRequestFilter.
* – AuthController handles signup/login requests
* – TestController has accessing protected resource methods with role based validations.

configure Spring Security & implement Security Objects here.

* WebSecurityConfig

(WebSecurityConfigurerAdapter is deprecated from Spring 2.7.0, you can check the source code for update. More details at:  
[WebSecurityConfigurerAdapter Deprecated in Spring Boot](https://www.bezkoder.com/websecurityconfigureradapter-deprecated-spring-boot/))

* UserDetailsServiceImpl implements UserDetailsService
* UserDetailsImpl implements UserDetails
* AuthEntryPointJwt implements AuthenticationEntryPoint
* AuthTokenFilter extends OncePerRequestFilter
* JwtUtils provides methods for generating, parsing, validating JWT

**controllers** handle signup/login requests & authorized requests.

* AuthController: @PostMapping(‘/signin’), @PostMapping(‘/signup’)
* TestController: @GetMapping(‘/api/test/all’), @GetMapping(‘/api/test/[role]’)

**repository** has intefaces that extend Spring Data MongoDB MongoRepository to interact with Database.

* UserRepository extends MongoRepository<User, String>
* RoleRepository extends MongoRepository<Role, String>

**models** defines two main models for Authentication (User) & Authorization (Role). They have many-to-many relationship.

* User: id, username, email, password, roles
* Role: id, name

**payload** defines classes for Request and Response objects

– @EnableWebSecurity allows Spring to find and automatically apply the class to the global Web Security.

– @EnableGlobalMethodSecurity provides AOP security on methods. It enables @PreAuthorize, @PostAuthorize, it also supports [JSR-250](https://en.wikipedia.org/wiki/JSR_250).

– Override the configure(HttpSecurity http) method from WebSecurityConfigurerAdapter interface. It tells Spring Security how we configure CORS and CSRF, when we want to require all users to be authenticated or not, which filter (AuthTokenFilter) and when we want it to work (filter before UsernamePasswordAuthenticationFilter), which Exception Handler is chosen (AuthEntryPointJwt).

– Spring Security will load User details to perform authentication & authorization. So it has UserDetailsService interface that we need to implement.

– The implementation of UserDetailsService will be used for configuring DaoAuthenticationProvider by AuthenticationManagerBuilder.userDetailsService() method.

–Need a PasswordEncoder for the DaoAuthenticationProvider. If not specified, then will use plain text.

In UserDetailsImpl ->   convert Set<Role> into List<GrantedAuthority>. It is important to work with Spring Security and Authentication object later.

Add some rows into **roles** collection before assigning any role to User.  
Run following MongoDB insert statements:

db.roles.insertMany([

{ name: "ROLE\_USER" },

{ name: "ROLE\_MODERATOR" },

{ name: "ROLE\_ADMIN" },

])

Register some users with /signup API:

* **admin** with ROLE\_ADMIN
* **moderator** with ROLE\_MODERATOR and ROLE\_USER
* **bezkoder** with ROLE\_USER

<http://localhost:7575/api/auth/signup>

{

“username”:“testuser”,

“email”:”test@gmail.com”,

“password”:”12345678”,

“roles”: [ “user”, “mod”]

}