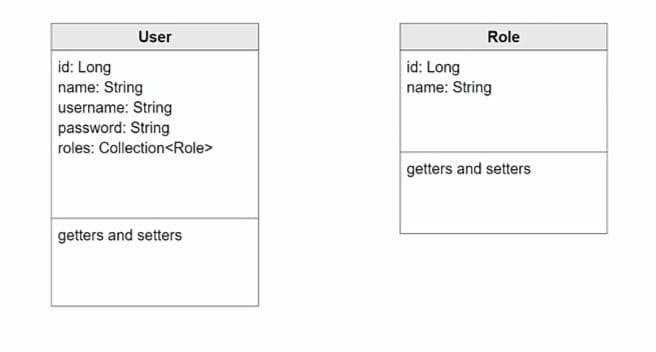
JSON Web Token

Application A talks with Application B when it wants to communicate. This communication hold process by tokens. It is very popular way of communication because it is pretty secure. It is also small and very easy to work with. It is mostly used for authoritazitaion. First, application wants to authotarize me. Once it authotritaze me, it gives me web Token. That Token will have my information, authotratization etc. So whenever the request comes in, the application will take it, look at the token, verify that the token has not been tempered with. So the token is good. And then I will check the permissions on the token to see if you have access to that spesific resource. And then if you do, it will give you back the resource. And Adjutant web token has 3 parts.

1. **Header**. In the header, you have 2 fields: algorithm and type. {“alg”: “HS256”, “typ”:”JWT”} Typ will always be “JWT” since we are working with “JWT”
2. **Payload**. It contains information about the user or the entity. {“sub”:”johnd24”, --username--“name”:”John Doe”, “iat”:1516239022, --authotirization number--“claims”:”create, edit”}
3. **Signature**. To create the signature, you have to take the encoded header. You take the encoded payload, and then you use a secret with the algorithm. You can take all that and then sign the token. 256-bit-secret

In a nutshell, JSON Web token is a way to applications to comminicate, and it mostly used with API ‘s for authorization.

New /Project/ Spring Initializr/ Name: userservice / Artifact: userservice/ Description: Security Service Application Demo/ Package name: groupismi.userservice Dependencies: Spring Data JPA, MySQL Driver(Oracle da olabilir bu), SpringWeb, Spring Security Spring Boot DevTools, Lombok.

Src / Main / Java ‘dan UserServiceApplication Class ı aç. Aşağıdakiler ders boyunca bize lazım olacak parametreler. Bunların hakkında biraz konuş.  
  
  
  
  
  
  
  
  
  
  
  
domain package ını oluştur. “User” ve “Role” u bunun içine tanımlayacağım. User & Role Classlarını oluştur. User Class ını açi Yukardaki table dan devam ederek user’ ın içindeki variableları oluştur.(

private Long id;

private String name;

private String username;--username in string içeren herhangi birşey olabilir—

private String password;

@ManyToMany(fetch = FetchType.EAGER –dedim çünkü I want to load all of the roles whenever I load the user.

private Collection<Role> roles = new ArrayList<>(); )

Şimdi de public class Role ‘un üzerine birkaç tane annotation ekleyelim. @Entity @Data @NoArgsConstructor @AllArgsConstructor public class User ın altına @Id @GeneratedValue(strategy = GenerationType.AUTO) larını da ekle. User ın üstündeki ve altındaki annotationsları, private Long id; ve private String name; I User dan kopyalayıp Role a yapıştır.

Repo isimli başka bir package oluştur. İçine de UserRepo Interface ini oluştur. (extends JpaRepository<User—userservice.domain’deki, Long> ile birlikte) I only need one method here, and it’s gonna return return.İçine User findByUsername(String username); methodunu oluştur. RoleRepo interface’ini de oluştur. (extends JpaRepository<Role—userservice.domain’deki, Long> ile birlikte ) .İçine Role findByname(String username); methodunu oluştur. So that is the role repository. It’s gonna return the role if we give it the name of the role.

Let’s create a service. First, I need to create the service package. Service package’ının içine UserService interface ini oluştur. In here, we’re gonna define all of the methods that we need on the application so that we can manage the users. Inside here, I need to create User saveUser(User user) method so that I can save the users. So whenever I called this method, I’m gonna save the user from the database. Next one that I need, is to be able to save a role, so that it can return role type to me. Create Role saveRole(Role role); And next, I need to add a role to spesific user, and this method is not gonna return anything. So I’m gonna create void addRoleToUser(String username, String roleName); And then, I need another method that’s gonna return user. Create User getUser(String username); And, I need one last method that returns a list of all users. Create List<User>getUsers(); There is one thing I also need to mention here. Whenever you have this kind of method, to get all users, to get all roles sth like that, in the real world applications you probably wouldn’t have that because this is gonna try to load everything that you have in your database. In the real world, you would usually return a page to a user. And the reason for this is because you don’t want the system to try to load all of the users that you have in a database because they’re not even going to be group C, let’s say 500000 users on the screen. It’s just very inefficient, and it’s gonna put a lot of strees on your backend, so you don’t want to do that. So usually when you have methods that are gonna return, like a list of sth, you want to return a page. In that case, you re not going to print 50000 users, you’ll just print first 10 veya 20 veya 50 veya 100 users, and that’s gonna be page one. If you want to see page two, maybe you can send me another request for page 2 so forth. The point is, you don’t want to load 500000 users with sth like get users, because that’s just going to get your application perform very poorly.

We created all the things we need, now it is time to continue with Implementations. Create UserServiceImpl Class(implements UserService ile birlikte). Hata verecek, karşına çıkan bütün methodları implement et. Class ın hemen altına private final UserRepo UserRepo; ve private final RoleRepo roleRepo; yu oluştur. So those are communicating with JPA directly, which in turn is doing a lot of things in the backend to create those queries for us and inquiry the database for us. So I’m gonna set the @Service—class ın üstüne— to understand this is the Service Class. Put also @RequiredArgsConstructor to put all of these inside the constructor and remove error. I need everything in this class to be transactional. Put @transactional. And then there is one last annotation that I need and that’s for logs. Use @Slf4j. Override methodlarının içini doldurmaya başlıyoruz. **LOGLAR OLMADAN**

SaveUser için: return userRepo.save(user);

SaveRole için: return roleRepo.save(role);

addRoleToUser için: User user = userRepo.findByUsername(username); ve Role role = roleRepo.findByName(roleName); ve user.getRoles().add(role);

getUser için: userRepo.findByUsername(username); So that’s gonna return the user. To get all the users, continue with

List<User> için: return userRepo.findAll(); So that’s gonna return all the users and interface.

Go ahead and put some logs to see whats going on in this method. BUTUN OVERRIDE METHODLARINA LOGLARI DA EKLE.

api Package ını oluştur. İçine de UserResource—controller yani-- class ını oluştur. In the UserResource, first thing I’m gonna do is I’m gonna inject the controller(private final UserService userService;). And then, I need to make this class controller, so I’m gonna add @RestController at the top of the class. Add @RequiredArgsConstructor so that inject this field into the constructor. Then I need @RequestMapping(“/api”) so that I can give the entire class a base or path. Lets create public ResponseEntity<List<User>>getUsers(){return ResponseEntity.ok().body(userService.getUsers());}method to return all the users. Add @GetMapping(“/users”) to determine whether this gonna be a pose or get request.

So now, only thing what we have to do is putting the configuration from the SQL. Go to the resources/application.properties. Yapışkan Notlardaki application properties’ I yapıştır. UserService’e gel ve kodu çalıştır. See, it created the security password, users\_roles table, users table and roles table successfully. There are couple of errors and warnings, but it doesn’t matter. We are going to fix them in the future. <http://localhost:8080/login> yi tarayıcından aç. Kullanıcı adını “user” şifreni consoldaki şifre olarak gir. Gördüğünüz gibi boş bir köşeli parantez var sadece, bu boş bir list anlamına geliyor. List imiz boş çünkü şu an için herhangi bir kullanıcı oluşturmadık arkadaşlar.

UserResorce a gel. @GetMapping(“/users”) ve altındaki ResponseEntity yi kopyala ve alt tarafına yapıştır. @GetMapping(“/users”)’I @PostMapping(“/user/save”) olarak değiştir. ResponseEntity yi de ResponseEntity<User>saveUser(@RequestBody User user) { URI uri = URI.create(ServletUriComponentBuilder.fromCurrentContextPath().path(“/api/user/save”).toUriString(); return ResponseEntity.created(uri).body(userService.saveUser(user)); olarak güncelle. Bunu yaptık çünkü DB e user kaydetmek istiyoruz. Şimdi de role kaydeden bir methoda ihtiyacımız var. saveUser I kopyala ve alta yapıştır. User la ilgili ne varsa role olarak değiştir. addRoleToUser methodunu oluştur. @Data yı kullan ve RoleToUserForm class ını oluştur.(class RoleToUserForm { private String username; private String roleName;}) There are two ways to test the application. We can either run the application or use Postman to test. UserServiceApplication’a git.

@Bean

CommandLineRunner run(UserService userService) {return args -> {Burada çalıştırdığım her şey uygulama tamamlandıktan sonra çalışacak

userService.saveRole(new Role(null, “ROLE\_USER”));

userService.saveRole(new Role(null, “ROLE\_MANAGER”));

userService.saveRole(new Role(null, “ROLE\_ADMIN”));

userService.saveRole(new Role(null, “ROLE\_SUPER\_ADMIN”));

userService.saveUser(new User(null, “Erol Buyukburc”, “Saksi”, “1234”, new ArrayList<>()));

userService.saveUser(new User(null, “Sezen Aksu”, “minikserce”, “1234”, new ArrayList) <>()));

userService.saveUser(new User(null, “Will Smith”, “will”, “1234”, new ArrayList<>())));

userService.saveUser(new User(null, “Jim Carrey”, “jim”, “1234”, new ArrayList<>())));

userService.addRoleToUser(“Saksi”, “ROLE\_USER”);

userService.addRoleToUser(“Saksi”, “ROLE\_MANAGER”);

userService.addRoleToUser(“minikserce”, “ROLE\_MANAGER”);

userService.addRoleToUser(“jim”, “ROLE\_ADMIN”);

};}

Run the code. Console daki şifreyi ve “user” kullanıcı adını localhost/api/users’a gir. See, everything is running as it expected.

Elimizde farklı roller var(Role Admin veya Role User gibi) ve bu rollerin DB deki yetkilelerinin sınırlandırılması gerekiyor. Bunun için de security package ını oluştur. İçine de SecurityConfig Class ını oluştur(extends WebSecurityConfigurerAdapter ile birlikte). @Configuration @EnableWebSecurity @RequiredArgsConstructor annotation larını class ın başına koy. Override configure(ikinci olan işte) and configure(http:HttpSecurity) Configure daki super methodunu sil ve şunu yaz: outh.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder); . Override methodlarının üzerine private final UserDetailsService userDetailsService; ve private final BcryptPasswordEncoder bCryptPasswordEncoder i oluştur. Application Class a git. **Main Method’unu altına** @Bean PasswordEncoder passwordEncoder(){return new BcyrptPasswordEncoder();} u yaz.Bunları yapıyoruz çünkü SecurityConfig i içindeki private variable larla birlikte main methoda bağlıyoruz. UserServiceImpl Class ına git. UserDetailsService I implement et. Hata verecek, override methodunu çağır. En alta yazacak, en üste taşı. İçindeki String s i String username olarak değiştir. Return ün üstüne User user = userRepo.findByUsername(username); i ve onun altına if(user == null) {

Log.error(“User not found in the database”); throw new UsernameNotFoundException(“User not found in the database”);} else { Log.info(“User found in the database: {}”, username);} I yaz. Return ün içine return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(), authorities); I ekle. Return’ün üzerine de user.getRoles().forEach(role -> {authorities.add(new SimpleGrantedAuthority(role.getName()));}); ü ekle.

Authentication ve Authorization Farkı

Authentication

Verifies you are who you say you are. Method: Login form, HTTP authentication, Custom auth, method

Binaya girebiliyor musun ona bakıyor

Authorization

Decides if you have permission to access a resource. Method: Access Control URL’s, Access Control List(ACLs)

Binaya girdin, içindeki herhangi bir odaya erişebiliyor musun ona bakıyor. SecurityConfig Class ına geri git. Override configure methodundaki super I sil ve şunu yaz: http.csrf().disable(); http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS); http.authorizeRequests().anyRequest().permitAll(); http.addFilter(null);

Create the filter package. İçine de CustomAuthenticationFilter Class ını oluştur(extends UsernamePasswordAuthenticationFilter ile birlikte) İçine de attemptAuthenitication Override method unu implement et. SuccessfulAuthentication methodunu da implement et. Log ları da import etmemiz lazım. @Slf4j Annotationını class ın üzerine çağır. Constructor da oluşturmamız lazım. Private final AuthenticationManager authenticationManager; public CustomAuthenticationFilter(AuthenticatioManager authenticationManager) {this.authenticationManager = authenticationmanager; } constructor unu da oluştur. So we have the constructor, we have Filter. We also have successfulAuthentication method. Whenever the authentication is successful, this method is going to be used. attemptAuthentication daki super methodunu sil ve şunu yaz: String username = request.getParameter(“username”); -- aynı yolla password için de bir tane oluştur-- Burada şunu yapıyoruz arkadaşlar hani karşımıza bi ekran çıktı ya localhostta username ve password girdiğimiz. O springboot la default gelmişti biz istediğimiz gibi düzenlemek için onu baştan yapıyoruz. Im gonna log this information so that we can see. Password un altına şunu yaz: log.info(“Username is: {} ”,username); log.info(“Password is: {}”, password); Then, I need to create an object for username and password.(UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(username, password); return authenticationManager.authenticate(authenticationToken); ) Burada şunu yaptık, username ve password u tanımladık ve authenticationManager I çağırarak girilen user I yetkilendirdik. SecurityConfig i aç ve http.addFilter()ın içini doldur: http.addFilter(new CustomAuthenticationFilter(authenticationManagerBean())); Aşağısına bir sınıf daha oluştur. @Bean @Override (public AuthenticationManager authenticationManagerBean() throws Exception{ return super.authenticationManagerBean(); }

**BİRİNCİ DERS BURADA SONLANDI**