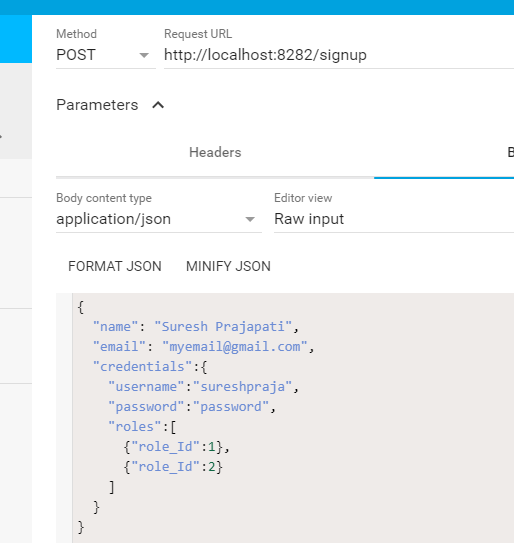
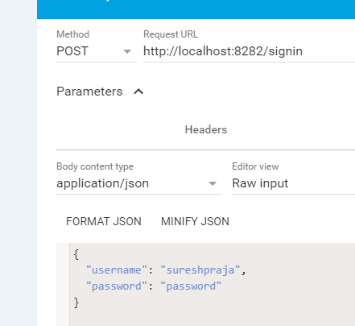
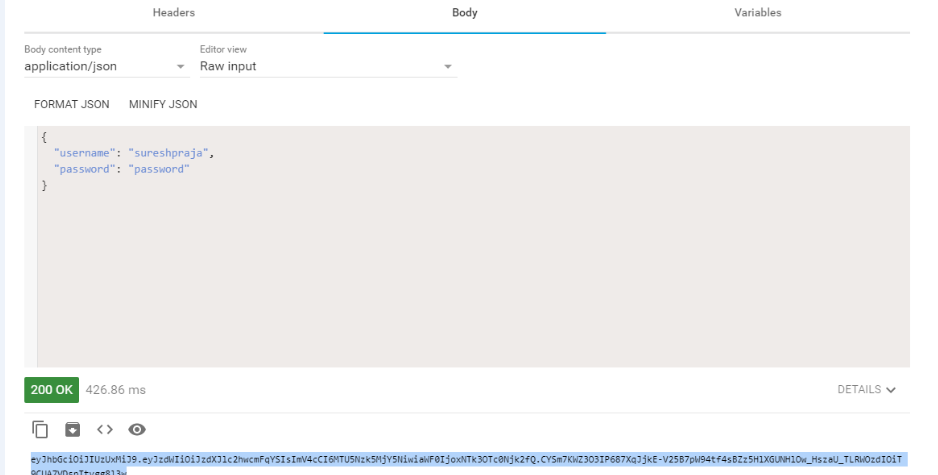
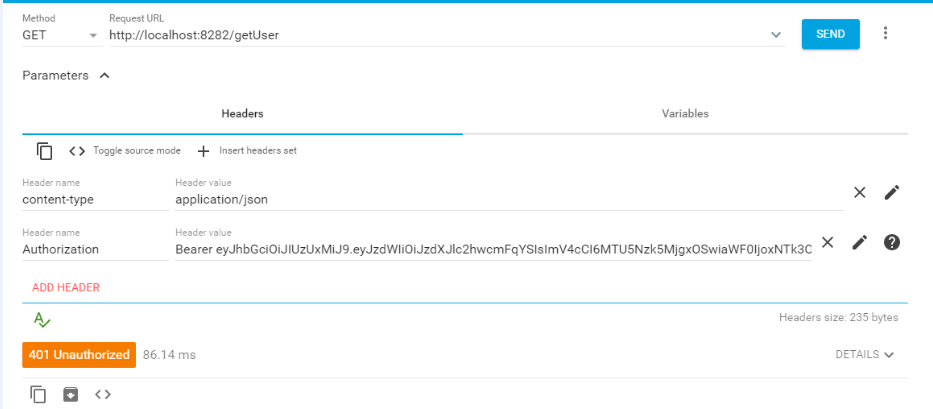
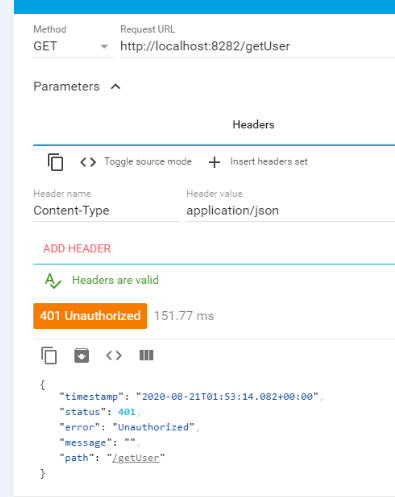
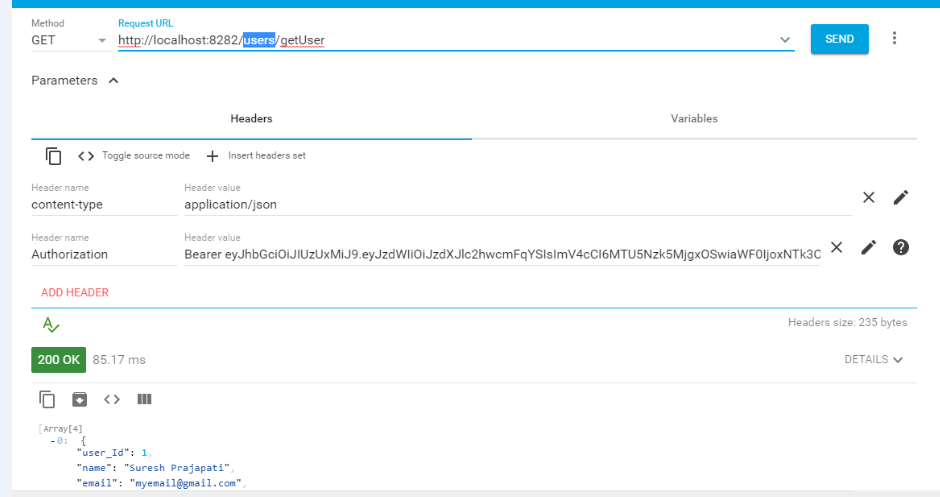
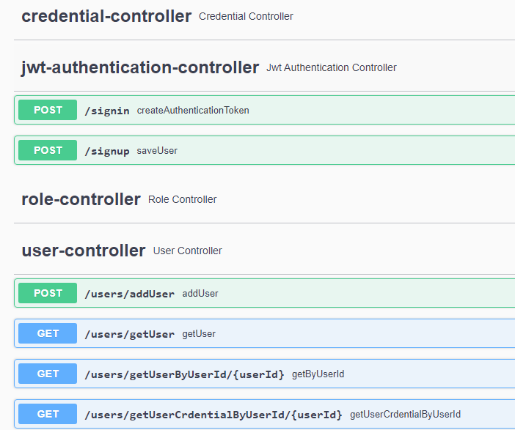
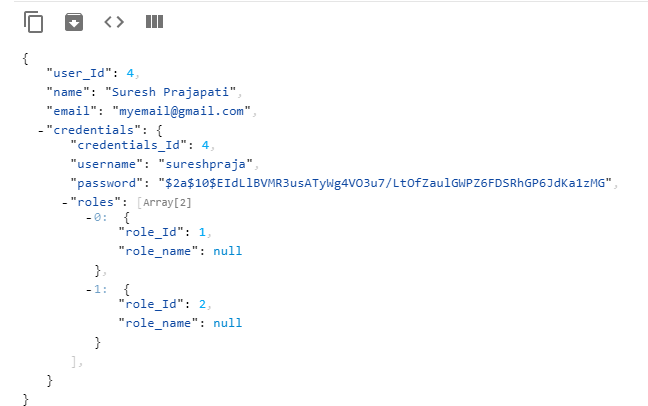
**JWT Authorization and PasswordEncoder(In ManyToMany RelationShip)**





**WebSecurityConfig.java**

**WebSecurityConfig.java**

**@Configuration**

**@EnableWebSecurity**

**public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {

@Autowired

**private** JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

@Autowired

**private** UserDetailsService jwtUserDetailsService;

@Autowired

**private** JwtRequestFilter jwtRequestFilter;

**@Autowired**

**private PasswordEncoder passwordEncoder;**

@Autowired

**public** **void** configureGlobal(AuthenticationManagerBuilder auth) **throws** Exception {

auth.userDetailsService(jwtUserDetailsService).passwordEncoder(passwordEncoder);

}

@Bean

@Override

**public** AuthenticationManager authenticationManagerBean() **throws** Exception {

**return** **super**.authenticationManagerBean();

}

@Override

**protected** **void** configure(HttpSecurity httpSecurity) **throws** Exception {

// We don't need CSRF for this example **httpSecurity.csrf().disable().cors().configurationSource(corsConfigurationSource()).and()**

// dont authenticate this particular request

**.authorizeRequests().antMatchers("/signin","/signup","/credentials/\*\*","/roles/\*\*","/swagger-ui/\*\*","/swagger-resources/\*\*","/v2/\*\*").permitAll().**

**antMatchers("/users/\*\*").hasAnyAuthority("USER","ADMIN").**

**anyRequest().authenticated().and().**

// make sure we use stateless session; session won't be used to store user's state.

**exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.*STATELESS*);**

// Add a filter to validate the tokens with every request

**httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);**

}

@Bean

**public** CorsConfigurationSource corsConfigurationSource() {

CorsConfiguration configuration = **new** CorsConfiguration();

configuration.setAllowedOrigins(Arrays.*asList*("\*"));

configuration.setAllowedMethods(Arrays.*asList*("GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"));

configuration.setAllowedHeaders(Arrays.*asList*("authorization", "content-type", "x-auth-token"));

configuration.setExposedHeaders(Arrays.*asList*("x-auth-token"));

UrlBasedCorsConfigurationSource source = **new** UrlBasedCorsConfigurationSource();

source.registerCorsConfiguration("/\*\*", configuration);

**return** source;

}

}

**JwtAuthenticationController.java**

@RestController

**public** **class** JwtAuthenticationController {

@Autowired

**private** AuthenticationManager authenticationManager;

@Autowired

**private** JwtTokenUtil jwtTokenUtil;

@Autowired

**private** JwtUserDetailsService userDetailsService;

@Autowired

UserService userService;

@Autowired

CredentialService credentialService;

@PostMapping(value = "/signin")

**public** ResponseEntity<?> createAuthenticationToken(@RequestBody Credentials authenticationRequest) **throws** Exception {

authenticate(authenticationRequest.getUsername(), authenticationRequest.getPassword());

**final** UserDetails userDetails = userDetailsService.loadUserByUsername(authenticationRequest.getUsername());

**final** String token = jwtTokenUtil.generateToken(userDetails);

**return** ResponseEntity.*ok*(token);

}

@PostMapping(value = "/signup")

**public** ResponseEntity<?> saveUser(@RequestBody User user) **throws** Exception {

**if** (credentialService.findByUserName(user.getCredentials().getUsername())==**null**) {

**return** ResponseEntity.*ok*(userDetailsService.save(user));

} **else** {

**return** **new** ResponseEntity<String>("{\"message\":\"User Already Used\"}", HttpStatus.***BAD\_REQUEST***);

}

}

**JwtUserDetailsService.java**

@Service

**public** **class** JwtUserDetailsService **implements** UserDetailsService {

@Autowired

UserRepository userRepository;

@Autowired

CredentialRepository credentialRepository;

@Autowired

**private** PasswordEncoder bcryptEncoder;

@Override

**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {

Credentials credential = credentialRepository.findByUsername(username);

**if** (credential == **null**) {

**throw** **new** UsernameNotFoundException("User not found with username: " + username);

}

**return** **new** org.springframework.security.core.userdetails.User(credential.getUsername(), credential.getPassword(),

*getAuthorities*(credential));

}

**private** **static** Collection<? **extends** GrantedAuthority> getAuthorities(Credentials credential) {

String[] userRoles = credential.getRoles().stream().map(x->x.getRole\_name()).toArray(String[]::**new**);

Collection<GrantedAuthority> authorities = AuthorityUtils.*createAuthorityList*(userRoles);

**return** authorities;

}

**public User save(User user) {**

**user.getCredentials().setPassword(bcryptEncoder.encode(user.getCredentials().getPassword()));**

**return userRepository.save(user);**

**}**}