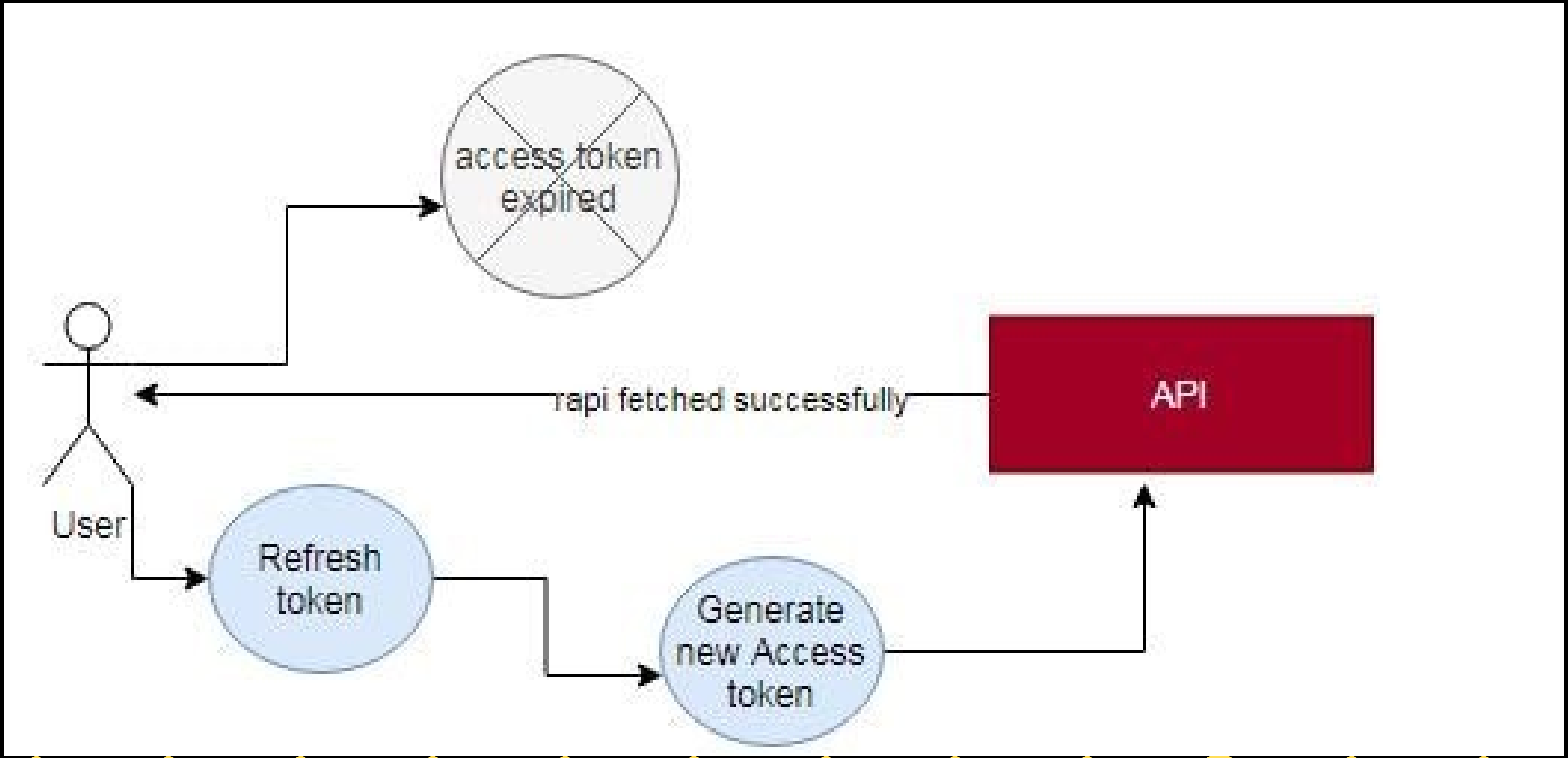
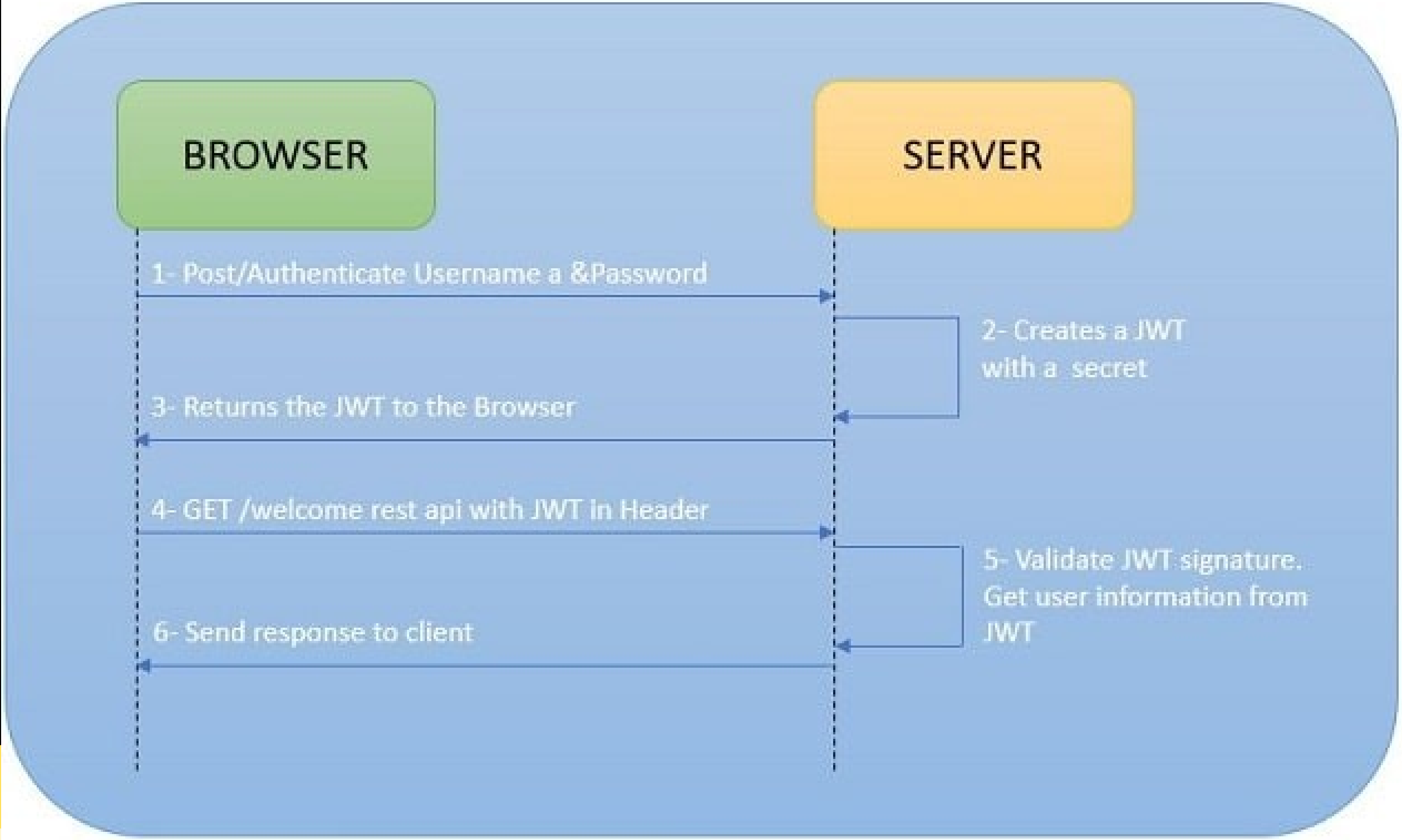
JSON Web Toolkit

JWT

JWT Architecture




BROWSER
SERVER
1- Post/Authenticate Username a &Password
2- Creates a JWT with a secret
3- Returns the JWT to the Browser
4- GET /welcome rest api with JWT in Header
5- Validate JWT signature. Get user information from JWT
6- Send response to client

JWT Maven dependency

- JSON Web Tokens (RFC 7519), is a standard that is mostly used for securing REST APIs
- The front end (client) firstly sends some credentials to authenticate itself (username and password in our case, since we're working on a web application).
- The server (the Spring app in our case) then checks those credentials, and if they are valid, it generates a JWT and returns it.
- After this step client has to provide this token in the request’s Authorization header in the “Bearer TOKEN” form. The back end will check the validity of this token and authorize or reject requests. The token may also store user roles and authorize the requests based on the given authorities.

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

JWT Maven dependency

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

- Generate JWT : Use /authenticate POST endpoint by using username and password to generate a JSON Web Token (JWT).
- Validate JWT : User can use /greeting GET endpoint by using valid JSON Web Token (JWT).

Gradle Dependency

```
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
     implementation 'org.springframework.boot:spring-boot-starter-security'
     implementation 'org.springframework.boot:spring-boot-starter-web'
     runtimeOnly 'mysql:mysql-connector-java'
     implementation group: 'javax.validation', name: 'validation-api', version: '2.0.1.Final'

     implementation group: 'com.gitee.wy3366', name: 'spring-boot-jwt', version: '1.0.0'
```

ERole.java

```java
public enum ERole {
    ROLE_USER,
    ROLE_MODERATOR,
    ROLE_ADMIN
}
```

JWT Token Utility

```
public static final long JWT_TOKEN_VALIDITY = 5 * 60 * 60;

@Value("$mysecrete.key")
private String sampleSecurityKey;

public String getUsernameFromToken(String token) {
return getClaimFromToken(token, Claims::getSubject);
}

public Date getIssuedAtDateFromToken(String token) {
return getClaimFromToken(token, Claims::getIssuedAt);
}

public Date getExpirationDateFromToken(String token) {
return getClaimFromToken(token, Claims::getExpiration);
}

public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {
final Claims claims = getAllClaimsFromToken(token);
return claimsResolver.apply(claims);
}

private Claims getAllClaimsFromToken(String token) {
return Jwts.parser().setSigningKey(sampleSecurityKey).parseClaimsJws(token).getBody();
}
```

JWT Token Utility

```
private Boolean isTokenExpired(String token) {
final Date expiration = getExpirationDateFromToken(token);
return expiration.before(new Date());
}

private Boolean ignoreTokenExpiration(String token) {
// here you specify tokens, for that the expiration is ignored
return false;
}

public String generateToken(UserDetails userDetails) {
Map<String, Object> claims = new HashMap<>();
return doGenerateToken(claims, userDetails.getUsername());
}

private String doGenerateToken(Map<String, Object> claims, String subject) {

return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))
.setExpiration(new Date(System.currentTimeMillis() + JWT_TOKEN_VALIDITY * 1000))
.signWith(SignatureAlgorithm.HS512, sampleSecurityKey).compact();
}

public Boolean canTokenBeRefreshed(String token) {
return (!isTokenExpired(token) || ignoreTokenExpiration(token));
}

public Boolean validateToken(String token, UserDetails userDetails) {
final String username = getUsernameFromToken(token);
return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
}
```

JWT Token Utility

```java
@Component
public class JwtUtils {
    private static final Logger logger = LoggerFactory.getLogger(JwtUtils.class);

    @Value("${bezkoder.app.jwtSecret}")
    private String jwtSecret;

    @Value("${bezkoder.app.jwtExpirationMs}")
    private int jwtExpirationMs;

    public String generateJwtToken(Authentication authentication) {

        UserDetailsImpl userPrincipal = (UserDetailsImpl) authentication.getPrincipal();

        return Jwts.builder()
                .setSubject((userPrincipal.getUsername()))
                .setIssuedAt(new Date())
                .setExpiration(new Date((new Date()).getTime() + jwtExpirationMs))
                .signWith(SignatureAlgorithm.HS512, jwtSecret)
                .compact();
    }

    public String getUserNameFromJwtToken(String token) {
        return Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody().getSubject();
    }

    public boolean validateJwtToken(String authToken) {
        try {
            Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(authToken);
            return true;
        }  catch (MalformedJwtException e) {
            logger.error("Invalid JWT token: {}", e.getMessage());
        } catch (ExpiredJwtException e) {
            logger.error("JWT token is expired: {}", e.getMessage());
        } catch (UnsupportedJwtException e) {
            logger.error("JWT token is unsupported: {}", e.getMessage());
        } catch (IllegalArgumentException e) {
            logger.error("JWT claims string is empty: {}", e.getMessage());
        }

        return false;
    }
}
```

Load Username and Password

- It searches the **username**, **password** and **GrantedAuthorities** for given user.
- This interface provide only one method called **loadUserByUsername**. **Authentication Manager** calls this method for getting the user details from the database when authenticating the user details provided by the user.
- Now we will using hard coded username password. Use BCrypt password, can use any online tool to BCrypt the password.

Load Username and Password

```
@Service
public class JwtUserDetailsService implements UserDetailsService{

@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
// TODO Auto-generated method stub
if ("tbarua1".equals(username)) {
return new User("tbarua1", "$2a$12$rT5KGSp7cpAQ3qku8MzKg.Fpay.Pe.yGbXCXKD.ujxSb80C5yak3W",new ArrayList<>());
} else {
throw new UsernameNotFoundException("User not found with username: " + username);
}
}

}
```

JWT Request Filter

- **JwtRequestFilter** class is executed for any incoming requests and validate JWT from the request and sets it in the context to indicate that logged in user is authenticated.

AuthController.java

```java
@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/auth")
public class AuthController {
    @Autowired
    AuthenticationManager authenticationManager;
    @Autowired
    UserRepository userRepository;
    @Autowired
    RoleRepository roleRepository;
    @Autowired
    PasswordEncoder encoder;
    @Autowired
    JwtUtils jwtUtils;
    private final Logger log= LoggerFactory.getLogger(AuthController.class);
    @PostMapping("/signin")
    public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {
    log.info("Request Received to signin "+loginRequest);
        Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
loginRequest.getPassword()));

        SecurityContextHolder.getContext().setAuthentication(authentication);
        String jwt = jwtUtils.generateJwtToken(authentication);

        UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrincipal();
        List<String> roles = userDetails.getAuthorities().stream()
                .map(item -> item.getAuthority())
                .collect(Collectors.toList());

        return ResponseEntity.ok(new JwtResponse(jwt,
                userDetails.getId(),
                userDetails.getUsername(),
                userDetails.getEmail(),
                roles));
    }
```

AuthController.java

```java
@PostMapping("/signup")
    public ResponseEntity<?> registerUser(@Valid @RequestBody SignupRequest signUpRequest) {
        log.info("Request Received signup "+signUpRequest);
        if (userRepository.existsByUsername(signUpRequest.getUsername())) {
            log.info("signup username already exists "+signUpRequest);
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: Username is already taken!"));
        }
        log.info("checking with email "+signUpRequest);
        if (userRepository.existsByEmail(signUpRequest.getEmail())) {
            log.info("user already exists by emailid "+signUpRequest);
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: Email is already in use!"));
        }
        log.info("All clear we can proceed to signup ");
        // Create new user's account
        User user = new User(signUpRequest.getUsername(),
                signUpRequest.getEmail(),
                encoder.encode(signUpRequest.getPassword()));

        Set<String> strRoles = signUpRequest.getRole();
        System.out.println("Roles received for "+strRoles);
        Set<Role> roles = new HashSet<>();

        if (strRoles == null) {
            Role userRole = roleRepository.findByName(ERole.ROLE_USER)
                    .orElseThrow(() -> new RuntimeException("Error: Role is not found."));
            log.info("found User role in database : "+userRole);
            roles.add(userRole);
        } else {
            strRoles.forEach(role -> {
                switch (role) {
                    case "admin":
                        Role adminRole = roleRepository.findByName(ERole.ROLE_ADMIN)
                                .orElseThrow(() -> new RuntimeException("Error: Role is not found."));
                        roles.add(adminRole);
                        log.info("Admin Role has been added ");
                        break;
                    case "mod":
                        Role modRole = roleRepository.findByName(ERole.ROLE_MODERATOR)
                                .orElseThrow(() -> new RuntimeException("Error: Role is not found."));
                        roles.add(modRole);
                        log.info("Mod Role has been added ");
                        break;
                    default:
                        Role userRole = roleRepository.findByName(ERole.ROLE_USER)
                                .orElseThrow(() -> new RuntimeException("Error: Role is not found."));
                        log.info("User Role has been added ");
                        roles.add(userRole);

                }
            });
        }

        user.setRoles(roles);
        log.info("saving user info in database");
        userRepository.save(user);

        return ResponseEntity.ok(new MessageResponse("User registered successfully!"));
    }
}
```

TestController.java

```
@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/test")
public class TestController {
    @GetMapping("/all")
    public String allAccess() {
        return "Public Content.";
    }

    @GetMapping("/user")
    @PreAuthorize("hasRole('USER') or hasRole('MODERATOR') or hasRole('ADMIN')")
    public String userAccess() {
        return "User Content.";
    }

    @GetMapping("/mod")
    @PreAuthorize("hasRole('MODERATOR')")
    public String moderatorAccess() {
        return "Moderator Board.";
    }

    @GetMapping("/admin")
    @PreAuthorize("hasRole('ADMIN')")
    public String adminAccess() {
        return "Admin Board.";
    }
}
```

AuthEntryPointJwt.java

```
@Component
public class AuthEntryPointJwt implements AuthenticationEntryPoint {

    private static final Logger logger = LoggerFactory.getLogger(AuthEntryPointJwt.class);

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response,
                         AuthenticationException authException) throws IOException,
ServletException {
        logger.error("Unauthorized error: {}", authException.getMessage());
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Error: Unauthorized");
    }

}
```

AuthTokenFilter.java

```java
public class AuthTokenFilter extends OncePerRequestFilter {
    @Autowired
    private JwtUtils jwtUtils;

    @Autowired
    private UserDetailsServiceImpl userDetailsService;

    private static final Logger logger = LoggerFactory.getLogger(AuthTokenFilter.class);

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        try {
            String jwt = parseJwt(request);
            if (jwt != null && jwtUtils.validateJwtToken(jwt)) {
                String username = jwtUtils.getUserNameFromJwtToken(jwt);

                UserDetails userDetails = userDetailsService.loadUserByUsername(username);
                UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities());
                authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

                SecurityContextHolder.getContext().setAuthentication(authentication);
            }
        } catch (Exception e) {
            logger.error("Cannot set user authentication: {}", e);
        }

        filterChain.doFilter(request, response);
    }

    private String parseJwt(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");

        if (StringUtils.hasText(headerAuth) && headerAuth.startsWith("Bearer ")) {
            return headerAuth.substring(7, headerAuth.length());
        }

        return null;
    }
}
```

JwtResponse.java

```
public class JwtResponse {
    private String jwt;
    private Long id;
    private String username;
    private String email;
    private List<String> roles;


   // Getter and Setter, Equals and Hash Code, ToString


    public JwtResponse(String jwt, Long id, String username, String email, List<String>
roles) {
        this.jwt = jwt;
        this.id = id;
        this.username = username;
        this.email = email;
        this.roles = roles;
    }
}
```

JwtResponse.java

```
public class LoginRequest {
    private String username;

    private String password;

// Getter and Setter, Equals and Hash Code, ToString

}
```

MessageResponse.java

```java
public class MessageResponse {
    private String msg;
    public MessageResponse(String s) {
        this.msg=s;
    }
// Getter and Setter, Equals and Hash Code, ToString

}
```

User.java

```
@Entity
@Table(   name = "users",
        uniqueConstraints = {
                @UniqueConstraint(columnNames = "username"),
                @UniqueConstraint(columnNames = "email")
        })
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    @Size(max = 20)
    private String username;

    @NotBlank
    @Size(max = 50)
    @Email
    private String email;

    @NotBlank
    @Size(max = 120)
    private String password;

    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(name = "user_roles",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<Role> roles = new HashSet<>();

// Getter and Setter, Equals and Hash Code, ToString

}
```

Role.java

```
@Entity
@Table(name = "roles")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Enumerated(EnumType.STRING)
    @Column(length = 20)
    private ERole name;

// Getter and Setter, Equals and Hash Code, ToString

}
```

RoleRepository.java

```java
package com.example.demo;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface RoleRepository extends JpaRepository<Role, Long> {
    Optional<Role> findByName(ERole name);
}
```

UserRepository.java

```java
package com.example.demo;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);

    Boolean existsByUsername(String username);

    Boolean existsByEmail(String email);
}
```

SignupRequest.java

```java
public class SignupRequest {
    private String username;
    private String email;
    private String password;
    private Set<String> roles = new HashSet<String>();


// Getter and Setter, Equals and Hash Code, ToString

}
```

UserDetailsImpl.java

```java
public class UserDetailsImpl implements UserDetails {
    private static final long serialVersionUID = 1L;
    private Long id;
    private String username;
    private String email;

    @JsonIgnore
    private String password;

    private Collection<? extends GrantedAuthority> authorities;

    public UserDetailsImpl(Long id, String username, String email, String password,
                           Collection<? extends GrantedAuthority> authorities) {
        this.id = id;
        this.username = username;
        this.email = email;
        this.password = password;
        this.authorities = authorities;
    }

    public static UserDetailsImpl build(User user) {
        List<GrantedAuthority> authorities = user.getRoles().stream()
                .map(role -> new SimpleGrantedAuthority(role.getName().name()))
                .collect(Collectors.toList());

        return new UserDetailsImpl(
                user.getId(),
                user.getUsername(),
                user.getEmail(),
                user.getPassword(),
                authorities);
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }
}
```

UserDetailsServiceImpl.java

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    UserRepository userRepository;

    @Override
    @Transactional
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("User Not Found with
username: " + username));

        return UserDetailsImpl.build(user);
    }

}
```

WebSecurityConfig.java

```java
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfigurerAdapter {
    @Autowired
    UserDetailsServiceImpl userDetailsService;

    @Autowired
    private AuthEntryPointJwt unauthorizedHandler;

    @Bean
    public AuthTokenFilter authenticationJwtTokenFilter() {
        return new AuthTokenFilter();
    }

    @Override
    public void configure(AuthenticationManagerBuilder authenticationManagerBuilder) throws Exception {
        authenticationManagerBuilder.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

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

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and().csrf().disable()
                .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
                .authorizeRequests().antMatchers("/api/auth/**").permitAll()
                .antMatchers("/api/test/**").permitAll()
                .anyRequest().authenticated();

        http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);
    }
}
```

DBOperationRunner.java

```java
@Component
public class DBOperationRunner implements CommandLineRunner {

    @Autowired
    private RoleRepository roleRepository;

    public DBOperationRunner() {
    }

    @Override
    public void run(String... args) throws Exception {
        Role ROLE_USER=new Role();
        ROLE_USER.setName(ERole.ROLE_USER);
        Role ROLE_MODERATOR=new Role();
        ROLE_MODERATOR.setName(ERole.ROLE_MODERATOR);
        Role ROLE_ADMIN=new Role();
        ROLE_ADMIN.setName(ERole.ROLE_ADMIN);
        roleRepository.save(ROLE_ADMIN);
        roleRepository.save(ROLE_USER);
        roleRepository.save(ROLE_MODERATOR);
    }
}
```