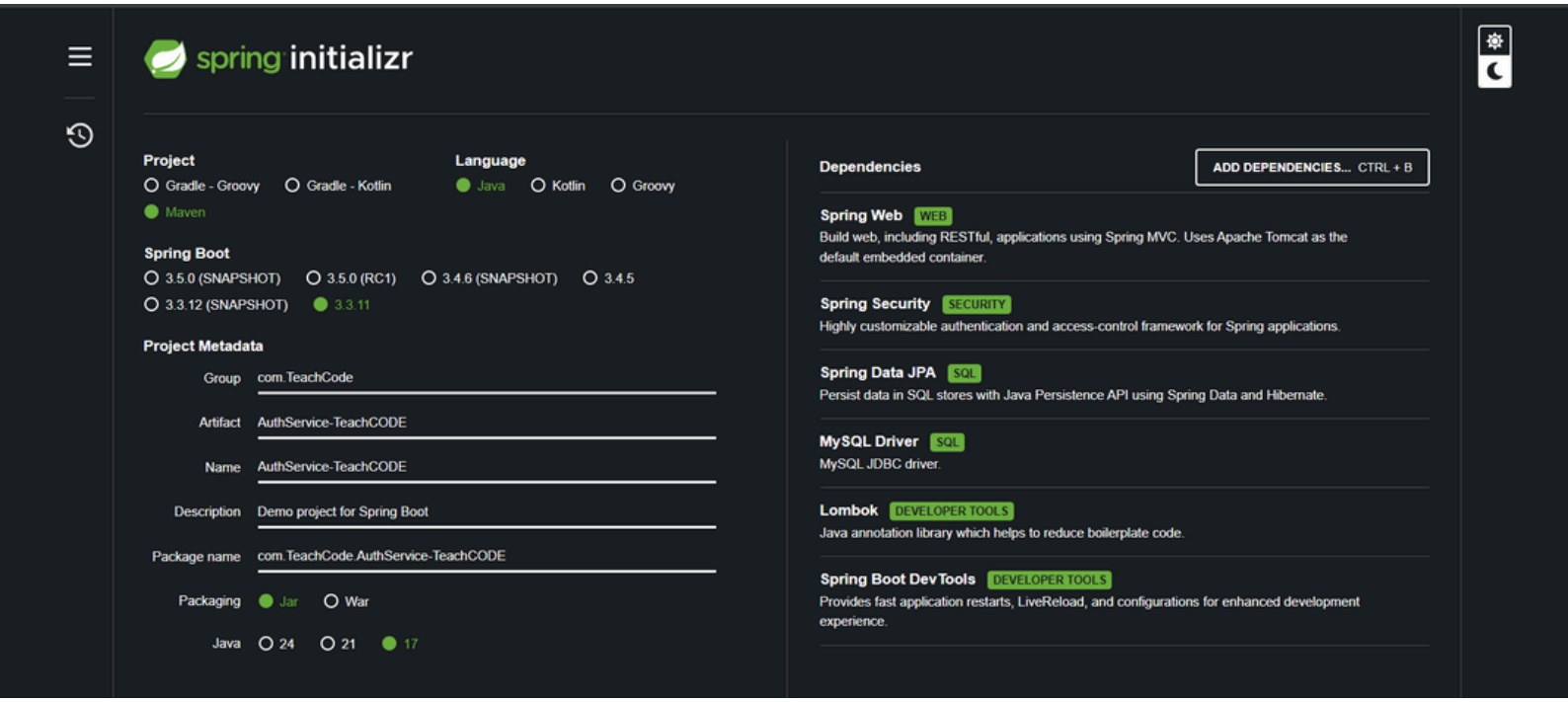
Td 1 : Spring Security

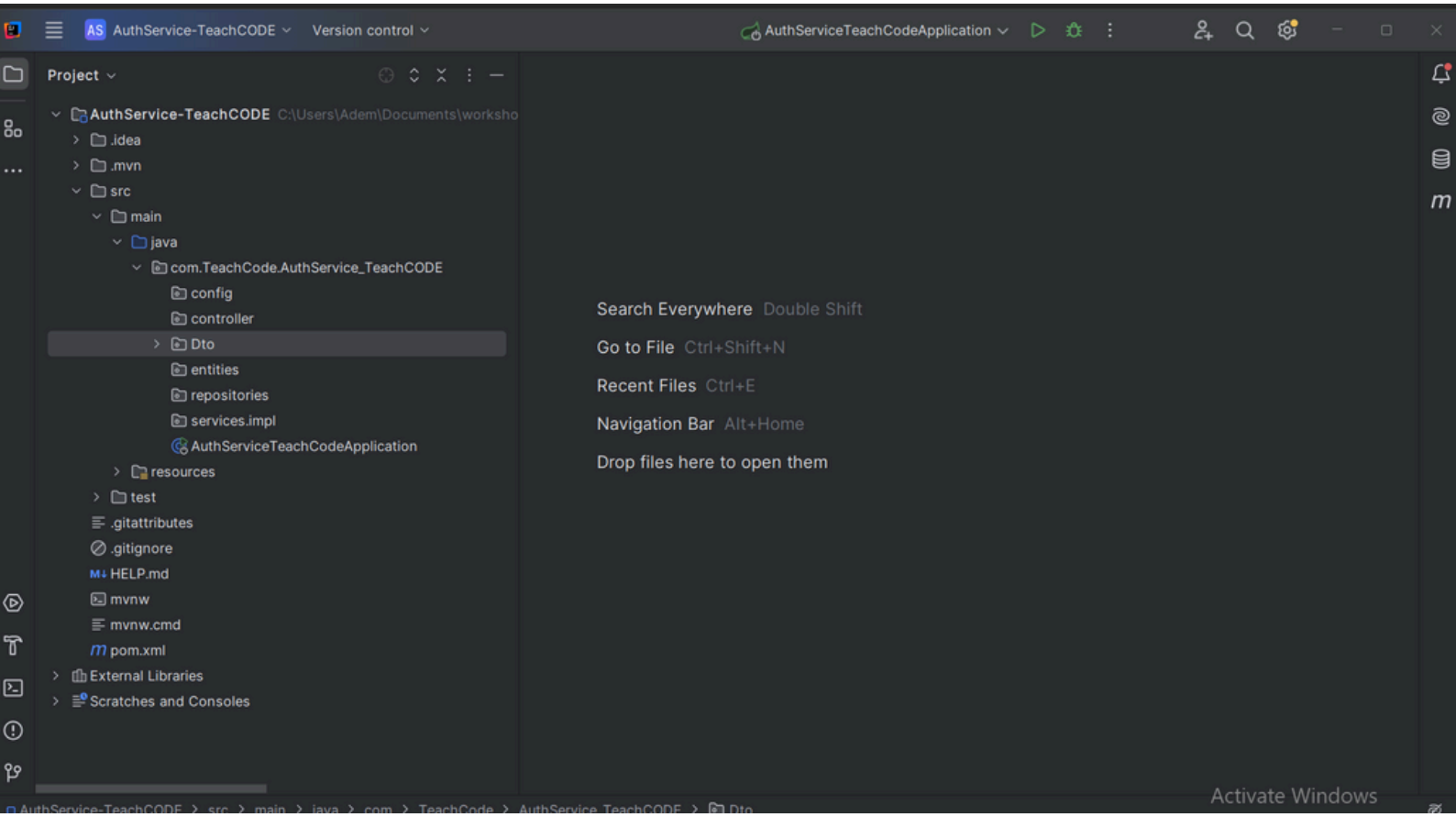
Key Features:

- User registration and authentication
- JWT token generation and validation
- Refresh token mechanism
- Role-based authorization (ADMIN, STUDENT, PARTNER, TRAINER)
- Password reset functionality

Create your Own Poject with Spring Initializr :



Create your own packages in your project



Entities Package

```java
@Data  Amira-Laabidi *
@Builder
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "Users")
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String name;

    @Column(unique = true, nullable = false)
    private String email;

    @Column(unique = true)
    private String phoneNumber;

    private String address;

    private Date dateOfBirth;

    private String password;

    @ElementCollection(fetch = FetchType.EAGER)
    @Enumerated(EnumType.STRING)
    private Set<Role> roles;
```

```java
// Add this method to get role names as strings
public Set<String> getRoleNames() {  no usages  new *
    return roles.stream()  Stream<Role>
            .map(Role::name)  Stream<String>
            .collect(Collectors.toSet());
}


@Override  1 usage  Amira-Laabidi
public Collection<? extends GrantedAuthority> getAuthorities() {
    return roles.stream()  Stream<Role>
            .map(role -> new SimpleGrantedAuthority(role.name()))  Stream<SimpleGrantedAuthority>
            .collect(Collectors.toList());
}

@Override  Amira-Laabidi
public String getUsername() {
    return email; // Email is used as the username
}
```

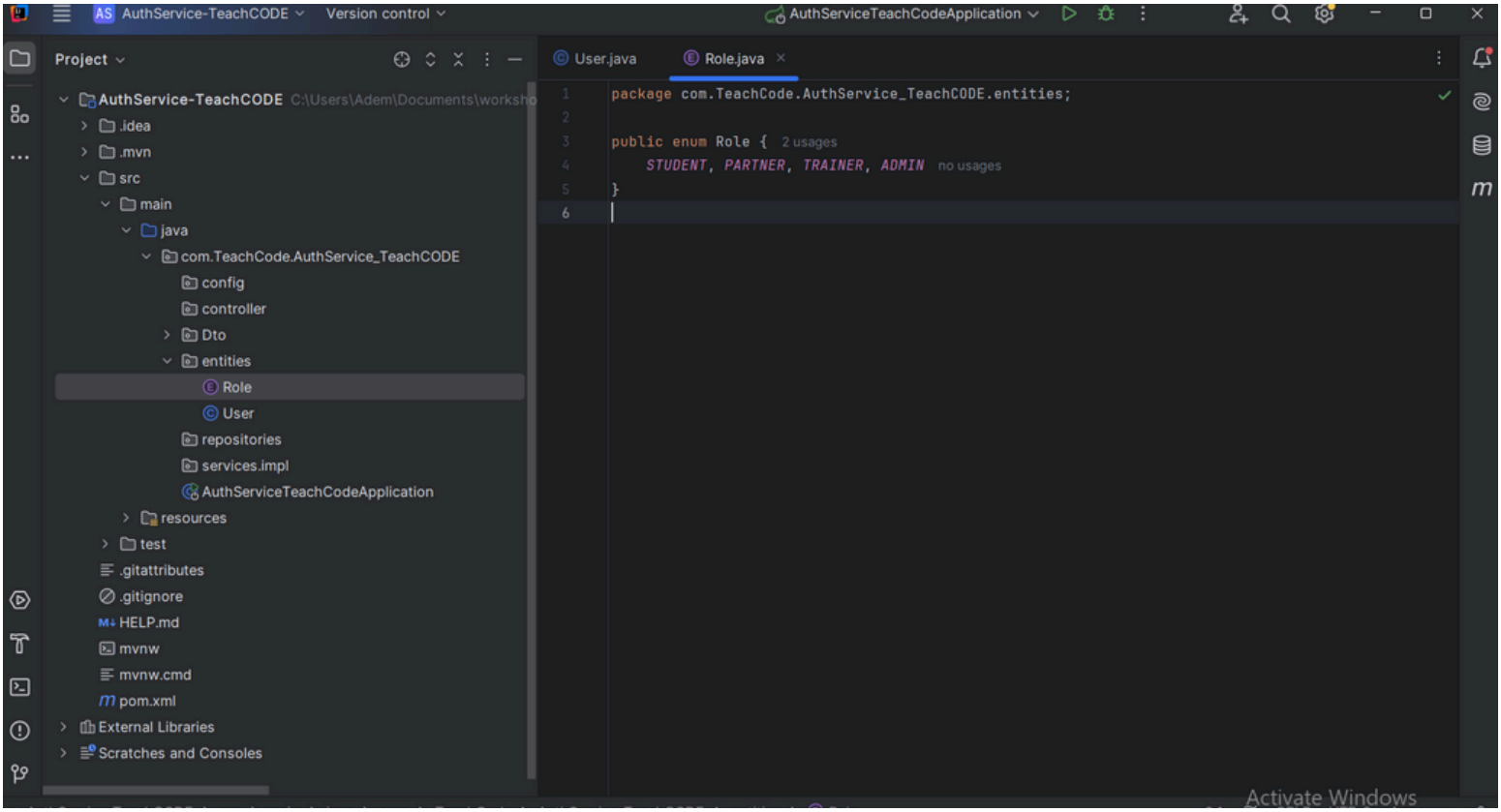
```java
@Override  no usages  Amira-Laabidi
public boolean isAccountNonExpired() {
    return true;
}

@Override  no usages  Amira-Laabidi
public boolean isAccountNonLocked() {
    return true;
}

@Override  no usages  Amira-Laabidi
public boolean isCredentialsNonExpired() {
    return true;
}

@Override  Amira-Laabidi
public boolean isEnabled() {
    return true;
}
```

Enum Role



UserRepository

```java
public interface UserRepository extends JpaRepository<User, Integer> {  no usages

    // Find user by email
    Optional<User> findByEmail(String email);  no usages
}
```

UserService

User.java | Role.java | UserRepository.java | UserService.java

```java
package com.TeachCode.AuthService_TeachCODE.services.impl;

import org.springframework.security.core.userdetails.UserDetailsService;

public interface UserService {  no usages
    // UserDetails contient les informations nécessaires pour créer
    // un objet d'authentification à partir de DAO ou d'une autre source de données de sécurité.
    //Cette déclaration de méthode définit une méthode appelée
    // userDetailsService qui retourne un objet de type UserDetailsService.
    UserDetailsService userDetailsService();  no usages
}

```

UserServiceImpl

User.java | Role.java | UserRepository.java | UserService.java | UserServiceImpl.java

```java
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {

    private final UserRepository userRepository;
    @Override  no usages
    public UserDetailsService userDetailsService() {
        return new UserDetailsService() {
            @Override  no usages
            public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
                //.orElseThrow(): C'est une méthode de la classe Optional qui permet de récupérer
                // la valeur contenue dans l'objet Optional. Si la valeur est présente, elle est retournée. Sinon, une exception est levée.
                //Une expression lambda est une fonction anonyme, c'est-à-dire une fonction sans nom,
                // qui peut être utilisée comme un argument ou retournée dans une méthode.
                return userRepository.findByEmail(username).orElseThrow(() -> new UsernameNotFoundException("User not found"));

            }
        };
    }
}
```

JwtService

```java
public interface JwtService {  no usages

    String extractUserName(String token);  no usages

    //Cette méthode est utilisée pour générer un jeton JWT. Elle prend un nom d'utilisateur en entrée,
    // crée un ensemble de claims  (e.g., subject, issued-at, expiration,
    // puis construit un jeton JWT en utilisant les claims et la clé de signature. Le jeton résultant est re
    String generateToken(UserDetails userDetails);  no usages

    boolean isTokenValid(String token, UserDetails userDetails);  no usages
}
```

configuration

```
spring.application.name=AuthService-TeachCODE
server.port=8081

# Database Configuration for phpMyAdmin
spring.datasource.url=jdbc:mysql://localhost:3306/AuthDb?createDatabaseIfNotExist=true&useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Hibernate/JPA Configuration
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.jpa.properties.hibernate.format_sql=true

# JWT Configuration
token.signing.key=413F4428472B4B6250655368566D597033733676397924422645294840 4D6351
jwt.refresh.expirationMs=86400000
```

JwtServiceImpl

```java
@Service
public class JwtServiceImpl implements JwtService {
    @Value("${token.signing.key}")
    private String jwtSigningKey;
    @Override 1 usage
    public String extractUserName(String token) { return extractClaim(token, Claims::getSubject); }

    @Override no usages
    public String generateToken(UserDetails userDetails) {

        return generateToken(new HashMap<>(), userDetails);
    }

    @Override no usages
    public boolean isTokenValid(String token, UserDetails userDetails) {
        final String userName = extractUserName(token);
        return (userName.equals(userDetails.getUsername())) && !isTokenExpired(token);
    }

    private <T> T extractClaim(String token, Function<Claims, T> claimsResolvers) { 2 usages
        final Claims claims = extractAllClaims(token);
        return claimsResolvers.apply(claims);
    }

    private String generateToken(Map<String, Object> extraClaims, UserDetails userDetails) { 1 usage
        return Jwts.builder().setClaims(extraClaims).setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 24))
                .signWith(getSigningKey(), SignatureAlgorithm.HS256).compact();
```

```java
    private String generateToken(Map<String, Object> extraClaims, UserDetails userDetails) { 1 usage
        return Jwts.builder().setClaims(extraClaims).setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 24))
                .signWith(getSigningKey(), SignatureAlgorithm.HS256).compact();
    }

    private boolean isTokenExpired(String token) { return extractExpiration(token).before(new Date()); }

    private Date extractExpiration(String token) { return extractClaim(token, Claims::getExpiration); }

    private Claims extractAllClaims(String token) { 1 usage
        return Jwts.parserBuilder().setSigningKey(getSigningKey()).build().parseClaimsJws(token)
                .getBody();
    }

    private Key getSigningKey() { 2 usages
        byte[] keyBytes = Decoders.BASE64.decode(jwtSigningKey);
        return Keys.hmacShaKeyFor(keyBytes);
    }
}
```

RefreshToken

```java

@Data  no usages
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity(name = "refreshtoekn")
public class RefreshToken {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @OneToOne
    @JoinColumn(name = "user_id",referencedColumnName = "id")
    private User user;

    @Column(nullable = false,unique = true)
    private String token;

    @Column(nullable = false)
    private Instant expirydate;


}
```

```java
public interface RefreshTokenService {  no usages


    public RefreshToken createRefreshToken(User user);  no usages
    public Optional<RefreshToken> findByToken(String token);  no usages
}
```

```java

import java.util.Optional;

public interface RefreshTokenRepository extends JpaRepository<RefreshToken, Integer> {  2 usages


    Optional<RefreshToken> findByUser(User user);  1 usage
    Optional<RefreshToken> findByToken(String token);  1 usage

}

```

```java
@Service
@RequiredArgsConstructor
public class RefreshTokenServiceImpl implements RefreshTokenService {

    @Value("86400000")
    private Long refreshTokenDuration;

    private final RefreshTokenRepository refreshTokenRepository;
    private final UserRepository userRepository;

    @Override  no usages
    public RefreshToken createRefreshToken(User user) {

        Optional<RefreshToken> existingToken = refreshTokenRepository.findByUser(user);

        RefreshToken refreshToken;
        if (existingToken.isPresent()) {
            refreshToken = existingToken.get();
            refreshToken.setToken(UUID.randomUUID().toString());
            refreshToken.setExpirydate(Instant.now().plusMillis(refreshTokenDuration));
        } else {
            refreshToken = new RefreshToken();
            refreshToken.setUser(user);
            refreshToken.setToken(UUID.randomUUID().toString());
            refreshToken.setExpirydate(Instant.now().plusMillis(refreshTokenDuration));
        }

        return refreshTokenRepository.save(refreshToken);
    }

```

```java
@Override  no usages
public Optional<RefreshToken> findByToken(String token) { return refreshTokenRepository.findByToken(token); }

@Override  no usages
public RefreshToken verifyExpiration(RefreshToken token) {
    if(token.getExpirydate().isBefore(Instant.now())){
        refreshTokenRepository.delete(token);
        throw new TokenRefreshException(token.getToken(),"Refresh token expired");
    }
    return token;
}
```

TokenRefreshException

```java
@ResponseStatus(HttpStatus.FORBIDDEN)  no usages
public class TokenRefreshException extends RuntimeException {
    private static final long serialVersionUID = 1L;  no usages

    public TokenRefreshException(String token,String message){  no usages
        super(String.format("Failed for [%s]: %s",token,message));
    }
}
```

JwtAuthenticationFilter

```java
@Component // Assure que Spring détecte et gère cette classe
@RequiredArgsConstructor// Lombok génère un constructeur avec les dépendances injectées
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    @Autowired
    JwtService jwtService;
    @Autowired
    UserService userService;

    @Override  no usages
    protected void doFilterInternal(@NonNull HttpServletRequest request,
                                    @NonNull HttpServletResponse response,
                                    @NonNull FilterChain filterChain)
            throws ServletException, IOException {

        final String authHeader = request.getHeader( s: "Authorization");
        final String jwt;
        final String userEmail;

        if (StringUtils.isEmpty(authHeader) || !StringUtils.startsWithIgnoreCase(authHeader, prefix: "Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }

        jwt = authHeader.substring( beginIndex: 7);
        userEmail = jwtService.extractUserName(jwt);

        if (StringUtils.hasText(userEmail) && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = userService.userDetailsService().loadUserByUsername(userEmail);
            if (jwtService.isTokenValid(jwt, userDetails)) {
```

```java
            if (jwtService.isTokenValid(jwt, userDetails)) {
                SecurityContext context = SecurityContextHolder.createEmptyContext();
                UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                        userDetails, credentials: null, userDetails.getAuthorities());
                authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                context.setAuthentication(authToken);
                SecurityContextHolder.setContext(context);
            }
        }

        filterChain.doFilter(request, response);
    }
}
```

SecurityConfiguration

```java

@Configuration
//L'annotation @EnableWebSecurity active la sécurité Web de Spring et configure les aspects
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfiguration {

    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    private final UserService userService;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http.csrf(AbstractHttpConfigurer::disable)
                //Une liste blanche des requêtes {/api/v1/auth/**},
                .authorizeHttpRequests(request -> request.requestMatchers("/api/v1/auth/**")
                        //toute autre requête doit être authentifiée
                        .permitAll().anyRequest().authenticated())

                //Gestion sans état,
                // ce qui signifie que nous ne devrions pas stocker l'état d'authentification
                .sessionManagement(manager -> manager.sessionCreationPolicy(STATELESS))
                //DaoAuthenticationProvider - qui est responsable de récupérer
                // les informations utilisateur
                // et d'encoder/décoder les mots de passe.
                //Ajout de JwtAuthenticationFilter avant UsernamePasswordAuthenticationFilter
                // car nous extrayons le nom d'utilisateur et le mot de passe,
                //puis les mettons à jour dans SecurityContextHolder dans JwtAuthenticationFilter.
                .authenticationProvider(authenticationProvider()).addFilterBefore(
                        jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
```



```java
    @Bean
    //Définition du bean passwordEncoder que Spring utilisera pour décoder les mots de passe
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }

    @Bean
    //Définition du bean authenticationProvider utilisé lors du processus d'authentification
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userService.userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }

    @Bean
    //Définition du bean authentication manager.
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config)
            throws Exception {
        return config.getAuthenticationManager();
    }
}
```

AuthenticationService

```java
package com.TeachCode.AuthService_TeachCODE.services;

public interface AuthenticationService {  no usages


    JwtAuthenticationResponse SignUp(SignUpRequest request);  no usages
    JwtAuthenticationResponse SignIn(SingninRequest request);  no usages
}
```

AuthenticationServiceImpl

```java
@Service
@RequiredArgsConstructor
public class AuthenticationServiceImpl implements AuthenticationService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final AuthenticationManager authenticationManager;
    private final JwtService jwtService;
    private final RefreshTokenService refreshTokenService;

    @Override  no usages
    public JwtAuthenticationResponse SignUp(SignUpRequest request) {
        // Assign default role if no roles are provided
        Set<Role> assignedRoles = request.getRoles() != null ? request.getRoles() : Set.of(Role.STUDENT);

        // Create new User entity
        var user = User.builder()
                .name(request.getName())
                .email(request.getEmail())
                .password(passwordEncoder.encode(request.getPassword()))
                .phoneNumber(request.getPhoneNumber())
                .address(request.getAddress())
                .dateOfBirth(request.getDateOfBirth())
                .roles(assignedRoles) // Updated to handle multiple roles
                .build();
```

```
        // Save user to the database
        userRepository.save(user);

        // Generate JWT & Refresh Token
        var jwt = jwtService.generateToken(user);
        var refreshToken = refreshTokenService.createRefreshToken(user);

        return JwtAuthenticationResponse.builder()
                .token(jwt)
                .refreshToken(refreshToken.getToken())
                .role(assignedRoles.toString()) // Convert Set<Role> to String
                .userId(user.getId())
                .build();
    }

    @Override no usages
```

```
@Override no usages
public JwtAuthenticationResponse SignIn(SinginRequest request) {
    // Authenticate user
    authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(request.getEmail(), request.getPassword()));

    // Fetch user from DB
    var user = userRepository.findByEmail(request.getEmail())
            .orElseThrow(() -> new IllegalArgumentException("Invalid email or password"));

    // Generate JWT & Refresh Token
    var jwt = jwtService.generateToken(user);
    var refreshToken = refreshTokenService.createRefreshToken(user);

    return JwtAuthenticationResponse.builder()
            .token(jwt)
            .refreshToken(refreshToken.getToken())
            .role(user.getRoles().toString()) // Convert Set<Role> to String
            .userId(user.getId())
            .build();
}
```

AuthenticationController

```java
@RestController
@RequestMapping("/api/v1/auth")
@RequiredArgsConstructor
public class AuthenticationController {
    private final AuthenticationService authenticationService;

    private final RefreshTokenService refreshTokenService;

    private final JwtServiceImpl jwtService;

    @PostMapping("/signup")
    public ResponseEntity<JwtAuthenticationResponse> signup(@RequestBody SignUpRequest request) {
        return ResponseEntity.ok(authenticationService.SignUp(request));
    }

    @PostMapping("/signin")
    public ResponseEntity<JwtAuthenticationResponse> signin(@RequestBody SinginRequest request, HttpServletResponse response) {
        JwtAuthenticationResponse jwtResponse = authenticationService.SignIn(request);

        if (jwtResponse != null && jwtResponse.getToken() != null) {
            // Set the token in the response header
            response.setHeader( s: "Access-Control-Expose-Headers", s1: "Authorization");
            response.setHeader( s: "Access-Control-Allow-Headers", s1: "Authorization, X-Pingother, Origin, X-Requested-with, " +
                    "Content-Type, Accept, X-Custom-header");
            response.setHeader( s: "Authorization", s1: "Bearer " + jwtResponse.getToken());
            // Return a response with user details in the body
            JSONObject responseBody = new JSONObject();
            responseBody.put("userID", jwtResponse.getUserId());
            responseBody.put("role", jwtResponse.getRole());
            return ResponseEntity.ok(jwtResponse);
        } else {
```



```java
        } else {
            return ResponseEntity.badRequest().body(jwtResponse); // Assuming jwtResponse can be null
        }
    }

    @PostMapping("/refresh-token")
    public ResponseEntity<JwtAuthenticationResponse> refreshToken(@RequestBody Map<String, String> request) {
        String refreshToken = request.get("refreshToken");
        RefreshToken token = refreshTokenService.findByToken(refreshToken).orElseThrow(
                () -> new TokenRefreshException(refreshToken, "Refresh token not found")
        );

        String newToken = jwtService.generateToken(token.getUser());

        return ResponseEntity.ok(JwtAuthenticationResponse.builder().token(newToken).refreshToken(refreshToken).build());
    }
}
```

AuthorizationController

```java
@RestController
@RequestMapping("/api/v1/resource")
@RequiredArgsConstructor
public class AuthorizationController {
    @GetMapping
    public ResponseEntity<String> sayHello() { return ResponseEntity.ok( body: "Here is your resource"); }
}
```