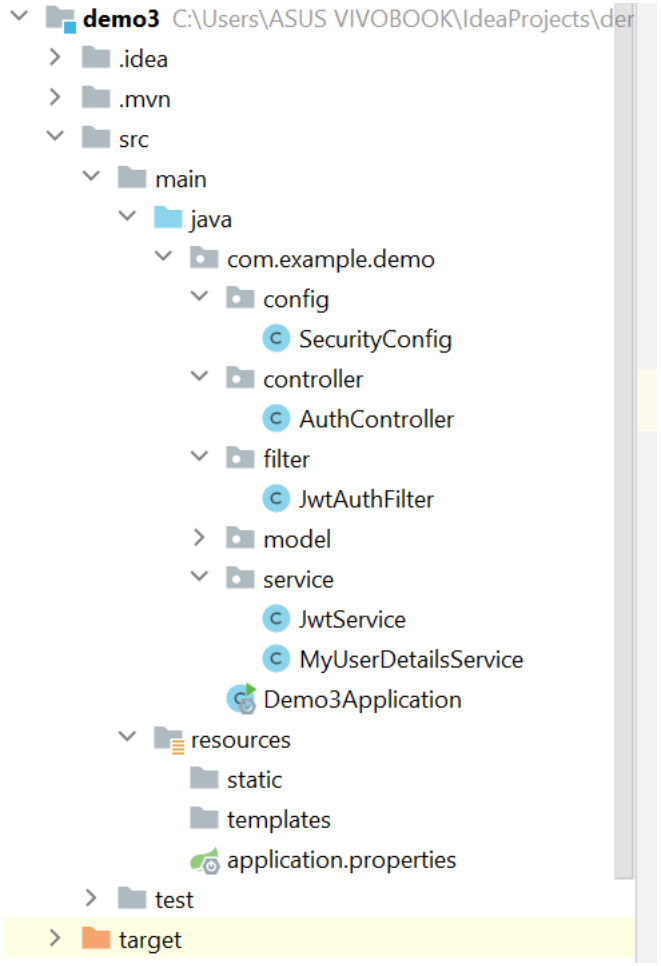
Implémentation avec Spring Boot

- **Structure du projet**



Implémentation avec Spring Boot

- **Les dépendances**

```
<!-- JWT (jjwt) -->
<dependency>
   <groupId>io.jsonwebtoken</groupId>
   <artifactId>jjwt-api</artifactId>
   <version>0.11.5</version>
</dependency>
<dependency>
   <groupId>io.jsonwebtoken</groupId>
   <artifactId>jjwt-impl</artifactId>
   <version>0.11.5</version>
   <scope>runtime</scope>
</dependency>
<dependency>
   <groupId>io.jsonwebtoken</groupId>
   <artifactId>jjwt-jackson</artifactId>
   <version>0.11.5</version>
   <scope>runtime</scope>
</dependency>
```

Implémentation avec Spring Boot

- **Securityconfig.java:** définir les règles de sécurité

```
@Configuration
@EnableWebSecurity  // optionnel mais aide l'IDE et la lisibilité
public class SecurityConfig {
   private final JwtAuthFilter jwtAuthFilter;
   public SecurityConfig(JwtAuthFilter jwtAuthFilter) {
      this.jwtAuthFilter = jwtAuthFilter;
   }
   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(reg -> reg
                  .requestMatchers("/api/auth/**").permitAll()
                  .anyRequest().authenticated()
            )
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);
      return http.build();
   }
   @Bean
   public AuthenticationManager authenticationManager(AuthenticationConfiguration cfg) throws Exception {
      return cfg.getAuthenticationManager();
   }
}
```

Implémentation avec Spring Boot

- **Authcontroller.java :** gérer la connexion et générer le token

```
@RestController
@RequestMapping("/api")
public class AuthController {
   private final AuthenticationManager authenticationManager;
   private final JwtService jwtService;
   private final UserDetailsService userDetailsService;
   public AuthController(AuthenticationManager authenticationManager, JwtService jwtService, UserDetailsService uds) {
      this.authenticationManager = authenticationManager;
      this.jwtService = jwtService;
      this.userDetailsService = uds;
   }
   @PostMapping("/auth/login")
   public AuthResponse login(@RequestBody AuthRequest req) {
      var auth = new UsernamePasswordAuthenticationToken(req.username(), req.password());
      authenticationManager.authenticate(auth);
      UserDetails user = userDetailsService.loadUserByUsername(req.username());
      String token = jwtService.generateToken(user.getUsername(), Map.of("roles", user.getAuthorities()));
      return new AuthResponse(token);
   }
   @GetMapping("/hello")
   public Map<String, String> hello() {
      return Map.of("message", "Bonjour, endpoint protégé OK ✅");
   }
}
```

Implémentation avec Spring Boot

- **Jwtauthfilter.java :** Intercepter chaque requête et valider le token

```
@Component
public class JwtAuthFilter extends OncePerRequestFilter {
    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;
    public JwtAuthFilter(JwtService jwtService, UserDetailsService userDetailsService) {
        this.jwtService = jwtService;
        this.userDetailsService = userDetailsService;
    }
```

Implémentation avec Spring Boot

- **Jwtauthfilter.java (Suite):** Intercepter chaque requête et valider le token

```
@Override
  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws ServletException, IOException {
    final String authHeader = request.getHeader("Authorization");
    if (authHeader == null || !authHeader.startsWith("Bearer ")) {
      chain.doFilter(request, response);
      return;
    }
    final String token = authHeader.substring(7);
    String username = null;
    try {
      username = jwtService.extractUsername(token);
    } catch (Exception e) {
      chain.doFilter(request, response);
      return;
    }
    if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
      UserDetails user = userDetailsService.loadUserByUsername(username);
      if (jwtService.isTokenValid(token, user.getUsername())) {
        var authToken = new UsernamePasswordAuthenticationToken(user, null, user.getAuthorities());
        authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
        SecurityContextHolder.getContext().setAuthentication(authToken);
      }
    }
    chain.doFilter(request, response);
  }
}
```

Implémentation avec Spring Boot

- **Jwtservice.java :** générer, signer et vérifier les tokens

```
@Service
public class JwtService {
   private final Key key;
   private final long expirationMs;
   public JwtService(
         @Value("${app.jwt.secret}") String secret,
         @Value("${app.jwt.expiration-ms}") long expirationMs
   ) {
      this.key = Keys.hmacShaKeyFor(secret.getBytes());
      this.expirationMs = expirationMs;
   }
   public String generateToken(String username, Map<String, Object> claims) {
      Date now = new Date();
      Date exp = new Date(now.getTime() + expirationMs);
      return Jwts.builder()
            .setClaims(claims)
            .setSubject(username)
            .setIssuedAt(now)
            .setExpiration(exp)
            .signWith(key, SignatureAlgorithm.HS256)
            .compact();
   }
```

Implémentation avec Spring Boot

- **Jwtservice.java (Suite) :** générer, signer et vérifier les tokens

```
public String extractUsername(String token) {
      return parse(token).getBody().getSubject();
   }
   public boolean isTokenValid(String token, String username) {
      try {
         return extractUsername(token).equals(username) && !isExpired(token);
      } catch (JwtException e) {
         return false;
      }
   }
   private boolean isExpired(String token) {
      return parse(token).getBody().getExpiration().before(new Date());
   }
   private Jws<Claims> parse(String token) {
      return Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token);
   }
}
```

Implémentation avec Spring Boot

- **MyUserdetailsService.java :** définir les utilisateurs disponibles

```
@Service
public class MyUserDetailsService implements UserDetailsService {

    private final InMemoryUserDetailsManager delegate;

    public MyUserDetailsService() {
        UserDetails user = User.withUsername("user")
                .password("{noop}password")
                .roles("USER")
                .build();
        UserDetails admin = User.withUsername("admin")
                .password("{noop}admin123")
                .roles("ADMIN")
                .build();

        this.delegate = new InMemoryUserDetailsManager(user, admin);
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return delegate.loadUserByUsername(username);
    }
}
```

Implémentation avec Spring Boot

- **AuthRequest / AuthResponse :** Simples classes pour la requête et la réponse du login.

```
public record AuthResponse(String token) {}
public record AuthRequest(String username, String password) {}
```

- **Application.properties**.

```
spring.application.name=demo3
server.port=8080
# Secret (exemple simple pour la démo : en prod, charge depuis les variables
d'environnement)
app.jwt.secret=ChangeThisSecretToAStrongOne_32chars_min
app.jwt.expiration-ms=3600000
```