**Assignment Wk05-06: Spring and Maven**

**Problem statement –**

Company XYZ is in TA [ Talent acquisition ] business and is in need of a backend services for integrating with various UI components. It wants to publish Api(s) which will return JSON to be integrated with UI components. It already has UI components and only the api(s) need to be developed.

**Following are the requirements –**

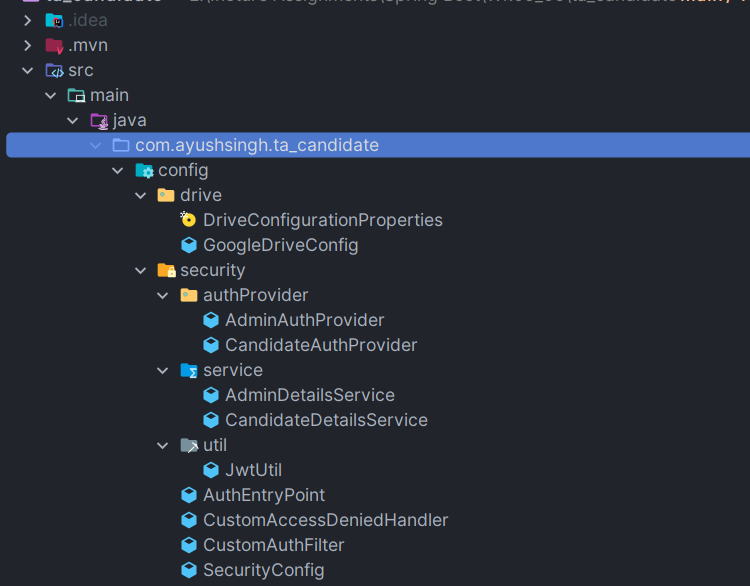
1. Facility for adding resumes of candidates

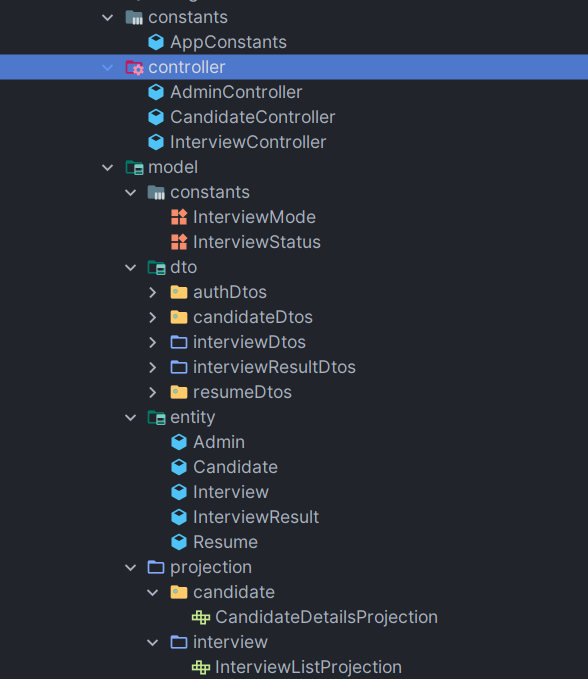
2. Facility for scheduling interviews

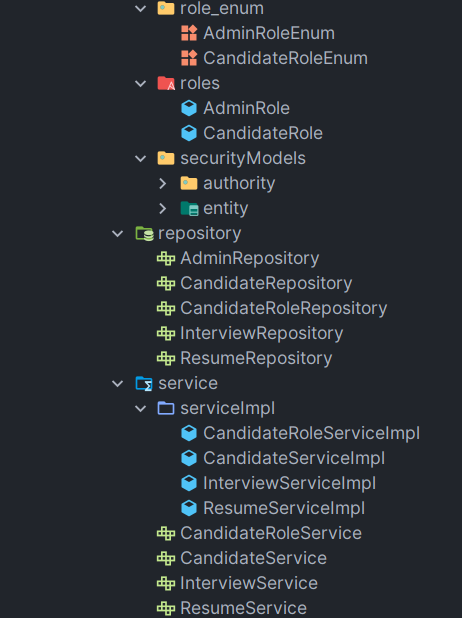
3. Facility for linking the resumes with interview results

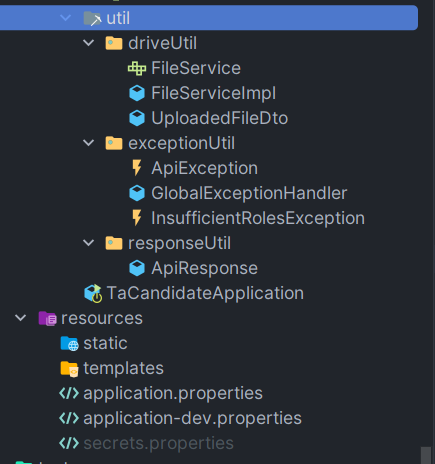
4. Facility for search and look up of a candidate and his / her interview result

# Project Structure









# Entities in the project

1. Candidate: Candidate profile

*package* com.ayushsingh.ta\_candidate.model.entity;  
  
*import* com.ayushsingh.ta\_candidate.model.roles.CandidateRole;  
*import* jakarta.persistence.\*;  
*import* lombok.*AllArgsConstructor*;  
*import* lombok.*Getter*;  
*import* lombok.*NoArgsConstructor*;  
*import* lombok.*Setter*;  
*import* org.hibernate.annotations.*CreationTimestamp*;  
*import* org.hibernate.annotations.*UpdateTimestamp*;  
*import* org.springframework.data.annotation.*CreatedDate*;  
*import* org.springframework.data.annotation.*LastModifiedDate*;  
  
*import* java.util.Date;  
*import* java.util.HashSet;  
*import* java.util.Set;  
*import* java.util.UUID;  
  
*@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table*(name="ta\_candidate")  
*public class* Candidate {  
  
 *@Id  
 @GeneratedValue*(strategy = GenerationType.IDENTITY)  
 *@Column*(name = "candidate\_id")  
 *private* Long candidateId;  
  
 *@Column*(name="candidate\_token",nullable = *false*,unique = *true*)  
 *private* String candidateToken;  
  
 *@Column*(name="first\_name",nullable = *false*)  
 *private* String firstName;  
  
 *@Column*(name="last\_name")  
 *private* String lastName;  
  
 *@Column*(name="password")  
 *private* String password;  
  
 *@Column*(name="email")  
 *private* String email;  
  
 *@OneToOne*(mappedBy = "candidate",cascade = CascadeType.ALL,orphanRemoval = *true*)  
 *private* Resume resume;  
  
 *@ManyToMany*(fetch = FetchType.EAGER)  
 *@JoinTable*(name = "ta\_candidate\_candidate\_role", joinColumns = *@JoinColumn*(name = "candidate\_id", referencedColumnName = "candidate\_id"), inverseJoinColumns = *@JoinColumn*(name = "role\_id", referencedColumnName = "role\_id"))  
 *private* Set<CandidateRole> roles;  
  
 *@OneToMany*(mappedBy = "candidate",cascade = CascadeType.ALL,fetch = FetchType.LAZY,orphanRemoval = *true*)  
 *private* Set<Interview> interviews=*new* HashSet<>();  
  
 *@CreatedDate  
 @CreationTimestamp  
 @Column*(name = "created\_at", nullable = *false*, updatable = *false*)  
 *private* Date createdAt;  
  
 *@LastModifiedDate  
 @UpdateTimestamp  
 @Column*(name = "updated\_at")  
 *private* Date updatedAt;  
  
 *@PrePersist  
 public void* generateToken() {  
 *if* (*this*.candidateToken==*null*) {  
 *this*.candidateToken=UUID.randomUUID().toString();  
 }  
 }  
}

1. Admin: Admin of the talent acquisition system

*package* com.ayushsingh.ta\_candidate.model.entity;  
  
*import* com.ayushsingh.ta\_candidate.model.roles.AdminRole;  
*import* jakarta.persistence.\*;  
*import* lombok.*AllArgsConstructor*;  
*import* lombok.*Getter*;  
*import* lombok.*NoArgsConstructor*;  
*import* lombok.*Setter*;  
*import* org.hibernate.annotations.*CreationTimestamp*;  
*import* org.hibernate.annotations.*UpdateTimestamp*;  
*import* org.springframework.data.annotation.*CreatedDate*;  
*import* org.springframework.data.annotation.*LastModifiedDate*;  
  
*import* java.util.Date;  
*import* java.util.Set;  
*import* java.util.UUID;  
  
*@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table*(name = "ta\_admin")  
*public class* Admin {  
  
 *@Id  
 @GeneratedValue*(strategy = GenerationType.IDENTITY)  
 *@Column*(name = "admin\_id")  
 *private* Long adminId;  
  
 *@Column*(name = "admin\_token", nullable = *false*, unique = *true*)  
 *private* String adminToken;  
  
 *@Column*(name = "admin\_email", nullable = *false*, unique = *true*)  
 *private* String adminEmail;  
  
 *@Column*(name = "admin\_password", nullable = *false*)  
 *private* String adminPassword;  
  
 *@ManyToMany*(fetch = FetchType.EAGER)  
 *@JoinTable*(name = "ta\_admin\_admin\_role", joinColumns = *@JoinColumn*(name = "admin\_id", referencedColumnName = "admin\_id"), inverseJoinColumns = *@JoinColumn*(name = "role\_id", referencedColumnName = "role\_id"))  
 *private* Set<AdminRole> roles;  
  
 *@CreatedDate  
 @CreationTimestamp  
 @Column*(name = "created\_at", nullable = *false*, updatable = *false*)  
 *private* Date createdAt;  
  
 *@LastModifiedDate  
 @UpdateTimestamp  
 @Column*(name = "updated\_at")  
 *private* Date updatedAt;  
  
 *@PrePersist  
 public void* generateToken() {  
 *if* (*this*.adminToken == *null*) {  
 *this*.adminToken = UUID.randomUUID().toString();  
 }  
 }  
}

1. AdminRole and CandidateRole: Roles for admin and candidate

*package* com.ayushsingh.ta\_candidate.model.roles;  
  
*import* com.ayushsingh.ta\_candidate.model.entity.Admin;  
*import* jakarta.persistence.\*;  
*import* lombok.*AllArgsConstructor*;  
*import* lombok.*Getter*;  
*import* lombok.*NoArgsConstructor*;  
*import* lombok.*Setter*;  
  
*import* java.util.Objects;  
*import* java.util.Set;  
  
*@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Table*(name = "ta\_admin\_role")  
*@Entity  
public class* AdminRole {  
 *@Id  
 @GeneratedValue*(strategy = GenerationType.IDENTITY)  
 *@Column*(name = "role\_id")  
 *private* Long roleId;  
  
 *@Column*(name = "role", nullable = *false*, unique = *true*)  
 *private* String roleName;  
  
 *@ManyToMany*(mappedBy = "roles")  
 *private* Set<Admin> admins;  
  
 *@Override  
 public boolean* equals(Object o) {  
 *if* (*this* == o) *return true*;  
 *if* (o == *null* || getClass() != o.getClass()) *return false*;  
 AdminRole adminRole = (AdminRole) o;  
 *return* Objects.equals(roleId, adminRole.roleId) && Objects.equals(roleName, adminRole.roleName);  
 }  
  
 *@Override  
 public int* hashCode() {  
 *return* Objects.hash(roleId, roleName);  
 }  
}

*package* com.ayushsingh.ta\_candidate.model.roles;  
  
*import* com.ayushsingh.ta\_candidate.model.entity.Candidate;  
*import* jakarta.persistence.\*;  
*import* lombok.*AllArgsConstructor*;  
*import* lombok.*Getter*;  
*import* lombok.*NoArgsConstructor*;  
*import* lombok.*Setter*;  
  
*import* java.util.Objects;  
*import* java.util.Set;  
  
*@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Table*(name = "ta\_candidate\_role")  
*@Entity  
public class* CandidateRole {  
 *@Id  
 @GeneratedValue*(strategy = GenerationType.IDENTITY)  
 *@Column*(name = "role\_id")  
 *private* Long roleId;  
  
 *@Column*(name = "role", nullable = *false*, unique = *true*)  
 *private* String roleName;  
  
 *@ManyToMany*(mappedBy = "roles")  
 *private* Set<Candidate> candidates;  
  
 *@Override  
 public boolean* equals(Object o) {  
 *if* (*this* == o) *return true*;  
 *if* (o == *null* || getClass() != o.getClass()) *return false*;  
 CandidateRole that = (CandidateRole) o;  
 *return* Objects.equals(roleId, that.roleId) && Objects.equals(roleName, that.roleName) && Objects.equals(candidates, that.candidates);  
 }  
  
 *@Override  
 public int* hashCode() {  
 *return* Objects.hash(roleName);  
 }  
}

1. Interview: Interview details

*package* com.ayushsingh.ta\_candidate.model.entity;  
  
*import* com.ayushsingh.ta\_candidate.model.constants.InterviewMode;  
*import* com.ayushsingh.ta\_candidate.model.constants.InterviewStatus;  
*import* jakarta.persistence.\*;  
*import* lombok.*AllArgsConstructor*;  
*import* lombok.*Getter*;  
*import* lombok.*NoArgsConstructor*;  
*import* lombok.*Setter*;  
*import* org.hibernate.annotations.*CreationTimestamp*;  
*import* org.hibernate.annotations.*UpdateTimestamp*;  
*import* org.springframework.data.annotation.*CreatedDate*;  
*import* org.springframework.data.annotation.*LastModifiedDate*;  
  
*import* java.time.LocalDateTime;  
*import* java.time.ZonedDateTime;  
*import* java.util.Date;  
  
*import* java.util.UUID;  
  
*@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Table*(name="ta\_interview")  
*@Entity  
public class* Interview {  
  
 *@Id  
 @GeneratedValue*(strategy = GenerationType.IDENTITY)  
 *@Column*(name="interview\_id")  
 *private* Long interviewId;  
  
 *@Column*(name = "interview\_token",nullable = *false*,unique = *true*)  
 *private* String interviewToken;  
  
 *@Column*(name = "interview\_subject",nullable = *false*,length = 500)  
 *private* String interviewSubject;  
  
  
 *@Column*(name = "meet\_link",nullable = *false*,length = 500)  
 *private* String meetLink;  
  
 *@ManyToOne*(cascade = {CascadeType.MERGE,CascadeType.PERSIST,CascadeType.REFRESH},fetch = FetchType.LAZY)  
 *@JoinColumn*(name="candidate\_id",referencedColumnName = "candidate\_id")  
 *private* Candidate candidate;  
  
 *@OneToOne*(mappedBy = "interview",cascade = CascadeType.ALL,orphanRemoval = *true*)  
 *private* InterviewResult interviewResult;  
  
 *@Column*(name = "meet\_time")  
 *private* ZonedDateTime meetTime;  
  
 *@Column*(name="interview\_status",nullable = *false*)  
 *@Enumerated*(value=EnumType.STRING)  
 *private* InterviewStatus interviewStatus;  
  
 *@Column*(name="interview\_mode",nullable = *false*)  
 *@Enumerated*(EnumType.STRING)  
 *private* InterviewMode interviewMode;  
  
 *@CreatedDate  
 @CreationTimestamp  
 @Column*(name = "created\_at", nullable = *false*, updatable = *false*)  
 *private* Date createdAt;  
  
 *@LastModifiedDate  
 @UpdateTimestamp  
 @Column*(name = "updated\_at")  
 *private* Date updatedAt;  
  
 *@PrePersist  
 public void* generateToken() {  
 *if* (*this*.interviewToken==*null*) {  
 *this*.interviewToken= UUID.randomUUID().toString();  
 }  
 }  
}

1. InterviewResult: Result of interview

*package* com.ayushsingh.ta\_candidate.model.entity;  
  
*import* jakarta.persistence.\*;  
*import* lombok.*AllArgsConstructor*;  
*import* lombok.*Getter*;  
*import* lombok.*NoArgsConstructor*;  
*import* lombok.*Setter*;  
*import* org.hibernate.annotations.*CreationTimestamp*;  
*import* org.hibernate.annotations.*UpdateTimestamp*;  
*import* org.springframework.data.annotation.*CreatedDate*;  
*import* org.springframework.data.annotation.*LastModifiedDate*;  
  
*import* java.util.Date;  
*import* java.util.UUID;  
  
*@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table*(name = "ta\_interview\_result")  
*public class* InterviewResult {  
  
 *@Id  
 @GeneratedValue*(strategy = GenerationType.IDENTITY)  
 *@Column*(name = "interview\_result\_id")  
 *private* Long interviewResultId;  
  
 *@Column*(name="interview\_result\_token",nullable = *false*,unique = *true*)  
 *private* String interviewResultToken;  
  
 *@OneToOne*(fetch = FetchType.LAZY)  
 *@JoinColumn*(name = "interview\_id",referencedColumnName = "interview\_id")  
 *private* Interview interview;  
  
 *@Column*(name="feedback",nullable = *false*,length = 500)  
 *private* String feedback;  
  
 *@Column*(name="decision",nullable = *false*)  
 *private* String decision;  
  
 *@CreatedDate  
 @CreationTimestamp  
 @Column*(name = "created\_at", nullable = *false*, updatable = *false*)  
 *private* Date createdAt;  
  
 *@LastModifiedDate  
 @UpdateTimestamp  
 @Column*(name = "updated\_at")  
 *private* Date updatedAt;  
  
 *@PrePersist  
 public void* generateToken() {  
 *if* (*this*.interviewResultToken == *null*) {  
 *this*.interviewResultToken = UUID.randomUUID().toString();  
 }  
 }  
  
}

1. Resume: Resume details

*package* com.ayushsingh.ta\_candidate.model.entity;  
  
*import* jakarta.persistence.\*;  
*import* lombok.*AllArgsConstructor*;  
*import* lombok.*Getter*;  
*import* lombok.*NoArgsConstructor*;  
*import* lombok.*Setter*;  
*import* org.hibernate.annotations.*CreationTimestamp*;  
*import* org.hibernate.annotations.*UpdateTimestamp*;  
*import* org.springframework.data.annotation.*CreatedDate*;  
*import* org.springframework.data.annotation.*LastModifiedDate*;  
  
*import* java.util.Date;  
*import* java.util.UUID;  
  
*@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Table*(name="ta\_resume")  
*@Entity  
public class* Resume {  
  
 *@Id  
 @GeneratedValue*(strategy = GenerationType.IDENTITY)  
 *@Column*(name="document\_id")  
 *private* Long documentId;  
  
 *@Column*(name="alert\_token",nullable = *false*,unique = *true*)  
 *private* String documentToken;  
  
 *@Column*(name="document\_name",nullable = *false*)  
 *private* String documentName;  
  
 *@Column*(name="format",nullable = *false*)  
 *private* String format;  
  
 *@Column*(name="document\_url",nullable = *false*)  
 *private* String documentUrl;  
  
 *@OneToOne*(cascade = {CascadeType.MERGE,CascadeType.PERSIST,CascadeType.DETACH},fetch = FetchType.LAZY)  
 *@JoinColumn*(name="candidate\_id",nullable = *false*)  
 *private* Candidate candidate;  
  
 *@CreatedDate  
 @CreationTimestamp  
 @Column*(name = "created\_at", nullable = *false*, updatable = *false*)  
 *private* Date createdAt;  
  
 *@LastModifiedDate  
 @UpdateTimestamp  
 @Column*(name = "updated\_at")  
 *private* Date updatedAt;  
  
}

## Utility classes and enums

1. AdminRoleEnum and CandidateRole Enum represent the admin and candidate roles.
2. InterviewMode and InterviewStatus represent the mode and status of interview respectively.
3. Projections are used to selectively fetch data from database tables.

# Security Configuration

1. Spring Security with Json Web Token is used to protect API.
2. The APIs for login and register are kept publicly accessible.
3. There are two types of users- SUPER\_ADMIN and CANDIDATE. Therefore, role based access is used to further apply authorization.

## Dependencies

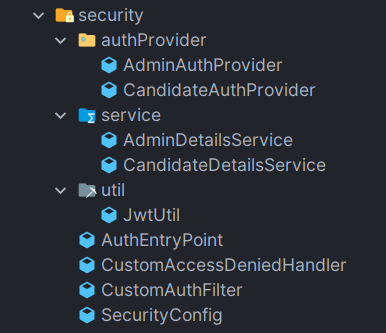
1. Add the Spring Security dependency.

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>

1. Add the JWT dependency

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

## Configuration



1. SecurityConfig: Configuration of Spring Security.

*package* com.ayushsingh.ta\_candidate.config.security;  
  
*@Configuration  
@EnableWebSecurity  
@EnableMethodSecurity  
@RequiredArgsConstructor  
public class* SecurityConfig {  
  
  
 *private final* AuthenticationEntryPoint authEntryPoint;  
 *private final* CandidateAuthProvider candidateAuthProvider;  
 *private final* AdminAuthProvider adminAuthProvider;  
 *private final* CustomAccessDeniedHandler customAccessDeniedHandler;  
  
  
 *public* AuthenticationManager authManager(HttpSecurity http) *throws* Exception {  
 AuthenticationManagerBuilder authenticationManagerBuilder = http.getSharedObject(AuthenticationManagerBuilder.*class*);  
 authenticationManagerBuilder.authenticationProvider(candidateAuthProvider).authenticationProvider(adminAuthProvider);  
  
 *return* authenticationManagerBuilder.build();  
 }  
  
  
 *@Bean  
 public* CustomAuthFilter customAuthenticationFilter(HttpSecurity http) *throws* Exception {  
 *return new* CustomAuthFilter(authManager(http));  
 }  
  
  
 *@Bean  
 public* SecurityFilterChain filterChain(HttpSecurity http) *throws* Exception {  
  
 http.cors().configurationSource(request -> {  
 CorsConfiguration configuration = *new* CorsConfiguration();  
 configuration.setAllowedOrigins(List.of("\*"));  
 configuration.setAllowedMethods(List.of("HEAD", "GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"));  
 configuration.setAllowCredentials(*true*);  
 configuration.addExposedHeader("Message");  
 configuration.setAllowedHeaders(List.of("Authorization", "Cache-Control", "Content-Type"));  
 *return* configuration;  
 })  
 .and()  
 .csrf()  
 .disable()  
 .authorizeHttpRequests()  
 .requestMatchers(AppConstants.PUBLIC\_URLS).permitAll()  
 .anyRequest().authenticated()  
 .and()  
 .exceptionHandling()  
 .authenticationEntryPoint(authEntryPoint)  
 .accessDeniedHandler(customAccessDeniedHandler)  
 .and()  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);  
  
 http.addFilterBefore(customAuthenticationFilter(http), UsernamePasswordAuthenticationFilter.*class*);  
  
 *return* http.build();  
 }  
}

1. CustomAuthFilter: Custom filter to handle login functionality and jwt verification.

*@Slf4j  
public class* CustomAuthFilter *extends* OncePerRequestFilter {  
  
 *private final* AuthenticationManager authenticationManager;  
 *@Autowired  
 @Qualifier*("handlerExceptionResolver")  
 *private* HandlerExceptionResolver exceptionResolver;  
  
 *@Autowired  
 private* CandidateDetailsService candidateDetailsService;  
  
 *@Autowired  
 private* AdminDetailsService adminDetailsService;  
  
  
 *public* CustomAuthFilter(AuthenticationManager authenticationManager) {  
 *this*.authenticationManager = authenticationManager;  
 }  
  
 *@Override  
 protected void* doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) *throws* ServletException, IOException {  
 String uri = request.getRequestURI(); *//-get the uri  
 //-if the uri is a login uri, then login  
 if* (uri.endsWith(AppConstants.SIGN\_IN\_URI\_ENDING)) {  
 *//-obtain username and password* LoginRequestDto jwtAuthRequest = *new* ObjectMapper().readValue(request.getInputStream(), LoginRequestDto.*class*);  
 String username = jwtAuthRequest.getUsername();  
 String password = jwtAuthRequest.getPassword();  
 UsernamePasswordAuthenticationToken authenticationToken = *new* UsernamePasswordAuthenticationToken(username, password);  
 Authentication authenticationResult = *null*;  
 *try* {  
 authenticationResult = *this*.authenticationManager.authenticate(authenticationToken);  
  
 } *catch* (AuthenticationException e) {  
  
 SecurityContextHolder.getContext().setAuthentication(UsernamePasswordAuthenticationToken.unauthenticated(username, password));  
 exceptionResolver.resolveException(request, response, *null*, e);  
 }  
 *if* (authenticationResult != *null*) {  
 SecurityContextHolder.getContext().setAuthentication(authenticationResult);  
 }  
  
 filterChain.doFilter(request, response);  
 }  
 *//-if not a login uri, check for access token  
 else* {  
 String headerToken =*null*;  
 headerToken = request.getHeader(AUTH\_HEADER); *//-if no token, obtain token from header  
 //-if still not found, return  
 if* (headerToken == *null*) {  
 log.info("Access token is not present");  
 *//-match uri with public urls  
 try*{  
 *boolean* isPublicUrl = Arrays.stream(PUBLIC\_URLS).anyMatch(uri::endsWith);  
 *if*(isPublicUrl) {  
 filterChain.doFilter(request, response);  
 *return*;  
 }  
 *else*{  
 *throw new* ApiException("Access token is not present");  
 }  
 }  
 *catch* (ApiException e){  
 exceptionResolver.resolveException(request, response, *null*, e);  
 *return*;  
 }  
 }  
 UserDetails userDetails = *null*;  
 *try* {  
 headerToken = StringUtils.delete(headerToken, AppConstants.BEARER\_TOKEN\_PREFIX).trim();  
 String entityType = JwtUtil.extractEntityType(headerToken);  
 String username = JwtUtil.extractUsername(headerToken);  
 *if* (username != *null* && SecurityContextHolder.getContext().getAuthentication() == *null*) {  
 *if* (entityType.equals(AppConstants.ENTITY\_TYPE\_CANDIDATE)) {  
 userDetails = *this*.candidateDetailsService.loadUserByUsername(username);  
 }  
 *else if*(entityType.equals(AppConstants.ENTITY\_TYPE\_ADMIN)){  
 userDetails = *this*.adminDetailsService.loadUserByUsername(username);  
 }  
 *if* (userDetails == *null*) {  
 *throw new* ApiException("User not found with username: " + username);  
 } *else if* (JwtUtil.validateToken(headerToken, userDetails)) {  
 UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = *new* UsernamePasswordAuthenticationToken(userDetails, *null*, userDetails.getAuthorities());  
 usernamePasswordAuthenticationToken.setDetails(*new* WebAuthenticationDetailsSource().buildDetails(request));  
 SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);  
 filterChain.doFilter(request, response);  
 } *else* {  
 *throw new* ApiException("Token validation returned false");  
 }  
 } *else* {  
 *throw new* ApiException("Username not found in token");  
 }  
 } *catch* (ExpiredJwtException | AccessDeniedException | UnsupportedJwtException | MalformedJwtException |  
 SignatureException | IllegalArgumentException | ApiException | InsufficientRolesException e) {  
 SecurityContextHolder.getContext().setAuthentication(UsernamePasswordAuthenticationToken.unauthenticated(userDetails, *null*));  
 exceptionResolver.resolveException(request, response, *null*, e);  
 }  
 }  
 }  
  
  
}

1. JwtUtil: Utility class to handle JWT.

*public class* JwtUtil {  
  
 *public static* String extractUsername(String token) {  
 String subject = extractClaim(token, Claims::getSubject);  
 System.out.println("Extracted subject: " + subject);  
 *return* subject;  
 }  
  
 *public static* String extractEntityType(String token){  
 *final* Claims claims=extractAllClaims(token);  
 *return* (String) claims.get(AppConstants.ENTITY\_TYPE);  
 }  
  
 *public static* Date extractExpiration(String token) {  
 *return* extractClaim(token, Claims::getExpiration);  
 }  
  
 *public static* <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {  
 *final* Claims claims = extractAllClaims(token);  
 *return* claimsResolver.apply(claims);  
 }  
  
 *private static* Claims extractAllClaims(String token) {  
  
 Claims parsedClaims = Jwts.parser().setSigningKey(AppConstants.SECRET\_KEY).parseClaimsJws(token).getBody();  
 System.out.println("Parsed Claims: " + parsedClaims.getSubject());  
 *return* parsedClaims;  
 }  
  
 *private static* Boolean isTokenExpired(String token) {  
 *return* extractExpiration(token).before(*new* Date());  
 }  
  
 *public static* String generateToken(String username, String entityType) {  
 Map<String, Object> claims = *new* HashMap<>();  
 claims.put(AppConstants.ENTITY\_TYPE,entityType);  
 *return* createToken(claims, username);  
 }  
  
 *private static* String createToken(Map<String, Object> claims, String subject) {  
 Date issueDate = *new* Date(System.currentTimeMillis());  
 System.out.println("issueDate: " + issueDate + " time: " + issueDate.getTime() + " issueDate formatted: "  
 + issueDate);  
 Date expirationDate = *new* Date(System.currentTimeMillis() + AppConstants.ACCESS\_TOKEN\_EXPIRATION\_TIME  
 );  
 System.out.println("Expiration date: " + expirationDate + " formatted: " + expirationDate);  
 *return* Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(issueDate)  
 .setExpiration(expirationDate)  
 .signWith(SignatureAlgorithm.HS256, AppConstants.SECRET\_KEY).compact()  
 ;  
 }  
  
  
 *public static* Boolean validateToken(String token, UserDetails userDetails) {  
 *final* String username = extractUsername(token);  
 *boolean* isUsernameValid = username.equals(userDetails.getUsername());  
 *boolean* isJwtTtokenExpired = isTokenExpired(token);  
 System.out.println("Is token expired: " + isJwtTtokenExpired + " is username valid: " + isUsernameValid);  
 *if* (!isUsernameValid) {  
 System.out.println("Username in the token is invalid");  
 }  
 *if* (isJwtTtokenExpired) {  
 System.out.println("Token is expired!");  
 }  
 *return* (isUsernameValid && !isJwtTtokenExpired);  
 }  
  
 *public static* String[] decodedBase64(String token) {  
  
 *byte*[] decodedBytes = Base64.getDecoder().decode(token);  
 String pairedCredentials = *new* String(decodedBytes);  
  
 *return* pairedCredentials.split(":", 2);  
  
 }  
  
}

1. AdminDetailsService and CandidateDetailsService: UserDetailsService implementations for Admin and Candidate

*package* com.ayushsingh.ta\_candidate.config.security.service;  
  
  
  
*@Service  
@RequiredArgsConstructor  
public class* AdminDetailsService *implements* UserDetailsService {  
  
 *private final* AdminRepository adminRepository;  
 *@Override  
 public* UserDetails loadUserByUsername(String username) *throws* UsernameNotFoundException {  
 Optional<Admin> admin =adminRepository.findByAdminEmail(username);  
  
 *return* admin.map(SecurityAdmin::*new*).orElse(*null*);  
 }  
}

*package* com.ayushsingh.ta\_candidate.config.security.service;  
  
  
*@Service  
@RequiredArgsConstructor  
public class* CandidateDetailsService *implements* UserDetailsService {  
  
  
 *private final* CandidateRepository candidateRepository;  
 *@Override  
 public* UserDetails loadUserByUsername(String username) *throws* UsernameNotFoundException {  
 Optional<Candidate> candidate = candidateRepository.findByCandidateEmail(username);  
  
 *return* candidate.map(SecurityCandidate::*new*).orElse(*null*);  
 }  
}

1. AdminAuthProvider and CandidateAuthProvider: AuthenticationProvider implementations for Admin and Candidate

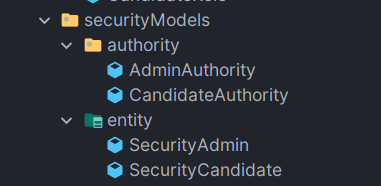
*package* com.ayushsingh.ta\_candidate.config.security.authProvider;  
  
  
*@RequiredArgsConstructor  
@Component  
public class* AdminAuthProvider *implements* AuthenticationProvider {  
  
 *private final* AdminDetailsService adminDetailsService;  
 *private final* PasswordEncoder passwordEncoder;  
  
  
 *@Override  
 public* Authentication authenticate(Authentication authentication) *throws* AuthenticationException {  
 String username = String.valueOf(authentication.getPrincipal());  
 String password = String.valueOf(authentication.getCredentials());  
  
 UserDetails adminDetails = adminDetailsService.loadUserByUsername(username);  
 *if* (adminDetails != *null*) {  
 *if* (passwordEncoder.matches(password, adminDetails.getPassword())) {  
  
 *return new* UsernamePasswordAuthenticationToken(username, password, adminDetails.getAuthorities());  
  
 }  
 }  
 *throw new* BadCredentialsException("Wrong Credentials");  
 }  
  
 *@Override  
 public boolean* supports(Class<?> authentication) {  
 *return* UsernamePasswordAuthenticationToken.*class*.equals(authentication);  
 }  
}

*package* com.ayushsingh.ta\_candidate.config.security.authProvider;  
  
  
*@RequiredArgsConstructor  
@Component  
public class* CandidateAuthProvider *implements* AuthenticationProvider {  
  
 *private final* CandidateDetailsService candidateDetailsService;  
 *private final* PasswordEncoder passwordEncoder;  
 *@Override  
 public* Authentication authenticate(Authentication authentication) *throws* AuthenticationException {  
 String username = String.valueOf(authentication.getPrincipal());  
 String password=String.valueOf(authentication.getCredentials());  
  
 UserDetails candidateDetails = candidateDetailsService.loadUserByUsername(username);  
 *if*(candidateDetails!=*null*){  
 *if*(passwordEncoder.matches(password,candidateDetails.getPassword())){  
  
 *return new* UsernamePasswordAuthenticationToken(username,password,candidateDetails.getAuthorities());  
  
 }  
 }  
 *throw new* BadCredentialsException("Wrong Credentials");  
 }  
  
 *@Override  
 public boolean* supports(Class<?> authentication) {  
 *return* UsernamePasswordAuthenticationToken.*class*.equals(authentication);  
 }  
}

1. AuthEntryPoint and CustomAccessDeniedHandler are used for exception handling.

## Security models

Instead of directly using the entity classes as implementations of UserDetails, we can create separate classes to handle the UserDetails method implementations and Granted Authorities (roles for users).



1. SecurityAdmin

*package* com.ayushsingh.ta\_candidate.model.securityModels.entity;  
  
*import* com.ayushsingh.ta\_candidate.model.entity.Admin;  
*import* lombok.AllArgsConstructor;  
*import* org.springframework.security.core.GrantedAuthority;  
*import* org.springframework.security.core.authority.SimpleGrantedAuthority;  
*import* org.springframework.security.core.userdetails.UserDetails;  
  
*import* java.util.Collection;  
*import* java.util.stream.Collectors;  
  
@AllArgsConstructor  
*public class* SecurityAdmin *implements* UserDetails {  
  
 *private final* Admin admin;  
  
 @Override  
 *public* Collection<? *extends* GrantedAuthority> getAuthorities() {  
 *return* admin.getRoles()  
 .stream()  
 .map(role -> *new* SimpleGrantedAuthority(role.getRoleName()))  
 .collect(Collectors.toList());  
 }  
  
 @Override  
 *public* String getPassword() {  
 *return this*.admin.getAdminPassword();  
 }  
  
 @Override  
 *public* String getUsername() {  
 *return this*.admin.getAdminEmail();  
 }  
  
 @Override  
 *public boolean* isAccountNonExpired() {  
 *return true*;  
 }  
  
 @Override  
 *public boolean* isAccountNonLocked() {  
 *return true*;  
 }  
  
 @Override  
 *public boolean* isCredentialsNonExpired() {  
 *return true*;  
 }  
  
 @Override  
 *public boolean* isEnabled() {  
 *return true*;  
 }  
}

1. SecurityCandidate

*package* com.ayushsingh.ta\_candidate.model.securityModels.entity;  
  
  
*import* com.ayushsingh.ta\_candidate.model.entity.Candidate;  
*import* com.ayushsingh.ta\_candidate.model.roles.CandidateRole;  
*import* lombok.AllArgsConstructor;  
*import* org.springframework.security.core.GrantedAuthority;  
*import* org.springframework.security.core.authority.SimpleGrantedAuthority;  
*import* org.springframework.security.core.userdetails.UserDetails;  
  
*import* java.util.Collection;  
*import* java.util.stream.Collectors;  
  
@AllArgsConstructor  
*public class* SecurityCandidate *implements* UserDetails {  
 *private final* Candidate candidate;  
  
 @Override  
 *public* Collection<? *extends* GrantedAuthority> getAuthorities() {  
 *return* candidate.getRoles()  
 .stream()  
 .map(role -> *new* SimpleGrantedAuthority(role.getRoleName()))  
 .collect(Collectors.toList());  
 }  
  
 @Override  
 *public* String getPassword() {  
 *return this*.candidate.getPassword();  
 }  
  
 @Override  
 *public* String getUsername() {  
 *return this*.candidate.getEmail();  
 }  
  
 @Override  
 *public boolean* isAccountNonExpired() {  
 *return true*;  
 }  
  
 @Override  
 *public boolean* isAccountNonLocked() {  
 *return true*;  
 }  
  
 @Override  
 *public boolean* isCredentialsNonExpired() {  
 *return true*;  
 }  
  
 @Override  
 *public boolean* isEnabled() {  
 *return true*;  
 }  
}

1. AdminAuthority

*package* com.ayushsingh.ta\_candidate.model.securityModels.authority;  
  
*import* com.ayushsingh.ta\_candidate.model.roles.AdminRole;  
*import* lombok.*AllArgsConstructor*;  
*import* org.springframework.security.core.GrantedAuthority;  
  
*@AllArgsConstructor  
public class* AdminAuthority *implements* GrantedAuthority {  
 *private final* AdminRole adminRole;  
  
 *@Override  
 public* String getAuthority() {  
 *return this*.adminRole.getRoleName();  
 }  
}

1. CandidateAuthority

*package* com.ayushsingh.ta\_candidate.model.securityModels.authority;  
  
*import* com.ayushsingh.ta\_candidate.model.roles.CandidateRole;  
*import* lombok.*AllArgsConstructor*;  
*import* org.springframework.security.core.GrantedAuthority;  
  
*@AllArgsConstructor  
public class* CandidateAuthority *implements* GrantedAuthority {  
  
 *private final* CandidateRole candidateRole;  
  
 *@Override  
 public* String getAuthority() {  
 *return this*.candidateRole.getRoleName();  
 }  
  
}

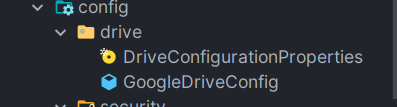
# Exception Handling

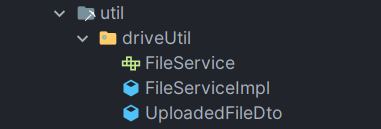


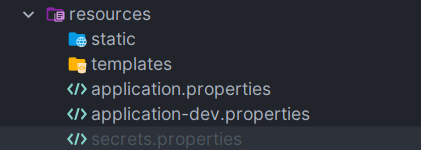
1. GlobalExceptionHandler is used to handle responses for custom and predefined exceptions.
2. ApiResponse: A generic class to provide response body.

# File Handling and Constants

1. In order to store the resume documents, we can use Google Drive API to upload the resumes to a Google Drive Folder.
2. For this purpose we need to add the configurations and secrets for Google Drive.







secrets.properties: This file holds the folder id for the storage folder.

1. AppConstants file hold the constants like response codes, public urls and jwt secret key. (In production, this key must be added to environment variable and not pushed to repository)



# APIs

For all the endpoints other than register and login, we will have to pass the Bearer token in the authorization header.

## Candidate

*package* com.ayushsingh.ta\_candidate.controller;  
  
  
*@RestController  
@RequestMapping*("/api/v1/candidate")  
*@RequiredArgsConstructor  
public class* CandidateController {  
  
 *private final* CandidateService candidateService;  
 *private final* ResumeService resumeService;  
  
 *@PostMapping*("/new")  
 *public* ResponseEntity<ApiResponse<String>> newCandidate(*@RequestBody* CandidateCreateDto candidateCreateDto) {  
 String token = candidateService.createNewCandidate(candidateCreateDto);  
 *return new* ResponseEntity<>(*new* ApiResponse<>(token), HttpStatus.CREATED);  
 }  
  
 *@PostMapping*("/login")  
 *public* ResponseEntity<ApiResponse<LoginResponseDto>> login() {  
 *if* (SecurityContextHolder.getContext().getAuthentication().isAuthenticated()) {  
 String username = (String) SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
 String accessToken = JwtUtil.generateToken(username, AppConstants.ENTITY\_TYPE\_CANDIDATE);  
 LoginResponseDto loginResponseDto = *new* LoginResponseDto();  
 loginResponseDto.setAccessToken(accessToken);  
 loginResponseDto.setUsername(username);  
 *return new* ResponseEntity<>(*new* ApiResponse<>(loginResponseDto), HttpStatus.OK);  
  
 }  
 *throw new* ApiException("User authentication failed!");  
 }  
  
 *@GetMapping*("/details")  
 *public* ResponseEntity<ApiResponse<CandidateDetailsProjection>> candidateDetails(*@RequestParam*(value = "candidateEmail") String candidateEmail) {  
 *return new* ResponseEntity<>(*new* ApiResponse<>(candidateService.getCandidateDetails(candidateEmail)), HttpStatus.OK);  
 }  
  
 *@PreAuthorize*("hasRole('ROLE\_CANDIDATE')")  
 *@PostMapping*("/resume/upload")  
 *public* ResponseEntity<ApiResponse<String>> uploadFile(*@RequestParam*("candidateToken") String candidateToken, *@RequestPart*("file") MultipartFile multipartFile) {  
 *return new* ResponseEntity<>(*new* ApiResponse<>(resumeService.uploadResume(candidateToken, multipartFile)), HttpStatus.CREATED);  
 }  
}

1. Register Candidate

**POST:** [**http://localhost:8085/api/v1/candidate/new**](http://localhost:8085/api/v1/candidate/new)

**Request Body:**

{

    "firstName": "Ayush",

    "lastName" : "Singh",

    "password": "123abc",

    "email": "ayushsingh20november@gmail.com"

}

**Response Body:**

{

    "code": 2000,

    "message": "Success",

    "data": "b8a03130-af06-40f5-aee6-de2d790768b4"

}

1. Candidate Login

**POST:** [**http://localhost:8085/api/v1/candidate/login**](http://localhost:8085/api/v1/candidate/login)

**Request Body:**

{

    "username": "ayushsingh20november@gmail.com",

    "password": "123abc"

}

**Response Body:**

{

    "code": 2000,

    "message": "Success",

    "data": {

        "accessToken": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJheXVzaHNpbmdoMjBub3ZlbWJlckBnbWFpbC5jb20iLCJleHAiOjE3MTE0NjcxNzMsIkVOVElUWV9UWVBFIjoiQ0FORElEQVRFIiwiaWF0IjoxNzExNDMxMTczfQ.R0Phsn29h\_Wm08b88u27O-BV8xh9zyTg5uZGzVcAp2I",

        "username": "ayushsingh20november@gmail.com"

    }

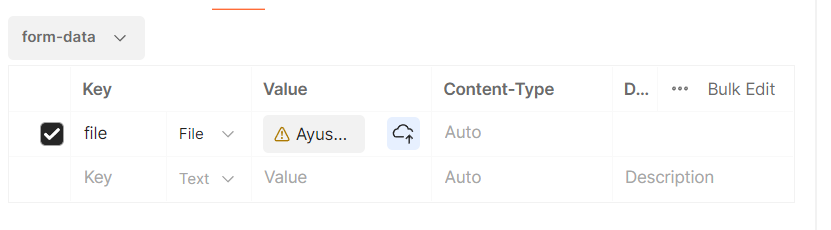
}

Note- The access token is the jwt token which must be used for all protected candidate endpoints. For now, this token will be sent as a Bearer Token in the authorization header, but for better protection and security, we can use **HttpOnly Cookie**  to store this token, along with a short expiry time for access token and use a refresh token to obtain a new token.

1. Upload resume

**POST:** [**http://localhost:8085/api/v1/candidate/resume/upload?candidateToken=b8a03130-af06-40f5-aee6-de2d790768b4**](http://localhost:8085/api/v1/candidate/resume/upload?candidateToken=b8a03130-af06-40f5-aee6-de2d790768b4)

**Request Body:**

****

**Response Body:**

{

    "code": 2000,

    "message": "Success",

    "data": "1TdLl6LJNVP517AQlRksJN8hXHm3dp\_K0"

}

1. Get candidate details

**GET:** [**http://localhost:8085/api/v1/candidate/details?candidateEmail=ayushsingh20november@gmail.com**](http://localhost:8085/api/v1/candidate/details?candidateEmail=ayushsingh20november@gmail.com)

**Response Body:**

{

    "code": 2000,

    "message": "Success",

    "data": {

        "token": "b8a03130-af06-40f5-aee6-de2d790768b4",

        "email": "ayushsingh20november@gmail.com",

        "firstName": "Ayush",

        "lastName": "Singh",

        "documentToken": "1TdLl6LJNVP517AQlRksJN8hXHm3dp\_K0",

        "resumeUrl": "https://drive.google.com/file/d/1TdLl6LJNVP517AQlRksJN8hXHm3dp\_K0"

    }

}

## Admin

1. Admin Login

**POST:** [**http://localhost:8085/api/v1/admin/login**](http://localhost:8085/api/v1/admin/login)

**Request Body:**

{

    "username":"ayush20april@gmail.com",

    "password": "123abc"

}

**Response Body:**

{

    "code": 2000,

    "message": "Success",

    "data": {

        "accessToken": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJheXVzaDIwYXByaWxAZ21haWwuY29tIiwiZXhwIjoxNzExNDY4NzA2LCJFTlRJVFlfVFlQRSI6IkFETUlOIiwiaWF0IjoxNzExNDMyNzA2fQ.F--bjeSxCd5lHy2LgAtrCYLuWvz8X1NUkosDX\_FDz38",

        "username": "ayush20april@gmail.com"

    }

}

## Interview

1. Schedule Interview

**POST:** [**http://localhost:8085/api/v1/interview/create**](http://localhost:8085/api/v1/interview/create)

**Request Body:**

{

    "interviewSubject" : "Java fresher developer at XYZ",

    "candidateToken": "b8a03130-af06-40f5-aee6-de2d790768b4",

    "meetLink": "meet.xyz.com/abc-123-abc",

    "interviewMode": "ONLINE",

    "dateTime": "2024-04-01T10:00:00",

    "timeZone": "Asia/Kolkata"

}

**Response Body:**

{

    "code": 2000,

    "message": "Success",

    "data": "eebfc80a-c145-4fc8-83d5-60c24ab547aa"

}

1. Interview List

**GET:** [**http://localhost:8085/api/v1/interview/all?token=b8a03130-af06-40f5-aee6-de2d790768b4**](http://localhost:8085/api/v1/interview/all?token=b8a03130-af06-40f5-aee6-de2d790768b4)

**Response Body:**

{

    "code": 2000,

    "message": "Success",

    "data": [

        {

            "interviewMode": "ONLINE",

            "interviewSubject": "Java fresher developer at XYZ",

            "interviewToken": "eebfc80a-c145-4fc8-83d5-60c24ab547aa",

            "interviewStatus": "SCHEDULED",

            "interviewTime": "2024-04-01T04:30:00Z"

        }

    ]

}

1. Change interview status

**PATCH:** [**http://localhost:8085/api/v1/interview/status**](http://localhost:8085/api/v1/interview/status)

**Request Body:**

{

    "interviewToken": "eebfc80a-c145-4fc8-83d5-60c24ab547aa",

    "interviewStatus": "COMPLETED"

}

**Response Body:**

{

    "code": 2000,

    "message": "Success",

    "data": "eebfc80a-c145-4fc8-83d5-60c24ab547aa"

}

1. Save interview result

**POST:** [**http://localhost:8085/api/v1/interview/feedback**](http://localhost:8085/api/v1/interview/feedback)

**Request Body:**

{

    "feedback": "Was able to answer most of the questions.",

    "decision": "Passed",

    "interviewToken": "eebfc80a-c145-4fc8-83d5-60c24ab547aa"

}

**Response Body:**

{

    "code": 2000,

    "message": "Success",

    "data": "eebfc80a-c145-4fc8-83d5-60c24ab547aa"

}

1. Get interview details

**GET:** [**http://localhost:8085/api/v1/interview/details?interviewToken=eebfc80a-c145-4fc8-83d5-60c24ab547aa&candidateToken=b8a03130-af06-40f5-aee6-de2d790768b4**](http://localhost:8085/api/v1/interview/details?interviewToken=eebfc80a-c145-4fc8-83d5-60c24ab547aa&candidateToken=b8a03130-af06-40f5-aee6-de2d790768b4)

**Response Body:**

{

    "code": 2000,

    "message": "Success",

    "data": {

        "interviewSubject": "Java fresher developer at XYZ",

        "meetLink": "meet.xyz.com/abc-123-abc",

        "meetTime": "2024-04-01T04:30:00Z",

        "interviewResult": {

            "interviewResultToken": "2a602413-cfc1-4ca1-8645-4ef331542a45",

            "feedback": "Was able to answer most of the questions.",

            "decision": "Passed"

        },

        "resumeDetails": {

            "documentToken": "1TdLl6LJNVP517AQlRksJN8hXHm3dp\_K0",

            "documentName": "Ayush Singh Resume.pdf",

            "format": "pdf",

            "documentUrl": "https://drive.google.com/file/d/1TdLl6LJNVP517AQlRksJN8hXHm3dp\_K0"

        },

        "interviewMode": "ONLINE",

        "interviewStatus": "COMPLETED",

        "interviewToken": "eebfc80a-c145-4fc8-83d5-60c24ab547aa"

    }

}

# Source Code Repository

The complete code for this project can be found here-

<https://github.com/singhayush20/Assignments/tree/main/Spring%20Boot/Wk05_06>