Spring Security and JWT Token Authentication

Session 3

Session Overview:
Training Approach and Interaction

Session 3

Presentation of Training Materials

Practical Training

Discussion

Session 3 Objective

Participants understand the importance of API security and recognize common vulnerabilities, implement Spring Security for managing access control and user authentication, generate and validate JWT tokens for secure API access, and secure REST APIs using authorization filters and effective exception handling techniques

MATERI PELATIHAN

- **Understanding the need for API security and common vulnerabilities.**
- **Implementing Spring Security for access control and user authentication.**
- **Generating and validating JWT tokens for secure API access.**
- **Securing REST APIs with authorization filters and exception handling.**

What is API Security?

- API Security adalah sebuah tindakan preventif untuk melindungi API dari serangan atau penyalahgunaan yang dilakukan individu maupun kelompok.
-

Common Vulnerabilities

- **Broken Object Level Authorization (BOLA)**: Terjadi saat API gagal melindungi objek, memungkinkan akses data tidak sah oleh penyerang.
- **Broken Authentication**: Risiko yang memungkinkan penyerang menyamar sebagai pengguna lain akibat kelemahan autentikasi API.
- **Excessive Data Exposure**: API mengekspos lebih banyak data daripada yang diperlukan, berisiko menyebabkan kebocoran data sensitif.
- **Lack of Resources and Rate Limiting**: Tanpa adanya batasan, API rentan terhadap serangan denial-of-service atau sistem yang kelebihan beban.
- **Injection Flaws**: Penyerang memanfaatkan input tidak tepercaya untuk menjalankan perintah atau mengakses data tidak sah melalui API.

Common Vulnerabilities

- **Improper Assets Management**: API yang sudah lama, salah konfigurasi, atau kurangnya dokumentasi dapat mengekspos risiko keamanan.
- **Insufficient Logging and Monitoring**: Tanpa pencatatan dan monitoring yang baik, pelanggaran keamanan lebih sulit dideteksi dan ditangani.
- **Security Misconfiguration**: Kesalahan konfigurasi API dapat membuka celah seperti penyimpanan terekspos, header HTTP salah.
- **Cross-Site Scripting (XSS)**: Terjadi saat data yang tidak aman dimasukkan ke konten server, memungkinkan skrip berbahaya berjalan di browser pengguna.
- **Insecure Deserialization**: Kerentanan yang memungkinkan eksekusi kode jarak jauh atau serangan lain melalui data yang tidak tepercaya.

Spring Security

- Spring Security menyediakan berbagai mekanisme untuk mengamankan REST API. Salah satunya adalah API key. API key adalah token yang diberikan klien saat memanggil API.
- REST API bersifat stateless. Oleh karena itu, tidak boleh menggunakan sesi atau cookie. Sehingga perlu menggunakan autentikasi dasar, API Key, JWT, atau token berbasis OAuth2

JWT Tokens

- JSON Web Token (JWT) adalah token berbentuk string panjang yang menggunakan format JSON (JavaScript Object Notation) dengan karakter yang sangat acak.
- Token ini memungkinkan pengiriman data yang dapat diverifikasi oleh dua pihak atau lebih.

Contoh Kode Generate Token :

```
1  private String doGenerateToken(Map<String, Object> claims, String subject) {
2          return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))
3                  .setExpiration(new Date(System.currentTimeMillis() + JWT_TOKEN_VALIDITY * 1000))
4                  .signWith(SignatureAlgorithm.HS256, secret).compact();
5      }
```

Authorization filters

- Authorization filters memungkinkan spring boot memeriksa permintaan akses sebelum diteruskan ke endpoint API.
- Spring Security menyediakan berbagai filter yang dapat disesuaikan untuk mengelola otorisasi salah satunya custom filter.
- Salah satunya yaitu OncePerRequestFilter, filter ini dapat memeriksa token, role, atau izin pengguna sebelum akses.

Contoh:

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .authorizeRequests(authorizeRequests -> authorizeRequests
                        .requestMatchers("/api/auth/**").permitAll()
                        .requestMatchers("/api/users/**").permitAll()
                        .requestMatchers("/api/task/**").authenticated()
                        .anyRequest().permitAll())
                .csrf(csrf -> csrf.disable())
                .sessionManagement(
                        sessionManagement -> sessionManagement.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

        http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
```

Exception handling

- Untuk menangani adanya kesalahan atau gagalnya suatu proses

Contoh:

```java
@PostMapping("/login")
    public LoginResponse authenticateUser(@RequestBody LoginRequest loginRequest) {
        try {
            Authentication authentication = authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(
                            loginRequest.getEmail(),
                            loginRequest.getPassword()));

            SecurityContextHolder.getContext().setAuthentication(authentication);
            UserDetails userDetails = customUserDetailsService.loadUserByUsername(loginRequest.getEmail());
            String token = jwtTokenUtil.generateToken(userDetails);

            return new LoginResponse(token);
        } catch (BadCredentialsException e) {
            throw new RuntimeException("Invalid email or password");
        }

    }
```

Implementation Spring Security

- Menambahkan dependency spring security dan JWT Token

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

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.2</version>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.2</version>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.2</version>
</dependency>
```

Database

Edit Table User

```sql
CREATE TABLE IF NOT EXISTS tb_users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    role ENUM('ADMIN', 'USER') NOT NULL
) ENGINE = InnoDB;
```

- Menambahkan field password dan role pada table user untuk keperluan autentikasi dan role pengguna

Model

User

```java
package com.example.demo.model;

import java.util.List;

import jakarta.persistence.CascadeType;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.OneToMany;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "tb_users")
public class User {

    @Id
    private int id;

    private String name;
    private String email;
    private String password;
    private String roles;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Task> tasks;
}

```

Menambahkan field password dan role pada model user sesuai dengan data database

Repository

Register User

```java
package com.example.demo.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import com.example.demo.model.User;

@Repository
public interface UserRepository extends JpaRepository<User, Integer> {

    @Query("SELECT u FROM User u WHERE u.name = :name")
    User findByName(@Param("name") String name);

    Optional<User> findByEmail(String email);
}

```

Menambahkan method findByEmail pada

Service

Register User

```java
@Transactional
    public String register(RegisterUserRequest request) {
        User user = new User();
        user.setName(request.getName());
        user.setEmail(request.getEmail());
        user.setPassword(passwordEncoder.encode(request.getPassword()));
        user.setRole("USER");

        userRepository.save(user);

        return "Create success";
    }
```

Menambahkan field password dan role pada service Register User, password menggunakan encoder Bcrypt

Service

Penambahan CustomUserDetailsService

```java
package com.example.demo.security;

import com.example.demo.model.User;
import com.example.demo.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.Collections;

@Service
public class CustomUserDetailsService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new UsernameNotFoundException("User not found with email: " + email));

        GrantedAuthority authority = new SimpleGrantedAuthority(user.getRole());
        return new org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(),
                Collections.singletonList(authority));
    }
}
```

Custom User Details Service Method digunakan sebagai method untuk load data user beserta otoritasnya.

Util

Penambahan JwtTokenUtil

```java
package com.example.demo.security;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Component
public class JwtTokenUtil {

    private static final long JWT_TOKEN_VALIDITY = 5 * 60 * 60;

    private String secret = "5367566B5970337336763979 2F423F4528482B4D6251655468576D5A71347437";

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

    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(secret).parseClaimsJws(token).getBody();
    }

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

    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.HS256, secret).compact();
    }

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

JWT Token Util digunakan untuk mengenerate Jwt Token, Claim token dan validate token.

Util

Penambahan Jwt Authentication Filter

```java
package com.example.demo.security;

import io.jsonwebtoken.ExpiredJwtException;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    @Autowired
    private CustomUserDetailsService customUserDetailsService;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        final String requestTokenHeader = request.getHeader("Authorization");

        String username = null;
        String jwtToken = null;

        if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {
            jwtToken = requestTokenHeader.substring(7);
            try {
                username = jwtTokenUtil.getUsernameFromToken(jwtToken);
            } catch (IllegalArgumentException e) {
                System.out.println("Unable to get JWT Token");
            } catch (ExpiredJwtException e) {
                System.out.println("JWT Token has expired");
            }
        } else {
            logger.warn("JWT Token does not begin with Bearer String");
        }

        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = this.customUserDetailsService.loadUserByUsername(username);

            if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {
                UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities());
                usernamePasswordAuthenticationToken
                        .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
            }
        }
        filterChain.doFilter(request, response);
    }

}
```

Jwt Authentication Filter digunakan untuk melakukan pengecekan autentikasi sbelum proses request HTTP klien dan memastikan Jwt token ada dan valid.

Util

Penambahan Security Config

```java
package com.example.demo.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private CustomUserDetailsService customUserDetailsService;

    @Autowired
    private JwtAuthenticationFilter jwtAuthenticationFilter;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .authorizeRequests(authorizeRequests -> authorizeRequests
                        .requestMatchers("/api/auth/**").permitAll()
                        .requestMatchers("/api/users/**").permitAll()
                        .requestMatchers("/api/task/**").authenticated()
                        .anyRequest().permitAll())
                .csrf(csrf -> csrf.disable())
                .sessionManagement(
                        sessionManagement -> sessionManagement.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

        http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

    @Bean
    public AuthenticationManager authenticationManager(HttpSecurity http) throws Exception {
        return http.getSharedObject(AuthenticationManagerBuilder.class)
                .userDetailsService(customUserDetailsService)
                .passwordEncoder(passwordEncoder())
                .and()
                .build();
    }

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

```

Secutiry Config adalah bagian dari spring security yang memungkinkan untuk mengatur filter sebelum proses request HTTP klien, dan validasi autentikasi dari setiap endpoint yang diperlukan.

Controller

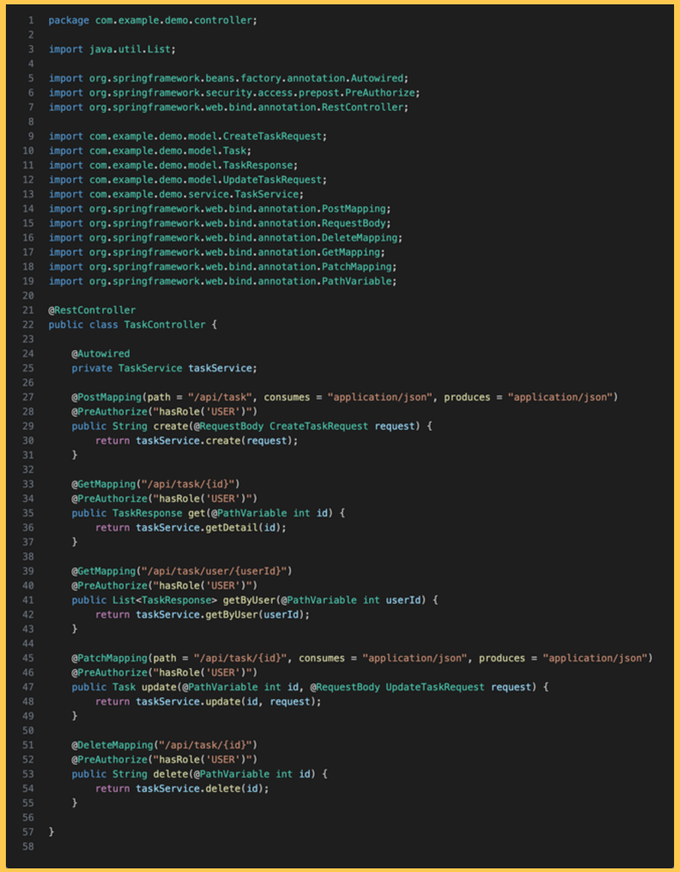
Penambahan Auth Controller

```java
package com.example.demo.controller;

import com.example.demo.model.LoginRequest;
import com.example.demo.model.LoginResponse;
import com.example.demo.model.User;
import com.example.demo.security.JwtTokenUtil;
import com.example.demo.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private UserDetailsService customUserDetailsService;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @PostMapping("/login")
    public LoginResponse authenticateUser(@RequestBody LoginRequest loginRequest) {
        try {
            Authentication authentication = authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(
                            loginRequest.getEmail(),
                            loginRequest.getPassword()));

            SecurityContextHolder.getContext().setAuthentication(authentication);
            UserDetails userDetails = customUserDetailsService.loadUserByUsername(loginRequest.getEmail());
            String token = jwtTokenUtil.generateToken(userDetails);

            return new LoginResponse(token);
        } catch (BadCredentialsException e) {
            throw new RuntimeException("Invalid email or password");
        }

    }

    @PostMapping("/logout")
    public String logoutUser() {
        SecurityContextHolder.clearContext();
        return "User logged out successfully";
    }
}
```

Auth Controller digunakan sebagai controller untuk menangani request POST untuk login dan logout user.

Controller

Edit Task Controller

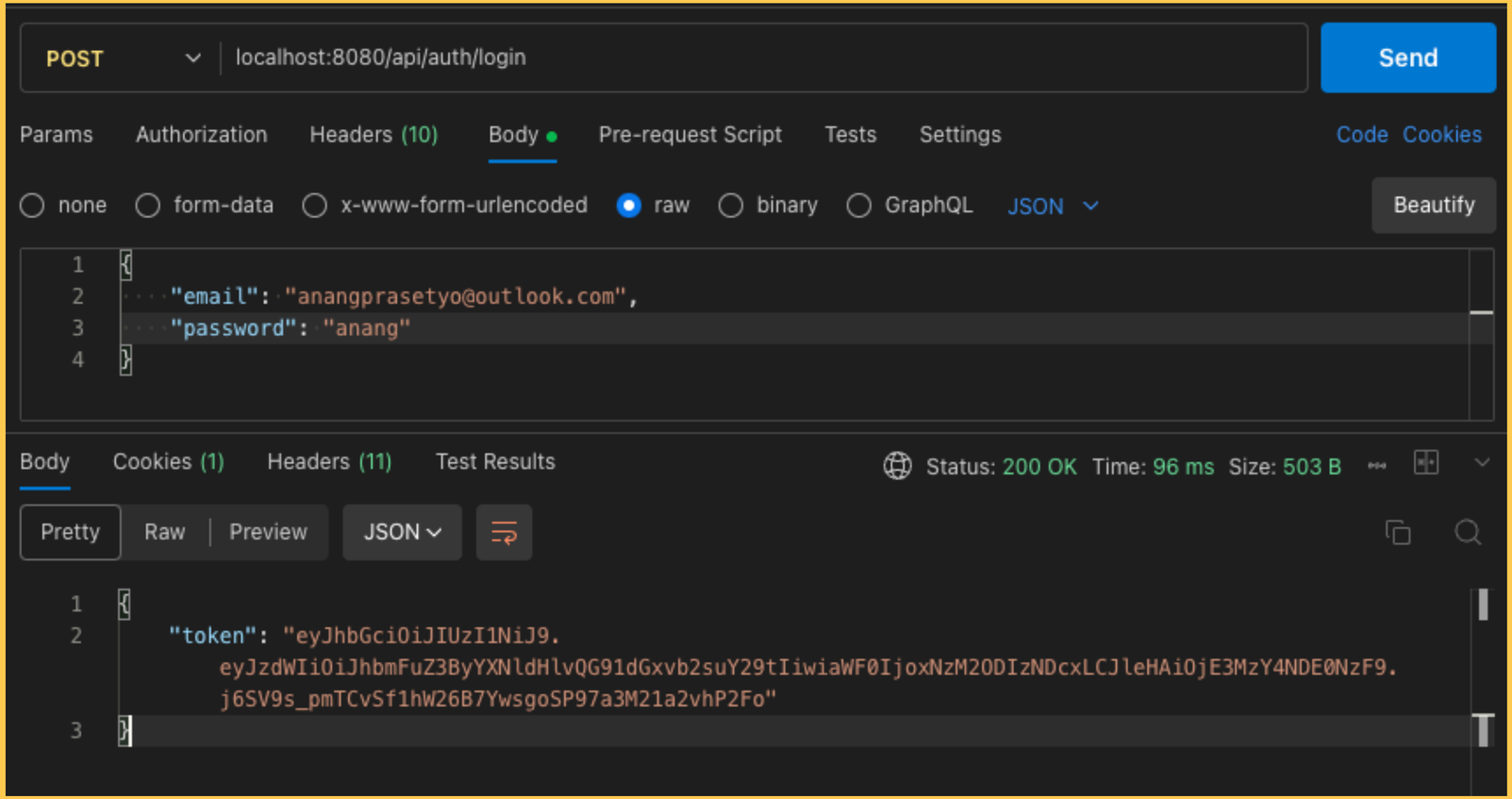


```java
package com.example.demo.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.model.CreateTaskRequest;
import com.example.demo.model.Task;
import com.example.demo.model.TaskResponse;
import com.example.demo.model.UpdateTaskRequest;
import com.example.demo.service.TaskService;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PathVariable;

@RestController
public class TaskController {

    @Autowired
    private TaskService taskService;

    @PostMapping(path = "/api/task", consumes = "application/json", produces = "application/json")
    @PreAuthorize("hasRole('USER')")
    public String create(@RequestBody CreateTaskRequest request) {
        return taskService.create(request);
    }

    @GetMapping("/api/task/{id}")
    @PreAuthorize("hasRole('USER')")
    public TaskResponse get(@PathVariable int id) {
        return taskService.getDetail(id);
    }

    @GetMapping("/api/task/user/{userId}")
    @PreAuthorize("hasRole('USER')")
    public List<TaskResponse> getByUser(@PathVariable int userId) {
        return taskService.getByUser(userId);
    }

    @PatchMapping(path = "/api/task/{id}", consumes = "application/json", produces = "application/json")
    @PreAuthorize("hasRole('USER')")
    public Task update(@PathVariable int id, @RequestBody UpdateTaskRequest request) {
        return taskService.update(id, request);
    }

    @DeleteMapping("/api/task/{id}")
    @PreAuthorize("hasRole('USER')")
    public String delete(@PathVariable int id) {
        return taskService.delete(id);
    }

}

```

Penambahan anotation @PreAuthorize("hasRole('USER')") untuk memastikan request yang dapat akses endpoint harus memiliki role “USER”

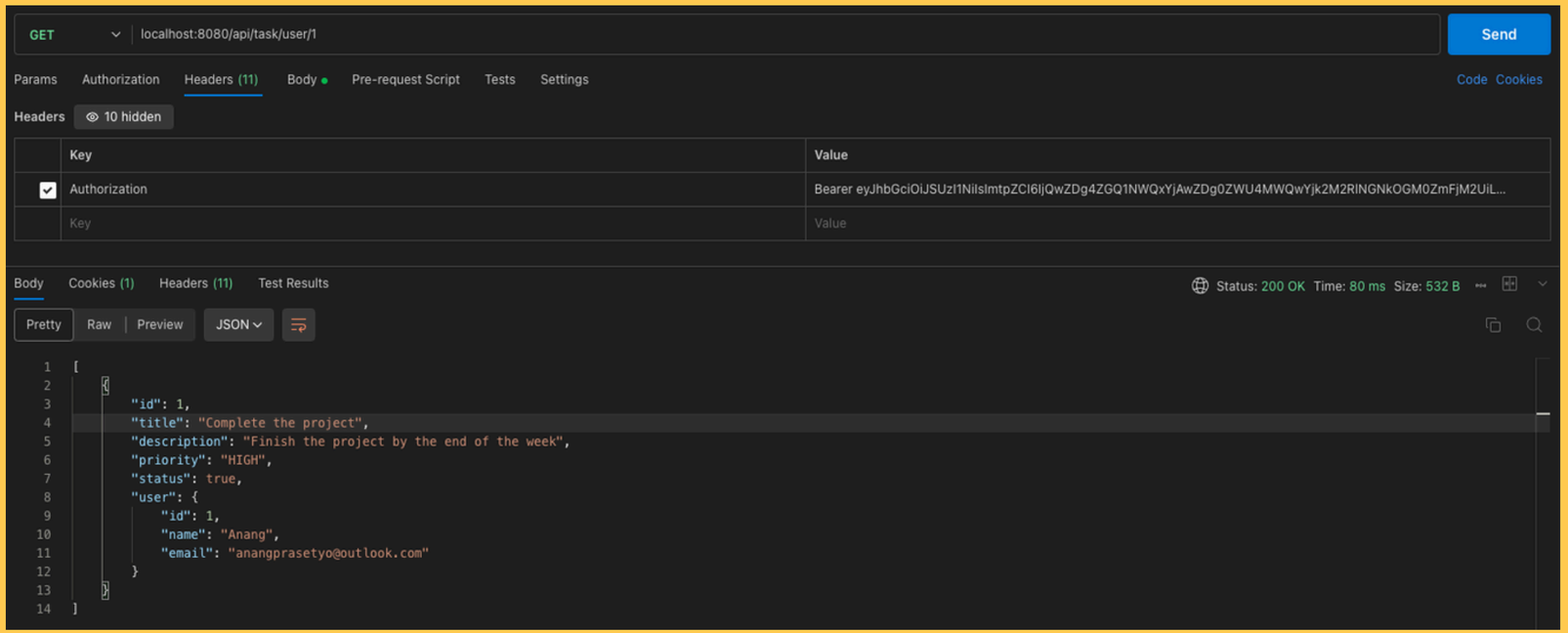
Testing

Endpoint Login User



Testing

Endpoint Task User



REFERENCES

- What is API Security? | API Security Best Practices | Akamai. (n.d.). Akamai. https://www.akamai.com/glossary/what-is-api-security
- Kumar, V. (2025, January 5). Top 10 API security vulnerabilities: Essential Guide for developers. Practical DevSecOps. https://www.practical-devsecops.com/top-api-security-vulnerabilities-guide/
- Sharifi, H. R., & Sharifi, H. R. (2024, May 11). Securing Spring Boot API with API key and secret | Baeldung. Baeldung. https://www.baeldung.com/spring-boot-api-key-secret

BINUS CENTER

THANK YOU

Thank you for following session 3 to the end.
See you at the next meeting