MODUL 6 Json Web Token (JWT)

6.1 Deskripsi Singkat

JWT (JSON Web Token) adalah format token yang digunakan untuk mengamankan pertukaran data antara dua pihak. JWT digunakan dalam konteks otentikasi dan otorisasi, seperti saat Anda ingin mengotentikasi pengguna pada aplikasi web atau API.

Berikut adalah beberapa komponen utama dari JWT:

1. Header:

Header JWT mengandung tipe token dan algoritma yang digunakan untuk mengenkripsi data. Contoh header JWT:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Dalam contoh ini, "alg" adalah algoritma HMAC SHA-256 yang digunakan untuk mengenkripsi token.

2. Payload (Isi)

Payload JWT berisi klaim-klaim atau informasi yang ingin Anda kirim. Klaim-klaim ini dapat dibagi menjadi tiga jenis:

- Klaim terdaftar: Klaim yang telah ditentukan sebelumnya oleh spesifikasi JWT, seperti "iss" (issuer), "exp" (expiration time), dan "sub" (subject).

- Klaim pribadi: Klaim yang dibuat oleh pengguna atau pihak yang menghasilkan token, seperti "user_id" atau "role".

- Klaim publik: Klaim yang dapat digunakan oleh semua pihak yang mengakses token, tetapi tidak dapat diandalkan untuk keamanan, seperti "name" atau "email".

Contoh payload JWT:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "exp": 1632537600
}
```

Dalam contoh ini, "sub" adalah subjek token, "name" adalah nama pengguna, dan "exp" adalah waktu kedaluwarsa token (dalam bentuk UNIX timestamp).

3. Signature (Tandatangan):

Untuk membuat token yang aman, JWT digabungkan dengan header dan payload, lalu ditandatangani dengan menggunakan kunci rahasia (shared secret) atau kunci privat. Tandatangan ini digunakan untuk memverifikasi integritas token ketika diterima kembali oleh pihak yang menerima token.

Proses kerja JWT adalah sebagai berikut:

1. Pihak yang menghasilkan token (biasanya server) membuat header dan payload.
2. Header dan payload digabungkan menjadi satu string JSON.
3. String JSON dienkripsi dengan menggunakan algoritma yang ditentukan dalam header dan kunci rahasia.
4. Tandatangan ditambahkan ke hasil enkripsi untuk menghasilkan token JWT.
5. Token JWT dikirim ke pihak lain (misalnya, klien).
6. Pihak lain dapat memverifikasi token dengan menguraikan header dan payload, mengenkripsi ulang dengan kunci yang sama, dan membandingkan tandatangan.

JWT memiliki beberapa keuntungan, termasuk kemampuan untuk mentransfer klaim yang aman, format yang ringan, dan tidak memerlukan penyimpanan server. Namun, penting untuk menjaga kunci rahasia dengan baik, karena siapa pun yang memiliki kunci tersebut dapat membaca dan membuat token JWT.

6.2 Tujuan Praktikum

Tujuan praktikum ini adalah membuat authentication web service menggunakan JWT yang diimplementasikan dengan Spring Framework.

Setelah praktikum ini kompetensi yang akan dicapai adalah mahasiswa mampu memahami konsep pengamanan web service menggunakan JWT dan memahami implementasinya.

6.3 Alokasi Waktu

Alokasi waktu pada praktikum 1 adalah sebagai berikut :

1. Tatap muka di kelas : 50 menit
2. Kerja mandiri (mengerjakan penugasan) : 120 menit

6.4 Material Praktikum

Praktikum ini memerlukan beberapa material sebagai berikut:

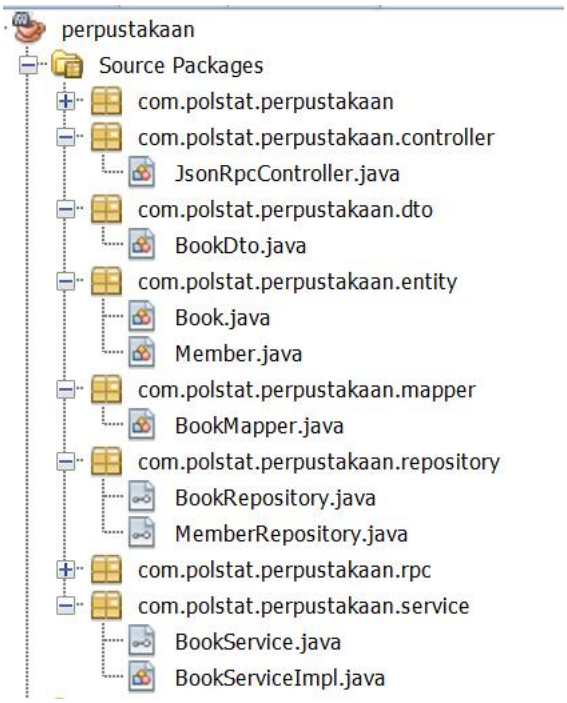
1. Java Development Kit (JDK) versi 17,
2. Java IDE: Netbeans, IntelliJ IDEA, atau Spring Tool Suite (STS),
3. Maven 3.5+,
4. MySQL,
5. Postman https://www.postman.com/downloads/.

6.5 Kegiatan Praktikum

Pada praktikum ini kita akan memodifikasi aplikasi perpustakaan yang telah dibuat pada praktikum 4 dengan menambahkan otentikasi dengan JWT. Ikuti langkah-langkahnya sebagai berikut.

Langkah 1: Menyiapkan Proyek

1) Gunakan proyek perpustakaan pada modul praktikum 4 dengan struktur folder awal sebagai berikut.



2) Tambahkan dependency Spring GraphQL pada file pom.xml.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.1</version>
</dependency>
```

Langkah 2: Membuat Kode Program

1) Membuat Kelas Entity User

Buatlah kelas entity User.java pada package com.polstat.perpustakaan.entity. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.entity;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
```

```
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Setter
@Getter
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@Table(name = "users")
public class User{
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   @Column(nullable = false)
   private String name;
   @Column(nullable = false)
   private String email;
   @Column(nullable = false)
   private String password;
}
```

2) Membuat Repository User

Buatlah kelas entity UserRepository.java pada package com.polstat.perpustakaan.repository. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.repository;

import com.polstat.perpustakaan.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long>{
   public User findByEmail(String email);
}
```

3) Membuat Kelas UserDto

Buatlah kelas entity UserDto.java pada package com.polstat.perpustakaan.dto. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.dto;

import java.util.Collection;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class UserDto  implements UserDetails{

    private Long id;
    private String name;
    private String email;
    private String password;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return null;
    }
    @Override
    public String getUsername() {
        return this.email;
    }
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

4) Membuat Kelas UserMapper

Buatlah kelas entity UserMapper.java pada package com.polstat.perpustakaan.mapper. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.mapper;

import com.polstat.perpustakaan.dto.UserDto;
import com.polstat.perpustakaan.entity.User;

public class UserMapper {
   public static User mapToUser(UserDto userDto){
      return User.builder()
            .id(userDto.getId())
            .name(userDto.getName())
            .email(userDto.getEmail())
            .password(userDto.getPassword())
            .build();
   }
   public static UserDto mapToUserDto(User user){
      return UserDto.builder()
            .id(user.getId())
            .name(user.getName())
            .password(user.getPassword())
            .email(user.getEmail())
            .build();
   }
}
```

5) Membuat Service User

Buatlah interface UserService.java pada package com.polstat.perpustakaan.service. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.service;

import com.polstat.perpustakaan.dto.UserDto;

public interface UserService{
   public UserDto createUser(UserDto user);
   public UserDto getUserByEmail(String email);
}
```

6) Membuat Implementasi Service User

Buatlah kelas UserServiceImpl.java pada package com.polstat.perpustakaan.service. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.service;

import com.polstat.perpustakaan.dto.UserDto;
import com.polstat.perpustakaan.entity.User;
import com.polstat.perpustakaan.mapper.UserMapper;
import com.polstat.perpustakaan.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

@Service
public class UserServiceImpl implements UserService{

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Override
    public UserDto createUser(UserDto userDto) {
        userDto.setPassword(passwordEncoder.encode(userDto.getPassword()));
        User user = userRepository.save(UserMapper.mapToUser(userDto));
        return UserMapper.mapToUserDto(user);
    }

    @Override
    public UserDto getUserByEmail(String email) {
        User user = userRepository.findByEmail(email);
        return UserMapper.mapToUserDto(user);
    }
}
```

7) Membuat Kelas CustomUserDetail

Buatlah kelas CustomUserDetailsService.java pada package com.polstat.perpustakaan.service. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.service;

import com.polstat.perpustakaan.entity.User;
```

```
import com.polstat.perpustakaan.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
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;

@Service
public class CustomUserDetailsService implements UserDetailsService {

   @Autowired
   private UserRepository userRepository;

   @Override
   public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
      User user = userRepository.findByEmail(email);
      if (user == null) {
         throw new UsernameNotFoundException("User not found with username: " +
email);
      }
      UserDetails userDetails =
            org.springframework.security.core.userdetails.User.builder()
                  .username(user.getEmail())
                  .password(user.getPassword())
                  .build();
      return userDetails;
   }
}
```

8) Membuat Kelas JwtUtil

Buatlah kelas JwtUtil.java pada package com.polstat.perpustakaan.auth. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.auth;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.ExpiredJwtException;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.MalformedJwtException;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.SignatureException;
import io.jsonwebtoken.UnsupportedJwtException;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
```

```
import java.io.Serializable;
import java.util.Date;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;

@Component
public class JwtUtil implements Serializable {

   @Value("${jwt.secret}")
   private String SECRET_KEY;

   @Value("${jwt.expiration}")
   private long EXPIRE_DURATION;

   public String generateAccessToken(Authentication authentication) {
      UserDetails userDetails = (UserDetails) authentication.getPrincipal();

      return Jwts.builder()
            .setSubject(userDetails.getUsername())
            .setIssuer("Polstat")
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + EXPIRE_DURATION))
            .signWith(SignatureAlgorithm.HS512, SECRET_KEY)
            .compact();
   }

   public boolean validateAccessToken(String token) {
      try {
         Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token);
         return true;
      } catch (ExpiredJwtException ex) {
         System.out.println("JWT expired" + ex.getMessage());
      } catch (IllegalArgumentException ex) {
         System.out.println("Token is null, empty or only whitespace" +
ex.getMessage());
      } catch (MalformedJwtException ex) {
         System.out.println("JWT is invalid" + ex);
      } catch (UnsupportedJwtException ex) {
         System.out.println("JWT is not supported" + ex);
      } catch (SignatureException ex) {
         System.out.println("Signature validation failed");
      }

      return false;
   }


   public String getSubject(String token) {
      return parseClaims(token).getSubject();
   }
```

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

Tambahkan pengaturan jwt.secret dan jwt.expiration pada file src/main/resources/application.properties sebagai berikut.

```
jwt.secret=mysecretkey
jwt.expiration=86400000
```

9) Membuat Kelas JwtFilter

Buatlah kelas JwtFilter.java pada package com.polstat.perpustakaan.auth. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.auth;

import com.polstat.perpustakaan.dto.UserDto;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;

import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTok
en;
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.util.ObjectUtils;
import org.springframework.web.filter.OncePerRequestFilter;

@Component
public class JwtFilter extends OncePerRequestFilter {

    @Autowired
    private JwtUtil jwtUtil;
```

```
    @Override
    protected void doFilterInternal(HttpServletRequest request,
            HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {

        if (!hasAuthorizationBearer(request)) {
            filterChain.doFilter(request, response);
            return;
        }
        String token = getAccessToken(request);
        if (!jwtUtil.validateAccessToken(token)) {
            filterChain.doFilter(request, response);
            return;
        }

        setAuthenticationContext(token, request);
        filterChain.doFilter(request, response);
    }

    private boolean hasAuthorizationBearer(HttpServletRequest request) {
        String header = request.getHeader("Authorization");
        if (ObjectUtils.isEmpty(header) || !header.startsWith("Bearer")) {
            return false;
        }
        return true;
    }

    private String getAccessToken(HttpServletRequest request) {
        String header = request.getHeader("Authorization");
        String token = header.split(" ")[1].trim();
        return token;
    }

    private void setAuthenticationContext(String token, HttpServletRequest request) {
        UserDetails userDetails = getUserDetails(token);

        UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(userDetails, null, null);

        authentication.setDetails(
                new WebAuthenticationDetailsSource().buildDetails(request));

        SecurityContextHolder.getContext().setAuthentication(authentication);
    }
    private UserDetails getUserDetails(String token) {
        String subject = jwtUtil.getSubject(token);
        return UserDto.builder().email(subject).build();
    }
}
```

10) Membuat Kelas SecurityConfig

Buatlah kelas SecurityConfig.java pada package com.polstat.perpustakaan.auth. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.auth;

import com.polstat.perpustakaan.service.CustomUserDetailsService;
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.Authenticat
ionManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecur
ity;
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.UsernamePasswordAuthenticatio
nFilter;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

   @Autowired private JwtFilter jwtTokenFilter;
   private final CustomUserDetailsService userDetailsService;

   public SecurityConfig(CustomUserDetailsService userDetailsService) {
      this.userDetailsService = userDetailsService;
   }


   @Bean
   public AuthenticationManager authenticationManager(HttpSecurity http,
PasswordEncoder passwordEncoder) throws Exception {
      AuthenticationManagerBuilder authenticationManagerBuilder =
http.getSharedObject(AuthenticationManagerBuilder.class);

authenticationManagerBuilder.userDetailsService(userDetailsService).passwordEnc
oder(passwordEncoder);
      return authenticationManagerBuilder.build();
   }
```

```
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{

        http.csrf().disable();

http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        http.authorizeRequests()
                .requestMatchers("/register", "/login").permitAll()
                .anyRequest().authenticated();
        http.addFilterBefore(jwtTokenFilter,
UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

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

11) Membuat Kelas AuthRequest

Buatlah kelas AuthRequest.java pada package com.polstat.perpustakaan.auth. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.auth;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.Max;
import jakarta.validation.constraints.NotNull;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter @Setter @NoArgsConstructor @AllArgsConstructor
public class AuthRequest {
    @NotNull @Email @Max(50)
    private String email;
    @NotNull @Max(16)
    private String password;
}
```

12) Membuat Kelas AuthResponse

Buatlah kelas AuthResponse.java pada package com.polstat.perpustakaan.auth. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.auth;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class AuthResponse {
   private String email;
   private String accessToken;
}
```

13) Membuat AuthenticationController

Buatlah kelas AuthController.java pada package com.polstat.perpustakaan.auth. Tulis kode program seperti berikut ini.

```
package com.polstat.perpustakaan.auth;

import com.polstat.perpustakaan.dto.UserDto;
import com.polstat.perpustakaan.service.UserService;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTok
en;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class AuthController {

   @Autowired
```

```
    AuthenticationManager authManager;
    @Autowired
    JwtUtil jwtUtil;
    @Autowired
    UserService userService;

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody @Valid AuthRequest request) {
        try {
            Authentication authentication = authManager.authenticate(
                    new UsernamePasswordAuthenticationToken(
                            request.getEmail(), request.getPassword())
            );

            String accessToken = jwtUtil.generateAccessToken(authentication);
            AuthResponse response = new AuthResponse(request.getEmail(),
accessToken);
            return ResponseEntity.ok().body(response);

        } catch (BadCredentialsException ex) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
        }
    }

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody UserDto request) {
        UserDto user = userService.createUser(request);
        return ResponseEntity.ok().body(user);
    }
}
```

Langkah 3: Menguji Proyek

1) Menjalan Proyek

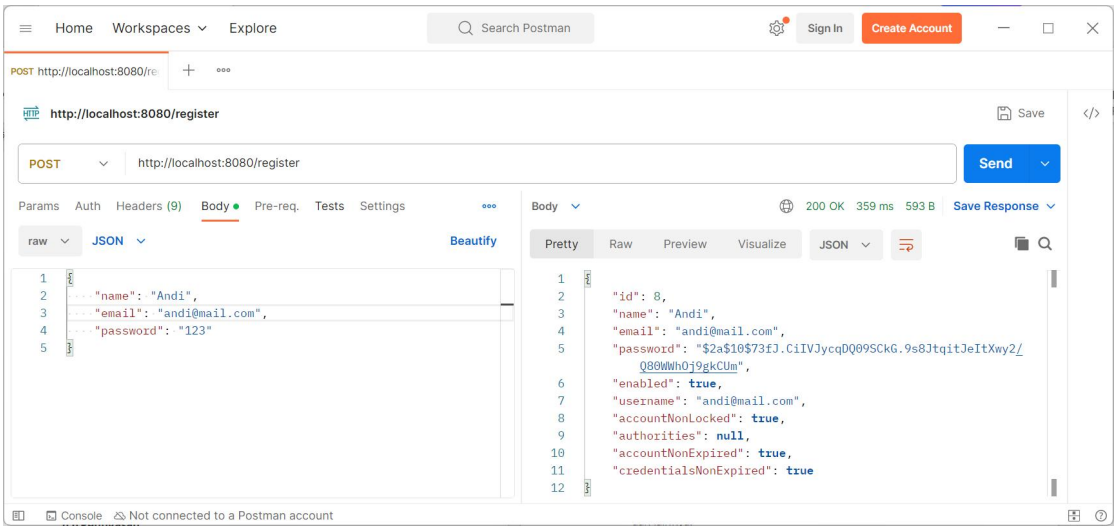
   Jalankan proyek dengan klik menu Run File pada file *PerpustakaanApplication.java*.

2) Menguji RESTful API dengan Postman

   Lakukan pengujian API melalui Postman untuk beberapa endpoint sebagai berikut:

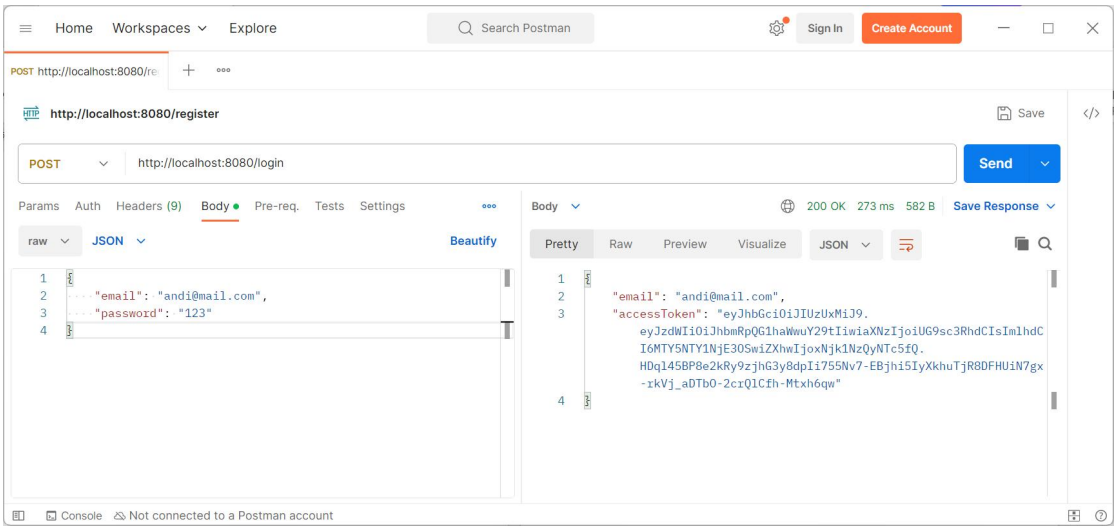
   a) Registrasi

      Registrasi pengguna menggunakan endpoint /registrasi dengan method POST dan data pengguna seperti gambar berikut.



b) Login

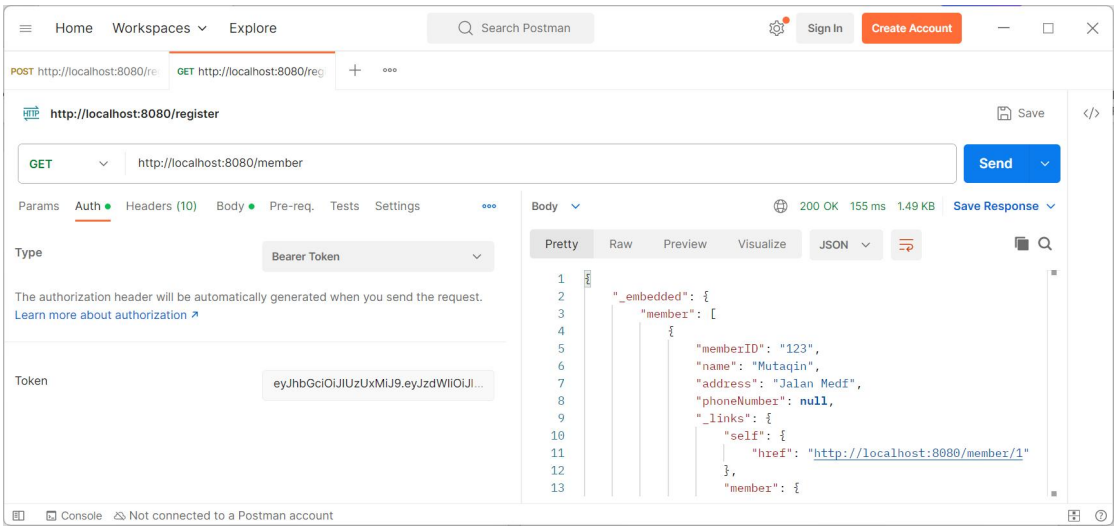
Setelah registrasi berhasil, login menggunakan email dan password yang telah terdaftar. Login pengguna menggunakan endpoint /login dengan method POST dan data email & password.



Setelah dieksekusi, hasilnya adalah email dan accessToken. Token tersebut yang akan digunakan untuk mengakses endpoint selain /register dan /login.

c) Mengakses Data Member

Data member dapat diakses pada endpoint /member. Untuk mengakses endpoint ini harus menyertakan accessToken yang telah di-*generate* pada endpoint /login sebelumnya. Berikut cara mengakses endpoint /member pada Postman.



Pada tab Auth, pilih type Bearer Token. Setelah itu, masukkan accessToken pada field Token.

6.6 Penugasan

Buatlah penjelasan lengkap mengenai beberapa hal pada kegiatan praktikum di atas.

1) Jelaskan fungsi/tugas atau kegunaan masing-masing kelas yang dibuat pada praktikum ini!
2) Jelaskan alur eksekusi endpoint /login dan proses otentikasi endpoint /member! Jelaskan kelas/objek/metode apa saja yang terlibat!