

## Membuat Proyek

Sebelum membuat proyek spring boot, kita harus melakukan persiapan kebutuhan untuk menggunakan framework spring boot terlebih dahulu. Untuk informasi lengkap mengenai persiapan ini dapat di lihat pada dokumentasi [official spring boot](#). Setelah persiapan telah siap, baru kita dapat memulai membuat proyek spring boot. Ada dua cara untuk membuat proyek awal spring boot, pertama menggunakan [Spring Initializr web tool](#) atau menggunakan ekstensi yang ada di Visual Studio.

Untuk membuat proyek menggunakan [Spring Initializr web tool](#) dapat di lihat pada dokumentasi [ini](#) dan untuk cara membuat proyek menggunakan ekstensi pada aplikasi visual studio, dapat di lihat pada dokumentasi [ini](#).

## PENJELASAN KODE PROGRAM API PONLINE

### Hubungkan proyek dengan basis data

1. Hapus file application.properties pada direktori resource (jika ada) dan buat file baru dengan nama application.yml. alasan menggunakan file yml adalah supaya lebih terlihat jelas hirarkinya.
2. Isikan kode berikut pada file application.yml

```
1. spring:
2.   # Konfigurasi Properti Database
3.   datasource:
4.     url: jdbc:mysql://localhost:3306/datbes_online?useSSL=false
5.     username: root
6.     password:
7.   # Konfigurasi Properti JPA (spring.jpa)
8.   jpa:
9.     show-sql: true
10.    hibernate:
11.      ddl-auto: update
12.      naming-strategy: org.hibernate.cfg.ImprovedNamingStrategy
13.    properties:
14.      hibernate:
15.        dialect: org.hibernate.dialect.MySQL5InnoDBDialect
16.
```

#### Penjelasn :

url → berisi nama nama driver host database (jdbc:mysql)/domain atau alamat IP host database (localhost) : port database (3306)/nama database (datbes\_online)?opsional (useSSL=false)

username → berisi username autentifikasi untuk mengakses database

password → berisi password autentifikasi untuk mengakses database

jpa → adalah standar akses database untuk bahasa pemrograman Java. Karena Spring Boot menggunakan bahasa pemrograman Java, kita dapat menggunakan JPA untuk mengakses database. JPA mempermudah kita untuk mengakses database di Java.

Show-sql → adalah konfigurasi untuk menampilkan perintah sql dalam terminal ketika mengakses/mengelola database.

Ddl-auto → akan Secara otomatis memvalidasi atau mengekspor skema DDL ke database saat SessionFactory dibuat. Ada beberapa pilihan untuk skema ini yaitu :

Validate	: memvalidasi skema, tidak membuat perubahan pada database.
Update	: memperbarui skema.
Create	: membuat skema, menghapus data sebelumnya.

- create-drop : menghapus skema saat SessionFactory ditutup secara eksplisit, biasanya saat aplikasi dihentikan.
- None : tidak melakukan apa pun dengan skema, tidak membuat perubahan pada database

## Membuat objek entitas pengguna

1. Buat file baru dan isikan dengan kode berikut

**Note : kelas ini juga biasa di sebut sebagai kelas model**

```
1. @Entity
2. @Table(name = "TAB_USER")
3. public class User implements Serializable{
4.
5.     // Field Entiti User
6.     @Id
7.     @GeneratedValue(strategy = GenerationType.IDENTITY)
8.     @Column(name = "cfvdcqbnjl", nullable = false)
9.     private Long id;
10.
11.     @Column(name = "ggxafwykfd" ,nullable = false)
12.     private String name;
13.
14.     @Email
15.     @Column(name = "tbfewjerbs", nullable = false)
16.     private String email;
17.
18.     @Column(name = "kwbrftpgtm")
19.     private String imageUrl;
20.
21.     @Column(name = "lfcmluvud", nullable = false)
22.     private Boolean emailVerified = false;
23.
24.     @Column(name = "xvfeeumirr", length = 64)
25.     private String tokEmailVerified;
26.
27.     @JsonIgnore
28.     @Column(name = "jtixhfcdmv")
29.     private String password;
30.
31.     @Column(name = "typyhbwgqm", length = 64)
32.     private String tokResetPassword;
33.
34.     @NotNull
35.     @Enumerated(EnumType.STRING)
36.     @Column(name = "ccuombbxii")
37.     private AuthProvider provider;
38.
39.     @Column(name = "oebwazzvdi")
40.     private String providerId;
41.
42.     @Column(name = "uwegewxuf", nullable = false)
43.     @Enumerated(EnumType.STRING)
44.     private UserRole role;
45.
46.     // setter getter
47. }
```

**Penjelasan :**

Anotasi `@entity` → untuk memberitahu sistem bahwa ini adalah kelas entitas

Anotasi `@Table` → (pilihan) di gunakan untuk menentukan properti tabel seperti name tabel, sehingga nanti nama tabel yang akan di buat di database menggunakan nama tersebut. Jika hal ini tidak ditentukan maka nama tabel yang akan di buat di database adalah nama dari kelasnya.

`Serializable` → adalah sebuah kelas antarmuka bawaan java. `Serialization` adalah suatu proses mengubah objek menjadi byte stream, agar bisa disimpan dalam file, memori ataupun berguna dalam proses transmisi jaringan.

Anotasi `@Id` → untuk memberitahu bahwa kolom/atribut adalah primary key dari entitas/tabel yang akan di buat.

Anotasi `@GeneratedValue` → digunakan untuk mengisi otomatis nilai dari atribut. Anotasi ini akan membuat *Auto Increment* pada struktur tabel.

Anotasi `@Column` → digunakan untuk mengatur properti dari kolom/atribut. Seperti nama, panjang, tidak boleh kosong, dsb. Nama pada anotasi akan digunakan untuk nama kolom pada tabel di database.

Anotasi `@Enumerated` → digunakan untuk memberitahu bahasa kolom ini menggunakan tipe enum, biasanya di gunakan untuk data yang berupa pilihan seperti tipe, kategori, level, dsb. *Contoh membuat kelas enum dapat dilihat pada poin selanjutnya (2. Membuat kelas enum).*

Setter Getter → di gunakan untuk mengambil atau mengatur data dari setiap data. Bisa di ketik manual atau menggunakan alat yang di sediakan teks editor dengan cara. E.g. visual studio : klik kanan → source action → generate setter getter. Maka akan otomatis di buat kode seperti ini.

```
public Long getId() {  
    return id;  
}  
  
public void setId(Long id) {  
    this.id = id;  
}  
  
public String getName() {  
    return name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
  
public String getEmail() {  
    return email;  
}  
  
public void setEmail(String email) {  
    this.email = email;  
}  
  
public String getImageUrl() {  
    return imageUrl;  
}  
  
public void setImageUrl(String imageUrl) {
```

```
        this.imageUrl = imageUrl;
    }

    public Boolean getEmailVerified() {
        return emailVerified;
    }

    public void setEmailVerified(Boolean emailVerified) {
        this.emailVerified = emailVerified;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public AuthProvider getProvider() {
        return provider;
    }

    public void setProvider(AuthProvider provider) {
        this.provider = provider;
    }

    public String getProviderId() {
        return providerId;
    }

    public void setProviderId(String providerId) {
        this.providerId = providerId;
    }

    public UserRole getRole() {
        return role;
    }

    public void setRole(UserRole role) {
        this.role = role;
    }

    public String getTokEmailVerified() {
        return tokEmailVerified;
    }

    public void setTokEmailVerified(String tokEmailVerified) {
        this.tokEmailVerified = tokEmailVerified;
    }

    public String getTokResetPassword() {
        return tokResetPassword;
    }

    public void setTokResetPassword(String tokResetPassword) {
        this.tokResetPassword = tokResetPassword;
    }
}
```

```
}
```

2. Membuat kelas enum.

Berikut adalah contoh dari kelas enum untuk menyimpan data provider autentifikasi.

```
public enum AuthProvider {  
    local,  
    google  
}
```

Berikut adalah contoh dari kelas enum untuk menyimpan data role/peran/level pengguna.

```
public enum UserRole {  
    ROLE_PQOWNEDRETVY, ROLE_USER, ROLE_EMPLOYEE, ROLE_OWNER  
}
```

#### Penjelasan:

Data pilihan dipisahkan dengan koma (,).

## Membuat objek entitas untuk komunitas

1. Buat file dan isikan kode berikut :

```
1. @Entity  
2. @Table(name = "TAB_KOMUNITAS")  
3. public class Komunitas implements Serializable{  
4.  
5.     @Id  
6.     @GeneratedValue(strategy = GenerationType.IDENTITY)  
7.     @Column(name = "zygmqsabko")  
8.     private Long id;  
9.  
10.    @Column(name = "dborbhqzvn", length = 50)  
11.    private String nama;  
12.  
13.    @Column(name = "wapkapoyju")  
14.    private String alamat;  
15.  
16.    // setter getter  
17. }
```

Penjelasan : bisa di pelajari dari penjelasan pada poin (membuat objek entitas [pengguna](#))

## Membuat objek entitas untuk anggota

1. Buat file baru dan isikan kode berikut

```
1. @Entity  
2. @Table(name = "TAB_ANGGOTA")  
3. public class Anggota implements Serializable{  
4.  
5.     @Id  
6.     @GeneratedValue(strategy = GenerationType.IDENTITY)  
7.     @Column(name = "qriijjxteb")  
8.     private Long id;  
9.  
10.    @ManyToOne  
11.    private Komunitas komunitas;  
12.  
13.    @OneToOne  
14.    private User user;
```

Penjelasan :

Anotasi `@ManyToOne` → digunakan untuk menghubungkan tabel 1(anggota) *many to one* ke tabel 2 (komunitas) dari sisi tabel 1 (anggota). Desain ini di dapat dari bisnis model: terdapat banyak anggota dalam satu komunitas.

Anotasi `@OneToOne` → digunakan untuk menghubungkan tabel 1(anggota) *one to one* ke tabel 2 (user) dari sisi tabel 1 (anggota). Desain ini di dapat dari bisnis model: satu anggota hanya punya satu akun pengguna.

Penjelasan lain dapat di lihat pada poin (membuat entitas [pengguna](#))

## Membuat objek entitas untuk kolam

1. Buat file baru dan isikan kode berikut

```
1. @Entity
2. @Table(name = "TAB_KOLAM")
3. public class Kolam implements Serializable{
4.
5.     @Id
6.     @GeneratedValue(strategy = GenerationType.IDENTITY)
7.     @Column(name = "bwvecxxrfj")
8.     private Long id;
9.
10.    @Column(name = "jvvagcwjau", length = 50)
11.    private String name;
12.
13.    @ManyToOne
14.    private Komunitas komunitas;
15.
16.    // setter getter
17. }
```

Penjelasan : dapat dilihat pada poin (membuat entitas untuk [pengguna](#) dan [anggota](#))

## Membuat objek entitas jadwal

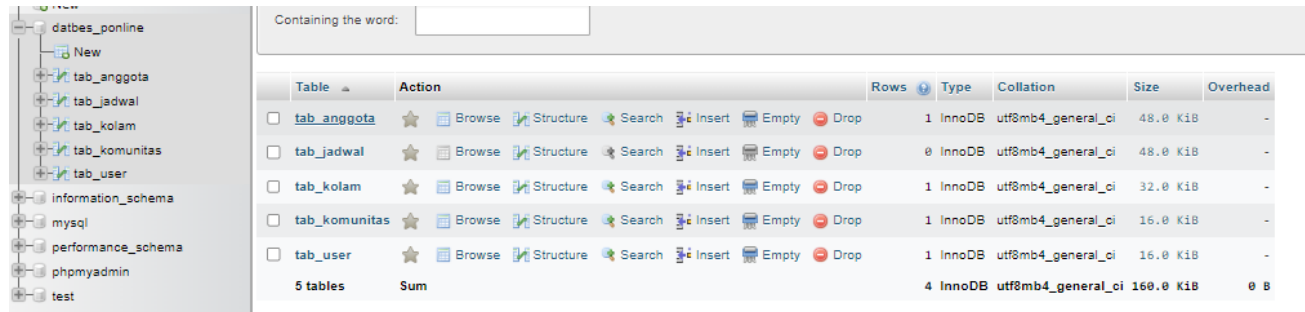
1. Buat file baru dan isikan kode berikut

```
1. @Entity
2. @Table(name = "TAB_JADWAL")
3. public class Jadwal implements Serializable{
4.
5.     @Id
6.     @GeneratedValue(strategy = GenerationType.IDENTITY)
7.     @Column(name = "khelyqlath")
8.     private Long id;
9.
10.    @Column(name = "jyiycynlnd")
11.    private Date dateToDo;
12.
13.    @ManyToOne
14.    private Kolam kolam;
15.
16.    @ManyToOne
17.    private Anggota anggota;
18.
19.    // setter getter
20. }
```

Penjelasan : dapat dilihat pada poin (membuat entitas untuk pengguna dan anggota)

Hasil dari program ini adalah sebagai berikut :

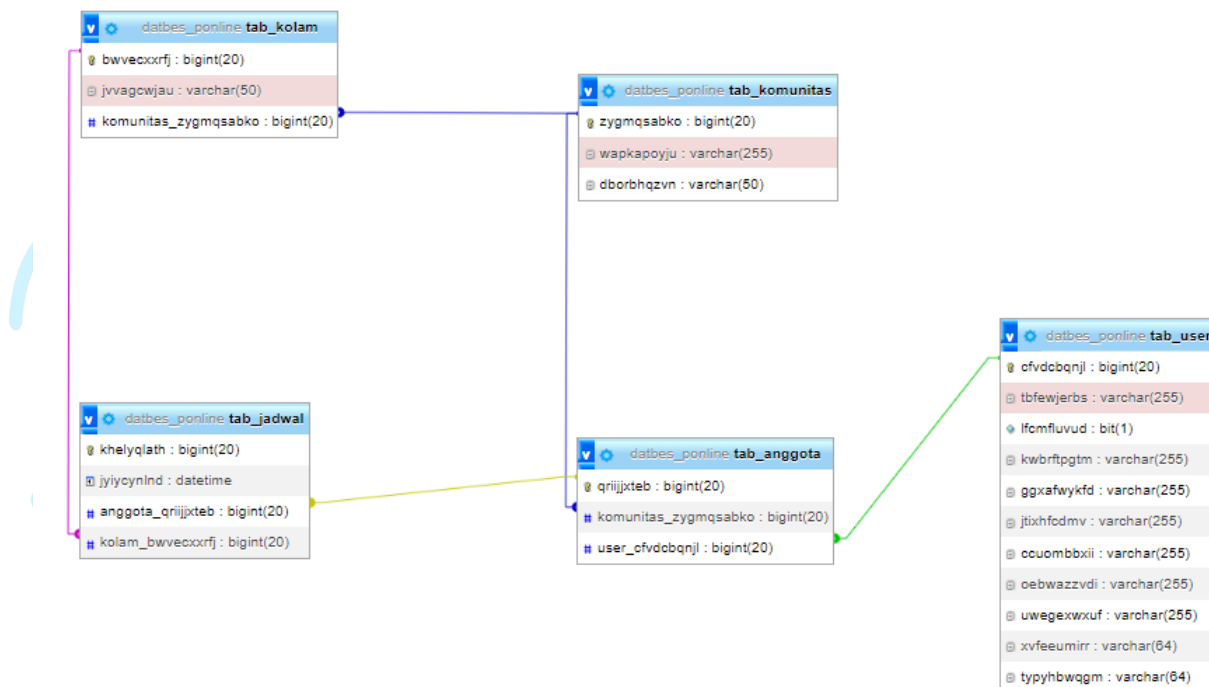
Tabel :



The screenshot shows the phpMyAdmin interface for the 'datbes\_ponline' database. On the left, a tree view lists the database and its tables: tab\_anggota, tab\_jadwal, tab\_kolam, tab\_komunitas, tab\_user, information\_schema, mysql, performance\_schema, phpmyadmin, and test. The main panel displays a table list for 'datbes\_ponline' with columns: Table, Action, Rows, Type, Collation, Size, and Overhead. The table list shows 5 tables: tab\_anggota, tab\_jadwal, tab\_kolam, tab\_komunitas, and tab\_user. A summary row at the bottom indicates 5 tables and a total size of 160.0 KiB.

Table	Action	Rows	Type	Collation	Size	Overhead
tab_anggota	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	48.0 KiB	-
tab_jadwal	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	48.0 KiB	-
tab_kolam	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	32.0 KiB	-
tab_komunitas	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
tab_user	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
5 tables	Sum	4	InnoDB	utf8mb4_general_ci	160.0 KiB	0 B

Desain Diagram :



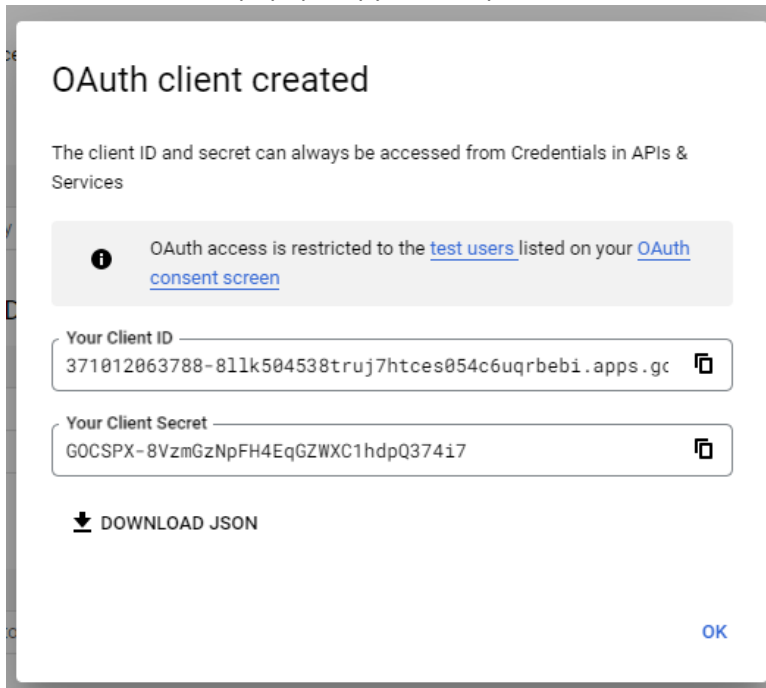
## Membuat autentikasi pengguna

Karena kita juga akan menggunakan autentikasi SSO, maka kita perlu menyiapkan atau mendaftarkan credential di sisi provider. Dalam program ini, provider yang ingin di gunakan adalah google.

Langkah untuk membuat credential google adalah,

1. Buka alamat <https://console.cloud.google.com/>
2. Pilih APIs & Service ++> Credentials
3. Klik tombol + CREATE CREDENTIALS
4. Pilih OAuth client ID
5. Pilih application type "Web Application"
6. Masukkan nama dan tambahkan URI

7. Isikan url "http://localhost:8080/oauth2/callback/google" pada URI yang baru di tambahkan
8. Klik create
9. Maka akan muncul popup, copy dan simpan Client ID dan Client Secret.



Langkah selanjutnya setelah membuat credentials di provider adalah,

1. Kita deklarasikan client id dan client secret pada proyek kita, tambahkan kode berikut pada file application.yml

```

1. # Konfigurasi Keamanan Program (spring.oauth2)
2. security:
3.   oauth2:
4.     client:
5.       registration:
6.         # Konfigurasi Client id & Secret Google Console
7.         google:
8.           clientId: 371012063788-811k504538truj7htces054c6uqrbebi.apps.googleusercontent.com
9.           clientSecret: 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
10.          redirectUriTemplate: "{baseUrl}/oauth2/callback/{registrationId}"
11.          scope:
12.            - email
13.            - profile
14. app:
15.   auth:
16.     tokenSecret: PQOWNELRITNYEUSIEOCPRQEWTEKRETYU # Base token untuk akses pengguna
17.     tokenExpirationMsec: 86400000 # Token Kadalwarsa dalam 86400000 ms / 24 jam
18.   oauth2:
19.     authorizedRedirectUris:
20.       - http://192.168.43.245:3000/oauth2/redirect
21.       - http://localhost:8080/documentation
22.       - myandroidapp://oauth2/redirect
23.       - myiosapp://oauth2/redirect
24.

```

**Penjelasan :**

Security: ➔ menandakan ini adalah konfigurasi untuk keamanan



Oauth2: → adalah jenis autentikasi yang akan di konfigurasi  
Client: → menandakan ini adalah konfigurasi sebagai client (pengguna layanan)  
google: → adalah provider autentikasi pengguna  
clientId: → client id credentials  
clientSecret → client secret credentials  
redirectUriTemplate → URIs credentials. (note: sebab mungkin nanti akan di adakan multi provider, sehingga uri di deklarasikan seperti tertampil supaya memungkinkan untuk menkonfigurasi dengan dinamis)  
scope → adalah data yang akan kita ambil dari provider  
app → berisi tentang properti aplikasi  
auth → berisi tentang properti kebutuhan autentikasi  
tokenSecret → berisi token rahasia yang nanti sebagai acuan untuk membuat token akses  
tokenExpirationMsec → batas waktu kadaluwarsa terhitung dari waktu token di buat, satuannya adalah mili detik  
oauth2 → berisi tentang kebutuhan properti autentikasi 2 (autentikasi dari provider)  
authorizedRedirectUris → adalah uri client yang diizinkan untuk request autentikasi

2. Selanjutnya kita ikat properti aplikasi ke kelas POJO. Kelas pojo atau java beans merupakan kumpulan class-class yang dapat dengan mudah digunakan kembali atau dikombinasikan dengan suatu aplikasi. Buat file baru dan tambahkan kode berikut.

```
@ConfigurationProperties(prefix = "app")
public class AppProperties {
    private final Auth auth = new Auth();
    private final OAuth2 oauth2 = new OAuth2();

    public static class Auth {
        private String tokenSecret;
        private long tokenExpirationMsec;

        public String getTokenSecret() {
            return tokenSecret;
        }

        public void setTokenSecret(String tokenSecret) {
            this.tokenSecret = tokenSecret;
        }

        public long getTokenExpirationMsec() {
            return tokenExpirationMsec;
        }

        public void setTokenExpirationMsec(long tokenExpirationMsec) {
            this.tokenExpirationMsec = tokenExpirationMsec;
        }
    }

    public static final class OAuth2 {
        private List<String> authorizedRedirectUris = new ArrayList<>();

        public List<String> getAuthorizedRedirectUris() {
            return authorizedRedirectUris;
        }

        public OAuth2 authorizedRedirectUris(List<String> authorizedRedirectUris) {
```

```
        this.authorizedRedirectUri = authorizedRedirectUri;
        return this;
    }

    public Auth getAuth() {
        return auth;
    }

    public OAuth2 getOAuth2() {
        return oauth2;
    }
}
```

#### Penjelasan :

@ConfigurationProperties → memberitahu sistem bahwa ini adalah kelas pojo. Lebih lengkap bisa di lihat [di sini](#).

Data dari application.yml yang di ambil adalah data data yang berada di dalam induk "app". Dimana di dalam properti "app" terdapat dua properti utama yaitu "auth" dan "auth2"

3. Kita harus mengaktifkan properti konfigurasi dengan menambahkan anotasi @EnableConfigurationProperties. Silakan buka kelas aplikasi utama SpringSocialApplication.java dan tambahkan anotasi seperti.

```
@SpringBootApplication
@EnableConfigurationProperties(AppProperties.class)
public class ApiPonlineApplication {

    public static void main(String[] args) {
        SpringApplication.run(ApiPonlineApplication.class, args);
    }

    @Bean
    public ModelMapper modelMapper() {
        return new ModelMapper();
    }
}
```

4. Mari aktifkan CORS agar klien frontend kita dapat mengakses API dari asal yang berbeda. Buat file baru dan tambahkan kode berikut.

```
@Configuration
public class WebMvcConfig implements WebMvcConfigurer {

    private final long MAX_AGE_SECS = 3600;

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins("*")
            .allowedMethods("GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS")
            .allowedHeaders("*")
            .allowCredentials(true)
            .maxAge(MAX_AGE_SECS);
    }
}
```

```
@Configuration
public class WebMvcConfig implements WebMvcConfigurer {

    private final long MAX_AGE_SECS = 3600;

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins("**")
            .allowedMethods("GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS")
            .allowedHeaders("**")
            .allowCredentials(true)
            .maxAge(MAX_AGE_SECS);
    }
}
```

**Penjelasan :**

@Configuration → memberitahu sistem bahwa ini adalah kelas konfigurasi.

WebMvcConfigurer → Kelas konfigurasi beranotasi @EnableWebMvc dapat mengimplementasikan antarmuka ini untuk dipanggil kembali dan diberi kesempatan untuk menyesuaikan konfigurasi default.

Tambahkan metode request yang di inginkan dalam kelas addCorsMappings.

5. Membuat kelas repositori untuk pengguna. Dalam kelas ini, terdapat query untuk mengakses atau mengelola database khususnya untuk entitas pengguna. Buat file baru dan masukan kode berikut.

```
// Tambahkan Anotasi @Repository untuk memberitahu bahwa ini adalah class repository
// Turunkan class JpaRepository dari lib 'JPA'
@Repository
public interface UserRepository extends JpaRepository<User, Long> {

    // (JPA)Query untuk cari user berdasarkan email return optional
    Optional<User> findByEmail(String email);

    // (JPA)Query untuk cari user apakah ada atau tidak di database berdasarkan email
    Boolean existsByEmail(String email);

    // (JPA)Query untuk cari user berdasarkan tokenResetPassword
    List<User> findByTokResetPassword(String tokResetPassword);

    // (JPA)Query untuk cari user berdasarkan tokenEmailVerified
    List<User> findByTokEmailVerified(String tokEmailVerified);

    // (MANUAL)Query untuk cari user berdasarkan email return 1 user
    @Query("SELECT u FROM User u WHERE u.email = :email")
    public User findOneByEmail(@PathParam("email") String email);

    // (MANUAL)Query untuk cari user berdasarkan email return 1 user
    @Query("SELECT u FROM User u WHERE u.id = :id")
    public User findOneById(@PathParam("id") Long id);
}
```

**Penjelasan :**

@Repository → anotasi untuk memberitahu bahwa ini adalah kelas repositori

JpaRepository → adalah kelas repositori bawaan JPA, dalam kelas ini sudah terdapat fungsi fungsi untuk DML.

@Query → di gunakan untuk membuat fungsi kueri kustom sesuai dengan kebutuhan.

6. Kelas SecurityConfig berikut adalah inti dari implementasi keamanan kami. Ini berisi konfigurasi untuk login provider OAuth2 serta login berbasis email dan kata sandi. Buat file baru dan isikan kode berikut.

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(
    securedEnabled = true,
    jsr250Enabled = true,
    prePostEnabled = true
)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private CustomUserDetailsService customUserDetailsService;

    @Autowired
    private CustomOAuth2UserService customOAuth2UserService;

    @Autowired
    private OAuth2AuthenticationSuccessHandler oAuth2AuthenticationSuccessHandler;

    @Autowired
    private OAuth2AuthenticationFailureHandler oAuth2AuthenticationFailureHandler;

    // @Autowired
    // private HttpCookieOAuth2AuthorizationRequestRepository
    httpCookieOAuth2AuthorizationRequestRepository;

    @Bean
    public TokenAuthenticationFilter tokenAuthenticationFilter() {
        return new TokenAuthenticationFilter();
    }

    @Bean
    public HttpCookieOAuth2AuthorizationRequestRepository cookieAuthorizationRequestRepository() {
        return new HttpCookieOAuth2AuthorizationRequestRepository();
    }

    @Override
    public void configure(AuthenticationManagerBuilder authenticationManagerBuilder) throws
    Exception {
        authenticationManagerBuilder
            .userDetailsService(customUserDetailsService)
            .passwordEncoder(passwordEncoder());
    }

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

    @Bean(Beans.AUTHENTICATION_MANAGER)
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}
```

```
}

@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .cors()
        .and()
        .sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and()
        .csrf()
        .disable()
        .formLogin()
        .disable()
        .httpBasic()
        .disable()
        .exceptionHandling()
        .authenticationEntryPoint(new RestAuthenticationEntryPoint())
        .and()
        .authorizeRequests()
        .antMatchers("/",
            "/error",
            "/favicon.ico",
            "**/*.png",
            "**/*.gif",
            "**/*.svg",
            "**/*.jpg",
            "**/*.html",
            "**/*.css",
            "**/*.js")
        .permitAll()
        .antMatchers("/auth/**", "/oauth2/**", "/documentation/**")
        .permitAll()
        // .antMatchers("/anggota/**")
        // .hasAnyAuthority("ROLE_OWNER", "ROLE_PQOWNEDRETVY")
        .anyRequest()
        .authenticated()
        .and()
        .oauth2Login()
        .authorizationEndpoint()
        .baseUri("/oauth2/authorize")
        .authorizationRequestRepository(cookieAuthorizationRequestRepository())
        .and()
        .redirectionEndpoint()
        .baseUri("/oauth2/callback/*")
        .and()
        .userInfoEndpoint()
        .userService(customOAuth2UserService)
        .and()
        .successHandler(oAuth2AuthenticationSuccessHandler)
        .failureHandler(oAuth2AuthenticationFailureHandler);

    // Add our custom Token based authentication filter
    http.addFilterBefore(tokenAuthenticationFilter(),
UsernamePasswordAuthenticationFilter.class);
}
}
```

**Penjelasan :**

Kelas di atas pada dasarnya menyatukan berbagai komponen untuk membuat kebijakan keamanan di seluruh aplikasi.

**Alur Login OAuth2 (login dengan provider)**

- Alur login OAuth2 akan dimulai oleh klien frontend dengan mengirimkan pengguna ke titik akhir `http://hostFrontend/oauth2/authorize/{provider}?redirect_uri=<redirect_uri_after_login>`.
- Parameter jalur penyedia adalah salah satu dari google. `redirect_uri` adalah URI tempat pengguna akan dialihkan setelah autentikasi dengan penyedia OAuth2 berhasil. Ini berbeda dari `redirectUri` OAuth2.
- Saat menerima permintaan otorisasi, klien OAuth2 Spring Security akan mengarahkan pengguna ke `AuthorizationUrl` dari penyedia yang disediakan.
- Semua status yang terkait dengan permintaan otorisasi disimpan menggunakan `otorisasiRequestRepository` yang ditentukan di `SecurityConfig`.
- Pengguna sekarang mengizinkan/menolak izin ke aplikasi Anda di halaman penyedia. Jika pengguna mengizinkan izin ke aplikasi, penyedia akan mengarahkan pengguna ke url callback `http://localhost:8080/oauth2/callback/{provider}` dengan kode otorisasi. Jika pengguna menolak izin, dia akan dialihkan ke `callbackUrl` yang sama tetapi dengan kesalahan.
- Jika panggilan balik OAuth2 menghasilkan kesalahan, keamanan Spring akan memanggil `oAuth2AuthenticationFailureHandler` yang ditentukan dalam `SecurityConfig` di atas.
- Jika callback OAuth2 berhasil dan berisi kode otorisasi, Spring Security akan menukar `otorisasi_code` dengan `access_token` dan memanggil `customOAuth2UserService` yang ditentukan dalam `SecurityConfig` di atas.
- `customOAuth2UserService` mengambil detail pengguna yang diautentikasi dan membuat entri baru di database atau memperbarui entri yang ada dengan email yang sama.
- Terakhir, `oAuth2AuthenticationSuccessHandler` dipanggil. Itu membuat token otentikasi JWT untuk pengguna dan mengirim pengguna ke `redirect_uri` bersama dengan token JWT dalam string kueri.

**7. Membuat kustom untuk autentikasi OAuth2****7.1. buat file dengan nama "HttpCookieOAuth2AuthorizationRequestRepository"**

Protokol OAuth2 merekomendasikan penggunaan parameter status untuk mencegah serangan CSRF. Selama autentikasi, aplikasi mengirimkan parameter ini dalam permintaan otorisasi, dan penyedia OAuth2 mengembalikan parameter ini tanpa perubahan dalam panggilan balik OAuth2.

Aplikasi membandingkan nilai parameter status yang dikembalikan dari penyedia OAuth2 dengan nilai yang awalnya dikirim. Jika mereka tidak cocok maka menolak permintaan otentikasi.

Untuk mencapai alur ini, aplikasi perlu menyimpan parameter status di suatu tempat agar nanti dapat membandingkannya dengan status yang dikembalikan dari penyedia OAuth2.

Kami akan menyimpan status serta `redirect_uri` dalam cookie yang berumur pendek. Kelas berikut menyediakan fungsionalitas untuk menyimpan permintaan otorisasi dalam cookie dan mengambilnya.

**Source Code :**

```
@Component
public class HttpCookieOAuth2AuthorizationRequestRepository implements
AuthorizationRequestRepository<OAuth2AuthorizationRequest> {
    public static final String OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME = "oauth2_auth_request";
```

```

public static final String REDIRECT_URI_PARAM_COOKIE_NAME = "redirect_uri";
private static final int cookieExpireSeconds = 180;

@Override
public OAuth2AuthorizationRequest loadAuthorizationRequest(HttpServletRequest request) {
    return CookieUtils.getCookie(request, OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME)
        .map(cookie -> CookieUtils.deserialize(cookie, OAuth2AuthorizationRequest.class))
        .orElse(null);
}

@Override
public void saveAuthorizationRequest(OAuth2AuthorizationRequest authorizationRequest,
    HttpServletRequest request, HttpServletResponse response) {
    if (authorizationRequest == null) {
        CookieUtils.deleteCookie(request, response,
            OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME);
        CookieUtils.deleteCookie(request, response, REDIRECT_URI_PARAM_COOKIE_NAME);
        return;
    }

    CookieUtils.addCookie(response, OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME,
        CookieUtils.serialize(authorizationRequest), cookieExpireSeconds);
    String redirectUriAfterLogin = request.getParameter(REDIRECT_URI_PARAM_COOKIE_NAME);
    if (StringUtils.isNotBlank(redirectUriAfterLogin)) {
        CookieUtils.addCookie(response, REDIRECT_URI_PARAM_COOKIE_NAME,
            redirectUriAfterLogin, cookieExpireSeconds);
    }
}

@Override
public OAuth2AuthorizationRequest removeAuthorizationRequest(HttpServletRequest request) {
    return this.loadAuthorizationRequest(request);
}

public void removeAuthorizationRequestCookies(HttpServletRequest request, HttpServletResponse
response) {
    CookieUtils.deleteCookie(request, response, OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME);
    CookieUtils.deleteCookie(request, response, REDIRECT_URI_PARAM_COOKIE_NAME);
}
}

```

## 7.2. buat file baru dengan nama "CustomOAuth2UserService"

CustomOAuth2UserService memperluas DefaultOAuth2UserService dari Spring Security dan mengimplementasikan metode loadUser() nya. Metode ini dipanggil setelah token akses diperoleh dari penyedia OAuth2.

Dalam metode ini, pertama-tama kita mengambil detail pengguna dari penyedia OAuth2. Jika pengguna dengan email yang sama sudah ada di database kami, maka kami memperbarui detailnya, jika tidak, kami mendaftarkan pengguna baru.

### Source Code:

```

@Service
public class CustomOAuth2UserService extends DefaultOAuth2UserService {

    // injeksi class user repository

```

```
@Autowired
private UserRepository userRepository;

// tulis ulang (override) class loadUser dari lib 'DefaultOAuth2UserService'
@Override
public OAuth2User loadUser(OAuth2UserRequest oAuth2UserRequest) throws
OAuth2AuthenticationException {
    OAuth2User oAuth2User = super.loadUser(oAuth2UserRequest);

    try {
        return processOAuth2User(oAuth2UserRequest, oAuth2User);
    } catch (AuthenticationException ex) {
        throw ex;
    } catch (Exception ex) {
        throw new InternalAuthenticationServiceException(ex.getMessage(), ex.getCause());
    }
}

private OAuth2User processOAuth2User(OAuth2UserRequest oAuth2UserRequest, OAuth2User
oAuth2User) {
    OAuth2UserInfo oAuth2UserInfo =
OAuth2UserInfoFactory.getOAuth2UserInfo(oAuth2UserRequest.getClientRegistration().getRegistrationId(), oAuth2User.getAttributes());
    if(StringUtils.isEmpty(oAuth2UserInfo.getEmail())) {
        throw new OAuth2AuthenticationProcessingException("Email not found from OAuth2
provider");
    }

    Optional<User> userOptional = userRepository.findByEmail(oAuth2UserInfo.getEmail());
    User user;
    if(userOptional.isPresent()) {
        user = userOptional.get();
        if(!user.getProvider().equals(AuthProvider.valueOf(oAuth2UserRequest.getClientRegistra
tion().getRegistrationId()))) {
            throw new OAuth2AuthenticationProcessingException("Looks like you're signed up
with " +
                user.getProvider() + " account. Please use your " + user.getProvider() +
                " account to login.");
        }
        user = updateExistingUser(user, oAuth2UserInfo);
    } else {
        user = registerNewUser(oAuth2UserRequest, oAuth2UserInfo);
    }

    return UserPrincipal.create(user, oAuth2User.getAttributes());
}

private User registerNewUser(OAuth2UserRequest oAuth2UserRequest, OAuth2UserInfo
oAuth2UserInfo) {
    User user = new User();

    user.setProvider(AuthProvider.valueOf(oAuth2UserRequest.getClientRegistration().getRegistr
ationId()));
    user.setEmailVerified(true);
    user.setRole(UserRole.ROLE_USER);
    user.setProviderId(oAuth2UserInfo.getId());
    user.setName(oAuth2UserInfo.getName());
}
```



```
        user.setEmail(oAuth2UserInfo.getEmail());
        user.setImageUrl(oAuth2UserInfo.getImageUrl());
        return userRepository.save(user);
    }

    private User updateExistingUser(User existingUser, OAuth2UserInfo oAuth2UserInfo) {
        existingUser.setName(oAuth2UserInfo.getName());
        existingUser.setImageUrl(oAuth2UserInfo.getImageUrl());
        return userRepository.save(existingUser);
    }
}
```

### 7.3. Buat file baru dengan nama "OAuth2UserInfo mapping".

Setiap penyedia OAuth2 mengembalikan respons JSON yang berbeda saat kami mengambil detail pengguna yang diautentikasi. Keamanan pegas mem-parsing respons dalam bentuk peta umum pasangan kunci-nilai.

Kelas berikut digunakan untuk mendapatkan detail pengguna yang diperlukan dari peta umum pasangan nilai kunci.

#### Source Code :

```
public abstract class OAuth2UserInfo {
    protected Map<String, Object> attributes;

    public OAuth2UserInfo(Map<String, Object> attributes) {
        this.attributes = attributes;
    }

    public Map<String, Object> getAttributes() {
        return attributes;
    }

    public abstract String getId();

    public abstract String getName();

    public abstract String getEmail();

    public abstract String getImageUrl();
}
```

### 7.4. Buat file baru dan beri nama "GoogleOAuth2UserInfo".

Karena mungkin nanti provider yang digunakan lebih dari satu. Maka setiap provider kita samakan objeknya ke bentuk OAuth2UserInfo.

#### Source Code :

```
public class GoogleOAuth2UserInfo extends OAuth2UserInfo {

    public GoogleOAuth2UserInfo(Map<String, Object> attributes) {
        super(attributes);
    }

    @Override
```

```
public String getId() {  
    return (String) attributes.get("sub");  
}  
  
@Override  
public String getName() {  
    return (String) attributes.get("name");  
}  
  
@Override  
public String getEmail() {  
    return (String) attributes.get("email");  
}  
  
@Override  
public String getImageUrl() {  
    return (String) attributes.get("picture");  
}  
}
```

Pada dasarnya kelas di atas adalah kelas OAuth2UserInfo, namun karena response dari setiap provider berbeda-beda. Sehingga setiap response dari provider di pisah dan dipetakan ke kelas objek OAuth2UserInfo.

7.5. Sekarang kita buat perilaku apa yang akan di lakukan jika autentikasi dengan provider berhasil. Buat file baru dan beri nama "OAuth2AuthenticationSuccessHandler".

Saat autentikasi berhasil, keamanan Spring memanggil metode onAuthenticationSuccess() dari OAuth2AuthenticationSuccessHandler yang dikonfigurasi di SecurityConfig.

Dalam metode ini, kami melakukan beberapa validasi, membuat token otentikasi JWT, dan mengarahkan ulang pengguna ke redirect\_uri yang ditentukan oleh klien dengan token JWT yang ditambahkan dalam string kueri - dan beri nama "OAuth2AuthenticationSuccessHandler".

**Source Code :**

```
@Component  
public class OAuth2AuthenticationSuccessHandler extends SimpleUrlAuthenticationSuccessHandler {  
  
    private TokenProvider tokenProvider;  
  
    private AppProperties appProperties;  
  
    private HttpCookieOAuth2AuthorizationRequestRepository  
httpCookieOAuth2AuthorizationRequestRepository;  
  
    @Autowired  
    OAuth2AuthenticationSuccessHandler(TokenProvider tokenProvider, AppProperties appProperties,  
                                        HttpCookieOAuth2AuthorizationRequestRepository  
httpCookieOAuth2AuthorizationRequestRepository) {  
        this.tokenProvider = tokenProvider;  
        this.appProperties = appProperties;  
        this.httpCookieOAuth2AuthorizationRequestRepository =  
httpCookieOAuth2AuthorizationRequestRepository;  
    }  
  
    @Override
```

```

    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
Authentication authentication) throws IOException, ServletException {
        String targetUrl = determineTargetUrl(request, response, authentication);

        if (response.isCommitted()) {
            logger.debug("Response has already been committed. Unable to redirect to " +
targetUrl);
            return;
        }

        clearAuthenticationAttributes(request, response);
        getRedirectStrategy().sendRedirect(request, response, targetUrl);
    }

    protected String determineTargetUrl(HttpServletRequest request, HttpServletResponse response,
Authentication authentication) {
        Optional<String> redirectUri = CookieUtils.getCookie(request,
REDIRECT_URI_PARAM_COOKIE_NAME)
            .map(Cookie::getValue);

        if(redirectUri.isPresent() && !isAuthorizedRedirectUri(redirectUri.get())) {
            throw new BadRequestException("Sorry! We've got an Unauthorized Redirect URI and
can't proceed with the authentication");
        }

        String targetUrl = redirectUri.orElse(getDefaultTargetUrl());

        String token = tokenProvider.createToken(authentication);

        return UriComponentsBuilder.fromUriString(targetUrl)
            .queryParam("token", token)
            .build().toUriString();
    }

    protected void clearAuthenticationAttributes(HttpServletRequest request, HttpServletResponse
response) {
        super.clearAuthenticationAttributes(request);
        httpCookieOAuth2AuthorizationRequestRepository.removeAuthorizationRequestCookies(request,
response);
    }

    private boolean isAuthorizedRedirectUri(String uri) {
        URI clientRedirectUri = URI.create(uri);

        return appProperties.getOAuth2().getAuthorizedRedirectUris()
            .stream()
            .anyMatch(authorizedRedirectUri -> {
// Only validate host and port. Let the clients use different paths if they
want to

                URI authorizedURI = URI.create(authorizedRedirectUri);
                if(authorizedURI.getHost().equalsIgnoreCase(clientRedirectUri.getHost())
                    && authorizedURI.getPort() == clientRedirectUri.getPort()) {
                    return true;
                }
                return false;
            });
    }
}

```

```
}
```

7.5. Begitu juga sebaliknya, kita buat perilaku apa yang akan di lakukan jika autentikasi dengan provider gagal. Buat file baru dan beri nama "OAuth2AuthenticationFailureHandler".

**Source Code:**

```
@Component
public class OAuth2AuthenticationFailureHandler extends SimpleUrlAuthenticationFailureHandler {

    @Autowired
    HttpCookieOAuth2AuthorizationRequestRepository httpCookieOAuth2AuthorizationRequestRepository;

    @Override
    public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response,
        AuthenticationException exception) throws IOException, ServletException {
        String targetUrl = CookieUtils.getCookie(request, REDIRECT_URI_PARAM_COOKIE_NAME)
            .map(Cookie::getValue)
            .orElse("/");

        targetUrl = UriComponentsBuilder.fromUriString(targetUrl)
            .queryParams("error", exception.getLocalizedMessage())
            .build().toUriString();

        httpCookieOAuth2AuthorizationRequestRepository.removeAuthorizationRequestCookies(request,
            response);

        getRedirectStrategy().sendRedirect(request, response, targetUrl);
    }
}
```

8. Yang selanjutnya adalah kita konfigurasi untuk autentifikasi pengguna yang menggunakan email dan password atau http basic.

8.1. Buat file Controller untuk autentifikasi. Isikan kode berikut.

```
// REST CONTROLLER API AUTHENTIFIKASI
@RestController
@RequestMapping("/auth")
public class AuthController {

    // AuthenticationManager => untuk manajemen autentifikasi
    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private TokenProvider tokenProvider;

    @Autowired
    private JavaMailSender mailSender;
}
```

```
private PonTools ponTools;

public AuthController(){
    this.ponTools = new PonTools();
}

// ENDPOINT LOGIN LOKAL
@PostMapping("/login")
public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest)
throws UnsupportedEncodingException, MessagingException {

    // Cari user di database berdasarkan email
    User user = userRepository.findOneByEmail(loginRequest.getEmail());

    // jika user ada di database
    if (user!=null) {
        // jika email user sudah terverifikasi
        if (user.getEmailVerified()) {
            // buat autentifikasi baru
            Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                    loginRequest.getEmail(),
                    loginRequest.getPassword()
                )
            );

            // Setting autentifikasi
            SecurityContextHolder.getContext().setAuthentication(authentication);
            // buat bareer token untuk akses
            String token = tokenProvider.createToken(authentication);

            // Respon 200 dan kembalikan token autentifikasi
            return ResponseEntity.ok(new AuthResponse(token));

            // jika email belum terverifikasi
        }else{
            // Buat token verifikasi email
            String token = RandomString.make(64);
            user.setTokEmailVerified(token);
            userRepository.save(user);

            // Kirimkan link untuk verifikasi email (media:irim email)
            ponTools.sendMailWithButton(
                mailSender,
                user,
                "PONLINE SUPPORT o verify email",
                "Click the button below to verify your email.",
                "",
                "",
                "If the 'Verify Email' button cannot be used, click on the following link: ",
                "Verify Email",
                ponTools.ponlineBaseUrl()+"/auth/verifymail?token=" + token
            );

            // response 500 dan kembalikan pesan untuk memverifikasi email
        }
    }
}
```

```
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new ApiResponse(false,
"Akun anda belum diverifikasi, Silahkan cek email untuk memverifikasi akun anda"));
    }
    // Jika user tidak ada di database
} else {
    // response 500 dan kembalikan pesan bahwa akun belum terdaftar
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new ApiResponse(false,
"Akun anda belum terdaftar"));
}
}

// ENPOINT REGISTER LOKAL
@PostMapping("/signup")
public ResponseEntity<?> registerUser(@Valid @RequestBody SignUpRequest signUpRequest) throws
UnsupportedEncodingException, MessagingException {
    // Jika user sudah ada di database
    if (userRepository.existsByEmail(signUpRequest.getEmail())) {
        // Respon 500, pesan user sudah terdaftar
        throw new BadRequestException("Alamat email sudah terdaftar");
    }

    // Jika belum terdaftar
    // Buat objek user
    User user = new User();
    user.setName(signUpRequest.getName());
    user.setEmail(signUpRequest.getEmail());
    user.setPassword(signUpRequest.getPassword());
    user.setProvider(AuthProvider.local);
    user.setRole(UserRole.ROLE_USER);

    // Buat token verifikasi email
    String token = RandomString.make(64);
    user.setEmailVerified(false);
    user.setTokenEmailVerified(token);

    // Enkripsi password user
    user.setPassword(passwordEncoder.encode(user.getPassword()));

    // Simpan user ke database
    userRepository.save(user);

    // Kirimkan email verifikasi
    ponTools.sendMailWithButton(
        mailSender,
        user,
        "PONLINE SUPPORT o verify email",
        "Click the button below to verify your email.",
        "",
        "",
        "If the 'Verify Email' button cannot be used, click on the following link: ",
        "Verify Email",
        ponTools.ponlineBaseUrl() + "/auth/verifyemail?token=" + token
    );

    // Respon 200, pesan perintahkan verifikasi email
    return ResponseEntity.ok(new ApiResponse(true, "Berhasil mendaftar, Silahkan cek email
untuk verifikasi"));
}
```

```
}

// ENDPOINT verifikasi email
@GetMapping("/verifymail")
public ResponseEntity<?> verifymail(@RequestParam String token) {
    // Cek apakah user ada di database atau tidak
    if(userRepository.findByTokEmailVerified(token).isEmpty()) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(new ApiResponse(false, "User
not Found"));
    }

    // Jika user ada setting email verifikasi = true
    User user = userRepository.findByTokEmailVerified(token).get(0);
    user.setEmailVerified(true);
    user.setTokEmailVerified(null);
    userRepository.save(user);
    // response 200
    return ResponseEntity.ok(new ApiResponse(true, "Berhasil memverifikasi email"));
}

// ENDPOINT lupa password
@GetMapping("/forgotpassword")
public ResponseEntity<?> forgotpassword(@RequestParam String email, HttpServletRequest
request) throws UnsupportedEncodingException, MessagingException {
    // Cek user ada atau tidak
    if(!userRepository.existsByEmail(email)) {
        throw new ResourceNotFoundException("User", "Email", email);
    }

    // jika ada, buat token untuk reset password
    User user = userRepository.findOneByEmail(email);
    String token = RandomString.make(64);
    user.setTokResetPassword(token);
    userRepository.save(user);

    // Kirimkan link/token untuk reset password
    ponTools.sendMailWithButton(
        mailSender,
        user,
        "PONLINE SUPPORT o forgot password",
        "You are receiving this email because we received a password reset request for your
account.",
        "This password reset link will expire in 60 minutes.",
        "If you did not request a password reset, no further action is required.",
        "If you're having trouble clicking the \"Reset Password\" button, copy and paste the
URL below into your web browser: ",
        "Reset Password",
        ponTools.ponlineBaseUrl()+"/auth/resetpassword?token=" + token
    );

    // respon 200
    return ResponseEntity.ok(new ApiResponse(true, "Silahkan cek email untuk mereset password
anda"));
}

// ENDPOINT proses buat/ubah password
```

```

@PostMapping("/resetpassword")
public ResponseEntity<?> resetPasswordUser(@RequestParam String token, @Valid @RequestBody
ResetPasswordRequest resetPasswordRequest) {
    // Cek user ada atau tidak
    if(userRepository.findByTokResetPassword(token).isEmpty()) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(new ApiResponse(false, "User
tidak di temukan"));
    }

    // Jika ada, ganti password menjadi password baru
    User user = userRepository.findByTokResetPassword(token).get(0);
    user.setPassword(passwordEncoder.encode(resetPasswordRequest.getPassword()));
    user.setTokResetPassword(null);
    userRepository.save(user);

    // respon 200
    return ResponseEntity.ok(new ApiResponse(true, "Berhasil memperbarui password"));
}
}

```

#### Penjelasan :

Khusus untuk pengguna yang mendaftar tanpa provider, pengguna harus memverifikasi emailnya terlebih dahulu untuk bisa login. Untuk penjelasan mengenai pengiriman email dapat di lihat pada poin ([Pengiriman Email](#)).

8.3. Yang selanjutnya adalah kita membuat kelas service untuk pengguna. Dalam kelas service, biasanya berisi bisnis model dari alur program. Di kelas ini juga berisi fungsi fungsi yang di butuhkan oleh kelas controller.

#### Source Code :

```

// Tambahkan anotasi @aervice untuk memberitahu sistem bahwa ini adalah class service
// Implementasi class 'UserDetailsService'
@Service
public class CustomUserDetailsService implements UserDetailsService {

    // Injeksi class 'UserRepository'
    @Autowired
    UserRepository userRepository;

    // Tulis ulang (override) method 'loadUserByUsername' dari lib 'UserDetailsService'
    @Override
    @Transactional
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        // Karena pada ERD tabel user tidak ada attribut username, maka username di ganti dengan
        email

        // Cari user di database dan simpan ke objek user
        User user = userRepository.findByEmail(email).orElseThrow(() ->
            // Jika error maka response user tidak di temukan
            new UsernameNotFoundException("Pengguna dengan email : " + email + " tidak ditemukan")
        );

        // Buat Kelas user utama dan kembalikan
        return UserPrincipal.create(user);
    }

    // Sama kaya sebelumnya tapi berdasarkan id
    @Transactional

```



```
public UserDetails loadUserById(Long id) {  
    User user = userRepository.findById(id).orElseThrow(  
        () -> new ResourceNotFoundException("User", "id", id)  
    );  
  
    return UserPrincipal.create(user);  
}  
}
```

10. Selanjutnya kita buat kelas untuk generate token. Dimana token ini nantinya dapat di gunakan oleh pengguna untuk mengakses data menggunakan autentikasi bearer.

Source Code :

```
@Service  
public class TokenProvider {  
  
    private static final Logger logger = LoggerFactory.getLogger(TokenProvider.class);  
  
    private AppProperties appProperties;  
  
    public TokenProvider(AppProperties appProperties) {  
        this.appProperties = appProperties;  
    }  
  
    // method buat autentifikasi token akses  
    public String createToken(Authentication authentication) {  
        // Ambil data user utama di objek autentifikasi  
        UserPrincipal userPrincipal = (UserPrincipal) authentication.getPrincipal();  
        // Buat objek tanggal sekarang / di buatnya token  
        Date now = new Date();  
        // Buat tanggal kadaluarsa untuk tokenya (diambil dari app properti)  
        Date expiryDate = new Date(now.getTime() +  
appProperties.getAuth().getTokenExpirationMsec());  
        // Proses buat token dan kembalikan  
        return Jwts.builder()  
            .setSubject(Long.toString(userPrincipal.getId()))  
            .setIssuedAt(new Date())  
            .setExpiration(expiryDate)  
            .signWith(SignatureAlgorithm.HS512, appProperties.getAuth().getTokenSecret())  
            .compact();  
    }  
  
    public Long getUserIdFromToken(String token) {  
        Claims claims = Jwts.parser()  
            .setSigningKey(appProperties.getAuth().getTokenSecret())  
            .parseClaimsJws(token)  
            .getBody();  
  
        return Long.parseLong(claims.getSubject());  
    }  
  
    public boolean validateToken(String authToken) {  
        try {  
            Jwts.parser().setSigningKey(appProperties.getAuth().getTokenSecret()).parseClaimsJws(a  
uthToken);  
  
            return true;  
        }  
    }  
}
```

```
    } catch (SignatureException ex) {
        logger.error("Invalid JWT signature");
    } catch (MalformedJwtException ex) {
        logger.error("Invalid JWT token");
    } catch (ExpiredJwtException ex) {
        logger.error("Expired JWT token");
    } catch (UnsupportedJwtException ex) {
        logger.error("Unsupported JWT token");
    } catch (IllegalArgumentException ex) {
        logger.error("JWT claims string is empty.");
    }
    return false;
}
```

11. Kita buat file baru untuk menyaring filter autentifikasi.

Source Code :

```
public class TokenAuthenticationFilter extends OncePerRequestFilter {

    @Autowired
    private TokenProvider tokenProvider;

    @Autowired
    private CustomUserDetailsService customUserDetailsService;

    private static final Logger logger = LoggerFactory.getLogger(TokenAuthenticationFilter.class);

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
    FilterChain filterChain) throws ServletException, IOException {
        try {
            String jwt = getJwtFromRequest(request);

            if (StringUtils.hasText(jwt) && tokenProvider.validateToken(jwt)) {
                Long userId = tokenProvider.getUserIdFromToken(jwt);

                UserDetails userDetails = customUserDetailsService.loadUserById(userId);
                UsernamePasswordAuthenticationToken authentication = new
                UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
                authentication.setDetails(new
                WebAuthenticationDetailsSource().buildDetails(request));

                SecurityContextHolder.getContext().setAuthentication(authentication);
            }
        } catch (Exception ex) {
            logger.error("Could not set user authentication in security context", ex);
        }

        filterChain.doFilter(request, response);
    }

    private String getJwtFromRequest(HttpServletRequest request) {
        String bearerToken = request.getHeader("Authorization");
        if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring(7, bearerToken.length());
        }
    }
}
```

```
        return null;
    }
}
```

12. Selanjutnya kita membuat kelas entri poin untuk autentifikasi. Jadi jika nanti ada pengguna yang ingin mengakses data tanpa atau salah autentikasi, sistem akan merespon 401 atau tidak sah.

**Source Code :**

```
public class RestAuthenticationEntryPoint implements AuthenticationEntryPoint {

    private static final Logger logger =
        LoggerFactory.getLogger(RestAuthenticationEntryPoint.class);

    @Override
    public void commence(HttpServletRequest httpServletRequest, HttpServletResponse
        httpServletResponse, AuthenticationException e) throws IOException, ServletException {
        logger.error("Menanggapi dengan kesalahan yang tidak sah. Pesan : {}", e.getMessage());
        httpServletResponse.sendError(HttpServletResponse.SC_UNAUTHORIZED,
            e.getLocalizedMessage());
    }
}
```

13. Selanjutnya kita buat kelas "UserPrincipal"

Kelas UserPrincipal mewakili kepala sekolah Spring Security yang diautentikasi. Ini berisi detail pengguna yang diautentikasi. Mudah-mudahan, kita satukan objek baik yang autentikasi menggunakan provider atau email dan password;

**Source Code :**

```
public class UserPrincipal implements OAuth2User, UserDetails {
    private Long id;
    private String email;
    private String password;
    private Collection<? extends GrantedAuthority> authorities;
    private Map<String, Object> attributes;

    public UserPrincipal(Long id, String email, String password, Collection<? extends
        GrantedAuthority> authorities) {
        this.id = id;
        this.email = email;
        this.password = password;
        this.authorities = authorities;
    }

    // Buat Kelas User Utama
    public static UserPrincipal create(User user) {
        // List otoritas yang akan di berikan ke user
        List<GrantedAuthority> authorities = Collections.singletonList(new
        SimpleGrantedAuthority(user.getRole().name()));

        // Kembalikan objek user utama
        return new UserPrincipal(
            user.getId(),
            user.getEmail(),
            user.getPassword(),
            authorities
        );
    }
}
```

```
}

public static UserPrincipal create(User user, Map<String, Object> attributes) {
    UserPrincipal userPrincipal = UserPrincipal.create(user);
    userPrincipal.setAttributes(attributes);
    return userPrincipal;
}

public Long getId() {
    return id;
}

public String getEmail() {
    return email;
}

@Override
public String getPassword() {
    return password;
}

@Override
public String getUsername() {
    return 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;
}

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return authorities;
}

@Override
public Map<String, Object> getAttributes() {
    return attributes;
}

public void setAttributes(Map<String, Object> attributes) {
```

```
        this.attributes = attributes;
    }

    @Override
    public String getName() {
        return String.valueOf(id);
    }
}
```

14. Selanjutnya adalah kita membuat kelas anotasi antarmuka untuk pengguna yang sedang login. Buat file baru dan beri nama "CurrentUser". Kelas ini nanti dapat kita suntikan ke dalam kelas controller untuk mendapatkan detail data user yang sedang login.

**Source Code:**

```
@Target({ElementType.PARAMETER, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@AuthenticationPrincipal
public @interface CurrentUser {
}
```

15. Selanjutnya kita membuat kelas controller untuk pengguna, di dalam kelas ini berisikan endpoint untuk mengelola user profil yang sedang login.

**Source Code :**

```
@RestController
public class UserController {

    @Autowired
    private UserRepository userRepository;

    @GetMapping("/user/details")
    @PreAuthorize("hasRole('USER')")
    public User getCurrentUser(@CurrentUser UserPrincipal userPrincipal) {
        return userRepository.findById(userPrincipal.getId())
            .orElseThrow(() -> new ResourceNotFoundException("User", "id",
                userPrincipal.getId()));
    }
}
```

16. Selanjutnya kita buat file kelas baru untuk mengatur Cookie.

**Source Code :**

```
public class CookieUtils {

    public static Optional<Cookie> getCookie(HttpServletRequest request, String name) {
        Cookie[] cookies = request.getCookies();

        if (cookies != null && cookies.length > 0) {
            for (Cookie cookie : cookies) {
                if (cookie.getName().equals(name)) {

```

```

        return Optional.of(cookie);
    }
}

return Optional.empty();
}

public static void addCookie(HttpServletResponse response, String name, String value, int
maxAge) {
    Cookie cookie = new Cookie(name, value);
    cookie.setPath("/");
    cookie.setHttpOnly(true);
    cookie.setMaxAge(maxAge);
    response.addCookie(cookie);
}

public static void deleteCookie(HttpServletRequest request, HttpServletResponse response,
String name) {
    Cookie[] cookies = request.getCookies();
    if (cookies != null && cookies.length > 0) {
        for (Cookie cookie: cookies) {
            if (cookie.getName().equals(name)) {
                cookie.setValue("");
                cookie.setPath("/");
                cookie.setMaxAge(0);
                response.addCookie(cookie);
            }
        }
    }
}

public static String serialize(Object object) {
    return Base64.getUrlEncoder()
        .encodeToString(SerializationUtils.serialize(object));
}

public static <T> T deserialize(Cookie cookie, Class<T> cls) {
    return cls.cast(SerializationUtils.deserialize(
        Base64.getUrlDecoder().decode(cookie.getValue())));
}
}

```

17. Selanjutnya kita membuat DAO atau pengelolaan data yang dibutuhkan dan di kirimkan untuk autentikasi.

17.1. Buat file baru dan beri nama "SignUpRequest"

**SourceCode :**

```

public class SignUpRequest {
    @NotBlank
    private String name;

    @NotBlank
    @Email
    private String email;
}

```

```
@NotBlank
private String password;

//setter getter
}
```

**Penjelasan :**

Untuk mendaftarkan pengguna baru, pengguna hanya mengirimkan data nama, email, dan passwordnya saja.

- 17.2. Buat file baru dan beri nama "LoginRequest"

**SourceCode :**

```
public class LoginRequest {
    @NotBlank
    @Email
    private String email;

    @NotBlank
    private String password;

    // setter getter
}
```

**Penjelasan :**

Untuk login pengguna, pengguna hanya mengirimkan data email, dan passwordnya saja.

- 17.3. Buat file baru dan beri nama "AuthResponse"

**SourceCode :**

```
public class AuthResponse {
    private String accessToken;
    private String tokenType = "qtwoekretnypuoino1p";

    public AuthResponse(String accessToken) {
        this.accessToken = accessToken;
    }

    public String getAccessToken() {
        return accessToken;
    }

    public void setAccessToken(String accessToken) {
        this.accessToken = accessToken;
    }

    public String getTokenType() {
        return tokenType;
    }

    public void setTokenType(String tokenType) {
        this.tokenType = tokenType;
    }
}
```

**Penjelasan :**

Untuk pengguna yang berhasil login, respon yang di kirimkan oleh sistem adalah token akses dan token type. Token akses adalah token yang nanti dapat digunakan untuk autentikasi bearer dalam mengakses atau meminta data.

18. Selanjutnya kita buat kelas respon untuk akses data yang gagal.

18.1. `BadRequestException`. Di gunakan untuk merespon permintaan yang salah. Contoh kesalahan url endpoint, atau kesalahan data yang di kirimkan dari sisi frontend.

**Source Code :**

```
@ResponseStatus(HttpStatus.BAD_REQUEST)
public class BadRequestException extends RuntimeException {
    public BadRequestException(String message) {
        super(message);
    }

    public BadRequestException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

18.2. `ResourceNotFoundException`. Digunakan untuk merespon permintaan yang datanya tidak ada di database.

**Source Code :**

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {
    private String resourceName;
    private String fieldName;
    private Object fieldValue;

    public ResourceNotFoundException(String resourceName, String fieldName, Object fieldValue) {
        super(String.format("%s dengan %s '%s' tidak di temukan", resourceName, fieldName,
            fieldValue));
        this.resourceName = resourceName;
        this.fieldName = fieldName;
        this.fieldValue = fieldValue;
    }

    public String getResourceName() {
        return resourceName;
    }

    public String getFieldName() {
        return fieldName;
    }

    public Object getFieldValue() {
        return fieldValue;
    }
}
```

18.3. `OAuth2AuthenticationProcessingException`. Digunakan untuk merespon permintaan, namun terdapat kesalahan dalam autentifikasinya. Contoh token bearer yang salah.

**Source Code :**

```
public class OAuth2AuthenticationProcessingException extends AuthenticationException {
```



```
public OAuth2AuthenticationProcessingException(String msg, Throwable t) {  
    super(msg, t);  
}  
  
public OAuth2AuthenticationProcessingException(String msg) {  
    super(msg);  
}  
}
```

Referensi :

[Spring Boot OAuth2 Social Login with Google, Facebook, and Github - Part 1 | CalliCoder](#)

[Spring Boot OAuth2 Social Login with Google, Facebook, and Github - Part 2 | CalliCoder](#)

[Spring Boot OAuth2 Social Login with Google, Facebook, and Github - Part 3 | CalliCoder](#)

## Pengiriman Email

Pertama, kita harus menambahkan properti akun email yang nantinya akan di gunakan untuk email pengirim.

Pada file application.yml, tambahkan kode berikut :

```
# Konfigurasi Properti Email Sender  
mail:  
  host: smtp.gmail.com  
  port: 587  
  username: fredihermawan1211@gmail.com  
  password: fmeFEXmZkN2FmQ  
  protocol: smtp  
  tls: true  
  properties.mail.smtp:  
    auth: true  
    starttls.enable: true  
    ssl.trust: smtp.gmail.com
```

Penjelasan :

host → adalah host email server

port → port email

email → alamat email akun pengirim

password → password akun pengirim (baca [di sini](#) untuk menyiapkan password aplikasi aman masuk ke google)

Setelah itu, buat fungsi untuk mengirim email seperti kode beriku :

```
public void sendMailWithButton(  

```

```
JavaMailSender mailSender, User user,
String subject, String contentBody1, String contentBody2, String contentBody3, String
contentBody4,
String labelButton, String link
) throws UnsupportedEncodingException, MessagingException {

String content = emailContentTemplate.getMailButton(
    subject,
    link,
    labelButton,
    contentBody1,contentBody2,contentBody3,contentBody4

);

MimeMessage message = mailSender.createMimeMessage();
MimeMessageHelper helper = new MimeMessageHelper(message);

helper.setFrom("support@ponline.com", "PONLINE SUPPORT");
helper.setTo(user.getEmail());
helper.setSubject(subject);
helper.setText(content, true);

mailSender.send(message);
}
```

#### Penjelasan :

emailContentTemplate → adalah string yang berisi sintak html desain dari email. Hal ini adalah pilihan

Untuk pengiriman email ini, kita dapat menggunakan library JavaMailSender. Dan kebutuhan untuk mengirim email menggunakan JMS ini adalah email pengirim, email penerima, subjek, dan isi atau body dari email tersebut.

## Pengujian Autentikasi

Dalam pengujian autentikasi kali ini, kita menggunakan aplikasi insomnia sebagai REST API Client. untuk

1. Mendaftar pengguna baru dengan email dan password.

Metode : POST

URLs endpoint : <http://localhost:8080/auth/signup>

Request Body :

```
{
  "name": "PONLINE USER 1",
  "email": "pon_user1@quickat.work",
  "password": "123456"
}
```

Response : ApiResponse

The screenshot shows the Insomnia REST client interface. At the top, it displays the status '200 OK', the time taken '7.36 s', and the response size '84 B'. Below this, there are tabs for 'Preview', 'Headers', 'Cookies', and 'Timeline'. The 'Preview' tab is selected, showing the JSON response body:

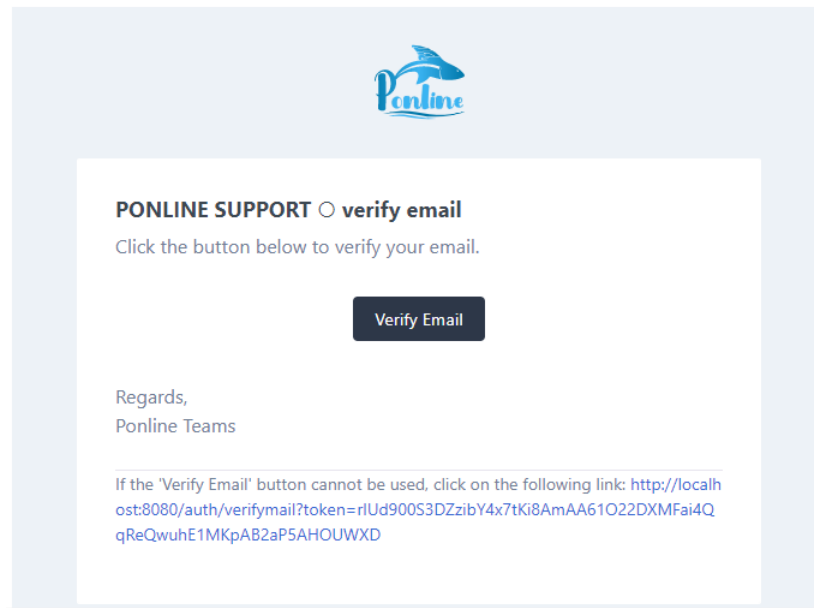
```
1 {
2   "success": true,
3   "message": "Berhasil mendaftar, Silahkan cek email untuk verifikasi"
4 }
```

Email Verifikasi :

PONLINE SUPPORT ○ verify email

PONLINE SUPPORT - fredihermawan1211@gmail.com

2022-12-11 04:28 PM



Database : status akun tidak aktif

Token verifikasi email

ofvdobanjli	tbfewjerbc	#omfuuvud	wbrfpgtm	ggxafxwykfd	jltxhtodmv	oouombbxli	oebwazzvdi	uwegexwuf	xvfeumirr	typhbwqgm
1 pon_user1@quickatwork	0	NULL	PONLINE USER 1	\$2a\$10\$R9XyqjpsUuM8BT3uJOG6JHD4Nc1C8DYShmh...	local	NULL	NULL	ROLE_USER	rlUd900S3DZzibY4x7tKi8AmAA61O22DXMFai4QqReQwuhE1MK	NULL

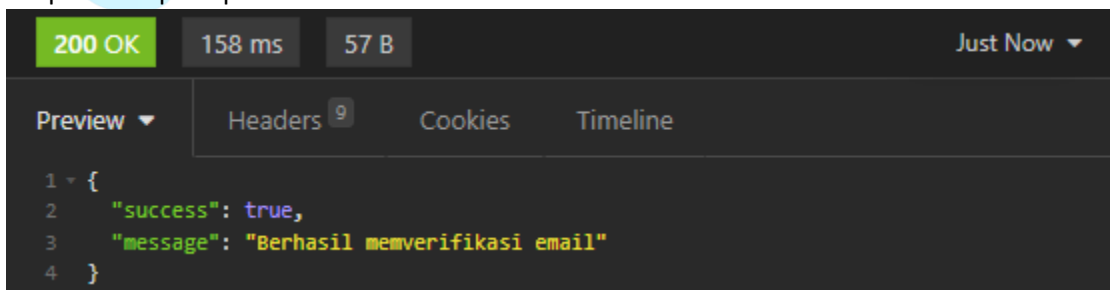
## 2. Memverifikasi Email

Metode: GET

URLs Endpoint : <http://localhost:8080/auth/verifyemail?token={tokenVerifyEmail}>

Request Body : none

Response : ApiResponse



Database : status akun aktif

Token email dihapus setelah verifikasi

ofvdobanjli	tbfewjerbc	#omfuuvud	kwbrfpgtm	ggxafxwykfd	jltxhtodmv	oouombbxli	oebwazzvdi	uwegexwuf	xvfeumirr	typhbwqgm
1 pon_user1@quickatwork	1	NULL	PONLINE USER 1	\$2a\$10\$R9XyqjpsUuM8BT3uJOG6JHD4Nc1C8DYShmh...	local	NULL	NULL	ROLE_USER	NULL	NULL

## 3. Login menggunakan email dan password

Metode : POST

URLs Endpoint : <http://localhost:8080/auth/login>

Request Body :

```
JSON ▾ Auth ▾ Query Headers 1 Docs
1 {
2   "email": "pon_user1@quickat.work",
3   "password": "123456"
4 }
```

Response sukses: AuthResponse

```
200 OK 208 ms 220 B Just Now ▾
Preview ▾ Headers 9 Cookies Timeline
1 {
2   "accessToken":
3     "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIxIiwiaWF0IjoxNjcwNzUxNjM2LCJleHAiOjE2NzA4MzgWmZz9.S
4     06PQsowSGwt5hh4wNqk7jKebAnlRzUM215MAN9LZWRJ_GXsYUSWM9UoZmOLBBaW41CoodVE8LzXyfmdm1f7
5     g",
6   "tokenType": "qtwoekretnypuoino1p"
7 }
```

Response email / password salah : BadRequestException

```
POST http://localhost:8080/auth/login Send ▾ 401 Unauthorized 117 ms 129 B Just Now ▾
JSON ▾ Auth ▾ Query Headers 8 Docs Preview ▾ Headers 9 Cookies Timeline
1 {
2   "email": "pon_user1@quickat.work",
3   "password": "1234"
4 }
1 {
2   "timestamp": "2022-12-11T09:56:28.588+0000",
3   "status": 401,
4   "error": "Unauthorized",
5   "message": "Bad credentials",
6   "path": "/auth/login"
7 }
```

#### 4. Permintaan lupa atau ganti password pengguna

Metode : GET

URL Endpoint : <http://localhost:8080/auth/forgotpassword?email={email yang akan direset}>

Request Body : none

Response : ApiResponse

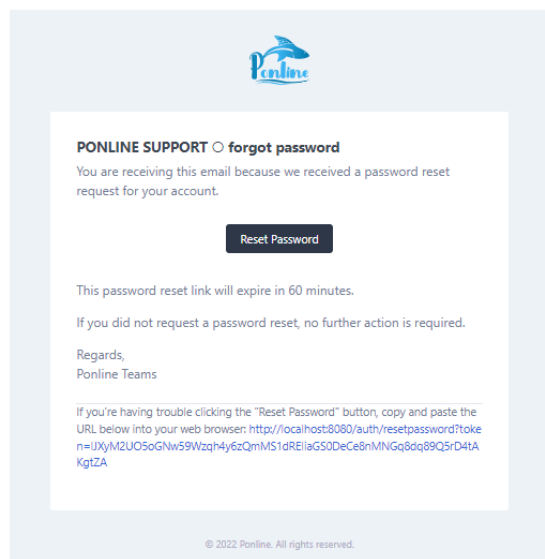
```
200 OK 4.76 s 75 B Just Now ▾
Preview ▾ Headers 9 Cookies Timeline
1 {
2   "success": true,
3   "message": "Silahkan cek email untuk mereset password anda"
4 }
```

Email Reset Password :

PONLINE SUPPORT ○ forgot password

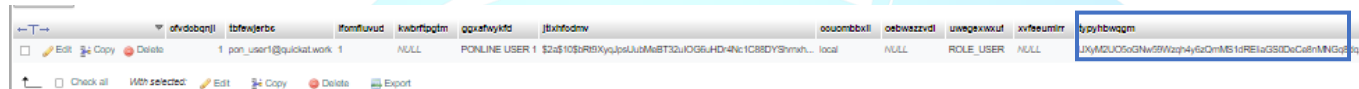
PONLINE SUPPORT - fredihermawan1211@gmail.com

2022-12-11 04:42 PM



Database :

Token Reset Password

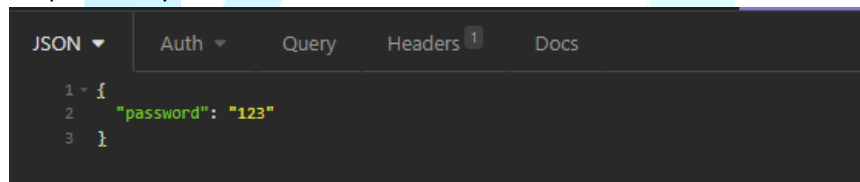


## 5. Mengganti password

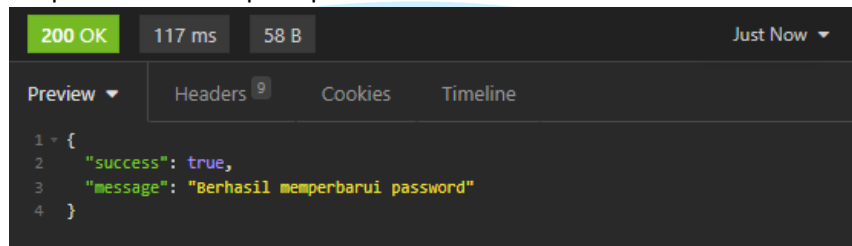
Metode : POST

URLs Endpoint : [http://localhost:8080/auth/resetpassword?token={token\\_reset\\_password}](http://localhost:8080/auth/resetpassword?token={token_reset_password})

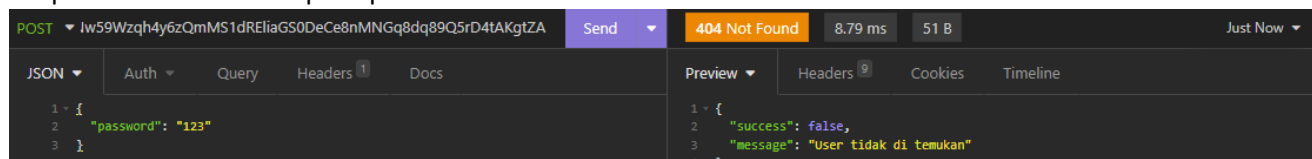
Request Body :



Response sukses : ApiResponse

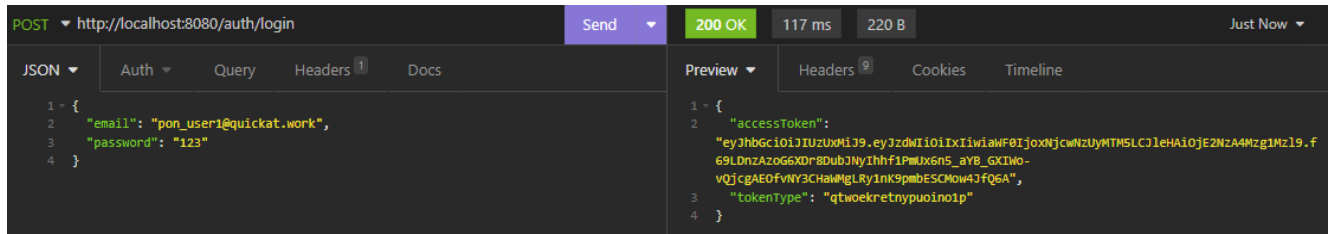


Response salah token : ApiResponse



Uji Coba login dengan password baru :

Author : Fredy Hermawan as BackEnd Developer

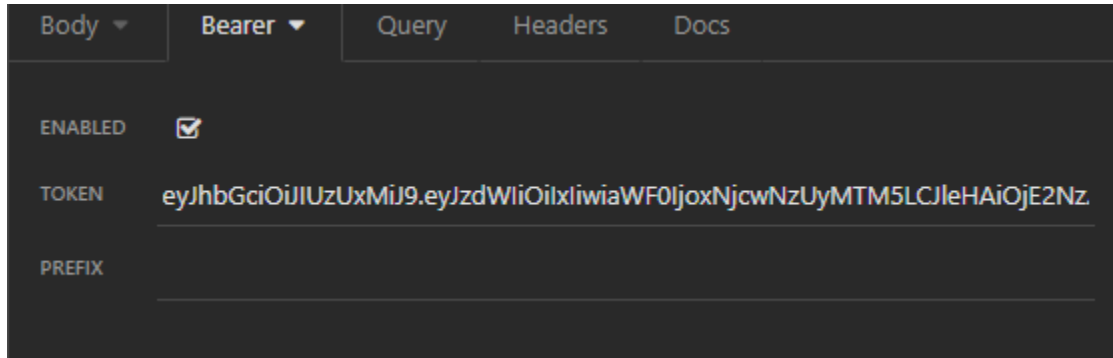


## 6. Mendapatkan Data user (user details/profil)

Metode : GET

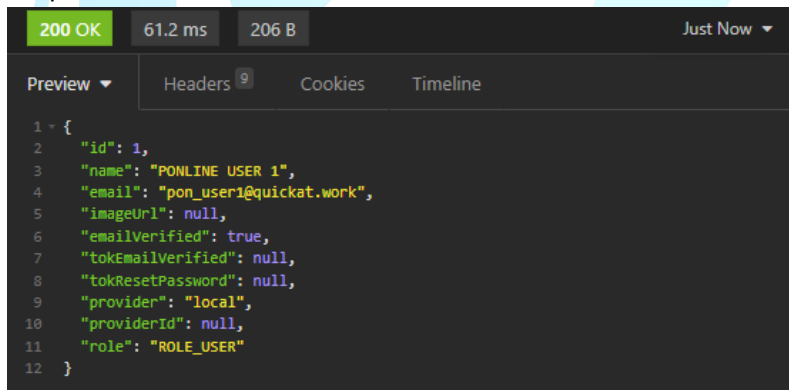
URLs Endpoint : <http://localhost:8080/user/details>

Autentikasi : bearer → accessToken



Request Body : none

Response : Entitas User



## API Untuk Data komunitas

1. Kita buat file repositorinya dulu untuk menampung perintah perintah kueri guna mengelola data di database  
**Source Code :**

```

public interface KomunitasRepo extends CrudRepository<Komunitas, Long>{
}

```

Penjelasan :

Kelas ini hanya menurunkan kelas dari CrudRespository. Dalam kelas CrusRepository sudah terdapat fungsi – fungsi untuk DML seperti Create, Update, Delete. Read.

2. Kita buat file service untuk menampung metode atau fungsi – fungsi perilaku yang sesuai dengan kebutuhan bisnis modelsnya.

**Source Code :**

```
@Service
@Transactional
public class KomunitasServices{

    @Autowired
    private KomunitasRepo komunitasRepo;

    public Komunitas save(Komunitas komunitas) {
        return komunitasRepo.save(komunitas);
    }

    public Komunitas findOne(Long id) {
        return komunitasRepo.findById(id).get();
    }

    public Boolean isExist(Long id) {
        Komunitas komunitas = findOne(id);
        if (komunitas!=null) {
            return true;
        }else{
            return false;
        }
    }

    public Iterable<Komunitas> findAll() {
        return komunitasRepo.findAll();
    }

    public Boolean deleteById(Long id) {
        if (isExist(id)) {
            komunitasRepo.deleteById(id);
            return true;
        }else{
            return false;
        }
    }
}
```

**Penjelasan :**

Fungsi save → untuk menyimpan data. Fungsi ini mengembalikan objek komunitas.

Fungsi findOne → untuk mencari data berdasarkan idnya. Fungsi ini mengembalikan objek komunitas.

Fungsi isExist → untuk memeriksa atau memastikan bahwa data ada di database. Fungsi ini mengembalikan nilai Boolean.

Fungsi findAll → untuk melihat semua data komunitas yang ada di database. Fungsi ini mengembalikan objek Iterable yang berisi list data komunitas.

Fungsi deleteById → untuk menghapus satu data komunitas. Sebelum data di hapus, data tersebut di periksa apakah berada di database atau tidak. Jika ya, maka proses menghapus akan di lakukan dan akan di kembalikan nilai Boolean true. Jika tidak, maka akan di kembalikan Boolean false.

3. Kita buat file controller untuk menampung dan menyusun fungsi fungsi yang ada di service. Di kelas ini juga menampung endpoint – endpoint yang nanti akan di konsumsi oleh frontend.

**Source Code :**

```
// Anotasi RESTController untuk menandakan bahwa ini kelas rest controller
@RestController
// Set base url endpoint (baseurl/komunitas)
```

```
@RequestMapping("/komunitas")
public class KomunitasController {

    // Inject komunitas service untuk memakai fungsi fungsi yg ada di kelas service
    @Autowired
    private KomunitasServices komunitasServices;

    // inject model mapper untuk memudahkan penyusunan data dari json yang di kirimkan frontend ke objek
    @Autowired
    private ModelMapper modelMapper;

    // endpoint untuk menyimpan data
    // anotasi untuk menandakan metode yang di gunakan adalah POST
    @PostMapping
    public ResponseEntity<AbstractResponse<Komunitas>> create(@Valid @RequestBody KomunitasRequest komunitasRequest, Errors errors ) {

        // Siapkan objek kosong untuk di kembalikan
        AbstractResponse<Komunitas> responseData = new AbstractResponse<>();

        // periksa jika ada error
        if (errors.hasErrors()) {
            for (ObjectError error : errors.getAllErrors()) {
                responseData.getMessage().add(error.getDefaultMessage());
            }
            responseData.setSuccess(false);
            responseData.setPayload(null);

            // kembalikan AbstractResponse dengan pesan gagal dan kode error 500
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(responseData);
        }

        // jika tidak ada error, maka akan di kembalikan respon 200
        Komunitas komunitas = modelMapper.map(komunitasRequest, Komunitas.class);
        responseData.setSuccess(true);
        responseData.setPayload(komunitasServices.save(komunitas));
        return ResponseEntity.ok(responseData);
    }

    // buat endpoint untuk update, penjelasannya sama kaya simpan data hanya saja respon yang di terima sudah ada id objeknya
    // buat metode PUT
    @PutMapping
    public ResponseEntity<AbstractResponse<Komunitas>> update(@Valid @RequestBody KomunitasRequest komunitasRequest, Errors errors ) {

        AbstractResponse<Komunitas> responseData = new AbstractResponse<>();

        if (errors.hasErrors()) {
            for (ObjectError error : errors.getAllErrors()) {
                responseData.getMessage().add(error.getDefaultMessage());
            }
            responseData.setSuccess(false);
            responseData.setPayload(null);
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(responseData);
        }
    }
}
```



```

    }

    Komunitas komunitas = modelMapper.map(komunitasRequest, Komunitas.class);
    responseData.setSuccess(true);
    responseData.setPayload(komunitasServices.save(komunitas));
    return ResponseEntity.ok(responseData);

}

// Endpoitn delete
// metode DELETE
// id di baca dari url
@DeleteMapping("/delete/{id}")
public ApiResponse deleteById(@PathVariable("id") Long id) {

    // Buat respon gagal
    ApiResponse response = new ApiResponse(false, "Data Gagal Di hapus");

    // Periksa jika data berhasil di hapus
    if (komunitasServices.deleteById(id)) {
        // update respon menjadi berhasil
        response.setSuccess(true);
        response.setMessage("Data Berhasil Di hapus");
    }
    // kembalikan respon
    return response;
}

// enpoind untuk membaca semua data
@GetMapping
public Iterable<Komunitas> findAll(){
    return komunitasServices.findAll();
}

// endpoin untuk mencari data
// contoh url (base_url/find/id/{idnya})
@GetMapping("/find")
public Komunitas findOne(@RequestParam Long id) {
    return komunitasServices.findOne(id);
}
}

```

## API Untuk Data Anggota

Data anggota adalah data relasi dari komunitas dengan user

### 1. Kelas Respositori

**Source Code :**

```

// Turunkan fungsi CrudRepository
public interface AnggotaRepo extends CrudRepository<Anggota, Long>{

    // kueri cari anggota berdasarkan user
    List<Anggota> findByUser(User user);

    // kueri cari anggota berdasarkan komunitas
    List<Anggota> findByKomunitas(Komunitas komunitas);
}

```

```
}
```

## 2. Kelas Service

### Source Code :

```
// Anotasi Bahwa ini adalah kelas service
@Service

// Anotasi Transactional untuk menjalankan transaksi JPA,
// sehingga tidak perlu mengaktifkan atau mematikan transaksinya
@Transactional
public class AnggotaServices{

    // inject repositori
    @Autowired
    private AnggotaRepo anggotaRepo;

    // fungsi simpan
    public Anggota save(Anggota anggota) {
        return anggotaRepo.save(anggota);
    }

    // fungsi cari satu data
    public Anggota findOne(Long id) {
        return anggotaRepo.findById(id).get();
    }

    // fungsi baca semua data
    public Iterable<Anggota> findAll() {
        return anggotaRepo.findAll();
    }

    // fungsi cari data berdasarkan pengguna
    public List<Anggota> findByUser(User user){
        return anggotaRepo.findByUser(user);
    }

    // cari data berdasarkan komunitas
    public List<Anggota> findByKomunitas(Komunitas komunitas){
        return anggotaRepo.findByKomunitas(komunitas);
    }

    // fungsi hapus data
    public Boolean deleteById(Long id) {
        if (isExist(id)) {
            anggotaRepo.deleteById(id);
            return true;
        }else{
            return false;
        }
    }

    // fungsi cek apakah data ada di database
    public Boolean isExist(Long id) {
        Anggota anggota = findOne(id);
        if (anggota!=null) {
```

```
        return true;
    }else{
        return false;
    }
}
```

### 3. Kelas Controller

#### Source Code :

```
// Anotasi RESTController untuk menandakan bahwa ini kelas rest controller
@RestController
// Set base url endpoint (baseurl/anggota)
@RequestMapping("/anggota")
public class AnggotaController {

    // Inject anggota service untuk memakai fungsi fungsi yg ada di kelas service
    @Autowired
    private AnggotaServices anggotaServices;

    // Inject komunitas service untuk memakai fungsi fungsi yg ada di kelas service
    @Autowired
    private KomunitasServices komunitasServices;

    // Inject user service untuk memakai fungsi fungsi yg ada di kelas service
    @Autowired
    private UserRepository userRepository;

    // endpoint untuk menyimpan data
    // anotasi untuk menandakan metode yang di gunakan adalah POST
    @PostMapping
    public ResponseEntity<AbstractResponse<Anggota>> create(@Valid @RequestBody AnggotaRequest
    anggotaRequest, Errors errors ) {

        // Siapkan objek kosong untuk di kembalikan
        AbstractResponse<Anggota> responseData = new AbstractResponse<>();

        // periksa jika ada error
        if (errors.hasErrors()) {
            for (ObjectError error : errors.getAllErrors()) {
                responseData.getMessage().add(error.getDefaultMessage());
            }
            responseData.setSuccess(false);
            responseData.setPayload(null);

            // kembalikan AbstractResponse dengan pesan gagal dan kode error 500
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(responseData);
        }

        // jika tidak ada error, maka akan di kembalikan respon 200
        Komunitas komunitas = komunitasServices.findOne(anggotaRequest.getKomunitas().getId());
        User user = userRepository.findOneById(anggotaRequest.getUser().getId());
        Anggota anggota = new Anggota();
        anggota.setKomunitas(komunitas);
        anggota.setUser(user);
        responseData.setSuccess(true);
    }
}
```

```
        responseData.setPayload(anggotaServices.save(anggota));
        return ResponseEntity.ok(responseData);
    }

    // buat endpoint untuk update, penjelasanya sama kaya simpan data hanya saja respon yang di
    // terima sudah ada id objeknya
    // buat metode PUT
    @PutMapping
    public ResponseEntity<AbstractResponse<Anggota>> upstate(@Valid @RequestBody AnggotaRequest
    anggotaRequest, Errors errors ) {

        AbstractResponse<Anggota> responseData = new AbstractResponse<>();

        if (errors.hasErrors()) {
            for (ObjectError error : errors.getAllErrors()) {
                responseData.getMessages().add(error.getDefaultMessage());
            }
            responseData.setSuccess(false);
            responseData.setPayload(null);
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(responseData);
        }

        Komunitas komunitas = komunitasServices.findOne(anggotaRequest.getKomunitas().getId());
        User user = userRepository.findOneById(anggotaRequest.getUser().getId());
        Anggota anggota = new Anggota();
        anggota.setKomunitas(komunitas);
        anggota.setUser(user);
        responseData.setSuccess(true);
        responseData.setPayload(anggotaServices.save(anggota));
        return ResponseEntity.ok(responseData);
    }

    // Endpoitn delete
    // metode DELETE
    // id di baca dari url
    @DeleteMapping("/delete/{id}")
    public ApiResponse deleteById(@PathVariable("id") Long id) {

        // Buat respon gagal
        ApiResponse response = new ApiResponse(false, "Data Gagal Di hapus");
        if (anggotaServices.deleteById(id)) {
            // update respon menjadi berhasil
            response.setSuccess(true);
            response.setMessage("Data Berhasil Di hapus");
        }
        // kembalikan respon
        return response;
    }

    // enpoinnd untuk membaca semua data
    @GetMapping
    public Iterable<Anggota> findAll() {
        return anggotaServices.findAll();
    }
}
```

```
// endpoin untuk mencari data berdasarkan user
@GetMapping("/find/user/{user}")
public List<Anggota> findByUser(@PathVariable("user") User user) {
    return anggotaServices.findByUser(user);
}

// endpoin untuk mencari data berdasarkan komunitas
@GetMapping("/find/komunitas/{komunitas}")
public List<Anggota> findByKomunitas(@PathVariable("komunitas") Komunitas komunitas) {
    return anggotaServices.findByKomunitas(komunitas);
}
}
```

## API Untuk Kolam

### 1. Kelas Respositori

Source Code :

```
public interface KolamRepo extends CrudRepository<Kolam, Long>{
}
```

### 2. Kelas Service

Source Code :

```
// Anotasi Bahwa ini adalah kelas service
@Service
// Anotasi Transactional untuk menjalankan transaksi JPA,
// sehingga tidak perlu mengaktifkan atau mematikan transaksinya
@Transactional
public class KolamServices{

    // inject repositori
    @Autowired
    private KolamRepo kolamRepo;

    // fungsi simpan
    public Kolam save(Kolam kolam) {
        return kolamRepo.save(kolam);
    }

    // fungsi cari satu data
    public Kolam findOne(Long id) {
        return kolamRepo.findById(id).get();
    }

    // fungsi baca semua data
    public Iterable<Kolam> findAll() {
        return kolamRepo.findAll();
    }

    // fungsi hapus data
    public Boolean deleteById(Long id) {
        if (isExist(id)) {
            kolamRepo.deleteById(id);
            return true;
        }
    }
}
```

```
    }else{
        return false;
    }
}

// fungsi cek apakah data ada di database
public Boolean isExist(Long id) {
    Kolam kolam = findOne(id);
    if (kolam!=null) {
        return true;
    }else{
        return false;
    }
}
}
```

### 3. Kelas Controller

Source Code :

```
// Anotasi RESTController untuk menandakan bahwa ini kelas rest controller
@RestController
// Set base url endpoint (baseurl/anggota)
@RequestMapping("/kolam")
public class KolamController {

    // Inject anggota service untuk memakai fungsi fungsi yg ada di kelas service
    @Autowired
    private KolamServices kolamServices;

    // inject model mapper untuk memudahkan penyusunan data dari json yang di kirimkan frontend ke objek
    @Autowired
    private ModelMapper modelMapper;

    // endpoint untuk menyimpan data
    // anotasi untuk menandakan metode yang di gunakan adalah POST
    @PostMapping
    public ResponseEntity<AbstractResponse<Kolam>> create(@Valid @RequestBody KolamRequest
    kolamRequest, Errors errors ) {

        // Siapkan objek kosong untuk di kembalikan
        AbstractResponse<Kolam> responseData = new AbstractResponse<>();

        // periksa jika ada error
        if (errors.hasErrors()) {
            for (ObjectError error : errors.getAllErrors()) {
                responseData.getMessage().add(error.getDefaultMessage());
            }
            responseData.setSuccess(false);
            responseData.setPayload(null);
            // kembalikan AbstractResponse dengan pesan gagal dan kode error 500
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(responseData);
        }

        // jika tidak ada error, maka akan di kembalikan respon 200
        Kolam kolam = modelMapper.map(kolamRequest, Kolam.class);
```

```
        responseData.setSuccess(true);
        responseData.setPayload(kolamServices.save(kolam));
        return ResponseEntity.ok(responseData);
    }

    // buat endpoint untuk update, penjelasanya sama kaya simpan data hanya saja respon yang di
    // terima sudah ada id objeknya
    // buat metode PUT
    @PutMapping
    public ResponseEntity<AbstractResponse<Kolam>> update(@Valid @RequestBody KolamRequest
        kolamRequest, Errors errors ) {

        AbstractResponse<Kolam> responseData = new AbstractResponse<>();

        if (errors.hasErrors()) {
            for (ObjectError error : errors.getAllErrors()) {
                responseData.getMessages().add(error.getDefaultMessage());
            }
            responseData.setSuccess(false);
            responseData.setPayload(null);
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(responseData);
        }

        Kolam kolam = modelMapper.map(kolamRequest, Kolam.class);
        responseData.setSuccess(true);
        responseData.setPayload(kolamServices.save(kolam));
        return ResponseEntity.ok(responseData);
    }

    // enpoint untuk membaca semua data
    @GetMapping
    public Iterable<Kolam> findAll() {
        return kolamServices.findAll();
    }

    // Endpoitn delete
    // metode DELETE
    // id di baca dari url
    @DeleteMapping("/delete/{id}")
    public ApiResponse deleteById(@PathVariable("id") Long id) {
        ApiResponse response = new ApiResponse(false, "Data Gagal Di hapus");
        if (kolamServices.deleteById(id)) {
            response.setSuccess(true);
            response.setMessage("Data Berhasil Di hapus");
        }
        return response;
    }
}
```

## API Untuk jadwal

### 1. Kelas Respositori

**Source Code :**

```
public interface JadwalRepo extends CrudRepository<Jadwal, Long>{  
  
}
```

### 2. Kelas Service

**Source Code :**

```
// Anotasi Bahwa ini adalah kelas service  
@Service  
// Anotasi Transactional untuk menjalankan transaksi JPA,  
// sehingga tidak perlu mengaktifkan atau mematikan transaksinya  
@Transactional  
public class JadwalServices{  
  
    // inject repositori  
    @Autowired  
    private JadwalRepo jadwalRepo;  
  
    // fungsi simpan  
    public Jadwal save(Jadwal jadwal) {  
        return jadwalRepo.save(jadwal);  
    }  
  
    // fungsi cari satu data  
    public Jadwal findOne(Long id) {  
        return jadwalRepo.findById(id).get();  
    }  
  
    // fungsi baca semua data  
    public Iterable<Jadwal> findAll() {  
        return jadwalRepo.findAll();  
    }  
  
    // fungsi hapus data  
    public Boolean deleteById(Long id) {  
        if (isExist(id)) {  
            jadwalRepo.deleteById(id);  
            return true;  
        }else{  
            return false;  
        }  
    }  
  
    // fungsi cek apakah data ada di database  
    public Boolean isExist(Long id) {  
        Jadwal jadwal = findOne(id);  
        if (jadwal!=null) {  
            return true;  
        }else{  
            return false;  
        }  
    }  
}
```



## 3. Kelas Controller

## Source Code :

```
// Anotasi RESTController untuk menandakan bahwa ini kelas rest controller
@RestController
// Set base url endpoint (baseurl/jadwal)
@RequestMapping("/jadwal")
public class JadwalController {

    // Inject komunitas service untuk memakai fungsi fungsi yg ada di kelas service
    @Autowired
    private JadwalServices jadwalServices;

    // inject model mapper untuk memudahkan penyusunan data dari json yang di kirimkan frontend ke objek
    @Autowired
    private ModelMapper modelMapper;

    // endpoint untuk menyimpan data
    // anotasi untuk menandakan metode yang di gunakan adalah POST
    @PostMapping
    public ResponseEntity<AbstractResponse<Jadwal>> create(@Valid @RequestBody JadwalRequest jadwalRequest, Errors errors ) {

        // Siapkan objek kosong untuk di kembalikan
        AbstractResponse<Jadwal> responseData = new AbstractResponse<>();

        // periksa jika ada error
        if (errors.hasErrors()) {
            for (ObjectError error : errors.getAllErrors()) {
                responseData.getMessage().add(error.getDefaultMessage());
            }
            responseData.setSuccess(false);
            responseData.setPayload(null);
            // kembalikan AbstractResponse dengan pesan gagal dan kode error 500
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(responseData);
        }

        // jika tidak ada error, maka akan di kembalikan respon 200
        Jadwal jadwal = modelMapper.map(jadwalRequest, Jadwal.class);
        responseData.setSuccess(true);
        responseData.setPayload(jadwalServices.save(jadwal));
        return ResponseEntity.ok(responseData);
    }

    // buat endpoint untuk update, penjelasanya sama kaya simpan data hanya saja respon yang di terima sudah ada id objeknya
    // buat metode PUT
    @PutMapping
    public ResponseEntity<AbstractResponse<Jadwal>> update(@Valid @RequestBody JadwalRequest jadwalRequest, Errors errors ) {

        AbstractResponse<Jadwal> responseData = new AbstractResponse<>();
    }
}
```

```
        if (errors.hasErrors()) {
            for (ObjectError error : errors.getAllErrors()) {
                responseData.getMessage().add(error.getDefaultMessage());
            }
            responseData.setSuccess(false);
            responseData.setPayload(null);
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(responseData);
        }

        Jadwal jadwal = modelMapper.map(jadwalRequest, Jadwal.class);
        responseData.setSuccess(true);
        responseData.setPayload(jadwalServices.save(jadwal));
        return ResponseEntity.ok(responseData);
    }

    // endpoint untuk membaca semua data
    @GetMapping
    public Iterable<Jadwal> findAll() {
        return jadwalServices.findAll();
    }

    // Endpoint delete
    // metode DELETE
    // id di baca dari url
    @DeleteMapping("/delete/{id}")
    public ApiResponse deleteById(@PathVariable("id") Long id) {
        ApiResponse response = new ApiResponse(false, "Data Gagal Di hapus");
        if (jadwalServices.deleteById(id)) {
            response.setSuccess(true);
            response.setMessage("Data Berhasil Di hapus");
        }
        return response;
    }
}
```