

## **7.1 Deskripsi Singkat**

Dokumentasi API memiliki peran krusial dalam menghubungkan pengembang dengan fungsionalitas yang disediakan oleh suatu aplikasi atau layanan. Salah satu pendekatan yang populer untuk membuat dokumentasi API yang terstruktur dan mudah dimengerti adalah dengan menggunakan OpenAPI dan Swagger UI.

OpenAPI, sebelumnya dikenal sebagai Swagger Specification, adalah standar spesifikasi API yang membantu dalam mendefinisikan struktur, endpoint, parameter, dan respons dari suatu API secara jelas dan terdokumentasi. Menggunakan OpenAPI memungkinkan pengembang untuk membuat API yang konsisten dan memudahkan penggunaan dan integrasi.

Melalui OpenAPI, pengembang dapat memberikan deskripsi yang rinci terkait setiap endpoint API. Ini mencakup informasi seperti metode HTTP yang didukung, parameter yang diperlukan, dan respons yang diharapkan. Dokumentasi ini memungkinkan pengguna API untuk dengan mudah memahami cara berinteraksi dengan layanan tanpa harus merujuk ke sumber kode.

Swagger UI adalah antarmuka pengguna yang berbasis web yang secara otomatis dihasilkan dari spesifikasi OpenAPI. Dengan Swagger UI, dokumentasi API tidak hanya informatif tetapi juga menarik secara visual. Pengguna dapat menjelajahi dan menguji endpoint secara langsung dari antarmuka web ini, membuat proses pengembangan dan debugging lebih efisien.

Swagger UI memungkinkan pengguna untuk menguji endpoint API langsung dari antarmuka pengguna. Dengan menggunakan formulir yang disediakan, pengguna dapat mengirimkan permintaan HTTP dengan parameter yang diperlukan dan melihat respons yang dihasilkan. Ini membuat pengembang dapat dengan cepat memvalidasi fungsionalitas API tanpa perlu menggunakan alat pihak ketiga.

Dengan dokumentasi API yang dibangun menggunakan OpenAPI dan Swagger UI, kolaborasi antara tim pengembang, pengguna, dan pihak terkait menjadi lebih efisien. Semua pihak dapat mengakses dokumentasi yang konsisten, dan pengembang dapat mengintegrasikan umpan balik dari pengguna langsung ke dalam dokumentasi untuk meningkatkan kejelasan dan kegunaan API secara keseluruhan.

## **7.2 Tujuan Praktikum**

Tujuan praktikum ini adalah membuat dokumentasi API dengan Open API dan Swagger UI menggunakan Spring Framework.

Setelah praktikum ini kompetensi yang akan dicapai adalah mahasiswa mampu memahami penerapan dokumentasi web service menggunakan Open API.

## **7.3 Alokasi Waktu**

Alokasi waktu pada praktikum 1 adalah sebagai berikut :

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

## **7.4 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.

## 7.5 Kegiatan Praktikum

Pada kegiatan praktikum ini kita akan memodifikasi proyek perpustakaan pada praktikum 6 dengan menambahkan dokumentasi menggunakan Open API dan Swagger UI. Berikut langkah-langkahnya.

- 1) Buka proyek perpustakaan dan tambahkan dependency Springdoc pada file pom.xml.

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.1.0</version>
</dependency>
```

- 2) Setelah dependency ditambahkan, Springdoc secara default akan menyediakan akses OpenAPI berformat json pada path */v3/api-docs*. Sedangkan OpenAPI berformat Yaml tersedia pada path */v3/api-docs.yaml*. Path tersebut dapat diubah dengan menambahkan pengaturan pada file *application.properties*.

Tambahkan pengaturan sebagai berikut.

```
springdoc.api-docs.path=/docs/open-api
```

- 3) Selain itu, Springdoc juga akan *men-generate* Swagger UI yang secara default dapat diakses pada path */swagger-ui.html*. Path tersebut dapat diubah dengan menambahkan pengaturan pada *application.properties*.

Tambahkan pengaturan sebagai berikut.

```
springdoc.swagger-ui.enabled = true
springdoc.swagger-ui.path = /docs/swagger-ui
springdoc.swagger-ui.tryItOutEnabled = false
springdoc.swagger-ui.filter = false
springdoc.swagger-ui.syntaxHighlight.activated = true
```

- 4) Karena pada proyek sebelumnya, kita menambahkan authentication pada setiap path kecuali /register dan /login, maka agar path Open API dan Swagger UI dapat diakses, tambahkan path /docs/\*\* pada request matcher pada file SecurityConfig.java seperti berikut ini.

```
// ada kode sebelum ini

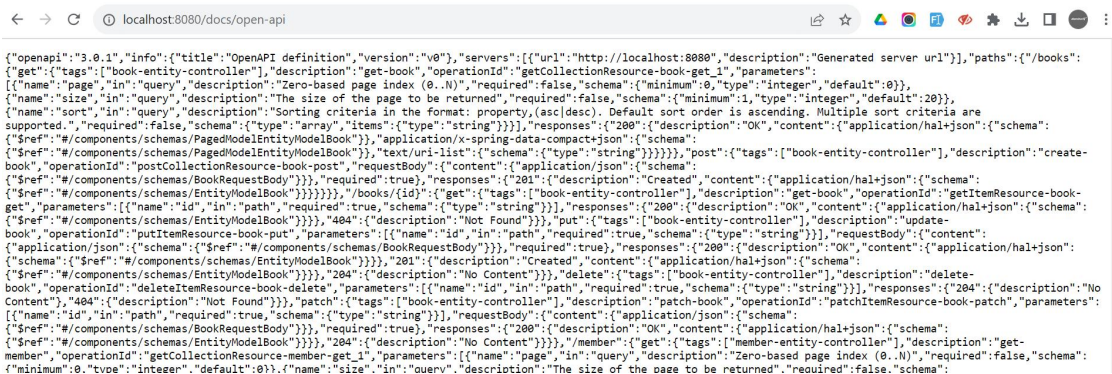
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{
    http.csrf().disable();

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

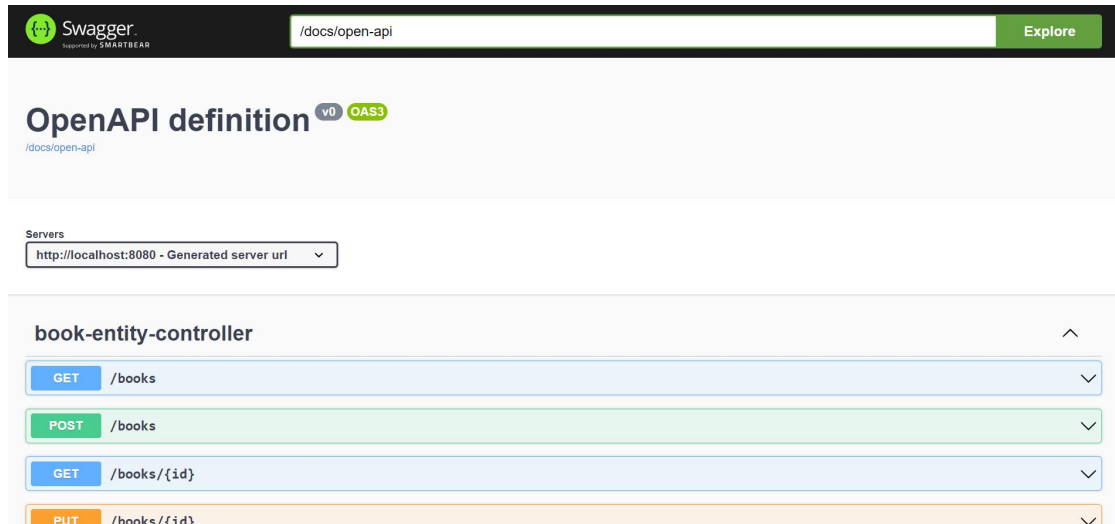
    http.authorizeRequests()
        .requestMatchers("/register", "/login", "/docs/**").permitAll()
        .anyRequest().authenticated();
    http.addFilterBefore(jwtTokenFilter,
        UsernamePasswordAuthenticationFilter.class);
    return http.build();
}

//ada kode setelah ini
```

- 5) Jalankan file PerpustakaanApplication.java.
- 6) Setelah itu, akses OpenAPI pada browser dengan url <http://localhost:8080/docs/open-api>. Berikut contoh tampilannya pada browser.



- 7) Buka Swagger UI pada browser melalui url <http://localhost:8080/docs/swagger-ui>. Berikut contoh tampilannya pada browser.



- 8) Selanjutnya, kita dapat melengkapi dokumentasi API pada kode program. Berikut contohnya pada kelas AuthController.

```
//ada kode program sebelum ini
@Operation(summary = "User login to get access token.")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200",
        description = "Email and access token",
        content = {
            @Content(mediaType = "application/json",
                schema = @Schema(implementation = Page.class))),
    @ApiResponse(responseCode = "401",
        description = "Invalid credentials",
        content = @Content)})
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody @Valid AuthRequest
request) {
    try {
        Authentication authentication = authManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                request.getEmail(), request.getPassword())
        );

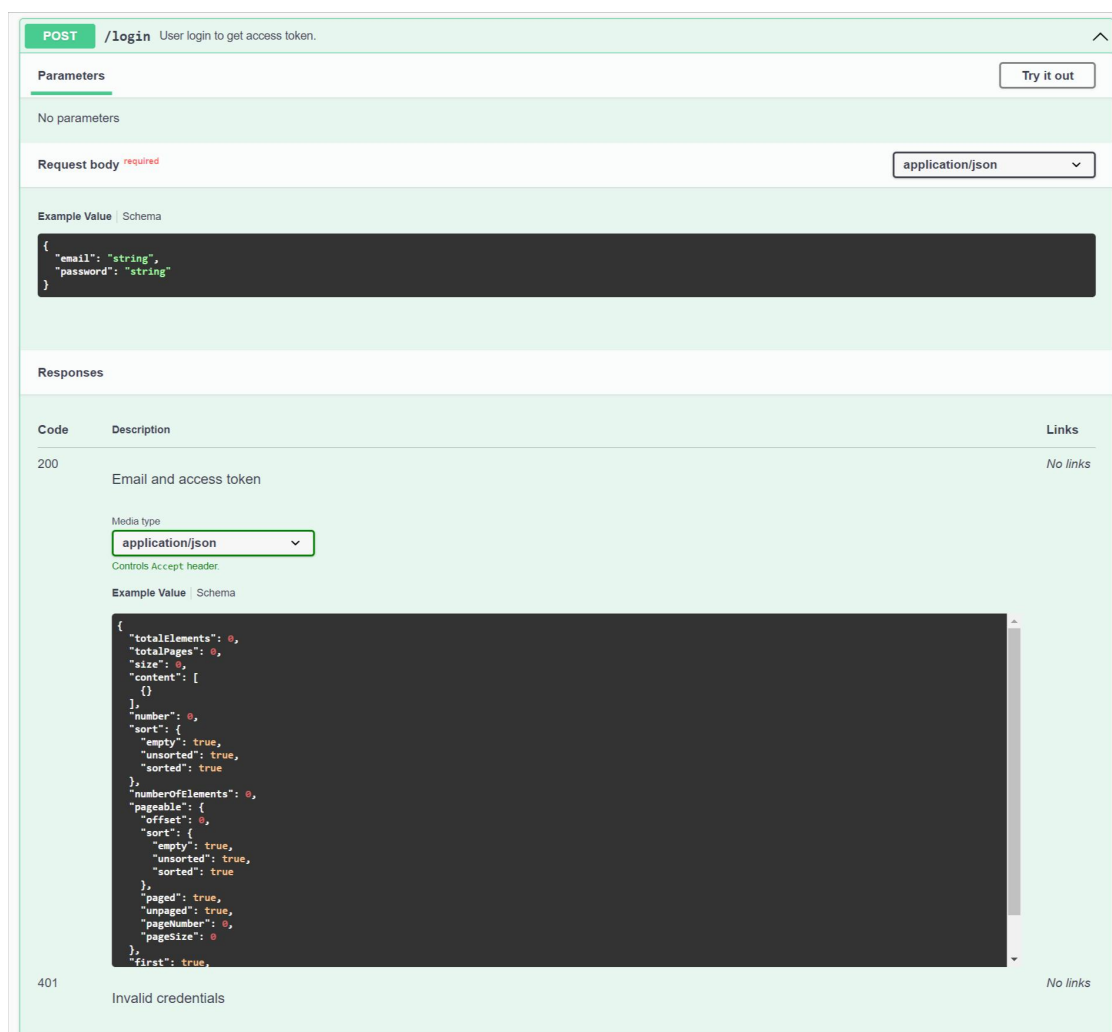
        String accessToken =
        jwtUtil.generateAccessToken(authentication);
        AuthResponse response = new
        AuthResponse(request.getEmail(), accessToken);
        return ResponseEntity.ok().body(response);
    }
}
```

```

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

```

- 9) Jalankan kembali file PerpustakaanApplication.java. Selanjutnya buka halaman Swagger UI dan lihat hasilnya pada bagian /login seperti gambar berikut.



## 7.6 Penugasan

Lengkapi dokumentasi API untuk setiap endpoint pada proyek perpustakaan. Tambahkan deskripsi pada setiap endpoint, parameter,

request body, responses dan bagian lainnya. Silakan pelajari lebih lanjut Springdoc pada halaman berikut <https://springdoc.org/#Introduction>.

Selanjutnya, buat laporan pekerjaan Anda yang menampilkan halaman Swagger UI dari masing-masing endpoint dan anotasi pada kode programnya.