Panduan Spring Boot

Konfigurasi Springboot

1. Versi

- Spring Boot : 2.6.x
- Java : java 11

2. Dependency

- Spring-boot-starter-data-jpa
- Spring-boot-starter-validation
- Spring-boot-starter-web
- Mysql-connector-java / postgresql
- spring-boot-starter-data-jdbc
- Lombok
- Modelmapper
- Jackson-core

3. Api Documentation

- springdoc-openapi-ui

> Untuk dependency setiap project mungkin akan berbeda tergantung dari keperluan dan kegunaan dependency di setiap projectnya.

Struktur Project

Basic struktur project menggunakan format **By Feature**

```
src
 ├── main
 │     ├── java\com\project\example
 │     │     ├── exception
 │     │     ├── feature
 │     │     ├── helper
 │     │     ├── security
 │     │     ├── utils
 │     │     └── ExampleAplication.java
 │     └── resources
 └── test
```

1. Package Exception

Package exception berisikan class java yang berupa kostum exception dan exception handler, sehingga output ketika terjadi runtime error akan selalu menggunakan format data JSON yang selalu sama atributnya.

2. Package Helper

Package Helper berisikan class-class helper yang digunakan untuk membantu programer dalam mengelompokkan suatu method/fungsi yang dapat digunakan secara berulang-ulang seperti model mapper atau mapper List. Package ini juga berisikan class java untuk get data dari other service (RestTemplate API).

3. Package Utils

Package Utils berisikan class yang digunakan untuk menyimpan static data seperti URL other service, path file, dll

4. Package Security

Package Security berisikan class security config dengan menggunakan Spring Security. Selain itu juga berisi class Custom filter pre-request jika menggunakan token JWT untuk Authentication.

5. Package Feature

Di dalam package feature terdapat package untuk setiap entity yang merepresentasikan suatu tabel. Contoh struktur package post di dalam package feature:

```
feature
  ├── post
  |     ├── model
  |     |    ├── CreatePostRequest.java
  |     |    └── PostResponse.java
  |     ├── service
  |     |    ├── PostService.java
  |     |    └── PostServiceImpl.java
  |     ├── Post.java
  |     ├── PostController.java
  |     └── PostRepo.java
  └── ...
```

Di dalam package category terdapat package model dan service, serta class Category.java yang merupakan class entity, CategoryController.java yang berisikan Controller untuk menerima request, dan CategoryRepo.java yang mengimplementasikan JPARepository untuk DAO (Data Access Object) ke basis data.

Di dalam package model terdapat class-class model yang digunakan untuk mapping data dan juga untuk Data Transfer Object (DTO). Package Service terdapat class CategoryService.java yang merupakan suatu class interface dan CategoryServiceImpl.java yang merupakan class yang mengimplementasikan CategoryService.java.

Penulisan Code

1. HTTP Status Code:

- Dokumentasi HTTP Status Code RESTFull API

2. Penamaan resource URL:

- Dokumentasi penamaan Resource
- Contoh sederhana : domain.com/service-name/resource-name/method

3. Penamaan Object:

- Entity/Class/Object Java menggunakan CamelCase
- Nama Variabel menggunakan lowerCamelCase
- Nama fungsi/method menggunakan lowerCamelCase
- Menggunakan Annotation (@) bukan xml untuk konfigurasi

4. Query:

Penggunaan Query bisa menggunakan JPA Query Creation, JPQL, Native Query dan pemanggilan procedure basis data.

- JPA Query Creation

  JPA Query Creation mempermudah dalam generate query untuk mengambil data. Penggunaan nya ketika kita membuat method untuk get data nama method tersebut harus mempresentasikan Query ke basis data yang nanti akan otomatis akan di olah oleh JPA. Dokumentasi JPA Query Creation

```
public interface PostRepo extends JpaRepository<Post, Long> {

    Post findByTitle(String title);
}

//JPA akan generate Query SQL : SELECT * FROM post WHERE title = 'title';
```

- JPQL

  JPQL menggunakan Entity untuk Query ke basis data. Dokumentasi JPQL

```
public interface PostRepo extends JpaRepository<Post, Long> {

    @Query("SELECT p FROM Post p WHERE p.title = :title ")
    Post getPostByTitle(@Param("title") String title);
}

//JPA akan generate Query SQL : SELECT * FROM post WHERE title = 'title';
```

- Native Query

Native query menggunakan query yang sama dengan yang ada di basis data, tidak lagi menggunakan nama class Entity melainkan menggunakan nama table dan nama kolom sesuai di basis data untuk query. Dokumentasi Native Query

```
public interface PostRepo extends JpaRepository<Post, Long> {

    @Query("SELECT p FROM post p WHERE p.title = :title ", nativeQuery = true)
    Post getPostByTitle(@Param("title") String title);
}
```

- Procedure Call Query Dokumentasi Procedure Call

5. Penggunaan DDL-AUTO:

Pembuatan model sebisa mungkin menggunakan fitur DDL Auto yang sudah disediakan di Spring Boot untuk efisiensi dalam mengambil data karena konfigurasi di Entity dan Tabel di basis data sama, terutama di relasi antar tabel.

6. Konfigurasi environtment:

Konfigurasi env menggunakan .properties

Konsep Alur Data

A. Menggunakan konsep Controller - Service - Data

- **Controller**

  Class yang bertanggung jawab untuk memproses permintaan REST API yang masuk, memvalidasi data, menyiapkan model, dan mengembalikan tampilan data sebagai response sesuai permintaan client.

- **Service**

  Class yang digunakan untuk menjalankan proses bisnis yang diperlukan untuk mengolah dan menampilkan data dari DAO ke Controller serta menyimpan atau mengedit data dari REST API yang masuk ke Controller untuk di simpan di basis data.

- **Data (DAO/Repo)**

  Class yang digunakan untuk melakukan koneksi ke Basis Data untuk mendapatkan data yang kemudian disimpan dalam class Entity atau Object Projection lainnya seperti class DTO, Interface atau Object lainnya. yang kemudian akan diolah di Service.

B. Menggunakan konsep Controller - Service - Data

Di dalam class Controller tidak boleh merespon/request data langsung menggunakan class entity karena akan ter expose struktur entity/table di dokumentasi springdoc openapi, sehingga class entity harus di mapping

terlebih dahulu ke dalam DTO. Class entity hanya digunakan untuk perpindahan data dari Service ke repo(DAO) dan sebaliknya.

Untuk setiap response dari server harus di mapping ke dalam suatu model/object Wrapper dan data yang sudah di olah di class Service kemudian disimpan di dalam suatu atribut payload/data. Contoh Wrapper untuk reponse seperti berikut ini :

```
@Data
public class ResponseDto <T> {
    private Boolean status = true;
    private List<String> errorMessage =  new ArrayList<>();
    private String message;
    private T payload;
}
```

C. Menggunakan konsep microservice

Dengan menggunakan microservice diharapkan project dapat berjalan secara mandiri tanpa ketergantungan dengan resource lain, sehingga ketika terjadi masalah di service A, service lain tetap masih bisa berjalan walaupun method yang menggunakan get Other Service di service A tidak dapat berjalan. Selain itu setiap service hanya memiliki akses ke satu basis data, sehingga tidak membebani satu resource untuk basis datanya, karena basis data terdistribusi berdasarkan swrvicenya.

Contoh Pembuatan Entity

A. Penggunaan Entity:

Class Entity digunakan untuk mereprentasikan tabel yang ada di basis data ke dalam java object/model. Hal ini membuat class Entity memiliki jumlah atribut yang sama dengan kolom tabel yang direpresentasikan. Di java Spring Boot terdapat fitur Hibernate DDL Auto untuk generate basis data berdasarkan atribut dan konfigurasi (relasi tabel, primary key, foreign key, dll) entity yang ada di project.

Untuk mempermudah pengembangan aplikasi DDL Auto digunakan sehingga struktur basis data yang digunakan akan selalu sama ketika menggunakan environment/basis data yang berbeda walaupun ada penambahan atau perubahan entity di project tersebut.

B. Konfigurasi Entity:

@Annotation yang di gunakan di setiap Entity

```
@Entity
@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
@JsonIgnoreProperties(value = { "deletedAt", "deleted", "userDelete" })
```

```
@SQLDelete(sql = "UPDATE post SET deleted_at = (NOW()), deleted = true WHERE id =
?")
@FilterDef(name = "deletedFilter", parameters = @ParamDef(name = "isDeleted", type
= "boolean"))
@Filter(name = "deletedFilter", condition = "deleted = :isDeleted")
```

> Akan berbeda untuk kasus kasus pembuatan entity yang memerlukan data khusus

C. Penamaan Entity:

Pastikan penamaan kelas Entity menggunakan format CamelCase yang nantinya akan di generate ke basis dengan Hibernate DDL-Auto data menjadi sebuat tabel dengan format nama tabel snake_case. Contohnya kelas Entity `NamaEntity` akan di generate nama tabel di basis data `nama_entity`.

D. Penamaan Atribut:

Penamaan atribut pastikan menggunakan format lowerCamelCase yang ketika di generate ke basis data menggunakan Hibernate DDL Auto akan berubah menjadi format dengan snake_case. Contohnya atribut createdAt akan di ubah menjadi kolom dengan nama created_at;

> contoh Entity Post:

```
//import

@Entity
@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
@JsonIgnoreProperties(value = { "deletedAt", "deleted", "userDelete" })
@SQLDelete(sql = "UPDATE post SET deleted_at = (NOW()), deleted = true WHERE id =
?")
@FilterDef(name = "deletedFilter", parameters = @ParamDef(name = "isDeleted", type
= "boolean"))
@Filter(name = "deletedFilter", condition = "deleted = :isDeleted")
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull
    private String title;

    @Lob
    @NotNull
    private String content;

    @NotNull
    @org.hibernate.annotations.Type(type="pg-uuid")
    @Column(unique=true)
    private UUID publicId;
```

```
    @ManyToMany(cascade = { CascadeType.PERSIST, CascadeType.MERGE })
    @JoinTable(name = "post_categories", joinColumns = @JoinColumn(name =
"post_id", referencedColumnName = "id"),
                inverseJoinColumns = @JoinColumn(name = "category_id",
referencedColumnName = "id"))
    @OrderBy("createdAt ASC")
    private Set<Category> categories;

    @NotNull
    @org.hibernate.annotations.Type(type="pg-uuid")
    private UUID userId;

    @CreatedDate
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    @Schema(hidden = true)
    @Column(updatable = false)
    private LocalDateTime createdAt;

    @NotNull(message = "deleted cannot be null")
    private Boolean deleted = false;

    @org.hibernate.annotations.Type(type="pg-uuid")
    private UUID userDeleteId;

    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    @Schema(hidden = true)
    private LocalDateTime deletedAt;

    @LastModifiedDate
    @Schema(hidden = true)
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    private LocalDateTime updatedAt;
}
```

Dari contoh di atas class Entity `Post` miliki atribut id, title, content, categories, userId, createdAt, deleted, userDeleteId, deletedAt, dan updatedAt. UserId yang disimpan dalam bentuk UUID karena Entity User ada di project lain (project Auth) yang memiliki basis data yang berbeda sehingga tidak bisa disimpan dalam bentuk Entity. Atribut userId, createdAt, deleted, userDeleteId, deletedAt, dan updatedAt adalah atribut yang selalu ada di setiap Entity untuk menyimpan record data nya. Delete Query tidak benar-benar menghapus record tapi menambahkan penanda deleted (Soft Delete) sehingga record data yang di delete masih tetap tersimpan. Deleted dan deletedAt digunakan untuk memfilter Query menggunakan anotasi @FilterDef dan @Filter Spring Boot sehingga data yang ditampilkan hanya data yang tidak di hapus jika status deleted nya false, begitu juga sebaliknya.

```
@FilterDef(name = "deletedFilter", parameters = @ParamDef(name = "isDeleted", type
= "boolean"))
@Filter(name = "deletedFilter", condition = "deleted = :isDeleted")
```

Controller-Service-Repo

Ilustrasi alur ringkas get data Post dari Controller - Service - Repo

A. Controller:

```
//PostController.java
//import
@RestController
@RequestMapping("/blog/post")
@RequiredArgsConstructor
public class PostController {
    @Autowired
    private PostService postService;

    @GetMapping("/title/")
    public ResponseEntity<ResponseDto<PostResponse>>
getPostByTitle(@PathVariable("title") String title) {
        return postService.getPostByTitle(title);
    }

}

//ResponseDto.java
//import
@Data
public class ResponseDto <T> {
    private Boolean status = true;
    private List<String> errorMessage =  new ArrayList<>();
    private String message;
    private T payload;
}
```

B. Service:

```
//PostServiceImpl.java
//import
@Service
public class PostServiceImpl implements PostService {

    @Autowired
    private PostRepo postRepo;
    @Autowired
    private ModelMapper modelMapper;
    @Autowired
    private SessionFilter sessionFilter;

    @Override
     public ResponseEntity<ResponseDto<PostResponse>> getPostByTitle(String title)
{
```

```
        ResponseDto<PostResponse> responseDto = new ResponseDto<>();
        sessionFilter.openSessionFilterDeleted(false);
        Optional<Post> post = postRepo.findByTitle(title);
        sessionFilter.closeSessionFilterDeleted();
        if(post.isEmpty()){
            responseDto.setStatus(false);
            responseDto.setMessage("Data post dengan title "+ title +" tidak
ditemukan");
            responseDto.getErrorMessage().add("Post tidak ditemukan di basis
data");
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(responseDto);
        }
        PostResponse postResponse = modelMapper.map(post.get(),
PostResponse.class);
        responseDto.setStatus(true);
        responseDto.setMessage("Data post dengan title "+ title +" ditemukan");
        responseDto.setErrorMessage(new ArrayList<>());
        responseDto.setPayload(postResponse);
        return ResponseEntity.ok().body(responseDto);
    }

}

//PostResponse.java
//import
@Data
public class PostResponse {
    private String title;
    private String content;
    private Set<CategoryResponse> categories;
    private UUID userId;
}

//CategoryReponse.java
//import
@Data
public class CategoryResponse {
    private String nama;
    private String kode;
    private UUID publicId;
}

//SessionFilter.java
//import
@Service
public class SessionFilter {
    @Autowired
    private EntityManager entityManager;

    public void openSessionFilterDeleted(Boolean deleted) {
        Session session = entityManager.unwrap(Session.class);
        Filter filterDeleted = session.enableFilter("deletedFilter");
        filterDeleted.setParameter("isDeleted", deleted);
    }
```

```
    public void closeSessionFilterDeleted() {
        Session session = entityManager.unwrap(Session.class);
        session.disableFilter("deletedFilter");
    }
}
```

C. Repo (DAO):

```
//PostRepo.java
//import
public interface PostRepo extends JpaRepository<Post, Long> {

    //menggunakan JPA Query Creation
    Optional<Post> findByTitle(String title);

}
```

Git-Flow Repository

references