BINAR

# Spring Data JPA (Part 2)

**Gold** - Chapter 4 - Topic 7

BINAR

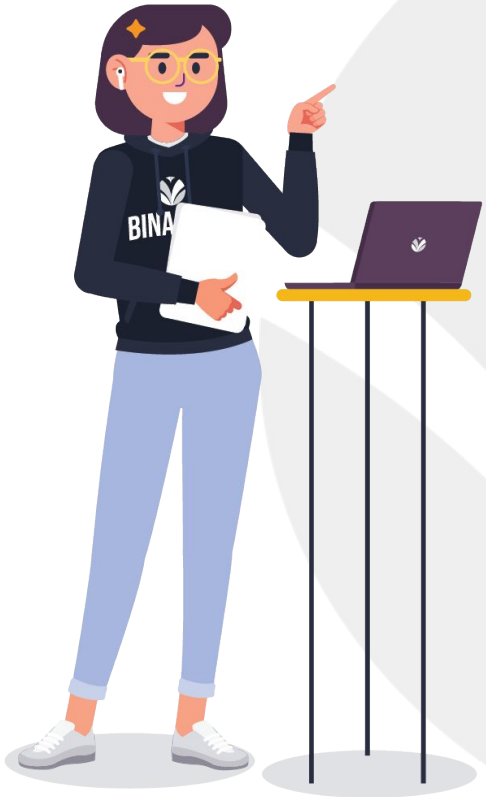
# Selamat datang di Chapter 4 Topic 7 online course Back End Java dari Binar Academy!

## Binarian, kita sampai di penghujung Chapter! 

Binarian, nggak kerasa kita udah belajar tentang konsep Spring Data JPA sejauh ini. Pada topic terakhir kita bakal tuntasin belajar tentang **Spring Data JPA Part 2.**

Mulai dari cara melakukan query dengan Spring Data JPA yang terdiri dari DTO Projections, sampai cara menggunakan Store Procedure dengan Spring JPA.

Yuk, langsung aja kita kepoin~



## Dari sesi ini, kita bakal bahas hal-hal berikut:

- Cara melakukan query dengan DTO Projections
- Cara melakukan query dengan Entity Projections
- Cara melakukan query dengan Dynamic Projections
- Query with Prepared Statement
- Business logic layer with @Service annotation and CRUD
- Bagaimana cara menggunakan Pagination
- Memanggil Store Procedure dengan Spring JPA

Masih tentang Spring Data JPA, kita bakal melanjutkan pembahasan, yaitu mengenal istilah **DTO Projection.**

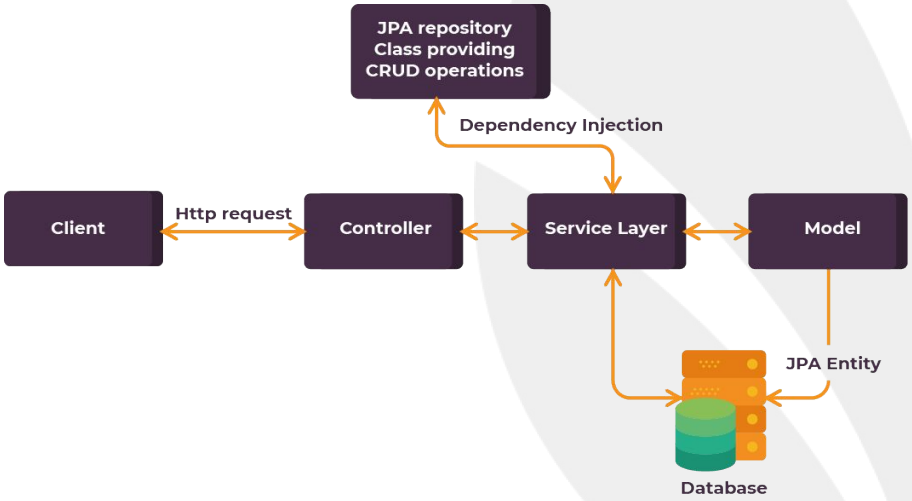
DTO sendiri merupakan akronim dari Data Transfer Object.

Hmmm.. kira-kira transfer gimana ya konsepnya?

## Sesuai namanya, berikut adalah arsitektur dari DTO yang bakal kita bikin di course ini~

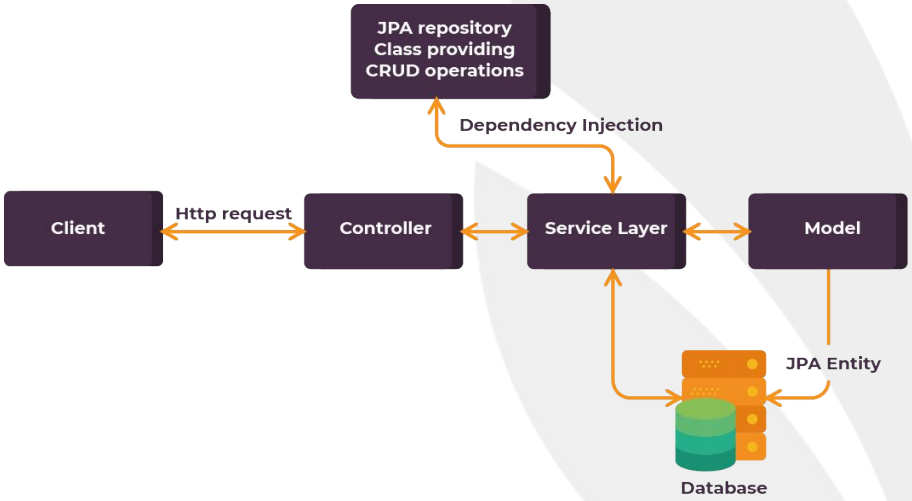
Jangan lupa perhatikan gambar yang di samping, ya!

- **Controller** merupakan layer yang bakal menghandle Http Request.
- **Service Layer** berisi business logic untuk mengolah request yang berasal dari sebuah controller.

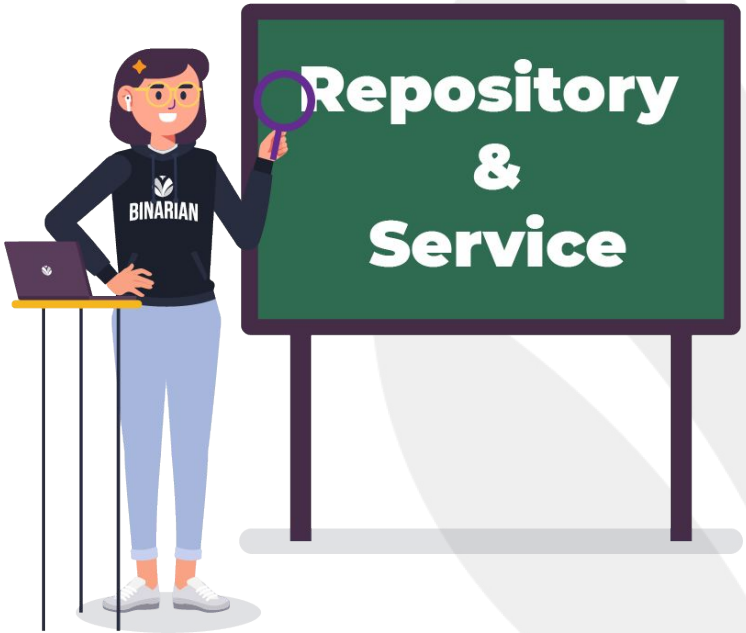


- **Model** merupakan class entity representasi dari table
- **Repository atau DAO (Data Access Object)** merupakan penghubung yang bakal melakukan query.

  Layer ini bakal jadi dependency bagi service layer.



## Pada topic ini ada dua layer yang bakal dibahas, yaitu Repository dan Service




Kita bahas yang Repository dulu, yaaa~

**Repository dipakai untuk melakukan berbagai aktivitas CRUD di database.**

Supaya pemahaman kita makin lengkap, selain bahas query di repository dan service, kita juga bakal bahas query tanpa menggunakan Spring Data JPA, yaitu dengan prepared statement.

## “Terus cara bikin Repository itu gimana?”

Untuk membuat si Repository layer ini, cukup simple kok.

Pertama, kita harus bikin class entity dan menentukan primary key dari class entity tersebut.

Terus, kita bikin suatu interface dengan meng-extend JPARepository kayak contoh di samping~

```
public interface UserRepository extends JpaRepository<User, Integer> {

}
```

Pada contoh si UserRepository merupakan repository dari class User yang punya primary key dengan tipe data integer.

Tanpa harus menuliskan implementasi, kita bisa pakai method yang disediakan oleh JPARepository.

Kira-kira method apa nih yang bisa dipakai? Coba simak slide selanjutnya, ya!

```
public interface UserRepository extends JpaRepository<User, Integer> {

}
```

Beberapa method yang bisa digunakan untuk query tanpa harus menuliskan yang baru, yaitu:

- **findById()**

  Perintah yang bakal mengembalikan optional dari object entity. Parameter dari findById punya tipe data sesuai sama tipe data primary key pada entity.

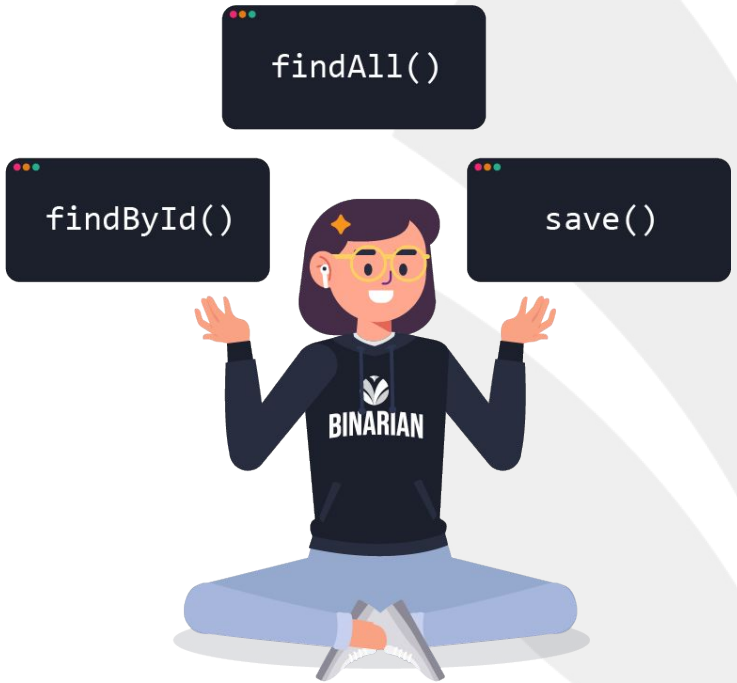
- **findAll()**

  Mengambil semua row yang ada di database.

- **save()**

  Menyimpan object entity agar bisa ter-persist di dalam database.

Dan masih banyak lagi~



## “Terus apakah query-nya bisa di customize?”

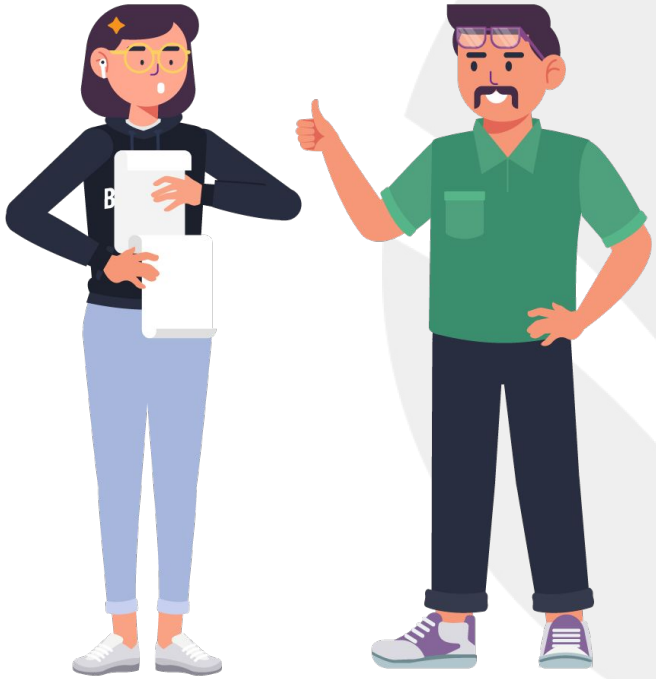
Tentu aja, bisa dong~

Caranya adalah dengan menuliskan method pada interface yang udah dibikin.

Method untuk melakukan customize bisa pakai Query.

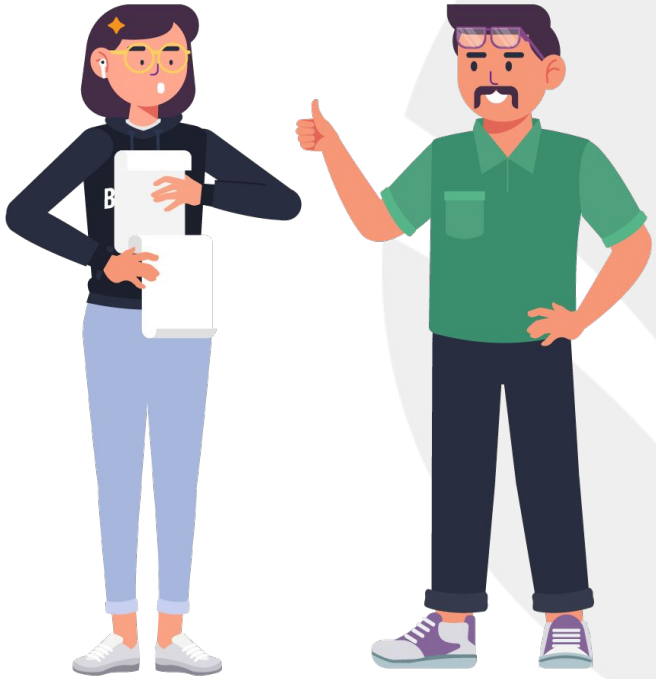
Yaitu query di method yang telah dimodifikasi sesuai dengan ketetapan oleh Spring Data JPA.

Selain itu  bisa juga pakai **JPQL** dan **Native Query**.



Berhubung bisa pakai dua cara, kira-kira bedanya JPLQ dan native Query itu apa sih?

Perbedaan JPQL dan Native Query, yaitu **JPQL mengacu pada nama entity di Java**, sedangkan **Native Query pakai query sesuai relational database**.



Berikut adalah contoh kalau pakai query yang customized~

```
public interface UserRepository extends JpaRepository<User, Integer> {
      @Query("select u from User u where u.firstname like %?1")
      List<User> findByFirstnameEndsWith(String firstname);
      List<Person> findByLastname(String lastname);

}
```

## Coba tebak deh, pada contoh tersebut kira-kira ada berapa query yang di-customize?

Jawabannya, ada dua query yang di-customize, yaitu:

- findByFirstnameEndsWith yang dicari pakai JPQL.
- Query yang udah disesuaikan sama Spring Data JPA.

Oh iya, dari Query itu kalau mau mengunakan fitur ini jangan lupa untuk menyesuaikan nama dari method-nya, ya.

Fitur ini disebut juga dengan Query Methods dan Method findByLastname cuma bisa bekerja kalau Entity User punya field LastName.

Dokumentasi query Spring Data JPA dan rulesnya, secara lengkap tersedia pada link berikut, yaaaa~

[Spring Data JPA - Query Method]

## DTO projection bisa dilakukan kalau data yang mau diambil cuma sebagian aja, gengs~

Selain itu, DTO juga perlu dibikin dengan jumlah field yang sesuai sama jumlah query yang dieksekusi.

Berikut adalah contoh dari DTO Projection~

```
public interface AuthorRepository extends CrudRepository<Author, Long> {

    List<AuthorSummaryDTO> findByFirstName(String firstName);
}
```


Gimanaaa? DTO sampai ke Query Methods udah aman belum?

Supaya makin clear, selanjutnya kita bakal membahas cara melakukan query dengan **Entity Projection.**

## Dalam Entity Projection, query dibuat dengan tipe data return yang sama kayak class entity

Disini **Entity Projection menggunakan tipe data return yang sama dengan class entity ketika membuat query-nya**.

Oleh sebab itu, operasi ini mirip banget kayak select *

Disamping ini adalah contohnya!

```
@Repository
public interface AuthorRepository extends CrudRepository<Author, Long> {
    @Query("select a from Author a left join fetch a.books")
    List<Author> getAuthorsAndBook();
}
```

Sipp deh!! Entity Projection udah beres. Kita lanjut ke cara yang ketiga yaitu **Dynamic Projection.**

Penasaran, kan? Yuk kita pelajari!

## Projection yang terakhir yaitu Dynamic projection~

Kalau pakai projection ini, maka tipe return bakal dibikin lebih generic.

Penjelasan tentang ketiga projection tadi bisa dilihat pada referensi berikut, ya.

[Spring Data JPA - Query Projections]

```
@Repository
public interface AuthorRepository extends CrudRepository<Author, Long> {
    <T> T findByLastName(String lastName, Class<T> type);
}
```

DTO Projection, udah ✔

Entity Projection, beres □□

Dynamic Projection, aman □□

Sekarang waktunya kita lanjut ke materi Query with Prepared Statement.

Berangkatt~

## “Emang gimana caranya melakukan query pakai prepared statement?”

Kadang kita mau menghubungkan database tanpa perlu melakukan mapping pada table, kan? Berarti, kita cuma perlu kebutuhan praktis untuk akses database.

Kalau nggak pakai ORM buat pakai database, kita bisa pakai PreparedStatement, nih.

Berbeda dengan Spring Data JPA, kalau pakai prepared statement, kita cuma mengeksekusi query-nya aja tanpa harus melakukan mapping.

Kalau pakai Prepared statement, koneksi ke database berlangsung hanya untuk sementara aja. Agak beda sama **Spring Data JPA yang punya pool size, bahkan selalu aktif di saat idle**.

Oh iya, ada rekomendasi. Untuk menggunakan PrepareStatement, sebaiknya kita menggunakan **try with resource**.

Pada subtopic pertama, kita udah bahas layar Repository.

Sekarang, kita bakal bahas layer kedua, yaitu Service.

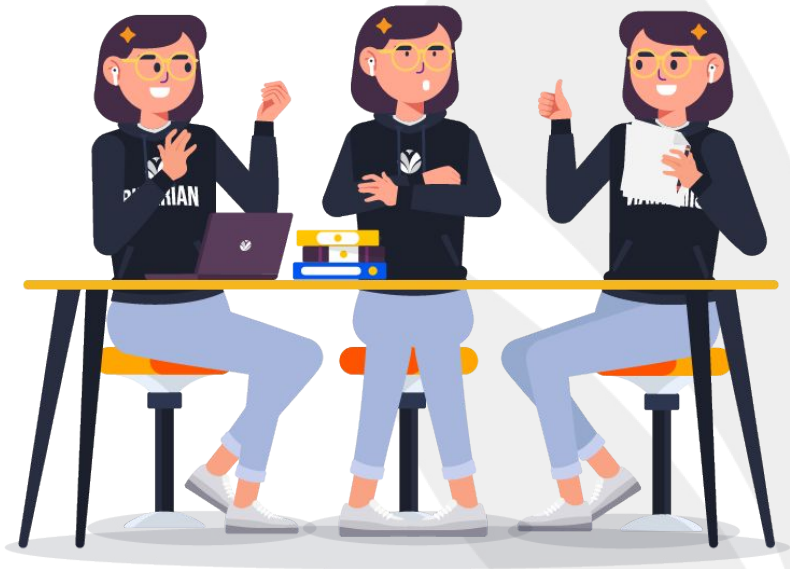
Jadi, selamat datang di materi **Business logic layer with @Service annotation and CRUD~**

## Service layer itu apa, ya?

Service layer merupakan **layer yang isinya business logic dari aplikasi**.

Di class yang berisi implementasi business logic, itu tuh menggunakan annotation **@Service**.

Best practice dari pembuatan layer ini adalah dengan **façade pattern**. Tapi, tenang aja, Façade pattern bakal dijelaskan di topic design pattern, kok~



## Kira-kira gimana ya cara pakai Layer Service?

Caranya adalah dengan membuat **interface yang isinya method-method utama bersifat public**.

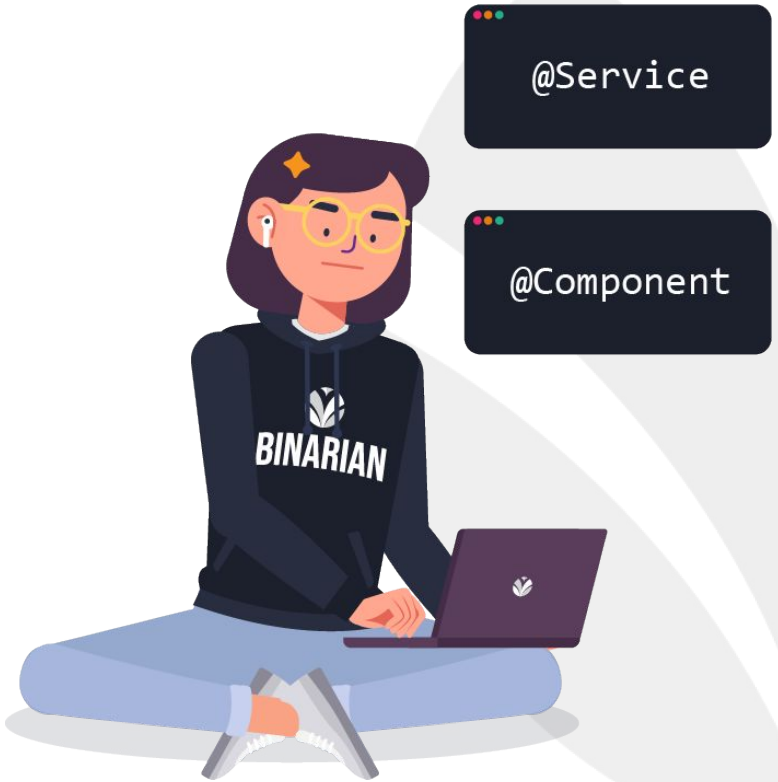
Penamaan dari interface ini adalah dengan nama domain yang ditambah dengan service.

Misalnya, karena servicenya berisi business logic dari User, maka nama interface yang dibuat adalah UserService.

Terus, kita harus bikin implementasi dari interface yang udah dibuat. Yaitu dengan meng-implement interface tersebut.

Pada class implementasi inilah kita pakai annotation **@Service** atau **@Component** pada classnya.

Di dalam class ini, kita juga bisa meng-inject dependency yang diperlukan, kayak service layer lain atau repository layer.

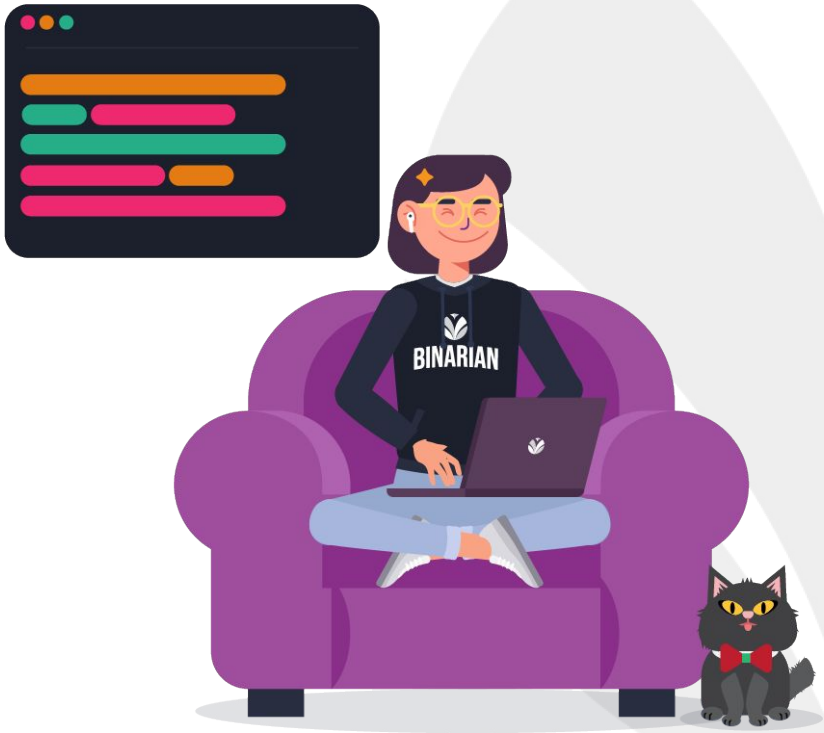


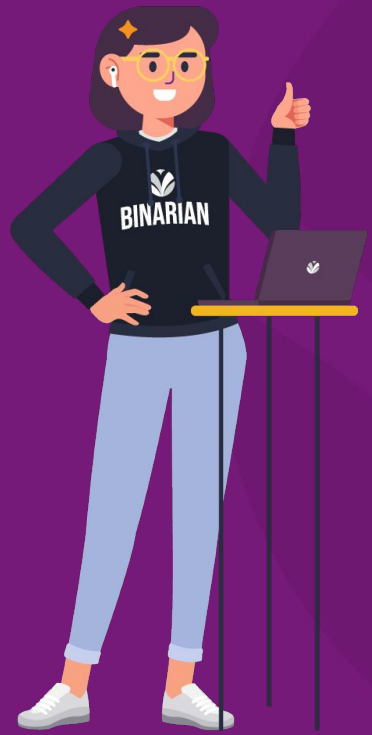
Untuk penamaannya bebas, kok. Di mana kita bisa memberi nama class ini dengan UserServiceFacade atau UserServiceImpl.

Semua service layer sebaiknya diletakkan di package yang sama, yaitu **package service**.

Untuk melihat contoh dari implementasi penggunaan service layer, kita bisa pakai referensi berikut:

**Spring Boot REST API CRUD Example With Spring Data JPA**






**Gimana? gimana? materi service layer-nya mudah diikutin, nggak?**

**Sambil memahami keterkaitan setiap topik, kita coba lanjut ke Pagination, yuk.**


## Kita mulai dari konsep Pagination dulu, yaaa

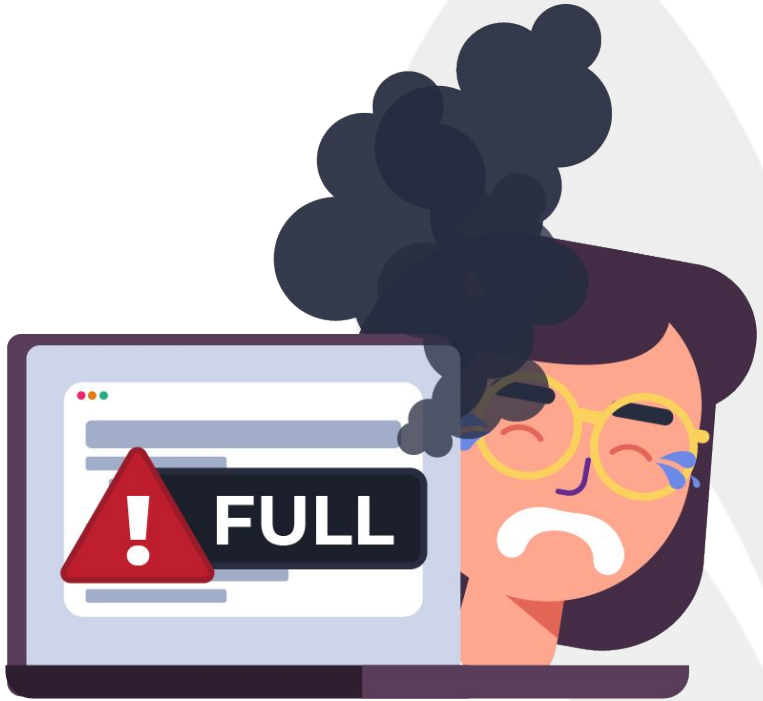
Pagination adalah **proses untuk membagi konten-konten ke dalam beberapa halaman**.

Mirip kayak pas kamu buka Google dan tampilan layarnya kayak di samping. Semua kontennya dibagi-bagi, sehingga kamu bisa ke halaman sebelum atau selanjutnya

**Dengan pagination, data yang ditarik nggak bakal terlalu besar karena udah dibatasi.** Kalo semuanya ditampilkan, udah keburu pegel scrollnya. Capek deh~

Bayangin, kalau ratusan juta data langsung ditarik dan di-load tanpa pagination. Kira-kira bakal gimana?

Selain prosesnya yang bakal lama banget, aplikasi juga bisa terkena OutOfMemory, yaitu heap sizenya nggak mencukupi. Haduhh~



## Untuk melakukan pagination, Spring data JPA udah menyediakan Pageable, lho~

Kalau pakai **Pageable kita bisa menentukan parameter query.**

Misalnya kayak banyaknya data pada satu halaman, halaman, atau sorting berdasarkan apa. Disini, Query yang dikembalikan bisa berbentuk suatu Collection.

Satu lagi, nih. Berikut adalah referensi pagination pada Spring Boot kalau pakai Spring Data JPA.

[Spring Boot JPA Pagination]

## Contoh source code untuk Pagination

Penerapan si pagination menggunakan JPA repository pertama ada pada method findAllByName.

Berikut adalah contoh pagination dalam satu halaman, untuk menampilkan data customer berdasarkan name dengan pagination.

```
@Repository
public interface
CustomerRepository extends
JpaRepository<Customer, String> {

    List<Customer>
findAllByName(String name,
Pageable pageable);
}
```

## Contoh source code untuk Pagination

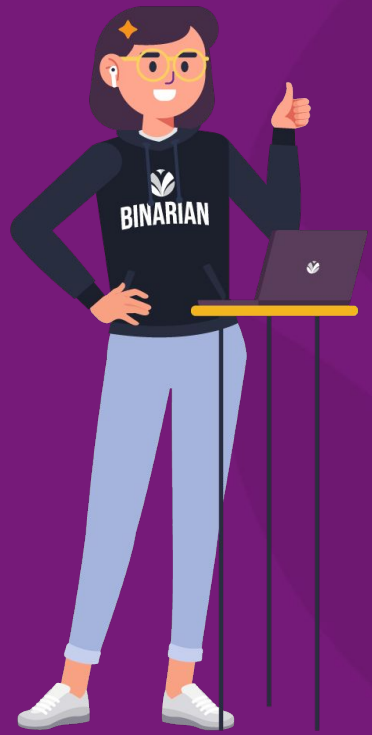
Contoh kedua yaitu menggunakan PageRequest untuk pagination custom di mana kita bisa mengatur jumlah data yang ditampilkan tiap page.

Contoh ini menampilkan nama tergantung nama yang mau di display (contohnya ini 20 nama setiap page-nya)

```
Page<Customer> customers =
customerRepository.findAll(PageReq
uest.of(0, 20));
```

Setelah selesai dengan Pagination, sekarang kita lanjut ke materi penutup topik ini.

Ada materi Store Procedure, yang lagi nungguin kamu. Yuk! Yuk! Yuk!



Ngomongin Store Procedure, sebenernya kita udah sempet mention ini di topik 4. Yuk kita coba recall bareng-bareng.

Store procedure merupakan sekelompok pernyataan SQL yang telah ditentukan sebelumnya dan tersimpan pada database.

Berhubung di Java ada variasi cara untuk mengakses Store procedure. Gimana kalau kita coba fokus di satu cara aja? Kita belajar bagaimana mengakses Store Procedure dari Spring Data JPA Repository.

## Project Setup

Cara mengakses Store Procedure dari Spring Data JPA Repository yang pertama adalah dengan membuat project setup-nya dulu. Kamu bisa cek yang ada di bawah ini □□

pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
```

Lanjut, nih. Setelah setup beres kita harus membuat entity class-nya. Iya, kayak contoh ini.

```java
@Entity
public class Car {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column
    private long id;

    @Column
    private String model;

    @Column
    private Integer year;

   // standard getters and setters
}
```

Setelah dibuat Entity Class-nya maka akan muncul Store Procedure yang menampilkan data dari cars. Di bawah ini ada dua macam contoh tampilannya.

```
CREATE PROCEDURE FIND_CARS_AFTER_YEAR(IN year_in INT)
BEGIN
    SELECT * FROM car WHERE year >= year_in ORDER BY year;
END
```

```
CREATE PROCEDURE GET_TOTAL_CARS_BY_MODEL(IN model_in VARCHAR(50), OUT count_out INT)
BEGIN
    SELECT COUNT(*) into count_out from car WHERE model = model_in;
END
```

Sipp deh! Setelah tampilan Store Procedure muncul disini kamu perlu syntax untuk memanggil store procedure yang udah dibuat sebelumnya. Total ada 3 cara. Pada slide ini merupakan cara yang pertama.

```
@Procedure
int GET_TOTAL_CARS_BY_MODEL(String model);
```

```
@Procedure("GET_TOTAL_CARS_BY_MODEL")
int getTotalCarsByModel(String model);
```

```
@Procedure(procedureName = "GET_TOTAL_CARS_BY_MODEL")
int getTotalCarsByModelProcedureName(String model);
```

```
@Procedure(value = "GET_TOTAL_CARS_BY_MODEL")
int getTotalCarsByModelValue(String model);
```

Nah, kalau slide ini merupakan cara kedua memanggil Store Procedure.

```java
@Entity
@NamedStoredProcedureQuery(name = "Car.getTotalCardsbyModelEntity",
  procedureName = "GET_TOTAL_CARS_BY_MODEL", parameters = {
    @StoredProcedureParameter(mode = ParameterMode.IN, name = "model_in", type = String.class),
    @StoredProcedureParameter(mode = ParameterMode.OUT, name = "count_out", type = Integer.class)})
public class Car {
    // class definition
}
```

```java
@Procedure(name = "Car.getTotalCardsbyModelEntity")
int getTotalCarsByModelEntiy(@Param("model_in") String model);
```

Lastly, d isini ada cara ketiga memanggil Store Procedure.

```
@Query(value = "CALL FIND_CARS_AFTER_YEAR(:year_in);", nativeQuery = true)
List<Car> findCarsAfterYear(@Param("year_in") Integer year_in);
```

## Misi terakhir di topic 7~

Binarian, materi kita udah selesai. Sebelum lanjut ke chapter selanjutnya ada latihan buat kamu.

Silakan **latihan membuat pagination dan CRUD**.

Latihan ini dilakukan di kelas dan jangan lupa untuk mendiskusikan hasil jawabannya bersama teman sekelas dan facilitator, ya.

Selamat mencoba~

Yay, materi Spring Data JPAnya udah rampung dikupas tuntas nih! 

Karena tadi bahas DTO projection, kenapa ya DTO projection itu cuma bisa dilakukan kalau data yang mau diambil cuma sebagian aja?

Nah, selesai sudah pembahasan kita di Chapter 4 ini. Kamu kereeeen!

Selanjutnya, kita bakal siap-siap untuk belajar pada Chapter 5 Topik 1 yang ngomongin tentang Spring Web.

Sampai jumpa pada chapter selanjutnya~