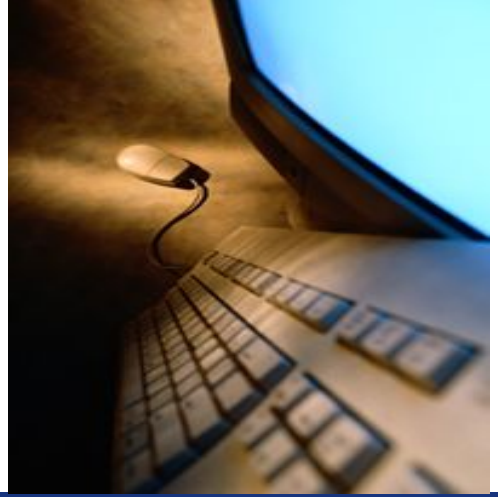
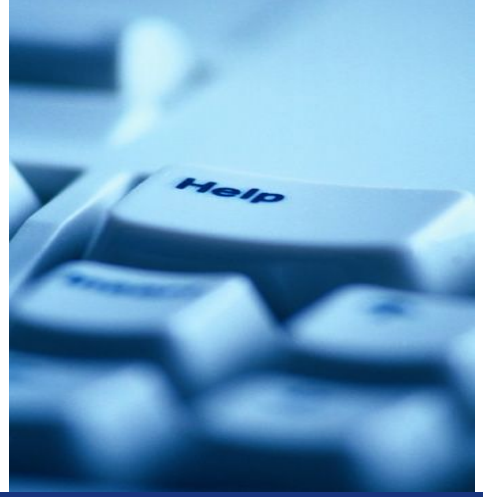
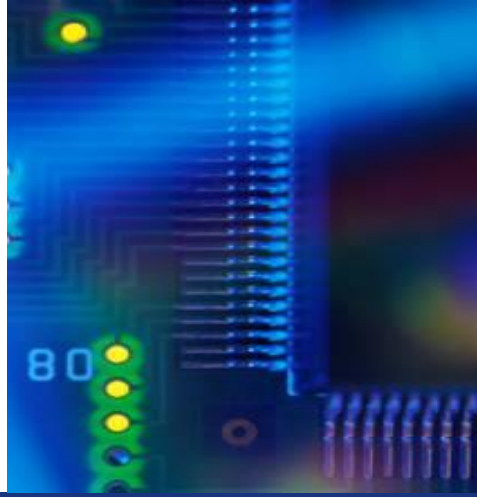
UNIVERSITAS
BUDI LUHUR







FAKULTAS TEKNOLOGI INFORMASI

Java Web Programming

[PG119/ 3 SKS]

Riskiana Wulan, MKom

Pertemuan 11

Connect with Database with Spring Data JPA

Spring Data JPA

Spring Data

- Turunan dari Spring framework
- Memudahkan kita untuk melakukan akses ke database operation, query read, create, update, delete
- Support transactional. Apa itu transactional? Jika gagal, data akan langsung di rollback
- Tidak perlu melakukan buka tutup koneksi ke database secara manual (membuat coding java), namun semua sudah di handle oleh Spring
- Minim Query. Kita hanya perlu mendefinisikan method dengan nama variable pada class model

Tambahkan **spring-boot-starter-data-jpa** pada gradle dependencies

implementation 'org.springframework.boot:spring-boot-starter-data-jpa:3.4.0'

Mysql

Gradle dependency :

implementation 'com.mysql:mysql-connector-j:9.1.0'

```
dependencies {
        implementation 'org.springframework.boot:spring-boot-starter-web'
        implementation 'org.springframework.boot:spring-boot-starter-data-jpa:3.4.0'

        implementation 'com.mysql:mysql-connector-j:9.1.0'

```


Koneksi ke MYSQL Database

- Konfigurasi database dilakukan melalui application.properties yang terletak pada /src/main/resources

- Dengan Spring boot, kita hanya perlu menyediakan informasi terkait dengan koneksi ke database tanpa harus melakukan coding dengan Java untuk mengkoneksikan database kedalam web application. Semua sudah disediakan oleh Spring.

- Perhatikan Spring documentation untuk mengetahui konfigurasi apa saja yang dapat kita pakai https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html

Tambahkan baris berikut pada application.properties

```
#Database connection
spring.jpa.hibernate.ddl-auto=none
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/dbname
spring.datasource.username=databaseuser
spring.datasource.password=databasepassord
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#spring.jpa.show-sql: true
```


Notes:

Spring.jpa.hibernate.ddl-auto adalah property dari hibernate schema autogeneration dengan beberapa value type seperti

- none : tidak ada perubahan yang dilakukan pada database structure
- update : Hibernate melakukan update database structure sesuai dengan struktur entity yang didefinisikan
- create : Melakukan create database setiap kali tetapi tidak melakukan drop database ketika close
- create-drop: Melakukan create database dan melakukan drop database ketikan session factory di close

https://spring.io/guides/gs/accessing-data-mysql/

Pada kelas kali ini kita menggunakan

```
Spring.jpa.hibernate.ddl-auto = none
```

Karena database structure akan kita trigger dengan mengeksekusi sql script

Contoh Koneksi ke database lain

Postgres

Gradle dependency:

implementation 'org.postgresql:postgresql:42.7.4'

application.properties

```
spring.datasource.url= jdbc:postgresql://localhost:5432/testdb
spring.datasource.username= postgres
spring.datasource.password= 123

spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation= true
spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.PostgreSQLDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto= update
```

Oracle:

Gradle dependency:

implementation 'com.oracle.database.jdbc:ojdbc11:23.6.0.24.10'

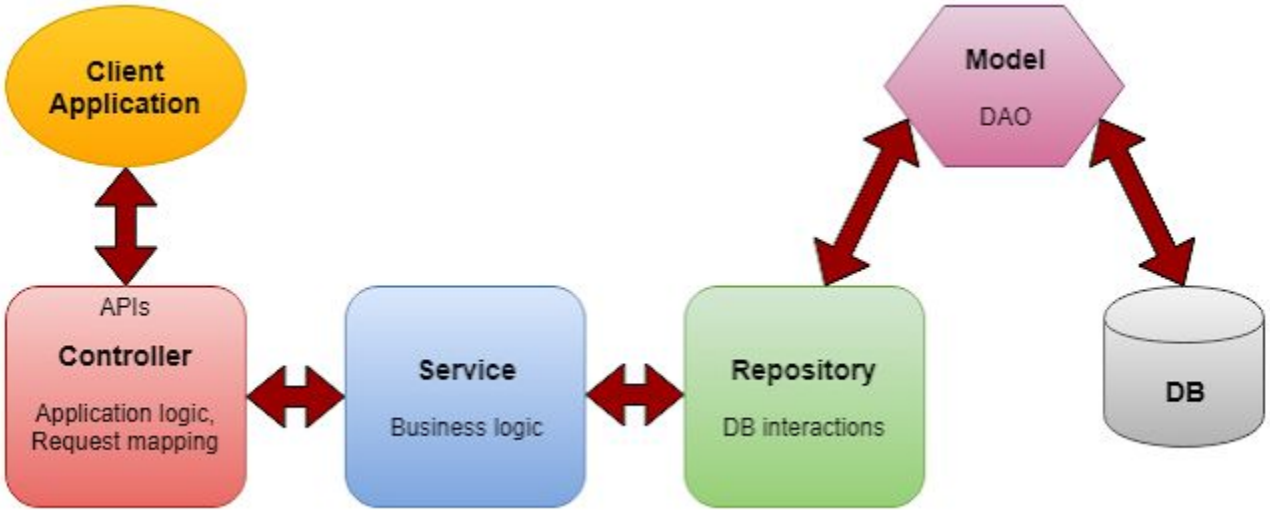
Application.properties

```
1  spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
2  spring.datasource.username=username
3  spring.datasource.password=password
```

Perhatikan bahwa setiap dependency library yang diperlukan bisa diambil dari Maven Repository

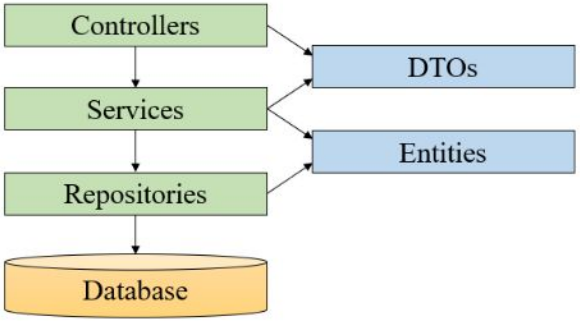
https://mvnrepository.com/

MVC Architecture



source:
http://randikatech.blogspot.com/2019/09/get-your-hands-dirty-with-micro-services.html

Project Conceptual Architecture



Controllers

Komponen yang mengexpose fungsional aplikasi dalam bentuk API endpoint yang menghandle HTTP Request. KOmponen ini juga menghandle berbagai type validasi dan exceptions

Services

- Class untuk menampung logic sebelum melakukan aksi ke dalam database
- Nama service biasanya nama model+service , contoh : StudentService.java
- Class ini yang menghubungkan Controller class saat ingin melakukan operation ke database


Repository

- Class yang menyediakan fungsi query ke database
- Nama interface repository class biasanya dengan nama entity class (model)+repository, contoh : StudentEntityRepository.java
- Setiap entity class memiliki satu interface repository
- Ditandai dengan annotation @Repository
- Dengan menggunakan Spring data jpa, kita terhindar dari menulis query secara manual

Entity

- Class yang merepresentasikan satu table di database. Contoh StudentEntity.Java, berarti merepresentasikan student table
- Setiap attribute di dalam entity class merepresentasikan field table
- Ditandai dengan annotation @Entity pada class, @Column pada attribute, @Id untuk menandakan primary key

HANDS-ON LAB

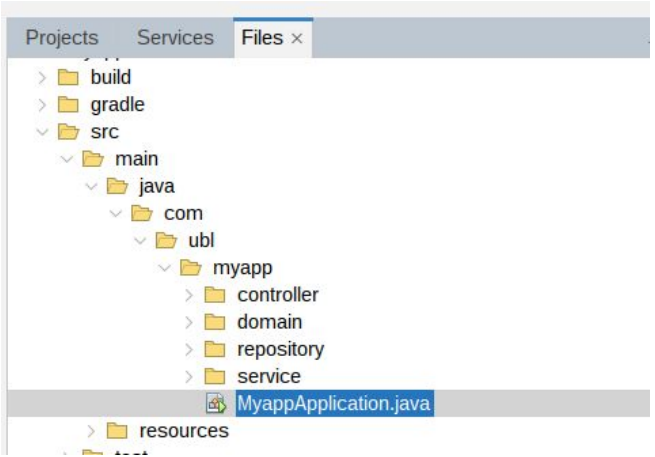
1. **Create database dengan nama studentapp_db**

2. **Create table dengan nama student**

```
CREATE TABLE student (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nim VARCHAR(10) UNIQUE,
    full_name VARCHAR(50),
    address TEXT,
    date_of_birth DATE
    );
```

3. Buat folder repository dan service

Struktur project




Tambahkan project lombok library pada gradle dependency

compileOnly 'org.projectlombok:lombok:1.18.36'

Lombok is a Java library that provides annotations to simplify Java development by automating the generation of boilerplate code.

Key features include automatic generation of getters, setters, equals, hashCode, and toString methods, as well as a facility for automatic resource management.

It aims to reduce the amount of manual coding, thereby streamlining the codebase and reducing potential for errors.

https://projectlombok.org/


4. Buat entity class dengan nama StudentEntity.java pada folder repository

```java
@Entity
@Table(name="student")
@Getter
@Setter
public class StudentEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name="nim", nullable=false)
    private String nim;

    @Column(name="full_name", nullable=false)
    private String fullName;

    @Column(name="address", nullable=false)
    private String address;

    @Column(name="date_of_birth", nullable=false)
    private String dateOfBirth;



}
```

5. Buat repository class dengan nama StudentRepository.java pada folder repository

```java
/**
 *
 * @author riskiana
 */
@Repository
public interface StudentRepository extends JpaRepository<StudentEntity,Long>{

}

```

6. Buat service class dengan nama StudentService.java pada folder service

```java
@Service
@RequiredArgsConstructor
public class StudentService {

    private final StudentRepository studentRepository;


    public List<Student> getAllStudents() {
        List<StudentEntity> studentEntities = studentRepository.findAll();
        return studentEntities.stream()
                .map(entity -> map(entity))
                .toList();


    }

    private Student map(StudentEntity entity){
        Student student = new Student();
        student.setFullName(entity.getFullName());
        student.setAddress(entity.getAddress());
        return student;
    }


}
```

7. Ubah endpoint get students pada controller class untuk memanggil data dari Service class

```java
 *
 * @author riskiana
 */
@RestController
@RequiredArgsConstructor
public class StudentController {

    private final StudentService studentService;

    @GetMapping("/students")
    public List<Student> getStudents(){
        List<Student> studentList = studentService.getAllStudents();

        return studentList;
    }


}
```


8. Insert data manual ke database pada table student

9. Test melalui browser dengan mengakses students endpoint lalu amati hasilnya bahwa data yang berasal dari database berhasil ditampilkan

GITHUB

Wajib!!!

Buat github account : https://github.com/