

# Materi 10 : Menggunakan JPA dengan Thymeleaf di Java

---

## DAFTAR ISI

1.1	Pengenalan JPA .....	1
1.1.1	Definisi JPA (Java Persistence API) .....	1
1.1.2	Perbedaan antara JDBC dan JPA: .....	1
1.1.3	Entity dan Entity Class.....	1
1.1.4	Anotasi JPA untuk Entity Class: .....	1
1.2	Konfigurasi Spring Boot dengan JPA.....	2
1.2.1	Mengatur sumber data (Data Source): .....	2
1.2.2	Konfigurasi Entity Manager Factory .....	2
1.2.3	Repository Interface.....	2
1.3	Membuat Project CRUD Dengan JPA.....	3
1.3.1	Membuat Project .....	3
1.3.2	Mengimplementasikan konsep CRUD menggunakan JPA di SpringBoot .....	5

## 1.1 Pengenalan JPA

### 1.1.1 Definisi JPA (Java Persistence API)

Java Persistence API (JPA) adalah spesifikasi Java untuk manajemen persistensi data dalam aplikasi Java Enterprise. JPA adalah bagian dari Jakarta EE (sebelumnya dikenal sebagai Java EE) dan merupakan cara standar untuk berinteraksi dengan database dalam aplikasi Java. JPA menyediakan abstraksi tingkat tinggi yang memungkinkan pengembang untuk bekerja dengan database menggunakan objek Java daripada SQL murni.

Dengan kata lain, JPA adalah spesifikasi Java yang memungkinkan aplikasi Java untuk berinteraksi dengan database menggunakan objek Java, menggantikan penggunaan SQL murni, dan digunakan dalam aplikasi Java Enterprise.

Note: Spesifikasi adalah dokumen atau set aturan dan definisi yang merinci cara sesuatu seharusnya berfungsi atau diimplementasikan

### 1.1.2 Perbedaan antara JDBC dan JPA:

- JDBC (Java Database Connectivity) adalah API Java yang digunakan untuk berinteraksi dengan database secara langsung dengan mengirimkan perintah SQL. JDBC **membutuhkan penulisan kode SQL** dan dapat memakan waktu.
- JPA, di sisi lain, adalah spesifikasi yang lebih tinggi dan abstrak yang memungkinkan pengembang untuk **bekerja dengan objek Java yang mewakili data dalam database**. Ini mengurangi penulisan SQL dan menyediakan cara yang lebih mudah dan efisien untuk mengakses dan memanipulasi data.

### 1.1.3 Entity dan Entity Class

- Entity dalam JPA  
Entity adalah objek Java yang merepresentasikan data dalam database. Setiap entitas mewakili entitas dalam database dan memiliki properti yang sesuai dengan kolom dalam tabel.
- Entity Class adalah kelas Java yang dianotasi dengan **@Entity** untuk menandakan bahwa itu adalah entitas dalam JPA.
- Properti dalam kelas ini sesuai dengan kolom dalam tabel yang diwakilinya.
- Primary key dari tabel dapat ditandai dengan anotasi **@Id**.

### 1.1.4 Anotasi JPA untuk Entity Class:

- @Entity**: Menandakan bahwa kelas adalah entitas dalam JPA.
- @Id**: Menandai properti sebagai primary key.
- @GeneratedValue**: Mengatur cara nilai primary key yang dihasilkan (**AUTO**, **IDENTITY**, dll.).

- **@Table**: Menghubungkan entitas dengan tabel dalam database.
- **@Column**: Mengonfigurasi properti entitas ke kolom dalam tabel.

## 1.2 Konfigurasi Spring Boot dengan JPA

### 1.2.1 Mengatur sumber data (Data Source):

Data Source adalah komponen yang menghubungkan aplikasi Spring Boot dengan database. Anda dapat mengkonfigurasi Data Source dalam file **application.properties** atau **application.yml**. Berikut adalah contoh konfigurasi database properties dalam **application.properties**:

```
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
spring.datasource.username=username
spring.datasource.password=password
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

Anda perlu mengganti URL, username, password, dan driver-class-name sesuai dengan pengaturan database Anda.

### 1.2.2 Konfigurasi Entity Manager Factory

**EntityManagerFactory** adalah komponen yang mengelola entitas dan koneksi database. Anda dapat mengkonfigurasi **EntityManagerFactory** dalam file **application.properties** atau dengan menggunakan Java Configuration. Berikut adalah Contoh konfigurasinya

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
```

Keterangan :

- **spring.jpa.hibernate.ddl-auto=update** akan mengizinkan Hibernate membuat tabel jika belum ada atau memperbarui skema jika diperlukan.
- **spring.jpa.properties.hibernate.dialect** menentukan dialek Hibernate yang sesuai dengan database yang digunakan.

### 1.2.3 Repository Interface

Repository Interface adalah komponen yang digunakan untuk berinteraksi dengan entitas dalam database. Repository Interface dapat didefinisikan dengan meng-extend

**JpaRepository** atau **CrudRepository** dari Spring Data JPA. Berikut adalah contoh pembuatan Repository Interface:

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

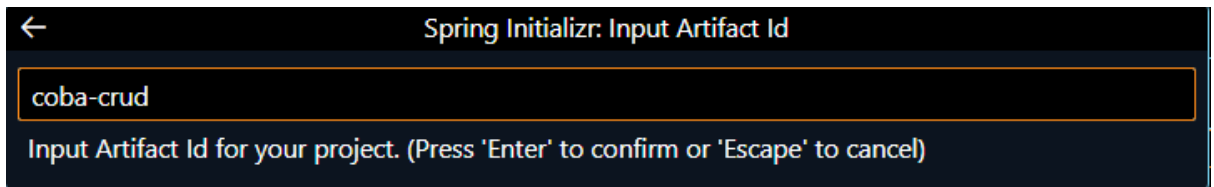
}
```

Ini adalah langkah-langkah dasar untuk mengkonfigurasi Spring Boot dengan JPA. Dengan mengatur Data Source, EntityManagerFactory, dan Repository Interface, Anda dapat mulai mengakses dan memanipulasi data dalam database menggunakan JPA dalam aplikasi Spring Boot.

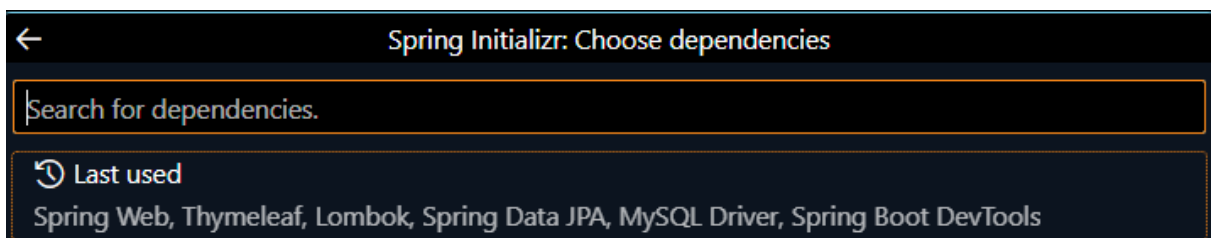
## 1.3 Membuat Project CRUD Dengan JPA

### 1.3.1 Membuat Project

Buatlah project SpringBoot Maven dengan nama *coba-crud*:



Tambahkan dependensi yang diperlukan :



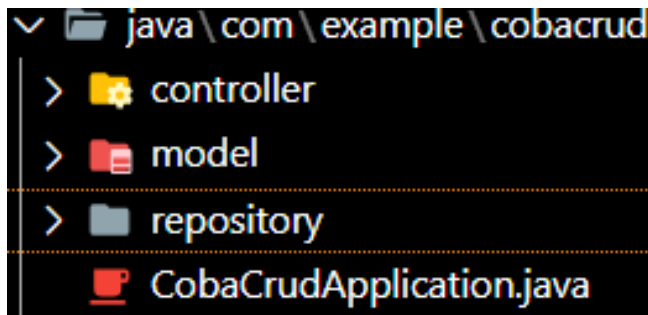
Berikut adalah penjelasan singkat mengenai kegunaan dari masing-masing dependensi yang umumnya digunakan dalam proyek Spring Boot:

- Spring Web:
  - a. Dependensi Spring Web menyediakan dukungan untuk pengembangan aplikasi web menggunakan Spring.
  - b. Ini mencakup fitur seperti manajemen permintaan HTTP, penanganan RESTful API, dan kemampuan untuk membuat controller dan view web.

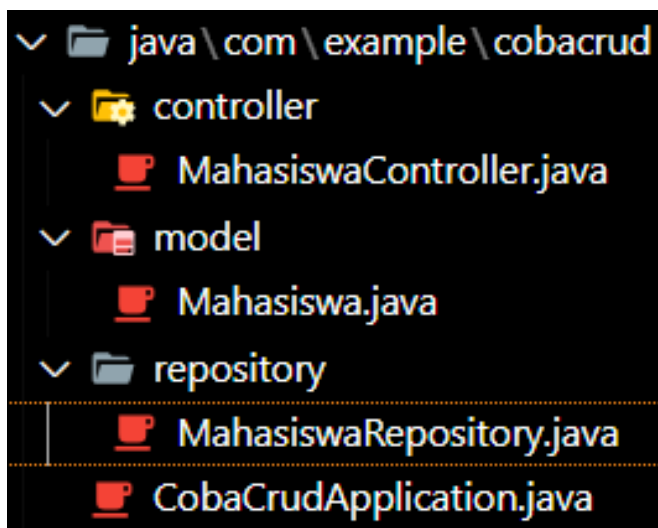
- c. Spring Web memungkinkan Anda untuk dengan mudah membangun aplikasi web dengan Spring Boot.
- Thymeleaf:
  - a. Thymeleaf adalah mesin template Java yang digunakan untuk membuat tampilan web dalam aplikasi Spring Boot.
  - b. Ini memungkinkan pengembang untuk memasukkan data dari model ke dalam tampilan HTML dengan cara yang dekat dengan HTML asli.
  - c. Thymeleaf sangat cocok untuk pengembangan aplikasi web berbasis Java dan sangat terintegrasi dengan Spring Boot.
- Lombok:
  - a. Lombok adalah sebuah framework yang memungkinkan pengembang untuk mengurangi penulisan kode boilerplate dalam Java.
  - b. Dengan Lombok, Anda dapat menggunakan anotasi seperti `@Data`, `@Getter`, `@Setter`, dan lainnya untuk secara otomatis menghasilkan metode `getter`, `setter`, `toString`, `hashCode`, dan `equals`, yang mengurangi penulisan kode berulang.
  - c. Ini mempermudah pengembangan aplikasi dengan mengurangi jumlah kode yang perlu ditulis.
- Spring Data JPA:
  - a. Dependensi Spring Data JPA adalah bagian dari proyek Spring Data yang menyediakan dukungan untuk JPA (Java Persistence API) dalam aplikasi Spring.
  - b. Ini memungkinkan pengembang untuk dengan mudah mengakses dan memanipulasi data dalam database menggunakan entitas Java dan Repository Interface.
  - c. Spring Data JPA mengurangi penulisan kode boilerplate yang terkait dengan operasi database.
- MySQL Driver:
  - a. MySQL Driver adalah ketergantungan yang diperlukan untuk menghubungkan aplikasi Spring Boot dengan database MySQL.
  - b. Ini adalah implementasi driver JDBC yang digunakan oleh aplikasi Spring Boot untuk berkomunikasi dengan server database MySQL.
- Spring Boot DevTools:
  - a. Spring Boot DevTools adalah dependensi yang digunakan selama pengembangan dan pengujian aplikasi Spring Boot.

- b. Ini menyediakan alat yang membantu dalam pengembangan seperti restart otomatis aplikasi saat kode berubah, pemantauan kode sumber, dan pemantauan perubahan properti aplikasi.
- c. Ini dapat mempercepat siklus pengembangan dan memudahkan debugging aplikasi.

Setelah menambahkan dependensi diatas dan membuat project, buka project tersebut lalu buat package/folder **controller**, **model**, dan **repository** :



Kemudian buat file untuk masing-masing package sebagai berikut :



Project untuk persiapan membuat aplikasi CRUD sederhana telah dibuat.

### 1.3.2 Mengimplementasikan konsep CRUD menggunakan JPA di SpringBoot

1. Bagian Membuat data (CREATE) :

Tambahkan kode berikut ke file **Mahasiswa.java** :

```
package com.example.cobacrud.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
```

```
import jakarta.persistence.Id;
import lombok.Data;

@Entity
@Data
public class Mahasiswa {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer id;
    String nim;
    String nama;
}
```

Keterangan :

- **@Entity**, digunakan untuk menandai bahwa kelas **Mahasiswa** adalah sebuah entitas JPA. Ini mengindikasikan bahwa kelas ini akan dipetakan ke sebuah tabel dalam database.
- **@Data**, adalah anotasi khusus yang menghasilkan boilerplate code seperti **getter**, **x**, **equals**, **hashCode**, dan **toString** secara otomatis. Dengan anotasi **@Data**, Anda tidak perlu menuliskan metode-metode tersebut secara manual.
- **@Id** digunakan untuk menandai bahwa properti **id** adalah primary key dari entitas **Mahasiswa**.
- **@GeneratedValue** digunakan bersamaan dengan **@Id** untuk mengonfigurasi bagaimana nilai primary key akan di-generate.
- **strategy = GenerationType.IDENTITY** menunjukkan bahwa nilai primary key akan di-generate secara otomatis oleh database dengan menggunakan tipe data yang mendukung auto-increment, seperti **AUTO\_INCREMENT** pada MySQL.

Kemudian tambahkan kode berikut di file **MahasiswaRepository.java** jangan lupa bahwa repository tersebut berupa interface dan bukan *class* :

```
package com.example.cobacrud.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import com.example.cobacrud.model.Mahasiswa;

public interface MahasiswaRepository
    extends JpaRepository <Mahasiswa, Integer> {

}
```

Dengan adanya `MahasiswaRepository`, kita dapat menggunakan metode yang disediakan oleh `JpaRepository` (seperti `save`, `findById`, `findAll`, `delete`, dll.) untuk melakukan operasi CRUD pada entitas `Mahasiswa` dalam database tanpa perlu menulis implementasi metode-metode tersebut secara manual. Repository ini memungkinkan kita untuk secara mudah berinteraksi dengan database dan mengelola data entitas `Mahasiswa` Anda dalam aplikasi Spring Boot.

Kemudian tambahkan kode berikut pada `MahasiswaController.java`:

```
package com.example.cobacrud.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import com.example.cobacrud.model.Mahasiswa;
import com.example.cobacrud.repository.MahasiswaRepository;

@Controller
public class MahasiswaController {

    @Autowired
    private MahasiswaRepository mahasiswaRepository;

    @GetMapping("/add-mahasiswa")
    public String addMahasiswa(Model model) {
        Mahasiswa mahasiswa = new Mahasiswa();
        model.addAttribute("mahasiswa", mahasiswa);
        return "addMahasiswa";
    }

    @PostMapping("/save-mahasiswa")
    public String saveMahasiswa (@ModelAttribute("mahasiswa")
                                Mahasiswa mahasiswa) {
        mahasiswaRepository.save(mahasiswa);
        return "redirect:/add-mahasiswa";
    }
}
```

Keterangan :

- `@Controller` adalah anotasi yang menandakan bahwa kelas `MahasiswaController` adalah komponen Spring dan berperan sebagai controller



dalam aplikasi. Controller ini akan menangani permintaan HTTP terkait operasi tambah data mahasiswa.

- `@Autowired` digunakan untuk melakukan injeksi ketergantungan. Dalam hal ini, `'MahasiswaRepository'` akan di-injeksi ke dalam controller untuk digunakan.
- `@GetMapping("/add-mahasiswa")` anotasi ini mengindikasikan bahwa metode `'addMahasiswa'` akan menangani permintaan HTTP GET ke URL `'/add-mahasiswa'`. Metode ini digunakan untuk menampilkan halaman tambah mahasiswa.
- `public String addMahasiswa(Model model)` Metode ini menerima objek `'Model'` sebagai parameter, yang digunakan untuk mengirim data ke tampilan. Di dalam metode, kita membuat objek `'Mahasiswa'` kosong yang akan digunakan untuk mengisi formulir tambah mahasiswa. Kemudian, objek `'mahasiswa'` tersebut ditambahkan ke model dengan nama `"mahasiswa"`. Metode ini mengembalikan nama tampilan `"addMahasiswa"`.
- `@PostMapping("/save-mahasiswa")` Anotasi ini mengindikasikan bahwa metode `'saveMahasiswa'` akan menangani permintaan HTTP POST ke URL `'/save-mahasiswa'`. Metode ini digunakan untuk menyimpan data mahasiswa yang telah diisi pada formulir.
- `public String saveMahasiswa(@ModelAttribute("mahasiswa") Mahasiswa mahasiswa)` Metode ini menerima objek `'Mahasiswa'` yang diisi dari formulir (dengan menggunakan anotasi `'@ModelAttribute'`) sebagai parameter. Kemudian, objek `'mahasiswa'` ini disimpan ke dalam database menggunakan `'mahasiswaRepository.save(mahasiswa)'`. Setelah penyimpanan data berhasil, metode ini mengembalikan redirect ke URL `'/add-mahasiswa'`, sehingga pengguna dapat kembali ke halaman tambah data mahasiswa.

Seluruh logika ini memungkinkan pengguna untuk mengisi formulir dengan data mahasiswa baru, dan data tersebut akan disimpan ke dalam database MySQL melalui Spring Data JPA ketika pengguna mengirimkan formulir. Setelah penyimpanan berhasil, pengguna akan diarahkan kembali ke halaman tambah mahasiswa.

Kemudian Buat `addMahasiswa.html` untuk tampilan HTML nya:

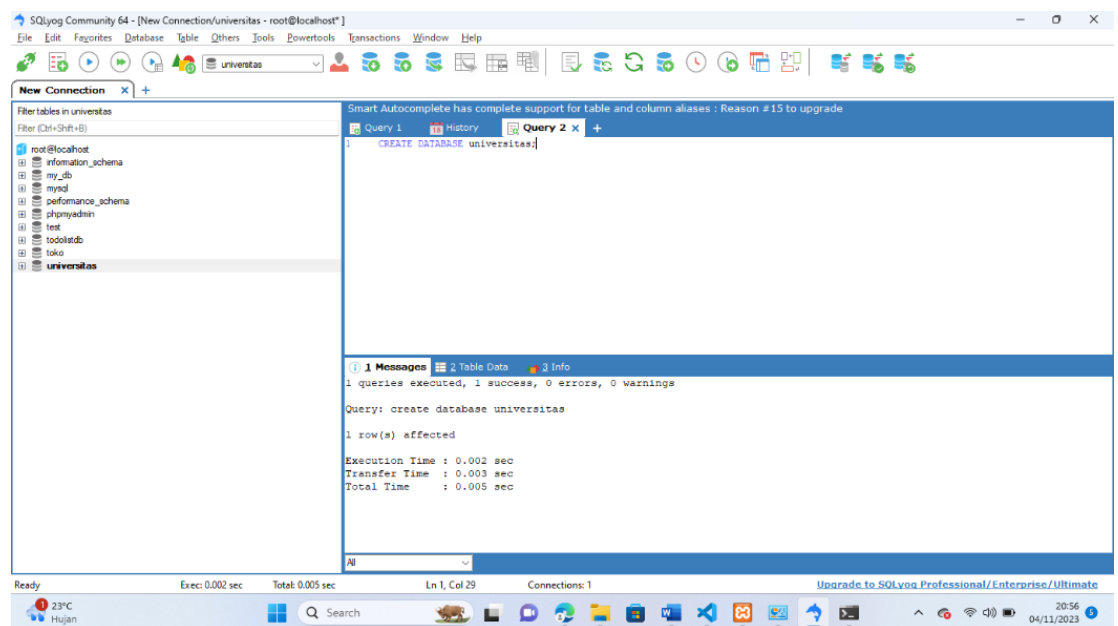


Isi `addMahasiswa.html` dengan kode berikut :

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Tambah Mahasiswa</title>
  </head>

  <body>
    <h1>Add Mahasiswa</h1>
    <form th:action="@{/save-mahasiswa}" method="post">
      <input type="text" th:field="${mahasiswa.nim}">
      <input type="text" th:field="${mahasiswa.nama}">
      <button type="submit">Save Mahasiswa</button>
    </form>
  </body>
</html>
```

Kemudian buat database dengan nama *universitas* di MySql :



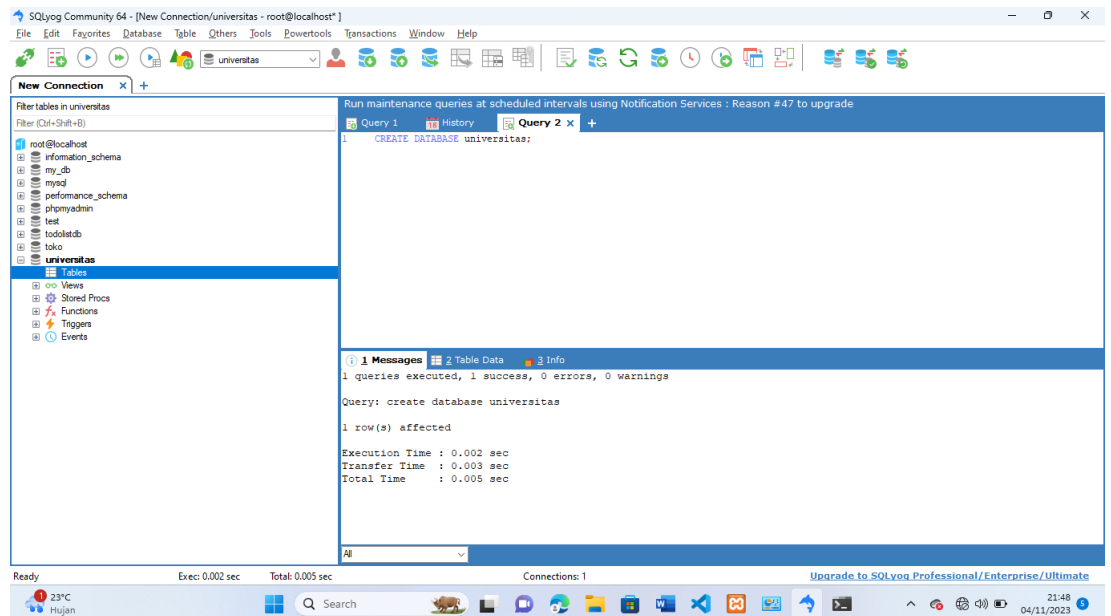
Lalu tambahkan kode berikut di `application.properties` :

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/universitas
spring.datasource.username=root
spring.datasource.password=
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect
.MySQLDialect
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type=TRACE
```

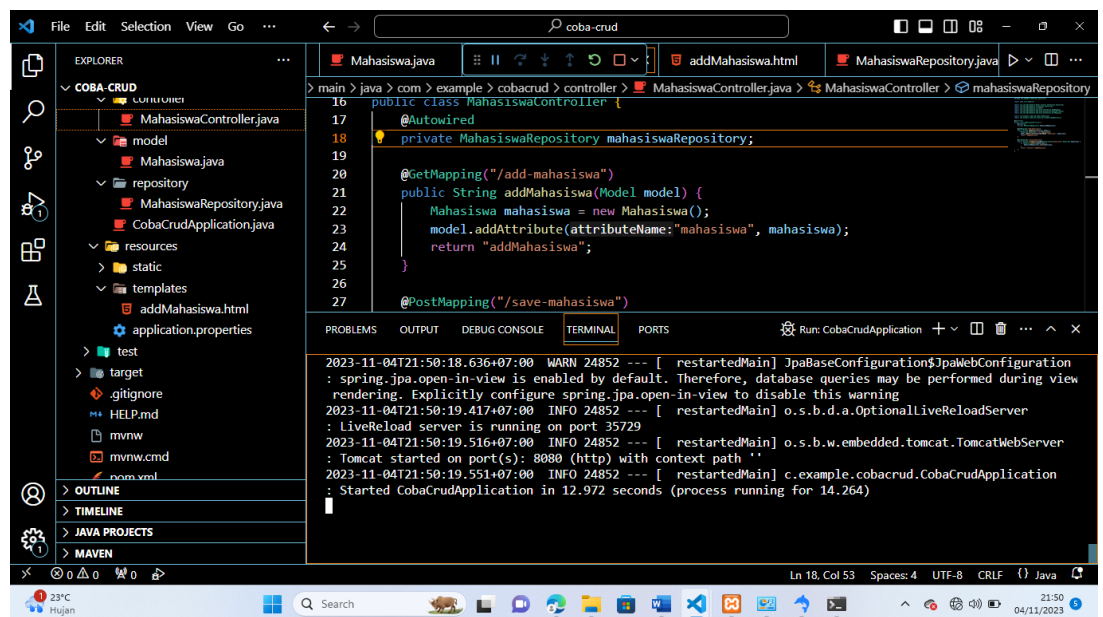
Keterangan :

- **spring.jpa.hibernate.ddl-auto=update** Properti ini mengontrol perilaku Hibernate DDL (Data Definition Language) auto-generate. Nilai **update** akan membuat Hibernate menghasilkan tabel yang sesuai dengan definisi entitas Java Anda dan memperbarui skema database jika diperlukan tanpa menghapus data yang ada.
- **spring.datasource.url=jdbc:mysql://localhost:3306/universitas** Ini adalah URL koneksi JDBC ke database MySQL.
- **spring.datasource.username=root** Ini adalah nama pengguna MySQL yang digunakan untuk mengakses database. Dalam contoh ini, nama pengguna adalah "root."
- **spring.datasource.password=** Ini adalah kata sandi pengguna MySQL yang digunakan untuk mengakses database. Dalam contoh ini, kata sandi dikosongkan (kosong).
- **spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect** Properti ini mengonfigurasi dialek Hibernate yang sesuai dengan database yang digunakan.
- **logging.level.org.hibernate.SQL=DEBUG** Properti ini mengonfigurasi tingkat log untuk SQL statements yang dihasilkan oleh Hibernate. Di sini, tingkat log diatur ke "DEBUG," sehingga pernyataan SQL akan dicetak ke log dalam tingkat debug.
- **logging.level.org.hibernate.type=TRACE**  
Properti ini mengonfigurasi tingkat log untuk jenis Hibernate. Di sini, tingkat log diatur ke **TRACE** sehingga informasi detail tentang jenis Hibernate akan dicetak ke log. Konfigurasi ini digunakan untuk mengatur koneksi ke database MySQL, mengatur perilaku Hibernate, dan mengontrol tingkat log yang dicetak selama proses interaksi dengan database dalam proyek Spring Boot yang menggunakan Spring Data JPA.

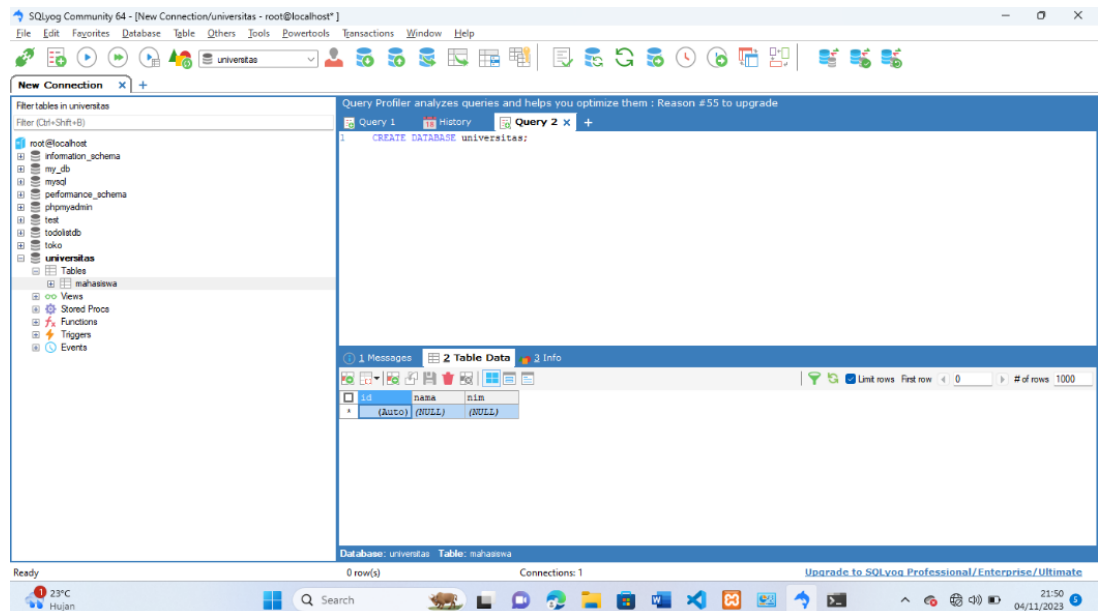
Pada mulanya database tersebut tidak akan memiliki table apapun



Namun setelah kita melakukan run java maka akan muncul semua model yang telah diberi `@Entity` menjadi sebuah table di database



Hasilnya adalah sebagai berikut :



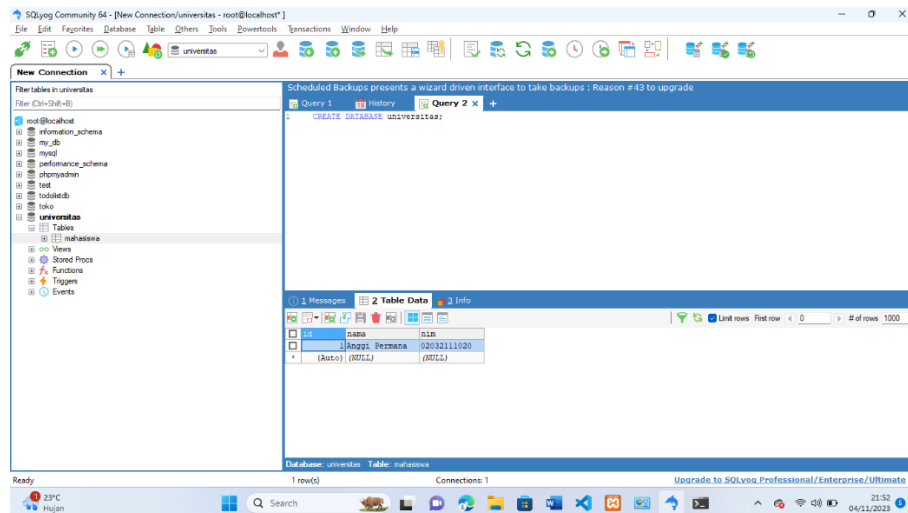
Namun table mahasiswa masih kosong atau belum memiliki data, kita coba tambah data melalui web browser, caranya akses <http://localhost:8080/add-mahasiswa> , maka tampilannya adalah sebagai berikut :

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/add-mahasiswa'. The page title is 'Add Mahasiswa'. Below the title, there are two input fields for 'id' and 'nama', and a 'Save Mahasiswa' button.

Lalu kita coba inputkan data nim dan namanya :

The screenshot shows the same web browser window, but now the 'id' field contains the value '02032111020' and the 'nama' field contains the value 'Anggi Permana'. The 'Save Mahasiswa' button is still visible.

Setelah tekan button Save Mahasiswa maka inputan akan kosong Kembali seperti tampilan awal, lalu kita coba kembali ke SqlYog dan refresh table mahasiswa tersebut, maka data akan masuk ke table mahasiswa :



Kita berhasil mengimplementasikan konsep CREATE dalam CRUD pada database menggunakan JPA SpringBoot.

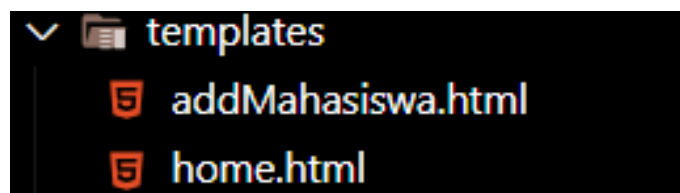
## 2. Bagian Membaca Data (READ)

Setelah kita berhasil membuat data dan menambahkannya ke dalam table, kini saatnya kita mengambil data tersebut dan menampilkannya melalui web browser.

Tambahkan kode berikut di *MahasiswaController.java*:

```
@GetMapping("/")
public String allMahasiswa(Model model) {
    model.addAttribute("allMahasiswa",
        mahasiswaRepository.findAll());
    return "home";
}
```

Kemudian tambah tampilan **home.html** untuk menampilkan semua data mahasiswa dari database :



Isi **home.html** dengan kode berikut :

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Data Mahasiswa</title>
    </head>
```

```

<body>
  <h1>Data Mahasiswa</h1>
  <table border="1">
    <thead>
      <tr>
        <th>Id</th>
        <th>Nim</th>
        <th>Nama</th>
      </tr>
    </thead>
    <tbody>
      <tr th:each="mahasiswa : ${allMahasiswa}">
        <td th:text="${mahasiswa.id}"></td>
        <td th:text="${mahasiswa.nim}"></td>
        <td th:text="${mahasiswa.nama}"></td>
      </tr>
    </tbody>
  </table>
  <a href="/add-mahasiswa">
    <button>Tambah Mahasiswa</button>
  </a>
</body>
</html>

```

Terakhir, agar setelah menambahkan data mahasiswa baru maka langsung Kembali ke home, maka ubah kode di controller **PostMapping** sebelumnya bagian redirect: kode sebelumnya :

```

@PostMapping("/save-mahasiswa")
public String saveMahasiswa (@ModelAttribute("mahasiswa")
                             Mahasiswa mahasiswa) {
    mahasiswaRepository.save(mahasiswa);
    return "redirect:/add-mahasiswa";
}

```

Kode baru :

```

@PostMapping("/save-mahasiswa")
public String saveMahasiswa(@ModelAttribute("mahasiswa")
                             Mahasiswa mahasiswa) {
    mahasiswaRepository.save(mahasiswa);
    return "redirect:/";
}

```

Kemudian jalankan/run project SpringBoot, lalu buka <http://localhost:8080/> maka akan menampilkan tampilan berikut :



Id	Nim	Nama
1	02032111020	Anggi Permana

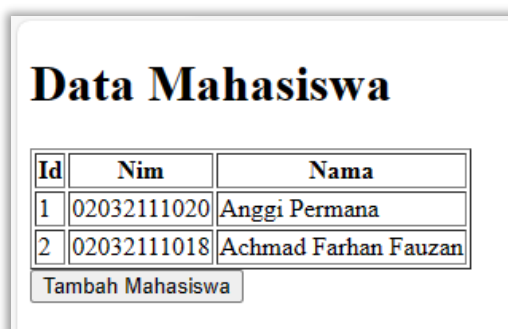
Tambah Mahasiswa

Jika kita tekan tombol *Tambah Mahasiswa* maka akan berpindah ke halaman **addMahasiswa.html**, lalu kita coba tambah data baru :



02032111018 Achmad Farhan Fauzan Save Mahasiswa

Setelah itu klik tombol Save Mahasiswa maka akan Kembali ke halaman home lagi dan data sudah otomatis bertambah :



Id	Nim	Nama
1	02032111020	Anggi Permana
2	02032111018	Achmad Farhan Fauzan

Tambah Mahasiswa

Dengan demikian kita telah berhasil Menambah (CREATE) dan Menampilkan (READ) data dari table menggunakan JPA.

### 3. Bagian Mengedit Data (UPDATE)



Selanjutnya kita akan mencoba mengedit data yang sudah ada pada table Mahasiswa di database kita. Pertama, tambahkan kode berikut di **MahasiswaController.java** :

```
@GetMapping("/update-mahasiswa/{id}")
public String updateMahasiswa(@PathVariable(value = "id")
                               Integer id, Model model) {
    Mahasiswa mahasiswa = mahasiswaRepository.getReferenceById(id);
    model.addAttribute("mahasiswa", mahasiswa);
    return "updateMahasiswa";
}
```

Keterangan :

- **@GetMapping("/update-mahasiswa/{id}")** adalah anotasi yang mengindikasikan bahwa metode ini akan menangani permintaan HTTP GET ke URL yang diakhiri dengan **/update-mahasiswa/{id}**. **{id}** adalah variabel path yang digunakan untuk menentukan ID mahasiswa yang akan diperbarui.
- **public String updateMahasiswa(@PathVariable(value = "id") Integer id, Model model)** Metode ini mengambil ID mahasiswa dari variabel path menggunakan anotasi **@PathVariable**. Selain itu, metode menerima objek **Model** sebagai parameter, yang digunakan untuk mengirim data ke tampilan.
- **Mahasiswa mahasiswa = mahasiswaRepository.getReferenceById(id)** Di dalam metode, kita menggunakan repository **mahasiswaRepository** untuk mengambil referensi ke data mahasiswa berdasarkan ID yang diberikan. Fungsi **getReferenceById(id)** digunakan untuk mengambil referensi data mahasiswa. Ini mungkin berbeda dari **findById(id)** karena itu hanya mengambil data yang sudah ada di cache atau di tingkat perusahaan jika digunakan. Hasilnya disimpan dalam objek **mahasiswa**.
- **model.addAttribute("mahasiswa", mahasiswa)** Kemudian, objek **mahasiswa** tersebut ditambahkan ke model dengan nama mahasiswa. Ini memungkinkan data mahasiswa yang akan diperbarui ditampilkan pada halaman yang sesuai.
- **return "updateMahasiswa"** Metode ini mengembalikan nama tampilan **updateMahasiswa**. Halaman ini akan menampilkan data mahasiswa yang akan diperbarui, dan pengguna dapat mengedit informasi mahasiswa pada halaman ini.

Selanjutnya tambahkan file html baru yaitu **updateMahasiswa.html** :

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Edit Mahasiswa</title>
  </head>

  <body>
    <h1>Update Mahasiswa</h1>
    <form action="/save-mahasiswa" method="post">
      <input type="hidden" th:field="${mahasiswa.id}">
      <input type="text" th:field="${mahasiswa.nim}">
      <input type="text" th:field="${mahasiswa.nama}">
      <button type="submit">Save Mahasiswa</button>
    </form>
  </body>
</html>

```

Hampir sama seperti tambah mahasiswa, namun ada tambahan yaitu : `<input type="hidden" th:field="${mahasiswa.id}">` Ini digunakan agar Ketika menyimpan data tidak membuat id baru, dan masih tetap menggunakan id yang lama, sehingga di table mahasiswa tidak bertambah data, melainkan data dengan id yang sama tersebut telah berubah/terupdate.

Terakhir, tambahkan tombol *update* dan *delete* pada *home.html* sehingga menjadi berikut :

```

<table border="1">
  <thead>
    <tr>
      <th>Id</th>
      <th>Nim</th>
      <th>Nama</th>
      <th>Aksi</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="mahasiswa : ${allMahasiswa}">
      <td th:text="${mahasiswa.id}"></td>
      <td th:text="${mahasiswa.nim}"></td>
      <td th:text="${mahasiswa.nama}"></td>
      <td>
        <a
          th:href="@{/update-mahasiswa/{id}
            (id=${mahasiswa.id})}"
        > <button>Update</button>
        </a>
      </td>
    </tr>
  </tbody>
</table>

```

```

<a th:href="@{/delete-mahasiswa/{id}
(id=${mahasiswa.id})}"
> <button>Delete</button>
</a>
</td>
</tr>
</tbody>
</table>

```

Keterangan :

- `<a th:href="@{/update-mahasiswa/{id}(id=${mahasiswa.id})}">` Ini adalah elemen HTML `<a>` (anchor) yang digunakan untuk membuat tautan (link). Atribut `th:href` adalah atribut Thymeleaf yang digunakan untuk menghasilkan URL tautan. Dalam kasus ini, itu menghasilkan URL dengan path `/update-mahasiswa/{id}` dimana `{id}` akan digantikan oleh nilai dari properti `id` dari objek `mahasiswa`.
- `(id=${mahasiswa.id})` Ini adalah sintaks Thymeleaf yang digunakan untuk menyisipkan nilai dari properti `id` dari objek `mahasiswa` ke dalam URL. Ini membuat tautan dinamis sesuai dengan ID mahasiswa yang sedang diproses.

Selanjutnya jalankan project dan buka <http://localhost:8080/> maka akan tampil sebagai berikut :

Data Mahasiswa			
Id	Nim	Nama	Aksi
1	02032111020	Anggi Permana	Update Delete
2	02032111018	Achmad Farhan Fauzan	Update Delete
Tambah Mahasiswa			

Jika kita klik tombol **update** yang paling atas pada data pertama, maka akan redirect ke halaman update dan akan menampilkan data yang kita pilih.

Update Mahasiswa		
02032111020	Anggi Permana	Save Mahasiswa

Jika kita mengganti nama nya seperti ini :

## Update Mahasiswa

Lalu tekan tombol Save Mahasiswa maka akan redirect ke halaman home dan data akan terupdate.

## Data Mahasiswa

Id	Nim	Nama	Aksi	
1	02032111020	Zayn Gibran	<input type="button" value="Update"/>	<input type="button" value="Delete"/>
2	02032111018	Achmad Farhan Fauzan	<input type="button" value="Update"/>	<input type="button" value="Delete"/>
<input type="button" value="Tambah Mahasiswa"/>				

#### 4. Bagian Hapus Data (DELETE)

Terakhir dari konsep CRUD adalah menghapus data atau DELETE, kita cukup menambahkan satu controller untuk menghapus data pada **MahasiswaController.java**:

```
@GetMapping("/delete-mahasiswa/{id}")
public String deleteMahasiswa(@PathVariable(value = "id") Integer id)
{
    mahasiswaRepository.deleteById(id);
    return "redirect:/";
}
```

Lalu jalankan project dan buka Kembali url <http://localhost:8080/>.

## Data Mahasiswa

Id	Nim	Nama	Aksi	
1	02032111020	Zayn Gibran	<input type="button" value="Update"/>	<input type="button" value="Delete"/>
2	02032111018	Achmad Farhan Fauzan	<input type="button" value="Update"/>	<input type="button" value="Delete"/>
<input type="button" value="Tambah Mahasiswa"/>				

Jika kita klik tombol **delete** mahasiswa Achmad Farhan Fauzan, maka akan langsung terhapus.

## Data Mahasiswa

Id	Nim	Nama	Aksi	
1	02032111020	Zayn Gibran	<input type="button" value="Update"/>	<input type="button" value="Delete"/>
<input type="button" value="Tambah Mahasiswa"/>				