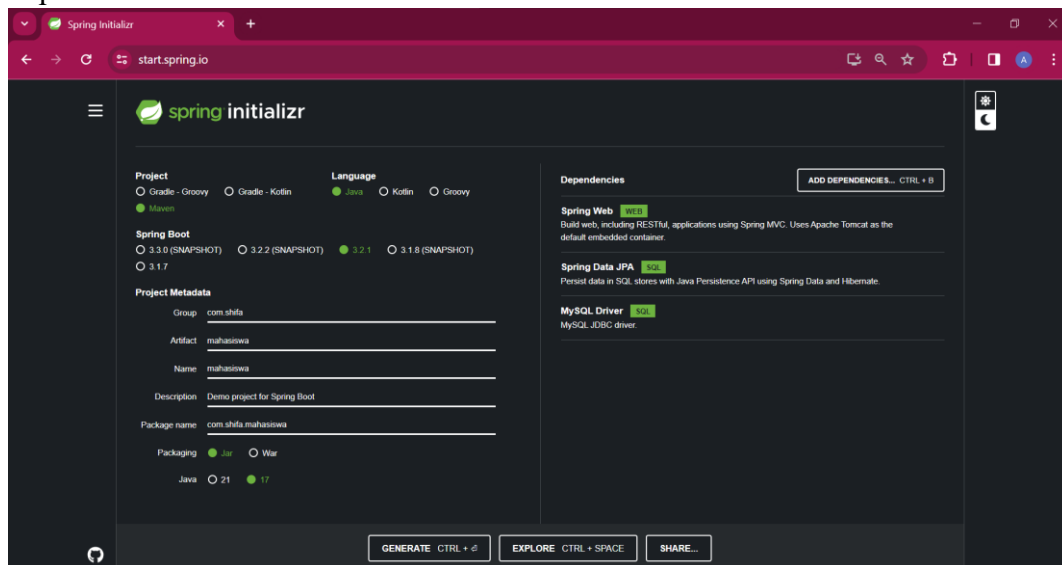


## Project Service Mahasiswa

Service adalah bagian yang menyediakan layanan atau fungsi tertentu kepada pengguna atau aplikasi lainnya melalui jaringan. Layanan ini dapat mencakup berbagai hal, seperti pemrosesan data, penyimpanan, komunikasi, atau fungsionalitas lainnya yang dapat diakses dan digunakan oleh komponen sistem terdistribusi yang berbeda.

### 1. Tool yang digunakan

Untuk membuat proyek Spring Boot baru dengan cepat dengan dependensi yang diperlukan. Bisa kunjungi website <https://start.spring.io/> dengan memilih project maven, language java, spring boot 3.2.1, java 17, dan menambahkan tiga dependencies.



### 2. Library yang digunakan

#### 2.1 Spring Framework

1. org.springframework.beans.factory.annotation.Autowired  
➔ Untuk mengindikasikan penggunaan fitur "Dependency Injection" dari Spring Framework.
2. org.springframework.stereotype.Service  
➔ Mengindikasikan bahwa kelas ini adalah bagian dari layanan (service) Spring.
3. org.springframework.web.bind.annotation.RestController  
➔ Anotasi ini menandakan bahwa kelas tersebut adalah bagian dari lapisan controller (controller) Spring untuk pengembangan aplikasi web.
4. org.springframework.boot.SpringApplication  
➔ Kelas ini adalah bagian dari Spring Boot, yang menyediakan utilitas untuk memulai dan mengelola aplikasi Spring Boot.
5. org.springframework.boot.autoconfigure.SpringBootApplication

- ➔ Anotasi ini digunakan untuk menandai kelas utama aplikasi Spring Boot. Anotasi ini menggabungkan tiga anotasi lainnya: `@Configuration`, `@EnableAutoConfiguration`, dan `@ComponentScan`.

## 2.2 Spring Framework (Spring Data Jpa)

1. `org.springframework.data.jpa.repository.JpaRepository`
  - ➔ Merupakan bagian dari Spring Data JPA, yang menyediakan antarmuka.
2. `JpaRepository`
  - ➔ Untuk melakukan operasi dasar pada entitas JPA.

## 2.3 Spring Framework (Spring Core)

1. `org.springframework.stereotype.Repository`
  - ➔ Anotasi yang digunakan untuk menandai bahwa kelas tersebut adalah bagian dari lapisan repository Spring.

## 2.4 Spring MVC (Model View Controller)

1. `org.springframework.web.bind.annotation.GetMapping`
  - ➔ Anotasi yang menandakan bahwa metode tersebut menangani permintaan HTTP GET.
2. `org.springframework.web.bind.annotation.PostMapping`
  - ➔ Anotasi yang menandakan bahwa metode tersebut menangani permintaan HTTP POST.
3. `org.springframework.web.bind.annotation.PutMapping`
  - ➔ Anotasi yang menandakan bahwa metode tersebut menangani permintaan HTTP PUT.
4. `org.springframework.web.bind.annotation.DeleteMapping`
  - ➔ Anotasi yang menandakan bahwa metode tersebut menangani permintaan HTTP DELETE.
5. `org.springframework.web.bind.annotation.RequestMapping`
  - ➔ Anotasi ini digunakan untuk menetapkan jalur dasar untuk semua metode di dalam kelas ini.

## 2.5 Jakarta Transaction API (JTA)

1. `jakarta.transaction.Transactional`
  - ➔ Anotasi ini digunakan dalam konteks transaksi dan biasanya digunakan bersama dengan JTA.

## 2.6 Java Standard Library

1. `java.util.Optional`
  - ➔ Kelas ini termasuk dalam Java Standard Library dan digunakan untuk menyatakan opsional nilai, yang sesuai dengan praktik baik dalam Java modern.
2. `java.util.List`

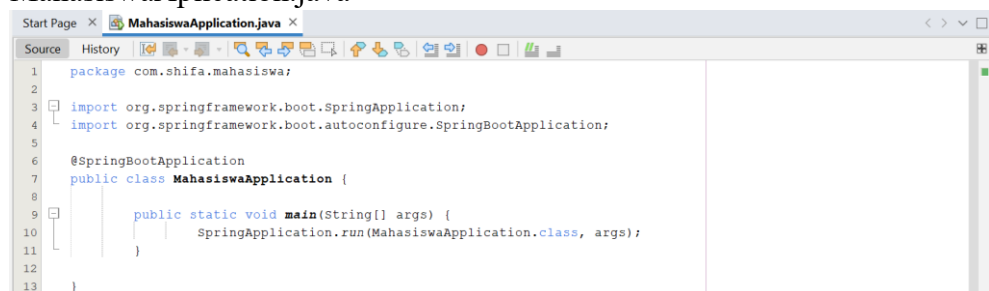
- ➔ Kelas ini termasuk dalam Java Standard Library dan digunakan untuk merepresentasikan daftar objek.

## 2.7 Java Persistence API

1. jakarta.persistence.Entity
  - ➔ Anotasi ini menandakan bahwa kelas tersebut adalah entitas JPA atau Jakarta Persistence API, yang dapat dipersistensi ke dalam database.
2. jakarta.persistence.GeneratedValue
  - ➔ Anotasi ini menandakan bahwa nilai dari suatu atribut (biasanya yang berfungsi sebagai kunci utama) akan dihasilkan secara otomatis oleh database.
3. jakarta.persistence.GenerationType
  - ➔ Enumerasi yang menyediakan strategi identitas (IDENTITY), sequence (SEQUENCE), atau tabel (TABLE) untuk menghasilkan nilai kunci.
4. jakarta.persistence.Id
  - ➔ Anotasi ini menandakan bahwa suatu atribut adalah identitas utama (primary key) dari entitas.
5. jakarta.persistence.Table
  - ➔ Anotasi ini dapat digunakan untuk menyesuaikan konfigurasi tabel database yang sesuai dengan entitas.

## 3. Kode Program

### 3.1 MahasiswaApplication.java



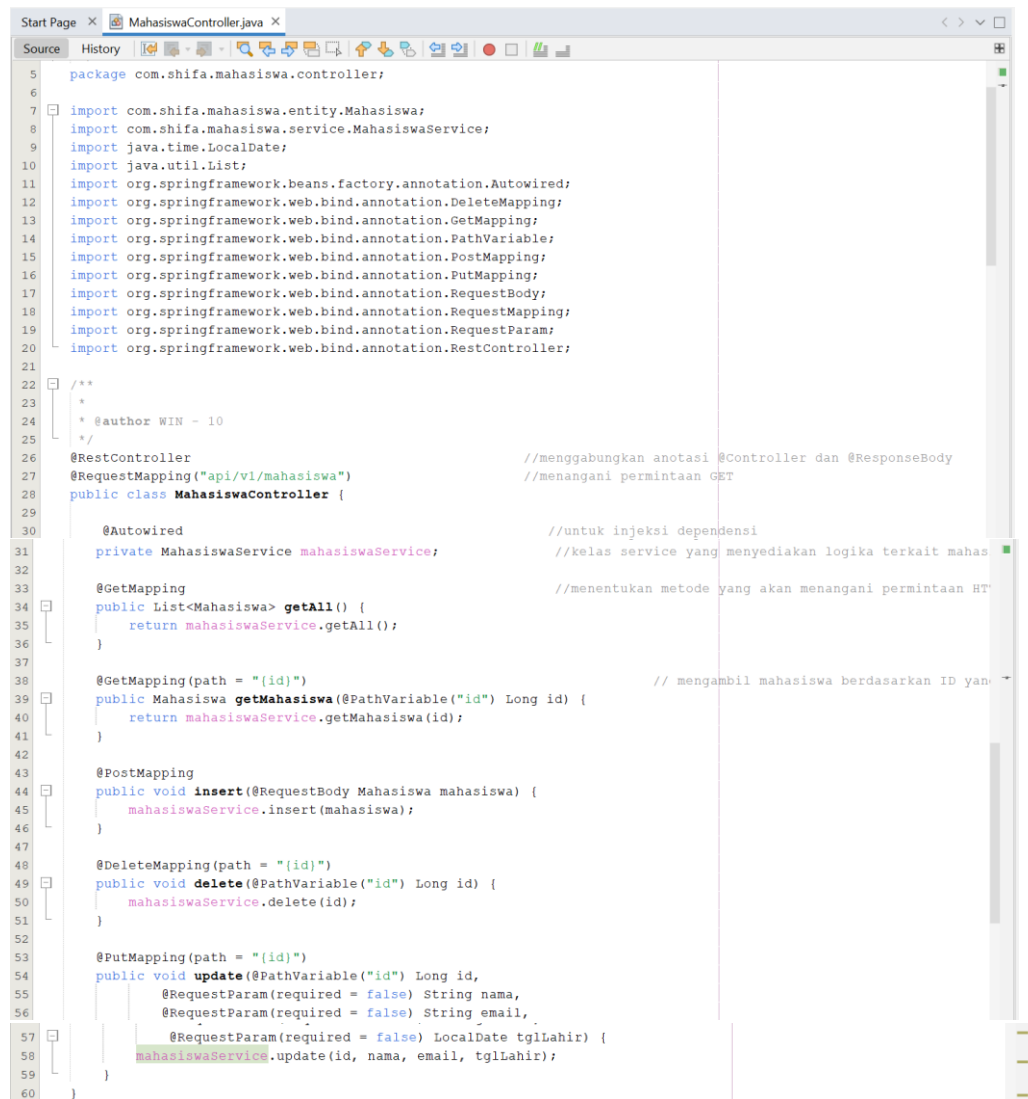
```
1 package com.shifa.mahasiswa;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class MahasiswaApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(MahasiswaApplication.class, args);
11     }
12
13 }
```

1. package com.shifa.mahasiswa
  - ➔ Menunjukkan bahwa kelas ini berada dalam package dengan nama "com.shifa.mahasiswa".
2. import org.springframework.boot.SpringApplication
  - ➔ Mengimpor kelas SpringApplication dari paket org.springframework.boot.
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
  - ➔ Mengimpor anotasi @SpringBootApplication dari paket org.springframework.boot.autoconfigure.
4. @SpringBootApplication Anotasi ini menyatukan tiga anotasi lainnya, yaitu @Configuration, @EnableAutoConfiguration, dan @ComponentScan. Ini menandakan bahwa kelas ini adalah kelas konfigurasi Spring Boot, mengaktifkan konfigurasi otomatis Spring Boot,

dan memberikan pemindaian komponen untuk mendeteksi dan mendaftarkan komponen Spring di paket ini dan paket-paket yang ada di bawahnya.

5. Kelas ini merupakan kelas utama dari aplikasi Spring Boot. Nama kelas ini mengikuti konvensi dengan menambahkan kata "Application" pada nama proyek atau aplikasi.
6. Metode main merupakan titik masuk utama untuk menjalankan aplikasi. Dalam metode ini, `SpringApplication.run(...)` digunakan untuk memulai dan menjalankan aplikasi Spring Boot. Parameter pertama adalah kelas utama (`MahasiswaApplication.class`), dan parameter kedua adalah argumen baris perintah (`args`).
7. Dengan adanya anotasi `@SpringBootApplication`, Spring Boot secara otomatis akan mengkonfigurasi dan memulai aplikasi.

### 3.2 Controller



```
package com.shifa.mahasiswa.controller;

import com.shifa.mahasiswa.entity.Mahasiswa;
import com.shifa.mahasiswa.service.MahasiswaService;
import java.time.LocalDate;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

/**
 *
 * @author WIN - 10
 */
@RestController
@RequestMapping("api/v1/mahasiswa")
public class MahasiswaController {

    @Autowired
    private MahasiswaService mahasiswaService;

    @GetMapping
    public List<Mahasiswa> getAll() {
        return mahasiswaService.getAll();
    }

    @GetMapping(path = "{id}")
    public Mahasiswa getMahasiswa(@PathVariable("id") Long id) {
        return mahasiswaService.getMahasiswa(id);
    }

    @PostMapping
    public void insert(@RequestBody Mahasiswa mahasiswa) {
        mahasiswaService.insert(mahasiswa);
    }

    @DeleteMapping(path = "{id}")
    public void delete(@PathVariable("id") Long id) {
        mahasiswaService.delete(id);
    }

    @PutMapping(path = "{id}")
    public void update(@PathVariable("id") Long id,
        @RequestParam(required = false) String nama,
        @RequestParam(required = false) String email,
        @RequestParam(required = false) LocalDate tglLahir) {
        mahasiswaService.update(id, nama, email, tglLahir);
    }
}
```

1. Import statements Mengimpor kelas-kelas dan anotasi yang diperlukan untuk pengembangan aplikasi Spring Boot, termasuk beberapa kelas dari

paket `org.springframework.web.bind.annotation` dan kelas Mahasiswa dari paket aplikasi sendiri.

2. `@RestController`
  - ➔ Anotasi yang menandakan bahwa kelas ini adalah controller Spring, yang akan menangani permintaan HTTP dan memberikan respons dalam bentuk data JSON.
3. `@RequestMapping("api/v1/mahasiswa")`
  - ➔ Anotasi untuk menetapkan awalan URI untuk setiap metode di dalam kelas ini.
4. `@Autowired`
  - ➔ `private MahasiswaService mahasiswaService;` Menggunakan fitur Dependency Injection Spring untuk menyuntikkan instance `MahasiswaService` ke dalam kelas ini.
5. `@GetMapping`
  - ➔ Menangani permintaan HTTP GET untuk mendapatkan daftar semua mahasiswa.
6. `@GetMapping(path = "{id}")`
  - ➔ Menangani permintaan HTTP GET untuk mendapatkan informasi mahasiswa berdasarkan ID.
7. `@PostMapping`
  - ➔ Menangani permintaan HTTP POST untuk menambahkan mahasiswa baru.
8. `@DeleteMapping`
  - ➔ Menangani permintaan HTTP DELETE untuk menghapus mahasiswa berdasarkan ID.
9. `@PutMapping`
  - ➔ Menangani permintaan HTTP PUT untuk memperbarui informasi mahasiswa berdasarkan ID. Menggunakan `@RequestParam` untuk menerima parameter nama dan email dari permintaan.
10. Menggunakan metode-metode dari kelas `MahasiswaService` (yang disuntikkan melalui Dependency Injection) untuk melaksanakan operasi terkait data mahasiswa

### 3.3 Entity

Nama : Arshifa Demuna  
NIM : 2111081002  
Kelas : TRPL 3C

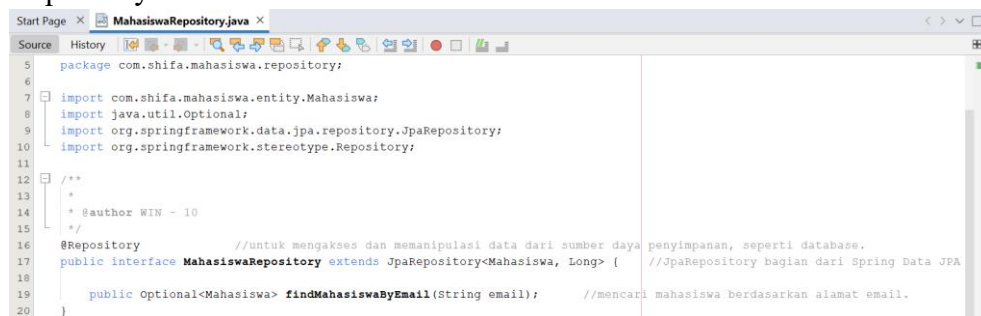
Sistem Terdistribusi

```
Start Page x Mahasiswajava x
Source History
5 package com.shifa.mahasiswa.entity;
6
7 import jakarta.persistence.Entity;
8 import jakarta.persistence.GeneratedValue;
9 import jakarta.persistence.GenerationType;
10 import jakarta.persistence.Id;
11 import jakarta.persistence.Table;
12 import java.time.LocalDate; // Import the LocalDate class
13
14 /**
15  *
16  * @author WIN - 10
17  */
18
19 @Entity //menandakan bahwa kelas ini adalah entitas JPA, yang akan di-map
20 @Table // menyesuaikan nama tabel dalam database.
21 public class Mahasiswa {
22     @Id //atribut id adalah kunci utama (primary key) dari entitas.
23     @GeneratedValue(strategy = GenerationType.IDENTITY) //menggunakan strategi penomoran otomatis
24     private long id; //menyimpan identitas unik (primary key) dari entitas mahasiswa.
25     private String nama; //menyimpan nama mahasiswa.
26     private String email; //menyimpan alamat email mahasiswa.
27     private LocalDate tglLahir; //menyimpan tanggal lahir mahasiswa.
28
29     public Mahasiswa() { //Konstruktor tanpa parameter untuk melakukan instansiasi objek secara
30     }
31
32     public Mahasiswa(long id, String nama, String email, LocalDate tglLahir) { //Konstruktor dengan parameter
33         this.id = id;
34         this.nama = nama;
35         this.email = email;
36         this.tglLahir = tglLahir;
37     }
38
39     //Getter dan setter setiap atribut. Menyediakan akses untuk membaca dan menulis nilai dari setiap atribut.
40     public long getId() {
41         return id;
42     }
43     public void setId(long id) {
44         this.id = id;
45     }
46     public String getNama() {
47         return nama;
48     }
49     public void setNama(String nama) {
50         this.nama = nama;
51     }
52     public String getEmail() {
53         return email;
54     }
55     public void setEmail(String email) {
56         this.email = email;
57     }
58     public LocalDate getTglLahir() {
59         return tglLahir;
60     }
61     public void setTglLahir(LocalDate tglLahir) {
62         this.tglLahir = tglLahir;
63     }
64     @Override //di-override dari kelas Object untuk memberikan representasi string dari objek Mahasiswa.
65     public String toString() {
66         return "Mahasiswa[" + "id=" + id + ", nama=" + nama + ", email=" + email + ", tglLahir=" + tglLahir + "]"
67     }
68 }
```

1. Import statements Mengimpor anotasi dan kelas-kelas dari paket jakarta.persistence yang digunakan untuk mendefinisikan entitas dan propertinya.
2. @Entity  
➔ Menandakan bahwa kelas ini adalah entitas JPA atau Jakarta Persistence API, yang dapat dipersistensi ke dalam database.
3. @Table  
➔ Anotasi ini dapat digunakan untuk menyesuaikan konfigurasi tabel database yang sesuai dengan entitas.
4. @Id  
➔ Menandakan bahwa properti di bawahnya (id) adalah identitas utama (primary key) dari entitas.
5. @GeneratedValue(strategy = GenerationType.IDENTITY)

- ➔ Menentukan bahwa nilai identitas utama akan dihasilkan secara otomatis oleh database dan mengikuti strategi identitas.
- 6. private Long id
  - ➔ Properti yang mewakili identitas utama (primary key).
- 7. private String nama
  - ➔ Properti yang mewakili nama mahasiswa.
- 8. private String email
  - ➔ Properti yang mewakili alamat email mahasiswa.
- 9. Konstruktor default (public Mahasiswa() {}) dan konstruktor parameterized (public Mahasiswa(Long id, String nama, String email) {}) untuk inisialisasi objek.
- 10. Metode-metode setter (setId, setName, setEmail) dan getter (getId, getName, getEmail) untuk mengakses dan mengubah nilai properti.
- 11. Override metode toString untuk memberikan representasi string dari objek Mahasiswa. Digunakan untuk debugging atau keperluan log.

### 3.4 Repository



```
1 package com.shifa.mahasiswa.repository;
2
3 import com.shifa.mahasiswa.entity.Mahasiswa;
4 import java.util.Optional;
5 import org.springframework.data.jpa.repository.JpaRepository;
6 import org.springframework.stereotype.Repository;
7
8 /**
9  *
10  * @author WIN - 10
11  */
12
13 @Repository //untuk mengakses dan memanipulasi data dari sumber daya penyimpanan, seperti database.
14 public interface MahasiswaRepository extends JpaRepository<Mahasiswa, Long> { //JpaRepository bagian dari Spring Data JPA
15
16     public Optional<Mahasiswa> findMahasiswaByEmail(String email); //mencari mahasiswa berdasarkan alamat email.
17
18 }
19
20 }
```

1. Import statements Mengimpor beberapa kelas yang diperlukan, termasuk Optional dari java.util dan beberapa kelas dari Spring Data JPA.
2. @Repository  
Anotasi yang menandakan bahwa kelas ini adalah komponen Spring dan akan diidentifikasi sebagai bean yang dapat dielusidasi secara otomatis oleh Spring Framework.
3. public interface MahasiswaRepository extends JpaRepository<Mahasiswa, Long> Antarmuka yang menggambarkan repositori untuk entitas Mahasiswa. Memperluas antarmuka JpaRepository yang disediakan oleh Spring Data JPA. Antarmuka ini menetapkan bahwa entitas adalah Mahasiswa dan kunci utamanya adalah Long.
4. public Optional<Mahasiswa> findMahasiswaByEmail(String email); Deklarasi metode kustom untuk mencari entitas Mahasiswa berdasarkan alamat email mengembalikan objek.

### 3.5 Service



```
package com.shifa.mahasiswa.service;

import com.shifa.mahasiswa.entity.Mahasiswa;
import com.shifa.mahasiswa.repository.MahasiswaRepository;
import jakarta.transaction.Transactional;
import java.time.LocalDate;
import java.util.List;
import java.util.Objects;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

/**
 * @author WIN - 10
 */
@Service //menandakan bahwa kelas ini adalah bagian dari layer service dalam arsitektur aplikasi Spring
public class MahasiswaService {

    @Autowired
    private MahasiswaRepository mahasiswaRepository; //interface Spring Data JPA operasi* dasar entitas didatabase.

    public List<Mahasiswa> getAll() {
        return mahasiswaRepository.findAll(); //mengembalikan daftar semua mahasiswa dari database
    }

    public Mahasiswa getMahasiswa(Long id) {
        Optional<Mahasiswa> mahasiswaOptional = mahasiswaRepository.findById(id); //mengembalikan objek Mahasiswa berdasar ID
        return mahasiswaOptional.get();
    }

    public void insert(Mahasiswa mahasiswa) { //memasukkan mahasiswa baru ke dalam database.
        Optional<Mahasiswa> mahasiswaOptional
            = mahasiswaRepository.findMahasiswaByEmail(mahasiswa.getEmail());
        if (mahasiswaOptional.isPresent()) {
            throw new IllegalStateException("Email sudah ada");
        }
        mahasiswaRepository.save(mahasiswa);
    }

    public void delete(Long id) { //menghapus mahasiswa berdasarkan ID yang diberikan dari database
        boolean ada = mahasiswaRepository.existsById(id);
        if (!ada) {
            throw new IllegalStateException("Mahasiswa ini tidak ada");
        }
        mahasiswaRepository.deleteById(id);
    }

    @Transactional //menandakan bahwa metode update di bawahnya harus dijalankan dalam suatu transaksi
    public void update(Long id, String nama, String email, LocalDate tglLahir) { //memperbarui data mahasiswa berdasarkan ID
        Mahasiswa mahasiswa = mahasiswaRepository.findById(id)
            .orElseThrow(() -> new IllegalStateException("Mahasiswa tidak ada"));

        //memeriksa setiap atribut (nama, email, tanggal lahir) apakah perlu diperbarui. jika ada perubahan, akan diperbarui
        if (nama != null && nama.length() > 0 && !Objects.equals(mahasiswa.getNama(), nama)) {
            mahasiswa.setNama(nama);
        }

        if (email != null && email.length() > 0 && !Objects.equals(mahasiswa.getEmail(), email)) {
            mahasiswa.setEmail(email);
        }

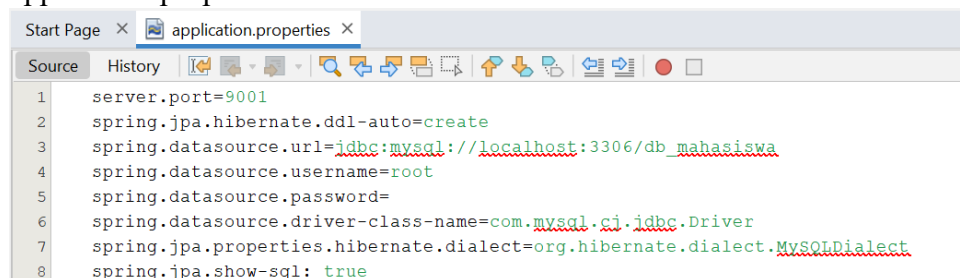
        if (tglLahir != null && !Objects.equals(mahasiswa.getTglLahir(), tglLahir)) {
            mahasiswa.setTglLahir(tglLahir);
        }
    }
}
```

1. Import statements mengimpor beberapa kelas yang diperlukan, termasuk List, Objects, Optional, dan anotasi @Autowired, @Service, dan @Transactional dari Spring Framework.
2. @Service
  - ➔ Anotasi yang menandakan bahwa kelas ini adalah komponen layanan Spring, yang akan dielusidasi secara otomatis dan dapat digunakan dalam konteks aplikasi Spring.
3. public class MahasiswaService Deklarasi kelas MahasiswaService.
4. @Autowired
  - ➔ private MahasiswaRepository mahasiswaRepository;  
Menggunakan fitur Dependency Injection Spring untuk menyuntikkan instance MahasiswaRepository ke dalam kelas ini.
5. public List<Mahasiswa> getAll() { ... }
  - ➔ Metode untuk mendapatkan daftar semua mahasiswa.
6. public Mahasiswa getMahasiswa(Long id) { ... }
  - ➔ Metode untuk mendapatkan informasi mahasiswa berdasarkan ID.



7. `public void insert(Mahasiswa mahasiswa) { ... }`  
Metode untuk menambahkan mahasiswa baru.
8. `public void delete(Long id) { ... }`  
Metode untuk menghapus mahasiswa berdasarkan ID.
9. `@Transactional`
  - ➔ `public void update(Long id, String nama, String email) { ... }`  
Metode untuk memperbarui informasi mahasiswa berdasarkan ID. Anotasi `@Transactional` menandakan bahwa metode ini akan dijalankan dalam sebuah transaksi.
10. Metode `insert`
  - ➔ dilakukan pengecekan apakah email mahasiswa sudah ada. Jika sudah ada, lemparkan pengecualian.
11. Metode `delete`
  - ➔ dilakukan pengecekan apakah mahasiswa dengan ID yang diberikan ada. Jika tidak ada, lemparkan pengecualian.
12. Metode `update`
  - ➔ menggunakan transaksi untuk menghindari masalah konsistensi data. Dilakukan pengecekan untuk memastikan nama dan email yang diupdate valid, dan bahwa email yang baru tidak konflik dengan email mahasiswa lain.

### 3.6 application.properties



```
1 server.port=9001
2 spring.jpa.hibernate.ddl-auto=create
3 spring.datasource.url=jdbc:mysql://localhost:3306/db_mahasiswa
4 spring.datasource.username=root
5 spring.datasource.password=
6 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
7 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
8 spring.jpa.show-sql: true
```

1. `server.port=9001` Menentukan port tempat aplikasi Spring Boot akan dijalankan. Dalam kasus ini, aplikasi akan dijalankan pada port 9001.
2. `spring.jpa.hibernate.ddl-auto=update` Mengatur strategi untuk Hibernate dalam mengelola skema database. Nilai `update` berarti Hibernate akan mencoba meng`u`update skema database sesuai dengan definisi entitas yang ada.
3. `spring.datasource.url=jdbc:mysql://localhost:3306/dbmahasiswa` URL koneksi ke database MySQL. Dalam hal ini, aplikasi akan terhubung ke database dengan nama `dbmahasiswa` yang berjalan di `localhost` pada port 3306.
4. `spring.datasource.username=root` Nama pengguna (username) yang akan digunakan untuk mengakses database MySQL. Dalam hal ini, pengguna adalah `'root'`.
5. `spring.datasource.password=` Kata sandi (password) yang akan digunakan untuk mengakses database MySQL. Dalam hal ini, password tidak diisi, yang artinya tidak ada password.

Nama : Arshifa Demuna  
NIM : 2111081002  
Kelas : TRPL 3C

Sistem Terdistribusi

6. `spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver` Nama kelas driver JDBC yang akan digunakan untuk membuat koneksi ke database MySQL
7. `spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect` Hibernate Dialect yang akan digunakan. Dalam hal ini, digunakan dialek untuk MySQL.
8. `spring.jpa.show-sql: true` Menentukan apakah Hibernate akan menampilkan pernyataan SQL yang dihasilkan di konsol atau tidak. Dengan nilai true, pernyataan SQL akan ditampilkan