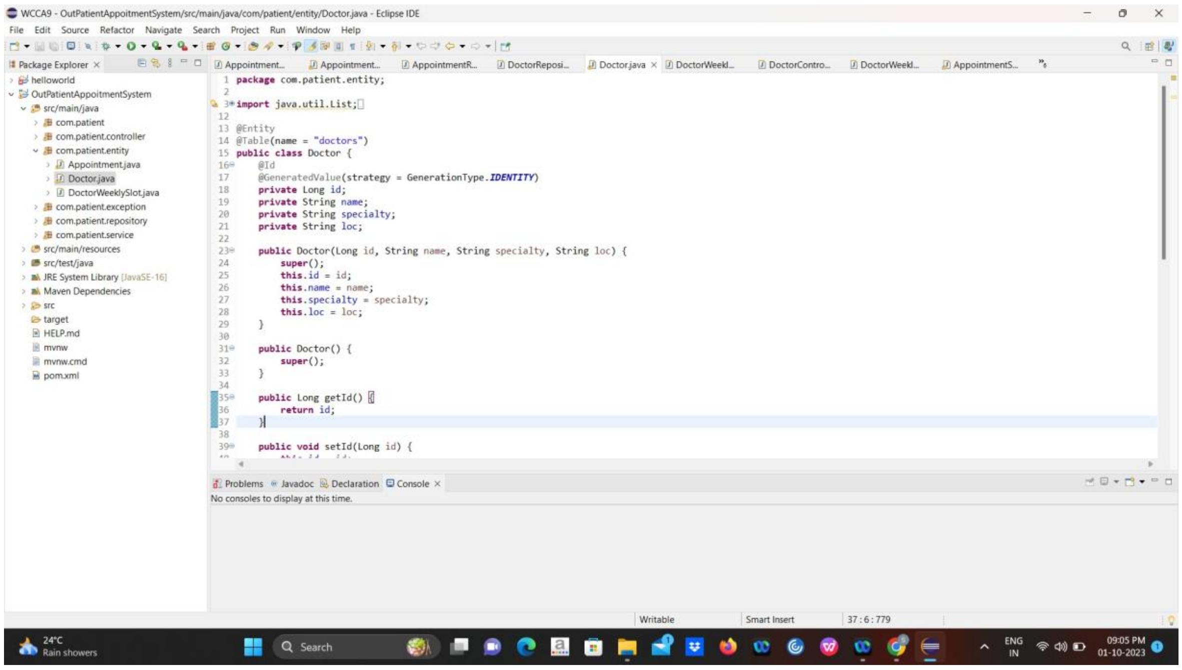


```java
package com.patient.entity;

import java.util.List;

@Entity
@Table(name = "doctors")
public class Doctor {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String specialty;
    private String loc;

    public Doctor(Long id, String name, String specialty, String loc) {
        super();
        this.id = id;
        this.name = name;
        this.specialty = specialty;
        this.loc = loc;
    }

    public Doctor() {
        super();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
```

```
No consoles to display at this time.
```

DoctorEntity:

@Entity: This annotation marks the class as a JPA entity, indicating that it should be persisted in the database.

@Table(name = "doctors"): This annotation specifies the name of the database table to which this entity is mapped. In this case, it maps to a table named "doctors."

@Id: This annotation marks the primary key field of the entity.

@GeneratedValue(strategy = GenerationType.IDENTITY): This annotation specifies the generation strategy for the primary key values. In this case, it uses an auto-incrementing identity column to generate primary key values.

private Long id: This field represents the primary key of the Doctor entity.

private String name: This field represents the name of the doctor.

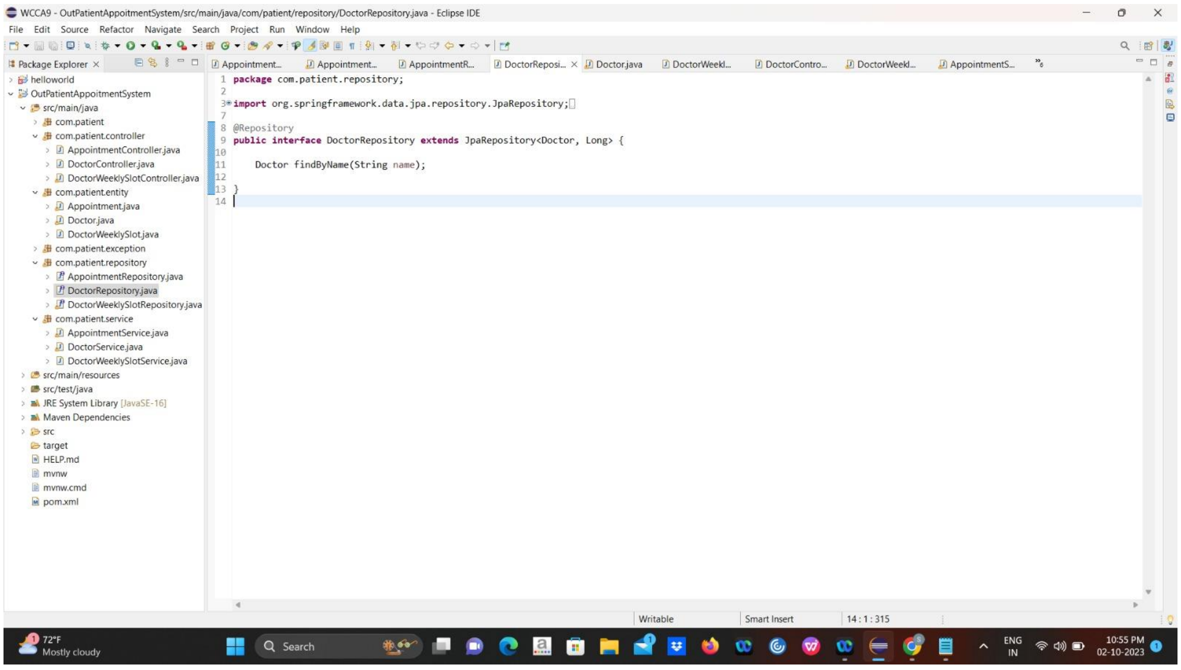
private String specialty: This field represents the specialty of the doctor.

private String loc: This field represents the location of the doctor.

Constructors:

There are two constructors provided:
The parameterized constructor with four parameters (id, name, specialty, loc) is used to create instances of the Doctor class with values for all its fields.
The default constructor with no parameters is provided for JPA compliance.
Getter and Setter methods:

Getter and setter methods are provided for all fields (id, name, specialty, loc) to access and modify their values.
@Override annotation is used to indicate that the toString method is overridden to provide a custom string representation of the Doctor object.
Overall, this Doctor class is designed to represent doctors in a database and provides the necessary annotations and methods for JPA-based database operations. It maps to a "doctors" table with fields for the doctor's ID, name, specialty, and location.



```java
package com.patient.repository;

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

@Repository
public interface DoctorRepository extends JpaRepository<Doctor, Long> {

    Doctor findByName(String name);

}
```

DoctorRepository:

The code you provided defines a Spring Data JPA repository interface named `DoctorRepository` that extends the `JpaRepository` interface. Let's break down what this code does:

1. `@Repository` Annotation: This annotation is used to indicate that the interface is a Spring Data repository. Spring Data repositories are a way to interact with a database using a more high-level and convenient API. In this case, it's a repository for the `Doctor` entity.

2. `DoctorRepository` Interface: This interface extends the `JpaRepository<Doctor, Long>` interface. The `JpaRepository` is part of Spring Data JPA and provides out-of-the-box methods for common CRUD (Create, Read, Update, Delete) operations on entities.

- `Doctor`: The first generic type parameter `Doctor` specifies the entity type that this repository is designed to manage, in this case, the `Doctor` entity.

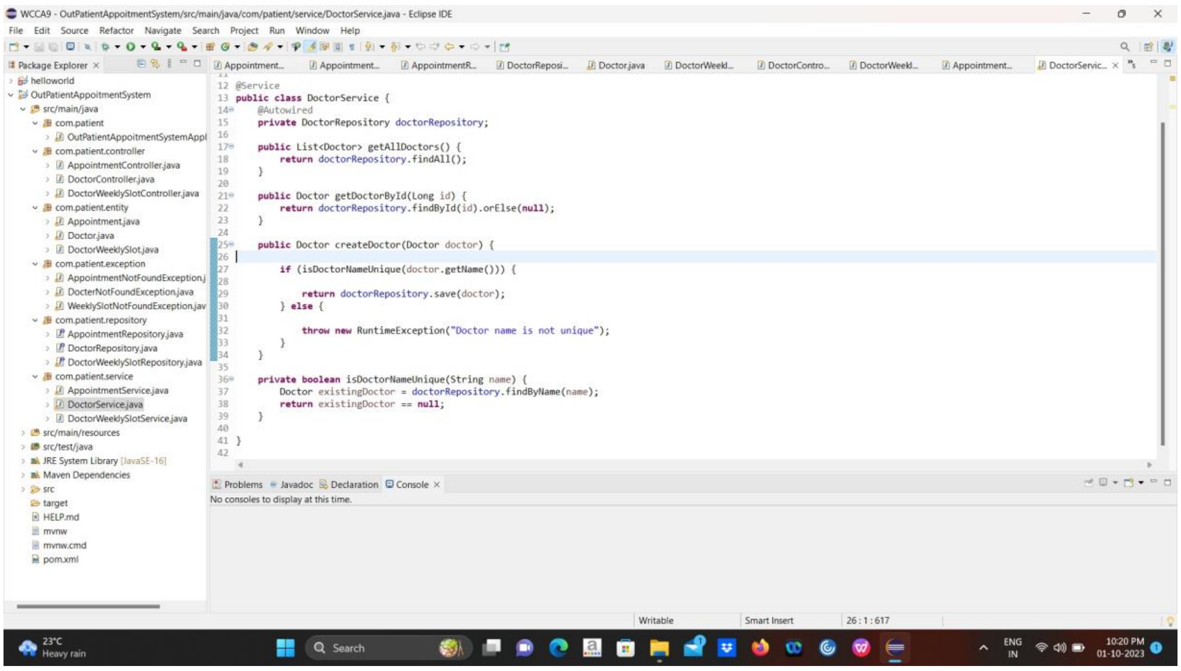
- `Long`: The second generic type parameter `Long` specifies the type of the entity's primary key, which is used as the identifier for database operations.

3. Custom Query Method: In addition to the standard CRUD methods provided by `JpaRepository`, you've defined a custom query method in this interface:

Doctor findByName(String name);

-This method allows you to find a `Doctor` entity by their name.

- The method name is derived from its signature, following the Spring Data JPA naming convention. Spring Data JPA will automatically generate the corresponding database query based on the method name, which means you don't need to write the SQL query explicitly.


WCCA9 - OutPatientAppoitmentSystem/src/main/java/com/patient/service/DoctorService.java - Eclipse IDE

```java
@Service
public class DoctorService {
    @Autowired
    private DoctorRepository doctorRepository;

    public List<Doctor> getAllDoctors() {
        return doctorRepository.findAll();
    }

    public Doctor getDoctorById(Long id) {
        return doctorRepository.findById(id).orElse(null);
    }

    public Doctor createDoctor(Doctor doctor) {

        if (isDoctorNameUnique(doctor.getName())) {

            return doctorRepository.save(doctor);
        } else {

            throw new RuntimeException("Doctor name is not unique");
        }
    }

    private boolean isDoctorNameUnique(String name) {
        Doctor existingDoctor = doctorRepository.findByName(name);
        return existingDoctor == null;
    }

}
```

No consoles to display at this time.

DoctorService:

The code you provided defines a Spring service class named DoctorService that interacts with a repository (assumed to be a Spring Data JPA repository) to perform various operations related to doctors. Let's break down what this code does:

@Service Annotation: This annotation marks the DoctorService class as a Spring service, indicating that it should be managed by the Spring framework. Services typically encapsulate business logic and handle interactions with repositories or other components.

@Autowired Annotation: This annotation is used for automatic dependency injection. It injects an instance of DoctorRepository into the DoctorService, allowing you to use it to interact with the database.

private DoctorRepository doctorRepository: This is an instance variable of type DoctorRepository, which is used to interact with the database to perform CRUD operations on Doctor entities.

getAllDoctors Method: This method retrieves a list of all doctors from the database using the doctorRepository

.findAll() method provided by Spring Data JPA's repository interface.

It returns the list of doctors to the caller.

getDoctorById Method:

This method takes a Long parameter id, which represents the ID of the doctor to be retrieved.

It uses the doctorRepository.findById(id) method to attempt to find a doctor by their ID in the database. If a doctor with the given ID is found, it returns the Doctor object; otherwise, it returns null.

createDoctor Method:

This method takes a Doctor object as a parameter, representing the doctor to be created.

It first checks whether the doctor's name is unique by calling the isDoctorNameUnique method.

If the name is unique, it saves the doctor to the database using doctorRepository.save(doctor) and returns the saved Doctor object.

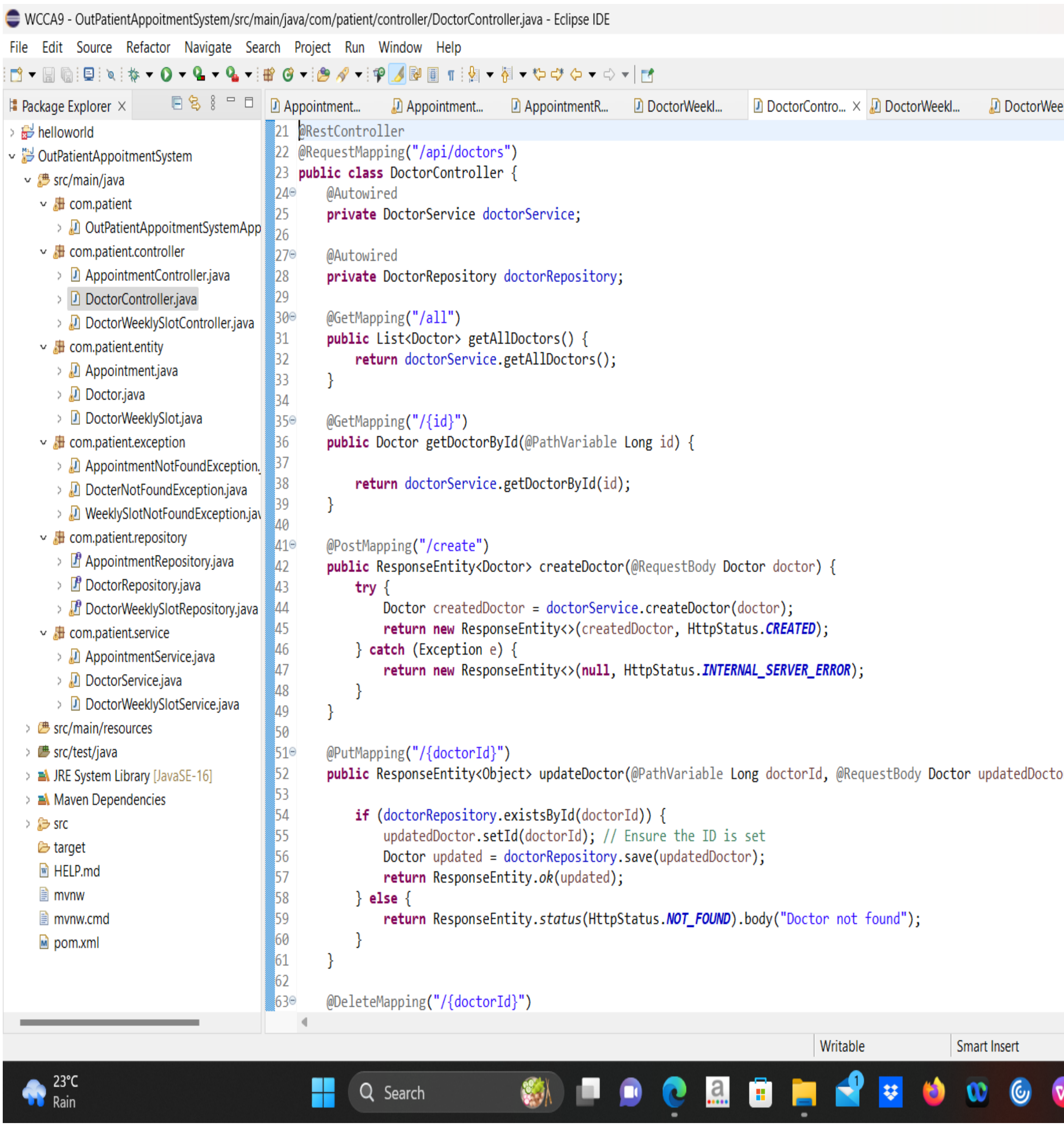
If the name is not unique, it throws a RuntimeException with the message "Doctor name is not unique."

isDoctorNameUnique Method:

This private helper method takes a String parameter name, representing the name of the doctor to be checked for uniqueness.

It uses the doctorRepository.findByName(name) method to attempt to find a doctor with the given name in the database.

If a doctor with the same name is found, it returns false, indicating that the name is not unique. Otherwise, it returns true, indicating that the name is unique.



```java
@RestController
@RequestMapping("/api/doctors")
public class DoctorController {
    @Autowired
    private DoctorService doctorService;

    @Autowired
    private DoctorRepository doctorRepository;

    @GetMapping("/all")
    public List<Doctor> getAllDoctors() {
        return doctorService.getAllDoctors();
    }

    @GetMapping("/{id}")
    public Doctor getDoctorById(@PathVariable Long id) {

        return doctorService.getDoctorById(id);
    }

    @PostMapping("/create")
    public ResponseEntity<Doctor> createDoctor(@RequestBody Doctor doctor) {
        try {
            Doctor createdDoctor = doctorService.createDoctor(doctor);
            return new ResponseEntity<>(createdDoctor, HttpStatus.CREATED);
        } catch (Exception e) {
            return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @PutMapping("/{doctorId}")
    public ResponseEntity<Object> updateDoctor(@PathVariable Long doctorId, @RequestBody Doctor updatedDocto

        if (doctorRepository.existsById(doctorId)) {
            updatedDoctor.setId(doctorId); // Ensure the ID is set
            Doctor updated = doctorRepository.save(updatedDoctor);
            return ResponseEntity.ok(updated);
        } else {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Doctor not found");
        }
    }

    @DeleteMapping("/{doctorId}")
```

DoctorController

The provided code defines a Spring Boot RESTful controller class named `DoctorController` that handles HTTP requests related to doctor entities. This controller interacts with the `DoctorService` and `DoctorRepository` to perform CRUD (Create, Read, Update, Delete) operations on doctor data. Let's break down what each method in this controller does:

1. `@RestController` and `@RequestMapping("/api/doctors")`: These annotations define the class as a REST controller and specify the base URL path for all the controller's endpoints. In this case, all endpoints are under "/api/doctors."

2. `@Autowired` Annotations: These annotations are used for automatic dependency injection of the `DoctorService` and `DoctorRepository` instances into the controller.

3. `getAllDoctors` Method (HTTP GET):

- This method handles GET requests to "/api/doctors/all."

- It calls the `getAllDoctors` method of the `DoctorService` to retrieve a list of all doctors from the database.

- It returns the list of doctors as a JSON response to the client.

4. `getDoctorById` Method (HTTP GET):

   - This method handles GET requests to "/api/doctors/{id}" where `{id}` is a path variable representing the ID of the doctor to retrieve.
   - It calls the `getDoctorById` method of the `DoctorService` to retrieve a doctor by their ID from the database.
   - It returns the retrieved doctor as a JSON response to the client.

5. `createDoctor` Method (HTTP POST):

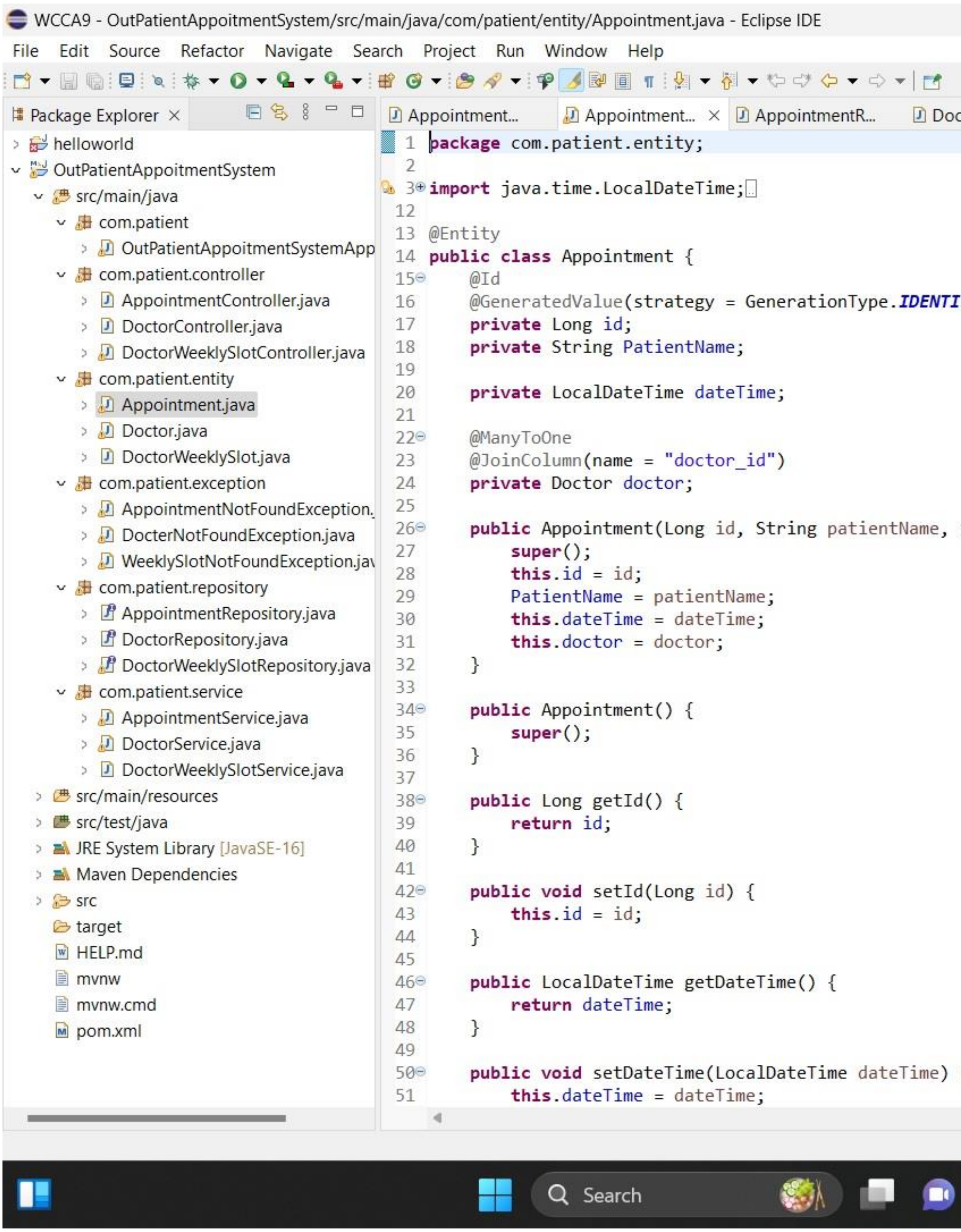
   - This method handles POST requests to "/api/doctors/create."
   - It takes a JSON request body containing a `Doctor` object, representing the doctor to be created.
   - It calls the `createDoctor` method of the `DoctorService` to create the doctor while ensuring the name is unique.
   - If the creation is successful, it returns a JSON response with the created doctor and a HTTP status code of 201 (Created). If there's an error (e.g., non-unique name), it returns a 500 (Internal Server Error) status code.

6. `updateDoctor` Method (HTTP PUT):

   - This method handles PUT requests to "/api/doctors/{doctorId}" where `{doctorId}` is a path variable representing the ID of the doctor to update.
   - It takes a JSON request body containing an updated `Doctor` object.
   - It checks if a doctor with the specified ID exists using `doctorRepository.existsById(doctorId)`.
   - If the doctor exists, it updates the doctor's information (including the ID) and saves it to the database using `doctorRepository.save(updatedDoctor)`. It returns the updated doctor as a JSON response with a 200 (OK) status code.
   - If the doctor with the specified ID does not exist, it returns a 404 (Not Found) status code with an error message.

7. `deleteDoctor` Method (HTTP DELETE):

   - This method handles DELETE requests to "/api/doctors/{doctorId}" where `{doctorId}` is a path variable representing the ID of the doctor to delete.
   - It checks if a doctor with the specified ID exists using `doctorRepository.existsById(doctorId)`.
   - If the doctor exists, it deletes the doctor from the database using `doctorRepository.deleteById(doctorId)` and returns a 204 (No Content) status code, indicating a successful deletion.
   - If the doctor with the specified ID does not exist, it returns a 404 (Not Found) status code.



WCCA9 - OutPatientAppoitmentSystem/src/main/java/com/patient/entity/Appointment.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Appointment...
Appointment...
AppointmentR...
Doc
Search

```
helloworld
OutPatientAppoitmentSystem
  src/main/java
    com.patient
      OutPatientAppoitmentSystemApp
    com.patient.controller
      AppointmentController.java
      DoctorController.java
      DoctorWeeklySlotController.java
    com.patient.entity
      Appointment.java
      Doctor.java
      DoctorWeeklySlot.java
    com.patient.exception
      AppointmentNotFoundException.
      DocterNotFoundException.java
      WeeklySlotNotFoundException.jav
    com.patient.repository
      AppointmentRepository.java
      DoctorRepository.java
      DoctorWeeklySlotRepository.java
    com.patient.service
      AppointmentService.java
      DoctorService.java
      DoctorWeeklySlotService.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-16]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

```java
package com.patient.entity;

import java.time.LocalDateTime;

@Entity
public class Appointment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTI
    private Long id;
    private String PatientName;

    private LocalDateTime dateTime;

    @ManyToOne
    @JoinColumn(name = "doctor_id")
    private Doctor doctor;

    public Appointment(Long id, String patientName, 
        super();
        this.id = id;
        PatientName = patientName;
        this.dateTime = dateTime;
        this.doctor = doctor;
    }

    public Appointment() {
        super();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public LocalDateTime getDateTime() {
        return dateTime;
    }

    public void setDateTime(LocalDateTime dateTime)
        this.dateTime = dateTime;
```

Appoiment entity

@Entity Annotation:

The @Entity annotation is used to indicate that this class represents a JPA (Java Persistence API) entity. In other words, instances of this class can be persisted to a database table.

@Id Annotation: The @Id annotation is used to mark the id field as the primary key of the entity. This field

uniquely identifies each Appointment instance.

@GeneratedValue Annotation:

The @GeneratedValue annotation is used to specify how the primary key values are generated. In this case, GenerationType.IDENTITY indicates that the primary key values are automatically generated by the database.

Instance Variables (Fields):

The class defines several instance variables, including id, PatientName, dateTime, and doctor. These variables represent properties of an appointment entity.

@ManyToOne Annotation:

The @ManyToOne annotation is used to establish a many-to-one relationship between Appointment and Doctor entities. This means that many appointments can be associated with one doctor.

@JoinColumn Annotation:

The @JoinColumn annotation is used to specify the name of the foreign key column in the database table that references the Doctor entity. In this case, it specifies that the foreign key column is named "doctor_id."

Constructors:

The class provides two constructors: a parameterized constructor and a default (no-argument) constructor. The parameterized constructor allows you to create instances of Appointment with values for all fields.

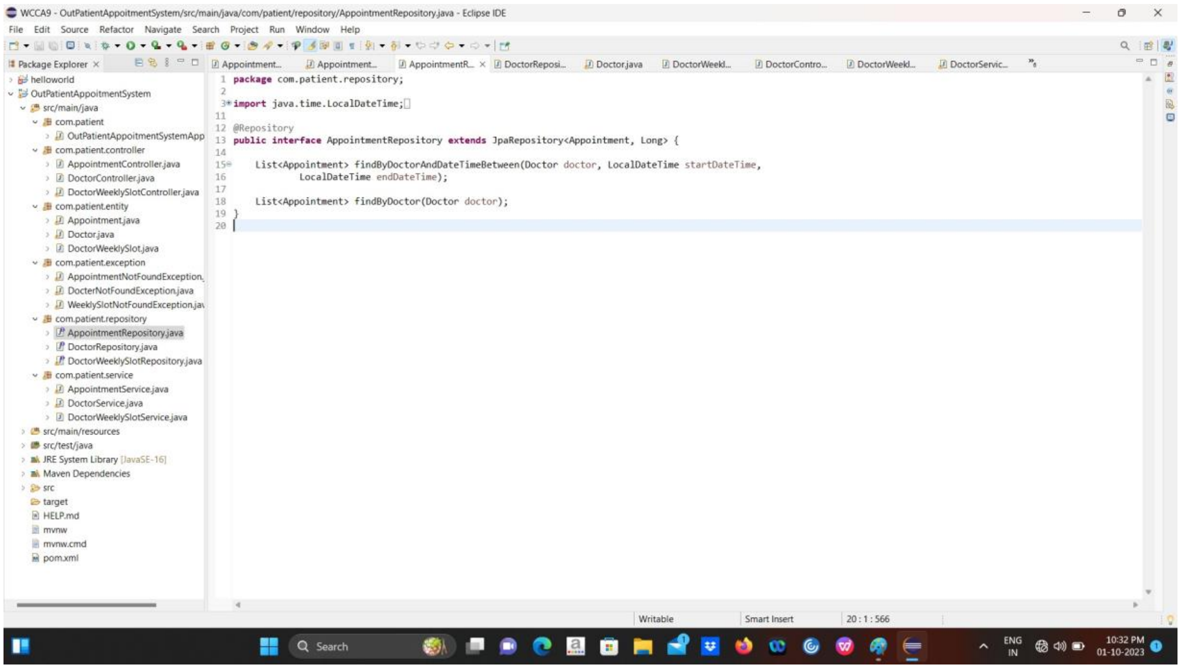
Getter and Setter Methods:

Getter methods are provided for accessing the values of the instance variables.

Setter methods are provided for modifying the values of the instance variables.

toString() Method:

The toString() method is overridden to provide a textual representation of an Appointment instance. It returns a string containing the values of all instance variables.



WCCA9 - OutPatientAppoitmentSystem/src/main/java/com/patient/repository/AppointmentRepository.java - Eclipse IDE

```java
package com.patient.repository;

import java.time.LocalDateTime;

@Repository
public interface AppointmentRepository extends JpaRepository<Appointment, Long> {

    List<Appointment> findByDoctorAndDateTimeBetween(Doctor doctor, LocalDateTime startDateTime,
            LocalDateTime endDateTime);

    List<Appointment> findByDoctor(Doctor doctor);
}
```

AppoimentRepository

@Repository Annotation:

The @Repository annotation marks this interface as a Spring Data repository. Spring Data repositories are used to interact with a database and perform CRUD (Create, Read, Update, Delete) operations on entities.

extends JpaRepository<Appointment, Long>:The AppointmentRepository interface extends the JpaRepository interface, which is provided by Spring Data JPA. By extending this interface, the AppointmentRepository inherits a set of predefined methods for working with Appointment entities, including methods for basic CRUD operations.

findByDoctorAndDateTimeBetween Method:

This method is used to retrieve a list of appointments based on specific criteria.

It takes the following parameters:

Doctor doctor: A Doctor object representing the doctor for whom the appointments are being searched.

LocalDateTime startDateTime: The start date and time for the search range.

LocalDateTime endDateTime: The end date and time for the search range.

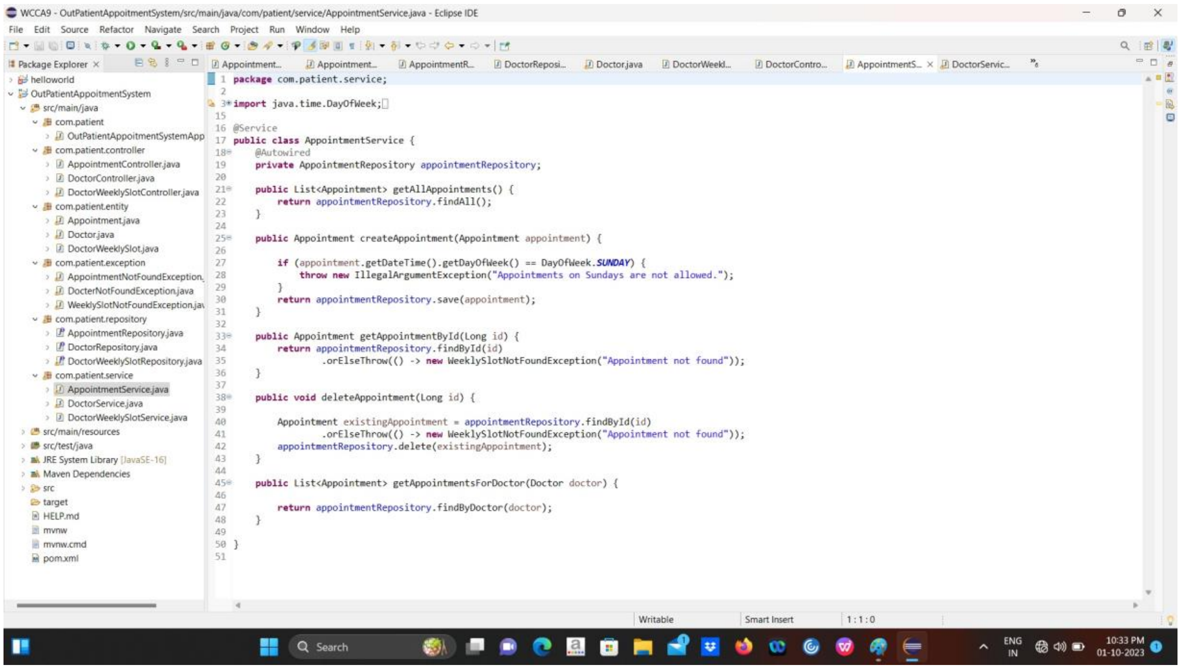
The method name follows a specific naming convention (findByXAndYBetween) that Spring Data JPA recognizes. It specifies that the query should filter by both the doctor and a date-time range between startDateTime and endDateTime.

findByDoctor Method:

This method is used to retrieve a list of appointments for a specific doctor.

It takes a single parameter: Doctor doctor, which represents the doctor for whom the appointments are being searched.

The method name, findByDoctor, indicates that the query should filter by the specified doctor.



```java
package com.patient.service;

import java.time.DayOfWeek;

@Service
public class AppointmentService {
    @Autowired
    private AppointmentRepository appointmentRepository;

    public List<Appointment> getAllAppointments() {
        return appointmentRepository.findAll();
    }

    public Appointment createAppointment(Appointment appointment) {

        if (appointment.getDateTime().getDayOfWeek() == DayOfWeek.SUNDAY) {
            throw new IllegalArgumentException("Appointments on Sundays are not allowed.");
        }
        return appointmentRepository.save(appointment);
    }

    public Appointment getAppointmentById(Long id) {
        return appointmentRepository.findById(id)
                .orElseThrow(() -> new WeeklySlotNotFoundException("Appointment not found"));
    }

    public void deleteAppointment(Long id) {

        Appointment existingAppointment = appointmentRepository.findById(id)
                .orElseThrow(() -> new WeeklySlotNotFoundException("Appointment not found"));
        appointmentRepository.delete(existingAppointment);
    }

    public List<Appointment> getAppointmentsForDoctor(Doctor doctor) {

        return appointmentRepository.findByDoctor(doctor);
    }

}
```

AppoimentService:

@Service Annotation:The @Service annotation marks this class as a Spring service component. Spring services are typically used to encapsulate business logic and are managed by the Spring framework.

@Autowired Annotation:

The @Autowired annotation is used to inject the AppointmentRepository dependency into the service. This allows the service to access and use the repository to interact with the database.

getAllAppointments Method:

This method retrieves a list of all appointments by calling the findAll method provided by the repository.

createAppointment Method:

This method creates a new appointment by saving it to the database using the save method from the repository.

Before saving, it checks if the appointment's date and time fall on a Sunday. If it does, an exception of type IllegalArgumentException is thrown with an error message indicating that Sunday appointments are not allowed.

getAppointmentById Method:

This method retrieves an appointment by its ID using the findById method from the repository.

If the appointment with the provided ID is not found, it throws a custom exception

(WeeklySlotNotFoundException) with an error message indicating that the appointment was not found.

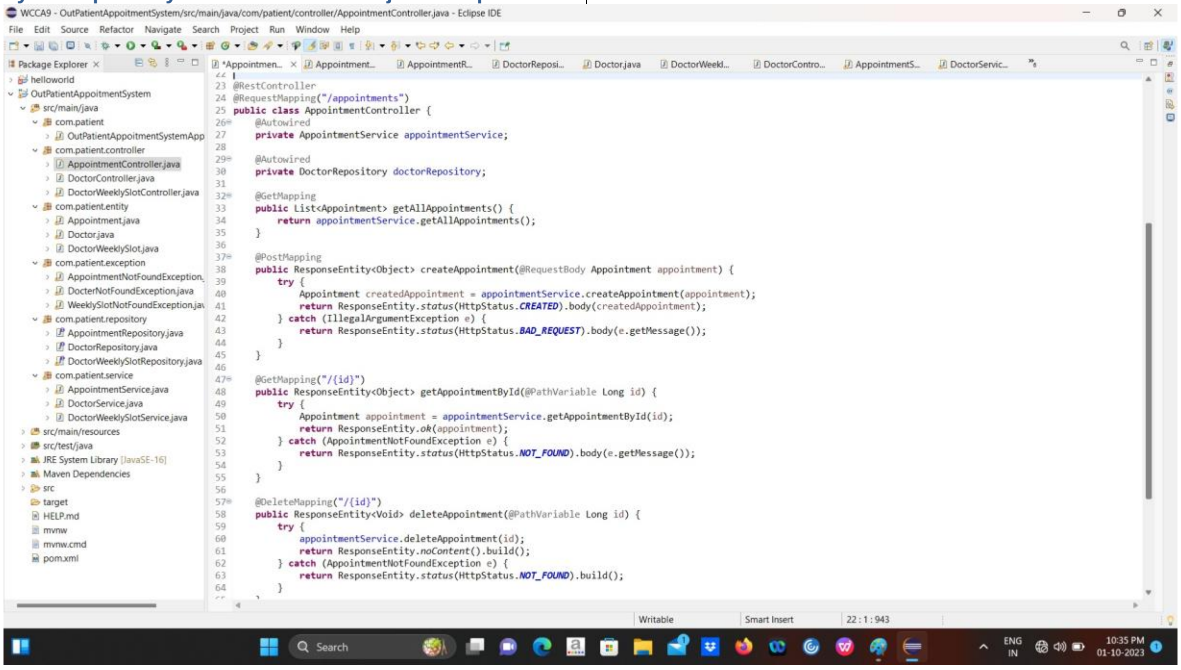
deleteAppointment Method:

This method deletes an appointment by its ID using the delete method from the repository.

If the appointment with the provided ID is not found, it throws a custom exception (WeeklySlotNotFoundException) with an error message indicating that the appointment was not found.

getAppointmentsForDoctor Method:

This method retrieves a list of appointments for a specific doctor by calling the findByDoctor method provided by the repository. It takes a Doctor object as a parameter.



```java
@RestController
@RequestMapping("/appointments")
public class AppointmentController {
    @Autowired
    private AppointmentService appointmentService;

    @Autowired
    private DoctorRepository doctorRepository;

    @GetMapping
    public List<Appointment> getAllAppointments() {
        return appointmentService.getAllAppointments();
    }

    @PostMapping
    public ResponseEntity<Object> createAppointment(@RequestBody Appointment appointment) {
        try {
            Appointment createdAppointment = appointmentService.createAppointment(appointment);
            return ResponseEntity.status(HttpStatus.CREATED).body(createdAppointment);
        } catch (IllegalArgumentException e) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());
        }
    }

    @GetMapping("/{id}")
    public ResponseEntity<Object> getAppointmentById(@PathVariable Long id) {
        try {
            Appointment appointment = appointmentService.getAppointmentById(id);
            return ResponseEntity.ok(appointment);
        } catch (AppointmentNotFoundException e) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteAppointment(@PathVariable Long id) {
        try {
            appointmentService.deleteAppointment(id);
            return ResponseEntity.noContent().build();
        } catch (AppointmentNotFoundException e) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }
```

WCCA9 - OutPatientAppoitmentSystem/src/main/java/com/patient/controller/AppointmentController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
helloworld
OutPatientAppoitmentSystem
src/main/java
com.patient
OutPatientAppoitmentSystemApp
com.patient.controller
AppointmentController.java
DoctorController.java
DoctorWeeklySlotController.java
com.patient.entity
Appointment.java
Doctor.java
DoctorWeeklySlot.java
com.patient.exception
AppointmentNotFoundException
DocterNotFoundException.java
WeeklySlotNotFoundException.java
com.patient.repository
AppointmentRepository.java
DoctorRepository.java
DoctorWeeklySlotRepository.java
com.patient.service
AppointmentService.java
DoctorService.java
DoctorWeeklySlotService.java
src/main/resources
src/test/java
JRE System Library [JavaSE-16]
Maven Dependencies
src
target
HELP.md
mvnw
mvnw.cmd
pom.xml
Writable
Smart Insert
22 : 1 : 943
Search
ENG IN
10:35 PM
01-10-2023

AppoimentController

@RestController Annotation:

The @RestController annotation is used to mark this class as a Spring RESTful controller. It indicates that this class handles HTTP requests and returns HTTP responses as JSON or XML, depending on the request and configuration.

@RequestMapping("/appointments") Annotation:

The @RequestMapping annotation at the class level specifies the base URL path for all the endpoints defined in this controller. In this case, all endpoints in this controller will start with "/appointments."

@Autowired Annotations:

The @Autowired annotation is used to inject dependencies into the controller.

appointmentService: Injects an instance of AppointmentService to handle business logic related to appointments.

doctorRepository: Injects an instance of DoctorRepository for accessing doctor information.

@GetMapping Annotation:

The @GetMapping annotation maps an HTTP GET request to the specified endpoint.

/ specifies the path for this endpoint relative to the base path defined in the class-level @RequestMapping.

This endpoint retrieves a list of all appointments by calling the getAllAppointments method in the AppointmentService.

@PostMapping Annotation:

The @PostMapping annotation maps an HTTP POST request to create a new appointment.

It expects a JSON request body containing the details of the new appointment.

If the appointment creation is successful, it returns an HTTP response with status code 201 (Created) and the created Appointment object in the response body.

If the appointment creation fails (e.g., if the appointment date is on a Sunday), it returns a response with status code 400 (Bad Request) and an error message in the response body.

@GetMapping("/{id}") Annotation:

This endpoint retrieves an appointment by its ID using an HTTP GET request.

/{id} is a path variable that represents the ID of the appointment to be retrieved.

If the appointment with the provided ID is found, it returns an HTTP response with status code 200 (OK) and the appointment object in the response body.

If the appointment is not found, it returns a response with status code 404 (Not Found) and an error message in the response body.

@DeleteMapping("/{id}") Annotation:

This endpoint deletes an appointment by its ID using an HTTP DELETE request.

/{id} is a path variable that represents the ID of the appointment to be deleted.

If the appointment with the provided ID is found and successfully deleted, it returns an HTTP response with status code 204 (No Content).

If the appointment is not found, it returns a response with status code 404 (Not Found).

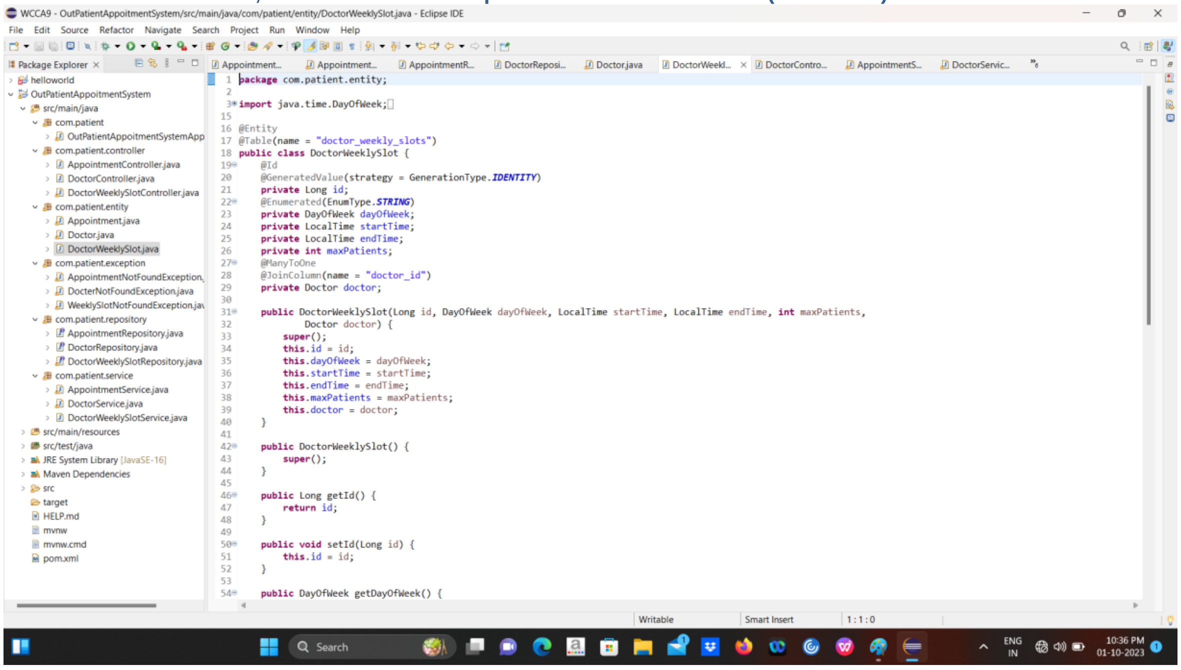
@GetMapping("/doctor/{doctorId}") Annotation:

This endpoint retrieves a list of appointments for a specific doctor based on the doctor's ID.

/doctor/{doctorId} is a path variable that represents the ID of the doctor for whom appointments are being retrieved.

If the doctor with the provided ID is found, it returns a list of appointments associated with that doctor.

If the doctor is not found, it returns an error response with status code 404 (Not Found).


WCCA9 - OutPatientAppointmentSystem/src/main/java/com/patient/entity/DoctorWeeklySlot.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
helloworld
OutPatientAppointmentSystem
src/main/java
com.patient
OutPatientAppointmentSystemApp
com.patient.controller
AppointmentController.java
DoctorController.java
DoctorWeeklySlotController.java
com.patient.entity
Appointment.java
Doctor.java
DoctorWeeklySlot.java
com.patient.exception
AppointmentNotFoundException.
DocterNotFoundException.java
WeeklySlotNotFoundException.ja
com.patient.repository
AppointmentRepository.java
DoctorRepository.java
DoctorWeeklySlotRepository.java
com.patient.service
AppointmentService.java
DoctorService.java
DoctorWeeklySlotService.java
src/main/resources
src/test/java
JRE System Library [JavaSE-16]
Maven Dependencies
src
target
HELP.md
mvnw
mvnw.cmd
pom.xml
Writable
Smart Insert
1 : 1 : 0

```java
package com.patient.entity;

import java.time.DayOfWeek;

@Entity
@Table(name = "doctor_weekly_slots")
public class DoctorWeeklySlot {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Enumerated(EnumType.STRING)
    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;
    private int maxPatients;
    @ManyToOne
    @JoinColumn(name = "doctor_id")
    private Doctor doctor;

    public DoctorWeeklySlot(Long id, DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime, int maxPatients,
            Doctor doctor) {
        super();
        this.id = id;
        this.dayOfWeek = dayOfWeek;
        this.startTime = startTime;
        this.endTime = endTime;
        this.maxPatients = maxPatients;
        this.doctor = doctor;
    }

    public DoctorWeeklySlot() {
        super();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public DayOfWeek getDayOfWeek() {
```

Doctorweeklyslot entity:

@Entity Annotation:

The @Entity annotation is used to indicate that this class represents a JPA (Java Persistence API) entity. In other words, instances of this class can be persisted to a database table.

@Table Annotation:

The @Table annotation is used to specify the details of the database table that corresponds to this entity. In this case, it specifies that the table name in the database is "doctor_weekly_slots."

@Id Annotation:

The @Id annotation is used to mark the id field as the primary key of the entity. This field uniquely identifies each DoctorWeeklySlot instance.

@GeneratedValue Annotation:

The @GeneratedValue annotation is used to specify how the primary key values are generated. In this case, GenerationType.IDENTITY indicates that the primary key values are automatically generated by the database.

@Enumerated(EnumType.STRING) Annotation:

The @Enumerated annotation is used to specify how the dayOfWeek field should be mapped to the database. EnumType.STRING indicates that the DayOfWeek enum values should be stored as strings in the database.

@ManyToOne Annotation:

The @ManyToOne annotation is used to establish a many-to-one relationship between DoctorWeeklySlot and Doctor entities. This means that many weekly slots can belong to one doctor.

@JoinColumn Annotation:

The @JoinColumn annotation is used to specify the name of the foreign key column in the doctor_weekly_slots table that references the Doctor entity. In this case, it specifies that the foreign key column is named "doctor_id."

Instance Variables (Fields):

The class defines several instance variables that represent properties of a doctor's weekly slot, including id, dayOfWeek, startTime, endTime, maxPatients, and doctor.

Constructors:

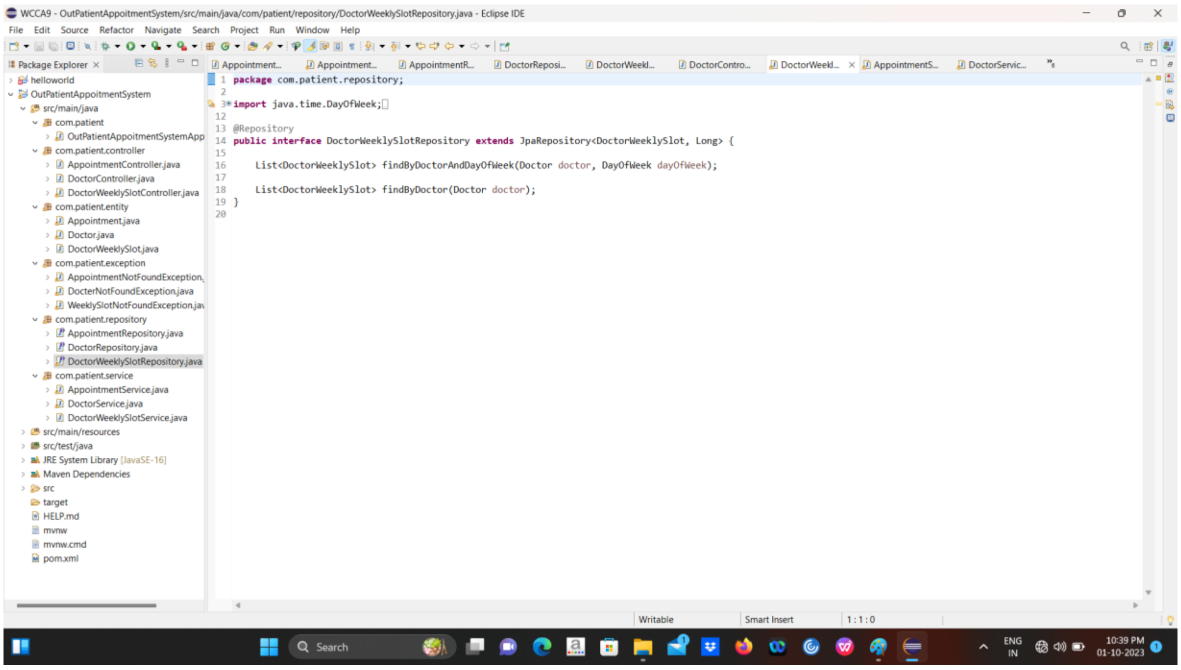
The class provides two constructors: a parameterized constructor and a default (no-argument) constructor. The parameterized constructor allows you to create instances of DoctorWeeklySlot with values for all fields.

Getter and Setter Methods:

Getter methods are provided for accessing the values of the instance variables.

Setter methods are provided for modifying the values of the instance variables.

toString() Method: The toString() method is overridden to provide a textual representation of a DoctorWeeklySlot instance. It returns a string containing the values of all instance variables.



WCCA9 - OutPatientAppoitmentSystem/src/main/java/com/patient/repository/DoctorWeeklySlotRepository.java - Eclipse IDE

```java
package com.patient.repository;

import java.time.DayOfWeek;

@Repository
public interface DoctorWeeklySlotRepository extends JpaRepository<DoctorWeeklySlot, Long> {

    List<DoctorWeeklySlot> findByDoctorAndDayOfWeek(Doctor doctor, DayOfWeek dayOfWeek);

    List<DoctorWeeklySlot> findByDoctor(Doctor doctor);
}
```

DoctorweeklyslotRepository:

@Repository Annotation:

This annotation is used to indicate that this interface is a Spring Data repository. Spring Data repositories are a convenient way to interact with data storage, particularly databases, using a simplified and consistent API. In this case, it suggests that this interface is used for interacting with DoctorWeeklySlot entities in the data storage.

public interface DoctorWeeklySlotRepository extends JpaRepository<DoctorWeeklySlot, Long> {

DoctorWeeklySlotRepository is the name of the interface.

extends JpaRepository<DoctorWeeklySlot, Long>: This interface extends JpaRepository, which is part of Spring Data JPA. It specifies two generic types:

DoctorWeeklySlot: This is the entity type that the repository manages, in this case, the DoctorWeeklySlot entity.

Long: This is the type of the primary key of the DoctorWeeklySlot entity. It indicates that the primary key is of type Long.

Repository Methods:

Inside the DoctorWeeklySlotRepository interface, you have defined two custom query methods:

List<DoctorWeeklySlot> findByDoctorAndDayOfWeek(Doctor doctor, DayOfWeek dayOfWeek);:

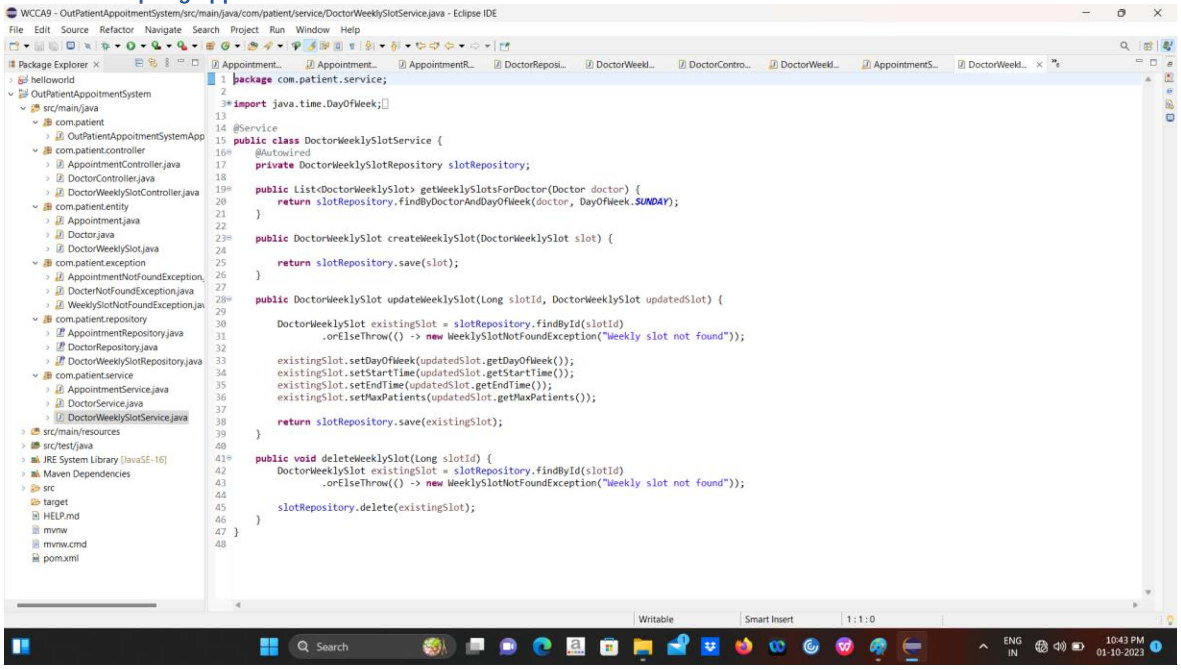
This method is used to find a list of DoctorWeeklySlot entities that match the provided Doctor and DayOfWeek. This is a custom query method with a specific naming convention. Spring Data JPA uses method name conventions to generate database queries. In this case, it will generate a query to find slots where the doctor matches the provided Doctor object and the dayOfWeek matches the provided DayOfWeek.

List<DoctorWeeklySlot> findByDoctor(Doctor doctor);:

This method is used to find a list of DoctorWeeklySlot entities associated with a specific Doctor. Again, this is a custom query method with a naming convention. It generates a query to find slots where the doctor matches the provided Doctor object.

These custom query methods make it easy to perform database operations related to DoctorWeeklySlot entities without writing SQL queries manually. Spring Data JPA will automatically generate the appropriate SQL queries based on the method names and parameters.

By extending JpaRepository, you also inherit various CRUD (Create, Read, Update, Delete) methods for managing DoctorWeeklySlot entities. This interface provides a high-level abstraction for working with data and simplifies data access in a Spring application.



```java
package com.patient.service;

import java.time.DayOfWeek;

@Service
public class DoctorWeeklySlotService {
    @Autowired
    private DoctorWeeklySlotRepository slotRepository;

    public List<DoctorWeeklySlot> getWeeklySlotsForDoctor(Doctor doctor) {
        return slotRepository.findByDoctorAndDayOfWeek(doctor, DayOfWeek.SUNDAY);
    }

    public DoctorWeeklySlot createWeeklySlot(DoctorWeeklySlot slot) {

        return slotRepository.save(slot);
    }

    public DoctorWeeklySlot updateWeeklySlot(Long slotId, DoctorWeeklySlot updatedSlot) {

        DoctorWeeklySlot existingSlot = slotRepository.findById(slotId)
                .orElseThrow(() -> new WeeklySlotNotFoundException("Weekly slot not found"));

        existingSlot.setDayOfWeek(updatedSlot.getDayOfWeek());
        existingSlot.setStartTime(updatedSlot.getStartTime());
        existingSlot.setEndTime(updatedSlot.getEndTime());
        existingSlot.setMaxPatients(updatedSlot.getMaxPatients());

        return slotRepository.save(existingSlot);
    }

    public void deleteWeeklySlot(Long slotId) {
        DoctorWeeklySlot existingSlot = slotRepository.findById(slotId)
                .orElseThrow(() -> new WeeklySlotNotFoundException("Weekly slot not found"));

        slotRepository.delete(existingSlot);
    }
}
```

DoctorWeeklyService:

@Service Annotation:

The @Service annotation marks this class as a Spring service component. Spring services are typically used to encapsulate business logic and are managed by the Spring framework.

@Autowired Annotation:

The @Autowired annotation is used to inject the DoctorWeeklySlotRepository dependency into the service. This allows the service to access and use the repository to interact with the database.

getWeeklySlotsForDoctor Method:

This method retrieves a list of doctor weekly slots for a specific doctor and the day of the week (in this case, Sunday) using the findByDoctorAndDayOfWeek method from the repository.

createWeeklySlot Method:

This method creates a new doctor weekly slot by saving it to the database using the save method from the repository.

updateWeeklySlot Method:

This method updates an existing doctor weekly slot with new information.

It first checks if the slot with the provided ID exists using the findById method from the repository. If not found, it throws a custom exception (WeeklySlotNotFoundException).

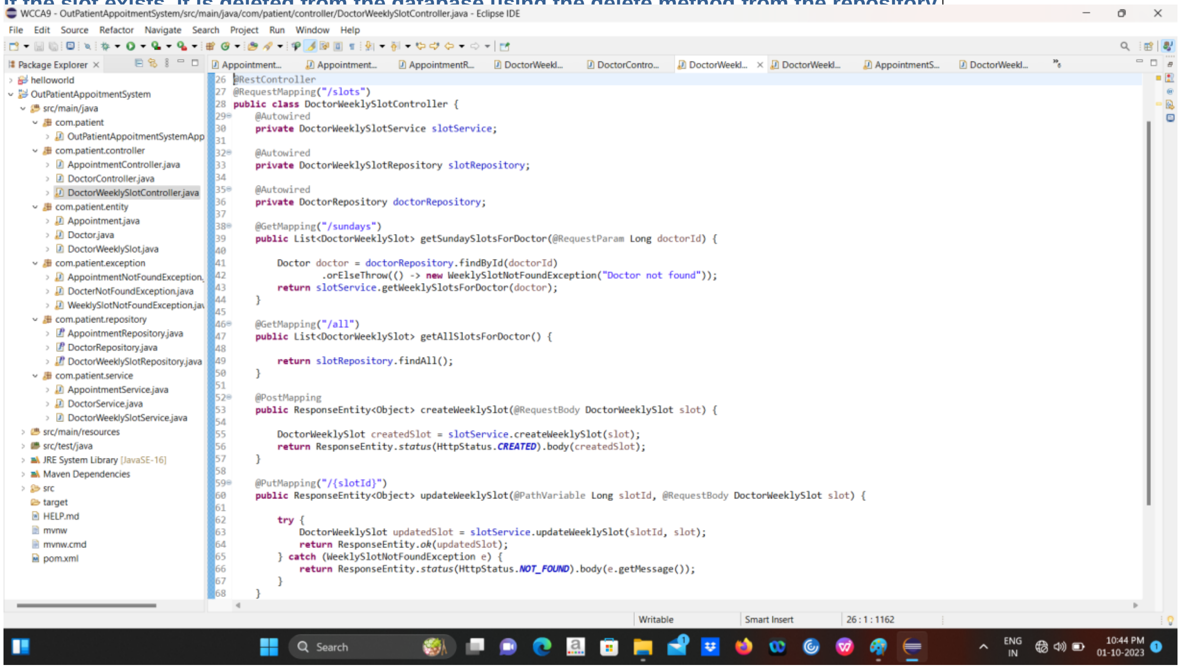
If the slot exists, it updates its properties (day of the week, start time, end time, max patients) with the values provided in the updatedSlot parameter and saves the updated slot back to the database.

deleteWeeklySlot Method:

This method deletes an existing doctor weekly slot by its ID.

It checks if the slot with the provided ID exists using the findById method from the repository. If not found, it throws a custom exception (WeeklySlotNotFoundException).

If the slot exists, it is deleted from the database using the delete method from the repository.



```java
@RestController
@RequestMapping("/slots")
public class DoctorWeeklySlotController {
    @Autowired
    private DoctorWeeklySlotService slotService;

    @Autowired
    private DoctorWeeklySlotRepository slotRepository;

    @Autowired
    private DoctorRepository doctorRepository;

    @GetMapping("/sundays")
    public List<DoctorWeeklySlot> getSundaySlotsForDoctor(@RequestParam Long doctorId) {

        Doctor doctor = doctorRepository.findById(doctorId)
                .orElseThrow(() -> new WeeklySlotNotFoundException("Doctor not found"));
        return slotService.getWeeklySlotsForDoctor(doctor);
    }

    @GetMapping("/all")
    public List<DoctorWeeklySlot> getAllSlotsForDoctor() {

        return slotRepository.findAll();
    }

    @PostMapping
    public ResponseEntity<Object> createWeeklySlot(@RequestBody DoctorWeeklySlot slot) {

        DoctorWeeklySlot createdSlot = slotService.createWeeklySlot(slot);
        return ResponseEntity.status(HttpStatus.CREATED).body(createdSlot);
    }

    @PutMapping("/{slotId}")
    public ResponseEntity<Object> updateWeeklySlot(@PathVariable Long slotId, @RequestBody DoctorWeeklySlot slot) {

        try {
            DoctorWeeklySlot updatedSlot = slotService.updateWeeklySlot(slotId, slot);
            return ResponseEntity.ok(updatedSlot);
        } catch (WeeklySlotNotFoundException e) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
        }
    }
```

DoctorWeeklyslot controller:

@RestController Annotation:

The @RestController annotation is used to mark this class as a Spring RESTful controller. It indicates that this class handles HTTP requests and returns HTTP responses as JSON or XML, depending on the request and configuration.

@RequestMapping("/slots") Annotation:

The @RequestMapping annotation at the class level specifies the base URL path for all the endpoints defined in this controller. In this case, all endpoints in this controller will start with "/slots."

@Autowired Annotations:

The @Autowired annotation is used to inject dependencies into the controller.

slotService: Injects an instance of DoctorWeeklySlotService to handle business logic related to doctor weekly slots.

slotRepository: Injects an instance of DoctorWeeklySlotRepository for direct data access to doctor weekly slots.

doctorRepository: Injects an instance of DoctorRepository for accessing doctor information.

@GetMapping("/sundays") Annotation:

The @GetMapping annotation maps an HTTP GET request to the specified endpoint.

/sundays specifies the path for this endpoint relative to the base path defined in the class-level @RequestMapping.

This endpoint retrieves doctor weekly slots for a specific doctor on Sundays.

@RequestParam Annotation:

@RequestParam Long doctorId: This annotation is used to extract the doctorId parameter from the URL query string. It's used to identify the doctor for whom to retrieve Sunday slots.

@GetMapping("/all") Annotation:

This endpoint retrieves all doctor weekly slots without filtering by day or doctor.

@PostMapping Annotation:

The @PostMapping annotation maps an HTTP POST request to create a new doctor weekly slot.

It expects a JSON request body containing the details of the new slot.

@RequestBody Annotation:

@RequestBody DoctorWeeklySlot slot: This annotation is used to deserialize the incoming JSON request body into a DoctorWeeklySlot object, which represents the slot to be created.

ResponseEntity:

ResponseEntity is used to customize the HTTP response returned by the controller.

ResponseEntity.status(HttpStatus.CREATED).body(createdSlot) is used to return a response with HTTP status code 201 (Created) and the created DoctorWeeklySlot object as the response body.

@PutMapping("/{slotId}") Annotation:

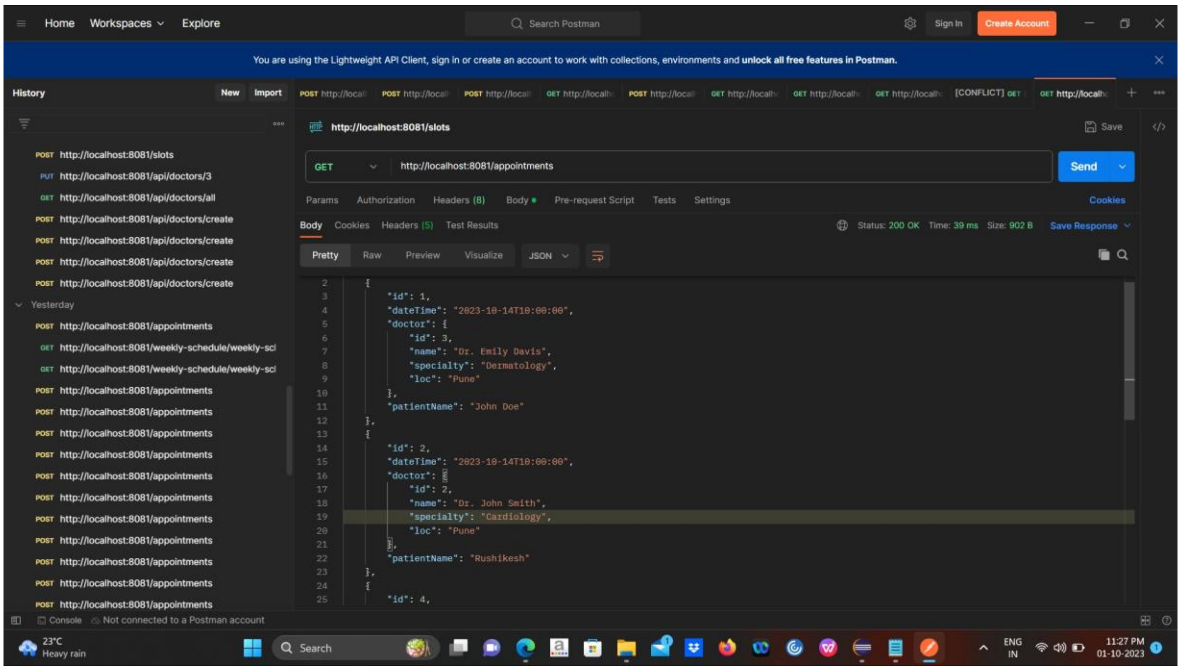
The @PutMapping annotation maps an HTTP PUT request to update an existing doctor weekly slot.

{slotId} is a path variable that represents the ID of the slot to be updated.

@DeleteMapping("/{slotId}") Annotation:

The @DeleteMapping annotation maps an HTTP DELETE request to delete an existing doctor weekly slot. {slotId} is a path variable that represents the ID of the slot to be deleted.

Getallappointments:



http://localhost:8081/slots

GET http://localhost:8081/appointments

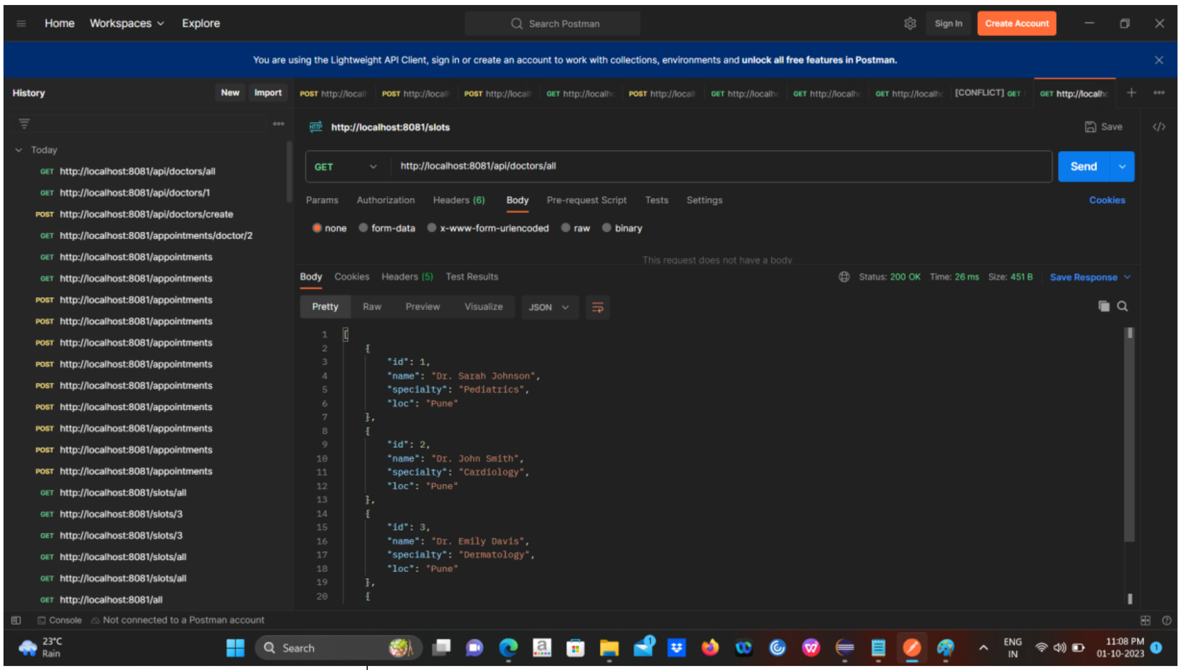
Status: 200 OK Time: 39 ms Size: 902 B

History:
- POST http://localhost:8081/slots
- PUT http://localhost:8081/api/doctors/3
- GET http://localhost:8081/api/doctors/all
- POST http://localhost:8081/api/doctors/create
- POST http://localhost:8081/api/doctors/create
- POST http://localhost:8081/api/doctors/create
- POST http://localhost:8081/api/doctors/create

Yesterday
- POST http://localhost:8081/appointments
- GET http://localhost:8081/weekly-schedule/weekly-scl
- GET http://localhost:8081/weekly-schedule/weekly-scl
- POST http://localhost:8081/appointments (×11)

```
    {
        "id": 1,
        "dateTime": "2023-10-14T10:00:00",
        "doctor": {
            "id": 3,
            "name": "Dr. Emily Davis",
            "specialty": "Dermatology",
            "loc": "Pune"
        },
        "patientName": "John Doe"
    },
    {
        "id": 2,
        "dateTime": "2023-10-14T10:00:00",
        "doctor": {
            "id": 2,
            "name": "Dr. John Smith",
            "specialty": "Cardiology",
            "loc": "Pune"
        },
        "patientName": "Rushikesh"
    },
    {
        "id": 4,
```

GetAllDoctors:



http://localhost:8081/slots

GET http://localhost:8081/api/doctors/all

Body: none

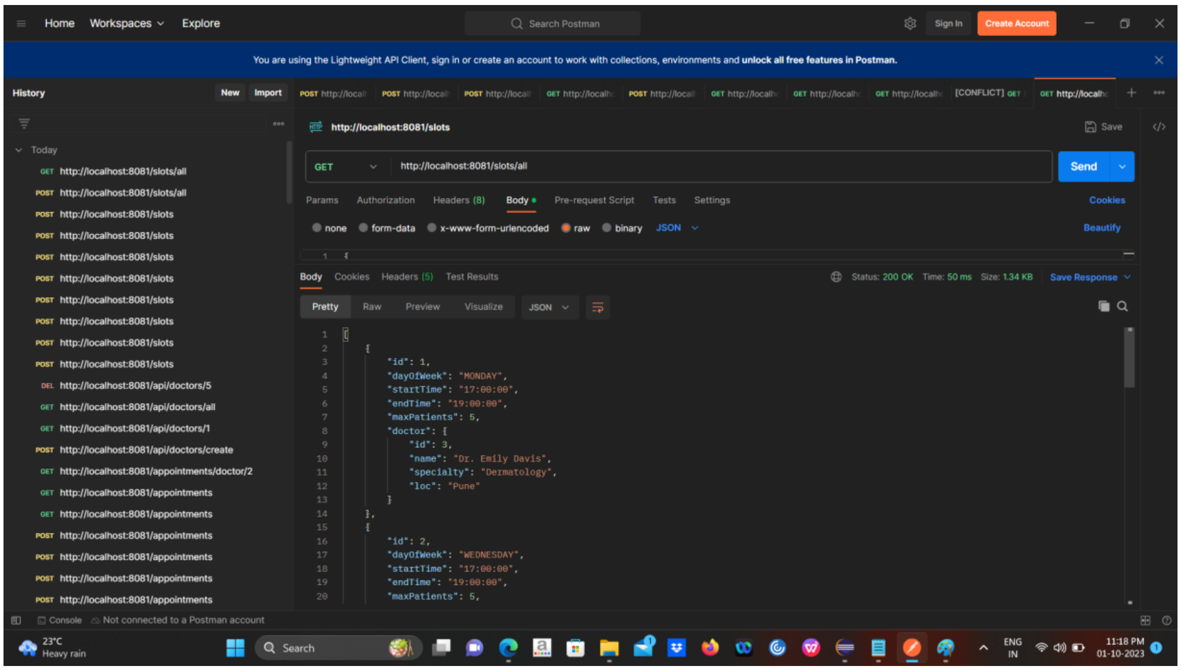
This request does not have a body

Status: 200 OK Time: 26 ms Size: 451 B

History — Today:
- GET http://localhost:8081/api/doctors/all
- GET http://localhost:8081/api/doctors/1
- POST http://localhost:8081/api/doctors/create
- GET http://localhost:8081/appointments/doctor/2
- GET http://localhost:8081/appointments
- GET http://localhost:8081/appointments
- POST http://localhost:8081/appointments (×9)
- GET http://localhost:8081/slots/all
- GET http://localhost:8081/slots/3
- GET http://localhost:8081/slots/3
- GET http://localhost:8081/slots/all
- GET http://localhost:8081/slots/all
- GET http://localhost:8081/all

```
[
    {
        "id": 1,
        "name": "Dr. Sarah Johnson",
        "specialty": "Pediatrics",
        "loc": "Pune"
    },
    {
        "id": 2,
        "name": "Dr. John Smith",
        "specialty": "Cardiology",
        "loc": "Pune"
    },
    {
        "id": 3,
        "name": "Dr. Emily Davis",
        "specialty": "Dermatology",
        "loc": "Pune"
    },
    {
```

GetAllDoctorsSlotWeekly:



You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.

History — Today

```
GET  http://localhost:8081/slots/all
POST http://localhost:8081/slots/all
POST http://localhost:8081/slots
POST http://localhost:8081/slots
POST http://localhost:8081/slots
POST http://localhost:8081/slots
POST http://localhost:8081/slots
POST http://localhost:8081/slots
POST http://localhost:8081/slots
POST http://localhost:8081/slots
DEL  http://localhost:8081/api/doctors/5
GET  http://localhost:8081/api/doctors/all
GET  http://localhost:8081/api/doctors/1
POST http://localhost:8081/api/doctors/create
GET  http://localhost:8081/appointments/doctor/2
GET  http://localhost:8081/appointments
GET  http://localhost:8081/appointments
POST http://localhost:8081/appointments
POST http://localhost:8081/appointments
POST http://localhost:8081/appointments
POST http://localhost:8081/appointments
```

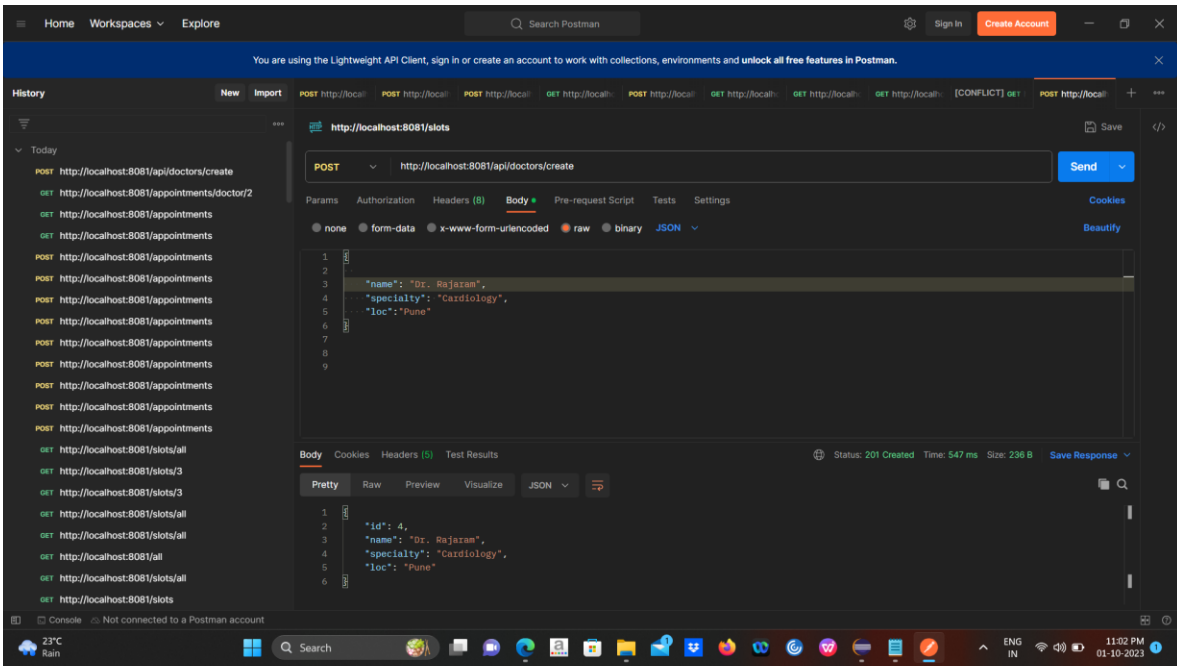
http://localhost:8081/slots

```
GET http://localhost:8081/slots/all
```

Status: 200 OK Time: 50 ms Size: 1.34 KB

```
[
    {
        "id": 1,
        "dayOfWeek": "MONDAY",
        "startTime": "17:00:00",
        "endTime": "19:00:00",
        "maxPatients": 5,
        "doctor": {
            "id": 3,
            "name": "Dr. Emily Davis",
            "specialty": "Dermatology",
            "loc": "Pune"
        }
    },
    {
        "id": 2,
        "dayOfWeek": "WEDNESDAY",
        "startTime": "17:00:00",
        "endTime": "19:00:00",
        "maxPatients": 5,
```

CreateDoctor:



You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.

History — Today

```
POST http://localhost:8081/api/doctors/create
GET  http://localhost:8081/appointments/doctor/2
GET  http://localhost:8081/appointments
GET  http://localhost:8081/appointments
POST http://localhost:8081/appointments
POST http://localhost:8081/appointments
POST http://localhost:8081/appointments
POST http://localhost:8081/appointments
POST http://localhost:8081/appointments
POST http://localhost:8081/appointments
POST http://localhost:8081/appointments
POST http://localhost:8081/appointments
POST http://localhost:8081/appointments
GET  http://localhost:8081/slots/all
GET  http://localhost:8081/slots/3
GET  http://localhost:8081/slots/3
GET  http://localhost:8081/slots/all
GET  http://localhost:8081/slots/all
GET  http://localhost:8081/all
GET  http://localhost:8081/slots/all
GET  http://localhost:8081/slots
```

http://localhost:8081/slots

```
POST http://localhost:8081/api/doctors/create
```

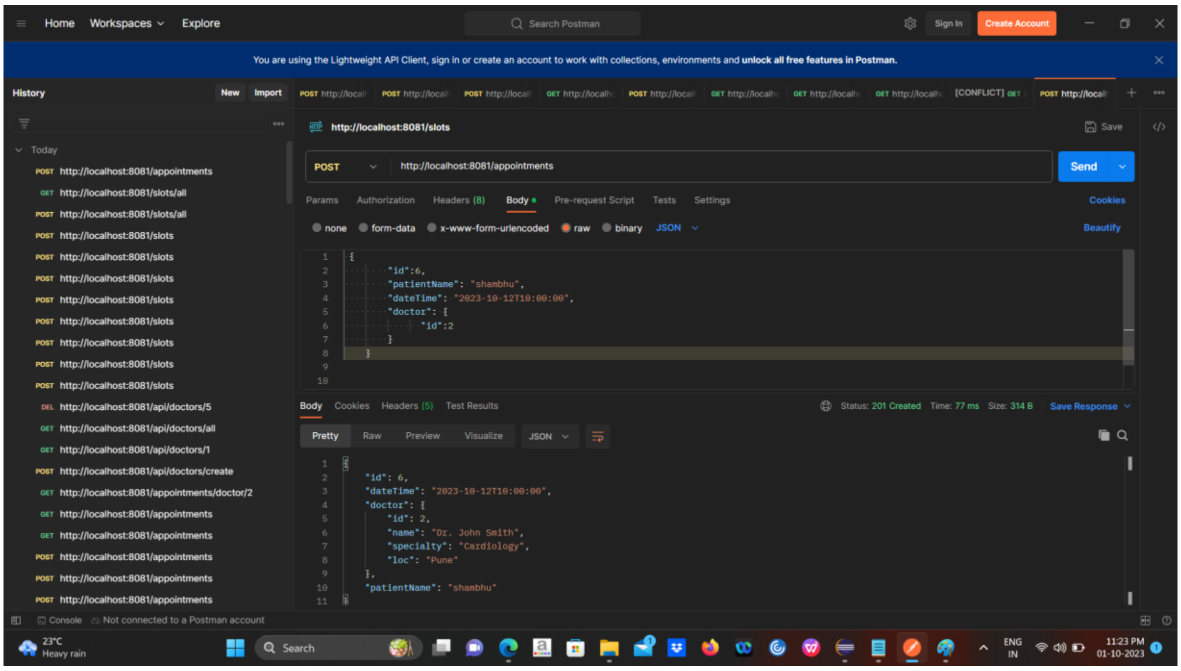
```
{

    "name": "Dr. Rajaram",
    "specialty": "Cardiology",
    "loc":"Pune"
}
```

Status: 201 Created Time: 547 ms Size: 236 B

```
{
    "id": 4,
    "name": "Dr. Rajaram",
    "specialty": "Cardiology",
    "loc": "Pune"
}
```

CreateAppoiment:



You are using the Lightweight API Client, sign in or create an account to work with collections, environments and **unlock all free features in Postman.**

History

- POST http://localhost:8081/appointments
- GET http://localhost:8081/slots/all
- POST http://localhost:8081/slots/all
- POST http://localhost:8081/slots
- POST http://localhost:8081/slots
- POST http://localhost:8081/slots
- POST http://localhost:8081/slots
- POST http://localhost:8081/slots
- POST http://localhost:8081/slots
- POST http://localhost:8081/slots
- POST http://localhost:8081/slots
- DEL http://localhost:8081/api/doctors/5
- GET http://localhost:8081/api/doctors/all
- GET http://localhost:8081/api/doctors/1
- POST http://localhost:8081/api/doctors/create
- GET http://localhost:8081/appointments/doctor/2
- GET http://localhost:8081/appointments
- GET http://localhost:8081/appointments
- POST http://localhost:8081/appointments
- POST http://localhost:8081/appointments
- POST http://localhost:8081/appointments

http://localhost:8081/slots

POST http://localhost:8081/appointments

```
{
    "id":6,
    "patientName": "shambhu",
    "dateTime": "2023-10-12T10:00:00",
    "doctor": {
        "id":2
    }
}
```

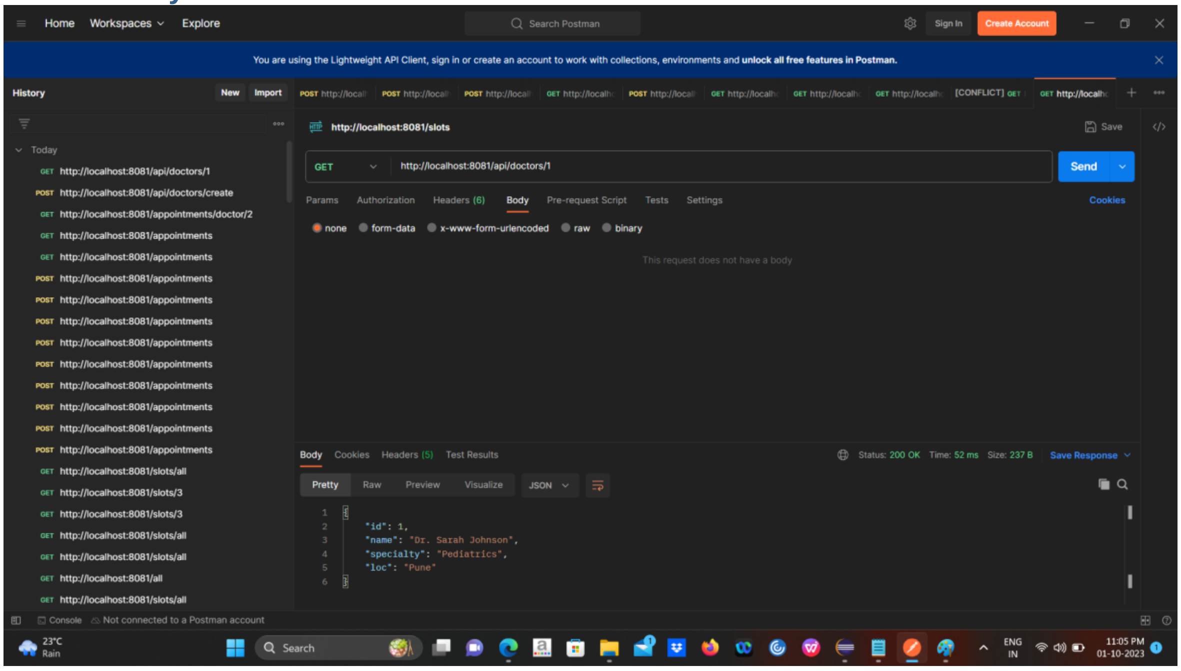
Status: 201 Created Time: 77 ms Size: 314 B

```
{
    "id": 6,
    "dateTime": "2023-10-12T10:00:00",
    "doctor": {
        "id": 2,
        "name": "Dr. John Smith",
        "specialty": "Cardiology",
        "loc": "Pune"
    },
    "patientName": "shambhu"
}
```

Console Not connected to a Postman account

23°C Heavy rain 11:23 PM 01-10-2023

GetDoctorById



You are using the Lightweight API Client, sign in or create an account to work with collections, environments and **unlock all free features in Postman.**

History

- GET http://localhost:8081/api/doctors/1
- POST http://localhost:8081/api/doctors/create
- GET http://localhost:8081/appointments/doctor/2
- GET http://localhost:8081/appointments
- GET http://localhost:8081/appointments
- POST http://localhost:8081/appointments
- POST http://localhost:8081/appointments
- POST http://localhost:8081/appointments
- POST http://localhost:8081/appointments
- POST http://localhost:8081/appointments
- POST http://localhost:8081/appointments
- POST http://localhost:8081/appointments
- POST http://localhost:8081/appointments
- POST http://localhost:8081/appointments
- GET http://localhost:8081/slots/all
- GET http://localhost:8081/slots/3
- GET http://localhost:8081/slots/3
- GET http://localhost:8081/slots/all
- GET http://localhost:8081/slots/all
- GET http://localhost:8081/all
- GET http://localhost:8081/slots/all

http://localhost:8081/slots

GET http://localhost:8081/api/doctors/1

Body: none

This request does not have a body

Status: 200 OK Time: 52 ms Size: 237 B

```
{
    "id": 1,
    "name": "Dr. Sarah Johnson",
    "specialty": "Pediatrics",
    "loc": "Pune"
}
```

Console Not connected to a Postman account

23°C Rain 11:05 PM 01-10-2023