SRI KRISHNA COLLEGE OF TECHNOLOGY
(An Autonomous Institution)
Approved by AICTE | Affiliated to Anna University Chennai| Accredited by NBA - AICTE| Accredited by NAAC with 'A' Grade
KOVAIPUDUR, COIMBATORE 641042

COURSE NAME: WEB FRAMEWORK USING RESTAPI

COURSE CODE: 23IT402

A MOBILE BEAUTY SERVICE APP

A PROJECT REPORT

Submitted by

MUGUNTHA PRAKASH V - 727823TUCS162

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

MARCH - 2025

SRI KRISHNA COLLEGE OF TECHNOLOGY
(An Autonomous Institution)
Approved by AICTE | Affiliated to Anna University Chennai| Accredited by NBA - AICTE| Accredited by NAAC with 'A' Grade
KOVAIPUDUR, COIMBATORE 641042

BONAFIDE CERTIFICATE

Certified that this project report **A MOBILE BEAUTY SERVICE APP** is the bonafide work of **MUGUNTHA PRAKASH V 727823TUCS162 ,** who carried out the project work under my supervision.

SIGNATURE

Ms. S.VIDHIYA

SUPERVISOR

Assistant Professor,

Department of Computer Science and Engineering,

Sri Krishna College of Technology, Coimbatore-641042

SIGNATURE

Dr. M. UDHAYAMOORTHI

HEAD OF THE DEPARTMENT

Associate Professor,

Department of Computer Science and Engineering,

Sri Krishna College of Technology, Coimbatore-641042

Certified that the candidate was examined by me in the Project Work Viva Voce examination held on_____at Sri Krishna College of Technology, Coimbatore-641042.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost we thank the **Almighty** for being our light and for showering his gracious blessings throughout the course of this project.

We express our gratitude to our beloved Principal, **Dr. M.G. Sumithra,** for providing all facilities.

We are grateful to our beloved Dean, Computing Sciences **Dr.T. Senthilnathan,** for her tireless and relentless support.

With the grateful heart, our sincere thanks to our Head of the Department **Dr.M.UDHAYAMOORTHI**, Department of Computer Science and Engineering for the motivation and all support to complete the project work.

We thank **Ms S.VIDHIYA,** Assistant Professor, Department of Computer Science and Engineering, for his motivation and support.

We are thankful to all the **Teaching and Non-Teaching Staff** of Department of Computer Science and Engineering and to all those who have directly and indirectly extended their help to us in completing this project work successfully.

We extend our sincere thanks to our **family members** and our beloved **friends**, who had been strongly supporting us in all our endeavour.

ABSTRACT

This project presents a **Mobile Beauty Service Application**, designed to provide on-demand beauty and wellness services at customers' preferred locations. The application connects clients with professional beauticians, offering services such as hairstyling, makeup, skincare, nail care, and spa treatments. The system ensures a seamless user experience by integrating online appointment booking, real-time service tracking, secure payment processing, and customer reviews. The platform is built using **React for the frontend**, **Spring Boot for backend services**, and **MySQL for database management**. AI-powered service recommendations enhance personalization, while automated scheduling and notifications improve efficiency. This application aims to **redefine convenience in the beauty and wellness industry**, making premium beauty services accessible to users anytime, anywhere. The mobile beauty service industry has revolutionized the traditional salon experience by offering beauty treatments at customers' preferred locations. This service provides convenience, flexibility, and personalized care, catering to busy professionals, elderly individuals, event attendees, and those seeking at-home pampering. The system integrates an online booking platform where users can schedule appointments, select beauty professionals, and choose from a variety of services, including hairstyling, makeup, skincare, and nail care. Advanced features such as real-time tracking, secure payments, and customer reviews enhance user experience. This paper explores the development, implementation, and impact of mobile beauty services, highlighting their role in transforming the beauty and wellness industry.

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The modern beauty industry is highly fragmented, with traditional salons often struggling to meet the diverse needs of clients. Customers face challenges such as **time constraints, location inconvenience, and inconsistent service quality**. In contrast, professional beauticians face difficulties in reaching a wider audience, managing appointments effectively, and optimizing their earnings.

1.1 PROBLEM STATEMENT

In today's fast-paced world, customers often struggle to find time for beauty treatments due to their busy schedules. Traditional salons require appointments and waiting times, leading to inconvenience. At the same time, professional beauticians face challenges in reaching potential clients and managing appointments effectively. A **Mobile Beauty Service Application** is required to:

- Provide an **on-demand platform** for booking beauty services.
- Enable customers to **browse and book professional beauticians** at home.
- Offer **secure payment options** and service tracking.
- Help beauty professionals **manage bookings, earnings, and customer feedback**.
- Enhance user experience with **AI-driven service recommendations**.

1.2 OVERVIEW

The **primary overview** of this mobile beauty service application is to revolutionize the way beauty services are delivered by providing an **efficient, user-friendly, and technologically advanced** platform for customers and service providers. The key objectives include:

- **Enhancing Accessibility**: Ensuring beauty services are available at the user's chosen location, removing the need for salon visits.
- **Optimizing Appointment Management**: Allowing users to book, reschedule, or cancel appointments easily.
- **Empowering Beauticians**: Providing professionals with tools to manage their schedules, improve visibility, and receive secure payments.
- **Personalized User Experience**: Using **AI-powered analytics** to recommend services tailored to user preferences.
- **Secure Transactions**: Ensuring all payments are encrypted, supporting multiple payment gateways for smooth and reliable financial transactions.
- **Building a Trustworthy Ecosystem**: Integrating a **rating and review system** to maintain service quality and enhance customer confidence.

1.3 OBJECTIVE

The **primary objective** of this mobile beauty service application is to revolutionize the way beauty services are delivered by providing an **efficient, user-friendly, and technologically advanced** platform for customers and service providers. The key objectives include:

- **Enhancing Accessibility**: Ensuring beauty services are available at the user's chosen location, removing the need for salon visits.
- **Optimizing Appointment Management**: Allowing users to book, reschedule, or cancel appointments easily.

- **Empowering Beauticians**: Providing professionals with tools to manage their schedules, improve visibility, and receive secure payments.
- **Personalized User Experience**: Using **AI-powered analytics** to recommend services tailored to user preferences.
- **Secure Transactions**: Ensuring all payments are encrypted, supporting multiple payment gateways for smooth and reliable financial transactions.
- **Building a Trustworthy Ecosystem**: Integrating a **rating and review system** to maintain service quality and enhance customer confidence.

CHAPTER 2

SYSTEM SPECIFICATION

In this chapter, we are going to see the software that we have used to build the website. This chapter gives you a small description about the software used in the project.

2.1 VS CODE

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux, and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences.

2.2 SPRING BOOT

Spring Boot is a popular Java-based framework for building standalone, production-grade applications. It simplifies the development process by providing auto-configuration and a wide range of built-in features, allowing developers to focus on writing business logic rather than boilerplate code.

2.3 MYSQL

MySQL is an open-source relational database management system. It's known for its reliability, performance, and ease of use, making it a popular choice for web applications. MySQL efficiently stores and retrieves data, supports complex queries, and can handle large volumes of information.

2.4 Java 17

Java 17 is a long-term support (LTS) version of Java, released in September 2021, offering significant performance, security, and stability

improvements. It introduces features like sealed classes, enhanced pattern matching, and an optimized garbage collection process for better memory management. Java 17 removes outdated features, ensuring a more efficient and modern programming environment. It supports cloud-native applications and integrates seamlessly with frameworks like Spring Boot. The inclusion of the Vector API and Foreign Function & Memory API enhances mathematical computations and native code interactions. With long-term security updates, Java 17 is ideal for enterprise applications and cloud-based solutions.

2.5 JVM (Java Virtual Machine)

The Java Virtual Machine (JVM) is a core component of the Java ecosystem that enables Java applications to run on multiple platforms without modification. It translates Java bytecode into machine code for execution, ensuring portability and efficiency. JVM manages memory allocation, garbage collection, and runtime optimizations through Just-In-Time (JIT) compilation. It provides security through bytecode verification and supports multithreading for concurrent processing. Used in frameworks like Spring and Hibernate, JVM remains fundamental to Java-based enterprise and web development.

2.6 JRE (Java Runtime Environment)

The Java Runtime Environment (JRE) is a software package that provides the necessary components to run Java applications. It includes the JVM, core libraries, and additional resources required for executing Java programs. JRE ensures smooth application performance by managing memory, security, and runtime optimizations. It supports Java applications across various operating systems, enabling cross-platform compatibility. With continuous updates and security patches, JRE remains a crucial part of Java’s ecosystem, powering enterprise, web, and mobile applications.

2.7 Maven

To efficiently develop and run a Spring Boot application, your system meet certain hardware and software requirements. A recommended setup mciuues a 64-bit processor, at least 8GB of RAM (16GB preferred for large applications), and an SSD for faster build times. The operating system can be Windows, macOS, or Linux. You'll need Java 17+ (or the required JDK version for your project), Apache Maven (latest version) for dependency management, and an IDE like IntelliJ IDEA, Eclipse, or VS Code. Additionally, ensure PostgreSQL, MySQL, or another preferred database is installed if your application requires persistent storage. Having Docker and Kubernetes is beneficial for containerized deployment.

CHAPTER 3

PROPOSED SYSTEM

This chapter provides a detailed description of the proposed idea behind the development of our **Mobile Beauty Service Application**. The platform is designed to seamlessly connect customers with professional beauticians, offering a convenient and efficient way to access beauty and wellness services at home, the workplace, or any preferred location.

3.1 EXISTING SOLUTIONS:

Competitors:

- **Salon Chains** (e.g., Naturals, Green Trends): Require prior in-person appointments.
- **Home Service Platforms** (e.g., UrbanClap, Housejoy): Lack real-time scheduling flexibility.
- **Freelance Beauticians** (Social Media): No structured booking or payment system.

Gap:

- A dedicated app that offers a **seamless booking experience, real-time availability, and professional management tools.**

3.2 PROPOSED SYSTEM

The **Mobile Beauty Service Application** is an intelligent and user-friendly solution that integrates various functionalities to assist users in **booking, managing, and receiving beauty services**. The proposed system includes the following core components:

1. User Registration & Profile Management

- Secure account creation and authentication via email, phone, or third-party login (Google, Facebook, Apple ID).
- Customizable user profiles including **beauty preferences, skin type, hair type, and preferred professionals**.
- Beautician profile creation with **certifications, service offerings, portfolio, and customer reviews**.

2. Beauty Service Listings & Booking System

- Comprehensive **service catalog** including hairstyling, makeup, facials, nail care, spa treatments, and more.
- **Search and filter** options based on service category, location, pricing, and beautician ratings.
- **Instant booking system** with real-time appointment availability and scheduling.
- **Option for recurring appointments** for customers who require regular services.

3. In-App Payment & Secure Transactions

- **Multiple payment options** including UPI, credit/debit cards, digital wallets, and cash on delivery.
- **Transparent pricing** with breakdowns for services, discounts, and taxes.
- **Secure payment processing** with **AES encryption and SSL protocols** to protect user transactions.

4. Real-Time Beautician Tracking & Notifications

- **Live GPS tracking** to monitor beautician arrival times.
- **Automated appointment reminders and confirmations** via SMS, email, and push notifications.
- Beautician route optimization for reduced travel time and improved efficiency.

5. Ratings, Reviews, & Feedback System

- **Customer rating system** for each completed service.
- Verified **user reviews** to help customers make informed choices.
- **Beautician feedback mechanism** to improve service quality and professionalism.

6. Beautician Management & Scheduling

- **Appointment dashboard** for beauticians to manage bookings, track earnings, and update availability.
- **Custom service packages and promotions** to attract new

customers.

- **Automated scheduling assistant** to reduce double bookings and optimize availability.

7. AI-Powered Recommendations & Personalized Services

- AI-driven **beauty recommendations** based on skin type, hair texture, and past bookings.
- **Smart service combos** suggesting treatments that pair well together.
- **Seasonal beauty care alerts** (e.g., summer skincare, winter hair care).

8. Community Engagement & Social Features

- **Social media integration** to allow users to share beauty experiences and reviews.
- **In-app beauty blogs and video tutorials** featuring expert tips.
- A **loyalty and referral program** rewarding customers for repeat bookings.

9. Optional Beauty Product Marketplace

- **E-commerce section** selling beauty products recommended by professionals.
- Direct purchasing options for beauty essentials like skincare, haircare, and cosmetics.
- Subscription-based beauty boxes curated by beauticians.

10. Security & Data Privacy

- **End-to-end encryption** for user data protection.
- **GDPR and HIPAA compliance** ensuring user privacy.
- Multi-factor authentication (MFA) for secure account access.

3.3 ADVANTAGES

Convenience & Accessibility

- Allows users to **book beauty services at their convenience**, eliminating the need for salon visits.
- Provides a **seamless mobile experience** accessible across smartphones and tablets.
- Enables beauticians to **expand their client base** without being restricted to a physical location.

Time Efficiency

- Reduces waiting times by offering **real-time scheduling and instant booking confirmation**.
- Helps beauticians **optimize their schedules and manage multiple clients efficiently**.
- Provides **GPS tracking** to estimate arrival times accurately.

Customization & Personalization

- AI-driven recommendations ensure **personalized beauty services** tailored to individual preferences.

- Users can save **favorite beauticians, preferred services, and custom beauty packages**.
- Flexible pricing models allow **subscription plans and bundled services** for cost-effective beauty care.

Secure & Seamless Transactions

- Ensures safe payments with **multiple payment gateway integrations**.
- Offers **cashless transactions**, reducing the hassle of handling money.
- Transparent billing ensures customers are aware of costs before booking.

Beautician Empowerment & Business Growth

- Provides an **efficient appointment management system** for beauticians to streamline their work.
- Offers an **earning dashboard** to track revenue, monitor service trends, and analyze customer feedback.
- Allows professionals to **set their own pricing, work schedules, and service offerings**.

Community & Engagement

- Encourages users to **engage with beauty experts and share tips** via in-app forums.
- Users can **review and recommend beauty services** to friends and family.
- Enhances user retention with **reward programs, discounts, and**

referral bonuses.

AI-Powered Smart Recommendations

- Suggests **personalized beauty treatments** based on past bookings and preferences.
- Offers **intelligent skin and hair analysis** to recommend suitable products and treatments.
- Helps beauticians **identify customer trends and tailor their services accordingly**.

Safety & Data Protection

- Uses **encrypted user authentication** to protect personal and payment information.
- Ensures compliance with **global data privacy regulations** (GDPR, HIPAA).
- Offers **verified beautician background checks** to enhance trust and safety.

On-Demand & Subscription Models

- Allows customers to **book one-time services or subscribe to monthly beauty packages**.
- Beauticians can create **loyalty-based plans to retain clients**.
- Supports promotional campaigns, helping beauticians **boost their earnings**.

Scalability & Future Enhancements

- Potential integration with **AR-based virtual try-ons** for makeup and hairstyling previews.

CHAPTER 4

METHODOLOGIES

The development of the **Mobile Beauty Service Application** follows a structured methodology to ensure a **robust, efficient, and user-friendly** experience. The methodology involves multiple phases, including **research, design, development, testing, and deployment**, following **agile software development practices** to allow iterative improvements and adaptability.

1. Requirement Analysis & Research

- Conduct **market research and user surveys** to identify key challenges in accessing beauty services.
- Analyze existing **salon booking and beauty service platforms** to assess their strengths and gaps.
- Define **core system requirements** based on user needs, including service selection, appointment scheduling, secure payments, and beautician management.

2. System Design & Architecture

- Develop a **modular system architecture** to ensure scalability, maintainability, and easy future enhancements.
- Use **cloud-based storage solutions** to store user data securely and enhance accessibility.
- Implement a **responsive UI/UX design** using wireframing tools to optimize usability across devices.
- Design a **RESTful API structure** for efficient communication

between frontend and backend.

3. Technology Stack Selection

- **Frontend Development**: React Native or Flutter for **cross-platform compatibility** on iOS and Android.
- **Backend Development**: Spring Boot with RESTful APIs for **efficient service management**.
- **Database**: MySQL or Firebase for **secure and scalable data storage**.
- **Security**: AES encryption, SSL protocols, and OAuth authentication for **data protection and secure transactions**.
- **AI & Machine Learning**: AI-powered **beauty service recommendations and predictive analytics** for personalized experiences.

4. Implementation & Development

- Develop **core application features** iteratively, starting with **user authentication and profile management**.
- Integrate **beautician service listings, appointment booking, and secure payment modules**.
- Implement **real-time tracking** for beautician location updates and estimated arrival times.
- Develop **automated notifications and reminders** for users and beauticians.
- Optimize the app for **fast response times, minimal resource consumption, and smooth navigation**.

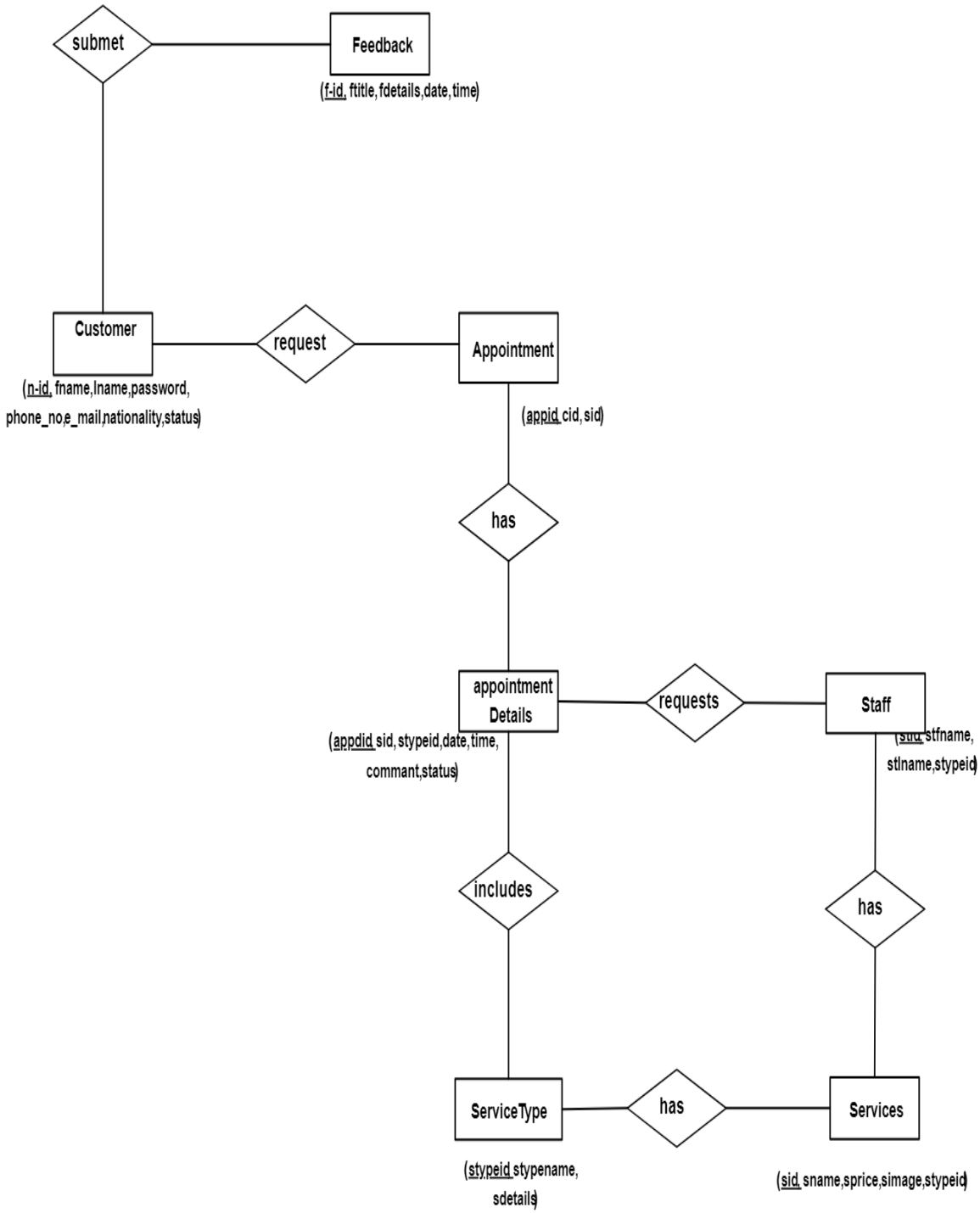
5. Testing & Quality Assurance

- Conduct **unit testing** for individual components to ensure functionality and stability.
- Perform **integration testing** to verify seamless interaction between different modules.
- Conduct **user acceptance testing (UAT)** with a sample group of customers and beauticians to gather feedback.
- Optimize app performance for **fast loading times, secure transactions, and bug-free operations**.

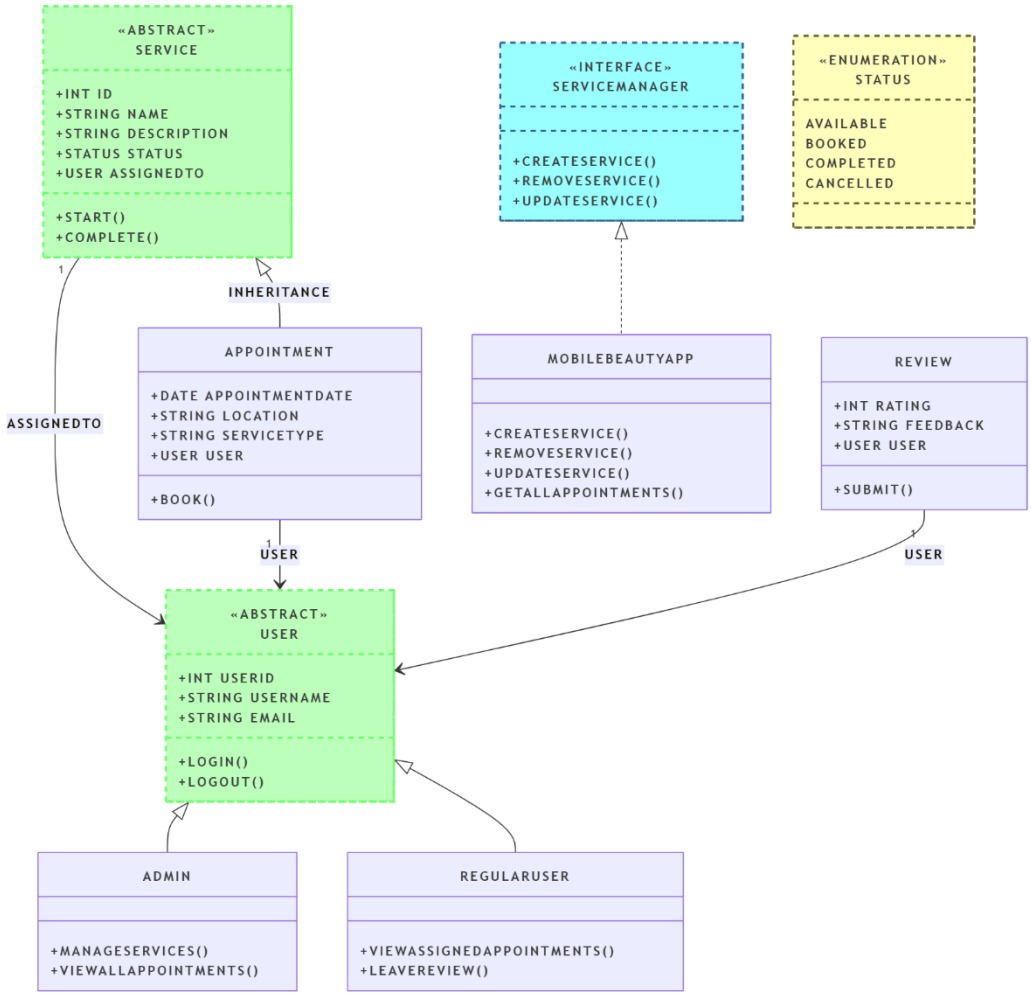
6. Deployment & Maintenance

- Deploy the application on **Google Play Store and Apple App Store** for wide accessibility.
- Provide **cloud-based data backup, periodic security updates, and bug fixes**.
- Implement **continuous monitoring and updates** based on user feedback and changing industry trends.
- Expand **feature set and integrations**, such as AR beauty try-ons and AI-powered skincare analysis, in future versions.

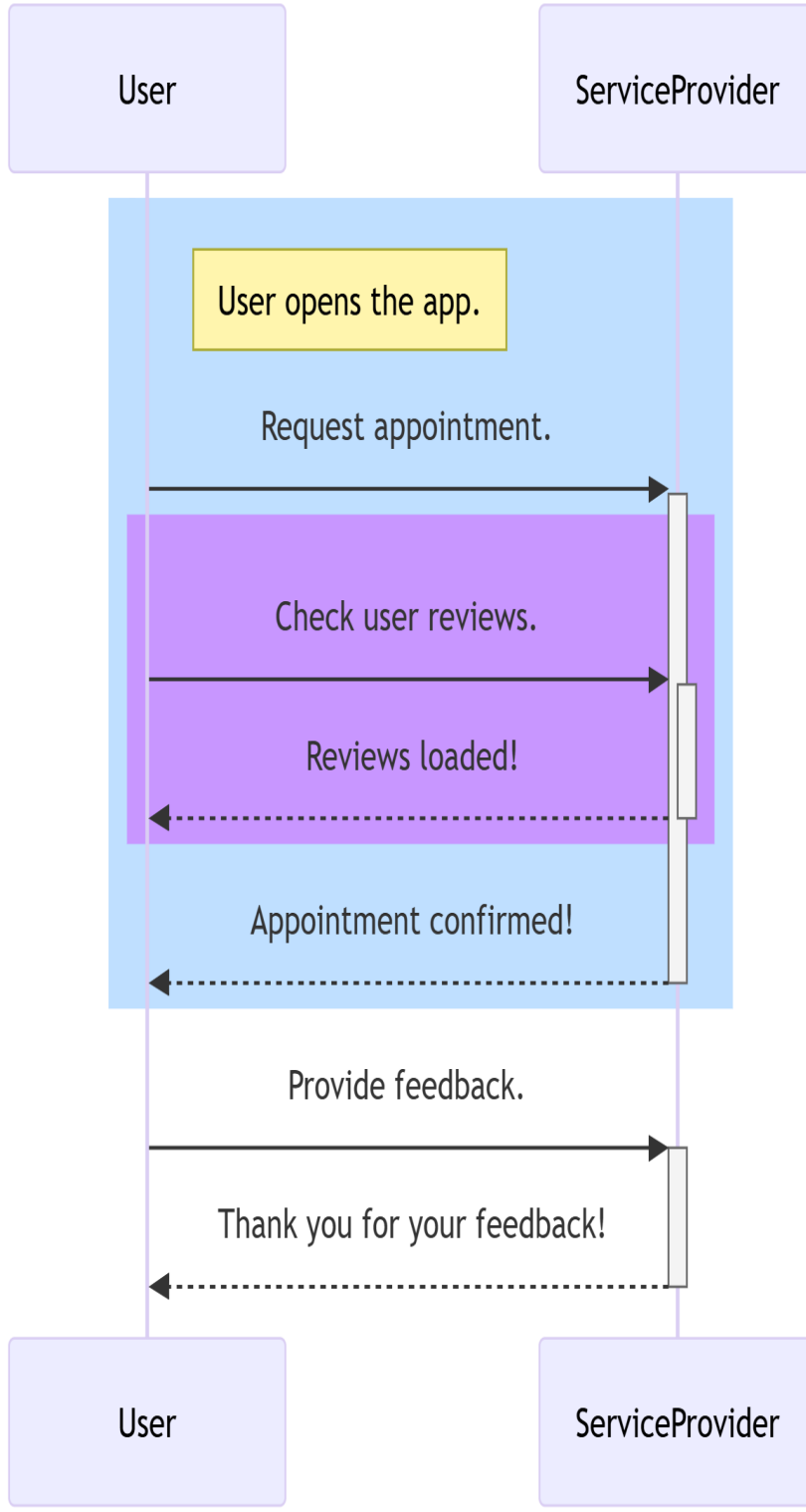
ER Diagram:


submet
Feedback
(f-id, ftitle, fdetails, date, time)
Customer
request
Appointment
(n-id, fname, lname, password,
phone_no, e_mail, nationality, status)
(appid cid, sid)
has
appointment
Details
requests
Staff
(appdid sid, stypeid, date, time,
commant, status)
(stfname,
stlname, stypeid)
includes
has
ServiceType
has
Services
(stypeid stypename,
sdetails)
(sid, sname, sprice, simage, stypeid)

Class Diagram:



Sequence Diagram:



CHAPTER 5

IMPLEMENTATION AND RESULT

BACKEND CODING

Controller

Appointment Controller

```
package com.example.demo.controller;


import com.example.demo.entity.Appointment;

import com.example.demo.service.AppointmentService;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.media.Content;

import io.swagger.v3.oas.annotations.media.Schema;

import io.swagger.v3.oas.annotations.responses.ApiResponse;

import io.swagger.v3.oas.annotations.responses.ApiResponses;

import io.swagger.v3.oas.annotations.tags.Tag;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Sort;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.*;


import java.util.List;

import java.util.Optional;
```

```
@RestController

@RequestMapping("/api/appointments")

@Tag(name = "Appointment", description = "Appointment management APIs")

public class AppointmentController {


private final AppointmentService appointmentService;


@Autowired

public AppointmentController(AppointmentService appointmentService) {

this.appointmentService = appointmentService;

}


@Operation(summary = "Get all appointments", description = "Returns all
appointments")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Successfully retrieved
appointments", content = @Content(schema = @Schema(implementation =
Appointment.class)))

})

@GetMapping

public ResponseEntity<List<Appointment>> getAllAppointments() {

List<Appointment> appointments = appointmentService.getAllAppointments();

return new ResponseEntity<>(appointments, HttpStatus.OK);

}


@Operation(summary = "Get appointments with pagination and sorting",

description = "Returns appointments with pagination and sorting capabilities")
```

```
@GetMapping("/paged")

public ResponseEntity<Page<Appointment>> getAppointmentsPaged(

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "10") int size,

@RequestParam(defaultValue = "appointmentTime") String sortBy,

@RequestParam(defaultValue = "asc") String direction) {


Sort.Direction sortDirection = direction.equalsIgnoreCase("desc") ?

Sort.Direction.DESC : Sort.Direction.ASC;


Page<Appointment> appointments =
appointmentService.getAllAppointmentsPaged(PageRequest.of(page, size,
Sort.by(sortDirection, sortBy)));


return new ResponseEntity<>(appointments, HttpStatus.OK);

}


@Operation(summary = "Get appointment by ID", description = "Returns a single
appointment by its ID")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Successfully retrieved the
appointment", content = @Content(schema = @Schema(implementation =
Appointment.class))),

@ApiResponse(responseCode = "404", description = "Appointment not found")

})

@GetMapping("/{id}")

public ResponseEntity<Appointment> getAppointmentById(@PathVariable Long id) {

Optional<Appointment> appointment = appointmentService.getAppointmentById(id);

return appointment.map(value -> new ResponseEntity<>(value, HttpStatus.OK))
```

```
.orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));

}



@Operation(summary = "Create a new appointment", description = "Creates a new
appointment")

@ApiResponses(value = {

@ApiResponse(responseCode = "201", description = "Appointment created successfully",
content = @Content(schema = @Schema(implementation = Appointment.class)))

})

@PostMapping

public ResponseEntity<Appointment> createAppointment(@RequestBody Appointment
appointment) {

Appointment createdAppointment = appointmentService.saveAppointment(appointment);

return new ResponseEntity<>(createdAppointment, HttpStatus.CREATED);

}



@Operation(summary = "Update an existing appointment", description = "Updates an
appointment by its ID")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Appointment updated successfully",
content = @Content(schema = @Schema(implementation = Appointment.class))),

@ApiResponse(responseCode = "404", description = "Appointment not found")

})

@PutMapping("/{id}")

public ResponseEntity<Appointment> updateAppointment(@PathVariable Long id,
@RequestBody Appointment appointment) {

Optional<Appointment> existingAppointment =
appointmentService.getAppointmentById(id);



if (existingAppointment.isPresent()) {
```

```
appointment.setId(id);

Appointment updatedAppointment = appointmentService.saveAppointment(appointment);

return new ResponseEntity<>(updatedAppointment, HttpStatus.OK);

} else {

return new ResponseEntity<>(HttpStatus.NOT_FOUND);

}

}


@Operation(summary = "Update appointment status", description = "Updates only the
status of an appointment")

@PatchMapping("/{id}/status")

public ResponseEntity<Appointment> updateAppointmentStatus(

@PathVariable Long id,

@RequestParam String status) {


Optional<Appointment> existingAppointment =
appointmentService.getAppointmentById(id);


if (existingAppointment.isPresent()) {

Appointment appointment = existingAppointment.get();

appointment.setStatus(status);

Appointment updatedAppointment = appointmentService.saveAppointment(appointment);

return new ResponseEntity<>(updatedAppointment, HttpStatus.OK);

} else {

return new ResponseEntity<>(HttpStatus.NOT_FOUND);

}

}
```

```
@Operation(summary = "Delete an appointment", description = "Deletes an appointment
by its ID")

@ApiResponses(value = {

@ApiResponse(responseCode = "204", description = "Appointment deleted
successfully"),

@ApiResponse(responseCode = "404", description = "Appointment not found")

})

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteAppointment(@PathVariable Long id) {

Optional<Appointment> appointment = appointmentService.getAppointmentById(id);



if (appointment.isPresent()) {

appointmentService.deleteAppointment(id);

return new ResponseEntity<>(HttpStatus.NO_CONTENT);

} else {

return new ResponseEntity<>(HttpStatus.NOT_FOUND);

}

}



@Operation(summary = "Get appointments by customer ID", description = "Returns all
appointments for a specific customer")

@GetMapping("/customer/{customerId}")

public ResponseEntity<List<Appointment>> getAppointmentsByCustomerId(@PathVariable
Long customerId) {

List<Appointment> appointments =
appointmentService.getAppointmentsByCustomerId(customerId);

return new ResponseEntity<>(appointments, HttpStatus.OK);

}

}
```

CustomerController

```
package com.example.demo.controller;


import com.example.demo.entity.Customer;

import com.example.demo.service.CustomerService;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.media.Content;

import io.swagger.v3.oas.annotations.media.Schema;

import io.swagger.v3.oas.annotations.responses.ApiResponse;

import io.swagger.v3.oas.annotations.responses.ApiResponses;

import io.swagger.v3.oas.annotations.tags.Tag;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.*;


import java.util.List;

import java.util.Optional;


@RestController

@RequestMapping("/api/customers")

@Tag(name = "Customer", description = "Customer management APIs")

public class CustomerController {


private final CustomerService customerService;
```

```
@Autowired

public CustomerController(CustomerService customerService) {

this.customerService = customerService;

}



@Operation(summary = "Get all customers", description = "Returns all customers")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Successfully retrieved customers",
content = @Content(schema = @Schema(implementation = Customer.class)))

})

@GetMapping

public ResponseEntity<List<Customer>> getAllCustomers() {

List<Customer> customers = customerService.getAllCustomers();

return new ResponseEntity<>(customers, HttpStatus.OK);

}



@Operation(summary = "Get customer by ID", description = "Returns a single customer
by ID")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Successfully retrieved the
customer", content = @Content(schema = @Schema(implementation = Customer.class))),

@ApiResponse(responseCode = "404", description = "Customer not found")

})

@GetMapping("/{id}")

public ResponseEntity<Customer> getCustomerById(@PathVariable Long id) {

Optional<Customer> customer = customerService.getCustomerById(id);

return customer.map(value -> new ResponseEntity<>(value, HttpStatus.OK))

.orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));
```

```
}


@Operation(summary = "Get customer with appointments", description = "Returns a
customer and their appointments in a single query")

@GetMapping("/{id}/with-appointments")

public ResponseEntity<Customer> getCustomerWithAppointments(@PathVariable Long id) {

Optional<Customer> customer = customerService.getCustomerWithAppointments(id);

return customer.map(value -> new ResponseEntity<>(value, HttpStatus.OK))

.orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));

}


@Operation(summary = "Get customer by username", description = "Returns a single
customer by username")

@GetMapping("/by-username/{username}")

public ResponseEntity<Customer> getCustomerByUsername(@PathVariable String username)
{

Customer customer = customerService.getCustomerByUsername(username);

if (customer != null) {

return new ResponseEntity<>(customer, HttpStatus.OK);

} else {

return new ResponseEntity<>(HttpStatus.NOT_FOUND);

}

}


@Operation(summary = "Create a new customer", description = "Creates a new
customer")

@ApiResponses(value = {

@ApiResponse(responseCode = "201", description = "Customer created successfully",
content = @Content(schema = @Schema(implementation = Customer.class)))
```

```
})

@PostMapping

public ResponseEntity<Customer> createCustomer(@RequestBody Customer customer) {

Customer createdCustomer = customerService.saveCustomer(customer);

return new ResponseEntity<>(createdCustomer, HttpStatus.CREATED);

}



@Operation(summary = "Update an existing customer", description = "Updates a
customer by ID")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Customer updated successfully",
content = @Content(schema = @Schema(implementation = Customer.class))),

@ApiResponse(responseCode = "404", description = "Customer not found")

})

@PutMapping("/{id}")

public ResponseEntity<Customer> updateCustomer(@PathVariable Long id, @RequestBody
Customer customer) {

Optional<Customer> existingCustomer = customerService.getCustomerById(id);



if (existingCustomer.isPresent()) {

customer.setId(id);

Customer updatedCustomer = customerService.saveCustomer(customer);

return new ResponseEntity<>(updatedCustomer, HttpStatus.OK);

} else {

return new ResponseEntity<>(HttpStatus.NOT_FOUND);

}

}
```

```
@Operation(summary = "Delete a customer", description = "Deletes a customer by ID")
@ApiResponses(value = {
@ApiResponse(responseCode = "204", description = "Customer deleted successfully"),
@ApiResponse(responseCode = "404", description = "Customer not found")
})
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteCustomer(@PathVariable Long id) {
Optional<Customer> customer = customerService.getCustomerById(id);

if (customer.isPresent()) {
customerService.deleteCustomer(id);
return new ResponseEntity<>(HttpStatus.NO_CONTENT);
} else {
return new ResponseEntity<>(HttpStatus.NOT_FOUND);
}
}
}
```

Reward controller

```
package com.example.demo.controller;

import com.example.demo.entity.Reward;
import com.example.demo.service.RewardService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.responses.ApiResponses;
```

```
import io.swagger.v3.oas.annotations.tags.Tag;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.*;


import java.util.List;

import java.util.Optional;


@RestController

@RequestMapping("/api/rewards")

@Tag(name = "Reward", description = "Reward management APIs")

public class RewardController {


private final RewardService rewardService;


@Autowired

public RewardController(RewardService rewardService) {

this.rewardService = rewardService;

}


@Operation(summary = "Get all rewards", description = "Returns all rewards")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Successfully retrieved rewards",
content = @Content(schema = @Schema(implementation = Reward.class)))

})

@GetMapping
```

```
public ResponseEntity<List<Reward>> getAllRewards() {

List<Reward> rewards = rewardService.getAllRewards();

return new ResponseEntity<>(rewards, HttpStatus.OK);

}



@Operation(summary = "Get reward by ID", description = "Returns a single reward by
ID")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Successfully retrieved the
reward", content = @Content(schema = @Schema(implementation = Reward.class))),

@ApiResponse(responseCode = "404", description = "Reward not found")

})

@GetMapping("/{id}")

public ResponseEntity<Reward> getRewardById(@PathVariable Long id) {

Optional<Reward> reward = rewardService.getRewardById(id);

return reward.map(value -> new ResponseEntity<>(value, HttpStatus.OK))

.orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));

}



@Operation(summary = "Get reward by customer ID", description = "Returns a reward
associated with a customer")

@GetMapping("/customer/{customerId}")

public ResponseEntity<Reward> getRewardByCustomerId(@PathVariable Long customerId) {

Optional<Reward> reward = rewardService.getRewardByCustomerId(customerId);

return reward.map(value -> new ResponseEntity<>(value, HttpStatus.OK))

.orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));

}
```

```
@Operation(summary = "Create a new reward", description = "Creates a new reward")

@ApiResponses(value = {

@ApiResponse(responseCode = "201", description = "Reward created successfully",
content = @Content(schema = @Schema(implementation = Reward.class)))

})

@PostMapping

public ResponseEntity<Reward> createReward(@RequestBody Reward reward) {

Reward createdReward = rewardService.saveReward(reward);

return new ResponseEntity<>(createdReward, HttpStatus.CREATED);

}



@Operation(summary = "Update an existing reward", description = "Updates a reward by
ID")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Reward updated successfully",
content = @Content(schema = @Schema(implementation = Reward.class))),

@ApiResponse(responseCode = "404", description = "Reward not found")

})

@PutMapping("/{id}")

public ResponseEntity<Reward> updateReward(@PathVariable Long id, @RequestBody
Reward reward) {

Optional<Reward> existingReward = rewardService.getRewardById(id);



if (existingReward.isPresent()) {

reward.setId(id);

Reward updatedReward = rewardService.updateReward(reward);

return new ResponseEntity<>(updatedReward, HttpStatus.OK);

} else {

return new ResponseEntity<>(HttpStatus.NOT_FOUND);
```

```
}

}


@Operation(summary = "Update reward points", description = "Updates only the points
of a reward")

@PatchMapping("/{id}/points")

public ResponseEntity<Reward> updateRewardPoints(

@PathVariable Long id,

@RequestParam int points) {


Optional<Reward> existingReward = rewardService.getRewardById(id);


if (existingReward.isPresent()) {

Reward reward = existingReward.get();

reward.setPoints(points);

Reward updatedReward = rewardService.updateReward(reward);

return new ResponseEntity<>(updatedReward, HttpStatus.OK);

} else {

return new ResponseEntity<>(HttpStatus.NOT_FOUND);

}

}


@Operation(summary = "Delete a reward", description = "Deletes a reward by ID")

@ApiResponses(value = {

@ApiResponse(responseCode = "204", description = "Reward deleted successfully"),

@ApiResponse(responseCode = "404", description = "Reward not found")

})
```

```
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteReward(@PathVariable Long id) {
Optional<Reward> reward = rewardService.getRewardById(id);

if (reward.isPresent()) {
rewardService.deleteReward(id);
return new ResponseEntity<>(HttpStatus.NO_CONTENT);
} else {
return new ResponseEntity<>(HttpStatus.NOT_FOUND);
}
}
}
```

Repository

Appointment Repository

```
package com.example.demo.Repository;

import com.example.demo.entity.Appointment;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.time.LocalDateTime;
import java.util.List;

@Repository
public interface AppointmentRepository extends JpaRepository<Appointment, Long> {
List<Appointment> findByCustomerId(Long customerId);
List<Appointment> findByStatus(String status);
```

```
List<Appointment> findByAppointmentTimeBetween(LocalDateTime start, LocalDateTime
end);


}
```

CustomerRepository

```
package com.example.demo.Repository;


import com.example.demo.entity.Customer;

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

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

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;


import java.util.Optional;


@Repository

public interface CustomerRepository extends JpaRepository<Customer, Long> {

Customer findByUsername(String username);


boolean existsByUsername(String username);


boolean existsByEmail(String email);


@Query("SELECT c FROM Customer c LEFT JOIN FETCH c.appointments WHERE c.id = :id")

Optional<Customer> findByIdWithAppointments(@Param("id") Long id);


}
```

RewardRepository

```
package com.example.demo.Repository;


import com.example.demo.entity.Reward;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;


import java.util.Optional;


@Repository
public interface RewardRepository extends JpaRepository<Reward, Long> {
Optional<Reward> findByCustomerId(Long customerId);


}
```

Entity

Appointment

```
package com.example.demo.entity;


import java.time.LocalDateTime;


import javax.persistence.*;



@Entity
@Table(name = "appointments")
public class Appointment {
```

```
@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;


@Column()

private String serviceName;


@Column()

private LocalDateTime appointmentTime;


@Column()

private String status;


@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "customer_id")

private Customer customer;


// Constructors

public Appointment() {

}


public Appointment(String serviceName, LocalDateTime appointmentTime, String status)
{

this.serviceName = serviceName;

this.appointmentTime = appointmentTime;

this.status = status;

}
```

```
// Getters and Setters
public Long getId() {
return id;
}

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

public String getServiceName() {
return serviceName;
}

public void setServiceName(String serviceName) {
this.serviceName = serviceName;
}

public LocalDateTime getAppointmentTime() {
return appointmentTime;
}

public void setAppointmentTime(LocalDateTime appointmentTime) {
this.appointmentTime = appointmentTime;
}

public String getStatus() {
```

```
return status;

}


public void setStatus(String status) {

this.status = status;

}


public Customer getCustomer() {

return customer;

}


public void setCustomer(Customer customer) {

this.customer = customer;

}


@Override

public String toString() {

return "Appointment [id=" + id + ", serviceName=" + serviceName + ",
appointmentTime=" + appointmentTime

+ ", status=" + status + "]";

}

}
```

Customer

```
package com.example.demo.entity;


import java.util.ArrayList;

import java.util.List;
```

```
import javax.persistence.*;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity
@Table(name = "customers")
public class Customer {

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Column()
private String username;

@Column()
private String email;

@JsonIgnore
@Column()
private String password;

@OneToMany(mappedBy = "customer", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
private List<Appointment> appointments = new ArrayList<>();
```

```
@OneToOne(mappedBy = "customer", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
private Reward reward;

// Constructors
public Customer() {
}

public Customer(String username, String email, String password) {
this.username = username;
this.email = email;
this.password = password;
}

// Getters and Setters
public Long getId() {
return id;
}

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

public String getUsername() {
return username;
}

public void setUsername(String username) {
```

```
this.username = username;
}

public String getEmail() {
return email;
}

public void setEmail(String email) {
this.email = email;
}

public String getPassword() {
return password;
}

public void setPassword(String password) {
this.password = password;
}

public List<Appointment> getAppointments() {
return appointments;
}

public void setAppointments(List<Appointment> appointments) {
this.appointments = appointments;
}
```

```
public Reward getReward() {
return reward;
}

public void setReward(Reward reward) {
this.reward = reward;
}

// Helper methods
public void addAppointment(Appointment appointment) {
appointments.add(appointment);
appointment.setCustomer(this);
}

public void removeAppointment(Appointment appointment) {
appointments.remove(appointment);
appointment.setCustomer(null);
}

@Override
public String toString() {
return "Customer [id=" + id + ", username=" + username + ", email=" + email + "]";
}
}
```

Reward

```
package com.example.demo.entity;
```

```
import javax.persistence.*;


@Entity

@Table(name = "rewards")

public class Reward {


@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;


@Column(nullable = false)

private int points;


@Column

private String name;


@OneToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "customer_id")

private Customer customer;


// Constructors

public Reward() {

}


public Reward(int points) {

this.points = points;

}
```

```
public Reward(String name, int points) {
this.name = name;
this.points = points;
}

// Getters and Setters
public Long getId() {
return id;
}

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

public int getPoints() {
return points;
}

public void setPoints(int points) {
this.points = points;
}

public String getName() {
return name;
}
```

```
public void setName(String name) {
this.name = name;
}

public Customer getCustomer() {
return customer;
}

public void setCustomer(Customer customer) {
this.customer = customer;
}

@Override
public String toString() {
return "Reward [id=" + id + ", points=" + points + ", name=" + name + "]";
}
}
```

SwaggerConfiguration

```
package com.example.demo.configuration;

import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Info;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SwaggerConfig {
```

```
@Bean
public OpenAPI customOpenAPI() {
return new OpenAPI()
.info(new Info()
.title("My API")
.version("1.0")
.description("API documentation using Swagger"));
}
}
```

LoggingAspect

```
package com.example.demo.aspect;

import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LoggingAspect {

private static final Logger logger = LoggerFactory.getLogger(LoggingAspect.class);

@Before("execution(* com.example.demo.service.*.*(..))")
```

```
public void logBeforeMethodExecution() {
logger.info("Method execution started.");
}

@AfterReturning("execution(* com.example.demo.service.*.*(..))")
public void logAfterMethodExecution() {
logger.info("Method execution completed.");
}
}
```

Service

Appointmentservice

```
package com.example.demo.service;

import com.example.demo.Repository.AppointmentRepository;
import com.example.demo.entity.Appointment;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

@Service
public class AppointmentService {
```

```
private final AppointmentRepository appointmentRepository;

@Autowired
public AppointmentService(AppointmentRepository appointmentRepository) {
this.appointmentRepository = appointmentRepository;
}

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

public Page<Appointment> getAllAppointmentsPaged(Pageable pageable) {
return appointmentRepository.findAll(pageable);
}

public Optional<Appointment> getAppointmentById(Long id) {
return appointmentRepository.findById(id);
}

@Transactional
public Appointment saveAppointment(Appointment appointment) {
return appointmentRepository.save(appointment);
}

@Transactional
public void deleteAppointment(Long id) {
```

```
appointmentRepository.deleteById(id);
}

public List<Appointment> getAppointmentsByCustomerId(Long customerId) {
return appointmentRepository.findByCustomerId(customerId);
}

public List<Appointment> getAppointmentsByStatus(String status) {
return appointmentRepository.findByStatus(status);
}

@Transactional
public Appointment updateAppointmentStatus(Long id, String status) {
Optional<Appointment> appointmentOpt = appointmentRepository.findById(id);
if (appointmentOpt.isPresent()) {
Appointment appointment = appointmentOpt.get();
appointment.setStatus(status);
return appointmentRepository.save(appointment);
}
return null;
}
}
```

Customerservice

```
package com.example.demo.service;

import com.example.demo.Repository.CustomerRepository;
import com.example.demo.entity.Customer;
```

```
import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;


import java.util.List;

import java.util.Optional;


@Service

public class CustomerService {


private final CustomerRepository customerRepository;


@Autowired

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}


public List<Customer> getAllCustomers() {

return customerRepository.findAll();

}


public Optional<Customer> getCustomerById(Long id) {

return customerRepository.findById(id);

}


public Optional<Customer> getCustomerWithAppointments(Long id) {
```

```
return customerRepository.findByIdWithAppointments(id);
}

public Customer getCustomerByUsername(String username) {
return customerRepository.findByUsername(username);
}

@Transactional
public Customer saveCustomer(Customer customer) {
return customerRepository.save(customer);
}

@Transactional
public void deleteCustomer(Long id) {
customerRepository.deleteById(id);
}

public boolean existsByUsername(String username) {
return customerRepository.existsByUsername(username);
}

public boolean existsByEmail(String email) {
return customerRepository.existsByEmail(email);
}
}
```

RewardService

```
package com.example.demo.service;
```

```
import com.example.demo.Repository.RewardRepository;
import com.example.demo.entity.Reward;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

@Service
public class RewardService {

private final RewardRepository rewardRepository;

@Autowired
public RewardService(RewardRepository rewardRepository) {
this.rewardRepository = rewardRepository;
}

public List<Reward> getAllRewards() {
return rewardRepository.findAll();
}

public Optional<Reward> getRewardById(Long id) {
return rewardRepository.findById(id);
```

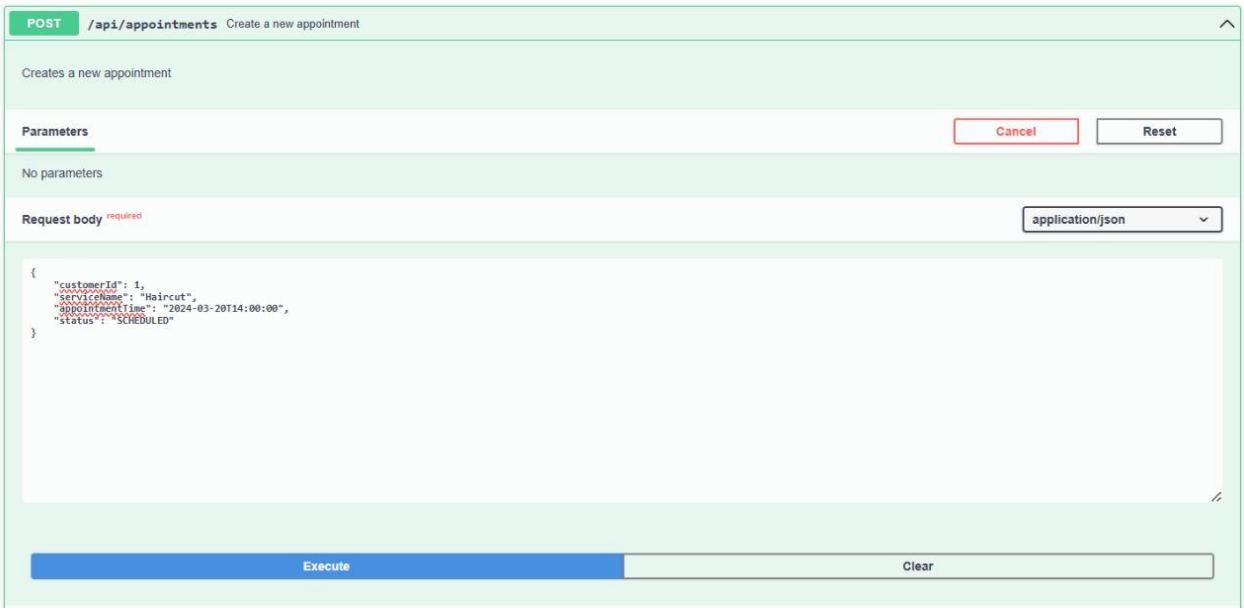
```
}

public Optional<Reward> getRewardByCustomerId(Long customerId) {
return rewardRepository.findByCustomerId(customerId);
}

@Transactional
public Reward saveReward(Reward reward) {
return rewardRepository.save(reward);
}

@Transactional
public Reward updateReward(Reward reward) {
// First check if the reward exists
if (reward.getId() != null && rewardRepository.existsById(reward.getId())) {
return rewardRepository.save(reward);
}
return null;
}

@Transactional
public void deleteReward(Long id) {
rewardRepository.deleteById(id);
}

@Transactional
public Reward addPoints(Long id, int points) {
```

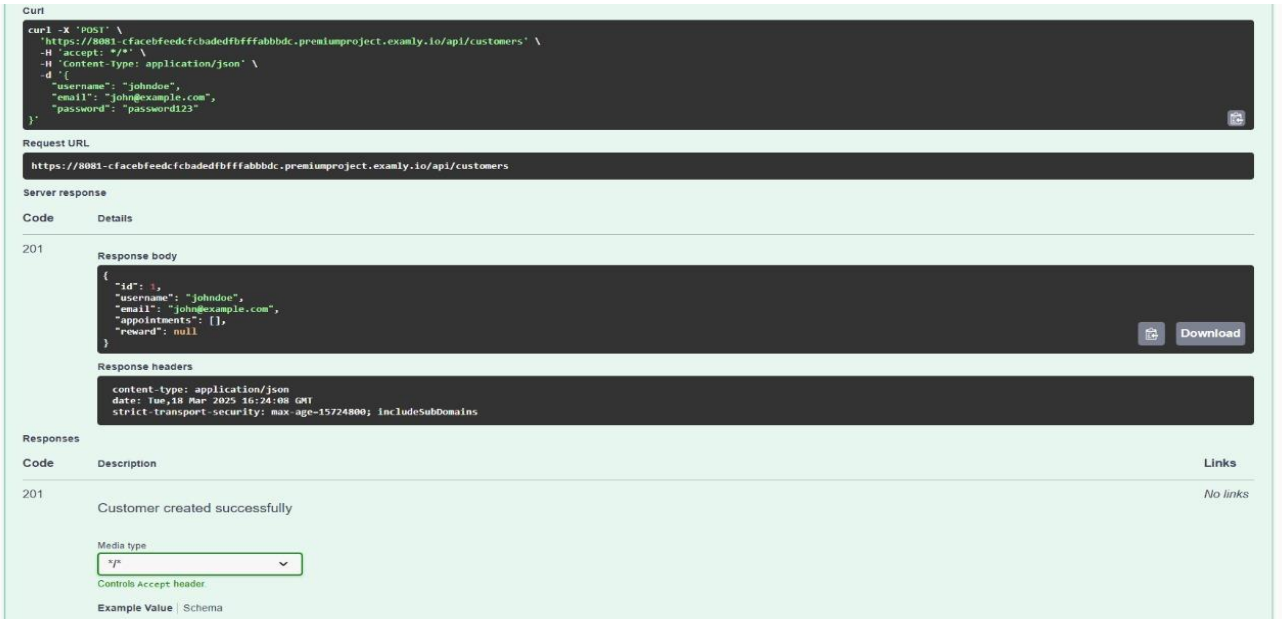
```
Optional<Reward> rewardOpt = rewardRepository.findById(id);

if (rewardOpt.isPresent()) {

Reward reward = rewardOpt.get();

reward.setPoints(reward.getPoints() + points);

return rewardRepository.save(reward);

}

return null;

}

}
```

CRUD OPERATION

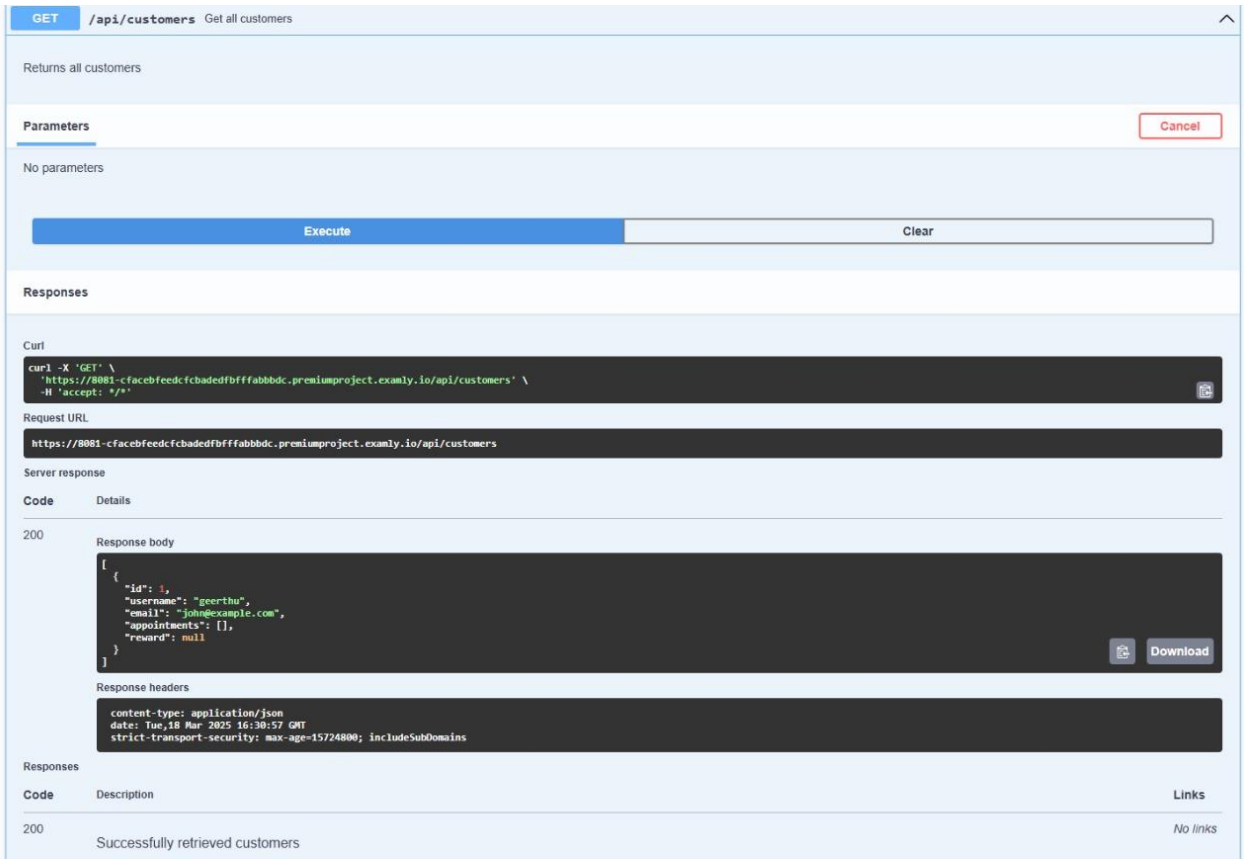
1.CUSTOMER CONTROLLER

POST OPERATION

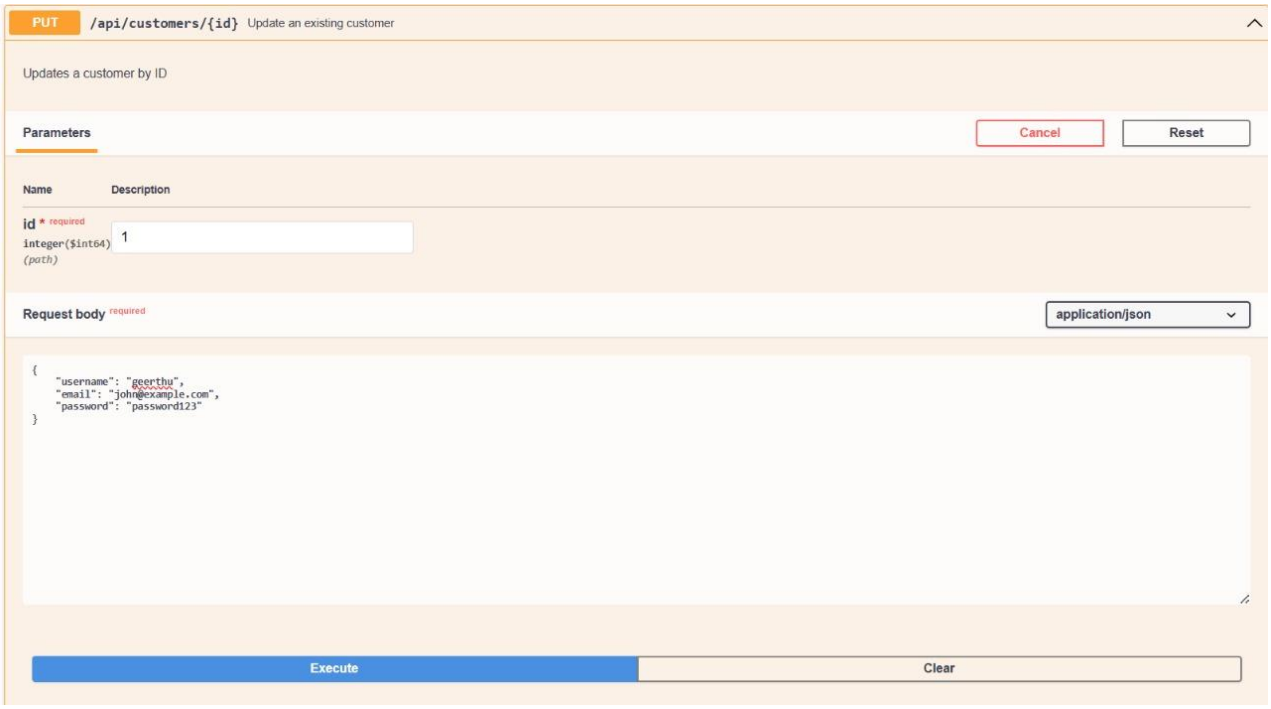


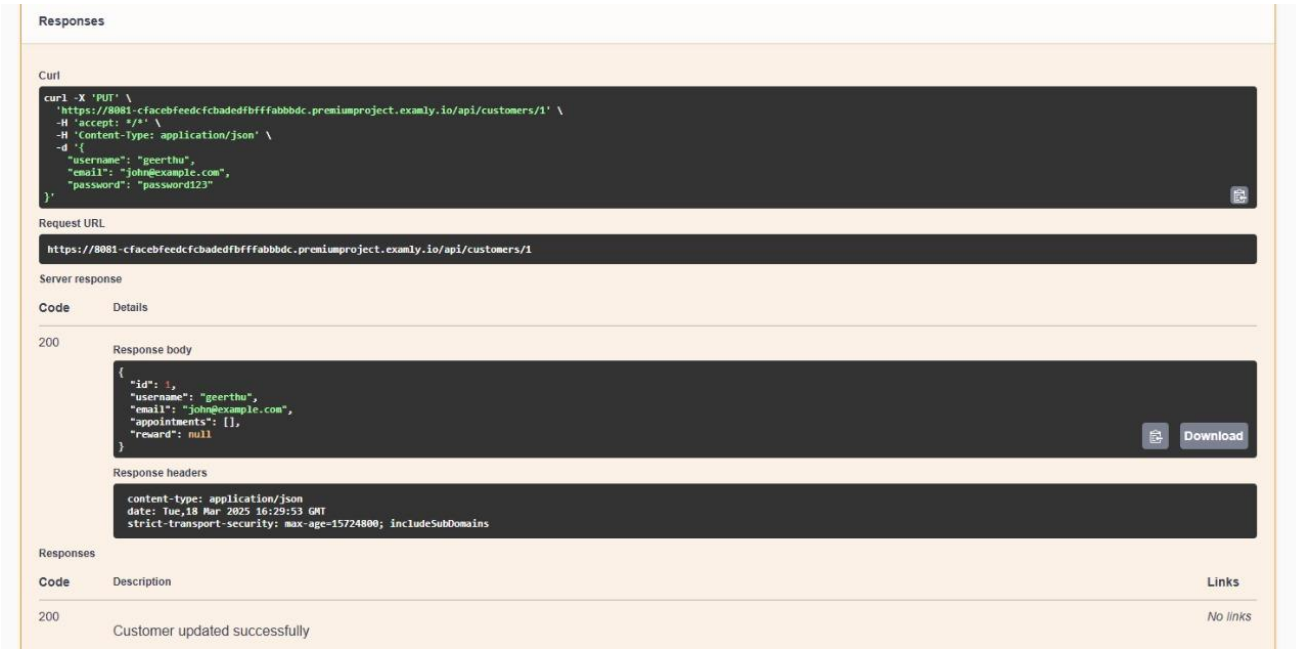


GET OPERATION

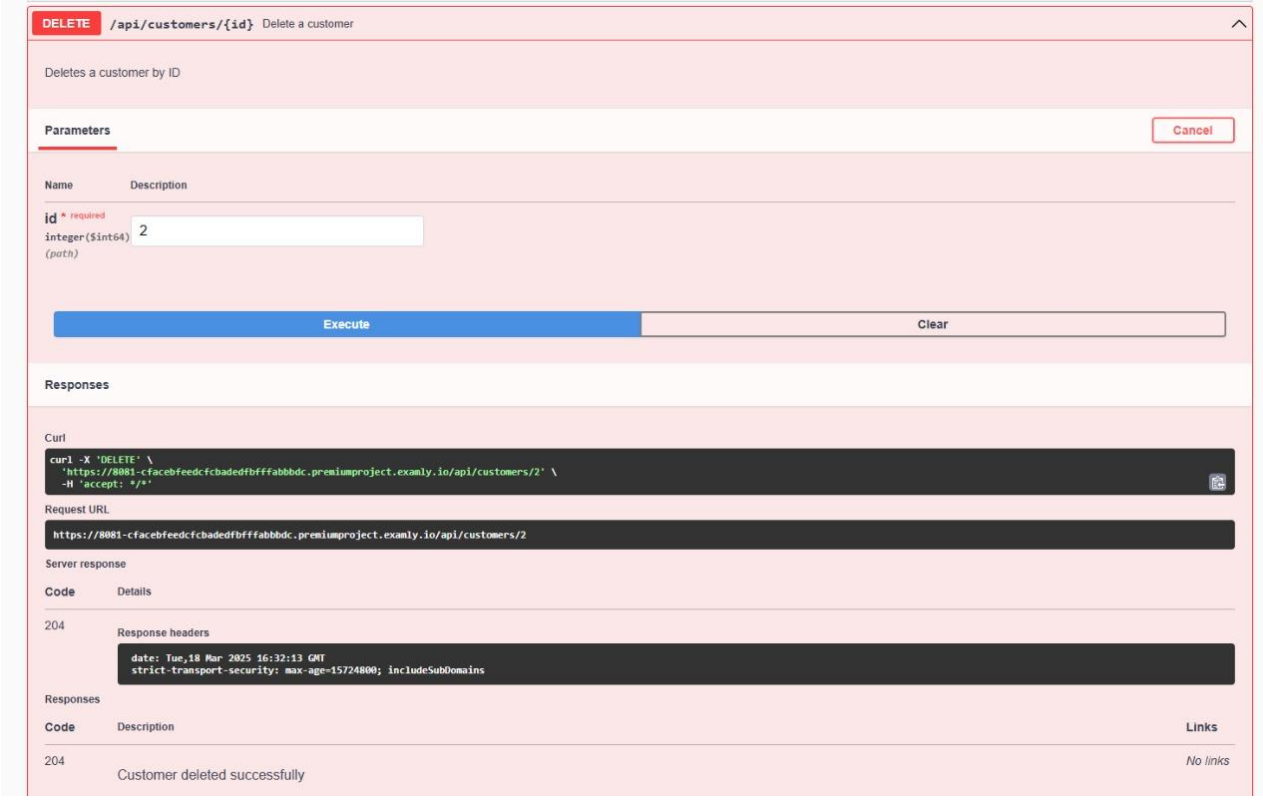


PUT OPERATION



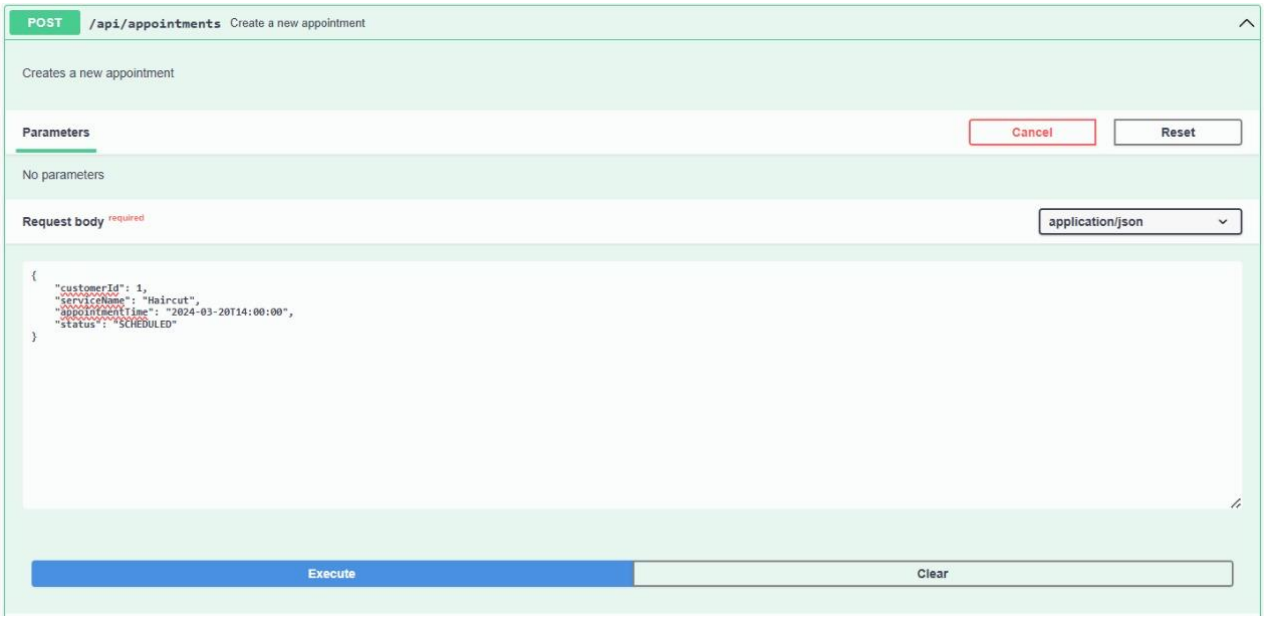


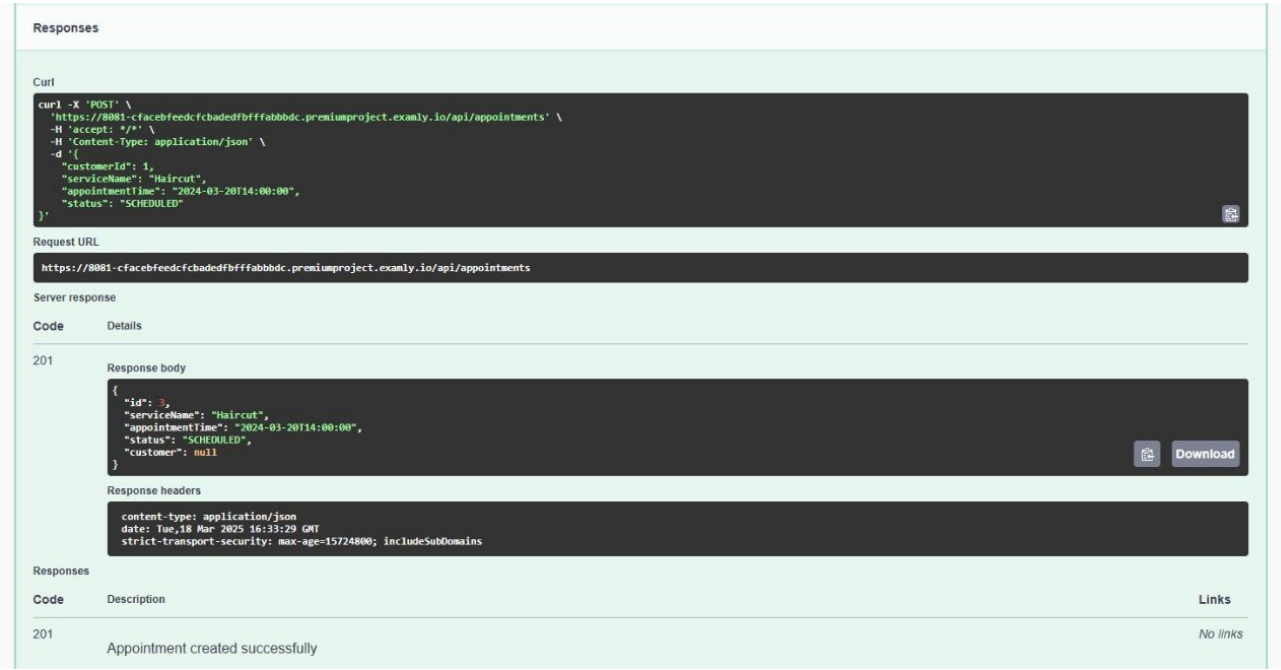
DELETE OPERATION



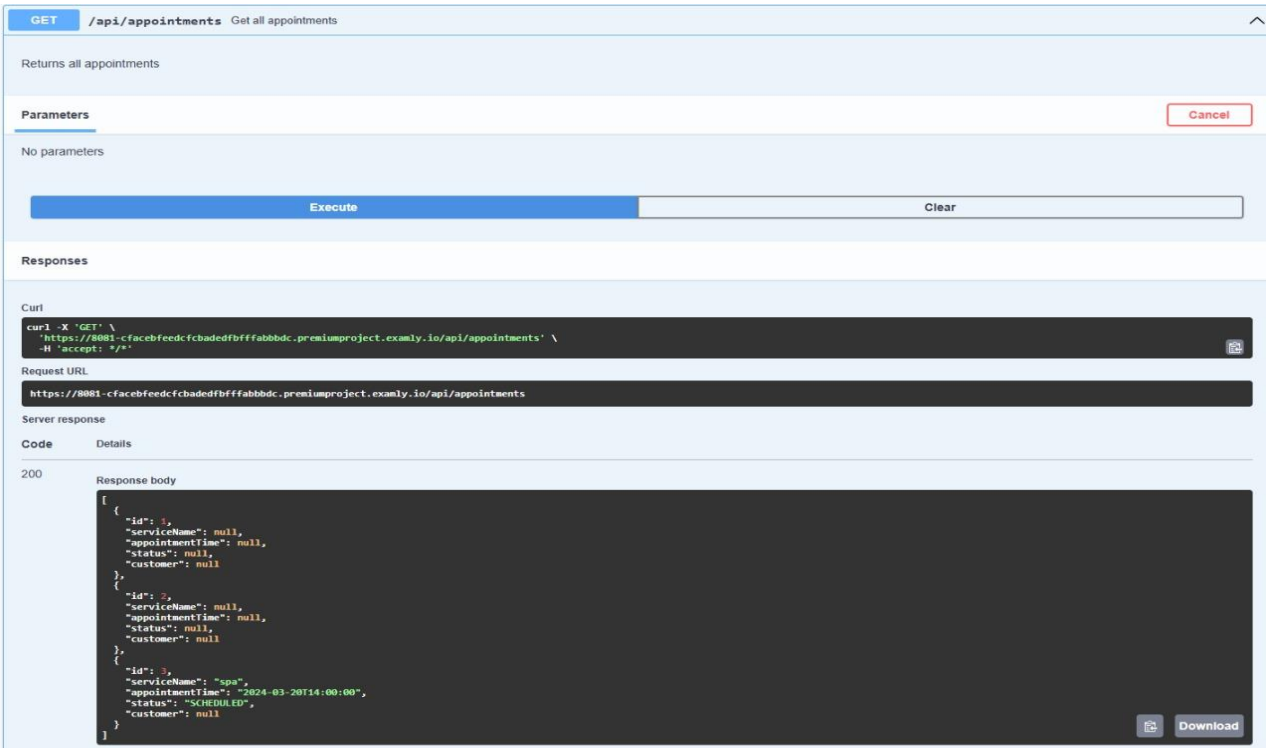
2.APPOINTMENT CONTROLLER

POST OPERATION

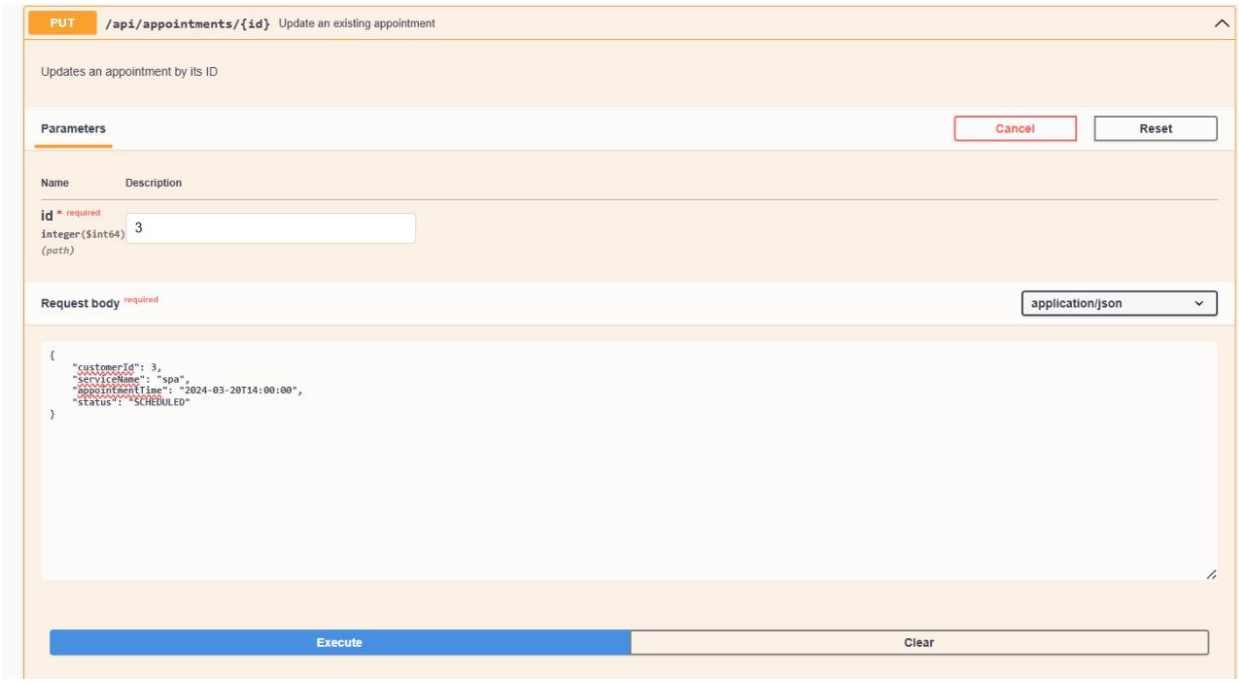


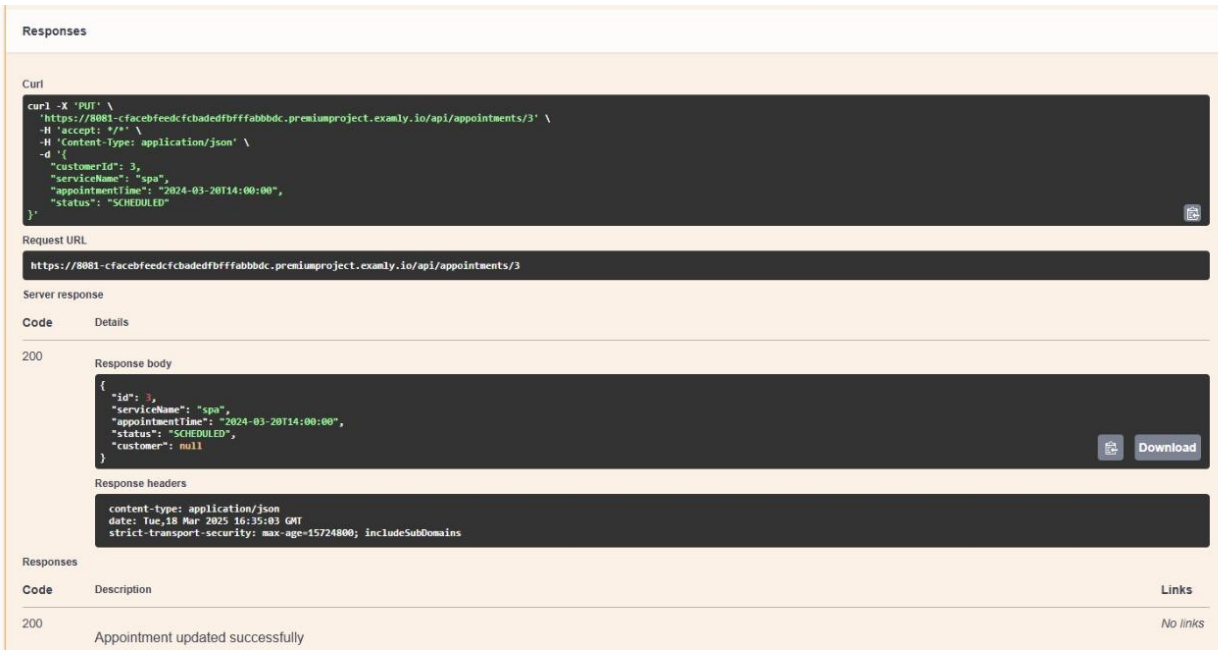


GET OPERATION

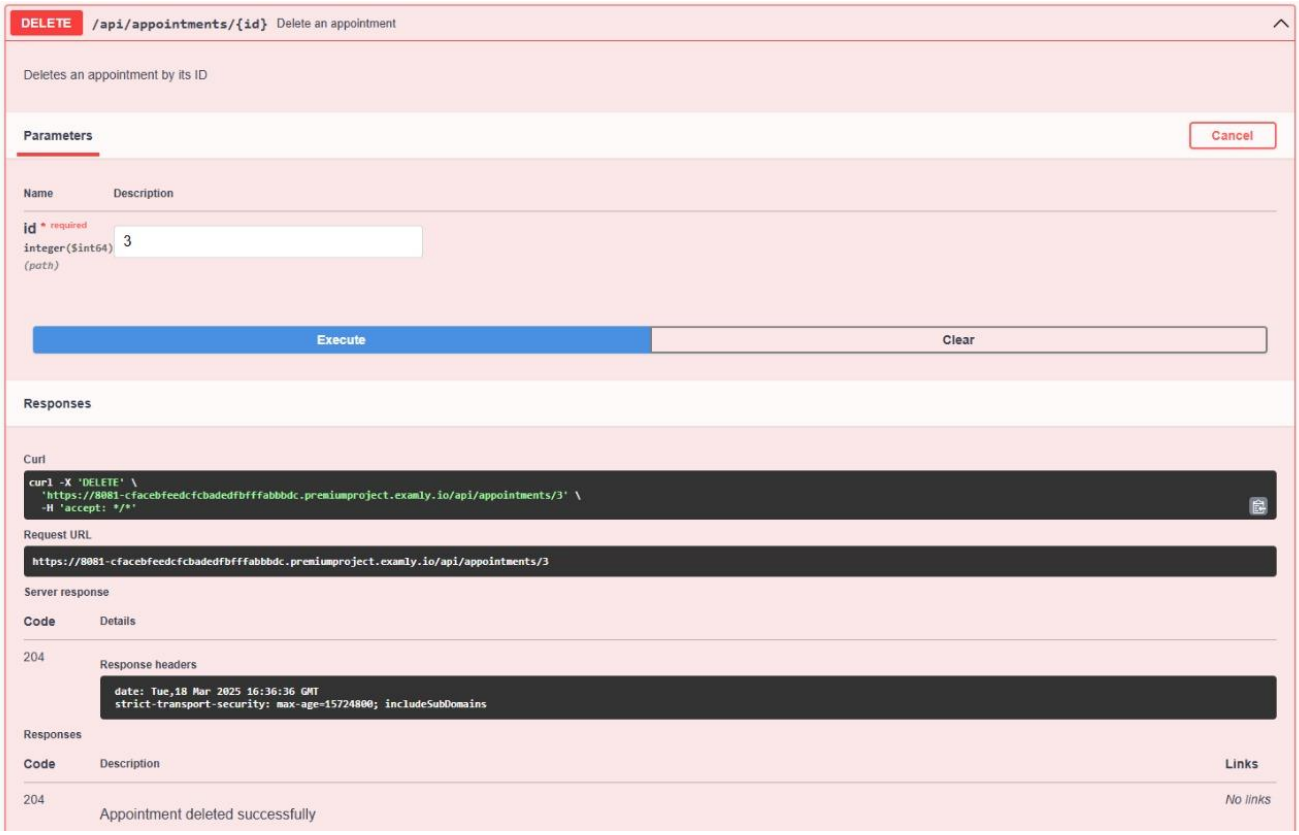


PUT OPERATION



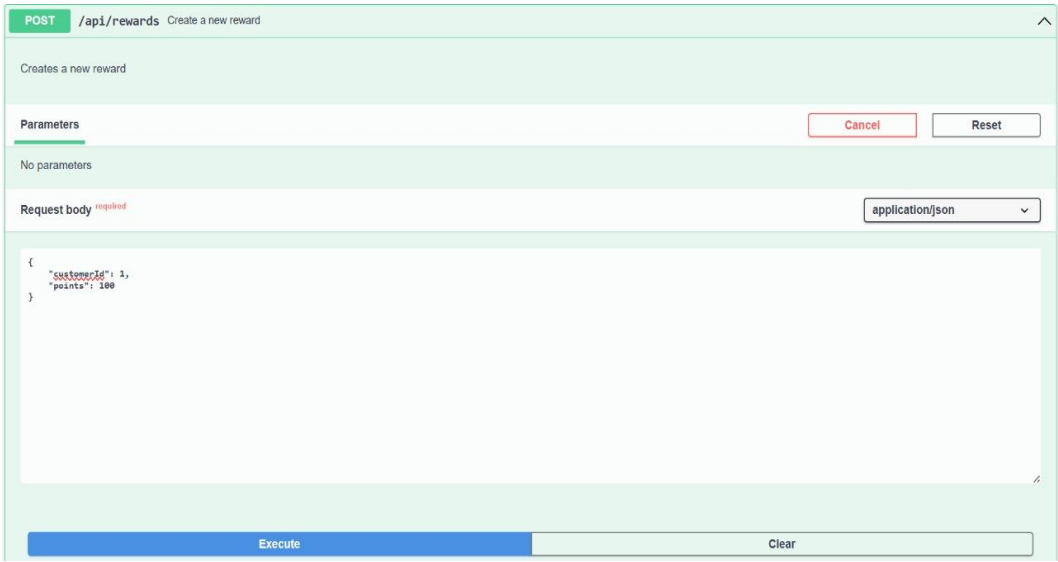


DELETE OPERATION



3.REWARD CONTROLLER

POST OPERATION



Responses

Curl

```
curl -X 'POST' \
  'https://8081-cfacebfeedcfcbadedfbfffabbbdc.premiumproject.examly.io/api/rewards' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "customerId": 1,
    "points": 100
}'
```

Request URL

```
https://8081-cfacebfeedcfcbadedfbfffabbbdc.premiumproject.examly.io/api/rewards
```

Server response

Code	Details
201	Response body

```
{
  "id": 2,
  "points": 100,
  "name": null,
  "customer": null
}
```

Download

Response headers

```
content-type: application/json
date: Tue,18 Mar 2025 16:38:24 GMT
strict-transport-security: max-age=15724800; includeSubDomains
```

Responses

Code	Description	Links
201	Reward created successfully	*No links*

GET OPERATION

GET /api/rewards Get all rewards

Returns all rewards

Parameters Cancel

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'https://8081-cfacebfeedcfcbadedfbfffabbbdc.premiumproject.examly.io/api/rewards' \
  -H 'accept: */*'
```

Request URL

```
https://8081-cfacebfeedcfcbadedfbfffabbbdc.premiumproject.examly.io/api/rewards
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 1,
    "points": 100,
    "name": null,
    "customer": null
  },
  {
    "id": 2,
    "points": 100,
    "name": null,
    "customer": null
  }
]
```

Download

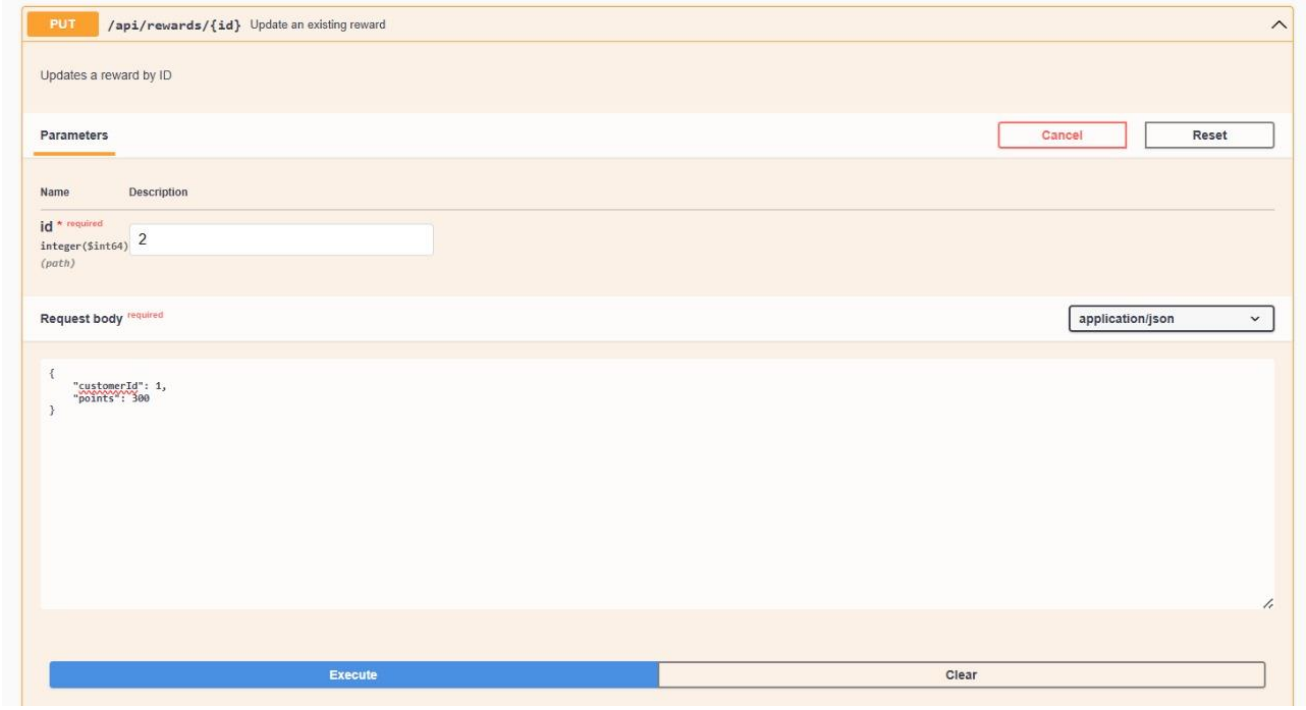
Response headers

```
content-type: application/json
date: Tue,18 Mar 2025 16:40:17 GMT
strict-transport-security: max-age=15724800; includeSubDomains
```

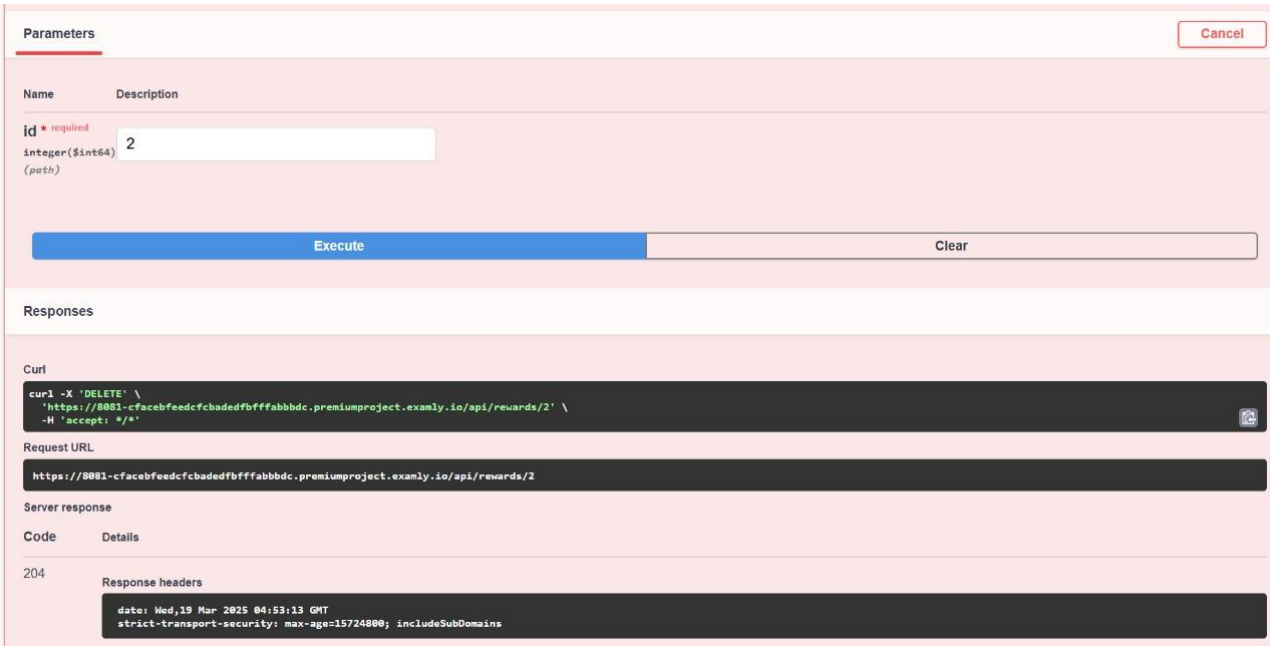
Responses

Code	Description	Links
200	Successfully retrieved rewards	*No links*

PUT OPERATION



DELETE OPERATION



CHAPTER 6

CONCLUSION

6.1 CONCLUSION

The **Mobile Beauty Service Application** is an innovative platform designed to revolutionize the beauty and wellness industry by providing on-demand, professional beauty services at customers' preferred locations. By integrating **automated appointment booking, AI-powered service recommendations, real-time beautician tracking, and secure transactions**, the application ensures convenience, efficiency, and quality service delivery.

With a user-friendly interface, customers can easily browse services, select skilled beauticians, and receive personalized beauty treatments, eliminating the need for salon visits. The platform also benefits beauticians by offering **business management tools, scheduling assistance, and an expanded client base**, allowing them to enhance their earnings and streamline their operations.

By leveraging **cutting-edge technology such as AI-driven recommendations, secure digital**

payments, and real-time notifications, the application ensures a seamless experience for both customers and service providers. The inclusion of **customer reviews, personalized beauty plans, and loyalty programs** further enhances user engagement and retention.

6.2 FUTURE SCOPE

The **Mobile Beauty Service Application** has immense potential for future enhancements and expansion. Below are some key features and improvements that could be integrated into upcoming versions:

Advanced AI & Personalization Features

- **AI-Powered Beauty Recommendations:** AI-driven algorithms could analyze user preferences, skin/hair type, and past bookings to suggest the most suitable beauty treatments.
- **Virtual Beauty Consultations:** Integration of AI-powered virtual consultations where users can get personalized beauty advice from professional stylists and dermatologists.
- **Seasonal Beauty Tips & Trends:** The app could provide personalized recommendations based on climate, seasonal trends, and skincare/haircare needs.

Enhanced Service Offerings & Booking Features

- **Group Booking System:** Users could schedule beauty services for groups, ideal for weddings, parties, or spa days with friends.
- **Subscription-Based Memberships:** Monthly or annual beauty service plans offering discounts and priority bookings.
- **Custom Beauty Packages:** Users could create customized service bundles (e.g., facial + hair spa + manicure) based on their needs.

Enhanced Beautician Empowerment & Business Growth

- **Beautician Training & Certification Programs:** Offering online training courses and certifications to help beauticians improve their skills and expand their expertise.
- **Income & Performance Analytics Dashboard:** Beauticians could track their earnings,

analyze customer feedback, and optimize their service offerings.

- **Dynamic Pricing System:** AI-driven dynamic pricing based on demand, beautician expertise, and peak service hours.

Community & Social Features

- **Beauty Blogs & Video Tutorials:** Experts could share beauty tips, tutorials, and skincare/haircare advice within the app.
- **User-Generated Content & Reviews:** Customers could share beauty transformations, before-and-after images, and detailed service reviews.
- **Referral & Rewards System:** Users could earn points and discounts for referring friends and leaving feedback.

Integration with Smart Devices & Wearables

- **Smart Mirror Connectivity:** Syncing with smart mirrors to provide real-time skin analysis and beauty tips.
- **Wearable Beauty Monitoring:** Integration with smartwatches and fitness bands to monitor skin hydration levels and recommend beauty routines.

Grocery Store & Product Partnerships

- **Beauty Product Marketplace:** Users could purchase recommended skincare, haircare, and beauty products directly within the app.
- **Partnerships with Brands & Salons:** Collaboration with beauty brands and professional salons to offer exclusive discounts and promotions.
- **Customized Beauty Boxes:** Subscription-based beauty boxes curated by experts based on user preferences.

Sustainability & Eco-Friendly Initiatives

- **Eco-Friendly Product Recommendations:** Highlighting sustainable beauty products and

organic treatments.

- **Carbon Footprint Tracking:** Providing users insights into the environmental impact of their beauty choices and encouraging greener alternatives.
- **Paperless Receipts & Digital Invoices:** Encouraging digital transactions to reduce paper waste and promote sustainability.

Multi-Language & Global Expansion

- **Localized Service Listings:** Adapting services and pricing for different regions based on market demand.
- **Multi-Language Support:** Expanding the app's language options to cater to diverse user bases worldwide.
- **Cultural Beauty Recommendations:** Customizing beauty tips and services to reflect different cultural preferences and traditions.

Blockchain for Beautician Verification & Service Transparency

- **Verified Beautician Credentials:** Using blockchain technology to verify beauticians' certifications, experience, and ratings.
- **Transparent Service History:** Blockchain could track customer interactions, ensuring credibility and trust between beauticians and clients.
- **Secure Transaction Records:** Ensuring financial transparency and preventing fraud through blockchain-enabled transactions.

REFERENCES

- https://spring.io/projects/spring-boot
- https://react.dev/
- https://dev.mysql.com/doc/