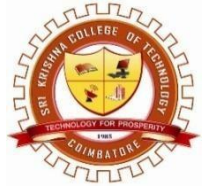




SRI KRISHNA COLLEGE OF TECHNOLOGY
(An Autonomous Institution)
Approved by AICTE | Affiliated to Anna University Chennai
Accredited by NBA - AICTE | Accredited by NAAC with 'A' Grade
KOVAIPUDUR, COIMBATORE 641042



Budget Management System

23IT402- WEB FRAMEWORKS USING REST API

A PROJECT REPORT

Submitted by

Prem S - 727823TUIO039

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

(INTERNET OF THINGS)

MARCH 2025

BONAFIDE CERTIFICATE

Certified that this project report **BUDGET MANAGEMENT SYSTEM** is the Bonafide work of **PREM S 727823TUIO039** who carried out the project work under my supervision.

SIGNATURE

MR.S.SHANMUGA RAJU
SUPERVISOR,
Assistant Professor,
Department of Internet of Things,
Sri Krishna College of Technology,
Coimbatore-641042.

SIGNATURE

Dr.SUMA SIRA JACOB,
PROGRAMME COORDINATOR,
Head Of Department,
Department of Internet Of Things,
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 the project.

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

We would like to express our deep gratitude to **Dr. T. Senthilnathan**, Dean of Computing Sciences and **Dr. Suma Sira Jacob**, Programme Coordinator of Internet Of Things, for their exceptional dedication and care towards success of the project.

We would like to extend our heartfelt gratitude to our Project guide **Ms. S. Shanmuga Raju** Assistant Professor, Department of Internet of Things for his valuable guidance and suggestions in all aspects that aided us to meliorate our skills.

We are thankful to all the **Teaching and Non-Teaching Staff** of Department of Information Technology 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 endeavor.

ABSTRACT

The Budget Management System is a modern, technology-driven platform designed to simplify and enhance financial management. It provides users with a seamless way to track, analyze, and manage their income, expenses, investments, and overall budget. With features such as real-time financial tracking, secure data management, and an intuitive reporting interface, the system ensures a user-friendly and efficient experience for all stakeholders, including individuals, businesses, and financial analysts.

This platform enables users to effortlessly categorize transactions, set financial goals, and generate insightful reports on their financial health. It also incorporates a comprehensive admin module that allows financial managers and business owners to efficiently handle budget allocations, investment portfolios, and expense tracking. Through an automated calculation and analysis system, users can ensure accurate financial planning, reducing errors and improving decision-making.

Financial advisors play a crucial role in the system, utilizing dedicated features that allow them to review financial trends, suggest budget optimizations, and ensure effective fund management. The system facilitates efficient financial forecasting and real-time analysis, ensuring better financial stability and growth.

Designed with scalability and security in mind, the Budget Management System leverages Spring Boot 3, Spring Data JPA, and MySQL to provide a robust and high-performance backend infrastructure. Authentication mechanisms, such as JWT-based security and role-based access control, protect sensitive financial data and prevent unauthorized access. The system is built to accommodate growing financial needs, making it suitable for individuals, startups, and enterprises alike.

By digitizing traditional financial management, this system not only enhances convenience for users but also optimizes business operations. It improves financial planning efficiency, reduces manual errors, and ensures a smooth, accurate, and reliable budgeting experience. Whether for individuals, business owners, or financial professionals, the Budget Management System is an innovative and indispensable solution for effective financial management.

TABLE OF CONTENT

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	IV
	LIST OF FIGURES	V
1	INTRODUCTION	01
	1.1.PROBLEM STATEMENT	01
	1.2.OVERVIEW	01
	1.3.OBJECTIVES	02
2	SYSTEM SPECIFICATIONS	03
	2.1.SPRING BOOT	03
	2.2.MYSQL	03
	2.3.SWAGGER UI	04
3	PROPOSED SYSTEM	05
	3.1.PROPOSED WORK	05
	3.2.ADVANTAGES	06
4	METHODOLOGIES	07
	4.1.FLOW DIAGRAM	09
	4.2.CLASS DIAGRAM	10
	4.3.SEQUENCE DIAGRAM	11
	4.4.ER DIAGRAM	12
5	IMPLEMENTATION AND RESULT	13
	5.1.CODING	13
	5.2.OUTPUT	40
6	CONCLUSION AND FUTURE SCOPE	43
	6.1.CONCLUSION	43
	6.2.FUTURE SCOPE	43

LIST OF FIGURES

Figure No.	TITLE	Page No
4.1	Flow diagram	09
4.2	Class diagram	10
4.3	Sequence Diagram	11
4.4	ER Diagram	12
5.1	Console	40
5.2	Swagger UI	40
5.3	Mysql	41
5.4	Implementation of EmailSMTP	41
5.5	Implementation of Weather API	42

CHAPTER 1

INTRODUCTION

1.1. The Budget Management System is a digital platform that enables users to track, manage, and optimize their financial resources efficiently. It offers real-time expense monitoring, secure data handling, and financial goal-setting for a seamless budgeting experience. Administrators and financial managers can efficiently oversee income, expenses, investments, and portfolio allocations to ensure financial stability. Users benefit from automated calculations, insightful reports, and real-time financial analytics, enabling informed decision-making. Built with scalable and secure technology, the system enhances efficiency and data protection. This platform revolutionizes financial management by providing a smart, reliable, and user-friendly solution for modern individuals and businesses.

1.2. PROBLEM STATEMENT

Traditional financial management methods involve inefficiencies such as manual expense tracking, lack of real-time insights, and no centralized system for monitoring income, expenses, and investments. Users face challenges in budgeting, analyzing financial data, and making informed financial decisions.

Business owners and individuals struggle to maintain accurate financial records, track expenditures effectively, and plan investments efficiently. The absence of an integrated solution leads to financial mismanagement, errors, and difficulties in achieving financial goals.

A robust, user-friendly system is required to address these challenges, offering streamlined financial tracking, insightful analytics, and an efficient budgeting process for individuals and businesses.

1.3. OVERVIEW

The **Budget Management System** is a digital platform that allows users to track, manage, and optimize their financial resources efficiently. It enhances financial planning with real-time expense monitoring, secure data management, and automated budget analysis, reducing the risk of financial mismanagement.

Admins oversee income, expenses, and investment portfolios, while users can set financial goals and generate insightful reports for better decision-making. Built with **Spring Boot 3, Spring Data JPA, and MySQL**, the system ensures scalability, security, and efficiency.

By automating financial tracking and analysis, this system optimizes budgeting, improves financial stability, and provides a seamless experience for individuals and businesses managing their finances.

1.4. OBJECTIVES:

The objective of the **Budget Management System** is to provide a seamless and efficient financial management experience for users by enabling them to track income, expenses, and investments in a structured manner. It aims to enhance financial planning through real-time expense monitoring, secure data management, and automated budget analysis.

The system also focuses on improving financial oversight by allowing administrators to monitor transactions, analyze spending patterns, and generate insightful reports effectively. Additionally, it optimizes financial decision-making by streamlining budget tracking, reducing errors, and ensuring accurate financial forecasting. Users benefit from real-time data insights and goal-setting features, making financial management more efficient.

Security and data protection are also key objectives, ensuring safe financial transactions and protecting user information. Designed for scalability, the system supports both individuals and businesses, helping them achieve financial stability and growth. By providing a user-friendly interface and intelligent financial recommendations, the system enhances financial awareness while improving overall budgeting efficiency.

CHAPTER 2

SYSTEM SPECIFICATION

2.1. SPRING BOOT

The grocery delivery system is an advanced digital platform designed to streamline the process of purchasing and delivering groceries. It enables customers to browse a wide range of products, add items to their cart, and complete transactions seamlessly with secure payment options. The system offers real-time order tracking, ensuring transparency and convenience for users. With an efficient admin panel, store owners can manage inventory, pricing, and promotions, reducing operational complexities. Delivery personnel benefit from optimized route planning and real-time updates, ensuring timely and accurate deliveries. Built using scalable and secure technologies, the platform ensures high performance and seamless integration with modern business operations. By automating key processes, the grocery delivery system enhances customer satisfaction, improves supply chain efficiency, and provides a reliable solution for retailers and consumers alike.

2.2. MYSQL

MySQL is the backbone of the grocery delivery system, efficiently managing structured data like users, products, orders, payments, and inventory. It enables fast queries, indexing, and secure transactions, ensuring data integrity and consistency. With ACID compliance and scalability, MySQL supports high-traffic operations and seamless data processing. Integrated with Spring Boot and Spring Data JPA, it optimizes CRUD operations and enhances system performance, making the grocery delivery system reliable, secure, and scalable..

2.3.SWAGGER UI

Swagger UI improves the grocery delivery system by providing an interactive interface for API documentation, making it easier for developers to test and understand RESTful APIs. It allows users to visualize API endpoints, send test requests, and inspect responses without needing additional tools. By integrating with Spring Boot, Swagger UI automatically generates documentation, ensuring consistency and clarity in API design. It simplifies backend communication, speeds up development, and enhances collaboration among teams. With features like API versioning support, request validation, and response visualization, Swagger UI ensures a well-documented and efficient API management process.

CHAPTER 3

PROPOSED SYSTEM

3.1. PROPOSED WORK

The **Budget Management System** proposes a centralized platform that integrates all aspects of financial management, from tracking income and expenses to monitoring investments and budgeting goals. Built using **Spring Boot 3**, the system provides a seamless experience for individuals, business owners, and financial analysts. Users can access a user-friendly interface that allows them to categorize transactions, set financial goals, generate reports, and analyze spending patterns in real time. This ensures financial clarity and helps users make informed decisions.

For financial managers and businesses, the system includes a robust admin panel that simplifies budget allocation, transaction tracking, investment portfolio management, and financial report generation. It enables efficient financial planning by providing insights into income and expenditures, helping users maintain financial stability and optimize investment strategies. The system also automates key calculations, reducing manual workload and minimizing errors in financial data handling.

Financial advisors and analysts benefit from a dedicated module designed to optimize financial oversight. They can review financial trends, provide recommendations, and update financial records in real time, ensuring accurate and strategic financial management. Automated alerts and real-time data updates help users stay informed about their financial health, reducing risks and improving long-term planning.

The system incorporates **secure authentication mechanisms**, allowing users to access their financial data safely with role-based access control and encryption. Real-time notifications keep users updated on budget status, upcoming expenses, and investment performance, enhancing their overall financial awareness. With its **scalable and efficient design**, the **Budget Management System** not only simplifies personal and business financial management but also improves financial literacy and decision-making, making it a reliable and modern solution for effective budget planning.

3.2. ADVANTAGES

The **Budget Management System** offers several advantages, ensuring an efficient and streamlined financial management experience for individuals, businesses, and financial analysts. Its **scalable and efficient architecture** allows smooth performance even as the volume of transactions and users increases. Built using **Spring Boot 3**, the system provides a **robust and secure backend**, enabling fast development while maintaining high security standards. With **RESTful API integration**, the platform ensures seamless communication between the frontend and backend, allowing easy expansion for future enhancements such as mobile applications or third-party financial service integrations.

For users, the system enhances financial planning and engagement by offering a **user-friendly interface, real-time financial tracking, and personalized budget recommendations**. Secure authentication mechanisms and automated notifications improve the overall financial management experience, ensuring transparency at every step of the process. Financial managers and business owners benefit from **efficient budget tracking, automated financial analysis, and insightful reporting tools**, helping them optimize financial decision-making and reduce manual workload.

Financial analysts and advisors gain access to **real-time data analytics and financial trend monitoring**, allowing for accurate forecasting and informed decision-making. The system also provides **role-based access control**, ensuring that only authorized users can manage financial records, investments, and transactions, enhancing security and preventing unauthorized access. Security is a major advantage, with **built-in data encryption, input validation, and protection against SQL injection and cross-site scripting attacks**. The system also ensures **high availability and reliability**, minimizing downtime and disruptions. Additionally, **analytics and reporting tools** provide valuable insights into financial trends, spending habits, and investment performance, helping users make data-driven decisions for financial stability and growth. By integrating **automation, security, and scalability**, the **Budget Management System** offers a reliable, efficient, and user-friendly solution for modern financial management.

CHAPTER 4

METHODOLOGY

1. User Registration and Authentication

Users can register with their name, email, and password, ensuring secure login with encrypted credentials. OAuth 2.0 integration allows seamless access through Google and other third-party authentication providers. Role-based access control ensures that only authorized users can manage specific functions, enhancing security.

2. Product Browsing and Smart Search

Customers can explore a wide range of grocery items categorized for easy access. A full-text search and filtering options help users quickly find specific products. Recommendations based on shopping history enhance the browsing experience.

3. Cart Management and Order Processing

Users can add, remove, or modify items in their cart before checkout. A seamless order placement process ensures secure payment through integrated gateways like Razorpay and Google Pay. Admins manage order processing and dispatch, minimizing delays.

4. Real-Time Order Tracking and Notifications

Customers can track their orders from placement to delivery using live tracking features. Automated notifications provide updates on order status, estimated delivery time, and successful transactions, improving customer engagement.

5. Inventory and Store Management

Store owners can efficiently manage inventory, update stock levels, and adjust pricing in real time. The system prevents overselling by syncing stock availability with customer orders. Admins can also categorize products and run promotions.

6. **Delivery Personnel Management**

A dedicated module allows delivery personnel to accept and update order statuses. Optimized route mapping ensures efficient deliveries. Live tracking helps customers and store admins monitor delivery progress.

7. **Secure Payment Gateway Integration**

The system supports multiple payment options, including credit cards, UPI, and wallets. Transactions are secured with encryption, fraud detection mechanisms, and multi-factor authentication, ensuring a safe shopping experience.

8. **Customer Feedback and Reviews**

Users can rate products and delivery services, providing valuable insights for businesses. Reviews help improve service quality, while store admins can address customer concerns through a feedback management system.

9. **Security and Data Protection**

The system ensures data privacy through HTTPS encryption, input validation, and protection against SQL injection and cross-site scripting attacks. Role-based access control limits sensitive operations to authorized users.

10. **Analytics and Reporting for Business Growth**

Admins can access detailed reports on order trends, customer preferences, and sales performance. These insights help businesses refine their strategies, optimize stock management, and enhance customer satisfaction.

4.1. Flow Diagram:

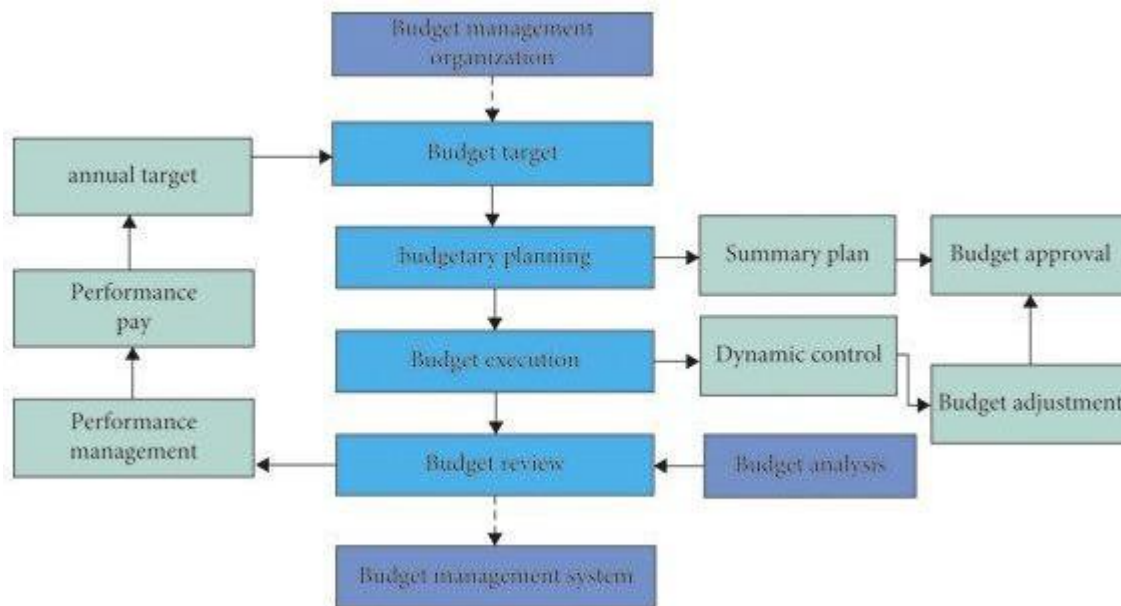


FIGURE NO:4.1.FLOW DIAGRAM

4.2. CLASS DIAGRAM:

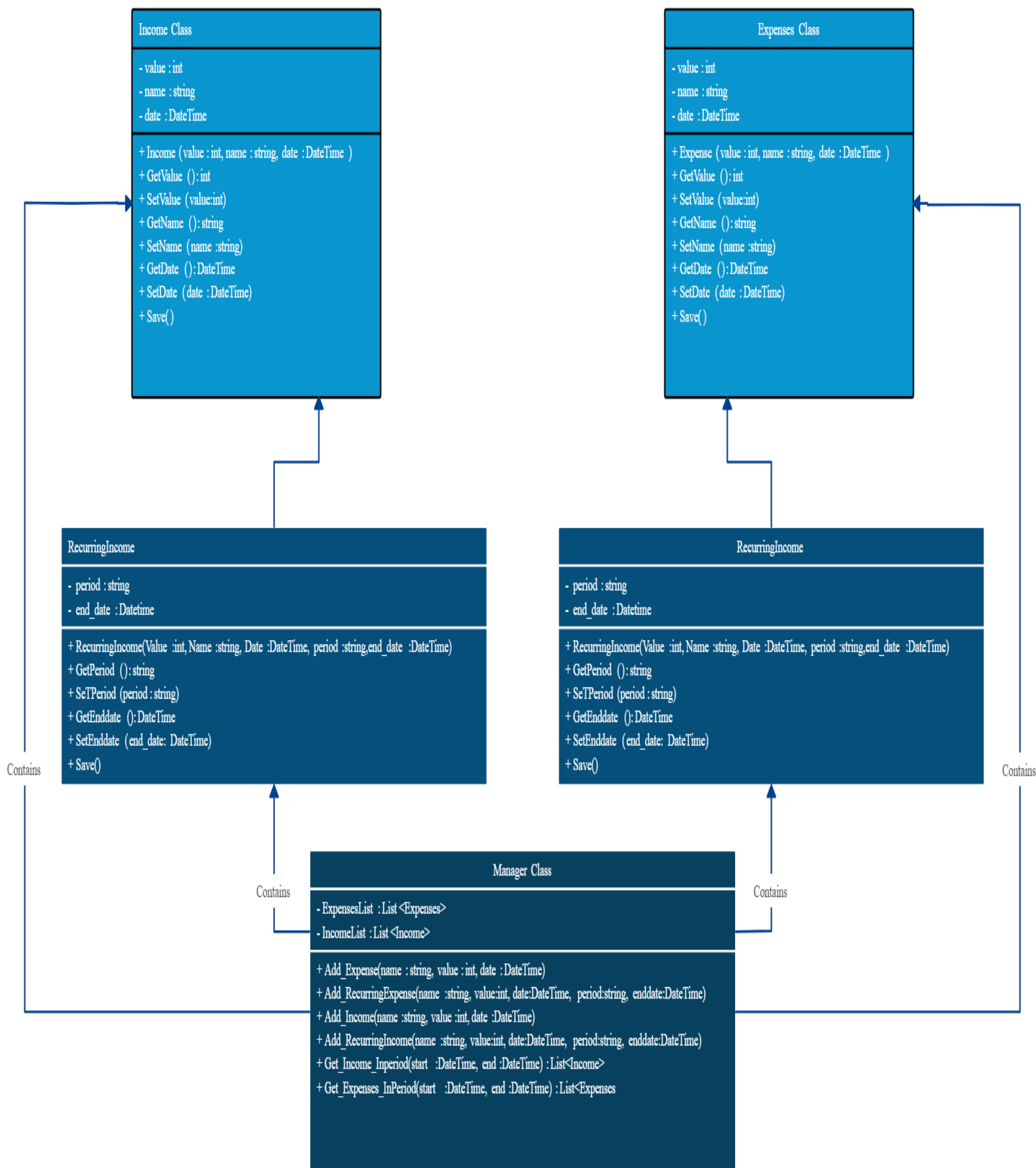


FIGURE NO:4.2.CLASS DIAGRAM

4.3. Sequence Diagram:

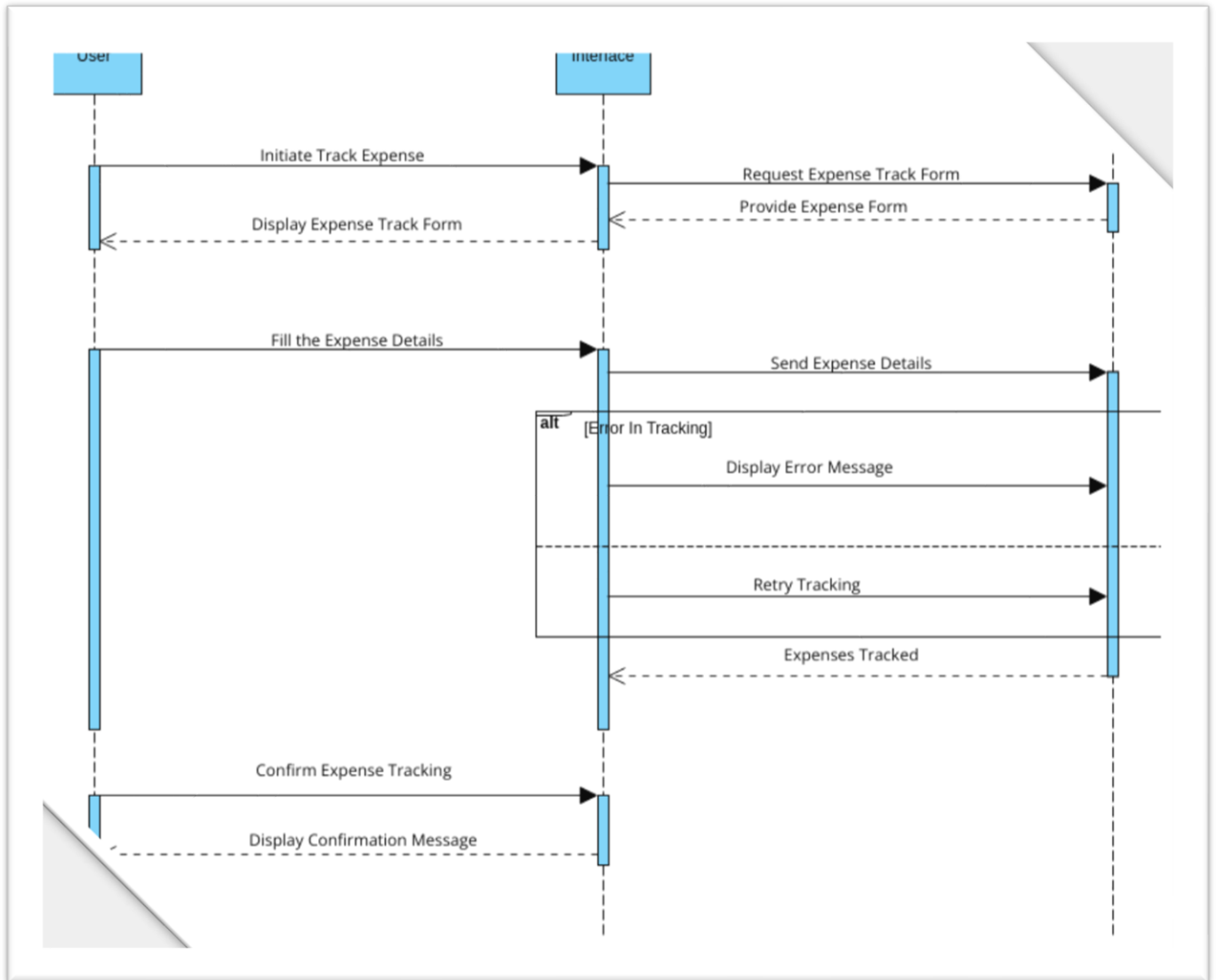


FIGURE NO:4.3.SEQUENCE DIAGRAM

4.4. ER Diagram:

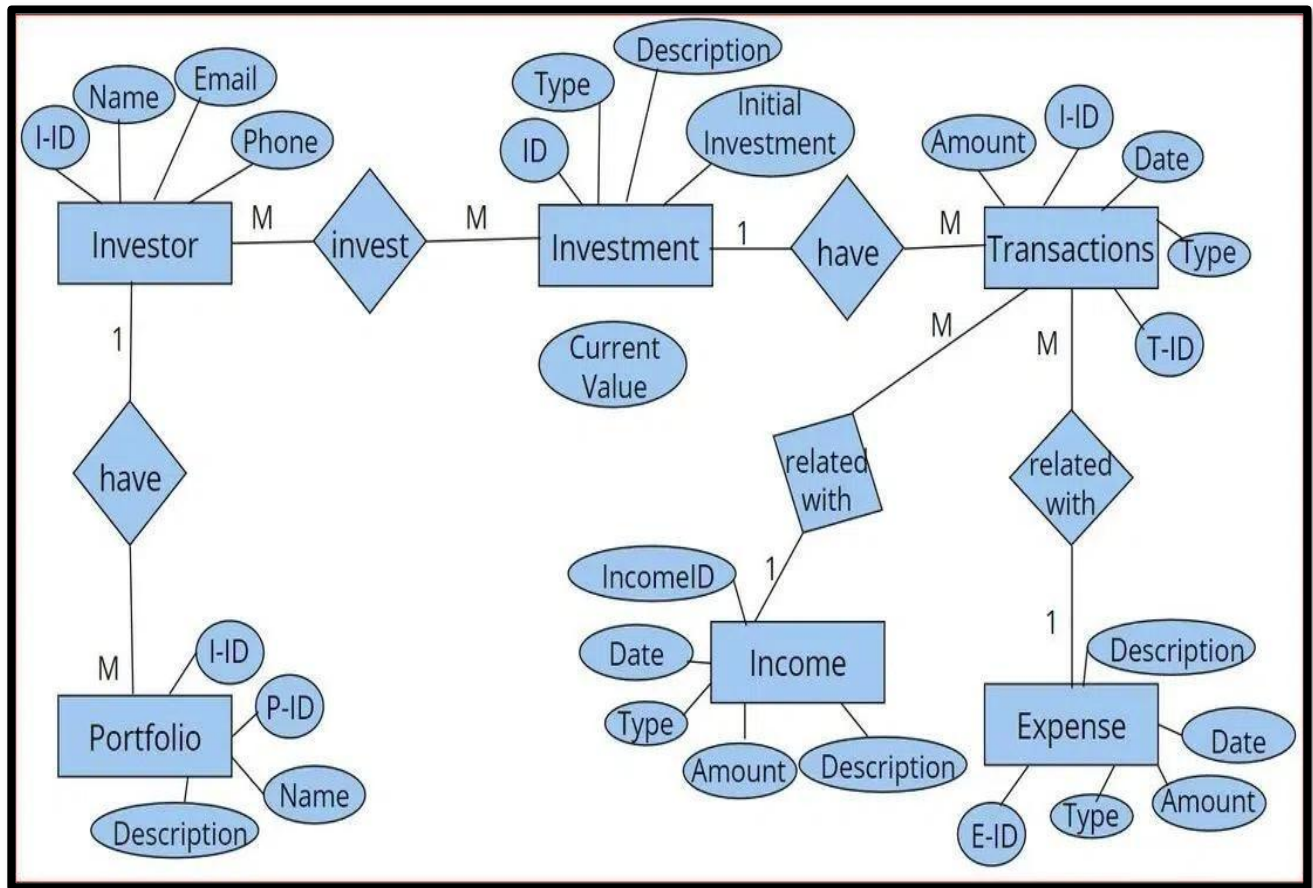


FIGURE NO:4.4.ER DIAGRAM

CHAPTER 5

IMPLEMENTATION AND RESULT

The practical execution of the grocery delivery system involves a structured approach to development, testing, and analysis to ensure seamless functionality and efficiency. The system was built using Spring Boot for the backend, React.js for the frontend, and MySQL for database management, ensuring scalability and high performance.

5.1. CODING:

6. Code Snippets & Explanations

emailController:

```
package com.example.budget_management.Controller;
```

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

```
import org.springframework.web.bind.annotation.CrossOrigin;
```

```
import org.springframework.web.bind.annotation.PostMapping;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RequestParam;
```

```
import org.springframework.web.bind.annotation.RestController;
```

```
import com.example.budget_management.Service.emailService;
```

```
@RestController
```

```
@RequestMapping("/email")
```

```
@CrossOrigin(origins = "*")
```

```
public class emailController {
```

```
@Autowired
```

```
private emailService emlService;
```

```
@PostMapping("/send")
```

```
public String sendEmail(@RequestParam String to,
```

```
@RequestParam String subject,
```

```
@RequestParam String body) {
```

```
    return emlService.sendEmail(to, subject, body);
```

```
}
```

```
}
```

emailEntity:

```
package com.example.budget_management.Entity;
```

```
import jakarta.persistence.*;
```

```
import lombok.*;
```

```
@Entity
```

```
@Table(name = "emails")
```

```
@Getter
```

```
@Setter
```

```
@NoArgsConstructor
```

```
@AllArgsConstructor
```

```
public class emailEntity {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    private String recipient;
```

```
    private String subject;
```

```
    private String body;
```

```
    public emailEntity() {
```

```
        super();
```

```
        // TODO Auto-generated constructor stub
```

```
    }
```

```
    public emailEntity(Long id, String recipient, String subject, String body) {
```

```
        super();
```

```
        this.id = id;
```

```
        this.recipient = recipient;
```

```
        this.subject = subject;
```

```
        this.body = body;
```

```
}  
  
public Long getId() {  
    return id;  
}  
  
public void setId(Long id) {  
    this.id = id;  
}  
  
public String getRecipient() {  
    return recipient;  
}  
  
public void setRecipient(String recipient) {  
    this.recipient = recipient;  
}  
  
public String getSubject() {  
    return subject;  
}  
  
public void setSubject(String subject) {  
    this.subject = subject;  
}  
  
public String getBody() {  
    return body;  
}  
  
public void setBody(String body) {  
    this.body = body;  
}
```

```
}
```

```
EmailRepository:
package com.example.budget_management.Repository;
```

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

```
import org.springframework.stereotype.Repository;
```

```
import com.example.budget_management.Entity.emailEntity;
```

@Repository

```
public interface emailRepository extends JpaRepository<emailEntity, Long> {
}
```

```
emailService:
package com.example.budget_management.Service;
```

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

```
import org.springframework.mail.SimpleMailMessage;
```

```
import org.springframework.mail.javamail.JavaMailSender;
```

```
import org.springframework.stereotype.Service;
```

```
import com.example.budget_management.Entity.emailEntity;
```

```
import com.example.budget_management.Repository.emailRepository;
```

@Service

```
public class emailService {
```

@Autowired

```
private JavaMailSender mailSender;
```

@Autowired

```
private emailRepository emRepository;
```

```
public String sendEmail(String recipient, String subject, String body) {  
    try {  
        // Send email  
  
        SimpleMailMessage message = new SimpleMailMessage();  
  
        message.setTo(recipient);  
  
        message.setSubject(subject);  
  
        message.setText(body);  
  
        message.setFrom("your-email@gmail.com");  
  
  
        mailSender.send(message);  
  
  
        // Save email details to the database  
  
        emailEntity email = new emailEntity();  
  
        email.setRecipient(recipient);  
  
        email.setSubject(subject);  
  
        email.setBody(body);  
  
        emRepository.save(email);  
  
  
        return "Email Sent Successfully!";  
    } catch (Exception e) {  
        return "Error Sending Email: " + e.getMessage();  
    }  
}
```


Output:

The image shows a Swagger UI interface in a web browser. The URL bar indicates the endpoint is `localhost:8080/swagger-ui/index.html#/email-controller/sendEmail`. The interface displays a **POST** request to `/email/send`. The parameters section shows the following values:

- to**: `727823tuoio037@skct.edu.in`
- subject**: `HI`
- body**: `hello bhai 10 kilo kari`

The **Responses** section shows a **200** status code with the message `Email sent successfully!`. The **Request Headers** section shows the following headers:

```
Content-Type: application/json
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4398.93 Safari/537.36
```

EMAIL :

The image shows a Gmail inbox interface. The email is from `727823tuoio037@skct.edu.in` with the subject `HI`. The body of the email is `hello bhai 10 kilo kari`. The email was received at `12:11 AM (0 minutes ago)`. The interface includes a left sidebar with navigation options like **Compose**, **Inbox** (3,025), **Starred**, **Snoozed**, **Sent**, **Drafts**, and **More**. The top bar shows the search bar and the status `Active`.

AdminEntity :

```

package com.example.Api_project.Entity;

import java.io.Serializable;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

import jakarta.persistence.Table;
import lombok.Data;

@Data
@Entity
@Table(name="Admin")

public class adminEntity implements Serializable{
    private static final long serialVersionUID=1L;
    @Id
    private int AdminID;
    private String adminName;
    private String AdminEmail;
    public String getAdminEmail() {
        return AdminEmail;
    }
    public void setAdminEmail(String adminEmail) {
        AdminEmail = adminEmail;
    }
    public String getAdminName() {
        return adminName;
    }
    public void setAdminName(String adminName) {
        this.adminName = adminName;
    }
}

```

AdminService:

```

package com.example.Api_project.Service;

import java.util.List;

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

import com.example.Api_project.Entity.adminEntity;
import com.example.Api_project.Repository.adminInterface;

@Service
public class adminService {
    @Autowired
    private adminInterface ar;
    public List<adminEntity> getAdminData(){
        return ar.findAll();
    }
    public adminEntity createAdmin(adminEntity ae) {
        return ar.save(ae);
    }
    public adminEntity updateAdmin(int AdminID, adminEntity ae) {
        return ar.save(ae);
    }
    public void deleteAdmin(int AdminID) {
        if(ar.existsById(AdminID)) {
            ar.deleteById(AdminID);
        }
        else {
            throw new RuntimeException("Requested VoterID not found");
        }
    }
    public List<adminEntity> getStartsWithQuery(String name) {
        return ar.findByAdminNameStartsWith(name);
    }

    public List<adminEntity> getEndsWithQuery(String name) {
        return ar.findByAdminNameEndsWith(name);
    }
    public List<adminEntity> getContainingQuery(String name) {

        return ar.findByAdminNameContaining(name);
    }
    public List<adminEntity> getIsContainingQuery(String name) {
        return ar.findByAdminNameIsContaining(name);
    }
    public List<adminEntity> getContainsQuery(String name) {
        return ar.findByAdminNameContains(name);
    }
}

```

```
public List<adminEntity> getNotContainingQuery(String name) {  
    return ar.findByAdminNameNotContaining(name);  
}  
public List<adminEntity> getNotLikeQuery(String name) {  
    return ar.findByAdminNameNotLike(name);  
}  
public List<adminEntity> getSelectQueries() {  
    return ar.selectquery();  
}  
public int getUpdateQueries(int id, String name) {  
    return ar.updatequery(id, name);  
}  
  
public int getDeleteQueries(int id) {  
    return ar.deletequery(id);  
}  
  
}
```

Admin Controller:

```

package com.example.Api_project.Controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.example.Api_project.Entity.adminEntity;
import com.example.Api_project.Service.adminService;

@RestController
public class adminController {
    @Autowired
    private adminService as;

    @GetMapping("/adminDetails")
    public List<adminEntity> getAdmin(){
        return as.getAdminData();
    }

    @PostMapping("/createAdmin")
    public adminEntity createAdmin(@RequestBody adminEntity ae) {
        return as.createAdmin(ae);
    }

    @PutMapping("/updateAdmin/{AdminID}")
    public adminEntity updateAdmin(@PathVariable int AdminID, @RequestBody
adminEntity ae) {
        return as.updateAdmin(AdminID, ae);
    }

    @DeleteMapping("/deleteAdmin/{AdminID}")
    public void deleteAdmin(@PathVariable int AdminID) {
        as.deleteAdmin(AdminID);
    }

    @GetMapping("/adminselect")
    public List<adminEntity> getSelectQuery(){
        return as.getSelectQueries();
    }

    @GetMapping("/adminupdate")
    public int getUpdateQuery(@RequestParam int id, @RequestParam String

```

```

name){
    return as.getUpdateQueries(id,name);
}

@GetMapping("/admindelete")
public int getDeleteQuery(@RequestParam int id){
    return as.getDeleteQueries(id);
}

@GetMapping("/adminStartsWith")
public List<adminEntity> retrieveStartsWithQuery(@RequestParam String name){
    return as.getStartsWithQuery(name);
}

@GetMapping("/adminEndsWith")
public List<adminEntity> retrieveEndsWithQuery(@RequestParam String name){
    return as.getEndsWithQuery(name);
}

@GetMapping("/adminContaining")
public List<adminEntity> retrieveContainingQuery(@RequestParam String
name){
    return as.getContainingQuery(name);
}

@GetMapping("/adminIsContaining")
public List<adminEntity> retrieveIsContainingQuery(String name){
    return as.getIsContainingQuery(name);
}

@GetMapping("/adminContains")
public List<adminEntity> retrieveContainsQuery(String name){
    return as.getContainsQuery(name);
}

@GetMapping("/adminNotContaining")
public List<adminEntity> retrieveNotContainingQuery(String name){
    return as.getNotContainingQuery(name);
}

@GetMapping("/adminNotLike")
public List<adminEntity> retrieveNotLikeQuery(String name){
    return as.getNotLikeQuery(name);
}

}

```

Contains key backend and frontend code snippets for implementing various system functionalities.

7. Output

The screenshot displays the Eclipse IDE interface. The top editor shows the `AdminController.java` file with the following code:

```

1 package com.example.Api_project.Controller;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.DeleteMapping;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.PathVariable;
9 import org.springframework.web.bind.annotation.PostMapping;
10 import org.springframework.web.bind.annotation.PutMapping;
11 import org.springframework.web.bind.annotation.RequestBody;
12 import org.springframework.web.bind.annotation.RequestMapping;
13 import org.springframework.web.bind.annotation.RestController;
14
15 import com.example.Api_project.Entity.adminEntity;
16 import com.example.Api_project.Service.adminService;
17
18
19
20 @RestController
21 public class AdminController {
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

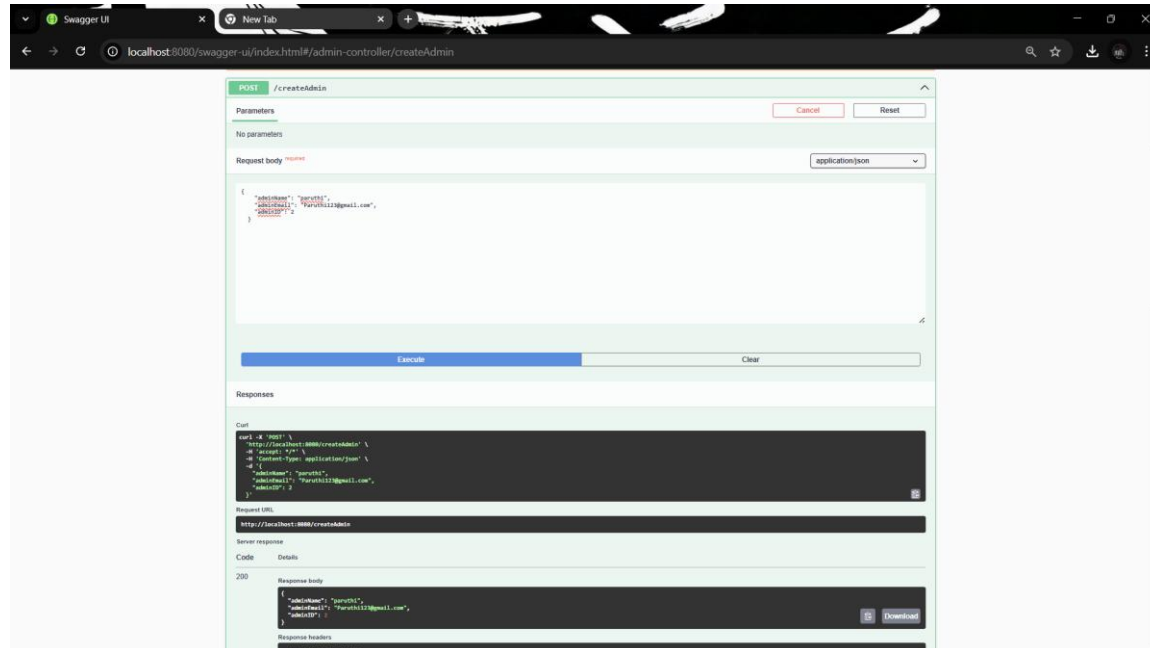
The bottom console shows the output of the application startup:

```

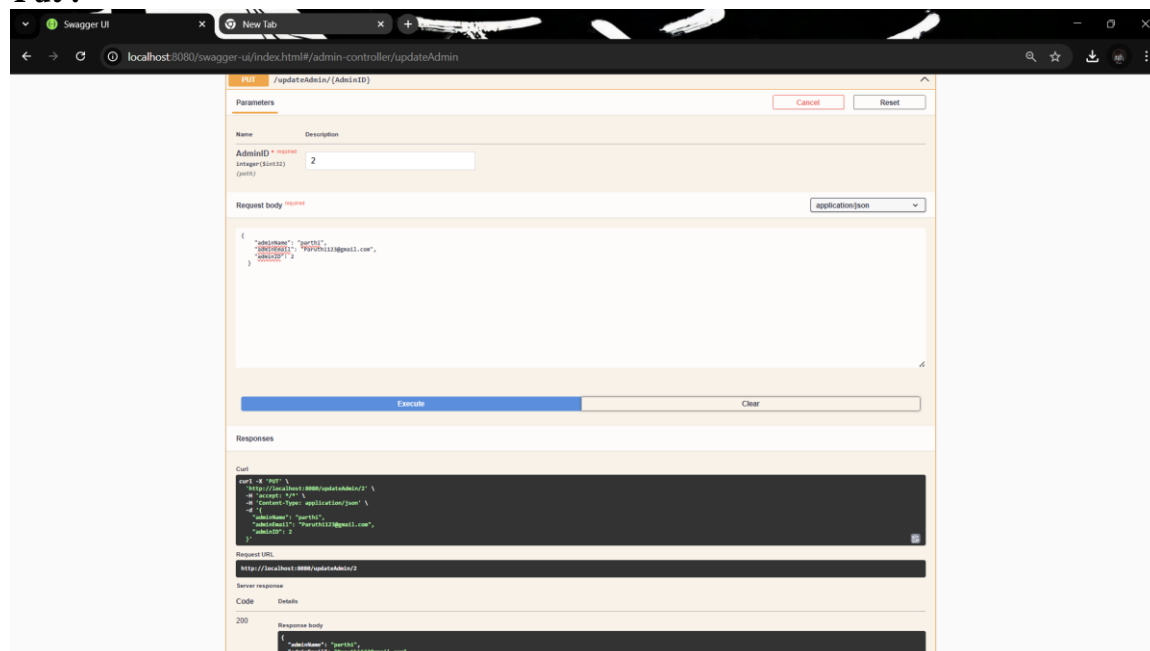
2025-03-19T10:33:55.458+05:30 INFO 9092 --- [Api_project] [main] c.e.Api_project.ApiProjectApplication : Starting A
2025-03-19T10:33:55.462+05:30 INFO 9092 --- [Api_project] [main] c.e.Api_project.ApiProjectApplication : No active
2025-03-19T10:33:56.684+05:30 INFO 9092 --- [Api_project] [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapp
2025-03-19T10:33:56.775+05:30 INFO 9092 --- [Api_project] [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished S
2025-03-19T10:33:57.213+05:30 WARN 9092 --- [Api_project] [main] trationDelegatedBeanPostProcessorChecker : Bean 'org
2025-03-19T10:33:57.312+05:30 INFO 9092 --- [Api_project] [main] .u.s.a.s.AnnotationActionEndpointMapping : Supporting
2025-03-19T10:33:57.696+05:30 INFO 9092 --- [Api_project] [main] o.s.b.w.embedded.tomcat.Tomcat10WebServer : Tomcat ini
2025-03-19T10:33:57.716+05:30 INFO 9092 --- [Api_project] [main] o.apache.catalina.core.StandardService : Starting s
2025-03-19T10:33:57.717+05:30 INFO 9092 --- [Api_project] [main] o.apache.catalina.core.StandardEngine : Starting S
2025-03-19T10:33:57.823+05:30 INFO 9092 --- [Api_project] [main] o.a.c.c.C.[tomcat].[localhost].[/] : Initializ

```


Post :



Put :



The image shows the Swagger UI interface for a REST API. The browser address bar shows the URL: `localhost:3080/swagger-ui/index.html#/admin-controller/updateAdmin`. The interface is for a **PUT** method on the `/updateAdmin/{AdminID}` endpoint.

Parameters:

Name	Description
AdminID	required integer (int32) (path)

The value `2` is entered in the input field for AdminID.

Request body: required
Type: `application/json`

```
{
  "username": "user123",
  "email": "user123@gmail.com",
  "password": "123456"
}
```

Responses:

curl

```
curl -X PUT \
  http://localhost:3080/updateAdmin/2 \
  -H 'Accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "username": "user123",
    "email": "user123@gmail.com",
    "password": "123456"
  }'
```

Request URL: `http://localhost:3080/updateAdmin/2`

Server response:

Code	Details
200	Response body <pre>{ "username": "user123", "email": "user123@gmail.com", "password": "123456" }</pre>

Startswith:

The screenshot displays the Swagger UI for a REST API. The endpoint being viewed is `GET /admin/startswith`. A query parameter `name` of type `string` is defined with the value `prem`. The `Execute` button has been clicked, showing the following details:

Request:

```
curl -X GET \
  http://localhost:8080/admin/startswith?name=prem \
  -H 'accept: */*'
http://localhost:8080/admin/startswith?name=prem
```

Response (200):

Response body:

```
{
  "id": 1,
  "name": "prem",
  "email": "prem@gmail.com",
  "password": "123456"
}
```

Response headers:

```
connection: keep-alive
content-type: application/json
date: Wed, 19 Apr 2023 05:15:17 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses table:

Code	Description	Links
200	OK	No link

The interface also shows a dropdown for the media type (set to `JSON`) and a section for the response schema.

Endswith:

The screenshot displays the Swagger UI interface for a REST API. The endpoint being viewed is `/admin-controller/retrieveEndsWithQuery`. The 'name' parameter is set to `http://localhost:3000/admin-controller/retrieveEndsWithQuery`. The response body is a JSON object:

```
{
  "name": "http://localhost:3000/admin-controller/retrieveEndsWithQuery",
  "email": "http://localhost:3000/admin-controller/retrieveEndsWithQuery",
  "password": "http://localhost:3000/admin-controller/retrieveEndsWithQuery"
}
```

The response headers are also visible:

```
Content-Type: application/json
Date: Mon, 10 Sep 2018 07:28:42 GMT
Keep-Alive: timeout=60
Transfer-Encoding: chunked
```

The response status is 200 OK. The response body is a JSON object with the following fields:

```
{
  "name": "http://localhost:3000/admin-controller/retrieveEndsWithQuery",
  "email": "http://localhost:3000/admin-controller/retrieveEndsWithQuery",
  "password": "http://localhost:3000/admin-controller/retrieveEndsWithQuery"
}
```

User Entity:

```

package com.example.Api_project.Entity;

import java.io.Serializable;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.Data;

@Data
@Entity
@Table(name="User")
public class userEntity implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    private int userID;
    private String userName;
    private String userEmail;

    public int getUserID() {
        return userID;
    }
    public void setUserID(int userID) {
        this.userID = userID;
    }
    public String getUserEmail() {
        return userEmail;
    }
    public void setUserEmail(String userEmail) {
        this.userEmail = userEmail;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
}

```

UserController:

```

package com.example.Api_project.Controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.example.Api_project.Entity.userEntity;
import com.example.Api_project.Service.userService;

@RestController
public class userController {
    @Autowired
    private userService us;

    @GetMapping("/userDetails")
    public List<userEntity> getUser() {
        return us.getUserData();
    }

    @PostMapping("/createUser")
    public userEntity createUser(@RequestBody userEntity ue) {
        return us.createUser(ue);
    }

    @PutMapping("/updateUser/{userID}")
    public userEntity updateUser(@PathVariable int userID, @RequestBody userEntity ue) {
        return us.updateUser(userID, ue);
    }

    @DeleteMapping("/deleteUser/{userID}")
    public void deleteUser(@PathVariable int userID) {
        us.deleteUser(userID);
    }

    @GetMapping("/userStartsWith")
    public List<userEntity> retrieveStartsWithQuery(@RequestParam String name) {
        return us.getStartsWithQuery(name);
    }

    @GetMapping("/userEndsWith")
    public List<userEntity> retrieveEndsWithQuery(@RequestParam String name) {
        return us.getEndsWithQuery(name);
    }
}

```

```
@GetMapping("/userContaining")
public List<userEntity> retrieveContainingQuery(@RequestParam String name) {
    return us.getContainingQuery(name);
}

@GetMapping("/userIsContaining")
public List<userEntity> retrieveIsContainingQuery(@RequestParam String name) {
    return us.getIsContainingQuery(name);
}

@GetMapping("/userContains")
public List<userEntity> retrieveContainsQuery(@RequestParam String name) {
    return us.getContainsQuery(name);
}

@GetMapping("/userNotContaining")
public List<userEntity> retrieveNotContainingQuery(@RequestParam String name) {
    return us.getNotContainingQuery(name);
}

@GetMapping("/userNotLike")
public List<userEntity> retrieveNotLikeQuery(@RequestParam String name) {
    return us.getNotLikeQuery(name);
}
}
```


UserService:

```

package com.example.Api_project.Service;

import java.util.List;

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

import com.example.Api_project.Entity.userEntity;
import com.example.Api_project.Repository.userInterface;

@Service
public class userService {
    @Autowired
    private userInterface us;

    public List<userEntity> getUserData() {
        return us.findAll();
    }

    public userEntity createUser(userEntity ue) {
        return us.save(ue);
    }

    public userEntity updateUser(int userID, userEntity ue) {
        return us.save(ue);
    }

    public void deleteUser(int userID) {
        if (us.existsById(userID)) {
            us.deleteById(userID);
        } else {
            throw new RuntimeException("Requested UserID not found");
        }
    }

    public List<userEntity> getStartsWithQuery(String name) {
        return us.findByUserNameStartsWith(name);
    }

    public List<userEntity> getEndsWithQuery(String name) {
        return us.findByUserNameEndsWith(name);
    }

    public List<userEntity> getContainingQuery(String name) {
        return us.findByUserNameContaining(name);
    }

    public List<userEntity> getIsContainingQuery(String name) {
        return us.findByUserNameIsContaining(name);
    }
}

```

```
public List<userEntity> getContainsQuery(String name) {  
    return us.findByUserNameContains(name);  
}  
  
public List<userEntity> getNotContainingQuery(String name) {  
    return us.findByUserNameNotContaining(name);  
}  
  
public List<userEntity> getNotLikeQuery(String name) {  
    return us.findByUserNameNotLike(name);  
}  
}
```

UserInterface:

```
package com.example.Api_project.Repository;

import java.util.List;

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

import com.example.Api_project.Entity.userEntity;
```

@Repository

```
public interface userInterface extends JpaRepository<userEntity, Integer> {

    public List<userEntity> findByUserNameStartsWith(String name);
    public List<userEntity> findByUserNameEndsWith(String name);
    public List<userEntity> findByUserNameContaining(String name);
    public List<userEntity> findByUserNameIsContaining(String name);
    public List<userEntity> findByUserNameContains(String name);
    public List<userEntity> findByUserNameNotContaining(String name);
    public List<userEntity> findByUserNameNotLike(String name);

}
```

UpdateUser:

The image displays two screenshots of the Swagger UI interface, showing the configuration and execution of the `UpdateUser` endpoint.

Top Screenshot: Endpoint Configuration

- Endpoint:** `PUT /updateUser/{userId}`
- Parameters:**
 - Name:** `userId`
 - Description:** `Integer (int32)`
 - Value:** `1`
- Request body:** `application/json`
- Request body content:**

```
{  "userId": 1,  "username": "jane",  "email": "jane@gmail.com"}
```
- Buttons:** `Execute` and `Clear`

Bottom Screenshot: Execution Results

- Request URL:** `http://localhost:8080/swagger-ui/index.html#/user-controller/updateUser`
- Request body:** `application/json`
- Request body content:**

```
{  "userId": 1,  "username": "jane",  "email": "jane@gmail.com"}
```
- Buttons:** `Execute` and `Clear`
- Responses:**
 - Code:** `200`
 - Response body:**

```
{  "userId": 1,  "username": "jane",  "email": "jane@gmail.com"}
```
 - Response headers:**

```
connection: keep-alivecontent-type: application/jsondate: Wed, 09 Apr 2025 15:42:45 GMTkeep-alive: timeout=60transfer-encoding: chunked
```

PostUser:

The image shows the Swagger UI for a REST API endpoint named `POST /createUser`. The interface is divided into several sections:

- Parameters:** A section indicating "No parameters".
- Request body:** A section with a dropdown menu set to `application/json`. The body contains a JSON object:

```
{  "username": "j",  "password": "P@ssw0rd",  "email": "j@domain.com"}
```
- Execute:** A blue button to execute the request, followed by a "Clear" button.
- Responses:** A section showing the response details for a `200` status code.
 - Code:** `200`
 - Response body:** A JSON object:

```
{  "username": "j",  "password": "P@ssw0rd",  "email": "j@domain.com"}
```
 - Request headers:** A section showing the headers for the request:

```
Content-Type: application/json
```

UserNotlike:

The image shows a Swagger UI interface for the `GET /userNotLike` endpoint. The browser address bar shows `localhost:8080/swagger-ui/index.html#/user-controller/retrieveNotLikeQuery`. The interface includes a "Parameters" section with a query parameter `name` of type `string` (required), with the value `Paluthi` entered. Below the parameters is an "Execute" button and a "Clear" button. The "Responses" section shows the "Server response" for a 200 status code. The response body is a JSON object: `{ "userIP": "1", "userEmail": "paluthe", "userPhone": "1234567890" }`. The response headers are: `Content-Type: application/json`, `Server: Apache/2.4.18 (Ubuntu)`, `Date: Mon, 10 Sep 2013 10:14:12 GMT`, `Keep-Alive: timeout=5`, and `Transfer-Encoding: chunked`. At the bottom, there is a "Responses" table with a 200 status code and an "OK" description. The table also shows a "Media type" dropdown set to `application/json` and a "Content type" dropdown set to `application/json`. The "Example Value" is `{ "userIP": "1", "userEmail": "paluthe", "userPhone": "1234567890" }`.

GET /userNotLike

Parameters

Name	Description
name	string (required)

Execute Clear

Responses

Server response

200

Response body

```
{
  "userIP": "1",
  "userEmail": "paluthe",
  "userPhone": "1234567890"
}
```

Response headers

```
Content-Type: application/json
Server: Apache/2.4.18 (Ubuntu)
Date: Mon, 10 Sep 2013 10:14:12 GMT
Keep-Alive: timeout=5
Transfer-Encoding: chunked
```

Responses

Code	Description	Links
200	OK	No links

Media type: application/json

Content type: application/json

Example Value: { "userIP": "1", "userEmail": "paluthe", "userPhone": "1234567890" }

IncomeEntity:

```

package com.example.Api_project.Entity;

import java.io.Serializable;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.Data;

@Data
@Entity
@Table(name="Income")
public class incomeEntity implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    private int incomeID;
    private String incomeAmount;
    private String incomeType;

    public int getIncomeID() {
        return incomeID;
    }
    public void setIncomeID(int incomeID) {
        this.incomeID = incomeID;
    }
    public String getIncomeAmount() {
        return incomeAmount;
    }
    public void setIncomeAmount(String incomeAmount) {
        this.incomeAmount = incomeAmount;
    }
    public String getIncomeType() {
        return incomeType;
    }
    public void setIncomeType(String incomeType) {
        this.incomeType = incomeType;
    }
    public static long getSerialVersionUID() {
        return serialVersionUID;
    }
}

```

IncomeInterface:

```
package com.example.Api_project.Repository;

import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;

import com.example.Api_project.Entity.incomeEntity;

public interface incomeInterface extends JpaRepository<incomeEntity, Integer> {
    public List<incomeEntity> findByIncomeTypeStartsWith(String incomeType);
    public List<incomeEntity> findByIncomeTypeEndsWith(String incomeType);
    public List<incomeEntity> findByIncomeTypeContaining(String incomeType);
    public List<incomeEntity> findByIncomeTypeIsContaining(String incomeType);
    public List<incomeEntity> findByIncomeTypeContains(String incomeType);
    public List<incomeEntity> findByIncomeTypeNotContaining(String incomeType);
    public List<incomeEntity> findByIncomeTypeNotLike(String incomeType);
}
```


IncomeController:

```
package com.example.Api_project.Controller;
```

```
import java.util.List;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
```

```
import com.example.Api_project.Entity.incomeEntity;
import com.example.Api_project.Service.incomeService;
```

```
@RestController
```

```
public class incomeController {
```

```
    @Autowired
```

```
    private incomeService incService;
```

```
    @GetMapping("/incomeDetails")
```

```
    public List<incomeEntity> getIncome() {
        return incService.getIncomeData();
    }
```

```
    @PostMapping("/createIncome")
```

```
    public incomeEntity createIncome(@RequestBody incomeEntity inc) {
        return incService.createInc(inc);
    }
```

```
    @PutMapping("/updateIncome/{incomeID}")
```

```
    public incomeEntity updateIncome(@PathVariable int incomeID, @RequestBody
incomeEntity inc) {
        return incService.updateInc(incomeID, inc);
    }
```

```
    @DeleteMapping("/deleteIncome/{incomeID}")
```

```
    public void deleteIncome(@PathVariable int incomeID) {
        incService.deleteInc(incomeID);
    }
```

```
    @GetMapping("/incomeStartsWith")
```

```
    public List<incomeEntity> getStartsWithQuery(@RequestParam String incomeType) {
        return incService.getStartsWithIncome(incomeType);
    }
```

```
    @GetMapping("/incomeEndsWith")
```

```

public List<incomeEntity> getEndsWithQuery(@RequestParam String incomeType) {
    return incService.retrieveEndsWithQuery(incomeType);
}

@GetMapping("/incomeContaining")
public List<incomeEntity> getContainingQuery(@RequestParam String incomeType) {
    return incService.retrieveContainingQuery(incomeType);
}

@GetMapping("/incomeIsContaining")
public List<incomeEntity> getIsContainingQuery(@RequestParam String incomeType) {
    return incService.retrieveIsContainingQuery(incomeType);
}

@GetMapping("/incomeContains")
public List<incomeEntity> getContainsQuery(@RequestParam String incomeType) {
    return incService.retrieveContainsQuery(incomeType);
}

@GetMapping("/incomeNotContaining")
public List<incomeEntity> getNotContainingQuery(@RequestParam String incomeType) {
    return incService.retrieveNotContainingQuery(incomeType);
}

@GetMapping("/incomeNotLike")
public List<incomeEntity> getNotLikeQuery(@RequestParam String incomeType) {
    return incService.retrieveNotLikeQuery(incomeType);
}
}

```

IncomeService:

```

package com.example.Api_project.Service;

import java.util.List;

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

import com.example.Api_project.Entity.incomeEntity;
import com.example.Api_project.Repository.incomeInterface;

@Service
public class incomeService {
    @Autowired
    private incomeInterface inter;

    public List<incomeEntity> getIncomeData() {
        return inter.findAll();
    }

    public incomeEntity createIncome(incomeEntity inc) {
        return inter.save(inc);
    }

    public incomeEntity updateIncome(int incomeID, incomeEntity inc) {
        return inter.save(inc);
    }

    public void deleteIncome(int incomeID) {
        if (inter.existsById(incomeID)) {
            inter.deleteById(incomeID);
        } else {
            throw new RuntimeException("income Not Found");
        }
    }

    public List<incomeEntity> getStartsWithIncome(String incomeType) {
        return inter.findByIncomeTypeStartsWith(incomeType);
    }

    public List<incomeEntity> retrieveEndsWithQuery(String incomeType) {
        return inter.findByIncomeTypeEndsWith(incomeType);
    }

    public List<incomeEntity> retrieveNotContainingQuery(String incomeType) {
        return inter.findByIncomeTypeNotContaining(incomeType);
    }

    public List<incomeEntity> retrieveContainsQuery(String incomeType) {
        return inter.findByIncomeTypeContaining(incomeType);
    }
}

```

```

    }

    public List<incomeEntity> retrieveIsContainingQuery(String incomeType) {
        return inter.findByIncomeTypeIsContaining(incomeType);
    }

    public List<incomeEntity> retrieveContainingQuery(String incomeType) {
        return inter.findByIncomeTypeContaining(incomeType);
    }

    public List<incomeEntity> retrieveNotLikeQuery(String incomeType) {
        return inter.findByIncomeTypeNotLike(incomeType);
    }

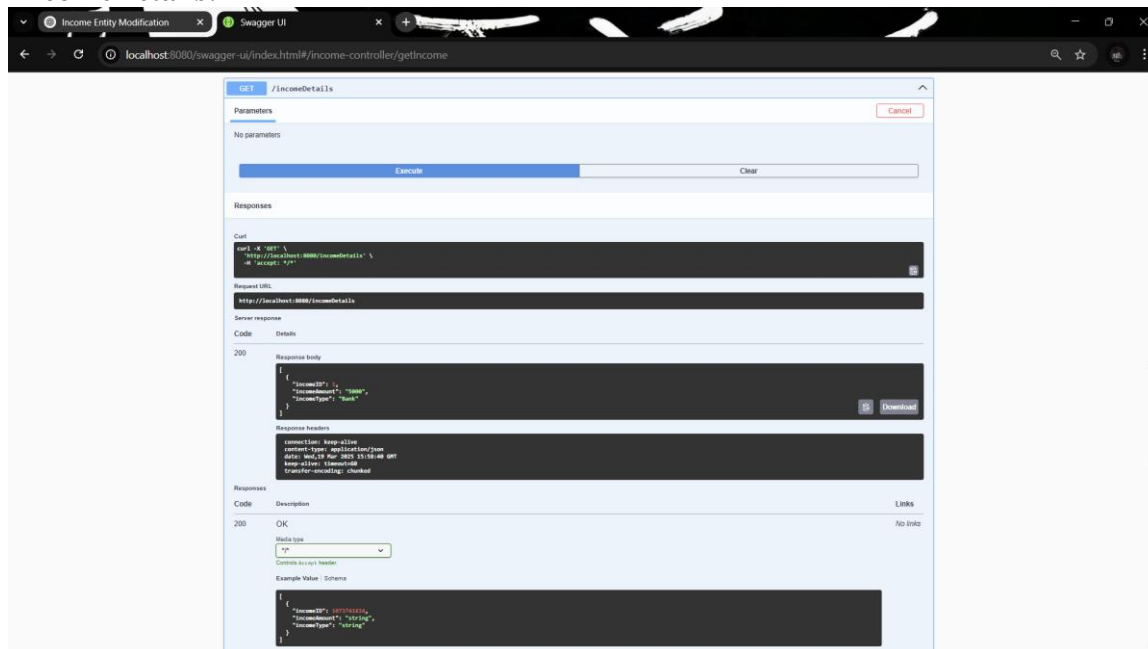
    public incomeEntity updateInc(int incomeID, incomeEntity inc) {
        // TODO Auto-generated method stub
        return null;
    }

    public incomeEntity createInc(incomeEntity inc) {
        // TODO Auto-generated method stub
        return null;
    }

    public void deleteInc(int incomeID) {
        // TODO Auto-generated method stub
    }
}

```

IncomeDetails:



The image shows a Swagger UI interface for the `IncomeDetails` endpoint. The browser address bar shows `localhost:3080/swagger-ui/index.html#/income-controller/getIncome`. The endpoint is a `GET` request with no parameters. The response is a `200` status with a JSON body. The response body is displayed in a dark-themed editor with a `Download` button. The response headers are also visible.

GET /IncomeDetails

Parameters

No parameters

[Cancel](#)

[Execute](#) [Clear](#)

Responses

Curl

```
curl -X GET \
  http://localhost:3080/IncomeDetails \
  -H 'accept: */*'

```

Request URL

`http://localhost:3080/IncomeDetails`

Server response

Code Details

200

Response body

```
{
  "incomeID": 1,
  "incomeAmount": "1000",
  "incomeType": "Salary"
}
```

[Download](#)

Response headers

```
connection: keep-alive
content-type: application/json
date: Wed, 19 Apr 2023 15:58:48 GMT
keep-alive: timeout=60
transfer-encoding: chunked

```

Responses

Code	Description	Links
200	OK	No links

Media type

JSON

Content Accept header

Example Value Schema

```
{
  "incomeID": 1,
  "incomeAmount": "1000",
  "incomeType": "Salary"
}
```

IncomeCreate:

The screenshot shows the Swagger UI interface for a REST API. The endpoint is a POST request to `/createIncome`. The request body is a JSON object with the following structure:

```
{  "incomeID": 1,  "incomeAmount": "7888",  "incomeType": "Indirect"}
```

The response is a 200 status code with the following headers:

```
Content-Type: keep-alive
Content-Length: 8
Date: Sat, 19 Apr 2025 10:06:10 GMT
Keep-Alive: timeout=60
```

The interface includes a 'Parameters' section (empty), a 'Request body' section (with a dropdown set to 'application/json'), an 'Execute' button, and a 'Responses' section showing the 200 response details.

ExpenseEntity:

```

package com.example.Api_project.Entity;

import java.io.Serializable;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

import jakarta.persistence.Table;
import lombok.Data;

@Data
@Entity
@Table(name="Expense")

public class expenseEntity implements Serializable{
    private static final long serialVersionUID=1L;
    @Id
    private int expenseID;
    private String expenseAmount;
    private String expenseType;

    public int getExpenseID() {
        return expenseID;
    }
    public void setExpenseID(int expenseID) {
        this.expenseID = expenseID;
    }
    public String getExpenseAmount() {
        return expenseAmount;
    }
    public void setExpenseAmount(String expenseAmount) {
        this.expenseAmount = expenseAmount;
    }
    public String getExpenseType() {
        return expenseType;
    }
    public void setExpenseType(String expenseType) {
        this.expenseType = expenseType;
    }
    public static long getSerialversionuid() {
        return serialVersionUID;
    }
}

```


ExpenseInterface:

```
package com.example.Api_project.Repository;

import java.util.List;

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

import com.example.Api_project.Entity.expenseEntity;

public interface expenseInterface extends JpaRepository <expenseEntity, Integer>{
    public List<expenseEntity> findByExpenseTypeStartsWith(String expenseType);
    public List<expenseEntity> findByExpenseTypeEndsWith(String expenseType);
    public List<expenseEntity> findByExpenseTypeContaining(String expenseType);
    public List<expenseEntity> findByExpenseTypeIsContaining(String expenseType);
    public List<expenseEntity> findByExpenseTypeContains(String expenseType);
    public List<expenseEntity> findByExpenseTypeNotContaining(String expenseType);
    public List<expenseEntity> findByExpenseTypeNotLike(String expenseType);
}
```

}

ExpenseController:

```

package com.example.Api_project.Controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.example.Api_project.Entity.expenseEntity;
import com.example.Api_project.Service.expenseService;

@RestController
public class expenseController {
    @Autowired
    private expenseService ex;

    @GetMapping("/expenseDetails")
    public List<expenseEntity> getexpense(){
        return ex.getExpenseData();
    }

    @PostMapping("/createexpense")
    public expenseEntity createExpense(@RequestBody expenseEntity exp) {
        return ex.createExpen(exp);
    }

    @PutMapping("/updateExpense/{expenseID}")
    public expenseEntity updateExpense(@PathVariable int expenseID, @RequestBody expenseEntity exp) {
        return ex.updateExpen(expenseID, exp);
    }

    @DeleteMapping("/deleteexpense/{expenseID}")
    public void deleteAExpense(@PathVariable int expenseID) {
        ex.deleteExpen(expenseID);
    }

    @GetMapping("/expenseStartsWith")
    public List<expenseEntity> getStartsWithQuery(@RequestParam String expenseType){
        return ex.getStartsWithExpense(expenseType);
    }
}

```

```

        @GetMapping("/expenseEndsWith")
        public List<expenseEntity> getEndsWithQuery(@RequestParam String
expenseType){
            return ex.retrieveEndsWithQuery(expenseType);
        }

        @GetMapping("/expenseContaining")
        public List<expenseEntity> getContainingQuery(@RequestParam String
expenseType){
            return ex.retrieveContainingQuery(expenseType);
        }

        @GetMapping("/expenseIsContaining")
        public List<expenseEntity> getIsContainingQuery(String expenseType){
            return ex.retrieveIsContainingQuery(expenseType);
        }

        @GetMapping("/expenseContains")
        public List<expenseEntity> getContainsQuery(String expenseType){
            return ex.retrieveContainsQuery(expenseType);
        }

        @GetMapping("/expenseNotContaining")
        public List<expenseEntity> getNotContainingQuery(String expenseType){
            return ex.retriveNotContainingQuery(expenseType);
        }

        @GetMapping("/expenseNotLike")
        public List<expenseEntity> getNotLikeQuery(String expenseType){
            return ex.retriveNotLikeQuery(expenseType);
        }
    }

```

ExpenseService:

```

package com.example.Api_project.Service;

import java.util.List;

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

import com.example.Api_project.Entity.expenseEntity;
import com.example.Api_project.Repository.expenseInterface;

@Service
public class expenseService {
    @Autowired
    private expenseInterface inter;

    public List<expenseEntity> getExpenseData() {
        // TODO Auto-generated method stub
        return inter.findAll();
    }

    public expenseEntity createExpen(expenseEntity exp) {
        // TODO Auto-generated method stub
        return inter.save(exp);
    }

    public expenseEntity updateExpen(int expenseID, expenseEntity exp) {
        // TODO Auto-generated method stub
        return inter.save(exp);
    }

    public void deleteExpen(int expenseID) {
        // TODO Auto-generated method stub
        if(inter.existsById(expenseID)) {
            inter.deleteById(expenseID);
        }
        else {
            throw new RuntimeException("Expense Not Found");
        }
    }

    public List<expenseEntity> getStartsWithExpense(String expenseType) {
        // TODO Auto-generated method stub
        return inter.findByExpenseTypeStartsWith(expenseType);
    }

    public List<expenseEntity> retrieveEndsWithQuery(String expenseType) {
        // TODO Auto-generated method stub
        return inter.findByExpenseTypeEndsWith(expenseType);
    }
}

```

```
public List<expenseEntity> retrieveNotContainingQuery(String expenseType) {  
    // TODO Auto-generated method stub  
    return inter.findByExpenseTypeNotContaining(expenseType);  
}  
  
public List<expenseEntity> retrieveContainsQuery(String expenseType) {  
    // TODO Auto-generated method stub  
    return inter.findByExpenseTypeContaining(expenseType);  
}  
  
public List<expenseEntity> retrieveIsContainingQuery(String expenseType) {  
    // TODO Auto-generated method stub  
    return inter.findByExpenseTypeIsContaining(expenseType);  
}  
  
public List<expenseEntity> retrieveContainingQuery(String expenseType) {  
    // TODO Auto-generated method stub  
    return inter.findByExpenseTypeContaining(expenseType);  
}  
  
public List<expenseEntity> retrieveNotLikeQuery(String expenseType) {  
    // TODO Auto-generated method stub  
    return inter.findByExpenseTypeNotLike(expenseType);  
}  
  
}
```

ExpenseDetails:

The image shows a Swagger UI interface for a REST API endpoint named `ExpenseDetails`. The browser address bar shows `localhost:3080/swagger-ui/index.html#/expense-controller/getexpense`. The interface includes a "Parameters" section with "No parameters" listed. Below this is an "Execute" button. The "Responses" section shows a 200 status code with a "Response body" containing a JSON object: `{ "expenseID": 1, "expenseAmount": "1000", "expenseType": "bank" }`. The "Response headers" section shows headers: `connection: keep-alive`, `content-type: application/json`, `date: Wed, 19 Apr 2023 15:07:05 GMT`, `keep-alive: timeout=60`, and `transfer-encoding: chunked`. The "Responses" table shows a 200 status code with a description "OK" and a link "No link". The "Example Value" section shows a JSON object: `{ "expenseID": 1, "expenseAmount": "1000", "expenseType": "bank" }`.

Swagger UI interface for the `ExpenseDetails` endpoint.

URL: `/expenseDetails`

Parameters: No parameters

Execute

Responses

200

Response body:

```
{
  "expenseID": 1,
  "expenseAmount": "1000",
  "expenseType": "bank"
}
```

Response headers:

```
connection: keep-alive
content-type: application/json
date: Wed, 19 Apr 2023 15:07:05 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses:

Code	Description	Links
200	OK	No link

Example Value:

```
{
  "expenseID": 1,
  "expenseAmount": "1000",
  "expenseType": "bank"
}
```

Expensecreate:

Income Entry Modification x Swagger UI x

localhost:8080/swagger-ui/index.html#/expense-controller/createExpense

POST /createexpense

Parameters

No parameters

Cancel Reset

Request body required

application/json

```
{
  "expenseID": 2,
  "expenseAmount": "1000",
  "expenseType": "ABC Bank"
}
```

Execute Clear

Responses

Curl

```
curl -X POST \
  http://localhost:8080/createExpense \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "expenseID": 2,
    "expenseAmount": "1000",
    "expenseType": "ABC Bank"
  }'
```

Request URL

http://localhost:8080/createExpense

Server response

Code Details

200

Response body

```
{
  "expenseID": 1,
  "expenseAmount": "1000",
  "expenseType": "ABC Bank"
}
```

Download

Response headers

connection: keep-alive

CHAPTER 6

CONCLUSION

6.1 CONCLUSION

The grocery delivery system provides a **secure, efficient, and scalable** solution for online grocery shopping, order management, and delivery tracking. It ensures seamless user experience through real-time order updates, secure payment gateways, and optimized delivery logistics. Store owners can efficiently manage inventory, process orders, and analyze customer feedback, while delivery personnel benefit from route optimization and real-time order updates. The system enhances operational efficiency by integrating advanced security measures such as **HTTPS encryption, role-based access control, and input validation** to protect user data. Future enhancements may include AI-driven product recommendations, expanded payment options, and enhanced analytics for business growth.

6.2 FUTURE SCOPE

1. **AI-Based Product Recommendations**

Implement an intelligent recommendation system that suggests products based on user preferences and purchase history. This enhances the shopping experience by offering personalized recommendations and increasing sales for store owners.

2. **Mobile Application for Enhanced Accessibility**

Develop a mobile app to provide users with a **faster and more convenient** way to browse products, place orders, and track deliveries. Push notifications will keep users updated on **order status, special discounts, and personalized offers**.

3. **Voice Search and Smart Assistance**

Enable voice-based search functionality, allowing users to add products to their cart and place orders using voice commands. Integration with smart assistants like **Google Assistant and Alexa** can further improve accessibility and convenience.

4. **Multi-Vendor Marketplace Support**

Expand the system to support **multiple grocery stores**, allowing users to order from

different vendors within a single platform. This feature increases choices for customers while helping local grocery businesses digitize their operations.

5. **Advanced Route Optimization for Faster Deliveries**

Implement AI-powered route optimization to help delivery personnel **navigate efficiently and reduce delivery times**. Traffic analysis and predictive route mapping will ensure timely and accurate deliveries.

6. **Automated Customer Support and Chatbots**

Introduce AI-driven chatbots for **instant query resolution, order status updates, and general assistance**. This feature enhances customer support by reducing response times and improving user satisfaction.

CHAPTER 7

REFERENCE

6.1 Conclusion

✓ **The Budget Management System provides a secure, efficient, and scalable solution for tracking income, expenses, investments, and overall financial planning. It ensures a seamless user experience through real-time financial insights, secure payment processing, and automated budgeting tools. Users can efficiently manage their personal or business finances, set financial goals, and monitor spending habits. The system integrates advanced security measures such as HTTPS encryption, role-based access control, and input validation to protect sensitive financial data.**

✓ **Future enhancements may include AI-powered financial recommendations, automated savings plans, and advanced data analytics for better financial decision-making.**

6.2 Future Scope

1. **AI-Based Financial Recommendations**
Implement an **intelligent financial advisory system** that suggests **budget optimizations, savings strategies, and investment opportunities** based on user income, expenses, and goals.
2. **Mobile Application for Accessibility**
Develop a **mobile app** to allow users to **track expenses, manage investments, and receive financial alerts on the go**. Push notifications will help users **stay informed about upcoming payments, investment performance, and budget status**.
3. **Voice-Based Transaction Management**
Enable **voice-controlled budgeting** to allow users to **record transactions, set financial reminders, and check their financial status using voice commands**. Integration with **smart assistants like Google Assistant and Alexa** will enhance convenience.
4. **Multi-User and Business Accounting Support**
Expand the system to support **multi-user financial tracking**, allowing families or business teams to **collaborate on budgeting and expense management**.
5. **AI-Powered Fraud Detection and Prevention**
Implement **AI-driven fraud detection** to analyze transactions in real time and **identify suspicious activities**, enhancing security and preventing financial fraud.
6. **Automated Customer Support and Chatbots**
Introduce **AI chatbots** for **financial guidance, budgeting tips, and instant support**. This feature will improve **customer assistance by answering queries quickly and providing financial insights**.

7. References

Spring Boot for Finance Management – Building a Secure System

<https://spring.io/blog/spring-boot-financial-management>

Financial Budgeting System in Java – GitHub Repository

<https://github.com/financial-budgeting-system>

Developing a Financial Management System with Spring Boot

<https://www.udemy.com/course/developing-financial-apps/>

REST API for Budget Tracking – Step-by-Step Guide

<https://blog.codingfinance.com/building-budget-api-spring-boot>

Secure Financial Transactions with Spring Boot – Stack Overflow Discussion

<https://stackoverflow.com/questions/secure-transactions-spring-boot>

Spring Boot and MySQL Integration for Finance Apps

<https://spring.io/blog/spring-boot-mysql-integration>

Spring Boot Budget App Example – GitHub Repository

<https://github.com/budget-app-spring-boot>

Developing AI-Powered Finance Systems

<https://medium.com/developing-ai-finance-apps>

Spring Boot and JWT Security for Financial Applications

<https://spring.io/blog/spring-boot-jwt-security>

Building a Secure Financial Platform – Best Practices

<https://karan18csu103.medium.com/spring-boot-financial-app-best-practices>