

Programming Application Framework (IT3030)

2022

ElectroGrid (EG)

Power Grid Management System

Batch: Y3.S1.WD.DS.05.02

Group: 160

Submitted by:

1. IT20273712– (Subasinghe S. S)
2. IT20155698– (Rajapaksha M.T.U.R)
3. IT20161088– (Pathirana D.P.C.H)
4. IT20216900– (De Silva S. R)
5. IT20785192– (Isurika W.B.M.A)

Submitted to:

Mr. Nalaka Dissanayake

**Table of Contents**

[Introduction 5](#_Toc101824703)

[1) User Service Implementation 5](#_Toc101824704)

[2) Bill Service Implementation 5](#_Toc101824705)

[3) Payment Service Implementation 5](#_Toc101824706)

[4) Overdue Payment Service Implementation 6](#_Toc101824707)

[5) Support Service Implementation 6](#_Toc101824708)

[GitHub Details 6](#_Toc101824709)

[Commit Log 7](#_Toc101824710)

[SE Methodologies 9](#_Toc101824711)

[ Description 9](#_Toc101824712)

[ The Usage 9](#_Toc101824713)

[ Advantages 9](#_Toc101824714)

[ Disadvantages 9](#_Toc101824715)

[Time schedule (Gantt chart) 10](#_Toc101824716)

[Requirements Analysis 11](#_Toc101824717)

[1) Stakeholder Analysis (onion diagram) 11](#_Toc101824718)

[2) Technical Requirements 12](#_Toc101824719)

[3) Functional Requirements 12](#_Toc101824720)

[4) Non – Functional Requirements 12](#_Toc101824721)

[5) Requirements Modelling (Use case Diagram) 13](#_Toc101824722)

[System’s overall design 14](#_Toc101824723)

[1) Overall Architecture 14](#_Toc101824724)

[2) Overall DB Design (ER) 15](#_Toc101824725)

[1) Activity Diagram 16](#_Toc101824726)

[2) Overall Class Diagram 17](#_Toc101824727)

[Individual Sections 18](#_Toc101824728)

[User Service 18](#_Toc101824729)

[1. Service design 18](#_Toc101824730)

[2. Service development and testing 21](#_Toc101824731)

[3. Assumptions 24](#_Toc101824732)

[Bill Service 25](#_Toc101824733)

[1. Service design 25](#_Toc101824734)

[2. Service development and testing 27](#_Toc101824735)

[e) Assumptions 31](#_Toc101824736)

[Payment Service 32](#_Toc101824737)

[1. Service design 32](#_Toc101824738)

[2. Service development and testing 34](#_Toc101824739)

[3. Assumptions 37](#_Toc101824740)

[Overdue Payment Service 38](#_Toc101824741)

[1. Service design 38](#_Toc101824742)

[2. Service development and testing 42](#_Toc101824743)

[3. Assumptions 47](#_Toc101824744)

[Support Service 48](#_Toc101824745)

[1. Service design 48](#_Toc101824746)

[2. Service development and testing 50](#_Toc101824747)

[3. Assumptions 54](#_Toc101824748)

[System’s Integration Details 55](#_Toc101824749)

[1) Tools Used, Testing Methodology and Results & API Documentation 55](#_Toc101824750)

[2) The Architecture used to Design the System 55](#_Toc101824751)

[References 55](#_Toc101824752)

**Work Distribution**

|  |  |  |
| --- | --- | --- |
| **Member** | **Web Service** | **Functions** |
| IT20161088– (Pathirana D.P.C.H) | User Service | * Register users (consumers) to the system. * View user list * Update and Delete users. |
| IT20216900– (De Silva S. R) | Bill Service | * Generate a new bill * View added bill list * Update and delete added bills |
| IT20155698– (Rajapaksha M.T.U.R) | Payment Service | * Add a new payment * View payments * Update and delete payment details |
| IT20273712 – (Subasinghe S. S) | Overdue Payment Service | * Add overdue payments * View overdue payment details * Update and delete overdue payment details |
| IT20785192– (Isurika W.B.M.A) | Support Service | * Add complaints * View complaint details * Update and delete complaint details |

# Introduction

ElectroGrid is an online system which manages the power consumption of users. The systems support automatic bill creation, sharing of bills to users, user payment handling, overdue payment handling and complaint handling.

## User Service Implementation

There are mainly 3 types of users in the system. Consumer, admin, and manager. To check the bill balance, monthly electricity usage, overdue payment information through the system consumer must be registered to the system by the admin. When registering to the system as a valid user consumer should provide name, address, email, phone number. The administrator of the system will check the details and accept their registering request. After that they can login to the system by providing valid credentials and check their electricity usage and do payments. As same the consumer, the manager also must register to the system to calculate bills, accept payments, and manage overdue information. The managers are added by the administrator of the system. Moreover, the administrator of the system can search for users, update details of the users and remove inactive users from the system.

## Bill Service Implementation

Bill service mainly focuses on the management of bills according to the consumption of the consumers. Bills are handled by the bill handler and this service uses the user service. The monthly payment amount is calculated in this service and added bills can be updated or deleted when a consumer complaint is received.

## Payment Service Implementation

After confirming the relevant payment, Consumer can add payment details such as payment method(visa/master), card details (card number, name on card, cvc, expire date), bill amount to go ahead with procedure. System will make sure to validate payment details by avoiding null values and inaccurate data formats. Consumer can view all his/her payment details by payment id. Admin can manage payment details by updating payment details and deleting unnecessary payment details of the system. Payment details will be saved after payment get authorized.

## Overdue Payment Service Implementation

Overdue Payment service is mainly focuses on the unpaid payments, service suspension and service restoration. User service is used by this service. After checking the due payments, the overdue payment handler can add new overdue payments to the system. Updating and deleting an overdue payment can be done if needed.

## Support Service Implementation

Support service is mainly focuses on helping consumers’ problems. User service is mainly

used by this service. After registering and logging into the system the existing customer can

search complaint submission from the system. Then, consumer can complaint their issues

directly by filling the form. After adding complaints if they want to change it, they can edit

and submit. Consumer can view all complaints and delete their complaints if they need.

# GitHub Details

(Database included)

<https://github.com/SameeriSubasinghe/ElectroGrid.git>

# Commit Log

A screenshot of a computer

Description automatically generated with medium confidence

**A screenshot of a computer

Description automatically generated with medium confidence**

# SE Methodologies

## Description

Agile approach is an iterative software development paradigm. This is a project management methodology that divides a project into stages. The project cycle goes through planning, implementing, executing, and evaluating, and it entails continual development at every stage. As a result, this approach enables to identify software flaws considerably sooner and deliver a viable and workable solution earlier than compared to following other software development approaches.

## The Usage

* + The requirements are well defined.
  + The system development is carried out using well known tools.

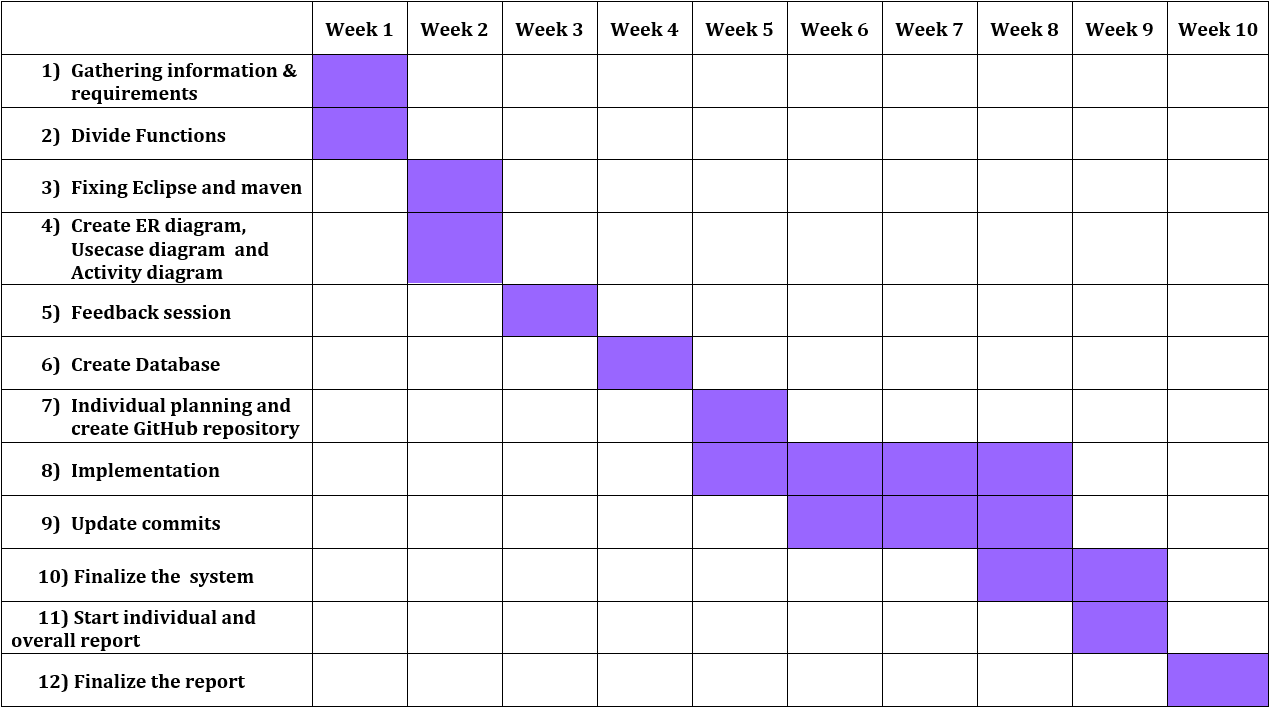
## Advantages

* + Continuous development.
  + Facilitate late changes to requirements.
  + Early detection of bugs.
  + Delivery of a working software at early stages of the project.

## Disadvantages

* + Lack of attention to designing.
  + Requires a considerable level of expertise to complete the project implementation.
  + Difficult to scale the scope of the project in early stages.

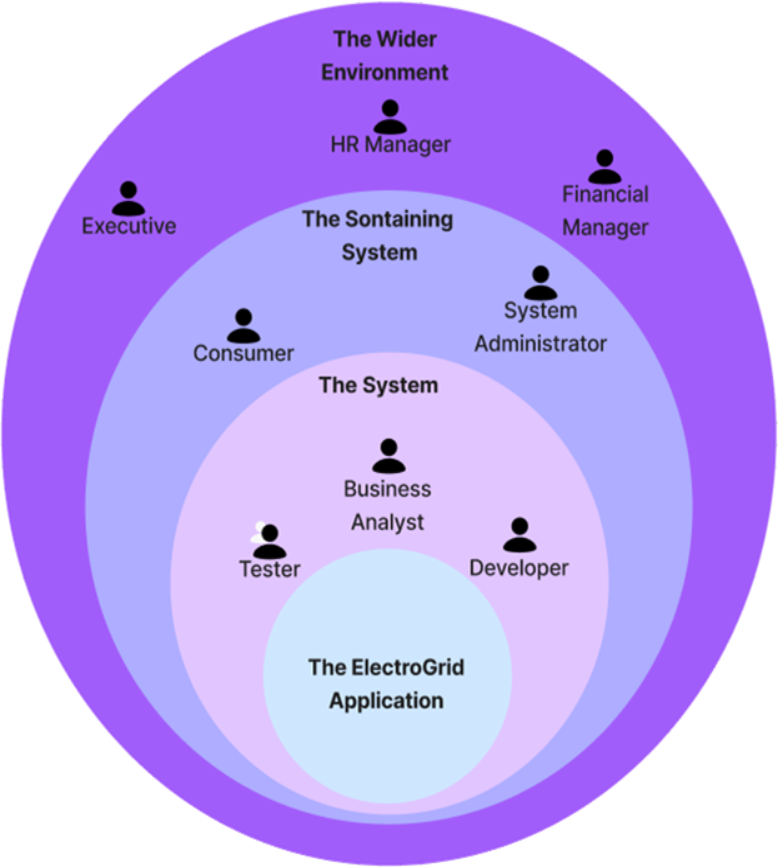
# Time schedule (Gantt chart)



# Requirements Analysis

Requirement analysis focuses on both user and system requirements. Main stakeholders of the ElectroGrid web system are Consumer, System administrator, finance manager etc. This application is a restful web application and application have five services. User service, Bill service, Payment service, Overdue Payment service and Support service.

## Stakeholder Analysis (onion diagram)



## Technical Requirements

* Technical requirements are the technical points that must be taken into consideration to deliver a successful system.
* User, Bills, Payments, Overdue Payments and Complaint details can be updated, deleted, and viewed as required.

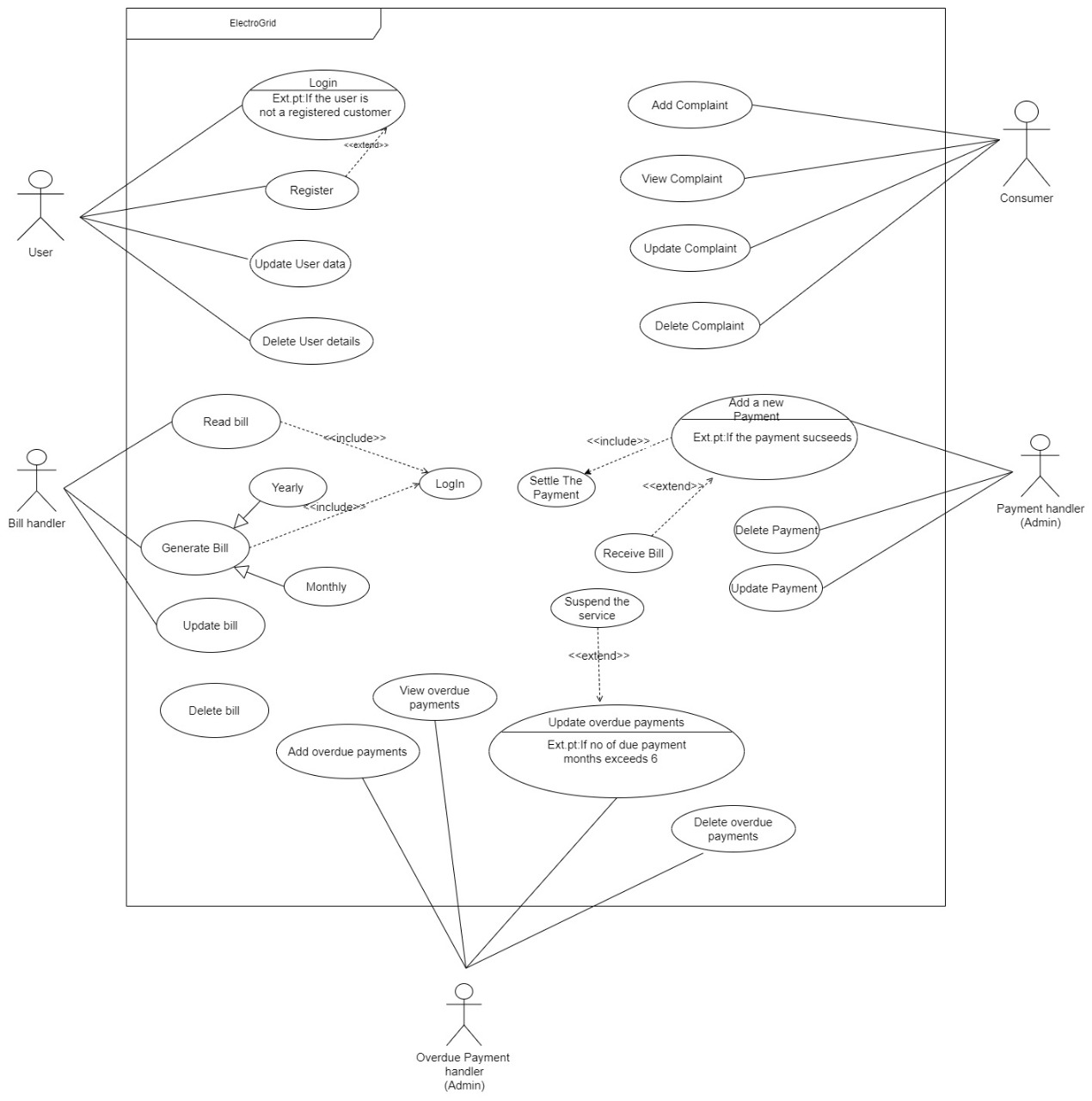
## Functional Requirements

* 1. **User management**
     + Register users (consumers) to the system.
     + View user list
     + Update and Delete users.
  2. **Bill management**
     + Generate a new bill
     + View added bill list
     + Update and delete added bills
  3. **Payment management**
     + Add a new payment
     + View payments
     + Update and delete payment details
  4. **Overdue Payment management**
     + Add overdue payments
     + View overdue payment details
     + Update and delete overdue payment details
  5. **Support management**
     + Add complaints
     + View complaint details
     + Update and delete complaint details

## Non – Functional Requirements

* Performance – Response time, Throughput, Utilization, user interface design, Conformity
* Security requirements - All data inside the system, or parts of it, is secured from virus attacks and illegal access, thanks to security regulations. The logins for each user are different. As a result, only the persons involved may modify it.
* Availability – The system accessibility 24 \* 7.
* Security – Maintenance of a system database backup.
* Software Quality Attributes
* Availability
* Maintainability
* Usability
* Accuracy
* Accessibility
* Reliability

## Requirements Modelling (Use case Diagram)

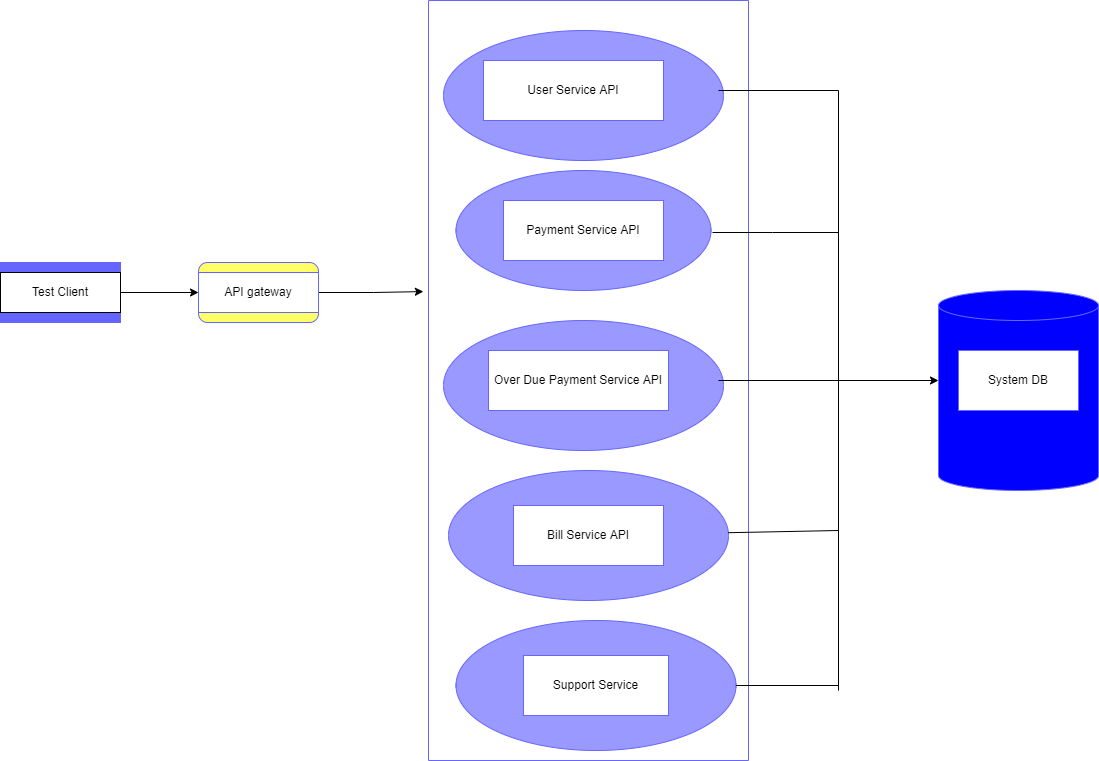


# System’s overall design

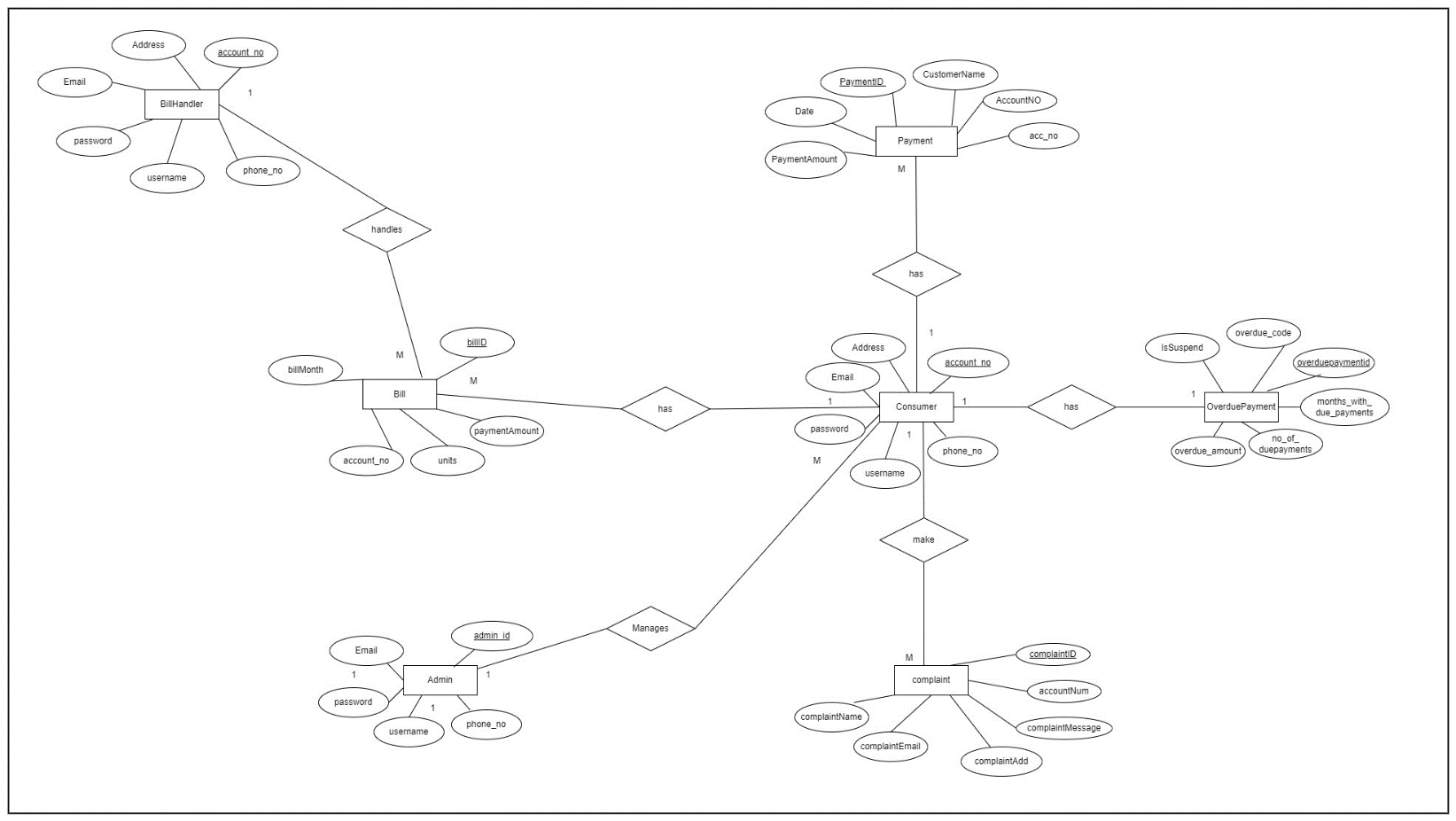
## Overall Architecture

ElectroGrid is a power management system where the registered users can record the user details, power consumption, billing, payments, overdue payments and complaints

This management system consists of five services as User Service, Payment Service, Bill Service, Overdue payment Service, and Support service All of the services share a database. Postman was also used to collect the inputs and test the outputs. When a client makes a request, the data is transferred through a gateway to the relevant service, and the response is provided back to the client in the same way. APIs (Application Programming Interfaces) allow these procedures, such as improving current services, to work independently and more efficiently.



## Overall DB Design (ER)



## Diagram, schematic Description automatically generatedActivity Diagram

## Overall Class DiagramDiagram Description automatically generated

# Individual Sections

## User Service

### Service design

Admin can register users to the system using registration form by providing their details. After that user can login to the system as a valid user by providing valid credentials. The access to the system varies according to the user type. Admin can update and delete users if needed.

#### API of the service

* + 1. **Create User (POST)**

**Resource:** Registration

**Request:** POST User\_Service/RegistrationAPI/Registration

**Media:** Form Data - URL encoded

**Data:**

regName : “Mia”

regAddress : “Malabe”

regEmail: “mia@gmail.com”

regDate: “2020.04.01”

regPNo: “0777788263”

**Response:** Inserted successfully

**URL:** <http://localhost:8083/User_Service/RegistrationAPI/Registration>

* + 1. **Update User (PUT)**

**Resource:** Registration

**Request:** PUT User\_Service/RegistrationAPI/Registration

**Media:** Form data – Application JSON

**Data:** {

"regName":"Mia",

"regAddress":"kaduwela",

"regEmail":"mia@gmail.com",

"regDate":"20.2.2022",

"regPNo":"0710768124",

"regID":"1"

}

**Response:** Updated successfully

**URL:** <http://localhost:8083/User_Service/RegistrationAPI/Registration>

* + 1. **View User (GET)**

**Resource:** Registration

**Request**: GET User\_Service/RegistrationAPI/Registration

**Media:** Form Data

**Response:** HTML table with all attributes in the User table

**URL:** <http://localhost:8083/User_Service/RegistrationAPI/Registration>

* + 1. **Delete User (DELETE)**

**Resource:** Registration

**Request:** DELETE User\_Service/RegistrationAPI/Registration

**Media:** Application XML

**Data:** <regData>

<regID>1</regID>

</regData>

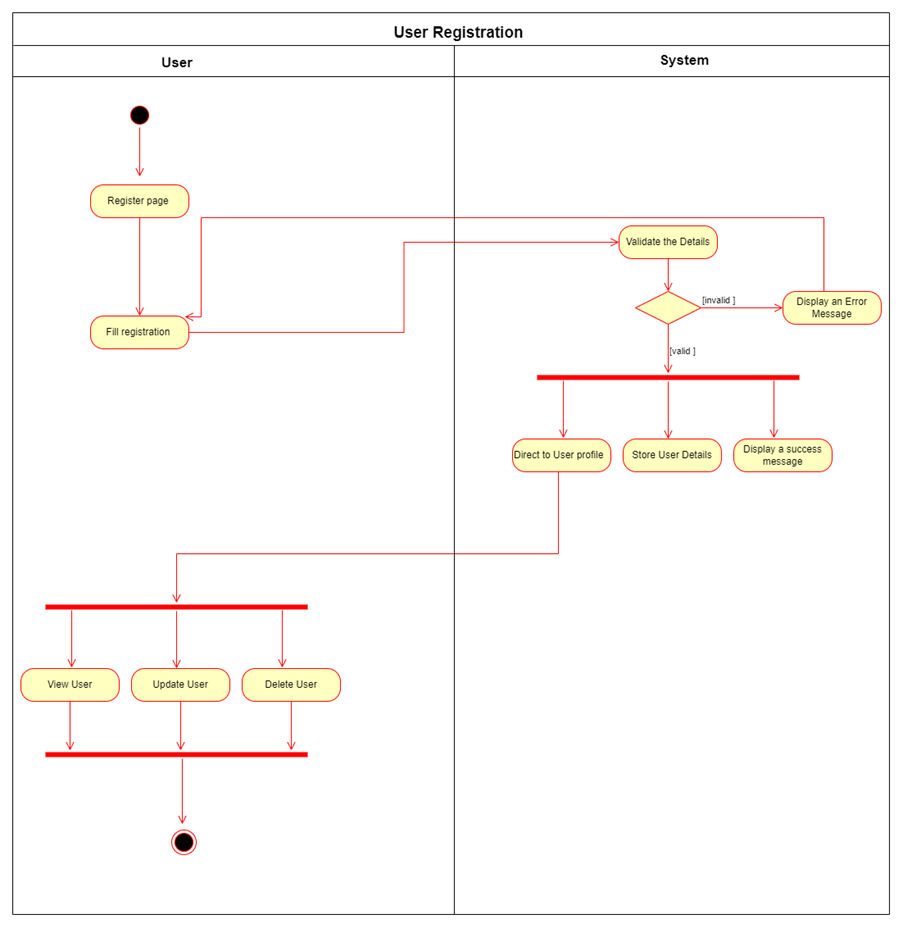
**Response:** Deleted successfully

**URL:** <http://localhost:8083/User_Service/RegistrationAPI/Registration>

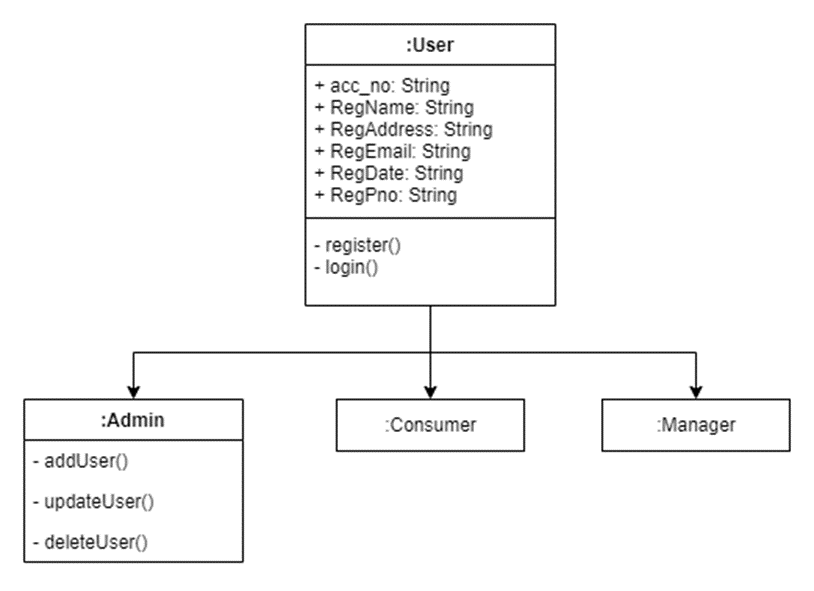
#### Internal logic

The responsibility of the user service is to manage the details of the users who register to the system. Mainly there are 3 types of users: consumer, manager, and admin. When registering to the system as a consumer they should provide name and contact details. The administrator of the system will check and accept their registering request. Then they are given the access to check their electricity usage and do payments. The manager can register to the system and calculate bills, check complains and requests of the consumers and handle overdue payment details.

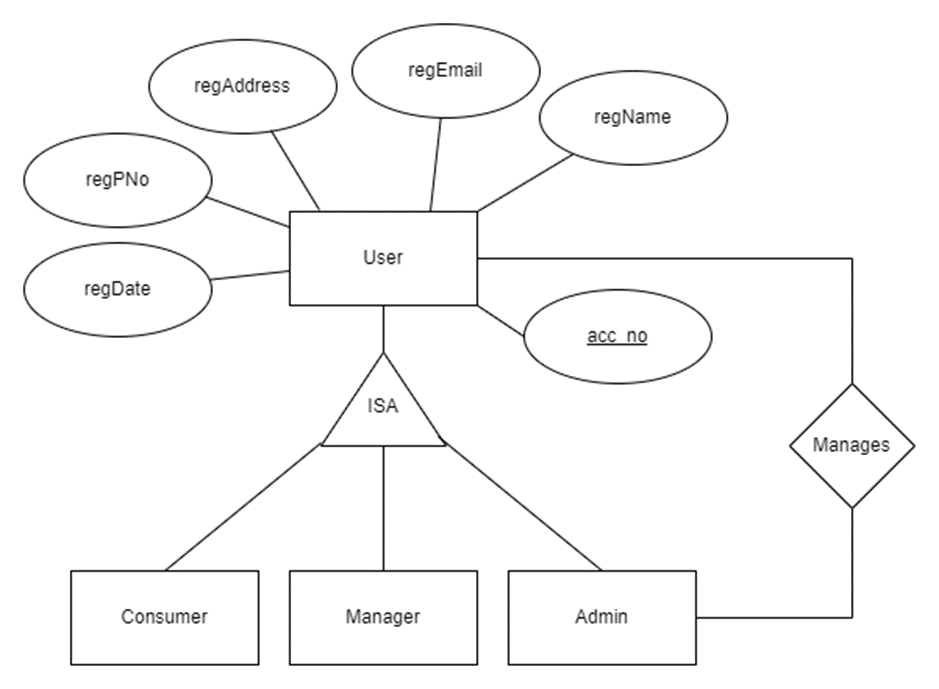
#### Activity Diagram



#### Class Diagram



#### Database for the service (ER)



### Service development and testing

#### Tools used

* Dependency Management Tool: Maven
* Testing Tool: Postman
* Version Control System: Git
* IDE: eclipse
* Programming Language: Jersey framework (JAX-RS)
* Programming Language: Java
* Database: phpMyAdmin (MySQL)
* Server: Apache Tomcat Server

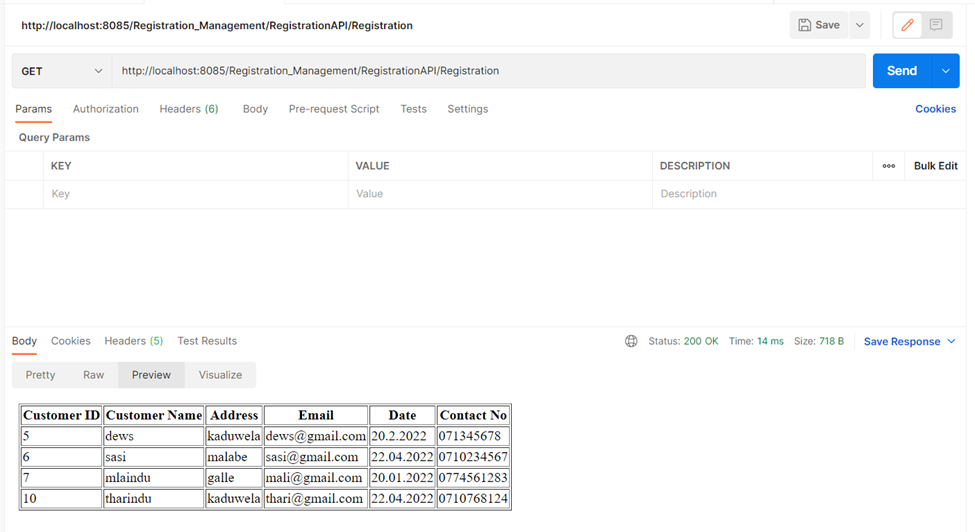
#### Testing methodology and results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Input** | **Expected Output** | **Actual Output** | **Result** |
| 1 | Create user | regName : “Mia”  regAddress : “Malabe”  regEmail: “[mia@gmail.com](mailto:mia@gmail.com)”  regDate: “2020.04.01”  regPNo: “0777788263” | Inserted Successfully | Inserted Successfully | Pass |
| 2 | View user |  | Display a HTML table with all the attributes in user table | Display a HTML table with all the attributes in user table | Pass |
| 3 | Update user | {  "regName":"Mia",  "regAddress":"kaduwela",  "regEmail":"[mia@gmail.com](mailto:mia@gmail.com)",  "regDate":"20.2.2022",  "regPNo":"0710768124",  "regID":"1"  } | Updated successfully | Updated successfully | Pass |
| 4 | Delete user | <regData>  <regID>1</regID>  </regData> | Deleted successfully | Deleted successfully | Pass |

* 1. Add User

**Graphical user interface, text, application, email

Description automatically generated**

* 1. ****View User
  2. Update User

**Graphical user interface, text, application, email

Description automatically generated**

* 1. **Graphical user interface, text, application, email

     Description automatically generated**Delete User

### Assumptions

* There are mainly three types of users in the system consumer, manager, and admin
* Admin has the responsibility to manage all the users of the system
* Manager and Consumers are added by the administrator to maintain the transactions of the system.

## Bill Service

### Service design

Bill handler is the one who mainly manages this service. This service is the one which oversees the bill creation by taking the consumption of the consumers.

#### API of the service

****

1. **Create Bill (POST)**

**Resource:** Bills

**Request:** POST -http://localhost:8080/BillService/ElectroService/Bills

**Media Type:** Form Data - APPLICATION\_FORM\_URLENCODED

**Data:** billCode, electricityAccountNo, billMonth , units , paymentAmount

**Response:** String status message “Inserted successfully”

**URL:** <http://localhost:8080/BillService/ElectroService/Bills>

1. **View Bill (GET)**

**Resource:** Bills

**Request:** GET - http://localhost:8080/BillService/ElectroService/Bills

**Media Type:** TEXT\_HTML

**Data:** billCode, electricityAccountNo, billMonth , units , paymentAmount

**Response:** HTML table with billCode, electricityAccountNo, billMonth , units , paymentAmount

**URL** : <http://localhost:8080/BillService/ElectroService/Bills>

1. **Update Bill (PUT)**

**Resource:** Bills

**Request:** PUT http://localhost:8080/BillService/ElectroService/Bills

**Media Type:** APPLICATION\_JSON, TEXT\_PLAIN

**Data:** billCode, electricityAccountNo, billMonth , units , paymentAmount

**Response:** String status message “Updated successfully”

**URL:** <http://localhost:8080/BillService/ElectroService/Bills>

1. **Delete Bill (DELETE)**

**Resource:** Bills

**Request:** DELETE http://localhost:8080/BillService/ElectroService/Bills

**Media Type:** APPLICATION\_XML

**Data:** <BillData >

<billID> 3 <billID>

<BillData >

**Response:** String status message “Deleted successfully”

**URL:** <http://localhost:8080/BillService/ElectroService/Bills>

#### Internal logic

The bills are handled by the bill handler. The bill is generated by calculating the monthly payment amount. If a consumer complaint is received, an added bill can be updated or deleted accordingly.

#### Activity Diagram

#### Class Diagram

#### Database for the service (ER)

### Service development and testing

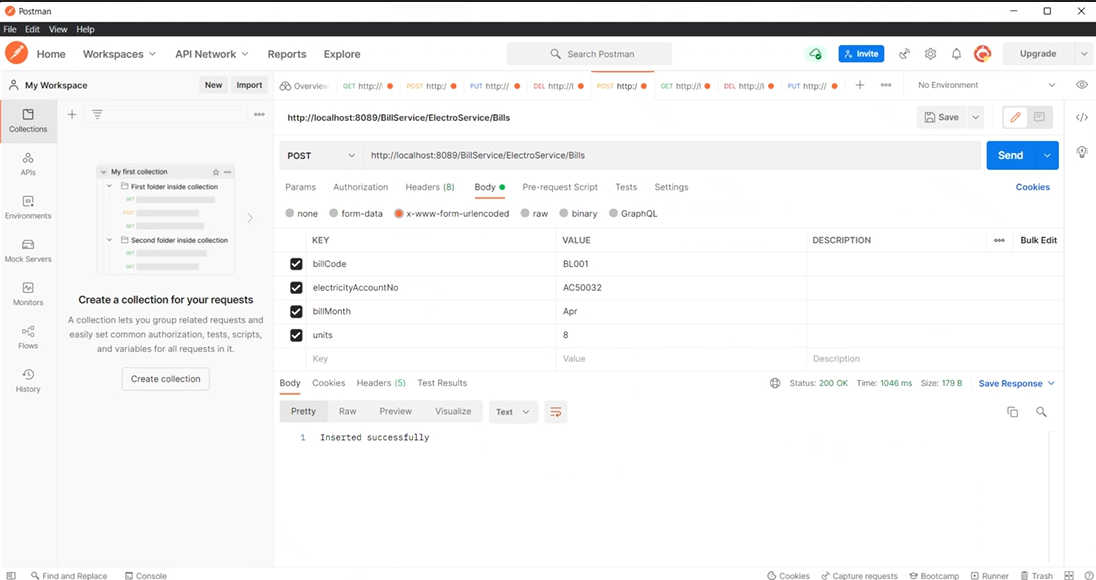
#### Tools used

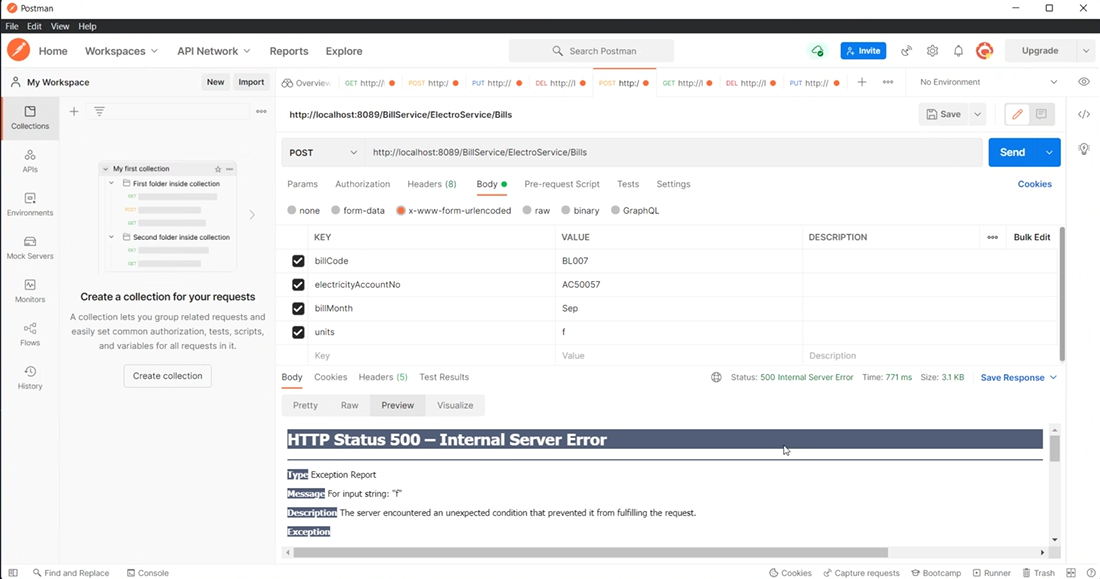
* Dependency Management Tool: Maven
* Testing Tool: Postman
* Version Control System: Git
* IDE: eclipse
* Programming Language: Jersey framework (JAX-RS)
* Programming Language: Java
* Database: phpMyAdmin (MySQL)
* Server: Apache Tomcat Server
* Code quality checking tool: Sonar Lint

#### Testing methodology and results

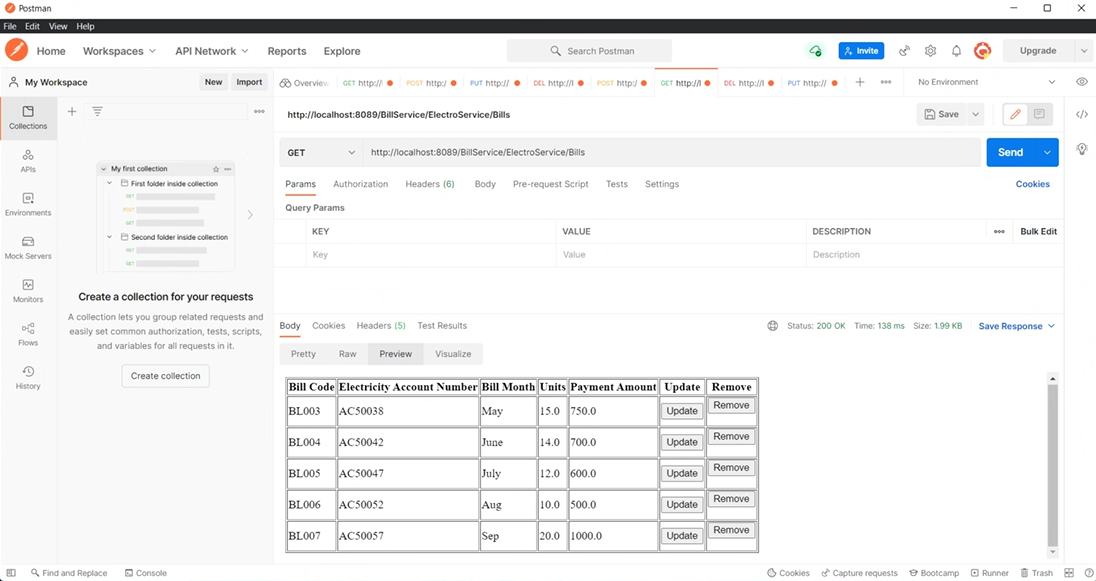
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Input** | **Expected Output** | **Actual Output** | **Result** |
| 1 | Add Bill details | billCode = “BL003”  electricityAccountNo = “AC50038”  billMonth = “May”  units = “15.0”  paymentAmount = “750.0” | Display message “Inserted successfully” | Display message “Inserted successfully” | Pass |
| 1 | Add Bill details | billCode = “BL003”  electricityAccountNo = “AC50038”  billMonth = “May”  units = “f”  paymentAmount = “750.0” | Display message “Error while inserting” | Display message “Error while inserting” | Pass |
| 2 | Update Bill | billCode = “BL004”  electricityAccountNo = “AC50042”  billMonth = “June”  units = “14.0”  paymentAmount = “700.0” | Display message “Updated successfully” | Display message “Updated successfully” | Pass |
| 3 | View Bills |  | Display Bill details.” | Display Bill details. | Pass |
| 4 | Delete Bill | <BillData >  <billID> 3 <billID>  <BillData > | Display message “Deleted successfully” | Display message “Deleted successfully” | Pass |

#### Postman Test Results

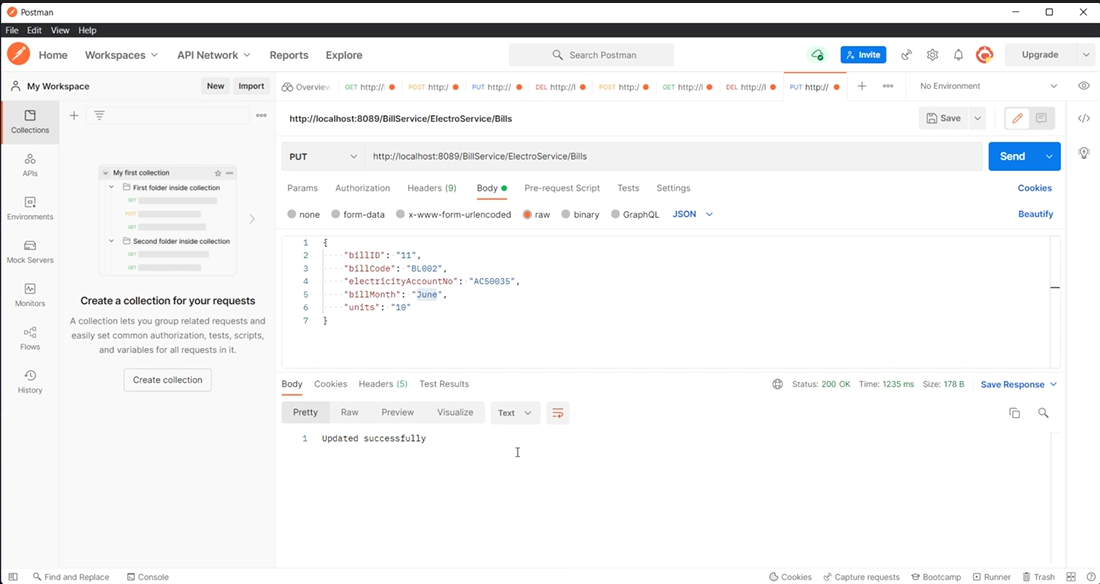
1. Add Bill



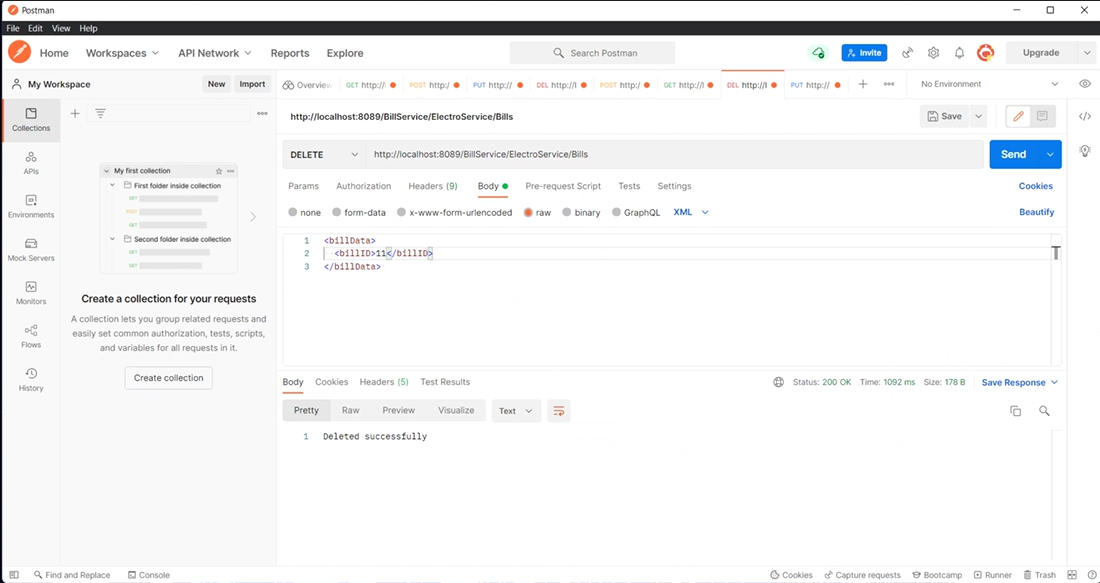
1. View Bill



1. Update Bill



1. Delete Bill



#### Graphical user interface Description automatically generatedCode quality check SonarLint Results

### Assumptions

* When a consumer makes a complaint, bill details can be updated and deleted
* Bills are managed by the Bill handler which is a role inherited from the system administrator.

## Payment Service

### Service design

Admin will approve customer’s payment after confirming payment details. After payment get authorized system will automatically save each customer’s payment details. Admin can add, update, or delete any transaction details.

#### API of the service

1. **Create Payment (POST)**

**Resource:** Payment

**Request:** POST

**Media:** Form data – URL encoded

**Data:** PaymentID,Customer Name,Data,Acccount number

**Response:** Inserted successfully

**URL:** <http://localhost:8080/PaymentManagement/PaymentAPI/Payment>

1. **Update Payment (PUT)**

**Resource:** Payment

**Request:** PUT

**Media**: form data – application JASON

**Data:** PaymentID,Customer Name,Data,Acccount numbers

**Response:** Inserted successfully

**URL:** <http://localhost:8080/PaymentManagement/PaymentAPI/Payment>

1. **View Payment (GET)**

**Resource:** Payment

**Request:** GET

**Media:** form data

**Response:** Updated successfully

**URL:** <http://localhost:8080/PaymentManagement/PaymentAPI/Payment>

1. **Delete Payment (DELETE)**

**Resource:** Payment

**Request:** DELETE

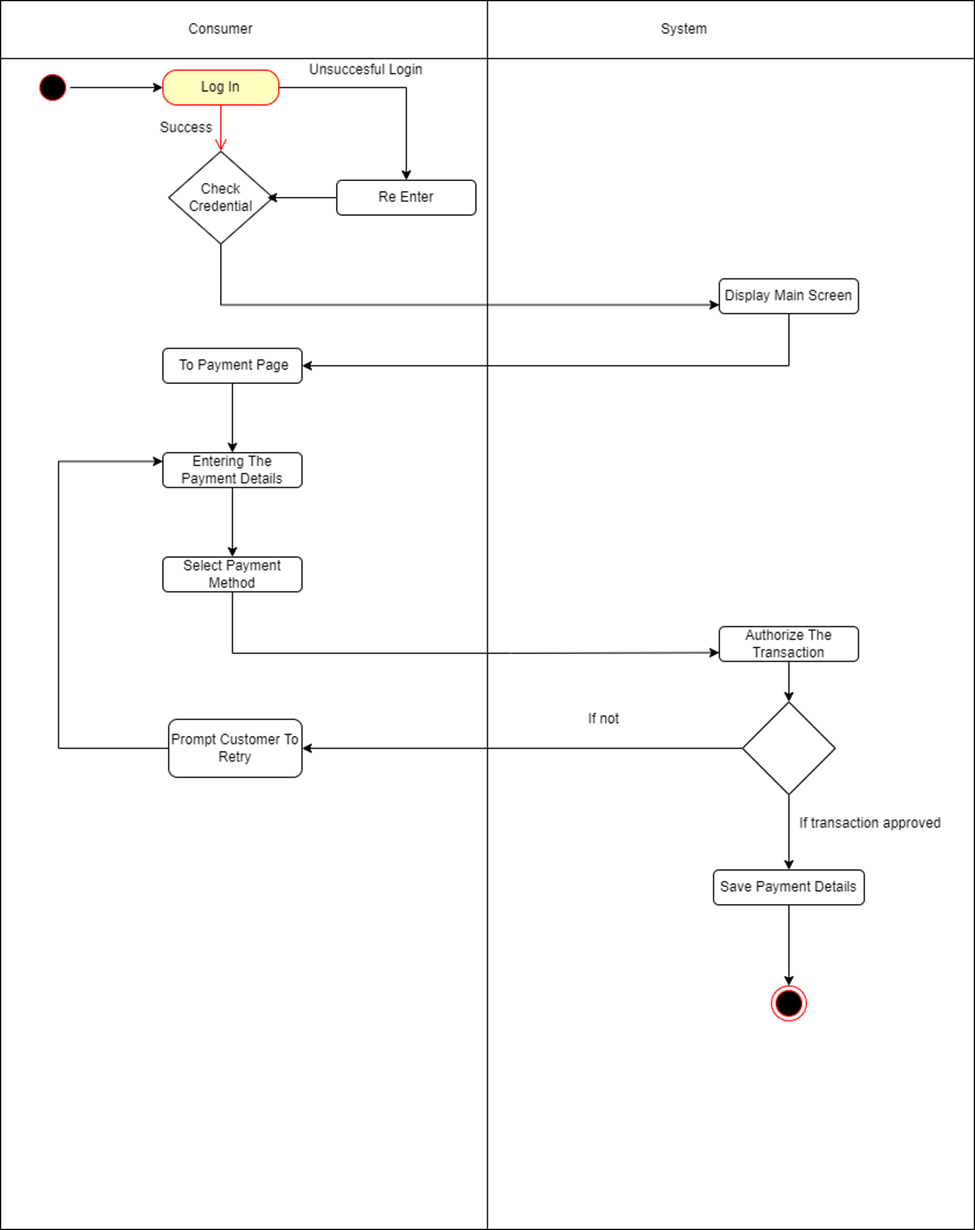
**Media:** application XML

**Data:** PaymentID

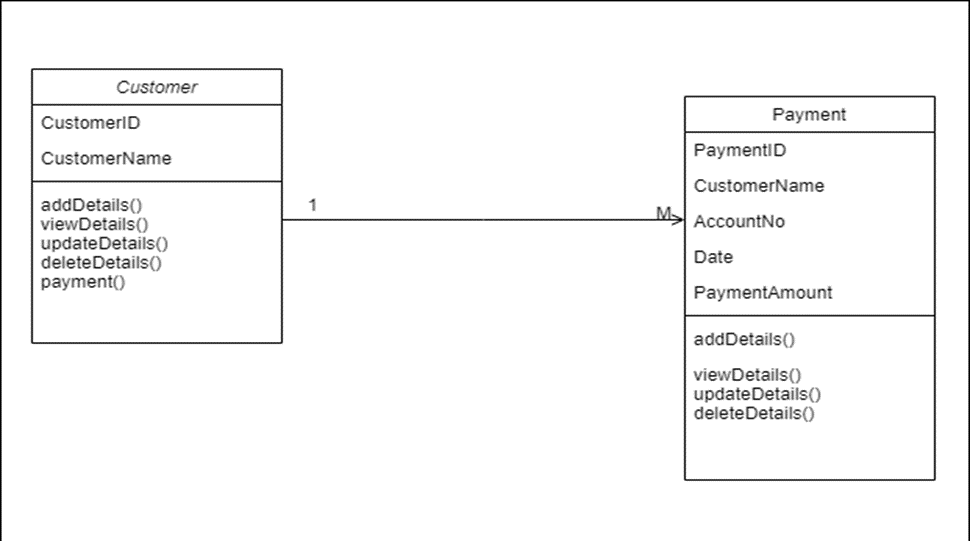
**Response:** Deleted successfully

**URL:** <http://localhost:8080/PaymentManagement/PaymentAPI/Payment>

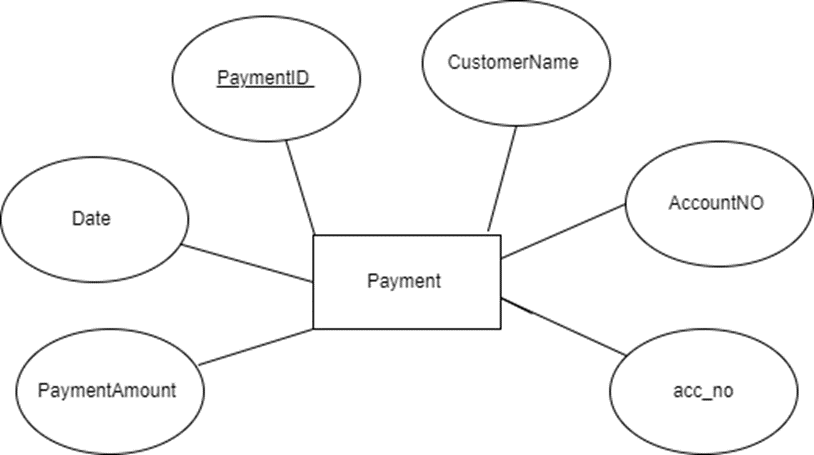
#### Internal logic (Activity Diagram)



#### Class Diagram



#### Database for the service (ER)



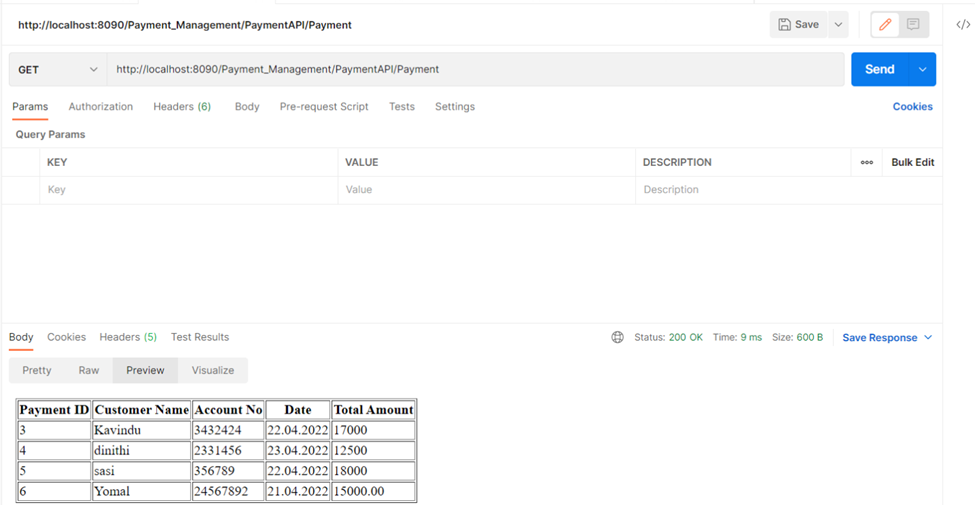
### Service development and testing

#### Tools used

* Dependency Management Tool: Maven
* Testing Tool: Postman
* Version Control System: Git
* IDE: eclipse
* Programming Language: Jersey framework (JAX-RS)
* Programming Language: Java
* Database: phpMyAdmin (MySQL)
* Server: Apache Tomcat Server

#### Testing methodology and results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Input** | **Expected Output** | **Actual Output** | **Result** |
| 1 | Create Payment | PaymentID CustomerName Payment Amount  AccountNo date | Display Message “Payment Done successfully” | “Payment Done Successfully “Message displayed | Pass |
| 2 | View Payment | PaymentID | Display Payment details for the given ID | Relevant details are displayed | Pass |
| 3 | Update Payment | PaymentID CustomerName Payment Amount  AccountNo date | Display Message “Payment Details Updated successfully” | “Payment Details Updated successfully” Message is displayed. | Pass |
| 4 | Delete Payment | PaymentID | Display message “Payment Details deleted successfully” | “Payment Details deleted successfully” message displayed. | pass |

* 1. View Payment
  2. Add Payment

Graphical user interface, application

Description automatically generated

* 1. Graphical user interface, text, application, email

     Description automatically generatedUpdate Payment
  2. Graphical user interface, text, application, email

     Description automatically generatedDelete Payment

### Assumptions

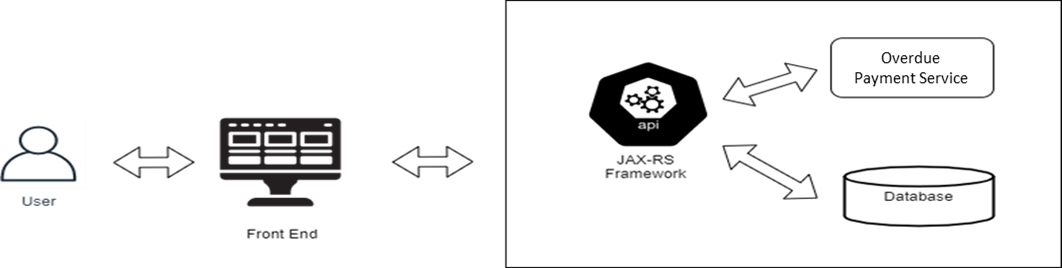
* Only admin can add new payment, update, and Delete payment entries.
* System will calculate the bill amount for relevant power usage.
* Only admin can View all payment details.
* User can delete Payment details within 7 days after they make the payment.

## Overdue Payment Service

### Service design

Overdue payment service manages the overdue payments and service suspension. Overdue payment handler facilitates the management of due payments, service suspension due to overdue payments and service restorations.

#### API of the service

****

1. **Read Overdue Payments (GET)**

**Resource:** ODPayments

**Request:** GET http://localhost:8083/OverduePaymentService/ODPayService/ODPayments

**Media Type:** TEXT\_HTML

**Response:** HTML table with all attributes in overdue payments(odspayments) table

**URL:** <http://localhost:8083/OverduePaymentService/ODPayService/ODPayments>

1. **Create Overdue Payments (POST)**

**Resource:** ODPayments

**Request:** POST http://localhost:8083/OverduePaymentService/ODPayService/ODPayments

**Media Type:** Form Data - APPLICATION\_FORM\_URLENCODED

**Data:** ODCode, dueAmount, dueMonthsNo, dueMonths, accountNo, IsSuspend

**Response:** String message displayed as “Inserted Successfully” or “Error while inserting the Overdue Payment”

**URL:** <http://localhost:8083/OverduePaymentService/ODPayService/ODPayments>

1. **Update Overdue Payments (PUT)**

**Resource:** ODPayments

**Request:** PUT http://localhost:8083/OverduePaymentService/ODPayService/ODPayments

**Media Type:** Form Data - APPLICATION\_JSON

**Data:** ODPaymentID, ODCode, dueAmount, dueMonthsNo, dueMonths, accountNo, IsSuspend

**Response:** String message displayed as “Updated Successfully” or “Error while updating the Overdue Payment.”

**URL:** <http://localhost:8083/OverduePaymentService/ODPayService/ODPayments>

1. **Delete Overdue Payments (DELETE)**

**Resource:** ODPayments

**Request:** DELETE http://localhost:8083/OverduePaymentService/ODPayService/ODPayments

**Media Type:** APPLICATION\_XML

**Data:** ODPaymentID

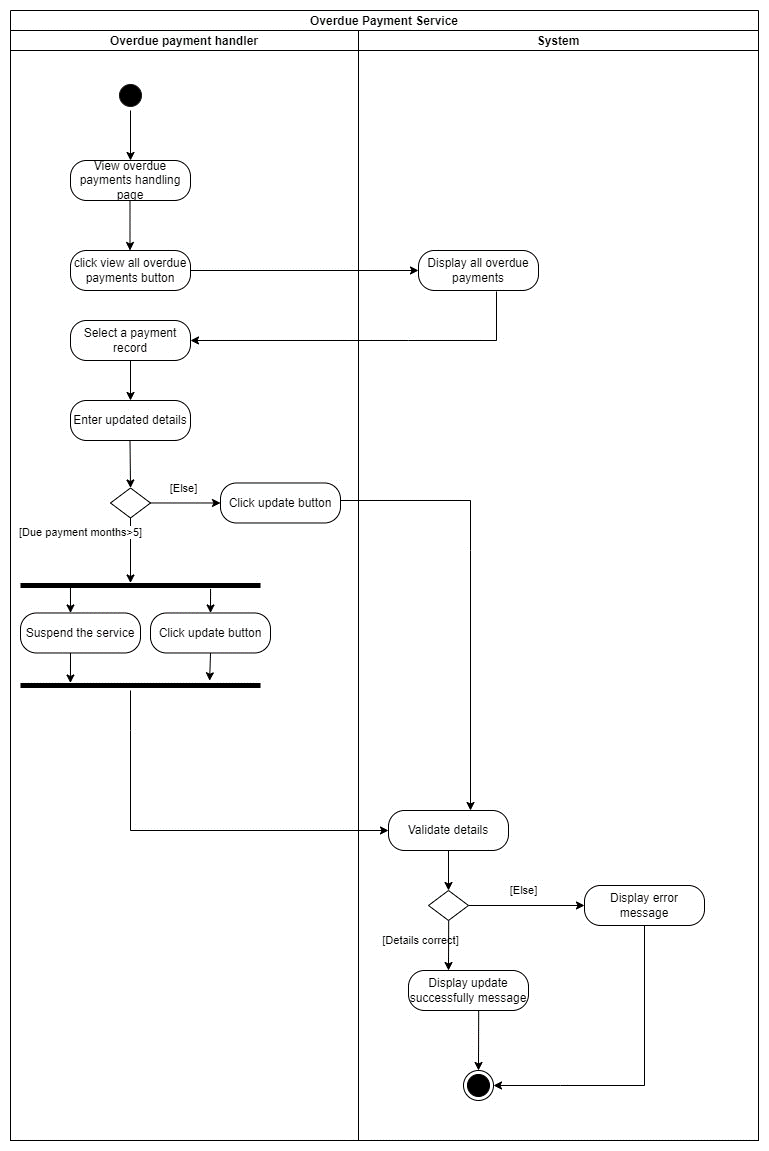
**Response:** String message displayed as “Deleted Successfully” or “Error while updating the Overdue Payment.”

**URL:** <http://localhost:8083/OverduePaymentService/ODPayService/ODPayments>

#### Internal logic

Overdue Payments are handled by the Overdue payment handler. In this service, overdue payments are created by considering the unpaid payments for three months. Added overdue payments can be updated if the relevant user is having more due payments. If the no of due months is six or exceeds six months, the overdue payment handler is responsible for the service suspension for the relevant consumer. If the consumer pays a full amount or a part of the total due amount overdue payments can be removed and if there’s a service suspension it can be restored.

#### Activity Diagram



#### Class Diagram

#### Flow chart

#### 

#### Database for the service (ER)

### Service development and testing

#### Tools used

• Dependency Management Tool: Maven

• Testing Tool: Postman

• Version Control System: Git (To integrate the system)

• IDE: eclipse

• Programming Language: Jersey framework (JAX-RS) / Java

• Database: phpMyAdmin (MySQL)

• Server: Apache Tomcat Server

• Code quality checking tool: Sonar Lint

#### Testing methodology and results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Input** | **Expected output** | **Actual output** | **Result** |
| **1** | Add Overdue payment | ODCode: “OD109”  dueAmount:”4520”  dueMonthsNo:”3”  dueMonths:” April\_2021, May\_2021, June\_2021”  accountNo:”UI208”  IsSuspend: “false” | Displays “Inserted Successfully” message | Displays “Inserted Successfully” message | Pass |
| **2** | Add Overdue payment | ODCode: “OD110”  dueAmount:” march”  dueMonthsNo:”3”  dueMonths: “April\_2021, May\_2021, June\_2021”  accountNo:”UI208”  IsSuspend:” false” | Displays “Error while inserting the overdue Payment” message | Displays “Error while inserting the overdue Payment” message | Pass |
| **3** | View Overdue payments |  | Displays HTML Table | Displays HTML Table | Pass |
| **4** | Update  Overdue payments | ODPaymentID: "1"  ODCode: "OD104"  dueAmount:"8500.00"  dueMonthsNo: “6",  dueMonths:"March\_2021, April\_2021, May\_2021, June\_2021, July\_2021, August\_2021",  accountNo: "U126",  IsSuspend": "true" | Displays “Updated Successfully” message | Displays “Updated Successfully” message | Pass |
| 5 | Update  Overdue payments | ODPaymentID: "1"  ODCode: "OD104"  dueAmount:"8500.00"  dueMonthsNo: “march",  dueMonths:"March\_2021, April\_2021, May\_2021, June\_2021, July\_2021, August\_2021",  accountNo: "U126",  IsSuspend": "true" | Displays “Error while updating the Overdue Payment” message | Displays “Error while updating the Overdue Payment” message | Pass |
| **6** | Delete Overdue payment | ODPaymentID: "8" | Displays “Deleted  Successfully” message | Displays “Deleted  Successfully” message | Pass |

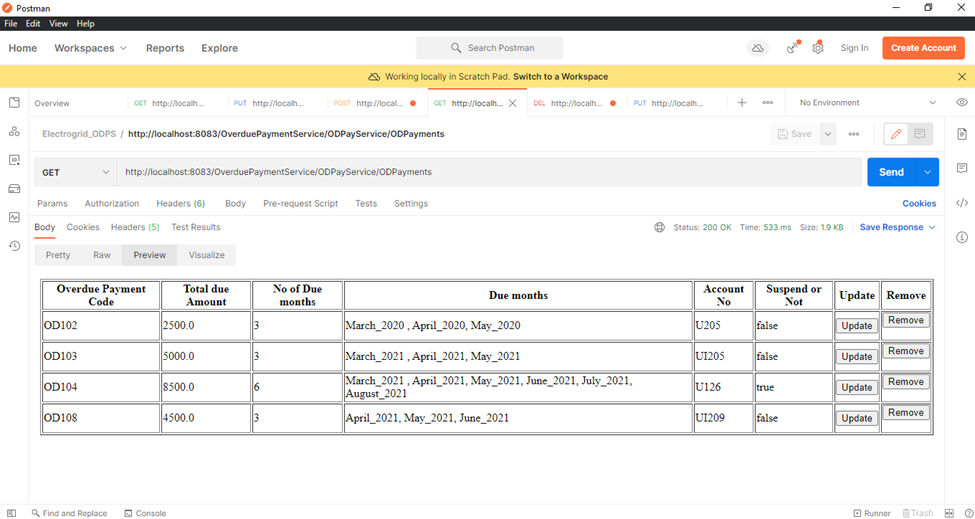
#### Postman Test Results

1. Graphical user interface, text, application, email, website

   Description automatically generatedAdd Overdue Payment

Graphical user interface, text, application, email

Description automatically generated

1. View Overdue Payments
2. Graphical user interface, text, application, email

   Description automatically generatedUpdate Overdue Payments

Graphical user interface, text, application, email

Description automatically generated

1. Delete Overdue Payments

Graphical user interface, text, application, email

Description automatically generated

#### A screenshot of a computer Description automatically generatedCode Quality Check Sonar Lint Results

-

### Assumptions

* Overdue payments are managed by the Overdue payment handler which is a role inherited from the system administrator.
* Overdue payments are added to the system by checking the due payments which exceeds three months.
* Added overdue payments can be updated every month for payments made.
* If the number of due payment months exceeds five months, then the service will be suspended.
* Overdue payments will be deleted when the consumer makes a payment.

## Support Service

### Service design

Support service is meant to manage all the complaints of consumers. This service is operated by the admin. Consumers can complaint the issue by entering their details. Also, they will update and delete the added complaints as needed. All consumers and admins can read complaints in detail.

#### API of the service

****

1. **Create Complaint (POST)**

**Resource:** SupportS

**Request:** POST - http://localhost:8080/SupportService/supportService/SupportS

**Media Type:** Form Data - APPLICATION\_FORM\_URLENCODED

**Data:** accountNum, complaintName , complaintAdd, complaintPhone, complaintEmail, complaintMessage

**Response:** String status message “Inserted successfully”

**URL:** <http://localhost:8080/SupportService/supportService/SupportS>

1. **View Complaint (GET)**

**Resource:** SupportS

**Request:** GET - http://localhost:8080/SupportService/supportService/SupportS

**Media Type**: Form Data - TEXT\_HTML

**Data:** complaintID , accountNum, complaintName , complaintAdd, complaintPhone, complaintEmail, complaintMessage

**Response:** : HTML table with complaintID , accountNum, complaintName , complaintAdd, complaintPhone, complaintEmail, complaintMessage

**URL:** <http://localhost:8080/SupportService/supportService/SupportS>

1. **Update Complaint (GET)**

**Resource:** SupportS

**Request:** http://localhost:8080/SupportService/supportService/SupportS

**Media Type:** APPLICATION\_JSON, TEXT\_PLAIN

**Data:** complaintID , accountNum, complaintName , complaintAdd, complaintPhone, complaintEmail, complaintMessage

**Response:** String status message “Updated successfully”

**URL:** <http://localhost:8080/SupportService/supportService/SupportS>

1. **Delete Complaint (DELETE)**

**Resource:** SupportS

**Request:** http://localhost:8080/SupportService/supportService/SupportS

**Media Type:** APPLICATION\_XML

**Data:** <ComplaintDetails>

<complaintID> 3 </complaintID>

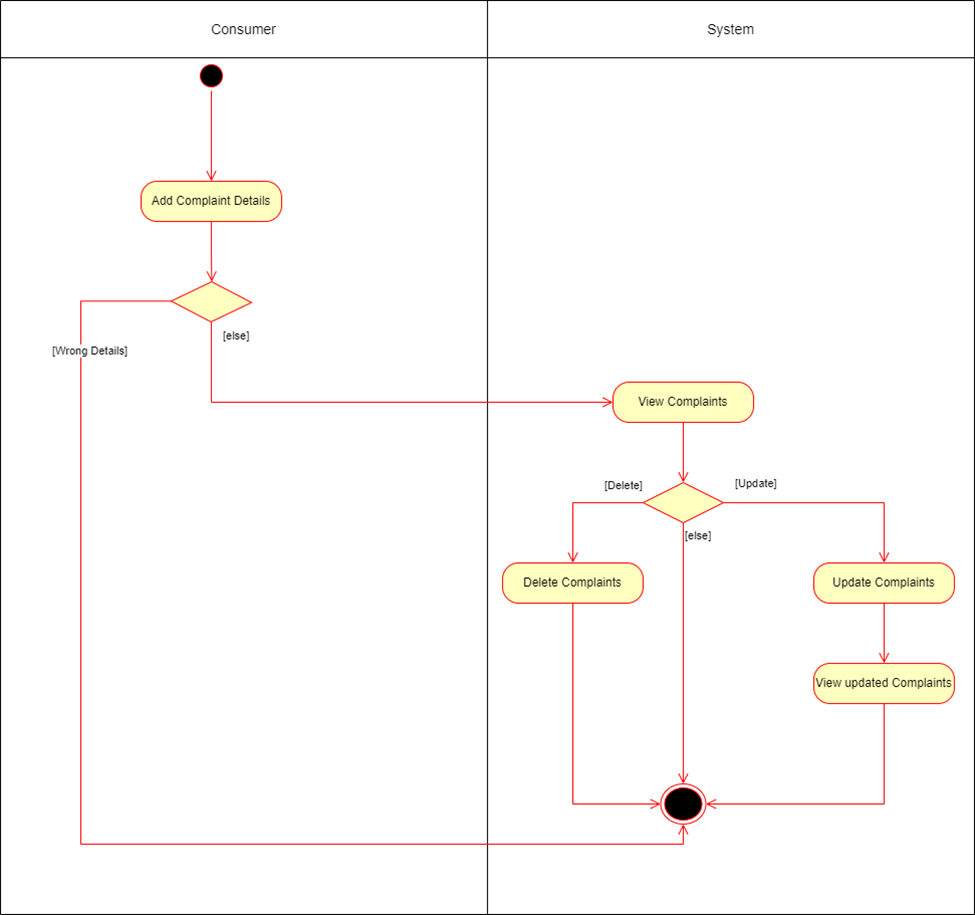
</ComplaintDetails>

**Response:** String status message “Deleted successfully”

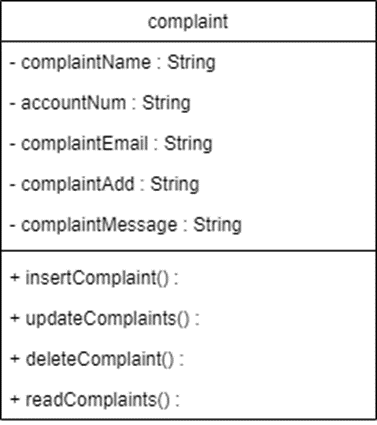
**URL:** <http://localhost:8080/SupportService/supportService/SupportS>

#### Internal logic (Activity Diagram)

Consumer can directly add their complaints. Consumer can add their details into form and add complaints. After adding they can delete or edit their complaints.



#### Class Diagram



#### Database for the service (ER)

### Service development and testing

#### Tools used

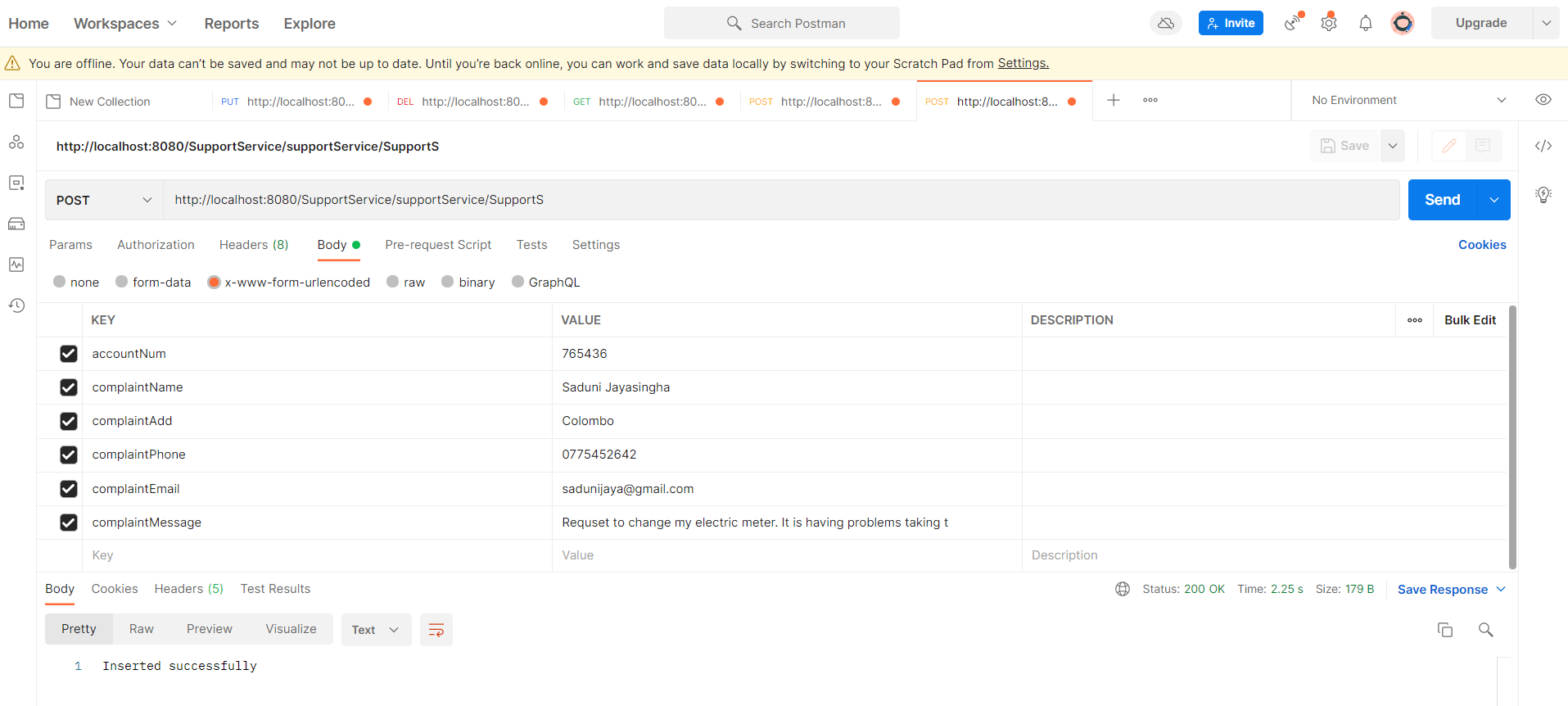
* Dependency Management Tool: Maven
* Testing Tool: Postman
* Version Control System: Git
* IDE: eclipse
* Programming Language: Jersey framework (JAX-RS)
* Programming Language: Java
* Database: phpMyAdmin (MySQL)
* Server: Apache Tomcat Server
* Code quality checking tool: Sonar Lint

#### Testing methodology and results

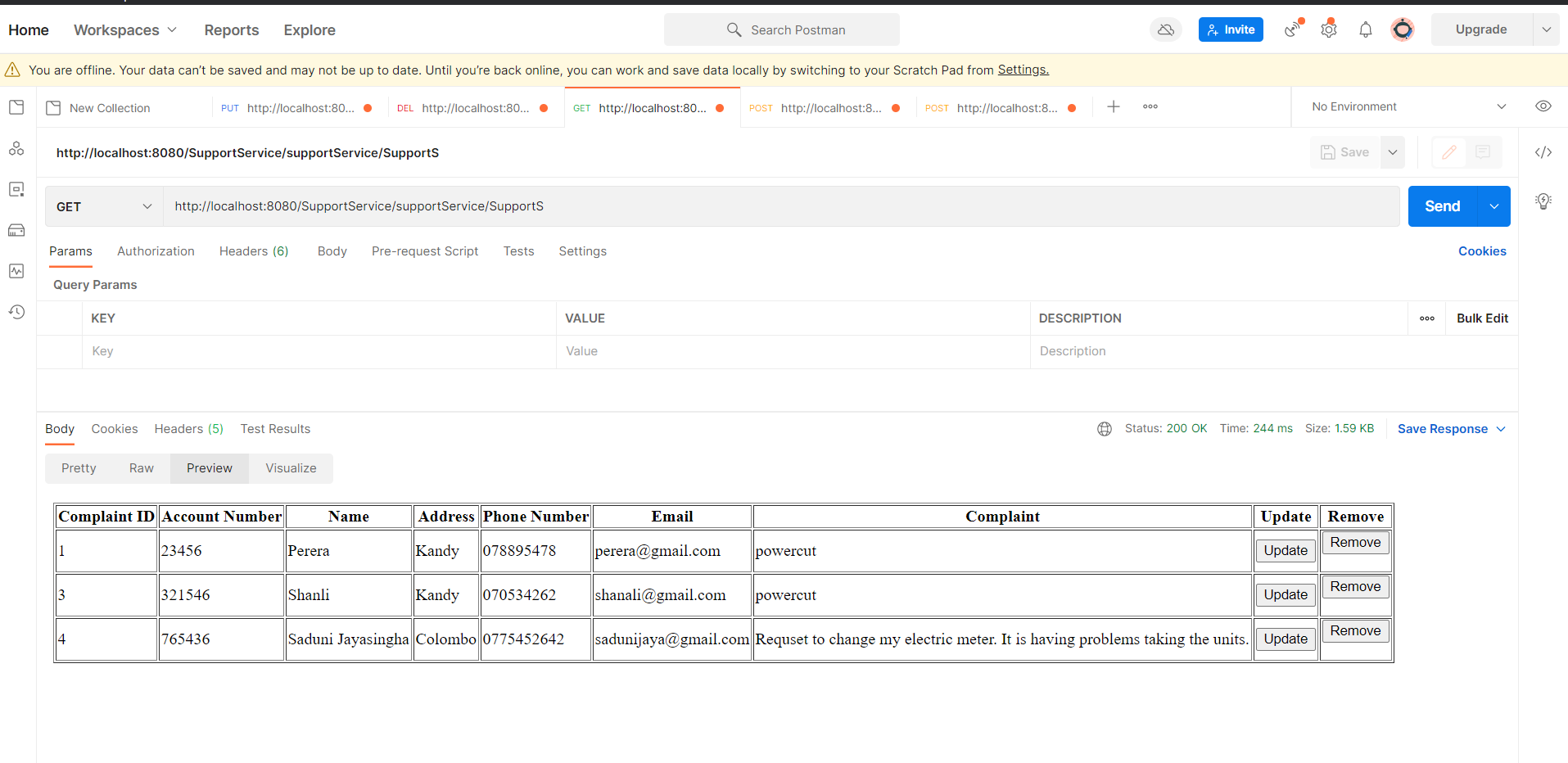
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Input** | **Expected output** | **Actual output** | **Result** |
| 1 | Add complaint | accountNum= “765432”  complaintName = “Saduni Jayasingha”  complaintAdd = “Colombo”  complaintPhone = 0775452642  complaintEmail = “sadunijaya@gmail.com”  complaintMessage = “Requset to change my electric meter. It is having problems taking the units.” | Display message “Inserted successfully” | Display message “Inserted successfully” | Pass |
| 2 | Update complaint | complaintID =”1”  accountNum= “764932”  complaintName = “Perera  complaintAdd = “NuwaraEliya”  complaintPhone = 0775452642  complaintEmail = “perera@gmail.com”  complaintMessage = “Requset to change my electric meter. It is having problems taking the units.” | Display message “Updated successfully” | Display message “Updated successfully” | Pass |
| 3 | View complaints |  | Display complaint details. | Display complaint details. | Pass |
| 4 | Delete complaint | <ComplaintDetails>  <complaintID> 3 </complaintID>  </ComplaintDetails> | Display message “Deleted successfully” | Display message “Deleted successfully” | Pass |

#### Postman Test Results

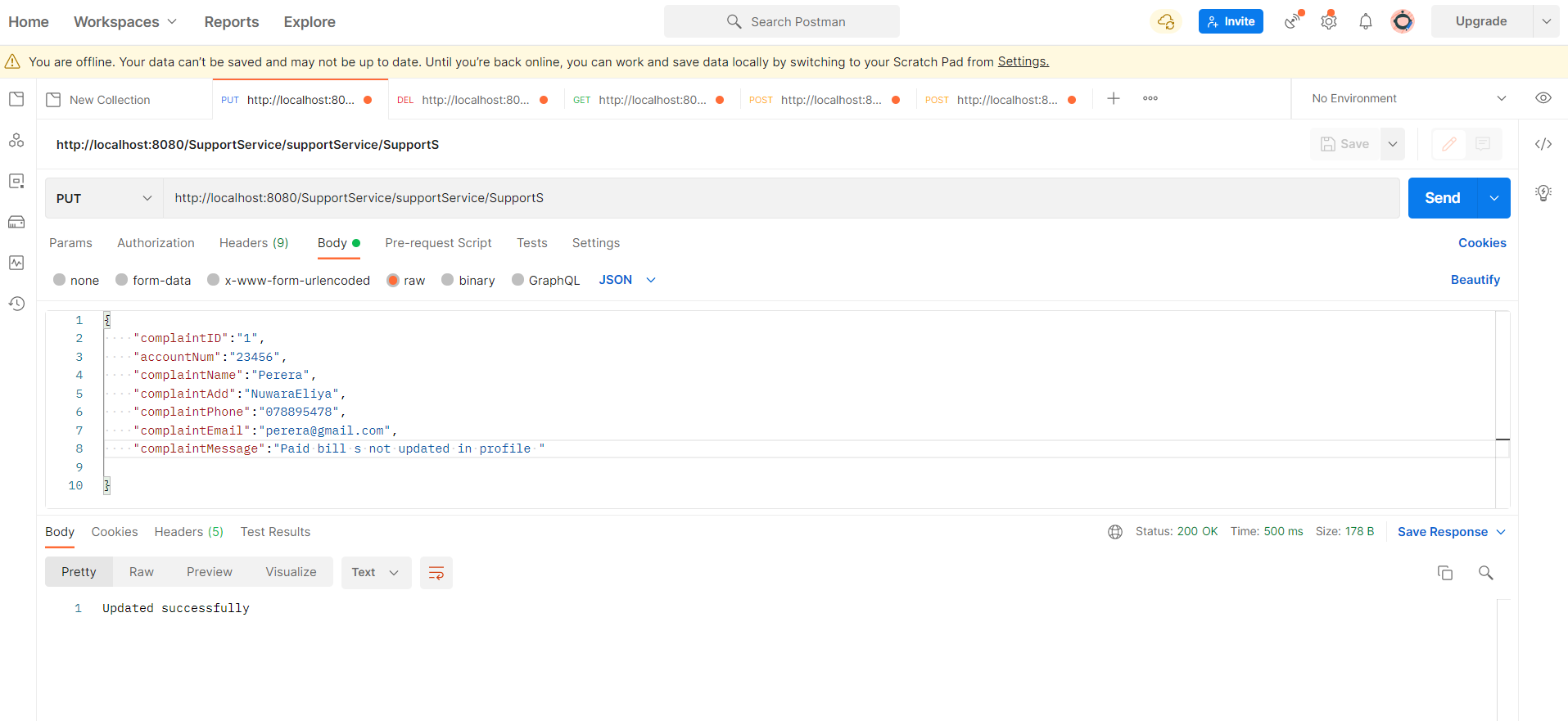
1. Add Complaints



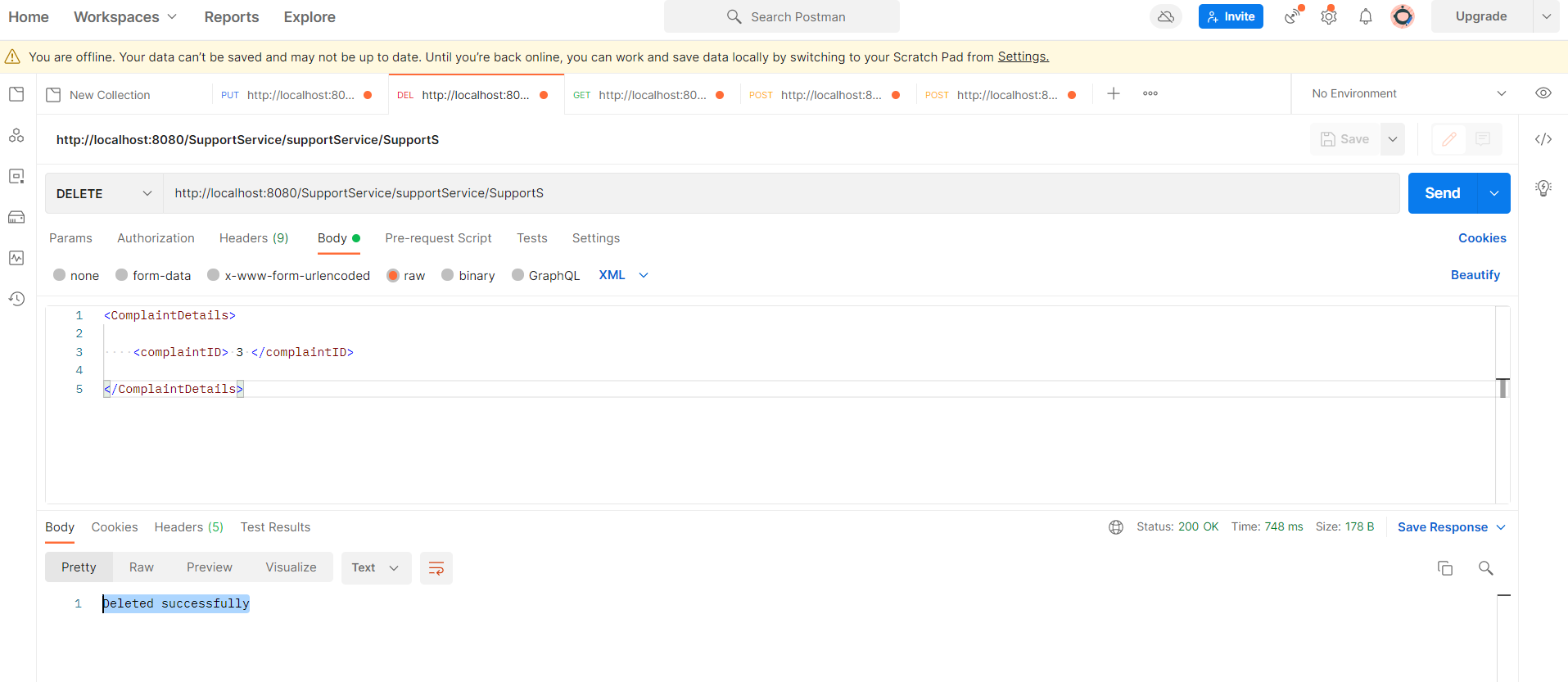
1. View Complaints



1. Update Complaint



1. Delete Complaint



#### Graphical user interface, text, application Description automatically generatedCode Quality Check Sonar Lint Results

### Assumptions

* Admin has the responsibility to reply and update the complaints.
* Consumer can add any number of complaints using same name.

# System’s Integration Details

## Tools Used, Testing Methodology and Results & API Documentation

* + The following tools were used to develop the project.
    - Dependency Management Tool: Maven
    - Testing Tool: Postman
    - Version Control System: Git
    - IDE: eclipse
    - Programming Language: Jersey framework (JAX-RS)
    - Programming Language: Java
    - Database: phpMyAdmin (MySQL)
    - Server: Apache Tomcat Server
  + For testing purpose postman was used.
  + For integration GitHub was used.

## The Architecture used to Design the System

* + The high-level architecture diagram was used to design the overall architecture of the system.
  + Use case diagram was used to identify the use cases.
  + ER diagram to identify the tables of the database.
  + Activity diagram and flow charts to identify the flow of the system.
  + Class diagram to identify the classes for the implementation.

# References

* JAX-RS Documentation

<https://howtodoinjava.com/>

* SE Methodologies

<https://acodez.in/12-best-software-development-methodologies-pros-cons/>

* Maven Documentation

<https://maven.apache.org/guides/>

* Java Development

<http://www.java2novice.com/>