

**Sri Lanka Institute of Information Technology**

**ElectroGrid**

**Project Report**

**Programming Applications and Frameworks project 2022**

**Project ID: 138**

Submitted by:

1. IT20201364 - Thathsarani R.P.H.S.R

2. IT20165352 - Peiris M.I.M

3. IT20187828 - Isurika M.D.A.

4. IT20157500 - Kandanaarachchi H.L.D.S

5. IT20212940 - A.K.F Hasna

6. IT20202354 – M.N Aaisha

April 25, 2022

**Table of Contents**

Table of Contents

[1. Member Details and Workload Distribution 3](#_Toc101773590)

[2. Link to the VCS and Commit Log 4](#_Toc101773591)

[3. SE Methodology 4](#_Toc101773592)

[4. Gantt Chart 5](#_Toc101773593)

[5. Requirements 5](#_Toc101773594)

[1) Stakeholder analysis (onion diagram) 5](#_Toc101773595)

[2) Requirement Analysis. 5](#_Toc101773596)

[3) Requirements Modelling 7](#_Toc101773597)

[6. System Overall Design 7](#_Toc101773598)

[1) Overall Architecture 7](#_Toc101773599)

[2) Activity Diagram 7](#_Toc101773600)

[3) Entity Relationship Diagram 7](#_Toc101773601)

[7. Individual Sections 8](#_Toc101773602)

[1) Team Member 1 – Thathsarani R.P.H.S.R / IT20201364 8](#_Toc101773603)

[2) Team Member 2 – Peiris M.I.M / IT20165352 12](#_Toc101773604)

[3) Team Member 3 – Isurika M.D.A / IT20187828 15](#_Toc101773605)

[4) Team Member 4 - Kandanaarachchi H.L.D.S/ IT20157500 20](#_Toc101773606)

[5) Team Member 5 – Hasna A.K.F / IT20212940 22](#_Toc101773607)

[6) Team Member 6 – M.N Aaisha / IT20202354 27](#_Toc101773608)

[8. System Integration Details (techniques used, how was it tested, etc.). 31](#_Toc101773609)

[9. References 31](#_Toc101773610)

[10. Appendix 32](#_Toc101773611)

[1. Group Diagrams 32](#_Toc101773612)

[Figure 1.1 – Onion Diagram 32](#_Toc101773613)

[Figure 1.2 – Use Case Diagram 33](#_Toc101773614)

[Figure 1.3 – Overall Architecture 34](#_Toc101773615)

[Figure 1.4 – ER Diagram 34](#_Toc101773616)

[Figure 1.5 – Activity Diagram 35](#_Toc101773617)

[2. Team Member 1 – Thathsarani R.P.H.S.R / IT20201364 35](#_Toc101773618)

[3. Team Member 2 – Peiris M.I.M / IT20165352 37](#_Toc101773619)

[4. Team Member 3 - Isurika M.D.A / IT20187828 39](#_Toc101773621)

[5. Team Member 4 - Kandanaarachchi H.L.D.S/ IT20157500 41](#_Toc101773622)

[6. Team Member 5 - A.K.F Hasna/ IT20212940 43](#_Toc101773623)

[7. Team Member 6 - M.N Aaisha/ IT20202354 45](#_Toc101773624)

# Member Details and Workload Distribution

|  |  |  |  |
| --- | --- | --- | --- |
| Registration No | Student Name | Web Service | Description |
| IT20201364 | Thathsarani R.P.H.S.R | E-Bill Handling | * Create E-Bill * Retrieve E-Bill / E-Bills * Update E-Bill * Delete E-Bill * Send E-Bill |
| IT20165352 | Peiris M.I.M | Device Handling | * Add a device. * Remove a device. * Update device information. * Calculate power consumption. * View power consumption. |
| IT20187828 | Isurika M.D.A. | Payment Handling | * Add the payment for calculated bill amount. * View all the transaction history. * Filter specific transaction and View. * Add card details. * Update card details. * Remove card details. |
| IT20157500 | Kandanaarachchi H.L.D.S | User Handling | * Insert User * Retrieve Users/User * Update User Details * Delete User |
| IT20212940 | A.K.F Hasna | Interrupt Handling | * Insert Interrupt Notice * View all Interrupt Notices * Retrieve details of a particular Interrupt Notice * Update interrupt details * Delete old or cancelled Interrupt Notices |
| IT20202354 | M.N Aaisha | Complaint Handling | * Make new complaint * Update complaint details * View complaint details * Delete old/sorted complaints * Admin view all existing complains. |

# Link to the VCS and Commit Log

**Link:** [**https://github.com/RuwanaraT/ElectroGrid-EG-GID\_138.git**](https://github.com/RuwanaraT/ElectroGrid-EG-GID_138.git)

**Commit Log:**

**Graphical user interface, text, application, email

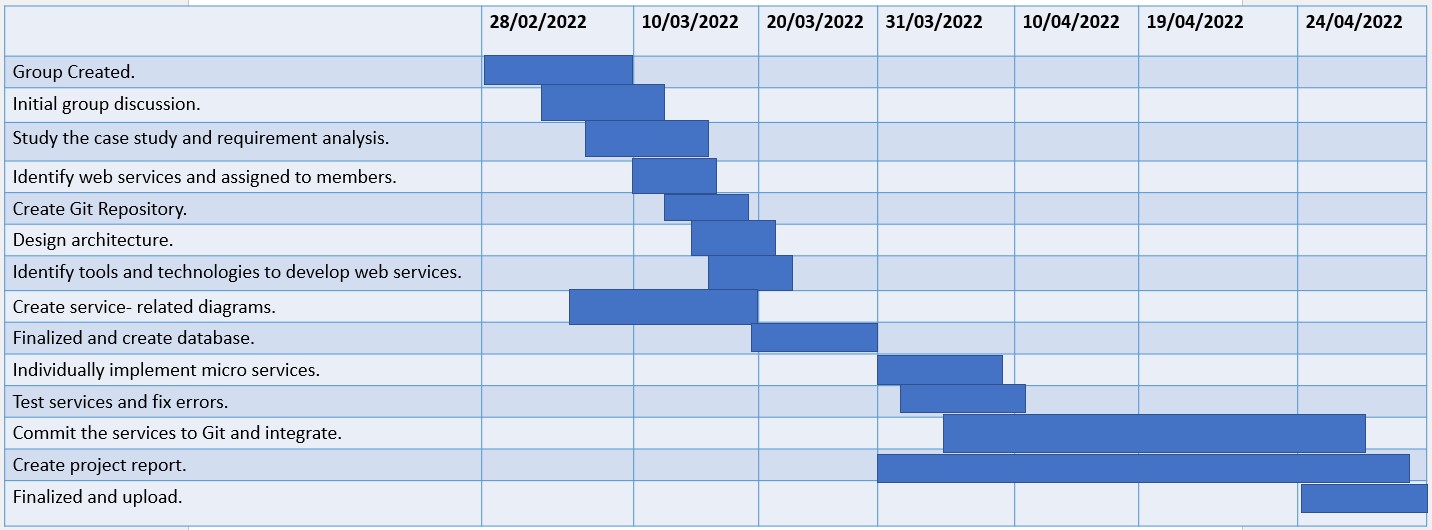
Description automatically generated**

Latest screenshot of the commit log is attached above and please access the repository through the mentioned link to view the complete commit log.

# SE Methodology

The methodology used to develop ElectroGrid(EG) system was one of the agile methods called scrum which is a management and control process that cuts through complexity to focus on building software that meets business needs. Even though the requirements did not change with time, the idea derived from the given requirements surely changed with time. Therefore, using scrum methodology, facilitates us to evolve those obstacles. In our project we have planning the API details in server-side application. In each web service we could invent ideas and we integrate as a project.

# Gantt Chart



# Requirements

## Stakeholder analysis (onion diagram)

The following onion diagram illustrated intercommunication among parts of a process. As shown elements in each circle depends on the elements in the smaller rings. The ElectroGrid(EG) system is shown in the smallest circle. This diagram indicates truly clear and precise representation of the system and gives the consumer an accurate idea of the system and its functionalities. .The onion diagram is attached under appendix (Group diagrams – figure 1.1)

## Requirement Analysis.

|  |  |  |  |
| --- | --- | --- | --- |
| Micro-Service | Functional | Non-Functional | Technical |
| E-Bill Handling | Create E-Bill  Retrieve E-Bills  Retrieve single E-Bill  Update E-Bill  Delete E-Bill  Send E-Bill | Availability  Reliability  Scalability  Performance  Usability  Data Integrity  Backup | Create E-Bill for the current month  Retrieve and display all the existing E-Bills  Retrieve and display specific E-Bill by passing billID  Update E-Bill/s as required  Delete previous month E-Bill, after deleting that entry is included to another table.  Send E-Bill to the customer |
| Device handling | Add a new device to user’s device lists.  View each user owns device list.  Update device information previously saved.  Delete device from device list that are no longer need for calculate power consumption. | Usability  Security  Portability  Compatibility  Reliability  Availability  Maintainability | Each user who are already registered in the system, owns a self-calculator which provides daily power consumption and monthly power consumption of each saved device. According to the user’s devices those are used in their houses , user can add, remove, or update devices. Device service is connected to E-Bill service which can calculate cost per each device’s monthly power consumption and calculate monthly total amount of power usage. |
| Payment handling | Add the payment for calculated bill amount.  View all the transaction history.  Filter specific transaction and View.  Add card details.  View card details.  Update card details.  Remove card details. | Availability  Reliability  Performance  Scalability  Capacity  Data Integrity  Usability  backup | Users make payments according to the e-bill generated by the system.  User can view all the payment history with the date it has done.  Administrator can view all the payments made by users.  User can add card details to the system and can save if they want.  This card details can be viewed, updated, or removed at any time. |
| User Handling | Add user.  Update User.  Delete User.  Retrieve Users. | Availability  Reliability  Performance  Capacity  Data Integrity  Usability | Users can register to the system as new users.  Retrieve and display all the existing Users.  Retrieve and display specific User by passing username.  User can update their details as required.  User can remove their details from the system. |
| Interrupt Handling | Add a new Interrupt Notice.  View Interrupt Notices.  View Details of a particular Interrupt Notice.  Update Interrupt Notice Details.  Delete old or Cancelled Interrupt Notices. | Availability  Reliability  Performance  Scalability  Capacity  Data Integrity  Usability | Administrator can add, update, and delete Interrupt Notices.  Interrupt Notice scheduled date, time region and reason can be updated.  Cancelled or old Interrupt Notices can be deleted  All the Interrupt Notices can be viewed.  The details of a particular Interrupt Notice can be viewed using the ID. |
| Complaint Handling | Create new complaint  Update details of a particular complaint  View details of a particular complaint  View all complains  Delete old/sorted complains | Availability  Reliability  Performance  Scalability  Capacity  Data Integrity  Usability  Security | Customer can make a new complaint when required.  Customer can update all details except status date and status.  Both customer and admin can delete complaints if necessary.  Admin can update only the status of the complaint.  Customers and admins can view complaints. |

## Requirements Modelling

**Use case diagram**

The following Use Case Diagram illustrated who are the actors of the ElectroGrid(EG) system and what sort of operations they can perform within the system scope. The use case diagram is attached under appendix (Group diagrams – figure 1.2)

The main actors of the ElectroGrid system in this use case diagram are customer, meter reader and administrator.

# System Overall Design

## Overall Architecture

The following Overall Architecture give the overview of the ElectroGrid(EG) system by representing dedicated micro service and interconnection between them .The overall architecture is attached under appendix (Group diagrams – figure 1.3)

## Activity Diagram

The following Activity Diagram describes the activity flow of the ElectroGrid(EG) system. The activity diagram is attached under appendix (Group diagrams – figure 1.5)

## Entity Relationship Diagram

The following Entity Relationship(ER) Diagram implies the database tables and connections between them. The Entity Relationship diagram is attached under appendix (Group diagrams – figure 1.4)

# Individual Sections

## Team Member 1 – Thathsarani R.P.H.S.R / IT20201364

* + Individual Service – E-Bill Handling

**Service Design**

* 1. **API Design of the Service**
  + **POST – Create E-Bill**

**URL:-** <http://localhost:8080/EBill-Handling/EBillServices/EBills>

A screenshot of a computer

Description automatically generated

* + Insert Electricity Account Number, Customer Name, Address, Billing Date and Bill Amount to create the E-Bill.
  + **GET – Retrieve E-Bills**

**URL:-** <http://localhost:8080/EBill-Handling/EBillServices/EBills>

Graphical user interface, text, application, email

Description automatically generated

* + Retrieve (Display) all the existing E-Bills from the system.
  + **GET – Retrieve E-Bill**

**URL:-** <http://localhost:8080/EBill-Handling/EBillServices/EBills/2>

Graphical user interface, text, application, email

Description automatically generated

* + Retrieve (Display) single E-Bill from the system by passing the bill id.
  + The above API design is illustrated request of bill id 2.
  + **PUT – Update E-Bill**

**URL:-** <http://localhost:8080/EBill-Handling/EBillServices/EBills>

A screenshot of a computer

Description automatically generated

* + Update or edit some details of the E-Bill as required.
  + The above API design is illustrated update request of bill id 5 from Jason format.
  + **DELETE – DELETE E-Bill**

**URL:-** <http://localhost:8080/EBill-Handling/EBillServices/EBills>

Graphical user interface, text, application, chat or text message, email

Description automatically generated

* + Delete particular E-Bill from the system.
  + The above API design is illustrated delete request of bill id 5 from xml format.
  1. **Internal Logic**
  + **Class Diagram**

Basically, EBill class is covered E-Bill Management service. The above, part of the class diagram is illustrated attributes and methods of the EBill class. . Class diagram is attached under Appendix (IT20201364 – figure 2.1)

* + **Activity Diagrams**

The activity diagram is illustrated steps of preparing a E-Bill. Activity diagram is attached under Appendix (IT20201364– figure2.2 )

* + **Flowchart**

The flowchart is illustrated mechanism of calculating electricity bill. . Flow chart is attached under Appendix (IT20201364– figure 2.3)

* + **Sequence Diagram**

The sequence diagram is illustrated sequence of preparing a E-Bill. Sequence diagram is attached under Appendix (IT20201364– figure 2.4)

* 1. **Service Development and Testing**

1. **Tools**
   * **IDE** - Eclipse
   * **Database** - phpMyAdmin
   * **Back End** - Java, JAX-RS (Jersy) on Tomcat
   * **Dependency Management Tools** - Maven
   * **Testing Tool** - Postman
   * **Code Quality Checking Tool** **-** SonarLint
   * **Version Control** - GitHub
2. **Testing Methodology and Results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Description / Test Steps | Test Input(s) | Expected Output(s) | Actual Output(s) | Result (Pass/Fail) |
| 01 | Insert E-Bill Details | eaNumber  cusName  address  billingDate  amount  (Attributes of the E-Bill  ) | E-Bill Created Successfully | E-Bill Created Successfully | Pass |
| 02 | Display existing E-Bills | URL for the API send GET Request | Display all the E-Bills | Display all the E-Bills | Pass |
| 03 | Display a single E-Bill | URL for the API send GET request along with the billID | Display relevent E-Bill according to the given billID | Display relevent E-Bill according to the given billID | Pass |
| 04 | Update E-Bill Details | Attributes to be updated along with billID | E-Bill Updated Successfully. | E-Bill Updated Successfully. | Pass |
| 05 | Delete E-Bill Details | billID of the E-Bill to be deleted | E-Bill Deleted Successfully | E-Bill Deleted Successfully | Pass |

* 1. **Assumptions**
  + Design the APIs , assuming that system is maintaining current month E-Bill operations.

## Team Member 2 – Peiris M.I.M / IT20165352

* + Individual Service – Device Handling

**Service Design**

* 1. **API Design of the Service**
  + **GET - Retrieve device list saved by a user**

**URL:** [**http://localhost:8010/Device-Handling/DeviceServices/EDevice/**](http://localhost:8010/Device-Handling/DeviceServices/EDevice/)

Table

Description automatically generated

* + Retrieve all the devices saved by one user. Daily power consumption and monthly power consumption for each device is also displayed in front of each device.
  + **POST - Add device by one user.**

**URL :** [**http://localhost:8010/Device-Handling/DeviceServices/EDevice/**](http://localhost:8010/Device-Handling/DeviceServices/EDevice/)

**Graphical user interface, application, website

Description automatically generated**

* + Add a device to the device list, device ID is auto generated.
  + **PUT – Update device information.**

**URL :** [**http://localhost:8010/Device-Handling/DeviceServices/EDevice/**](http://localhost:8010/Device-Handling/DeviceServices/EDevice/)

**Graphical user interface, text, application, email, Teams

Description automatically generated**

* + Update inserted device information.
  + Above API design depicts the PUT request of device id 20 using Jason format.
  + **DELETE- Delete a device from the device list.**

**URL :** [**http://localhost:8010/Device-Handling/DeviceServices/EDevice/**](http://localhost:8010/Device-Handling/DeviceServices/EDevice/)

**Graphical user interface, text, application, email

Description automatically generated**

* + Delete one or more devices from the device list.
  + Above API design depicts the DELETE request of device id 19 using XML format.

**II. Internal Logic**

* **Class Diagram**

Device class diagram depicts the attributes and methods owns by device class to build the operations to develop the Electro-Grid system . Class diagram is attached under Appendix (IT20165352 – figure 3.1 )

* **Activity Diagrams**

The activity diagrams depict the flow of each operation is reflected accordingly. Activity diagram is attached under Appendix (IT20165352– figure 3.2,3.3,3.4 )

* **Flow Chart**

The flow chart depicts the flow of calculating power consumption. Flow chart is attached under Appendix (IT20165352– figure 3.5)

* **Sequence Diagram**

The sequence diagram depicts the sequence of device management of the system. Sequence diagram is attached under Appendix (IT20165352– figure 3.6)

1. **Service Development and Testing**
2. **Tools**
   * **IDE** - Eclipse
   * **Database** - phpMyAdmin
   * **Back End** - Java, JAX-RS (Jersy) on Tomcat
   * **Dependency Management Tools** - Maven
   * **Testing Tool** - Postman
   * **Code Quality Checking Tool** **-** SonarLint
   * **Version Control** - GitHub
3. **Testing Methodology and Results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Description / Test Steps | Test Input(s) | Expected Output(s) | Actual Output(s) | Result (Pass/Fail) |
| 01 | Insert a device. | Attributes for device. | Display message as “Inserted Successfully.” | Display message as “Inserted Successfully.” | Pass |
| 02 | Update device details for a selected device. | Attributes to be updated along with device. | Display Message as “Updated Successfully.” | Display Message as “Updated Successfully.” | Pass |
| 03 | Delete a device. | Device id of the device to be deleted. | Display Message as “Deleted Successfully.” | Display Message as “Deleted Successfully.” | Pass |
| 04 | View device list. | URL for the API Send Get request | Display device details. | Device details displayed. | Pass. |

1. **Assumptions**

* Assume, above mentioned APIs are designed for a one customer.
* Assume, a customer already registered to manage devices.
* Assume, device handling service, monitor the power consumption of users of the system.

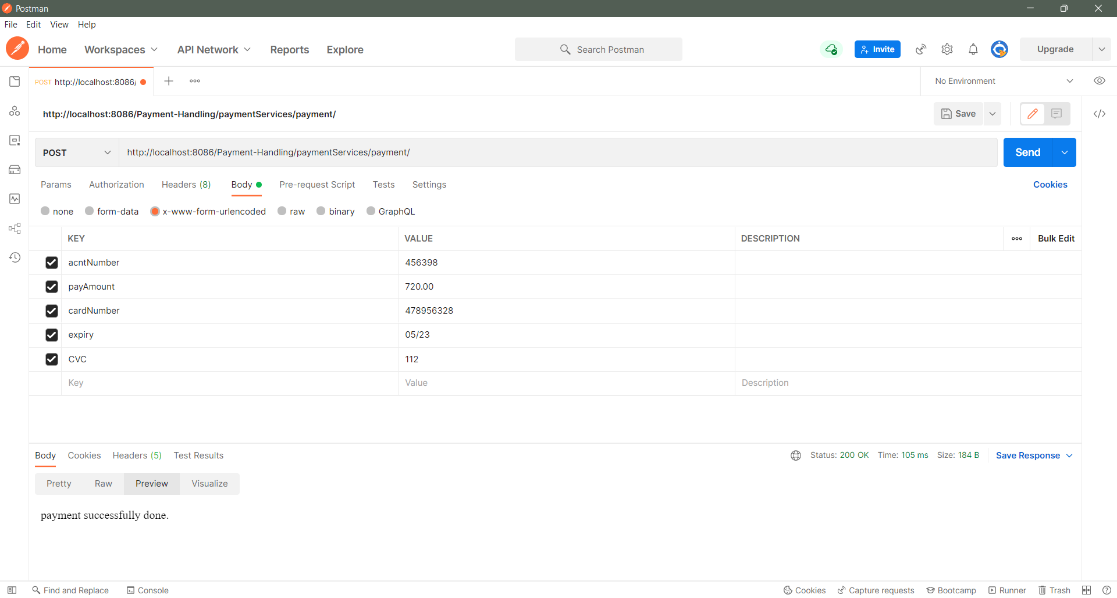
## Team Member 3 – Isurika M.D.A / IT20187828

* + Individual Service – Payment-Handling

**Service Design**

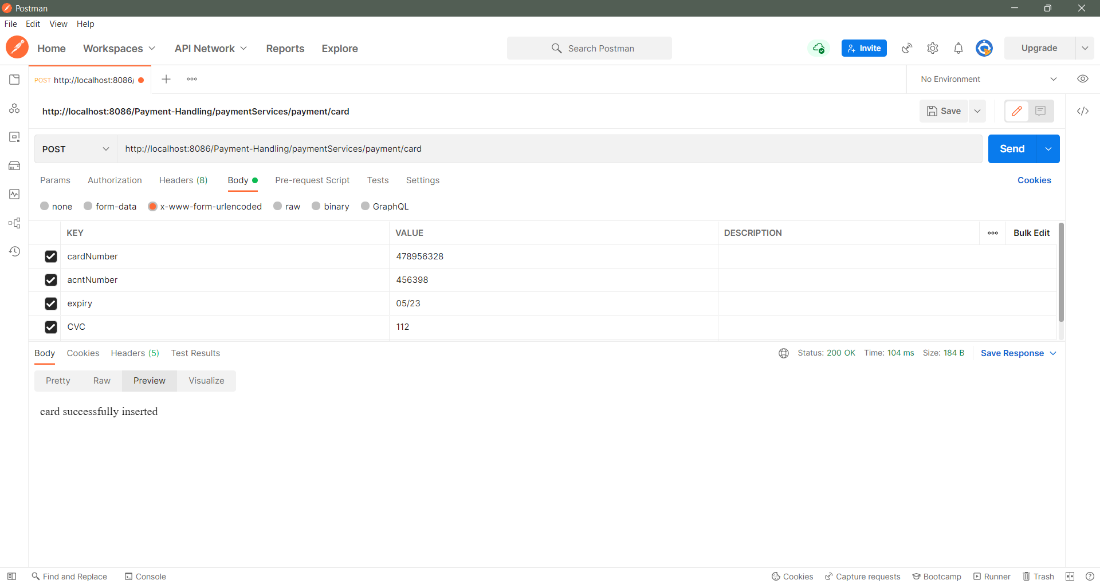
* 1. **API Design of the Service**
  + **POST – add the payment**

**URL:-** [**http://localhost:8086/Payment-Handling/paymentServices/payment/**](http://localhost:8086/Payment-Handling/paymentServices/payment/)



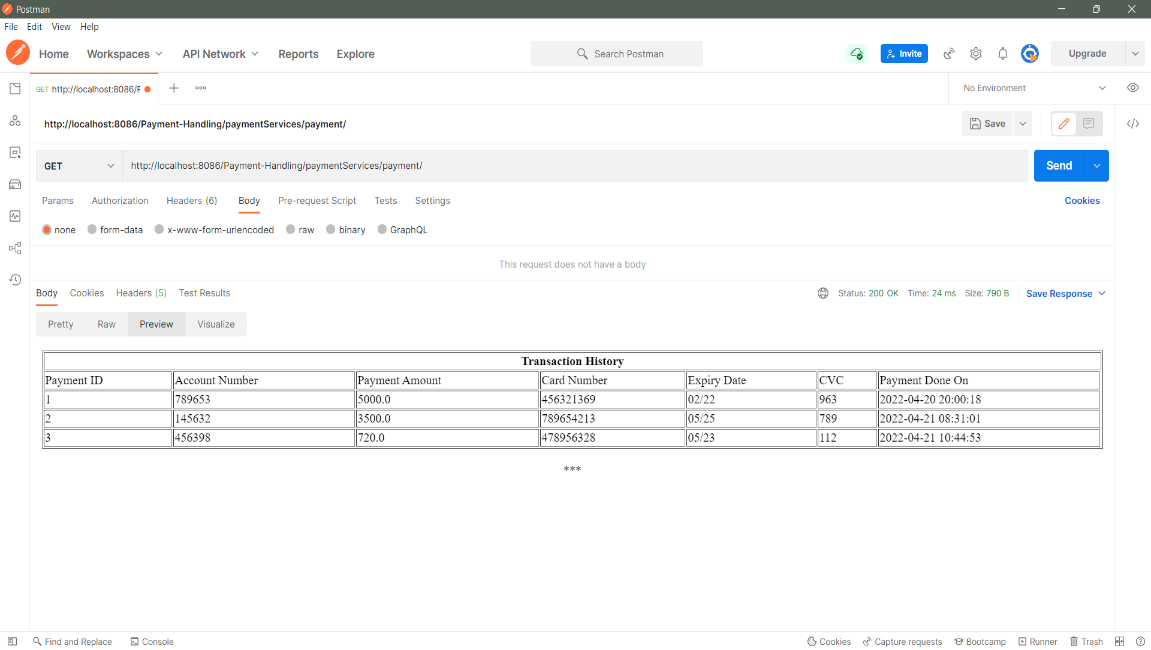
* + Insert account number, card number, expiry date and CVC to do the payment.
  + **POST – save card details.**

**URL: -** [**http://localhost:8086/Payment-Handling/paymentServices/payment/card**](http://localhost:8086/Payment-Handling/paymentServices/payment/card)



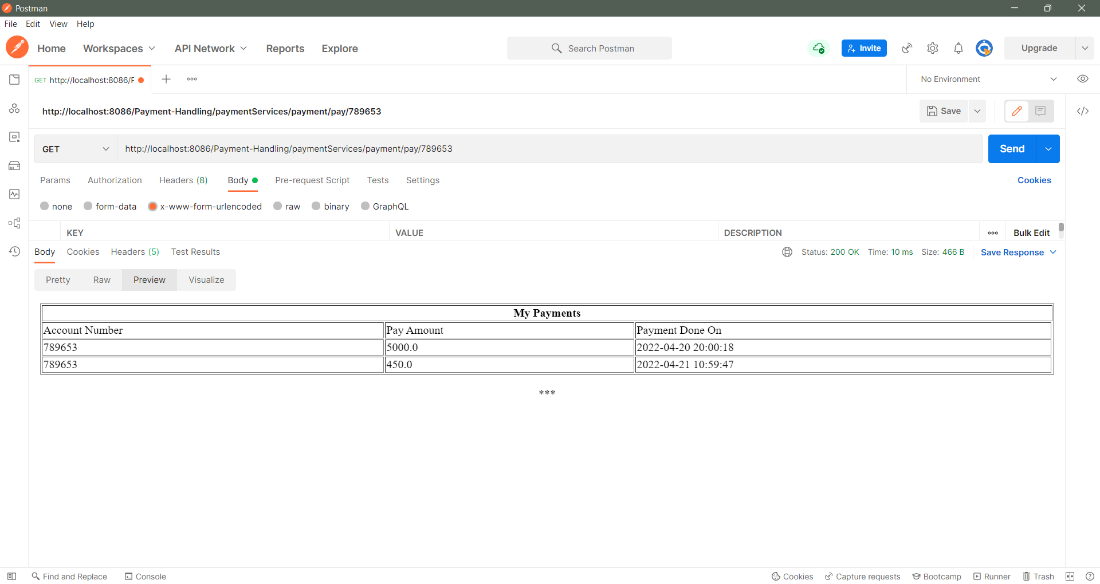
* + Insert card number, account number, expiry date and CVC to save card details in the system.
  + **GET – get all the payment history**

**URL: -** [**http://localhost:8086/Payment-Handling/paymentServices/payment/**](http://localhost:8086/Payment-Handling/paymentServices/payment/)



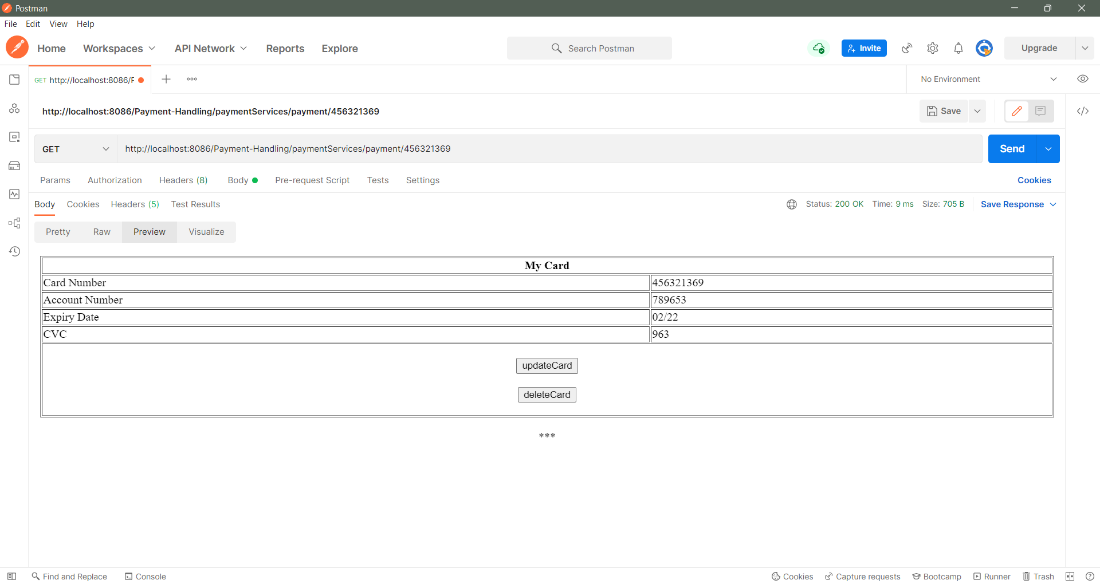
* + Retrieve all the existing payments in the system.
  + **GET – get all the payment history related to one user account**

**URL:**[**http://localhost:8086/PaymentHandling/paymentServices/payment/pay/789653**](http://localhost:8086/PaymentHandling/paymentServices/payment/pay/789653)



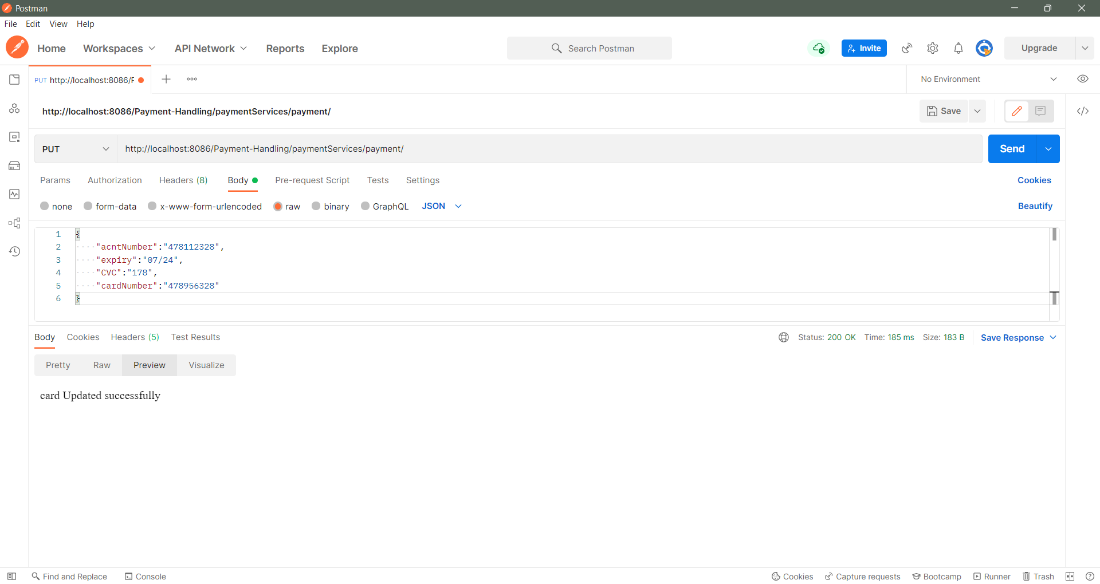
* + Retrieve only specific payment details from the system by passing the account number as parameter value.
  + The above API design illustrates the transaction details of a user whose account number is requested as ‘789653’.
  + **GET – get card details**

**URL:**[**http://localhost:8086/PaymentHandling/paymentServices/payment/456321369**](http://localhost:8086/PaymentHandling/paymentServices/payment/456321369)



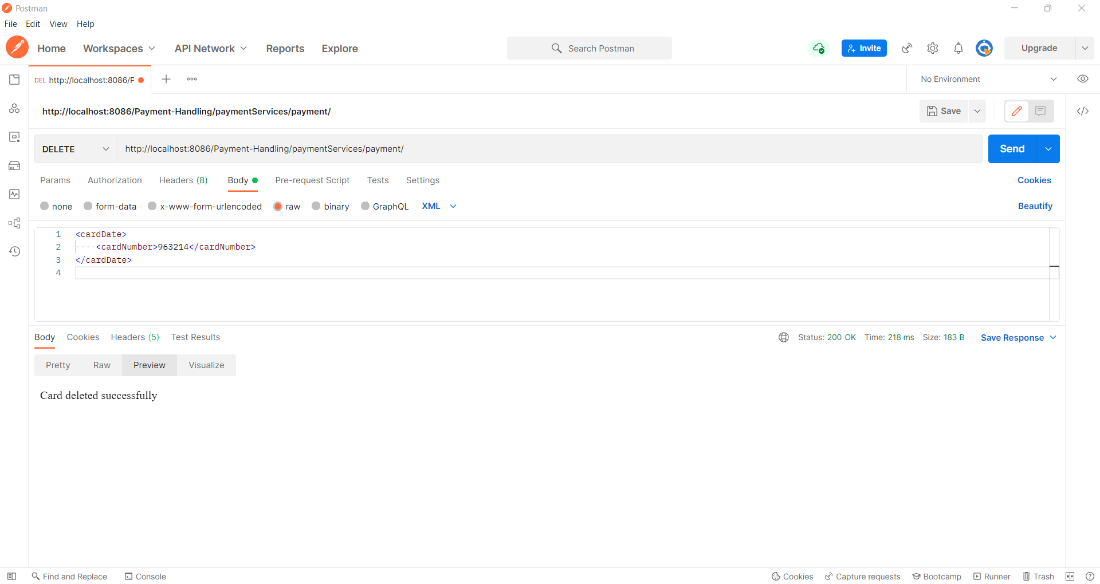
* + Retrieve all card details from the system by passing the card number as parameter value.
  + The above API design illustrates the card details of requested card number ‘456321369’.
  + **PUT – update card details**

**URL: -** [**http://localhost:8086/Payment-Handling/paymentServices/payment/**](http://localhost:8086/Payment-Handling/paymentServices/payment/)



* + Update some details of the card as required.
  + The above API design illustrates update card details of card number ‘478956328’ from Jason format.
  + **DELETE – delete card details**

**URL: -** [**http://localhost:8086/Payment-Handling/paymentServices/payment/**](http://localhost:8086/Payment-Handling/paymentServices/payment/)



* + Delete card details from the system.
  + The above API design illustrates deletion of the card ‘963214’ from xml format.
  1. **Internal Logic**
* **Class Diagram**

The part of the class diagram is illustrated attributes and methods of the pay class and card class. These two classes used to manage payment handling service. Class diagram is attached under Appendix (IT20187828 – figure 4.1 )

* **Activity Diagram**

The activity diagram is illustrated steps of preparing a payment and handling card details. Activity diagram is attached under Appendix (IT20187828– figure 4.2)

* **Flow Chart**

The flowchart is illustrated flow of steps of payment handling service. Flow chart is attached under Appendix (IT20187828– figure 4.3)

* **Sequence Diagram**

The sequence diagram is illustrated sequence of steps of preparing a payment. Sequence diagram is attached under Appendix (IT20187828– figure 4.4)

* 1. **Service Development and Testing**
  2. **Tools**
* **IDE** - Eclipse
* **Database** - phpMyAdmin
* **Back End** - Java
* **Dependency Management Tools**  - Maven
* **Testing Tool** - Postman
* **Code Quality Checking Tool** **-** SonarLint
* **Version Control** - GitHub
  1. **Testing Methodology and Results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Description / Test Steps | Test Input(s) | Expected Output(s) | Actual Output(s) | Result (Pass/Fail) |
| 01 | Create payment | payAmount  AcntNumber  billID  cardNumber  expiry  CVC | Add the Payment. | Add the payment. | Pass |
| 02 | View specific payment details. | URL for the API send GET Request with acntNumber. | Display payments according to the given acntNumber. | Display payments according to the given acntNumber. | Pass |
| 03 | View all the payments. | URL for the API send GET request. | Display all the payments. | Display all the payments. | Pass |
| 04 | View all the card details. | URL for the API send GET Request with cardNumber. | Display card details according to the given cardNumber. | Display card details according to the given cardNumber. | Pass |
| 05 | Update card Details | Attributes to be updated along with acntNumber. | Card details Updated Successfully. | Card details Updated Successfully. | Pass |
| 06 | Delete card Details | Card number to be deleted. | Card details Deleted Successfully. | Card details Deleted Successfully. | Pass |

## Team Member 4 - Kandanaarachchi H.L.D.S/ IT20157500

* Individual Service – User Handling

1. **API Design of the Service**

* **GET – Retrieve all Users in the system**

Graphical user interface, text, application, email

Description automatically generated

* **GET 1 – Retrieve particular User**

Graphical user interface, text, application, email

Description automatically generated

* **POST – Insert a User**

Graphical user interface, text, application, email

Description automatically generated

* **PUT – Update User**

Graphical user interface, text, application, email

Description automatically generated

* **DELETE – Delete User**

Graphical user interface, text, application, email

Description automatically generated

1. **Internal Logic**

* **Class Diagram**

User class is covered User Management service. The part of the class diagram is illustrated attributes and methods of the User class. Class diagram is attached under Appendix (IT20157500 – figure 5.1 )

* **Activity Diagrams**

The activity diagram is illustrated activities of User. Activity diagram is attached under Appendix (IT20157500– figure 5.2 )

* **Flow Chart**

The flowchart is illustrated separate steps of User Handling. Flow chart is attached under Appendix (IT20157500– figure 5.3 )

* **Other Diagrams**

The sequence diagram is illustrated sequence of preparing a User Handling. Sequence diagram is attached under Appendix (IT20157500– figure 5.4 )

1. **Service Development and Testing**
   1. **Tools**

* **IDE** - Eclipse
* **Database** - phpMyAdmin
* **Back End** - Java
* **Dependency Management Tools**  - Maven
* **Testing Tool** - Postman
* **Code Quality Checking Tool** **-** SonarLint
* **Version Control** - GitHub

* 1. **Testing Methodology and Results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Description / Test Steps | Test Input(s) | Expected Output(s) | Actual Output(s) | Result (Pass/Fail) |
| 01 | Insert User Details | acntNumber  fullName  email  NIC  address  mobileNumber  landpNumber  userName  password  (Attributes of the User  ) | User Created Successfully. | User Created Successfully. | Pass |
| 02 | Display existing Users | URL for the API send GET Request. | Display all the Users. | Display all the Users. | Pass |
| 03 | Display a single User | URL for the API send GET request along with the userName. | Display relevant User according to the given userName. | Display relevant User according to the given userName. | Pass |
| 04 | Update User Details | Attributes to be updated along with userName. | User Updated Successfully. | User Updated Successfully. | Pass |
| 05 | Delete User Details | UserName of the User to be deleted. | User Deleted Successfully. | User Deleted Successfully. | Pass |

## Team Member 5 – Hasna A.K.F / IT20212940

* Individual Service – Interrupt Handling

**Service Design**

* 1. **API Design of the Service**
  + **POST – Create Interrupt Notice**

**URL :-** [localhost:8086/Interrupt-Handling/interruptService/Interrupts](http://localhost:8086/Interrupt-Handling/interruptService/Interrupts)

Graphical user interface, text, application, email

Description automatically generated

* + Insert Interrupt Code, Date, Duration, start time, End time, Region, and Reason in order to create an Interrupt Notice.
  + **GET – Retrieve All Interrupt Notices**

**URL :-** [localhost:8086/Interrupt-Handling/interruptService/Interrupts](http://localhost:8086/Interrupt-Handling/interruptService/Interrupts)

Graphical user interface, application

Description automatically generated

* + Retrieve (Display) all the existing Interrupt Notices from the system.
  + **GET – Retrieve an Interrupt Notice**

**URL :-** [localhost:8086/Interrupt-Handling/interruptService/Interrupts/2](http://localhost:8086/Interrupt-Handling/interruptService/Interrupts)

Graphical user interface, text

Description automatically generated

* + Retrieve (Display) single Interrupt Notice from the system by passing the interrupt id.
  + The above API design is illustrated to request for Interrupt id 2.
  + **PUT – Update Interrupt Notice**

**URL :-** [localhost:8086/Interrupt-Handling/interruptService/Interrupts](http://localhost:8086/Interrupt-Handling/interruptService/Interrupts)

Graphical user interface, text, application, email

Description automatically generated

* + Update or edit some details of the Interrupt Notice as required.
  + The above API design is illustrated to update the Interrupt Notice with the id 6 from Jason format.
  + **DELETE – DELETE Interrupt Notice**

**URL :-** [localhost:8086/Interrupt-Handling/interruptService/Interrupts](http://localhost:8086/Interrupt-Handling/interruptService/Interrupts)

Graphical user interface, text, application, email

Description automatically generated

* + Delete a particular Interrupt Notice from the system.
  + The above API design is illustrated to delete Interrupt Notice with the id 9 from xml format.
  1. **Internal Logic**
* **Class Diagram**

Basically, Interrupt class covers Interrupt Notice Management service. The part of the class diagram illustrates the attributes and methods of the interrupt class. Class diagram is attached under Appendix (IT20212940 – figure 6.1 )

* **Activity Diagrams**

The insert, update, delete, retrieve activity diagrams depict the flow of each operation is reflected accordingly. Activity diagram is attached under Appendix (IT20212940– figure 6.2,6.3,6.4,6.5 )

* **Flow Chart**

The flowchart illustrates the mechanism of handling interrupt notices. Flow chart is attached under Appendix (IT20212940– figure 6.6)

* **Sequence Diagrams**

The sequence diagram illustrates sequence handling interrupt notices. Sequence diagram is attached under Appendix (IT20212940– figure6.7 )

* 1. **Service Development and Testing**
  2. **Tools**
* **IDE** - Eclipse
* **Database** - phpMyAdmin
* **Back End** - Java
* **Dependency Management Tools**  - Maven
* **Testing Tool** - Postman
* **Code Quality Checking Tool** **-** SonarLint
* **Version Control** - GitHub

* 1. **Testing Methodology and Results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Description / Test Steps | Test Input(s) | Expected Output(s) | Actual Output(s) | Result (Pass/Fail) |
| 01 | Insert Interrupt Notice Details | Valid interruptCode, valid Date, valid duration, valid start\_time, valid End\_time, valid region, valid reason and valid AdminID | Inserted Successfully | Inserted Successfully | Pass |
| 02 | Insert Interrupt Notice  Details | Valid interruptCode, invalid Date format, valid duration, valid start\_time, valid End\_time, valid region, valid reason and valid AdminID | Error while inserting | Error while inserting | Pass |
| 03 | Insert Interrupt Notice Details | Valid interruptCode, valid Date, invalid duration, valid start\_time, valid End\_time, valid region, valid reason and valid AdminID | Error while inserting | Error while inserting | Pass |
| 04 | Insert Interrupt Notice Details | Valid interruptCode, valid Date, valid duration, invalid start\_time format, valid End\_time, valid region, valid reason and valid AdminID | Error while inserting | Error while inserting | Pass |
| 05 | Insert Interrupt Notice Details | Valid interruptCode, valid Date, valid duration, valid start\_time format, valid End\_time, valid region, valid reason and unavailable AdminID | Error while inserting | Error while inserting | Pass |
| 06 | Display existing Interrupt Notices | URL for the API send GET Request | Display all the Interrupt Notices | Display all the Interrupt Notices | Pass |
| 07 | Display Interrupt Notices Based on Region | URL for the API send GET request along with the region | Display relevant interrupt Notices according to the given region | Display relevant interrupt Notices according to the given region | Pass |
| 08 | Display Interrupt Notices Based on Region | URL for the API send GET request along with the unavailable region | Error while Displaying interrupt based on region | Error while Displaying interrupt based on region | Pass |
| 07 | Update Interrupt Notice Details | Attributes to be updated along with the interrupt ID | Updated Successfully. | Updated Successfully. | Pass |
| 08 | Update Interrupt Notice Details | Valid interruptCode, invalid Date format, invalid duration, invalid start\_time format, invalid End\_time format, valid region, valid reason and unavailable AdminID along with the interrupt ID | Error while Updating interrupt notice | Error while Updating interrupt notice | pass |
| 09 | Update Interrupt Notice Details | Attributes to be updated along with unavailable interrupt ID | Error while Updating interrupt notice | Error while Updating interrupt notice | Pass |
| 10 | Delete Interrupt Notice Details | InterruptID of the Interrupt Notice to be deleted | Deleted Successfully | Deleted Successfully | Pass |
| 11 | Delete Interrupt Notice Details | Unavailable InterruptID of the Interrupt Notice to be deleted | Error while Deleting interrupt notice | Error while Deleting interrupt notice | Pass |

## Team Member 6 – M.N Aaisha / IT20202354

* Individual Service – Complaint Handling

**Service Design**

1. **API Design of the Service** 
   * **POST – Create New Complaint**

**URL :-** <http://localhost:8086/Complaint-Handling/complaintService/Complaint>

Graphical user interface, text, application, email

Description automatically generated

* Insert details such as account number, name, contact number, Email, type of complaint and details regarding the complaint to create a new complaint .
* Status is set to the default value of ‘Unresolved.’
  + **GET – Retrieve All Complaintss**

**URL :-** <http://localhost:8086/Complaint-Handling/complaintService/Complaint>

Graphical user interface, text, application, email

Description automatically generated

* Admin can retrieve details of all complains made by different customers.
  + **GET – Retrieve a particular complaint**

**URL:** http://localhost:8086/ComplaintHandling/complaintService/Complaint/7

Table

Description automatically generated

* Retrieve and display a single complain by passing the complaint ID.
* The above API design is illustrated to request for complaint ID 9.
* **PUT– Update Details of a Complaint**

**URL:** <http://localhost:8086/ComplaintHandling/complaintService/Complaint>

**Graphical user interface, text, application, email

Description automatically generated**

* The above API design is illustrated to update the complaint with the id 9 using Jason format
* **DELETE – DELETE Complaint**

**URL :-** <http://localhost:8086/Complaint-Handling/complaintService/Complaint>

**Graphical user interface, text, application, email

Description automatically generated**

* Delete a particular Complaint from the system.
* The above API design is illustrated to delete Complaint with the id 14 from xml format.

1. **Internal Logic**

* **Class Diagram**

The part of the class diagram illustrates the attributes and methods of the complaint class. Class diagram is attached under Appendix (IT20202354– figure 7.1)

* **Activity Diagrams**

The insert, update, delete, retrieve activity diagrams depict the flow of each operation is reflected accordingly. Activity diagram is attached under Appendix (IT20202354– figure 7.2,7.3,7.4,7.5,7.6,7.7)

* **Flow Chart**

The flowchart illustrates the mechanism of handling Complaints. Flow chart is attached under Appendix (IT20202354– figure 7.8,7.9,7.10)

* **Other Diagrams**

The sequence diagram illustrates sequence handling complaints. Sequence diagram is attached under Appendix (IT20202354– figure 7.11 )

1. **Service Development and Testing**
   1. **Tools**

* **IDE** - Eclipse
* **Database** - phpMyAdmin
* **Back End** - Java
* **Dependency Management Tools**  - Maven
* **Testing Tool** - Postman
* **Code Quality Checking Tool** **-** SonarLint
* **Version Control** - GitHub

* 1. **Testing Methodology and Results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Description / Test Steps | Test Input(s) | Expected Output(s) | Actual Output(s) | Result (Pass/Fail) |
| 01 | Insert new Complaint Details | Valid ComplaintID, valid Contact, valid type, valid details, valid date | Inserted Successfully | Inserted Successfully | Pass |
| 02 | Insert new Complaint  Details | Valid ComplaintID, invalid Contact Number valid type, valid details, valid date | Error while inserting | Error while inserting | Pass |
| 03 | Display existing Complaints | URL for the API send GET Request | Display all the complaints | Display all the Complaints | Pass |
| 04 | Display Complaint belonging to a particular ID | URL for the API send GET request with the ComplaintID | Display relevant Complaint | Display relevant Complaint | Pass |
| 05 | Update Complaint Details | Attributes to be updated along with the Compalint ID | Updated Successfully. | Updated Successfully. | Pass |
| 06 | Update Complaint Details | Valid ComplaintID, invalid Contact, valid type, valid details,valid status along with  complaint  ID | Error while Updating Complaint | Error while Updating Complaint | Pass |
| 07 | Update Complaint Details | Attributes to be updated along with unavailable Complaint ID | Error while Updating Complaint | Error while Updating Complaint | Pass |
| 08 | Delete Complaint | URL of API send DELETE with Complaint ID of the complaint to be deleted | Deleted Successfully | Deleted Successfully | Pass |
| 09 | Delete Complaint | URL of API send DELETE with an unavailable complaint ID | Error while Deleting Complaint | Error while Deleting Complaint | Pass |

# System Integration Details (techniques used, how was it tested, etc.).

The services of the ElectroGrid system were implemented separately following the MVC architecture. Three packages were made for each service to store database connection (util), one to implement the RESTful API services (com) and another package to implement the server model (model).

Separate maven projects were created in the git repository and once the services were implemented by the team members, it was integrated. It was discussed for each team member to use different tomcat ports. The Tomcat ports used by members are mentioned below:

|  |  |
| --- | --- |
| Web Service | Tomcat Port No |
| User | 8080 |
| Device | 8010 |
| E-Bill | 8080 |
| Payment | 8086 |
| Interrupts | 8084 |
| Complaints | 8086 |

Once the system is developed and executed in five separate ports, system was tested through postman. The black box testing method used to test services individually was followed when testing the system. Moreover, to detect code quality and code security of entire project ‘SonarLint’ plugin is used. All the requests under services were tested by passing the URL accordingly (with the allocated port number). A single database was maintained for the entire project.

# References

* [**https://courseweb.sliit.lk/**](https://courseweb.sliit.lk/)
* [**https://stackoverflow.com/**](https://stackoverflow.com/)
* [**https://helpdeskgeek.com/**](https://helpdeskgeek.com/)
* [**https://www.sonarlint.org/?gclid=EAIaIQobChMIvdKcoeOn9wIVVCUrCh2ShAJjEAAYASAAEgKBrfD\_BwE**](https://www.sonarlint.org/?gclid=EAIaIQobChMIvdKcoeOn9wIVVCUrCh2ShAJjEAAYASAAEgKBrfD_BwE)
* [**https://crunchify.com/how-to-build-restful-service-with-java-using-jax-rs-and-jersey/**](https://crunchify.com/how-to-build-restful-service-with-java-using-jax-rs-and-jersey/)
* [**https://learning.postman.com/docs/getting-started/introduction/**](https://learning.postman.com/docs/getting-started/introduction/)

# Appendix

## Group Diagrams

Diagram

Description automatically generated

### Figure 1.1 – Onion Diagram

Diagram, schematic

Description automatically generated

### Figure 1.2 – Use Case Diagram

Diagram

Description automatically generated

### Figure 1.3 – Overall Architecture

**Diagram, schematic

Description automatically generated**

### Figure 1.4 – ER Diagram

Chart, diagram, box and whisker chart

Description automatically generated

### Figure 1.5 – Activity Diagram

## Team Member 1 – Thathsarani R.P.H.S.R / IT20201364

**Class Diagram**

Table

Description automatically generated with medium confidence

Figure 2.1

**Activity Diagram**

Diagram

Description automatically generated

Figure 2.2

**Flow Chart**

Diagram

Description automatically generated

Figure2.3

**Sequence Diagram**

Diagram

Description automatically generated

Figure 2.4

## Team Member 2 – Peiris M.I.M / IT20165352

**Class Diagram**

Text

Description automatically generated

Figure3.1

**Activity Diagram**

|  |  |  |
| --- | --- | --- |
| **Insert a device** | **Update device details** | **Delete device** |
| Diagram  Description automatically generated | **Diagram  Description automatically generated** | **Diagram  Description automatically generated** |

### 

Figure 3.2

Figure 3.4

Figure 3.3

**Flow Chart**

Diagram

Description automatically generated

Figure 3.5

**Sequence Diagram**

Diagram

Description automatically generated with medium confidence

Figure 3.6

## Team Member 3 - Isurika M.D.A / IT20187828

**Class Diagram**

Diagram

Description automatically generated with medium confidence

Figure 4.1

**Activity Diagram**

Diagram

Description automatically generated

Figure 4.2

**Flow Chart**

Diagram, engineering drawing

Description automatically generated

Figure 4.3

**Sequence Diagram**

Diagram

Description automatically generated

Figure 4.4

## Team Member 4 - Kandanaarachchi H.L.D.S/ IT20157500

**Class Diagram**

Table

Description automatically generated with medium confidence

Figure 5.1

**Activity Diagram**

Diagram, schematic

Description automatically generated

Figure 5.2

**Flow Chart**

Diagram

Description automatically generated

Figure 5.3

**Sequence Diagram**

Diagram

Description automatically generated

Figure 5.4

## Team Member 5 - A.K.F Hasna/ IT20212940

**Class Diagram**

Table

Description automatically generated

Figure 6.1

Diagram

Description automatically generated **Activity Diagram**

Diagram

Description automatically generated

inserting an interrupt Notice. Updating an Interrupt Notice

Figure 6.3

Figure 6.2

Figure 6.5

Figure 6.4

Diagram

Description automatically generatedDiagram

Description automatically generated

deleting an Interrupt Notice retrieving Interrupt Notices

**Flow Chart**

Diagram

Description automatically generated

Figure 6.6

**Sequence Diagram**

Diagram, engineering drawing

Description automatically generated

Figure 6.7

## Team Member 6 - M.N Aaisha/ IT20202354

**Class Diagram**

A picture containing text

Description automatically generated

Figure 7.1

**Activity Diagram**

**Insert Update**

Graphical user interface, text, application

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

Figure 7.3

Figure 7.2

Diagram

Description automatically generated**Delete Update Status**

A picture containing graphical user interface

Description automatically generated

Figure 7.5

Figure 7.4

**View Flow Chart**

Diagram

Description automatically generatedDiagram

Description automatically generated

Figure 7.7

Figure 7.6

**Sequence Diagram**

**Create**  **Delete**

Text

Description automatically generatedText

Description automatically generated

Figure 7.9

Figure 7.8

**Update**

Text

Description automatically generated

Figure 7.10

**Status Update**

Diagram, text

Description automatically generated

Figure 7.11