

Sri Lanka Institute of Information Technology

**ElectroGrid System Web Services**

**Project Report**

IT3030 - Programming Applications and Frameworks 2022

Group Assignment

Batch No: Y3.S1.WE.DS.04

Group ID: 151

Group Members:

1. IT20186142 – Wijesooriya H.M.A.H.
2. IT20183554 – Chandrasena M.C.
3. IT20188054 – Benthota Arachchi B.A.T.P.
4. IT20012342 – Sanjeewa J.M.I.P.
5. IT20178840 – Weerasinghe T.R.

Date of submission: 25/04/2022

**Table of Contents**

1. Workload Distribution………………………………………………………………………3
2. Version Controlling…………………………………………………………………………4
   1. Repository Links……………………………………………………………………….....4

2.2. Commit Log………………………………………………………………………………4

1. SE Methodology. …………………………………………………………………………...5
2. Time Schedule(Gantt chart) ………………………………………………………………. .5
3. Requirements……………………………………………………………………………..... 6
   1. Stakeholder Analysis……………………………………………………………………..6
   2. Requirements Analysis…………………………………………………………………...6
      1. Functional Requirements………………………………………………..7
      2. Non-functional Requirements…………………………………………...8
      3. Technical Requirements…………………………………………………9
   3. Requirements Modelling…………………………………………………………………10
4. Overall System Design……………………………………………………………………..11
   1. Overall System Architecture……………………………………………………………..11
   2. Overall Database Architecture…………………………………………………………...12
   3. Overall Activity Diagram………………………………………………………………..13
5. Individual section…………………………………………………………………………..14

7.1. User Service & Breakdown Service…………………………………………………14

7.1.1. Activity Diagrams……………………………………………………..15

7.1.2. Class Diagram…………………………………………………………16

7.1.3. Service Development and Testing…………………………………….16

7.2. Power Management Service…………………………………………………………18

#### 7.2.1. Activity Diagram………………………………………………………19

#### 7.2.2. Class Diagram…………………………………………………………20

#### 7.2.3. Service Development and Testing…………………………………….20

#### 7.3. Technician Management Service……………………………………………………..22

#### 7.3.1. Activity Diagram……………………………………………………23

#### 7.3.2. Class Diagram ……………………………………………………...24

#### 7.3.3. Service Development and Testing………………………………….25

#### 7.4. Payment Management Service………………………………………………………...26

#### 7.4.1. Activity Diagram……………………………………………………...27

#### 7.4.2. Class Diagram ………………………………………………………...28

#### 7.4.3. Service Development and Testing…………………………………….28

#### 7.5. Supplier Service & Grid Service…………………………………………………….30

#### 7.5.1. Activity Diagram……………………………………………………..31

#### 7.5.2. Class Diagram ………………………………………………………..32

#### 7.5.3. Service Development and Testing…………………………………….32

1. System’s Integration Details.……………………………………………………………...34
2. References………………………………………………………………………………….34
3. Workload Distribution

|  |  |  |
| --- | --- | --- |
| **Re­gistration Number** | **Name with Initials** | **Web Service Allocated** |
| IT20012342 | Sanjeewa J.M.I.P. | User Service   * User Management * User Role Management * User Login & Password Reset   Breakdown Service   * Reporting and managing breakdowns * Updating status of breakdowns |
| IT20188054 | Benthota Arachchi B.A.T.P. | Power Management Service   * Manage the power consumption of the user * Maintain details of the power generation capacity inserted into the ElectroGrid * View power management monthly summary data * View power management annual summary data |
| IT20186142 | Wijesooriya H.M.A.H. | Technicians Service   * Manage the details of technicians * Marking and manage attendance details of all technicians * View attendance summary of each technician |
| IT20178840 | WeerasingheT.R. | Payment Service   * Payment Management * Managing payment information for each user and supplier |
| IT20183554 | Chandrasena M.C. | Supplier Service   * Supplier management * Power grid management of suppliers |

# Version Controlling

## 2.1. Repository Links

* Public GitHub Repository Link: <https://github.com/AvishkaHashenu99/ElectroGrid>
* Public GitHub Repository Clone Link: <https://github.com/AvishkaHashenu99/ElectroGrid.git>

**2.2. Commit Log**

The commit log extracted from the GitHub repository insights is attached below and it shows how the commits have been done from March 21 to Apr 21, 2021.

|  |  |
| --- | --- |
| **Registration Number** | **GitHub Username** |
| IT20186142 Wijesooriya H.M.A.H. | AvishkaHashenu99 |
| IT20183554 Chandrasena M.C. | ChiranjayaChandrasena |
| IT20188054 Benthota Arachchi B.A.T.P. | tinulpiumika |
| IT20012342 Sanjeewa J.M.I.P. | IT20012342 |
| IT20178840 Weerasinghe T.R | TolaniRidmini |

A screenshot of a computer

Description automatically generated with medium confidence

# SE Methodology

The waterfall model was the methodology used by us to do the project, because software requirements were clearly defined and known, and the production definition also was a stable one. All of our group members are familiar with the technologies and tools used by us. It was also a key point to use the waterfall model rather than other software methodologies.

Rather than defining sprint-vice targets, it was helpful to specify a set of features to be implemented in each web service by each member as the project’s final goal. This helped the team have a clear idea of the web service and their capabilities that would be developed.

Another motive for using the waterfall paradigm was to create detailed API documentation, which wasn't one of the key objectives in agile methodologies. Furthermore, because the project timeline was a fixed period and the deadline was known in advance, the waterfall approach was considered more feasible.

Apart from the development methodologies, as the software architecture of the Electro Grid system, the Microservice Architecture was used. To be compatible with the defined requirements, the Electro Grid system was supposed to be highly scalable. Therefore, a distributed system with independently developed web services seemed like the most suitable option. Development of the microservices was done using the resource-oriented architecture (REST) to establish a resource-oriented communication between web services and clients.

Chart

Description automatically generated

1. **Time Shedule**
2. **Requirements**

## 5.1. Stakeholder Analysis

**Chart, radar chart, sunburst chart

Description automatically generated**

# 5.2. Requirements Analysis

Used standard requirement gathering techniques for identifying, gathering, and recording the requirements. That’s namely, document analysis, research, and brainstorming. In the requirements gathering phase, all the specifics in the assignment specifications file have been carefully examined, and any potential requirements identified have been recorded. Furthermore, all members of the group have conducted research on existing research and innovation supportive web systems and web services, as well as other research papers on web services, to compile a comprehensive list of well-defined requirements and find possible web services. The next step was to identify stakeholders to determine their needs and expected functions from the previously defined web services.

Finally, the team used brainstorming to identify any requirements that were overlooked. The team came to the following conclusion on the finished set of functional requirements.

### 5.2.1. Functional Requirements

The functional requirements of each web service are shown in a tabular format along with the stakeholder type who requires it as follows.

|  |  |
| --- | --- |
| **Web Service** | **Functional Requirement** |
| User Service | Add user accounts |
| Update user accounts |
| Delete user accounts |
| View user accounts |
| Validate login details |
| Reset Password |
| Power Management Service | Add power consumption details of the user |
| View power consumption details of users |
| Update power consumption details of the user |
| Delete power consumption details of the user |
| Add power generation details |
| View power generation details |
| Update power generation details |
| Delete power generation details |
| View power management monthly summary data |
| View power management annual summary data |
| Technician Service | Add technician details |
| Update technician details |
| Delete technician details |
| View technicians details |
| Add attendance details of technicians for each day |
| Update attendance details of technicians |
| View attendance details of technicians |
| View attendance summary of individual technician |
| Delete attendance details of technicians |
| Payment Service | Add new payment records |
| Update payment records |
| Delete payment records |
| View payment records |
| Add new payment information |
| Update payment information |
| Delete payment information |
| View payment information |
| Supplier Service | Add new supplier details |
| View supplier details |
| Update supplier details |
| Delete supplier details |
| Add new power grid details |
| View power grid details |
| Update power grid details |
| Delete power grid details |
| Breakdown Service | Add breakdown complains |
| View breakdown complains |
| Update breakdown complains details |
| Delete breakdown complains |
| Update status of the breakdown |

### 5.2.2. Non-functional Requirements

**Availability**

Other than during system maintenance, online services should be available and accessible at all times. Except when inter-service communication is vital to fulfilling a functional requirement, a particular service should run correctly even if another web service in the system is down or under maintenance.

**Maintainability**

API documentation of the web services should be well-written and understandable by users or developers who implement client programs to communicate with the services and should support proper troubleshooting methods, media types consumed by endpoints, and URI formats. .

**Reliability**

The web services should have necessary backup features in case of an emergency, and it should support alternating the databases and should provide the required privileges to do so accordingly. Each web service is preferred to use its databases and tables to minimize data transfer delays. In case of an external attack, the user service should be able to handle user accounts accordingly in a way that will minimize the system downtime.

**Portability**

Regardless of the language or technology stack used to create client programs, any sort of client program should be able to call API end points and request information from web services.

**Security**

All user credentials and stored data must be secured and organized so that only system administrators and authorized users can view or alter them. When the password is forgotten, user can reset the password by providing user’s reset code and new password.

**Usability**

An easily usable URI format should be designed and implemented for the endpoints of the web service APIs, which can be easily comprehended by the system users or client applications.

### 5.2.3. Technical Requirements

* Technical requirements for the development of the web services are as follows.

**IDE:** Eclipse versions between 2021.3 and 2022.3 inclusive

**Server:** Apache Tomcat Server Version 9

**Java Development Kit**: JDK17

**Java Runtime Environment:** JRE 17

**Operating System:** Any OS that supports eclipse versions mentioned above

**Frameworks:**  JAX-RS/ Jersey 2.35

**Libraries:** Jersey core Servlet – version 2.35

HK2 InjectionManager - version 2.35

MySQL connector (JDBC)

Google GSON for JSON serialization/de-serialization

**Dependency Management:** Maven 3.8.5

**Version Control:** Git and GitHub

**Database Management:** MySQL Workbench

* Recommended Requirements for implementing the web services of ElectroGrid are as follows.

**Server:** Apache Tomcat Server Version 9 or any cloud service that accepts the hosting of war packages will be compatible.

**Java Runtime Environment:** JRE 17

**Operating System:** Any OS that supports JDK 17/JRE 17 and Apache Tomcat Server

**Database Server:** MySQL server or any cloud service that accepts the hosting of MySQL server will be compatible as well.

## Requirements Modelling

**Overall Use Case Diagram**

**Diagram

Description automatically generated**

1. **Overall System Design**

## 6.1. Overall System Architecture

Diagram

Description automatically generatedThe overall design of the Electrogrid system is shown in the following figure. Stakeholders who interface with the system directly may utilize a variety of client applications depending on their role; however, the client applications were not developed as part of this project. Instead of the client applications shown in the diagram, a test client was used.

## Diagram Description automatically generated Overall DB Design (ER Diagram)

## Diagram, schematic Description automatically generated Overall Activity Diagram

**Diagram, schematic

Description automatically generated**

# Individual Sections

* 1. **IT20012342 Sanjeewa J.M.I.P.**

## User Service & Breakdown Service

Diagram

Description automatically generated

**Diagram

Description automatically generated**

User Service is responsible for managing all the users who are directly interacting with all web services of ElectroGrid system. User service maintain user details, maintain user roles, maintain login functionality to the system by validating username & password and user can reset the password by providing the reset code when password is forgotten.

Breakdown Service is responsible for managing all the breakdown complains by directly interacting with relevant web services of ElectroGrid system. Breakdown service maintains breakdown details and the user can perform modifications when modifications needed. The date and time of breakdown insertion is automatically recorded in the database. Admin must update the status relevant to complain.

#### Activity Diagrams

Diagram, schematic

Description automatically generated

* + 1. **Class Diagram**

**Diagram

Description automatically generated**

### Service Development and Testing

* + - 1. **Tools Used**
* **Eclipse + JRE 17/JDK 17 + Apache Tomcat 9** with **Jersey** **Library** were used to develop the code base of the user service. The reason for using the specific versions mentioned above was that they are the latest versions supported by JAX-RS and Jersey version 2.35.
* **Maven 3.8.5** was used for dependency management to easily add required dependencies and plugins without any issues and setting any manual configurations.
* **GitHub** was used to enable team collaborations and have a comprehensive clean history that is recoverable in case of a crash or any other development-related issues.
* **JSON** was used to transfer data in every endpoint so that services/ clients can easily communicate with the web service rather than dealing with various media types in response.
* **JUnit** was usedas the testing tool to check the functionalities of the ElectroGrid system.
* **Postman** was used as a test client rather than using a regular browser, where all types of HTTP requests can be sent with proper data types included in the request header and payload.
  + - 1. **Testing methodology and results**

**Functional Testing –**

* Inserted values are validated and all the validations functions properly.
* Test cases are executed using JUnit testing tool providing expected output.
* APIs are tested using Postman by passing necessary payloads.

**Database testing –**

* CRUD operations are performed without any errors.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Case | Inputs | Expected Output | Actual Output | Status |
| 1 | Get all user details | - | A table that contains details of all the users | A table that contains details of all the users | Pass |
| 2 | Insert user details | JSON object containing new user data | Display insertion success message | Insertion success message has been displayed | Pass |
| 3 | Update user details | JSON object containing updated user data | Display updation success message | Updation success message has been displayed | Pass |
| 4 | Delete user account | JSON object containing user ID | Display deletion success message | Deletion success message has been displayed | Pass |
| 5 | Validate user login | JSON object containing user login details | Display login success message | Login success message has been displayed | Pass |
| 6 | Reset password | JSON object containing reset password details | Display password reset success message | Password reset success message has been displayed | Pass |
| 7 | Get all breakdown complains | - | A table that contains details of all the complains | A table that contains details of all the complains | Pass |
| 8 | Insert new breakdown complain | JSON object containing new breakdown data | Display insertion success message | Insertion success message has been displayed | Pass |
| 9 | Update breakdown complain | JSON object containing updated breakdown data | Display updation success message | Updation success message has been displayed | Pass |
| 10 | Delete breakdown complain | JSON object containing breakdown ID | Display deletion success message | Deletion success message has been displayed | Pass |

* 1. **IT20188054 Benthota Arachchi B.A.T.P.**

## Power Management Service

Diagram

Description automatically generatedDiagram

Description automatically generated

Diagram

Description automatically generated

Power consumption service is responsible for managing, viewing, and modifying all the power consumption details of the users who are directly interacting and consuming electricity from the ElectroGrid system. Power consumption service manages the consumption data which includes the record ID, account number, and no. of units in the current reading of the user (*Assuming the meter reader gets the current reading value from the consumer meter*), date of the reading, reader ID, and user ID. When these data are added to the service, the system will automatically read the current reading of the previous month for the relevant user and get the difference between the current reading and the previous reading as consuming units of the user for the relevant month.

Power capacity service is responsible for managing, viewing, and modifying all the power generation details of the power suppliers who are directly interacting and generating electricity for the ElectroGrid system. Power capacity service manages the power generating data which includes the power supplier ID, date of the power generation, type, and the number of units added to the system.

Power summary service is responsible for generating a summarized view by combining the power capacity and consumption data of the ElectroGrid according to a specific month or for a specific year.

#### Activity Diagrams

Diagram, schematic

Description automatically generated

* + 1. Diagram

       Description automatically generated**Class Diagram**

### Service Development and Testing

* + - 1. **Tools Used**
* **Eclipse + JRE 17/JDK 17 + Apache Tomcat 9** with **Jersey** **Library** were used to develop the code base of the user service. The reason for using the specific versions mentioned above was that they are the latest versions supported by JAX-RS and Jersey version 2.35.
* **Maven 3.8.5** was used for dependency management to easily add required dependencies and plugins without any issues and setting any manual configurations.
* **GitHub** was used to enable team collaborations and have a comprehensive clean history that is recoverable in case of a crash or any other development-related issues.
* **JSON** was used to transfer data in every endpoint so that services/ clients can easily communicate with the web service rather than dealing with various media types in response.
* **JUnit** was usedas the testing tool to check the functionalities of the ElectroGrid system.
* **Postman** was used as a test client rather than using a regular browser, where all types of HTTP requests can be sent with proper data types included in the request header and payload.
  + - 1. **Testing methodology and results**

**Functional Testing –**

* Inserted values are validated and all the validations function properly
* Test cases are executed using the JUnit testing tool providing expected output.
* APIs are tested using Postman by passing necessary payloads.

**Database testing –**

* CRUD operations are performed without any errors.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Case | Inputs | Expected Output | Actual Output | Status |
| 1 | Get all power consumption details | - | A table that contains details of all the power consumption | A table that contains details of all the power consumption | Pass |
| 2 | Insert power consumption details | JSON object containing new consumption data | Display insert success message | Insert success message has been displayed | Pass |
| 3 | Update power consumption details | JSON object containing updated consumption data | Display update success message | Update success message has been displayed | Pass |
| 4 | Delete power consumption account | JSON object containing record ID | Display delete success message | Delete success message has been displayed | Pass |
| 5 | Get all power capacity details | - | A table that contains details of all the capacity | A table that contains details of all the capacity | Pass |
| 6 | Insert new power capacity details | JSON object containing new power capacity data | Display insert success message | Insert success message has been displayed | Pass |
| 7 | Update power capacity details | JSON object containing updated power capacity data | Display update success message | Update success message has been displayed | Pass |
| 8 | Delete power capacity details | JSON object containing capacity ID | Display delete success message | Delete success message has been displayed | Pass |
| 9 | Get summarize the view of power management (Monthly) | JSON object containing month | display data that contains details of summarized data for the month | display data that contains details of summarized data for the month | Pass |
| 10 | Get summarize the view of power management (Annual) | JSON object containing year | A table that contains details of summarized data for the year | A table that contains details of summarized data for the year | Pass |

* 1. **IT20186142 Wijesooriya H.M.A.H.**

## Technician Management Service

Diagram

Description automatically generated

Technician Service is responsible for managing the details of technicians who is one of the major actors of ElectroGrid power company. Technician service maintain technician details and maintain technician roles which are hourly based technicians and contract-based technicians. Admin can perform modifications when modifications needed.

TechnicianAttendance Service is responsible for managing all the attendance details of all technicians for each day. Admin can perform modifications when modifications needed.

Both of these services are totally managed by the admin of the system.

#### Activity Diagrams

**Diagram, schematic

Description automatically generated**

* + 1. **Class Diagram**

**Diagram

Description automatically generated**



### Service Development and Testing

* + - 1. **Tools Used**
* **Eclipse + JRE 17/JDK 17 + Apache Tomcat 9** with **Jersey** **Library** were used to develop the code base of the user service. Reason for using the specific versions mentioned above was that they are the latest versions supported by JAX-RS and Jersey version 2.35.
* **Maven 3.8.5** was used for dependency management to easily add required dependencies and plugins without any issues and setting any manual configurations.
* **GitHub** was used to enable team collaborations and have a comprehensive clean history that is recoverable in case of a crash or any other development related issues.
* **JSON** was used to transfer data in every endpoint so that services/ clients can easily communicate with the web service rather than dealing with various media types in responses.
* **JUnit** was usedas the testing tool to check the functionalities of the ElectroGrid system.
* **Postman** was used as a test client rather than using a regular browser, where all types of HTTP requests can be sent with proper data types included in the request header and payload.
  + - 1. **Testing methodology and results**

**Functional Testing –**

* Inserted values are validated and all the validations function properly
* Test cases are executed using JUnit testing tool providing expected output.
* APIs are tested using Postman by passing necessary payloads.

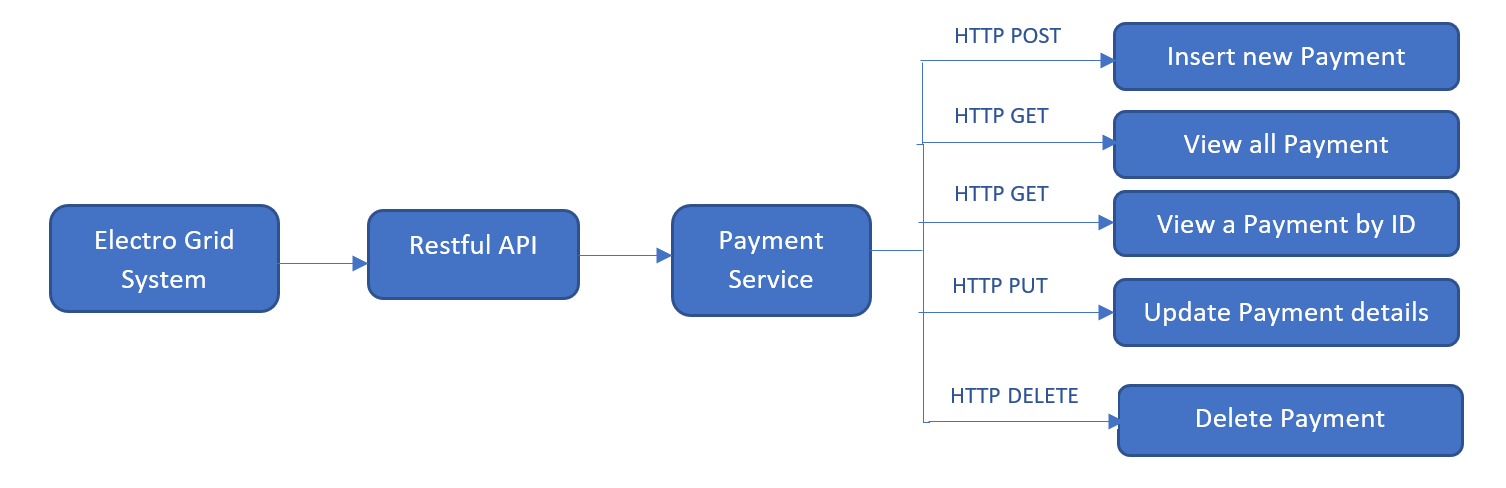
**Database testing –**

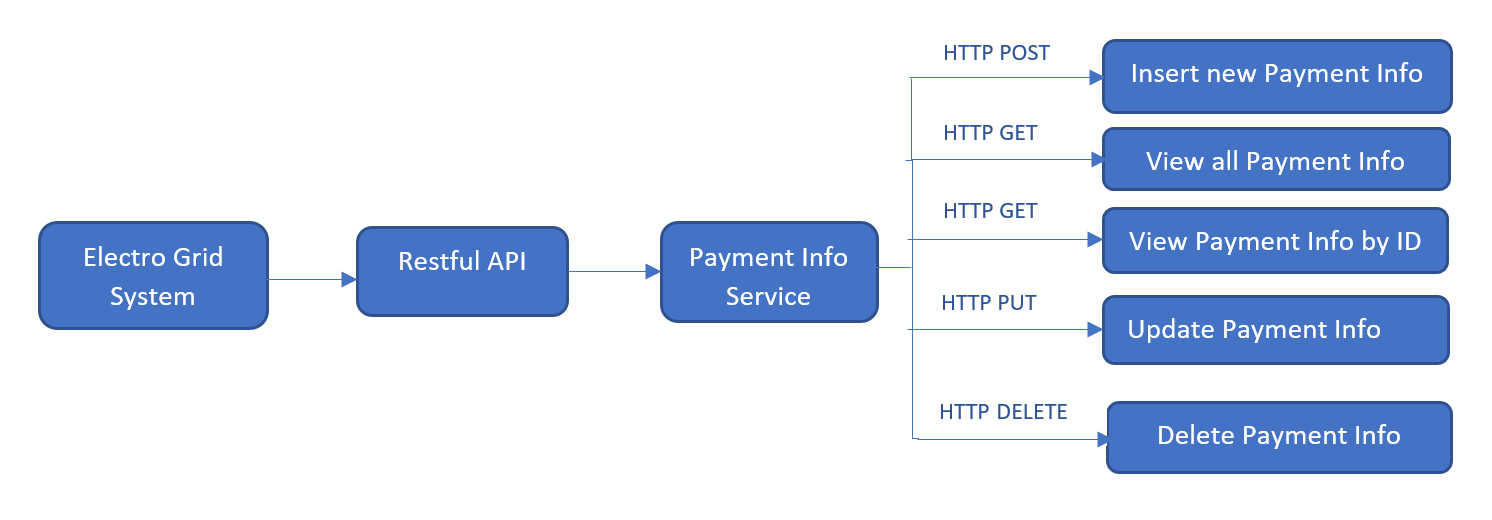
* CRUD operations are performed without any errors.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Case | Inputs | Expected Output | Actual Output | Status |
| 1 | Get all technicians details | - | A table that contains details of all the technicians | A table that contains details of all the technicians | Pass |
| 2 | Insert technician details | JSON object containing new technician data | Display insertion success message | Insertion success message has been displayed | Pass |
| 3 | Update technician details | JSON object containing updated technician data | Display updation success message | Updation success message has been displayed | Pass |
| 4 | Delete technician details | JSON object containing technician ID | Display deletion success message | Deletion success message has been displayed | Pass |
| 6 | Get all attendance details | - | A table that contains all attendance details | A table that contains all attendance details | Pass |
| 7 | Insert new attendance record | JSON object containing attendance details | Display insertion success message | Insertion success message has been displayed | Pass |
| 8 | Update attendance details | JSON object containing updated attendance details | Display updation success message | Updation success message has been displayed | Pass |
| 9 | Delete attendance details | JSON object containing attendance ID | Display deletion success message | Deletion success message has been displayed | Pass |

**7.4. IT20178840 Weerasinghe T.R**

**7.4.1. Payment Management Service**





Payment service is responsible for handling all the payments of the users and power suppliers who are interacting with ElectroGrid system. Payment service handles the payment according to the number of units consumed by the users and the number of units supplied by the power suppliers. Admin can insert,update, and delete records.

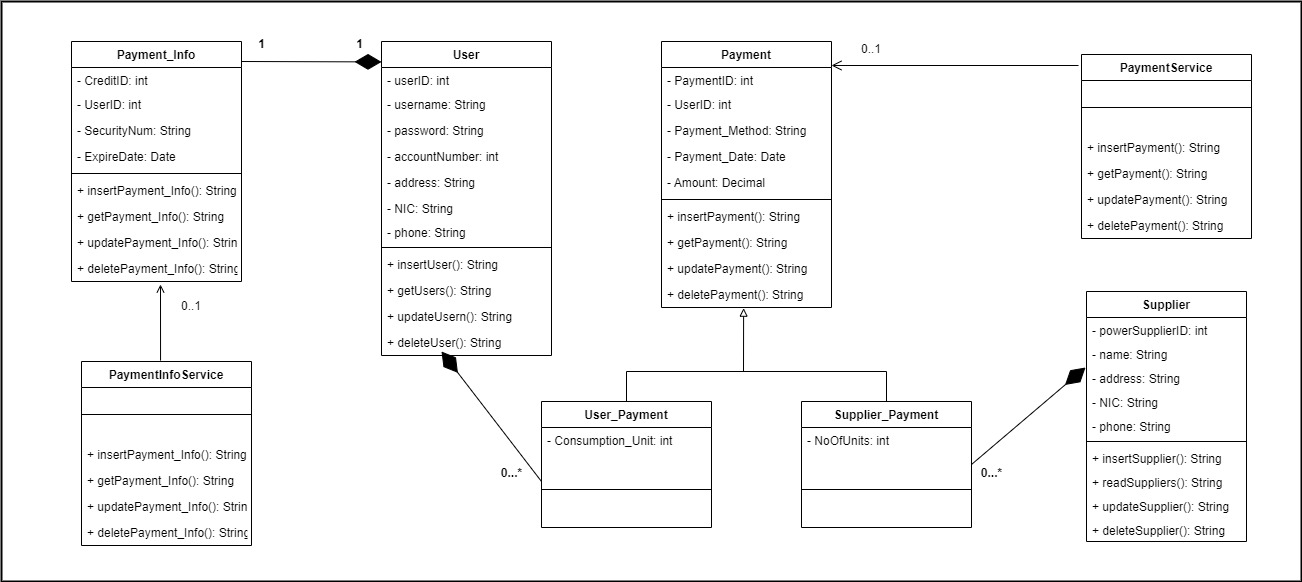
Payment Info service is responsible for managing payment information for each user. Payment Info service lists down information by including the UserID, CreditID, Security Number and Expire Date. Admin can perform any modifications for these payment information.

#### 7.4.2. Activity Diagrams

**Diagram, schematic

Description automatically generated**

* + 1. **Class Diagram**



### Service Development and Testing

* + - 1. **Tools Used**
* **Eclipse + JRE 17/JDK 17 + Apache Tomcat 9** with **Jersey** **Library** were used to develop the code base of the user service. Reason for using the specific versions mentioned above was that they are the latest versions supported by JAX-RS and Jersey version 2.35.
* **Maven 3.8.5** was used for dependency management to easily add required dependencies and plugins without any issues and setting any manual configurations.
* **GitHub** was used to enable team collaborations and have a comprehensive clean history that is recoverable in case of a crash or any other development related issues.
* **JSON** was used to transfer data in every endpoint so that services/ clients can easily communicate with the web service rather than dealing with various media types in responses.
* **JUnit** was usedas the testing tool to check the functionalities of the ElectroGrid system.
* **Postman** was used as a test client rather than using a regular browser, where all types of HTTP requests can be sent with proper data types included in the request header and payload.
  + - 1. **Testing methodology and results**

**Functional Testing –**

* Inserted values are validated and all the validations function properly
* Test cases are executed using JUnit testing tool providing expected output.
* APIs are tested using Postman by passing necessary payloads.

**Database testing –**

* CRUD operations are performed without any errors.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Case | Inputs | Expected Output | Actual Output | Status |
| 1 | Get all payment details | - | A table that contains details of all the payment | A table that contains details of all the payment | Pass |
| 2 | Insert payment details | JSON object containing new payment data | Display insertion success message | Insertion success message has been displayed | Pass |
| 3 | Update payment details | JSON object containing updated payment data | Display updation success message | Updation success message has been displayed | Pass |
| 4 | Delete payment details | JSON object containing payment ID | Display deletion success message | Deletion success message has been displayed | Pass |
| 6 | Get all payment method details | - | A table that contains payment method details | A table that contains all payment method details | Pass |
| 7 | Insert new payment method | JSON object containing payment method details | Display insertion success message | Insertion success message has been displayed | Pass |
| 8 | Update payment method details | JSON object containing updated payment method details | Display updation success message | Updation success message has been displayed | Pass |
| 9 | Delete payment method | JSON object containing Credit ID | Display deletion success message | Deletion success message has been displayed | Pass |

* 1. **IT20183554 Chandrasena M.C.**

## Diagram Description automatically generatedSupplier Service & Grid Service

Supplier Service is responsible for managing all the suppliers of the ElectroGrid system. Supplier service maintains supplier details and is used by the administration for adding new suppliers to the system, view them, do modifications for the details of the existing suppliers and remove suppliers.

Grid Service is responsible for managing all the power grids owned by the suppliers of the Electro Grid system. Grid service maintains grid details and is used by the administration for adding new grids to the system, view them, do modifications for the details of the existing grids and remove grids.

#### Activity Diagrams

Diagram, schematic

Description automatically generated

* + 1. **Class Diagram**

**Graphical user interface, application

Description automatically generated**

### Service Development and Testing

* + - 1. **Tools Used**
* **Eclipse + JRE 17/JDK 17 + Apache Tomcat 9** with **Jersey** **Library** were used to develop the code base of the user service. Reason for using the specific versions mentioned above was that they are the latest versions supported by JAX-RS and Jersey version 2.35.
* **Maven 3.8.5** was used for dependency management to easily add required dependencies and plugins without any issues and setting any manual configurations.
* **GitHub** was used to enable team collaborations and have a comprehensive clean history that is recoverable in case of a crash or any other development related issues.
* **JSON** was used to transfer data in every endpoint so that services/ clients can easily communicate with the web service rather than dealing with various media types in responses.
* **JUnit** was usedas the testing tool to check the functionalities of the ElectroGrid system.
* **Postman** was used as a test client rather than using a regular browser, where all types of HTTP requests can be sent with proper data types included in the request header and payload.
  + - 1. **Testing methodology and results**

**Functional Testing –**

* Inserted values are validated and all the validations functions properly
* Test cases are executed using JUnit testing tool providing expected output.
* APIs are tested using Postman by passing necessary payloads.

**Database testing –**

* CRUD operations are performed without any errors.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Test Case | Inputs | Expected Output | Actual Output | Status |
| 1 | Get all supplier details | - | A table that contains details of all the suppliers | A table that contains details of all the suppliers | Pass |
| 2 | Insert supplier details | JSON object containing new supplier data | Display insertion success message | Insertion success message has been displayed | Pass |
| 3 | Update supplier details | JSON object containing updated supplier data | Display updation success message | Updation success message has been displayed | Pass |
| 4 | Delete supplier | JSON object containing supplier ID | Display deletion success message | Deletion success message has been displayed | Pass |
| 5 | Get all grids | - | A table that contains details of all the grids | A table that contains details of all the grids | Pass |
| 6 | Insert new grid | JSON object containing new grid data | Display insertion success message | Insertion success message has been displayed | Pass |
| 7 | Update grid details | JSON object containing updated grid data | Display updation success message | Updation success message has been displayed | Pass |
| 8 | Delete grid | JSON object containing grid ID | Display deletion success message | Deletion success message has been displayed | Pass |

1. **System’s Integration Details**

* All of our five microservices are running in the same Tomcat server.
* Our project integration was done through the version control system, Github.
* Here there is possibility to check the system functionalities through Junit testing tool by providing expected output.
* Also, there is possibility to test the APIs through the test client, Postman by passing necessary payloads.
* All of our group members used the same database by executing each sql query in their own databases.

1. **References**

[1]. eclipse-ee4j.github.io, ‘jersey.github.io/documentation’,

[Jersey 2.35 User Guide (eclipse-ee4j.github.io)](https://eclipse-ee4j.github.io/jersey.github.io/documentation/latest/index.html)

[2]. mvnrepository.com, ‘Jersey Container Servlet’,

[MavenRepository:org.glassfish.jersey.containers»jersey-container-servlet (mvnrepository.com)](https://mvnrepository.com/artifact/org.glassfish.jersey.containers/jersey-container-servlet)

[3]. mvnrepository.com, ‘Jersey Inject HK2’,

[Maven Repository: org.glassfish.jersey.inject » jersey-hk2 (mvnrepository.com)](https://mvnrepository.com/artifact/org.glassfish.jersey.inject/jersey-hk2)

[4]. mvnrepository.com, ‘MySQL Connector’,

[Maven Repository: mysql » mysql-connector-java (mvnrepository.com)](https://mvnrepository.com/artifact/mysql/mysql-connector-java)

[5]. mvnrepository.com, ‘Gson’

[Maven Repository: com.google.code.gson » gson (mvnrepository.com)](https://mvnrepository.com/artifact/com.google.code.gson/gson)

[6]. mvnrepository.com, ‘JUnit Platforn Launcher’

[Maven Repository: org.junit.platform » junit-platform-launcher (mvnrepository.com)](https://mvnrepository.com/artifact/org.junit.platform/junit-platform-launcher)