IT3030 – Programming Applications and Frameworks

Batch – Y3.S2.DS.03
Assignment Group – 17

Group members:

Registration No.	Name
IT19142838	B.W.E.K. Senarathna
IT19188928	M.M. Saluka Udbhasa
IT19001708	L.L.P.S.M. Lelkada
IT19956954	U.W.D.G.B. Udupihilla

Submitted to:
Mr. Nalaka R. Dissanayake

Contents

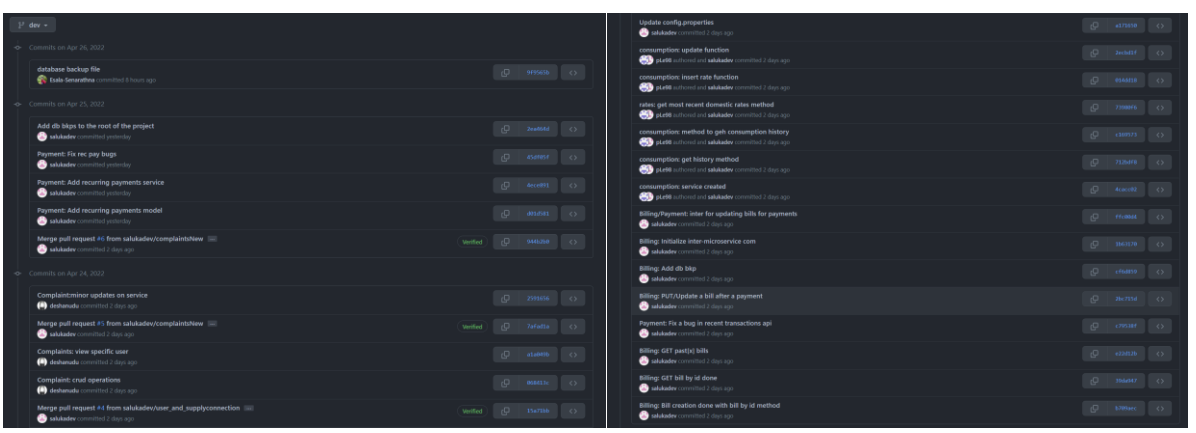
Group work

1. Member details and role

Student ID	Name	Web service	Description
IT19142838	B.W.E.K. Senarathna	Supply Connection and user service	Supply Connection Service - Create connection - Update connection - View all connections - Delete connection Supply Connection Service - Register user - Delete user - Update user details - login
IT19188928	M.M. Saluka Udbhasa	Payment Service Billing Service	Payment Service - Make payment - View payments - Cancel/Delete payments - Webhook for payment gateway providers to update transaction status - Create/view recurring payments - Update/cancel recurring payments Billing Service - Calculate/create Bill - View Bill History - View Specific Bill - Update Bills after payment - Communicate with payment service
IT19001708	L.L.P.S.M. Lelkada	Consumption Service Rates Service	- Calculate Consumption - Insert Consumption - View Consumption - View Recent Rates - Insert Rate - Update Rate - Delete Rate
IT19956954	U.W.D.G.B. Udupihilla	Complaints Service	- Add new Complaint - View all Complaints - View a specific Complaint - Update a Complaint - Delete a Complaint

2. Clickable link for GitHub Repository

https://github.com/salukadev/ElectroGrid-Information-System



3. Requirements Analysis (Functional, Non-functional, Technical requirements)

Functional Requirements

1. Managing connections

Each electricity meter can be considered as a connection. System should have the ability to maintain connection/account details along with other information.

2. User profile management

User registration, login and other user profile management operations can be included in this requirement. User records should be maintained by linking it to a connection/billing account.

3. Metering/Consumption calculation

Calculate the consumption of each meter/connection using the ingested data according to the business logic. Calculated values should be recorded and communicated to the other services

4. Billing operations

Automatically generate bills on the billing date, and update them after the payments are received. Users should be able to view past bills.

5. Payment management

System should have the ability to accept user payments using it's own payment gateway or from a 3rd party vendor. Either way payment information should be recorded in a company managed database and it should need to communicate & update the billing service when transactions are occurring.

User should be able to schedule payments which those recurring payment methods have to be automatically executed each month to charge the mentioned value.

6. Rate management

System records the rates and changes of the rates for the different types of accounts. Users should be able to view rates that is used to calculate the bill.

7. Complaints management

User related issues like reporting power failures and other related issues regarding the power management system needed to be managed through the system

Non-Functional Requirements

1. Availability – System should be accessible from any type of device (cross platform compatibility) and should be available 24/7.

2. Accuracy – Consumption, bill and payment details should be recorded accurately, and calculations of the consumptions and bills should be calculated according to the latest rates and payments details.

3. Security – External parties should not be able to change payment, bill, rate data and user information of the users should be well secured.

4. Privacy – User data should not be disclosed to anyone other than user himself and system admins.

5. Scalability – System should be able to handle increasing and decreasing workloads appropriately without affecting to the overall performance.

Technical Requirements

1. Performance

Average wait times and loading times should be minimized to serve a better customer experience. In addition to the user experience improving performance may helps to reduce the running cost of the system.

2. Scalability

The system should have the ability to scale itself up and down according to the demand of resources and the interacting user count. Containerization strategy best suits for these kinds of microservice deployments where scalability can be adopted from its nature.

3. Enforced privacy within datacentres

Since the system contains sensitive data like transactions and personal information it should maintain a computing/database environment which compliant to standards like GDPR, PDPA and etc.

4. Software Engineering Methodology - Agile Software Development

Initially we had several meetings to discuss the scope of the project and understood the user cases which needed to be implemented based on a requirement gathering. These use cases were structured based on the services and core services of the application were found. Based on the core services the application database was designed using the developed database we started implementing the services of the system.

Diagrams

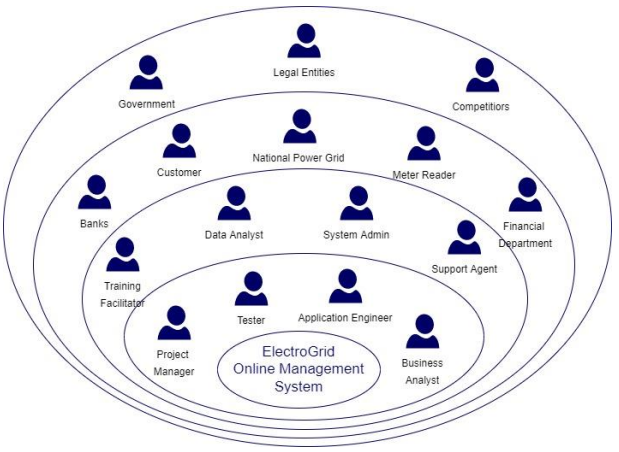
In the process of planning the project we used several diagrams like ER diagrams, use case diagrams, do design the project and diagrams like Activity diagrams are used to

In the process of developing the system we used Agile method in developing the system. We had daily stand-ups to discuss issues what each member faced during the day when he was involved in the development process, and they were sorted within the team.

Version Controlling System

We hosted our repository on GitHub and using git we did the version control and the merging of the individual components to using git.

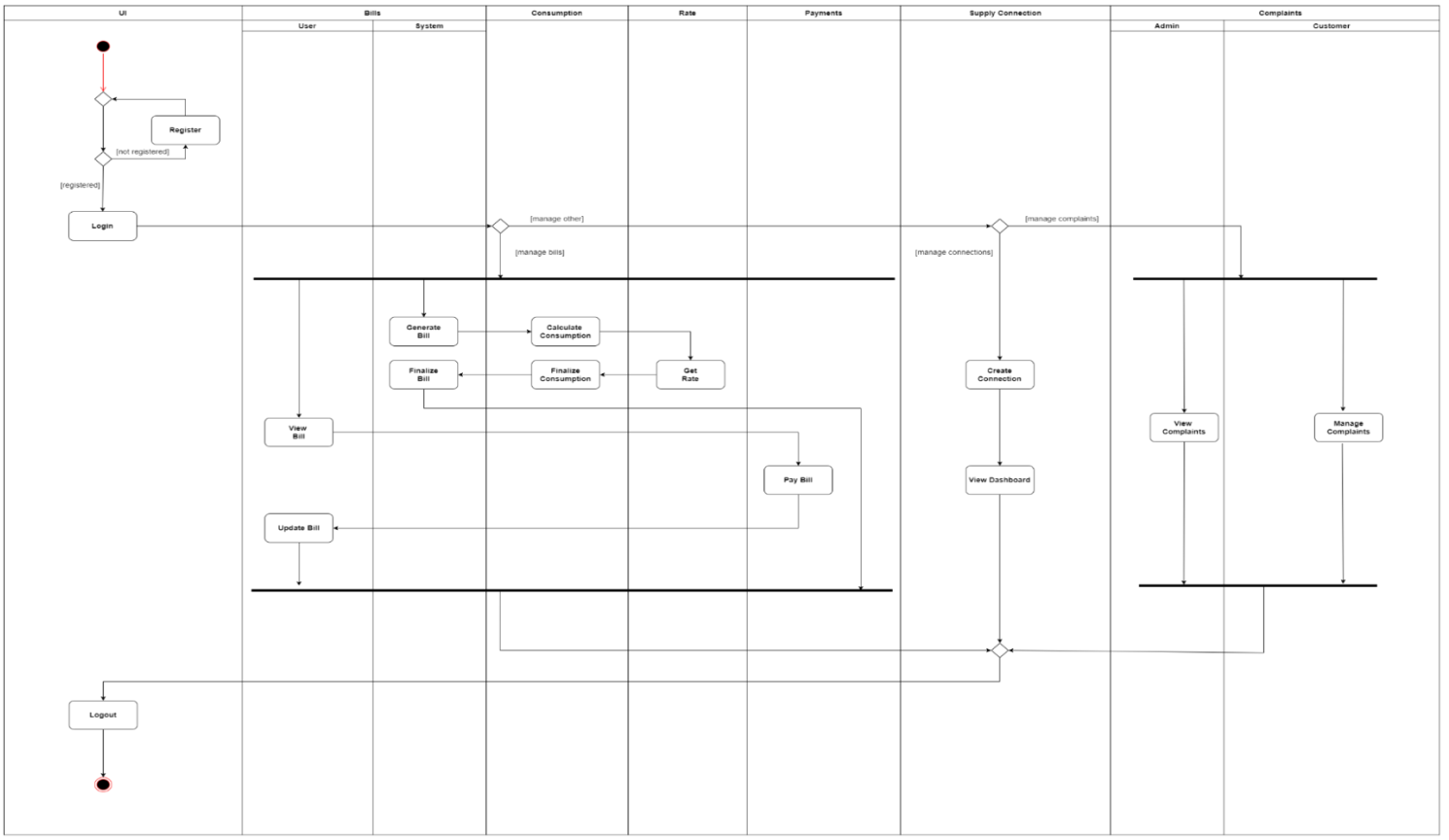
5. Stakeholder Analysis



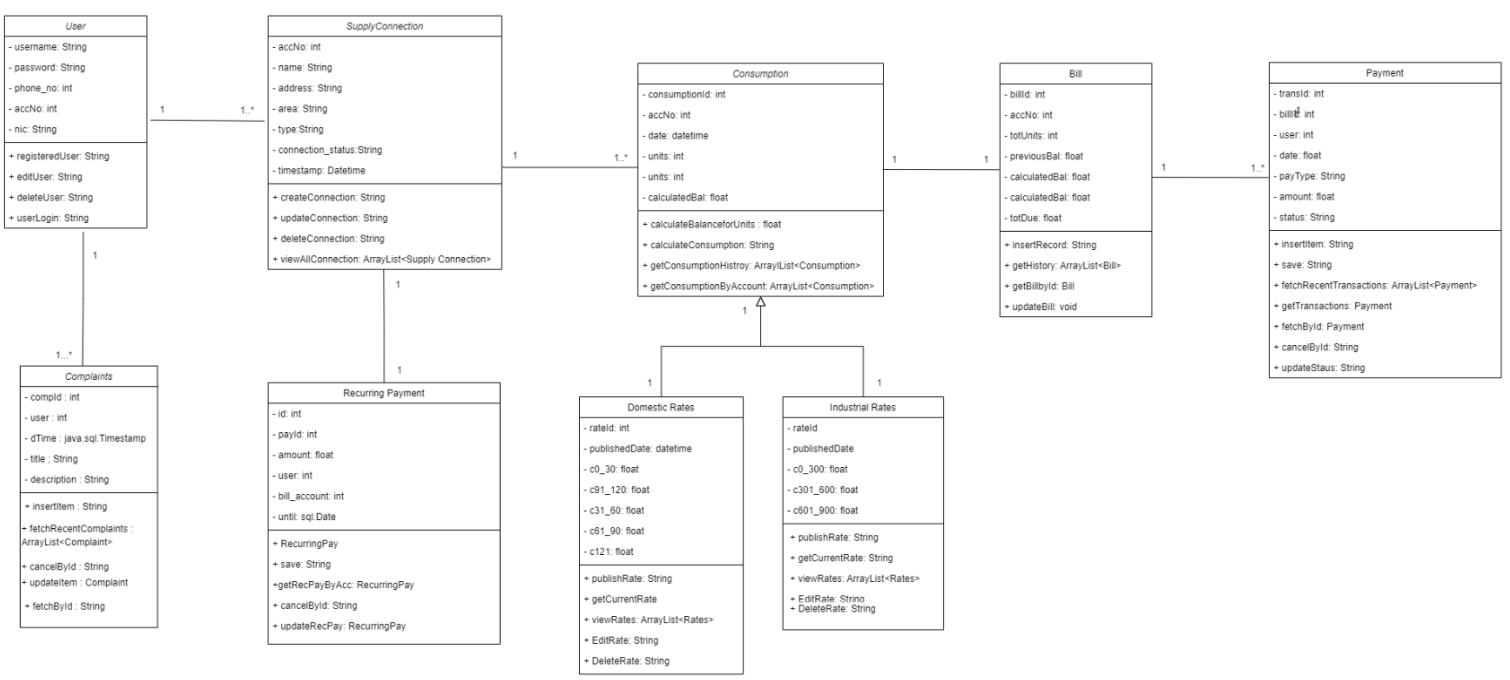
Here we have identified the stakeholders of the system

Overall Use Case Diagram

Overall Activity Diagram

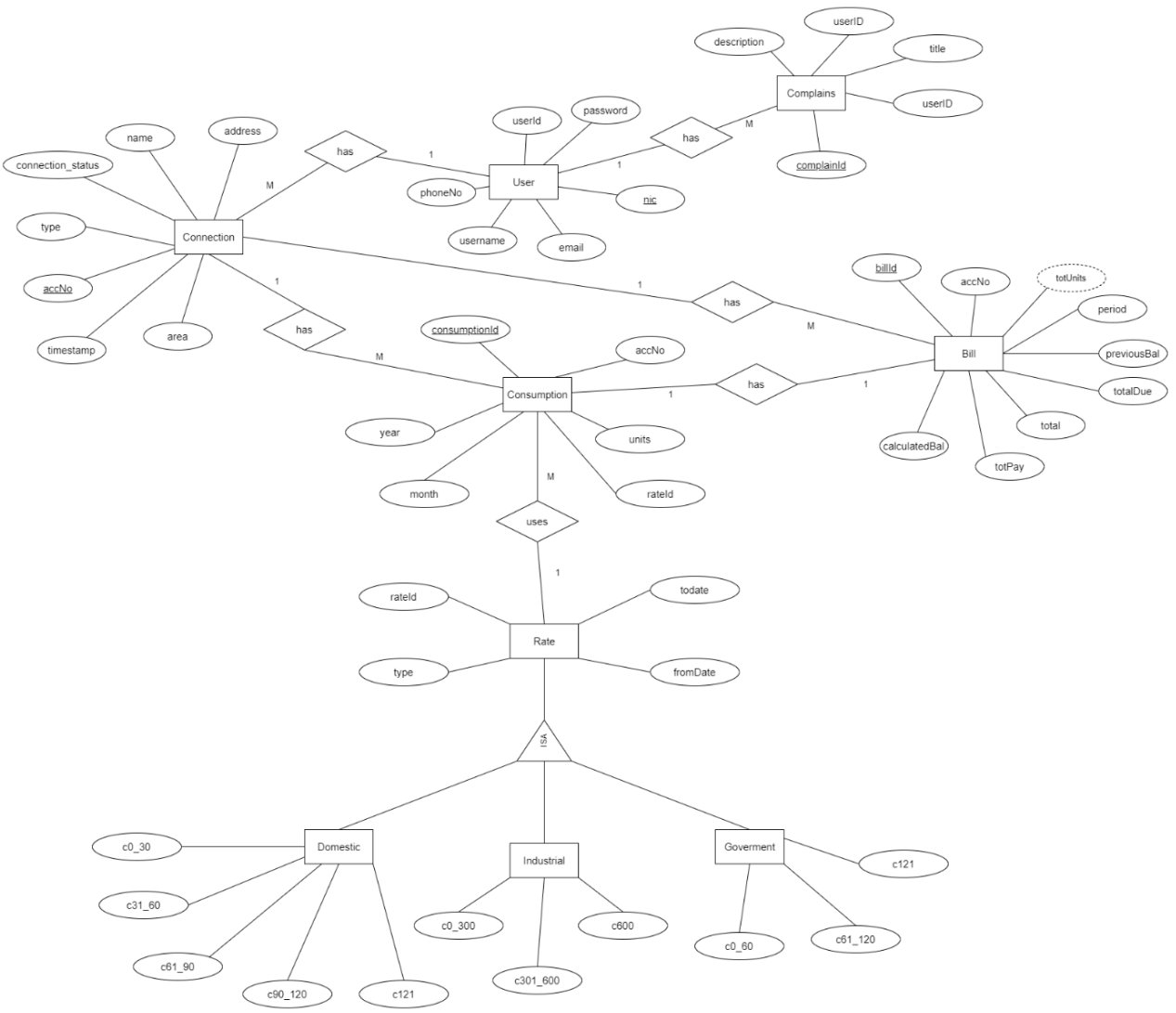


Overall Class Diagram

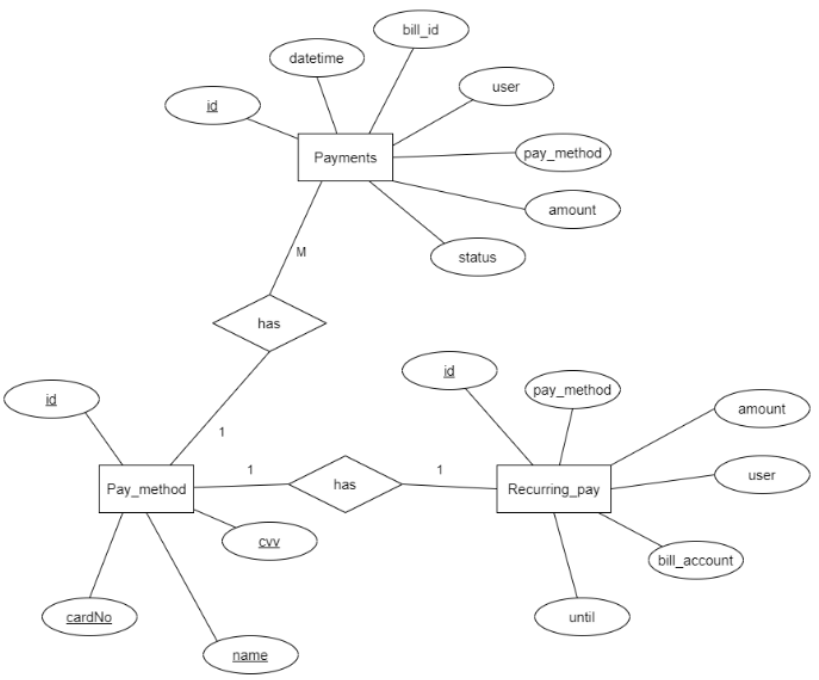


ER Diagrams

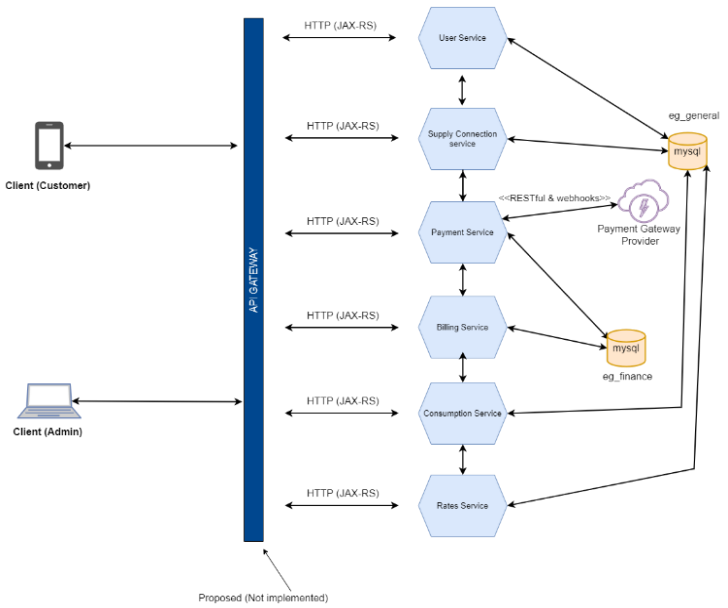
ER Diagram - General



ER Diagram - Finance



System Architecture Diagram



The Jersey library, a Jack-RS implementation is used to create RESTful endpoints for microservices. Here the API gateway act as a load balancer/distributer to increase the scalability and security of the application. Interservice communication is also established using RESTful services where jersey client is used for that purpose. Financial data is stored in a separate database to establish proper privacy and security. For payments, an external payment gateway provider is used and it has the ability to notify payments events through the supplied webhook.

1. **Project Timeline (Gantt Chart)**

Gannt Chart								
ACTIVITIES	**March**				**April**			
	week 1	week 2	week 3	week 4	week 1	week 2	week 3	week 4
Requirement Gathering								
Identifying core services								
Database Design								
Environment setup (developing)								
Developing the application								
Preparing the Final Document								

- Planning

- Development

Started date-15/03/2022
Proposed end date-26/04/2022

2. **Technology Selection – All Team members**

We as a team, discussed at the initial point of the project about the technologies which we will be using to develop the project.

IDE – Eclipse Enterprise

We used eclipse mainly because it has many tools which can be integrated and work on. It also has a good syntax prediction for Java when coding and also testing tools can also be integrated easily with the tool was the most comfortable IDE among the team.

Dependency management and build tool – Maven

Maven is good decency tool which covers many aspects of software development and a future proof tool. As this project has many Java dependencies it is easy to be handled

Version Control System (Decentralized) – Git

As we all worked on the project remotely the all the project, so we needed to integrate the project while staying remotely so we used GitHub

Documentation management Tool - SharePoint

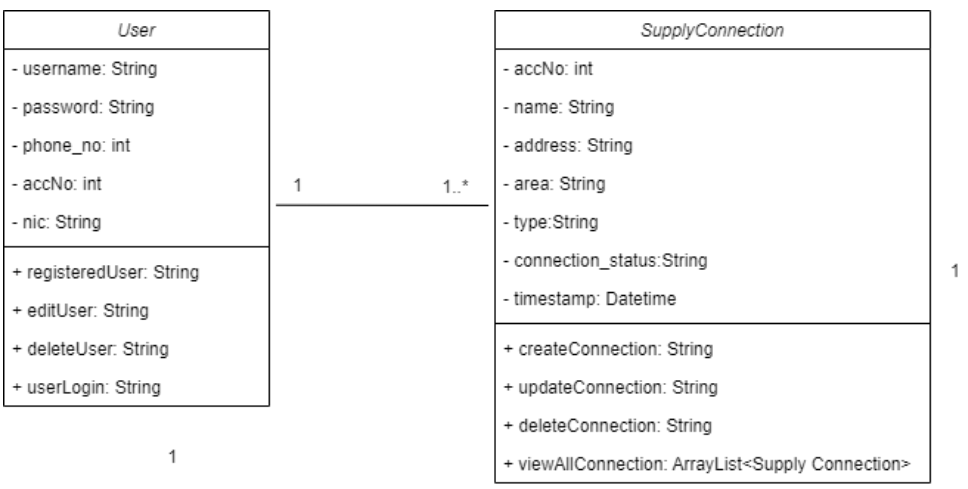
All documents needed to be managed online so we used a private SharePoint to manage the documentation and the diagrams which needed

3. Project Integration strategies

When developing the project we created separate projects for each service as we needed to run each service independently because it has its easy in the process of build and deployment process. These individual projects were combined using git to a remote repository in GitHub.

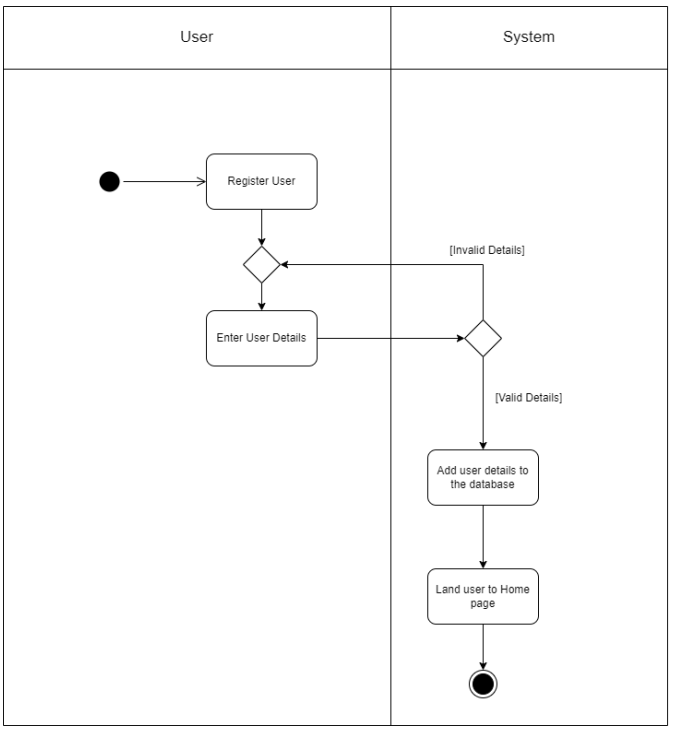
Individual contributions - B.W.E.K. Senarathna (IT19142838)

Class Diagram

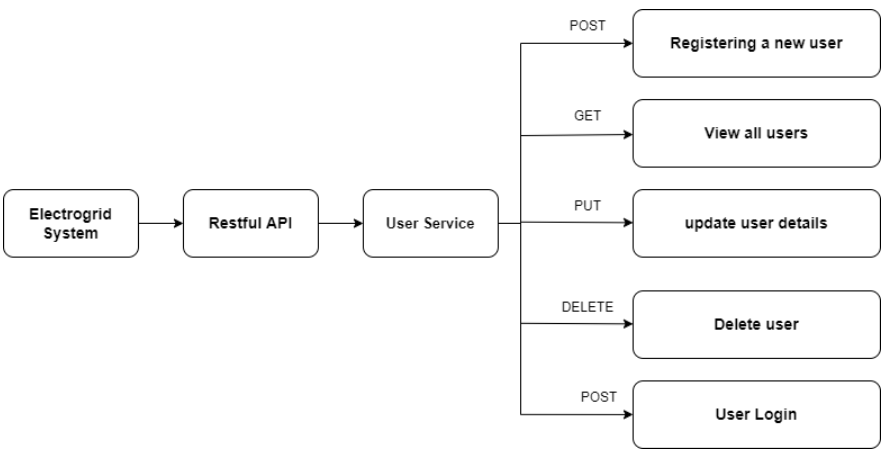


User service

Activity Diagram



API Diagram



Adding a new user to the system

Resource	Users
Request	POST
URL	http://localhost:8086/User/rest/user/registerUser

Input Data (JSON)	{
	"username": "randy",
	"email": "randy@localhost.com",
	"password": "randy123",
	"phone": "712346756",
	"accNo": "10024",
	"nic": "993462863V"
	}

Response (Plain Text)	User Registered successfully

Updating the details of a user

Resource	Users
Request	PUT
URL	http://localhost:8086/User/rest/user/editUser/1
Media	*MediaType.**APPLICATION_JSON***
Input Data (JSON)	{ "username": "esala", "email": "esala@localhost.com",

	"password": "talker124", "phone": "712343457", "accNo": "10023", "nic": "993452763V" }
Response (Plain Text)	User details Updated successfully

Deleting a specific user

Resource	Users
Request	DELETE
URL	http://localhost:8086/User/rest/user/deleteUser/2
Input Data (URl encoded)	2
Response(Plain Text)	User with the Account No :2 is Deleted successfully !

User Login

Resource	Users
Request	POST
URL	http://localhost:8086/User/rest/user/login

Input Data (JSON)	{
	"username": "randy",
	"password": "randy123"
	}

Response (Plain Text)	user authenticated

Testing cases and Results

The API end points are

Test Id	Test Case	Inputs	Expected Output	Actual Output	Status(Pass/Fail)

1	Register new user	{
		"username": "Sergio",
		"email": "sergio@localhost.com",
		"password": "sergio123",
		"phone": "711376756",
		"accNo": "10026",
		"nic": "993461853V"
		}
			Display “ User registered successfully”	User registered successfully	pass

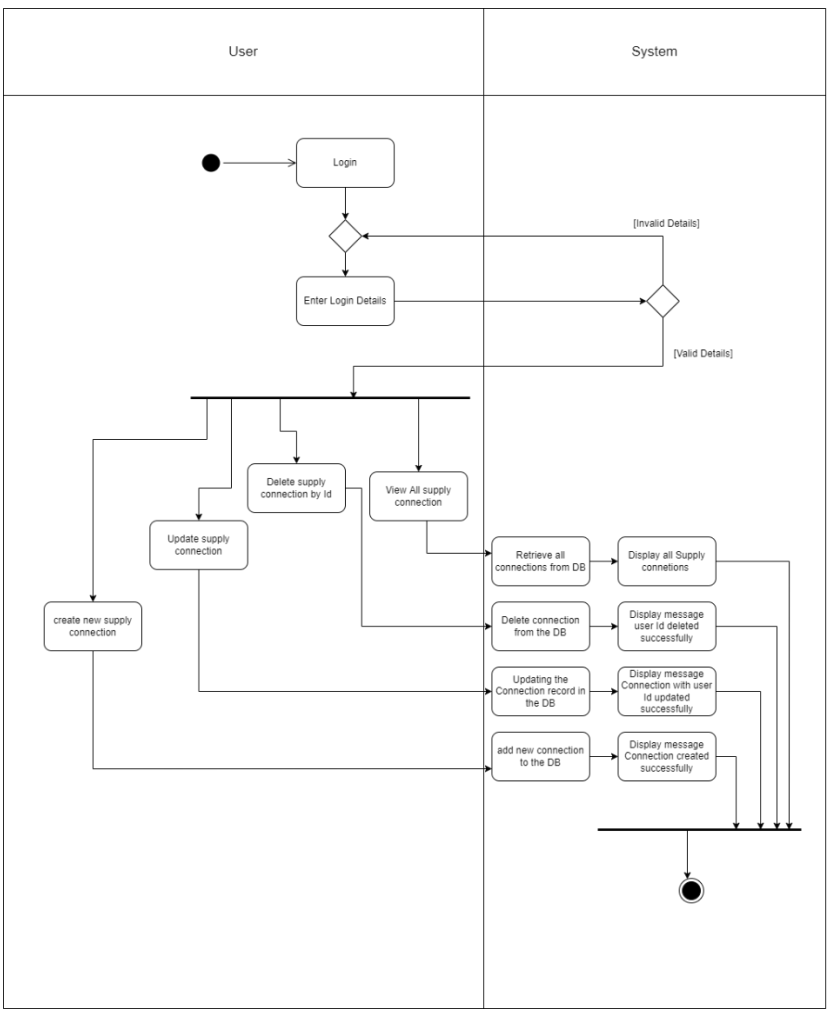
2	Update user details	{
		"username": "Sergio",
		"email": "sergio@localhost.com",
		"password": "sergio123",
		"phone": "711376756",
		"accNo": "10026",
		"nic": "993461853V"
		}
			Display “User details Updated successfully”	“User details Updated successfully”	pass

3	Delete user account	2	Display “User with the Account No. 2 Deleted Successfully”	“User with the Account No. 2 Deleted Successfully”	pass

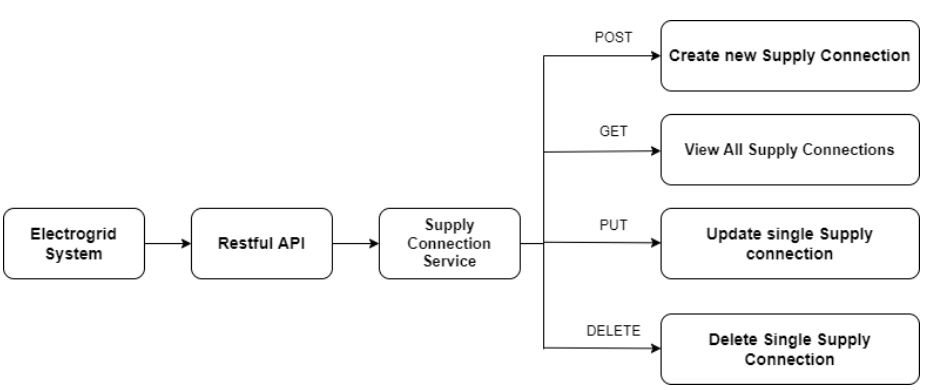
4	Login	{
		"username": "randy",
		"password": "randy123"
		}
			Display “User authenticated”	“User authenticated	pass

Supply Connection service

Activity Diagram



API diagram



Getting the details of all the Supply connections in the system

Resource	Supply Connection
Request	GET SupplyConnection/rest/supplyconnection/allconnections
URL	http://localhost:8086/SupplyConnection/rest/supplyconnection/allconnections
Input Data	No input
Response (JSON)	[{ "accNo": 10023, "name": "Esala Jay", "address": "12,troad,kandy", "area": "kandy", "type": "STD", "connection_status": "UP", "timestamp": "Apr 19, 2022 3:49:21 PM" }, { "accNo": 10026, "name": "Harry Coleman", "address": "17,Main Road,Kegalle", "area": "Kegalle", "type": "STD", "connection_status": "UP", "timestamp": "Apr 19, 2022 4:49:54 PM" }]

Create new Supply Connection

Resource	Supply Connection
Request	POST SupplyConnection/rest/supplyconnection/newconnection
URL	http://localhost:8086/SupplyConnection/rest/supplyconnection/newconnection

Input Data (JSON)	{
	"username": "esala",
	"email": "esala@localhost.com",
	"password": "talker124",
	"phone": "712343457",
	"accNo": "10023",
	"nic": "993452763V"
	}

Response (Plain Text)	Inserted successfully

Updating a specific supply connection

Resource	Users
Request	PUT User/rest/user/editUser/1
URL	http://localhost:8086/User/rest/user/editUser/1

Data (JSON)	{
	"accNo": "10029",
	"name": "Jerrt Filander",
	"address": "18,Tribrois Road,Kandy",
	"area": "Kandy",
	"type": "STD",
	"status": "UP"
	}

Response (Plain Text)	Connection Updated successfully

Deleting a specific supply connection

Resource	Supply Connection
Request	DELETE SupplyConnection/rest/supplyconnection/account/10029

URL	http://localhost:8086/SupplyConnection/rest/supplyconnection/account/10029
Input Data (URL Encoded)	10029
Response (Plain Text)	

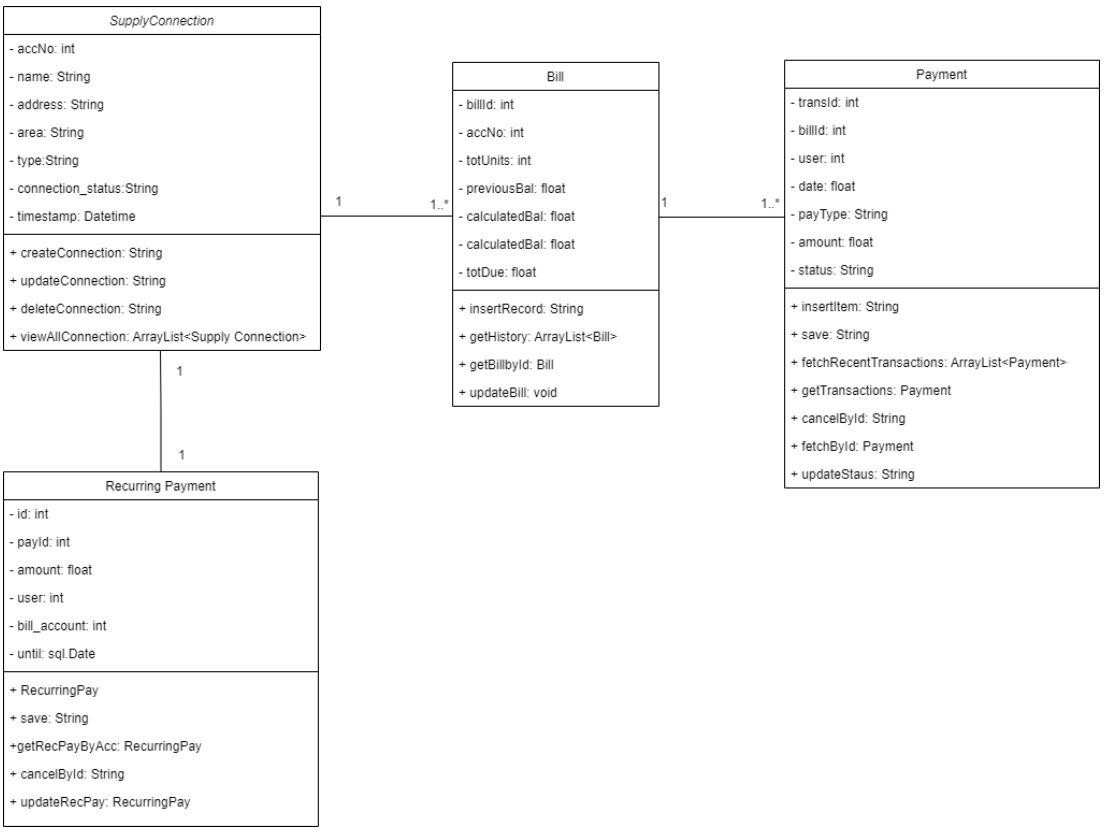
Testing cases and Results

The API end points are

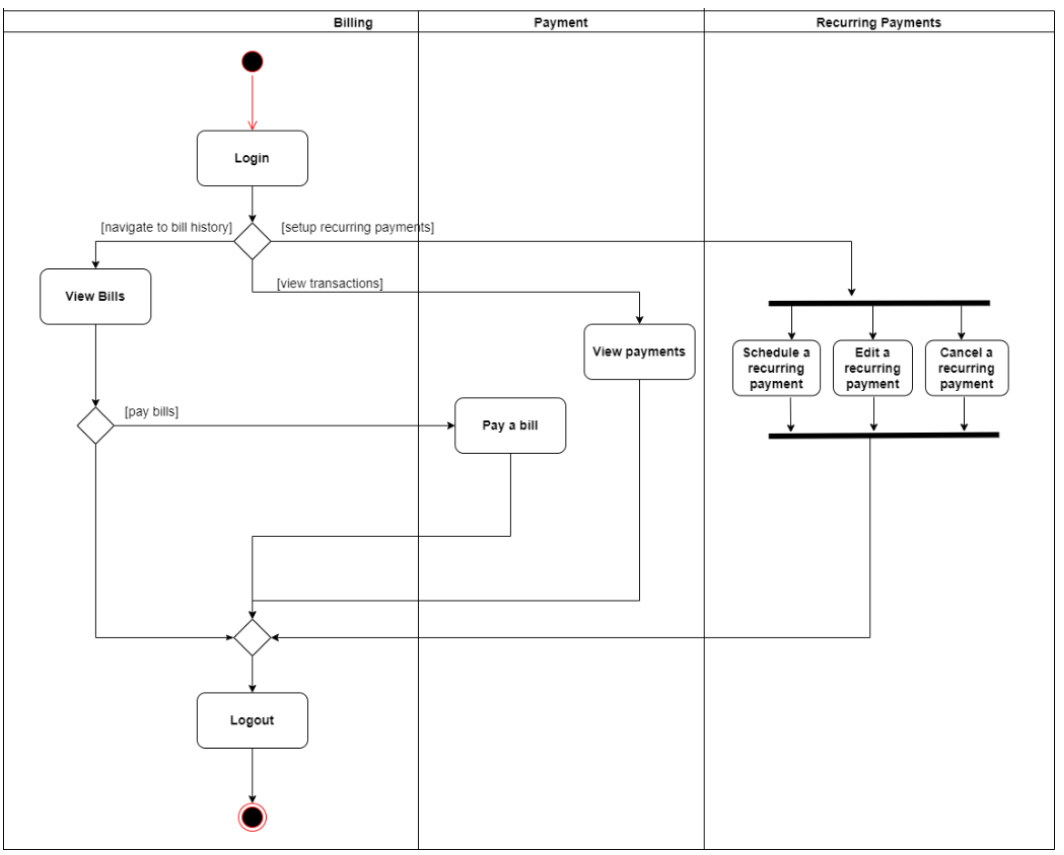
Test Id	Test Case	Inputs	Expected Output	Actual Output	Status(Pass/Fail)
1	Create new Supply Connection	{ "accNo": "10030", "name": "Saul Goodman", "address": "18,JRBay Road,Mathale", "area": "Mathale", "type": "STD", "status": "UP" }	Display " Inserted successfully"	Inserted successfully	pass
2	Update Supply Connection	{ "accNo": "10030", "name": "Saul Goodman", "address": "18,JRBay Road,Mathale", "area": "Mathale", "type": "STD", "status": "UP" }	Display "Connection updated successfully"	"Connection updated successfully"	pass
3	Delete user account	10029	Display "Connection with the Account No :10029 is Deleted successfully !"	"Connection with the Account No :10029 is Deleted successfully !"	pass
4	Get all supply connections	-	Display all supply connections	Display all supply connections	pass

Individual contributions – Udbhasa M M S (IT19188928)

Class diagram

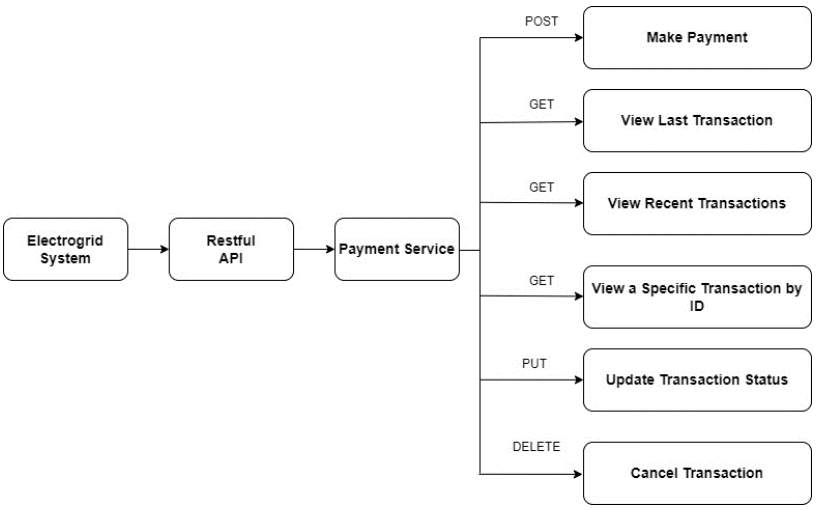


Activity diagram



Payment Service

API diagram



Make a payment using the bill id

Resource	Payment
Request	POST
URL	http://localhost:8080/PaymentService/rest/payment/pay

Input Data (JSON)	{
	"bill": "5",
	"user": "111",
	"pay_type": "MasterCard",
	"amount": "200"
	}

Response (JSON)	{
	"transId": 38,
	"dTime": "Apr 26, 2022 7:52:42 PM",
	"bill": 5,
	"user": 111,
	"pay_type": "MasterCard",
	"amount": 200.0,
	"status": "Processing"
	}

Get information on the last payment

Resource	Payment
Request	GET
URL	http://localhost:8080/PaymentService/rest/payment/pay
Input Data	No input

Response (JSON)	{
	"transId": 38,
	"dTime": "Apr 26, 2022 7:52:42 PM",
	"bill": 5,
	"user": 111,
	"pay_type": "MasterCard",
	"amount": 200.0,
	"status": "Done"
	}

Get recent payments of a user

Resource	Payment
Request	GET
URL	http://localhost:8080/PaymentService/rest/payment/recent/{id}
Input Data	Id : 444

Response (JSON)	[
	{
	"transId": 17,
	"dTime": "Apr 19, 2022 2:03:35 AM",
	"bill": 999,
	"user": 444,
	"pay_type": "MasterCard",
	"amount": 999.99,
	"status": "Processing"
	}
	]

Get payment by id

Resource	Payment
Request	GET
URL	http://localhost:8080/PaymentService/rest/payment/id/{id}
Input Data	Id : 37

Response (JSON)	{
	"transId": 37,
	"dTime": "Apr 24, 2022 2:59:50 PM",
	"bill": 5,
	"user": 111,
	"pay_type": "MasterCard",
	"amount": 200.0,
	"status": "Done"
	}

Cancel a payment by id

Resource	Payment
Request	DELETE
URL	http://localhost:8080/PaymentService/rest/payment/id/{id}
Input Data	Id : 37
Response (Plain Text)	Transaction id 37 terminated successfully !

Webhook for payment gateway

Resource	Payment
Request	PUT
URL	http://localhost:8080/PaymentService/rest/payment/gw_webhook

Input Data (JSON- By payment Gw)	{
	"id": "37",
	"status": "Done"
	}

Response (Plain Text with HTTP status code)	Operation Successful!

Recurring Payments

Create recurrent payments

Resource	Payment
Request	POST
URL	http://localhost:8080/PaymentService/rest/rec/record
Input Data (JSON)	{ "payId": 789, "amount":1800, "user":444,

	"bill_account":101, "until":"2022-12-2" }

Response (JSON echo)	{
	"id": 3,
	"payId": 789,
	"amount": 1800.0,
	"user": 444,
	"bill_account": 101,
	"until": "2022-12-02"
	}

Update recurrent payments

Resource	Payment
Request	PUT
URL	http://localhost:8080/PaymentService/rest/rec/record

Input Data (JSON)	{
	"id":2,
	"payId": 789,
	"amount":1100,
	"user":444,
	"bill_account":101,
	"until":"2022-12-25"
	}

Response (JSON echo)	{
	"id": 2,
	"payId": 789,
	"amount": 1100.0,
	"user": 444,
	"bill_account": 101,
	"until": "2022-12-25"
	}

Get recurrent payments using the connection number

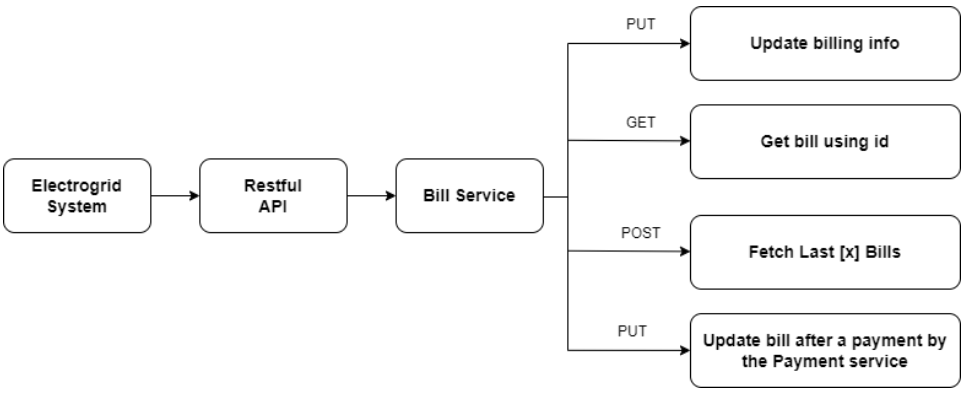
Resource	Payment
Request	GET
URL	http://localhost:8080/PaymentService/rest/rec/record/{id}
Input Data	Id:101

Response (JSON echo)	{
	"id": 2,
	"payId": 789,
	"amount": 1100.0,
	"user": 444,
	"bill_account": 101,
	"until": "2022-12-25"
	}

Remove recurrent payments using the id

Resource	Payment
Request	DELETE
URL	http://localhost:8080/PaymentService/rest/rec/record/{id}
Input Data	Id:101
Response (Plain Text)	Entry id 1 cancelled successfully !

Billing Service



Get last [x] bills

Resource	Billing
Request	POST
URL	http://localhost:8080/BillingService/rest/billing/history

Input Data (JSON)	{
	"accNo":"199",
	"length":"2"
	}

Response (JSON)	[{ "billId": 5, "accNo": 199, "totUnits": 320, "period": "2022-06-21", "previousBal": 7000.0, "calculatedBal": 4000.35, "totPay": 1900.0, "totalDue": 9100.3 }, { "billId": 3, "accNo": 199, "totUnits": 320, "period": "2022-05-20", "previousBal": 4000.35, "calculatedBal": 4000.35, "totPay": 6600.0, "totalDue": 1400.7 }]

Get bill using id

Resource	Billing
Request	GET
URL	http://localhost:8080/BillingService/rest/billing/bill/{id}
Input Data	Id:5

Response (JSON)	{
	"billId": 5,
	"accNo": 199,
	"totUnits": 320,
	"period": "2022-06-21",
	"previousBal": 7000.0,
	"calculatedBal": 4000.35,
	"totPay": 1900.0,
	"totalDue": 9100.3
	}

Update bill information

Resource	Billing
Request	PUT
URL	http://localhost:8080/BillingService/rest/billing/bill

Input Data (JSON)	{
	"billId":5,
	"amount":1500
	}

Response (JSON)	{
	"billId":5,
	"amount":1500
	}

Update bills after a payment completion (Interservice communication)

Resource	Billing
Request	POST
URL	http://localhost:8080/BillingService/rest/intercom/billpay

Input Data (JSON)	{
	"billId":3,
	"amount":500
	}

Response (Plain Text)	Updated Bill!

Testing cases and Results

Postman software is used to test the microservices after the integration. Since the system is integrated and services are running concurrently, inter-service communication could also be tested.

The API end points and test results are,

Test Id	**Test Case**	**Inputs**	**Expected Output**	**Actual Output**	**Status(Pass/Fail)**

1	Get last [x] bills	{
		"accNo":"199",
		"length":"2"
		}
			Display recent X no of Bills in JSON format	Recent 2 Bills in JSON format	pass

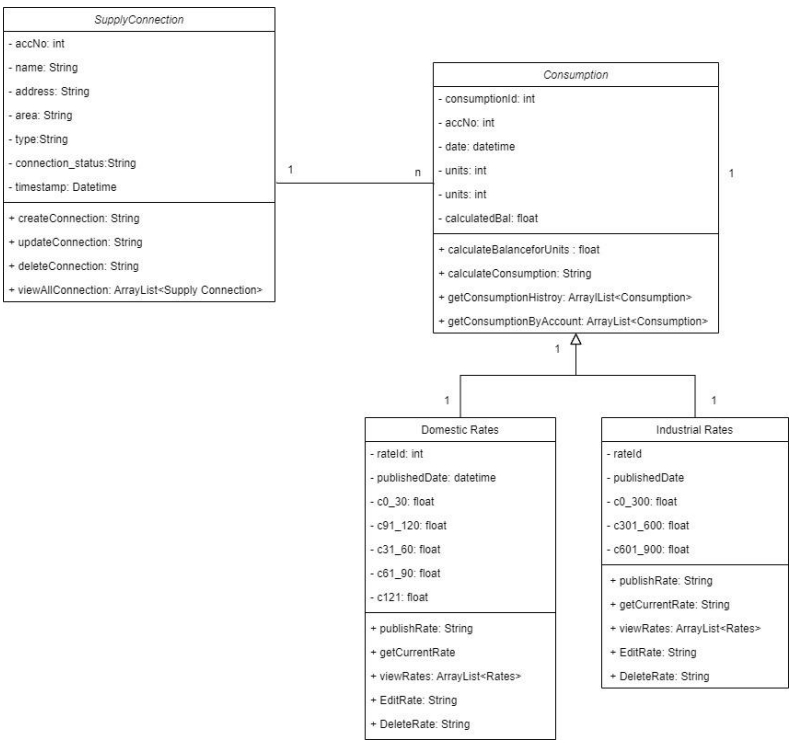
2	Update Bill	{
		"billId":3,
		"amount":500
		}
			Display “Updated Bill!”	Updated Bill!	pass

3	Cancel Bill	Id = 102	Display “Entry id {id} cancelled successfully !”	Entry id 102 cancelled successfully !”	pass

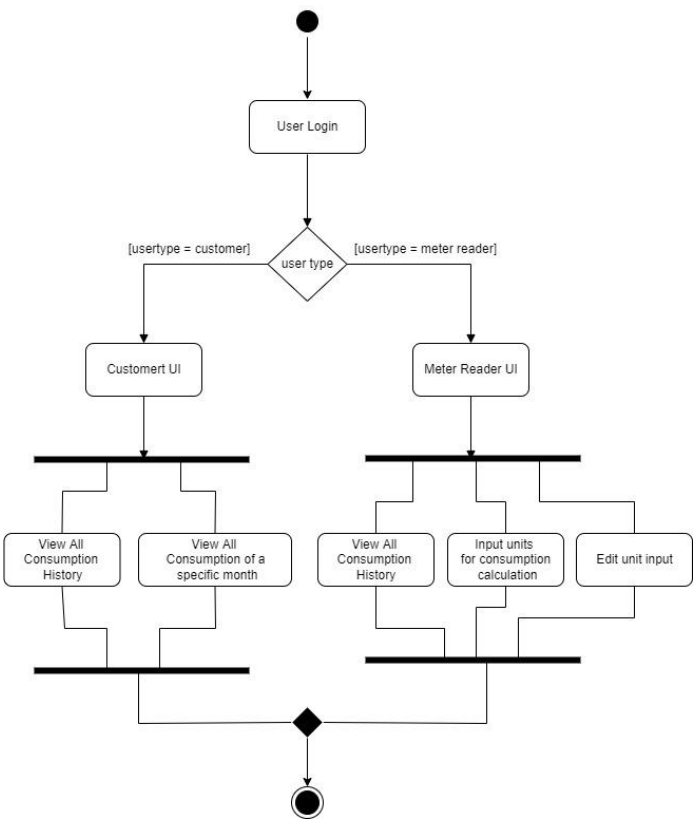
4	Create a recurrent payment	{
		"payId": 789,
		"amount":1800,
		"user":444,
		"bill_account":101,
		"until":"2022-12-2"
		}
			Display payment details in JSON format	Payment details in JSON format	pass

Individual contributions – Lelkada L L P S M (IT19001708)

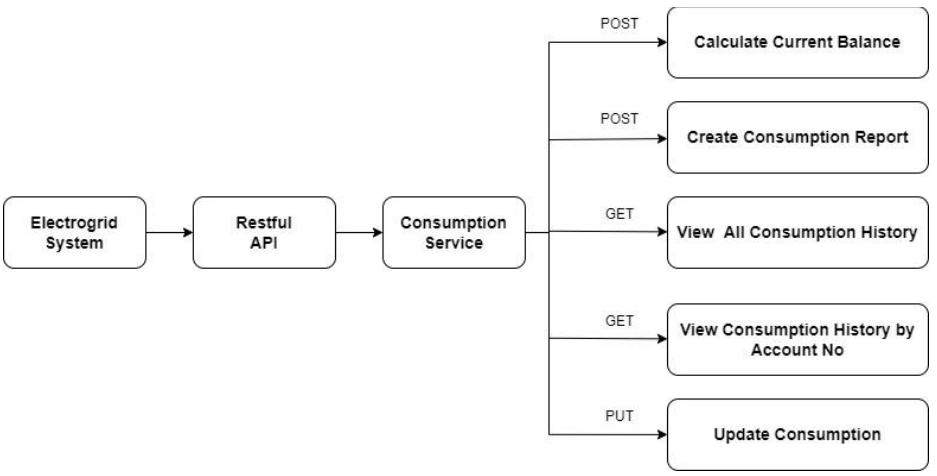
Class Diagram



Consumption Service



API diagram



View consumption history of all accounts

Resource	Consumption
Request	GET
URL	http://localhost:8080/ConsumptionService/rest/consumption/getall
Input Data	No input
Response (JSON)	[{ "consumptionId": 1233, "accNo": 49927993, "year": 2021, "month": 12, "units": 78, "calculatedBal": 975.84 }, { "consumptionId": 1243, "accNo": 652349934, "year": 2021, "month": 12, "units": 89, "calculatedBal": 1210.47 }]

View consumption of specific account

Resource	Consumption
Request	GET
URL	http://localhost:8080/ConsumptionService/rest/consumption/account/{acc}
Input Data(URl encoded)	652349934
Response (JSON)	[{ "consumptionId": 1233, "accNo": 652349934, "year": 2021, "month": 12, "units": 78, "calculatedBal": 975.84 }, { "consumptionId": 1533, "accNo": 652349934,

	"year": 2022, "month": 1, "units": 89, "calculatedBal": 1210.47 }]

Create new consumption record

Resource	Consumption
Request	POST
URL	http://localhost:8080/ConsumptionService/rest/consumption/addconsumption

Input Data (JSON)	{
	"accNo": 652349934,
	"year": 2022,
	"month": 2,
	"units": 89,
	}

Response (JSON)	{
	"consumptionId": 1793,
	"accNo": 652349934,
	"year": 2022,
	"month": 2,
	"units": 89,
	"calculatedBal": 1210.47
	}

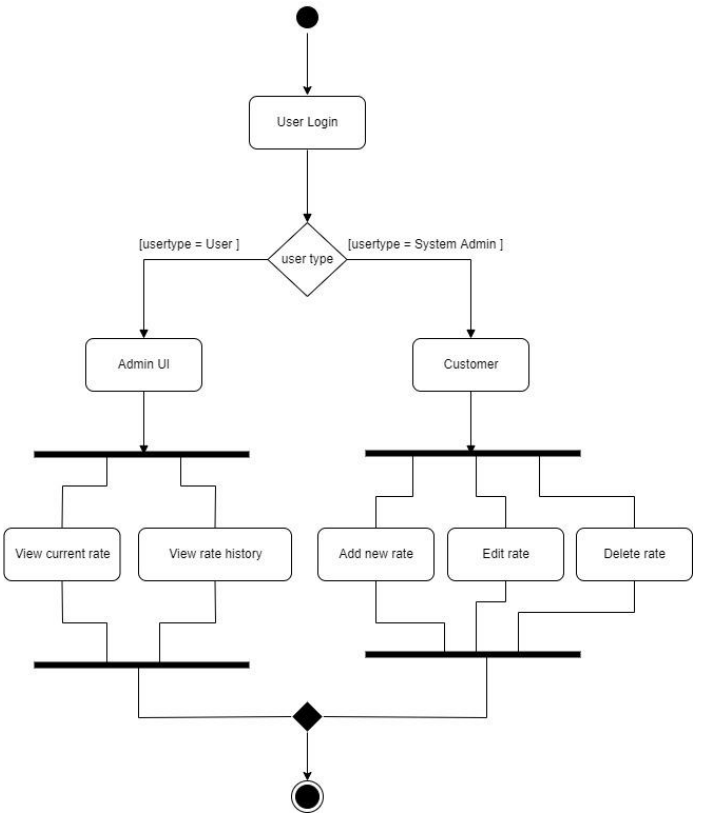
Update no of units

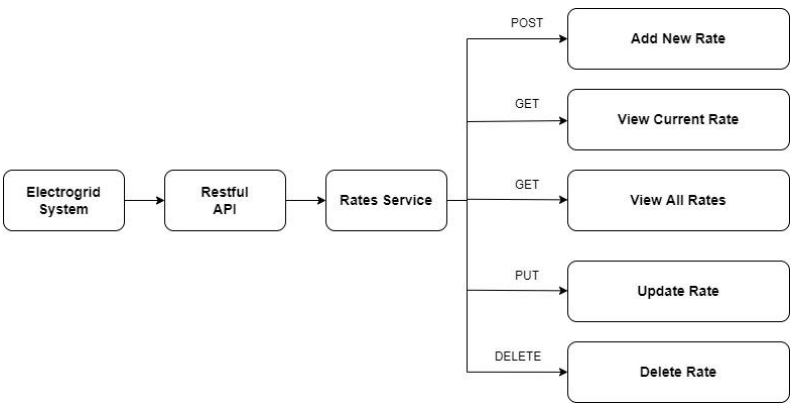
Resource	Consumption
Request	PUT
URL	http://localhost:8080/ConsumptionService/rest/consumption/updateunits

Input Data (JSON)	{
	"accNo": 652349934,
	"year": 2022,
	"month": 2,
	"units": 90,
	}

Response (JSON)	{
	"consumptionId": 1793,
	"accNo": 652349934,
	"year": 2022,
	"month": 2,
	"units": 90,
	"calculatedBal": 1231.8
	}

Rate Service





Get all rate records

Resource	Rates
Request	GET
URL	http://localhost:8080/ConsumptionService/rest/domestic/all
Input Data	No input
Response (JSON)	[{ "rateId": 12122, "year": 2021, "month": 6, "c0_30": 7.9, "c31_60": 12.23, "c61_90": 22.45, "c91_120": 33.9, "c121": 45.76 }, { "rateId": 12123, "year": 2022, "month": 1, "c0_30": 6.57, "c31_60": 14.53,

	"c61_90": 25.75, "c91_120": 38.56, "c121": 48.5 }]

Get current rate

Resource	Rates
Request	GET
URL	http://localhost:8080/ConsumptionService/rest/domestic/current
Input Data	No input
Response (JSON)	{ "rateId": 12122, "year": 2021, "month": 6, "c0_30": 7.9, "c31_60": 12.23, "c61_90": 22.45, "c91_120": 33.9, "c121": 45.76 }

Add new Rate record

Resource	Rates
Request	POST
URL	http://localhost:8080/ConsumptionService/rest/domestic/addrate

Input Data (JSON)	{
	"year": 2021,
	"month": 6,
	"c0_30": 7.9,
	"c31_60": 12.23,
	"c61_90": 22.45,
	"c91_120": 33.9,
	"c121": 45.76
	}

Response (Plain Text)	"Successfully created new Rate record"

Delete Rate record

Resource	Rates
Request	PUT
URL	http://localhost:8080/ConsumptionService/rest/domestic/deleterate/{id}
Input Data (URl encoded)	12122
Response (Plain Text)	"Successfully deleted Rate record"

Testing cases and Results

The API end points are

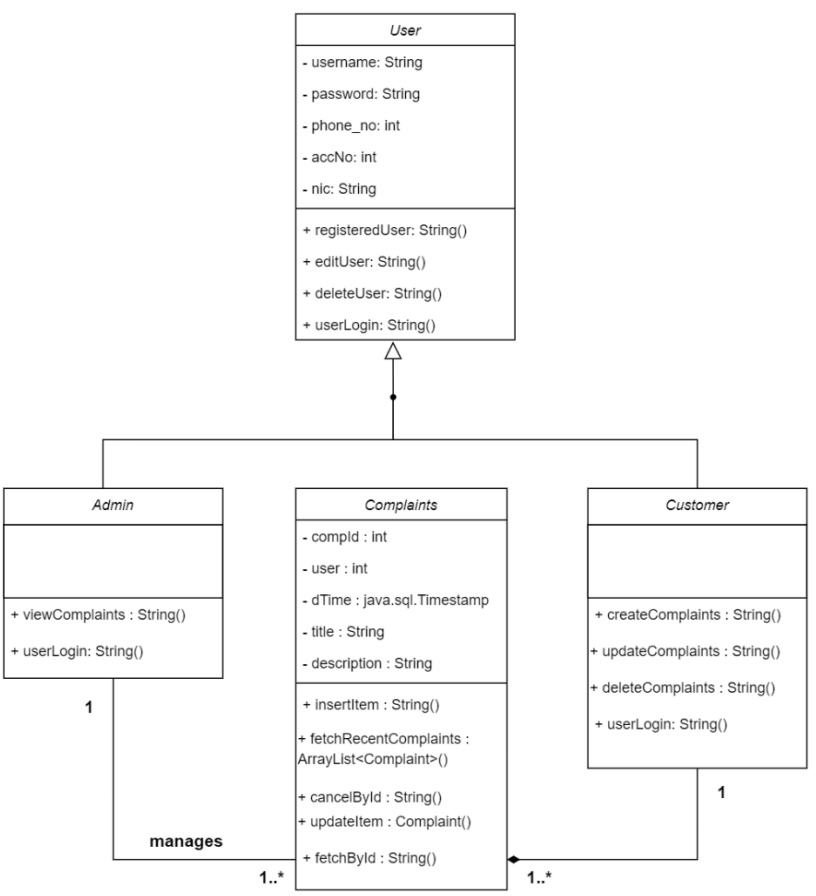
Test Id	Test Case	Inputs	Expected Output	Actual Output	Status(Pass/Fail)
1	Create new Consumption	{ "accNo": 49927993, "year": 2021, "month": 12, "units": 78, },	Display database entries in JSON format	Table entries in JSON format	pass

2	Update Consumption	{ "accNo": 49927993, "year": 2021, "month": 12, "units": 78, },	Display database entries with updated calculatedBal in JSON format	Updated table entries in JSON format	pass
3	Delete Rate	122144	Display “Rate deleted successfully”	Rate deleted successfully	pass
4	Get current rate	-	Display most recent Rate in JSON format	Most recent Rate in JSON format	pass

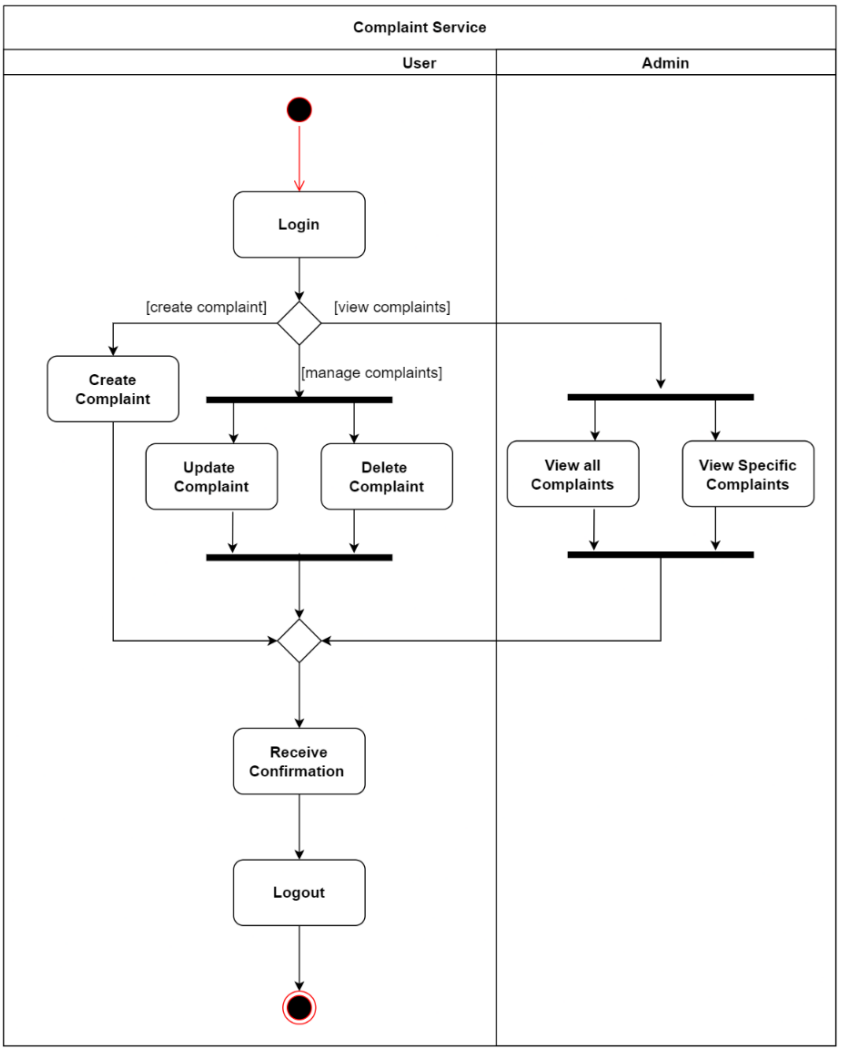
Individual contributions – Udupihilla U.W.D.G.B. (IT19956954)

Complaints Service

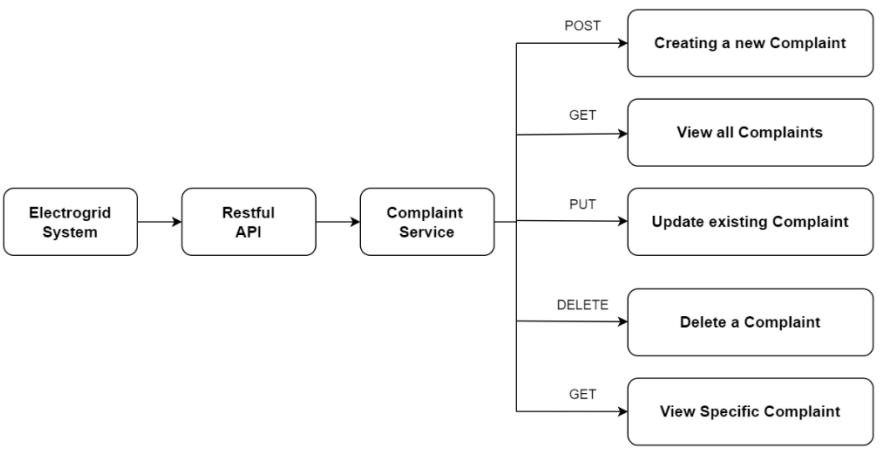
Class diagram



Activity diagram



API diagram



Viewing all recent complaints

Resource	Complaints
Request	GET ComplaintService/rest/complaint/recent/
URL	http://localhost:8080/ComplaintService/rest/complaint/recent/
Input Data	No inputs
Response	{ "compId": 1, "user": 12, "dTime": 1650648257000, "title": "Power Failure", "description": "updated description" }, {

	"compId": 2, "user": 19, "dTime": 1650818998000, "title": "Electricity Interruption", "description": "Suddenly, there was no electricity" }

Viewing a specific complaint

Resource	Complaints
Request	GET ComplaintService/rest/complaint/view/1
URL	http://localhost:8080/ComplaintService/rest/complaint/view/1
Input Data (JSON)	1

Response (JSON)	{
	"compId": 1,
	"user": 12,
	"dTime": 1650648257000,
	"title": "Power Failure",
	"description": "updated description"
	}

Creating a new complaint

Resource	Complaints
Request	POST ComplaintService/rest/complaint/new
URL	http://localhost:8080/ComplaintService/rest/complaint/new

Input Data (JSON)	{
	"user": 21,
	"title": "Test title",
	"description": "test data!"
	}

Response (Plain Text)	Created successfully!

Deleting a complaint

Resource	Complaints
Request	DELETE ComplaintService/rest/complaint/delete/9
URL	http://localhost:8080/ComplaintService/rest/complaint/delete/9
Input Data	9
Response (Plain Text)	Complaint id 9 terminated successfully !

Updating a complaint

Resource	Complaints
Request	PUT ComplaintService/rest/complaint/compUpdate
URL	http://localhost:8080/ComplaintService/rest/complaint/compUpdate

Data (JSON)	{
	"compId": 1,
	"description": "updated description "
	}

Response (Plain Text)	Complaint id 1 updated successfully !

Testing cases and Results

Test Id	Test Case	Inputs	Expected Output	Actual Output	Status(Pass/ Fail)

1	Create new complaint	{
		"user": 21,
		"title": "Test title",
		"description": "test data!"
		}
			Display
			“ Created successfully!”
				Created successfully!	pass

2	Update complaint	{
		"compId": 1,
		"description": "updated description "
		}
			Display
			“Complaint id 1 updated successfully!”
				Complaint id 1 updated successfully!	pass

3	Delete complaint	9	Display “Complaint id 9 terminated successfully!”	Complaint id 9 terminated successfully!	pass

4	View a complaint	1	Display JSON String.	{
				"compId": 1,
				"user": 12,
				"dTime": 1650648257000,
				"title": "Power Failure",
				"description": "updated description"
				}
					pass

References

[1] Jersey-Bundle 1.19.4 Documentation

https://javadoc.io/doc/com.sun.jersey/jersey-bundle/latest/index.html

[2] NGINX, “API Gateway”, [Online]. Available:

What is an API Gateway? | NGINX Learning

[3] Building RESTful services with Jax-RS

https://docs.oracle.com/javaee/6/tutorial/doc/giepu.html

[4] Application Developer’s Guide – Tomcat-9.0-doc

https://tomcat.apache.org/tomcat-9.0-doc/appdev/installation.html

[5] Introduction To Java Servlets and Its Life-Cycle

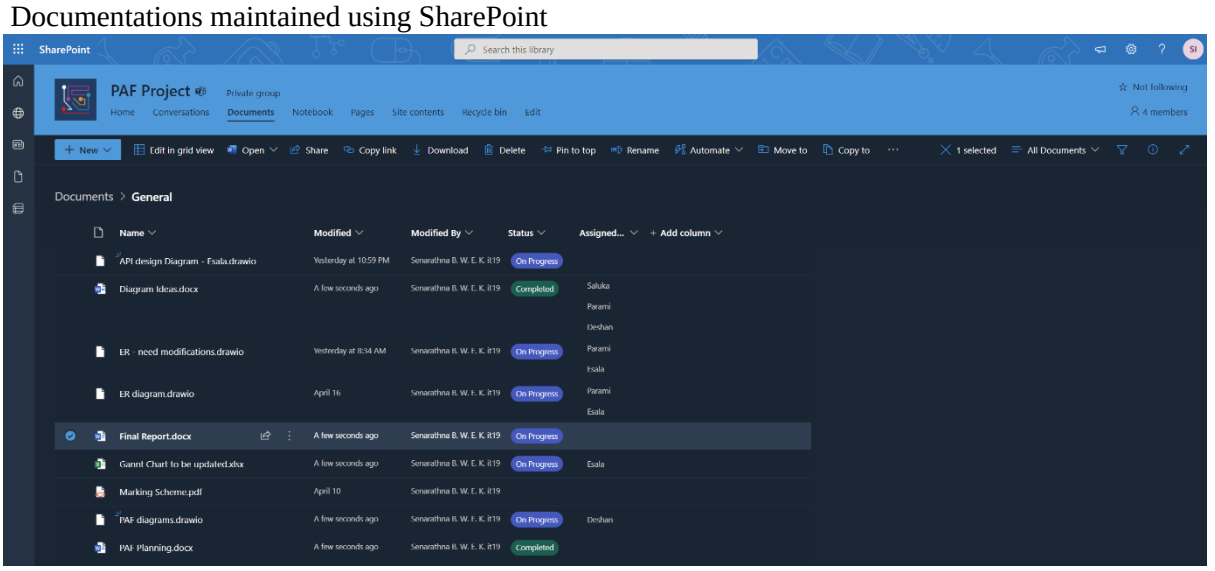
https://www.simplilearn.com/tutorials/java-tutorial/java-servlets

[6] Postman Documentation

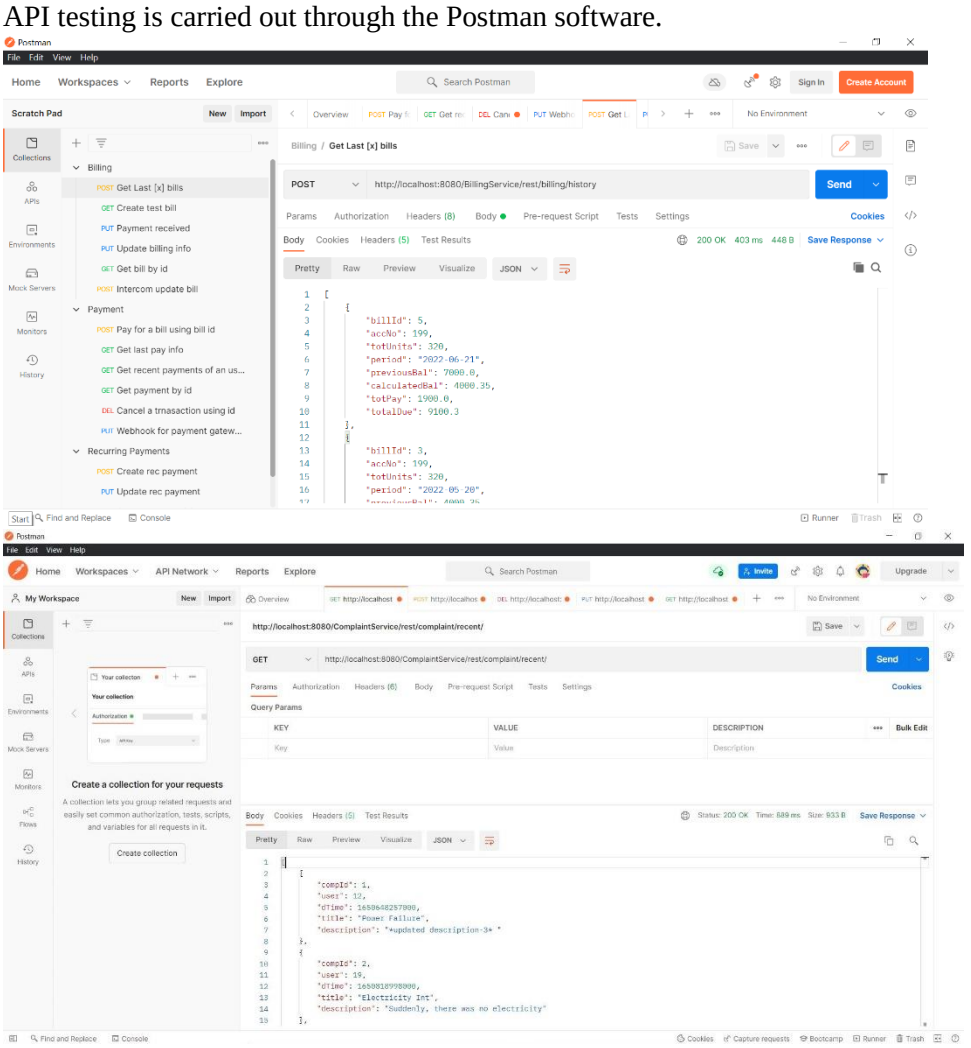
https://learning.postman.com/docs/getting-started/settings/

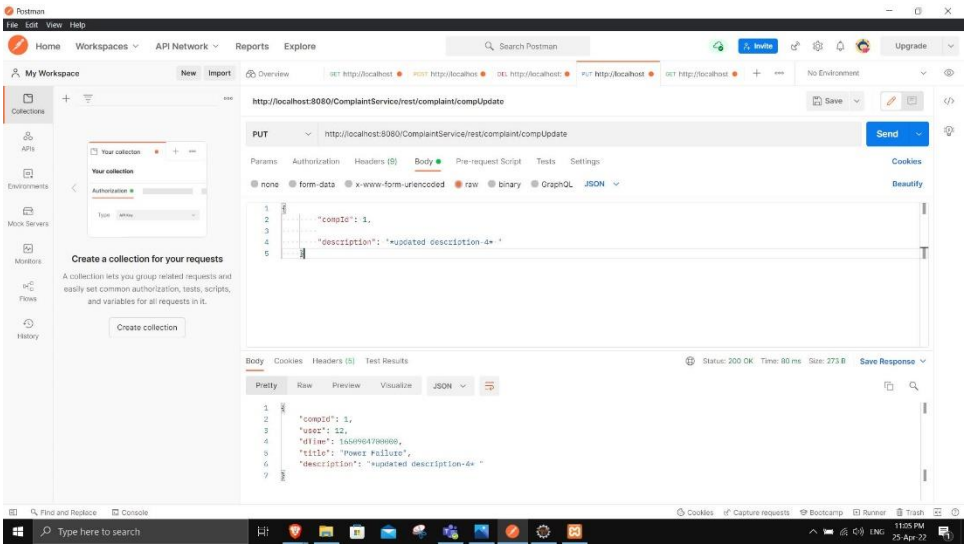
APPENDIX

Documentations maintained using SharePoint



API testing is carried out through the Postman software.





Github repository