System Documentation

for

Electricity Billing System

Version 3.0

Tutorial Section: T10L

Group No.: 2

Mohammed Yousef Mohammed Abdulkarem	**1221305727**
Mohammed Aamena Mohammed Abdulkarem	**1221305728**
Muhammad Faiz bin Ilyasa	**1221309435**
Chua Cheng Zong	**1211109173**

Date: **12th February 2025**

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Mohammed Yousef Mohammed Abdulkarem Mohammed Amena Mohammed Abdulkarem Muhammad Faiz bin Ilyasa Chua Cheng Zong	Generated the Project Group & Plan, System Overview, Scenario-Based Models, Class Models, and Behavioural & Flow Models.	08/12/2024
2.0	Mohammed Yousef Mohammed Abdulkarem Mohammed Amena Mohammed Abdulkarem Muhammad Faiz bin Ilyasa Chua Cheng Zong	Created Data Design Diagrams, Architecture Design Diagrams, Interface Design Diagrams, Component Design Diagrams and Deployment Design Diagrams.	12/01/2025
3.0	Mohammed Yousef Mohammed Abdulkarem Mohammed Amena Mohammed Abdulkarem Muhammad Faiz bin Ilyasa Chua Cheng Zong	Implement the specification to an actual web app.	12/02/2025

1 Project Management

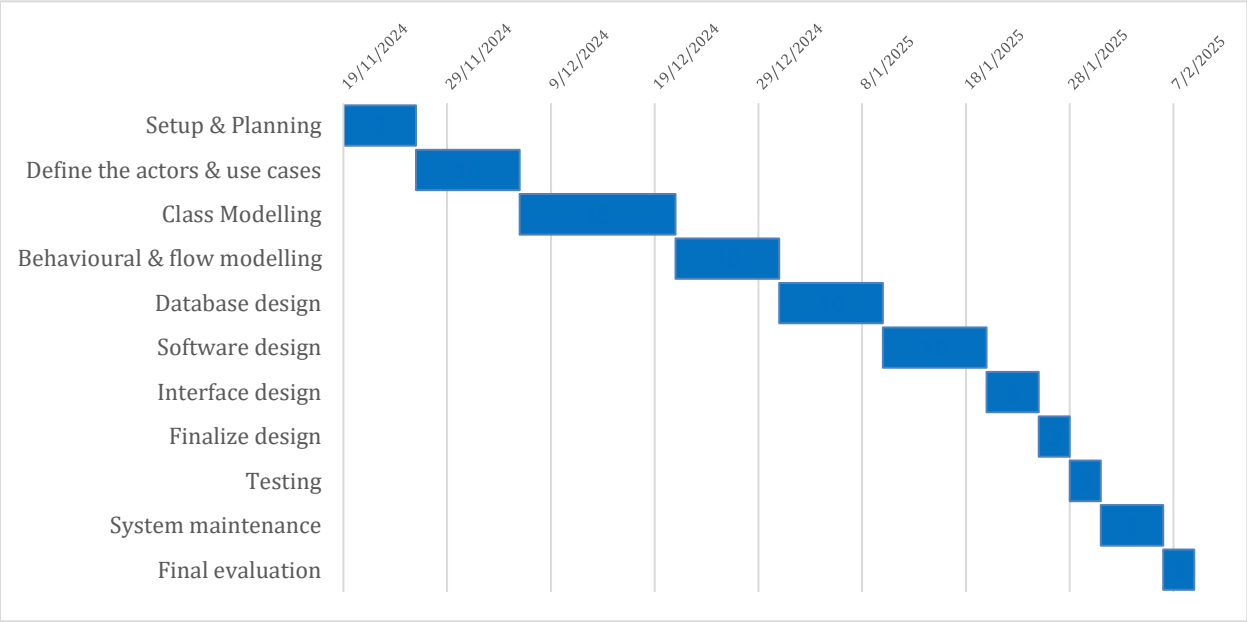
1.1 Team Members

Name	Actor/Processes
Mohammed Yousef Mohammed Abdulkarem	**Actor:** Customer **Processes:** System Registration, Log in to the System, Reset Password.
Mohammed Aamena Mohammed Abdulkarem	**Actor:** Support Admin **Processes:** View Bills Details, Submit and Resolve Customer Issues, Submit Feedback and Analysis.
Muhammad Faiz bin Ilyasa	**Actor:** Staff **Processes:** Manage Customer Accounts, Track Overdue Bills, Usage Monitoring.
Chua Cheng Zong	**Actor:** Utility Provider **Processes:** Track Overdue Bills, Update Meter Reading and Tariffs, Generate Monthly Bills.

1.2 Problem statement

Current approach of managing electricity lacks in terms of efficiency and accuracy, which could cause errors in meter readings, delays, and ineffective tracking of payment activities. This can lead to customers facing difficulties in paying their monthly bills, while the system's staff struggle in monitoring system performance and provide support to troubled customers. An automated electricity bill system with improved functionalities ensures user experience is enhanced while allowing for easier management for the staff.

1.3 Project Plan



2 System Overview

2.1 Description

The Electricity Billing System is a comprehensive solution aimed at improving the management of electricity services. It facilitates effective communication and coordination among key stakeholders, including customers, utility providers, support administrators, and staff. The system ensures secure user registration and authentication, accurate billing calculations, smooth payment processing, and comprehensive customer support. Additionally, it features a robust feedback mechanism to continually improve service quality and ensure customer satisfaction. The system prioritizes data integrity, security, and operational efficiency, providing an intuitive interface that simplifies service management, minimizes manual efforts, and ensures reliable performance. Utilizing advanced technology, the Electricity Billing System sets a new standard for operational excellence, offering a flexible solution that evolves to meet the dynamic needs of both utility providers and their customers.

2.2 Actors

The system integrates multiple roles, each with specific responsibilities to ensure smooth operation. Customers can create accounts, log in securely, reset passwords, and update their personal information. They can view detailed billing information, make payments using their preferred methods, receive receipts, provide feedback, and report any issues or concerns.

Utility providers manage critical operational tasks, such as logging into the system to update meter readings, adjusting tariff rates in response to changes in tax or regulatory policies, generating accurate monthly bills, and tracking overdue payments. When overdue payments exceed the given grace period, utility providers will set penalty fee for the associated bill and coordinate with the staff to notify the customer of the final payment deadline and initiate service disconnection if necessary.

Support administrators prioritize customer satisfaction by resolving issues related to billing, payments, and service concerns. They review billing data to identify discrepancies, analyse customer feedback to identify areas for system improvements, and respond promptly to customer inquiries, ensuring efficient issue resolution and maintaining a high level of service quality.

Staff members manage administrative and monitoring functions, such as tracking overdue bills for verification and timely payment, analysing electricity usage trends to assess consumption patterns, and maintaining customer account records to ensure accuracy and operational efficiency.

This well-structured system ensures that all participants fulfil their roles effectively, creating a seamless and transparent process for electricity service delivery. By consolidating all operations

into one platform, the Electricity Billing System fosters efficient communication, accurate billing, and improved service delivery, making it an essential tool for the electricity sector.

2.3 Assumptions and Dependencies

User Access and Connectivity

- Users will have access to devices capable of running modern web browsers and will have stable internet connectivity.
- Customers will provide valid email addresses, meter number and mobile numbers for notifications and account management.

Data Availability and Accuracy

- Utility providers will provide accurate and up-to-date meter readings and tariffs for accurate billing.
- Customers are expected to provide valid personal information.
- The system will automatically check the input data for validation, such as meter readings and customer information, to minimize errors in billing.

Scalability Requirements

The system will support up to 1000 users initially. As the user base grows, scaling the system will require infrastructure upgrades, such as database optimization and increased server capacity, to ensure continued performance and reliability.

Compliance with Standards

The system will comply with relevant legal and industry regulations, including:

- Data Privacy Laws: The system will follow GDPR to ensures that the system is in compliance with data protection regulations, protecting personal data and the privacy rights of users. This includes secure data storage, processing, and consent management to avoid legal violations.
- Payment Security Standards: Ensures the realization of compliance with the Payment Card Industry Data Security Standard (PCI DSS) for the safe handling of credit card information, helping to protect payment data from fraud or breaches during transactions.
- Utility Service Requirements: Ensures that the system meets regional standards and regulations of electricity services provided, regarding billing accuracy, service reliability, and customer protection.

Dependencies:

Integration with Payment Gateways

The system payment functionality relies on secure and reliable third-party payment services to process transactions efficiently while maintaining industry standards for data protection.

Accurate Tariff and Meter Data

The system's billing accuracy is dependent on regular updates of tariff rates and meter readings provided by utility providers. Any delays or inaccuracies in this data will directly affect billing and payment processes.

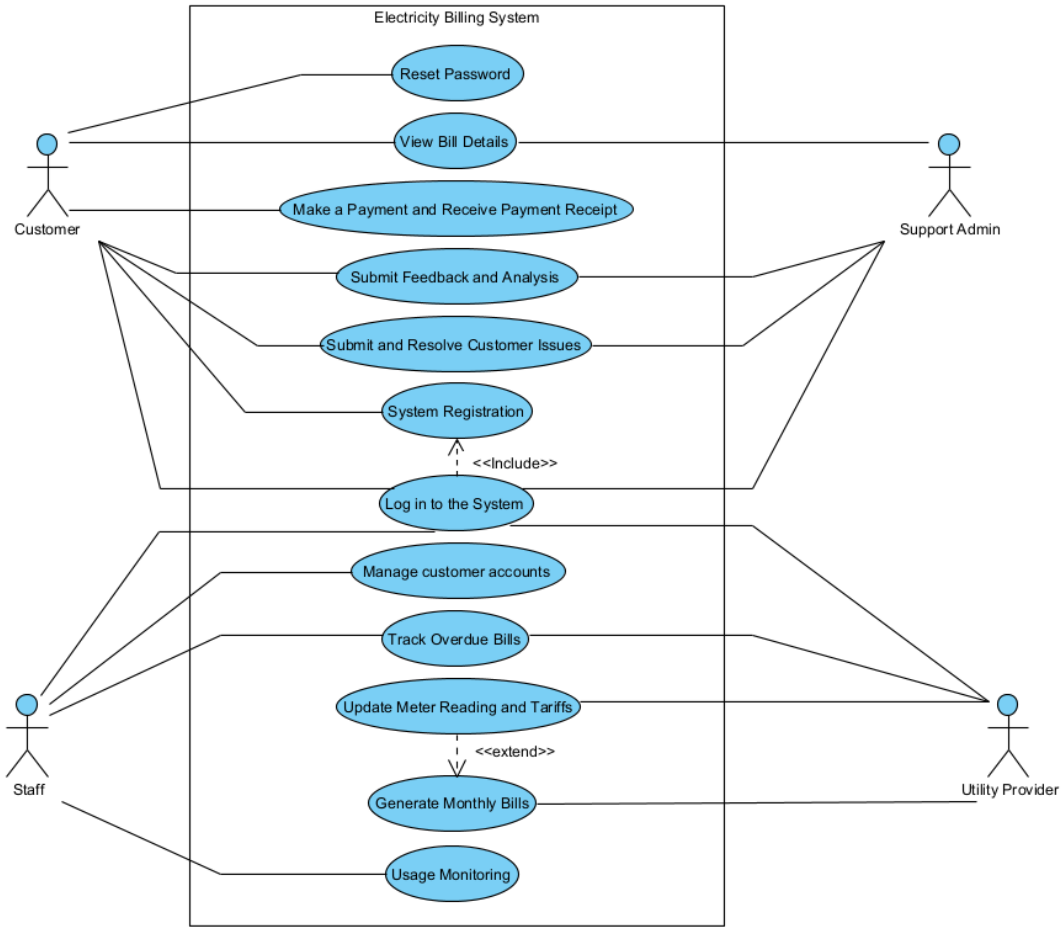
Database Management System

Microsoft Access, a robust and scalable relational database system, will manage all system data effectively. It ensures secure storage, fast retrieval, and the capability to scale as the system grows, supporting seamless operations and reliability.

Ongoing Maintenance and Support

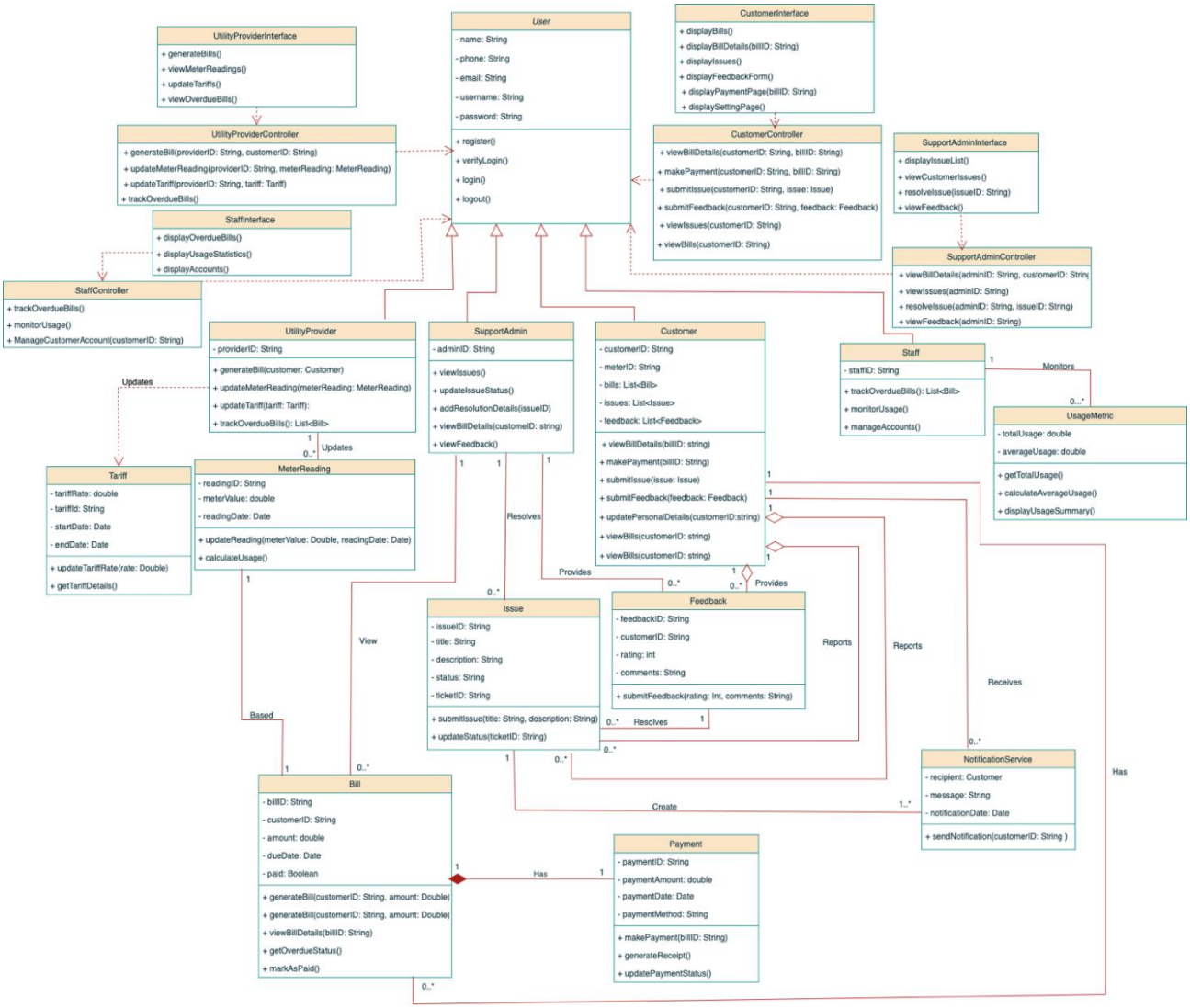
Continuous system updates, performance monitoring, and regular maintenance will be essential to address emerging needs, improve functionality and support best system performance.

2.4 Use Case Diagram



3 Requirements

3.1 Class Diagrams / ERD

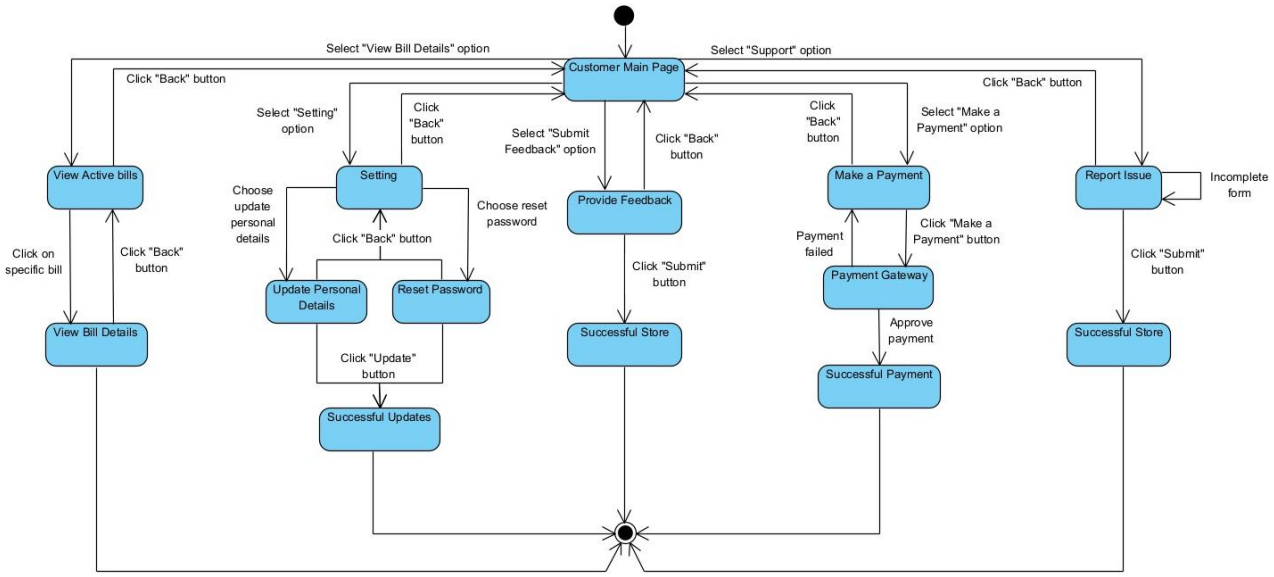


3.2 State Diagrams

3.2.1 Customer

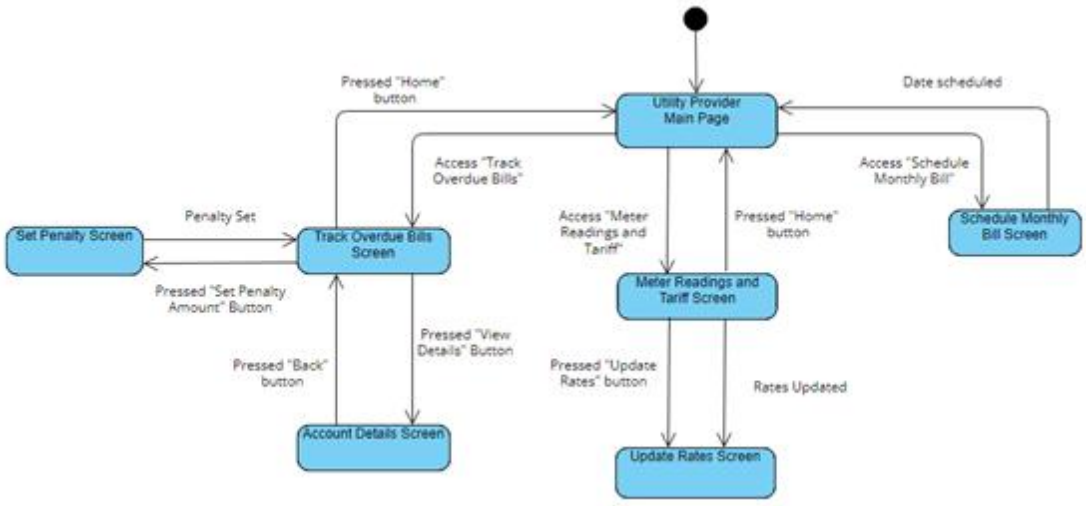
The customer state diagram models the customer workflow within an electricity billing system. The central element is the "Customer Main Page," which acts as a hub for various functionalities. From this page, customer can access different options, they can view their active bills and drill down to view detailed information about each bill. Additionally, they can access the "Setting" section to update their personal information or reset their password. The system also allows customers to provide feedback, make payments for their bills, or report any issues they may be experiencing. After performing any of these actions, users can return to the "Customer Main Page" to access other

functionalities. This diagram illustrates the dynamic nature of the system, enabling users to navigate seamlessly between different functionalities within the context of the electricity billing system.



3.2.2 Utility Provider

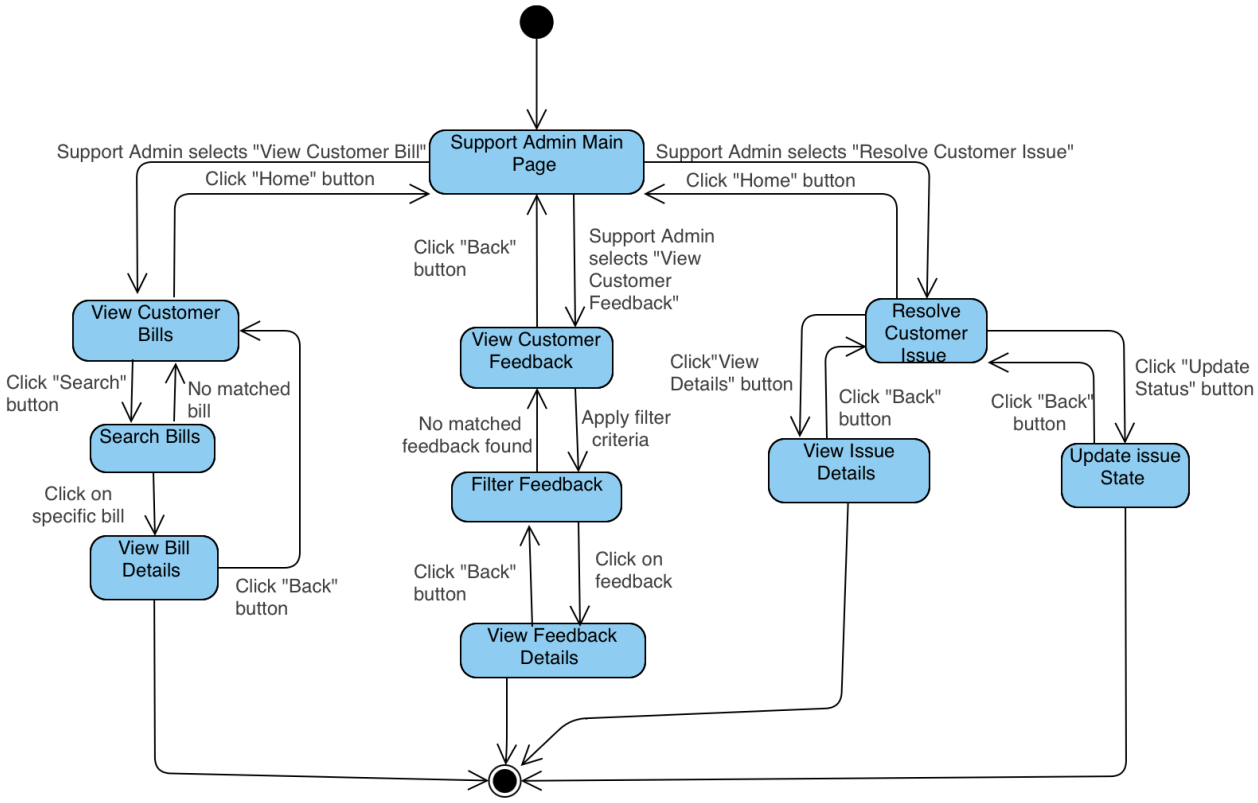
The Utility Provider state diagram shows the states the system can be inside the Utility Provider subsystem. From the main page, the utility provider can access three pages which are “Track Overdue Bills”, “Update Meter Readings and Tariffs”, and “Generate Monthly Bills”. The actions of the utility provider will trigger the transitions between states with the help of buttons like “Set Penalty Amount”, “View Details”, and “Update Rates”. Each state includes a “Back” button that allows the staff to return to the previous state.



3.2.3 Support Admin

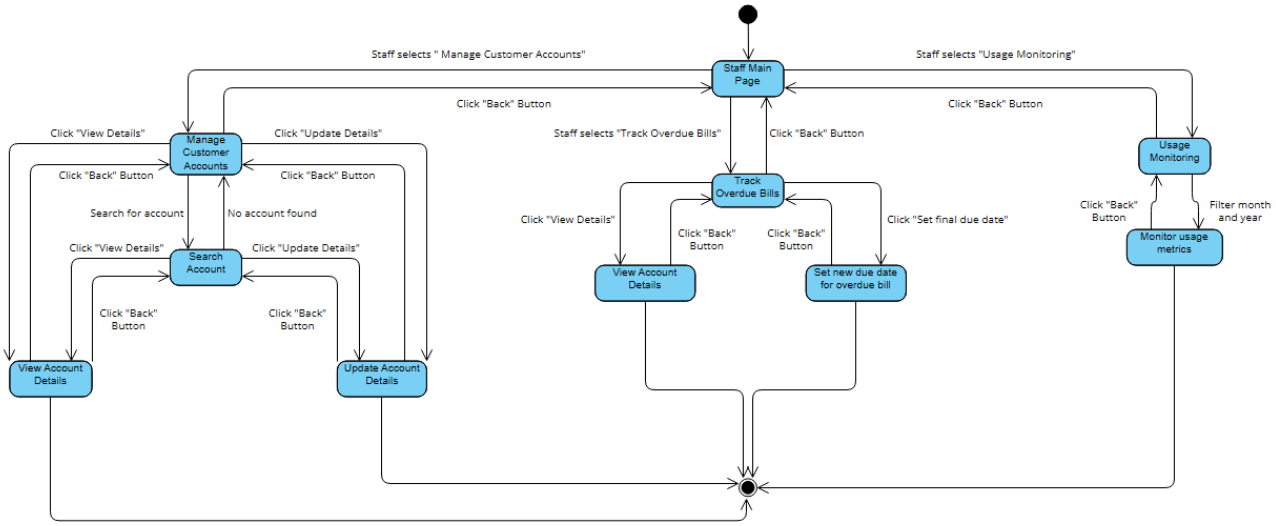
The support admin state diagram models the workflow of the Support Admin interface in an Electricity Billing System. It consists of multiple states representing different functional areas, such as "View Customer Bill," "View Customer Feedback," and "Resolve Customer Issue." Transitions

between states are triggered by admin actions, such as clicking buttons like "Search," "View Details," "Update Status," and "Home." The "Home" state acts as a central hub, allowing the administrator to return to the main page from any other state. This hierarchical structure enables efficient navigation and task management within the system.



3.2.4 Staff

The staff state diagram shows the various states the system can be in inside the staff subsystem starting from the staff main page where staff can navigate to three primary functionalities which are “Manage Customer Accounts”, “Track Overdue Bills”, and “Usage Monitoring”. In each state, the transitions are caused by the staff’s action such as clicking buttons like “View Details”, “Update Details”, “Set Final Due Date”, and filtering month and year. Each state includes a “Back” button that allows the staff to return to the previous state.


Staff Main Page
Staff selects " Manage Customer Accounts"
Staff selects "Usage Monitoring"
Click "Back" Button
Click "Back" Button
Manage Customer Accounts
Click "View Details"
Click "Update Details"
Click "Back" Button
Click "Back" Button
Search for account
No account found
Search Account
Click "View Details"
Click "Update Details"
Click "Back" Button
Click "Back" Button
View Account Details
Update Account Details
Staff selects "Track Overdue Bills"
Click "Back" Button
Track Overdue Bills
Click "View Details"
Click "Back" Button
Click "Back" Button
Click "Set final due date"
View Account Details
Set new due date for overdue bill
Usage Monitoring
Click "Back" Button
Filter month and year
Monitor usage metrics

3.3 Data Dictionary

The Data Dictionary outlines the structure of the database that supports the Electricity Billing System. Each table within the dictionary corresponds to a key component of the system, detailing the relationships, attributes, and constraints necessary for seamless data management. It provides a comprehensive overview of fields, data types, and constrains, ensuring data integrity, scalability, and optimal system performance. The following tables are described in detail:

- ***Customer Table***

Field Name	Data Type	Length	PK/FK	Required?	Null/Not Null	Description
customer_id	Char	6	PK	Yes	Not Null	Unique identifier for each customer.
username	Varchar	10		Yes	Not Null	Unique username for each customer to log in.
password	Varchar	10		Yes	Not Null	Encrypted customer password to secure access to the system.
customer_name	Varchar	30		Yes	Not Null	Full name of the customer.
customer_address	Varchar	100		Yes	Not Null	Residential address of the customer.
customer_phone	Varchar	13		Yes	Not Null	Phone number of the customer.
customer_email	Varchar	120		Yes	Not Null	Email address of the customer.
meter_id	Char	6	FK	Yes	Not Null	Reference to the meter connected to the customer for billing purposes.

- ***Utility Provider***

Field Name	Data Type	Length	PK/FK	Required?	Null/Not Null	Description
utility_id	Char	6	PK	Yes	Not Null	Unique identifier for each utility provider.
username	Varchar	10		Yes	Not Null	Unique username for each utility provider.
password	Varchar	10		Yes	Not Null	Encrypted utility provider password to secure access to the system.
up_name	Varchar	30		Yes	Not Null	Full name of the utility provider.
up_phone	Varchar	13		Yes	Not Null	Utility provider phone number.
up_email	Varchar	120		Yes	Not Null	Email address of the utility provider.

- ***Support Admin***

Field Name	Data Type	Length	PK/FK	Required?	Null/Not Null	Description
admin_id	Char	6	PK	Yes	Not Null	Unique identifier for each admin.
username	Varchar	10		Yes	Not Null	Unique username for each admin.
password	Varchar	10		Yes	Not Null	Encrypted admin password to secure access to the system.
admin_name	Varchar	30		Yes	Not Null	Full name of the admin.
admin_phone	Varchar	13		Yes	Not Null	Admin phone number.
admin_email	Varchar	120		Yes	Not Null	Email address of the admin.

- ***Staff Table***

Field Name	Data Type	Length	PK/FK	Required?	Null/Not Null	Description
staff_id	Char	6	PK	Yes	Not Null	Unique identifier for each staff.
username	Varchar	10		Yes	Not Null	Unique username for each staff.
password	Varchar	10		Yes	Not Null	Encrypted staff password to secure access to the system.
staff_name	Varchar	30		Yes	Not Null	Full name of the staff.

staff_phone	Varchar	13		Yes	Not Null	Staff phone number.
staff_email	Varchar	120		Yes	Not Null	Email address of the staff.

- ***Bill Table***

Field Name	Data Type	Length	PK/FK	Required?	Null/Not Null	Description
bill_id	Char	6	PK	Yes	Not Null	Unique identifier for each bill.
customer_id	Char	6	FK	Yes	Not Null	Reference to the customer responsible for the bill.
amount	Decimal	10, 2		Yes	Not Null	The bill amount that must be paid.
due_date	Date			Yes	Not Null	Due date for bill payment
paid	Boolean	1		Yes	Not Null	Status indicates if bill has been paid or not.
creation_date	Date			Yes	Not Null	Date when the bill was generated.
penalty_fee	Decimal	10,2		Yes	Not Null	A calculated fee for each late day.
usage	Decimal	10,2		Yes	Not Null	Electricity usage of the customer

- ***Issue Table***

Field Name	Data Type	Length	PK/FK	Required?	Null/Not Null	Description
issue_id	Char	6	PK	Yes	Not Null	Unique identifier for each reported issue.
customer_id	Char	6	FK	Yes	Not Null	Reference to the customer reporting the issue.
title	Varchar	15		Yes	Not Null	Short title of the issue.
description	Varchar	300		Yes	Not Null	The detailed description of the reported issue
status	Varchar	15		Yes	Not Null	Current status of the issue (e.g., "In

						Progress", "Resolved").
ticket_id	Char	6		Yes	Not Null	Unique ticket number for tracking the issue.

- ***Payment Table***

Field Name	Data Type	Length	PK/FK	Required?	Null/Not Null	Description
payment_id	Char	6	PK	Yes	Not Null	Unique identifier for each payment.
customer_id	Char	6	FK	Yes	Not Null	Reference to customer who made the payment.
payment_date	Date			Yes	Not Null	Date the payment was made.
payment_method	Varchar	15		Yes	Not Null	Payment method used (e.g., credit card, bank transfer).
amount	Decimal	10, 2		Yes	Not Null	The amount paid.

- ***Feedback Table***

Field Name	Data Type	Length	PK/FK	Required?	Null/Not Null	Description
feedback_id	Char	6	PK	Yes	Not Null	Unique identifier for each feedback entry.
customer_id	Char	6	FK	Yes	Not Null	Reference to the customer providing the feedback.
rating	Int	1		Yes	Not Null	Customer rating out of 5 stars.
comment	Varchar	200		Yes	Not Null	Feedback comment or suggestions from the customer.

feedback_date	Date			Yes	Not Null	Date the feedback was submitted.

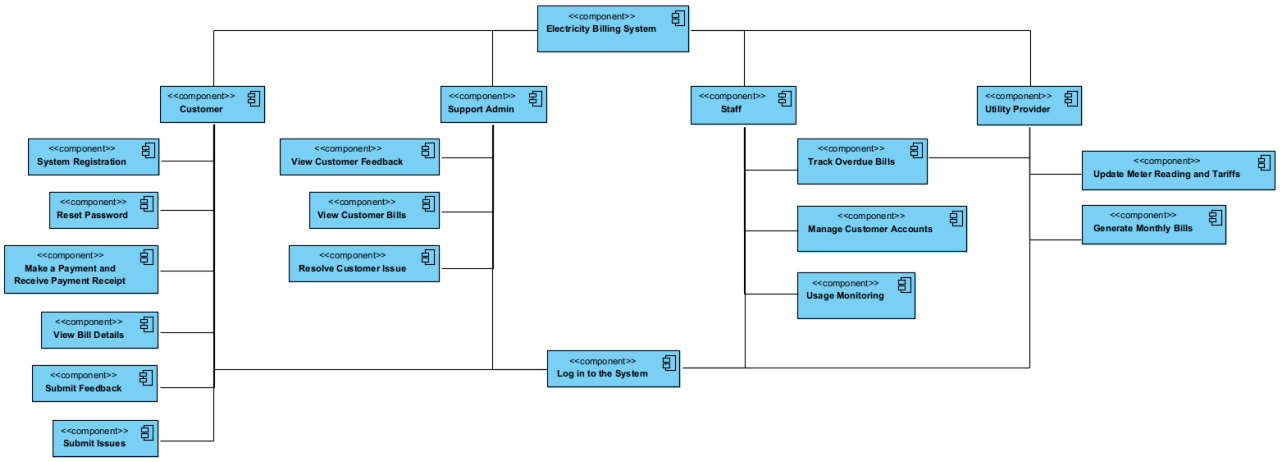
- ***Meter Table***

Field Name	Data Type	Length	PK/FK	Required?	Null/Not Null	Description
meter_id	Char	6	PK	Yes	Not Null	Unique identifier for each meter.
customer_id	Char	6	FK	Yes	Not Null	Reference to the customer associated with the meter.
meter_reading	Decimal	10, 2		Yes	Not Null	Latest recorded meter reading.
tariff_rate	Decimal	10, 2		Yes	Not Null	Rate per unit of electricity.

- ***Usage Metrics Table***

Field Name	Data Type	Length	PK/FK	Required?	Null/Not Null	Description
total_customers	Int	6		Yes	Not Null	Total customers registered in the system
total_bills	Int	6		Yes	Not Null	Total bills generated in the system
total_usage	Decimal	10, 2		Yes	Not Null	Total electricity usage
avg_usage	Decimal	10, 2		Yes	Not Null	Average electricity usage

3.4 Software Architecture

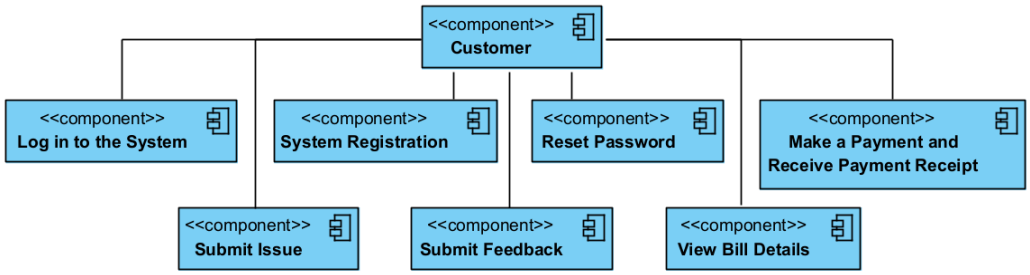


Subsystem	Team members
Customer	Mohammed Yousef Mohammed Abdulkarem
Support Admin	Mohammed Aamena Mohammed Abdulkarem
Staff	Muhammad Faiz bin Ilyasa
Utility Provider	Chua Cheng Zong

3.4.1 Subsystem 1: Customer

The Customer subsystem acts as the main interface through which end-users engage with the system. It is crafted to deliver a seamless and user-friendly experience while maintaining secure and efficient access to electricity services. The subsystem is organized into the following modules:

- **System Registration:** Enables new users to create accounts by providing necessary information.
- **Log in to the System:** Allows registered users to securely access their accounts using their credentials.
- **Reset Password:** Provides a mechanism for users to reset their forgotten passwords.
- **Make a Payment and Receive Payment Receipt:** Facilitates online bill payments and generates digital receipts for successful transactions.
- **Submit Feedback:** Enables users to provide feedback and suggestions to the utility provider.
- **Submit Issues:** Allows users to report issues or problems related to their electricity service.
- **View Bill Details:** Provides users with access to detailed information about their electricity bills, including consumption history, due dates, and payment history.

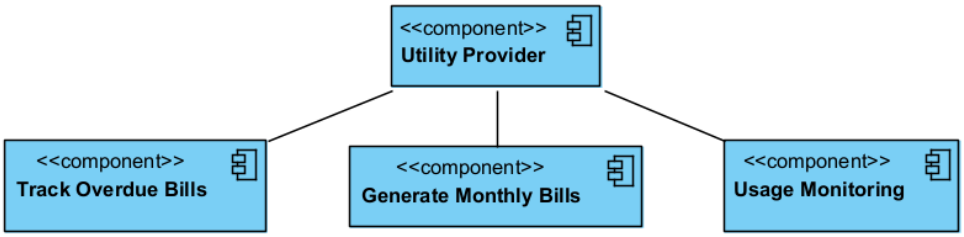


3.4.2 Subsystem 2: Utility Provider

The Utility Provider subsystem is used to optimize billing processes and monitor electricity usage. It is designed to manage billing functions and maintaining efficient communication with customers and the staff. It has the following three main components:

1. **Track Overdue Bills**
   This component enables the utility provider to identify, verify and handle overdue bills within the system. It provides them with a list of overdue users after the staff has updated the system. Utility provider can then take necessary actions such as notifying the customer, setting a penalty or even cutting their electricity, depending on the severity.
2. **Generate Monthly Bills**
   This component helps the utility provider automate the monthly billing process, ensuring timely and accurate bill generation for all customers. This helps the customers to track their usage accurately, it also stores the digital bill info in the database for future reference.
3. **Meter Reading and Tariff**
   This component lets the utility provider check data of the meter readings and tariffs and also change them when needed. This helps the bill generation process to be more accurate.

The Utility Provider subsystem ensures efficient management of financial billing and monitoring. Each components work together to enhance operational efficiency, maintain financial stability, and improve customer satisfaction.



3.4.3 Subsystem 3: Support Admin

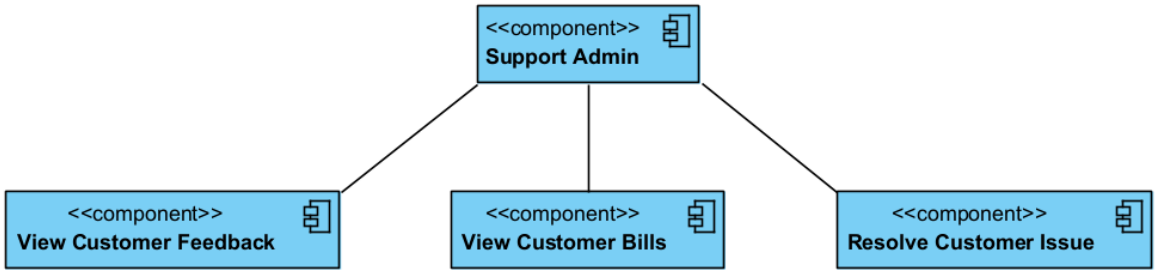
This subsystem is responsible for managing interactions between the support administrator and the various system components related to customer support. The diagram shows three key components under the "Support Admin" subsystem:

1. **View Customer Feedback**:
   This component allows the support admin to access and review feedback provided by

customers. It helps the administrator to analyze and address customer satisfaction and concerns.

2. **View Customer Bills**:
   This component enables the support admin to view and manage billing details for customers. It ensures that the admin has access to customer billing history for handling inquiries or disputes.
3. **Resolve Customer Issue**:
   This component provides the functionality for the support admin to address and resolve customer-related problems. It focuses on troubleshooting and ensuring customer satisfaction.

The central "Support Admin" component acts as the parent node, coordinating the activities of the three sub-components. Each of these components plays a crucial role in ensuring effective customer service and maintaining the overall functionality of the system.

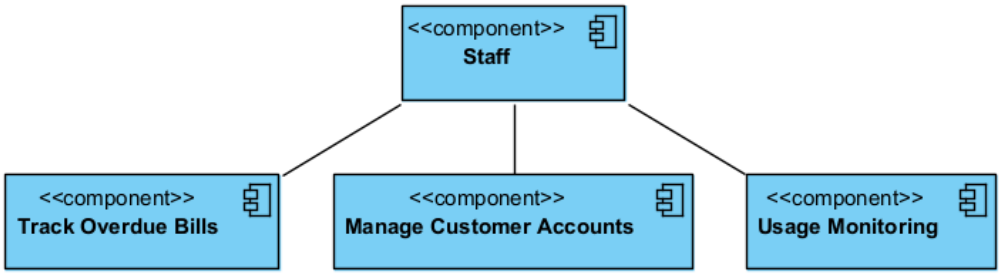


3.4.4 Subsystem 4: Staff

The staff subsystem allows the staff to efficiently handle and manage customer-related activities and ensure smooth system operations. This subsystem includes features to manage customer accounts, track overdue bills and monitor usage. Below is an overview of the subsystem's components:

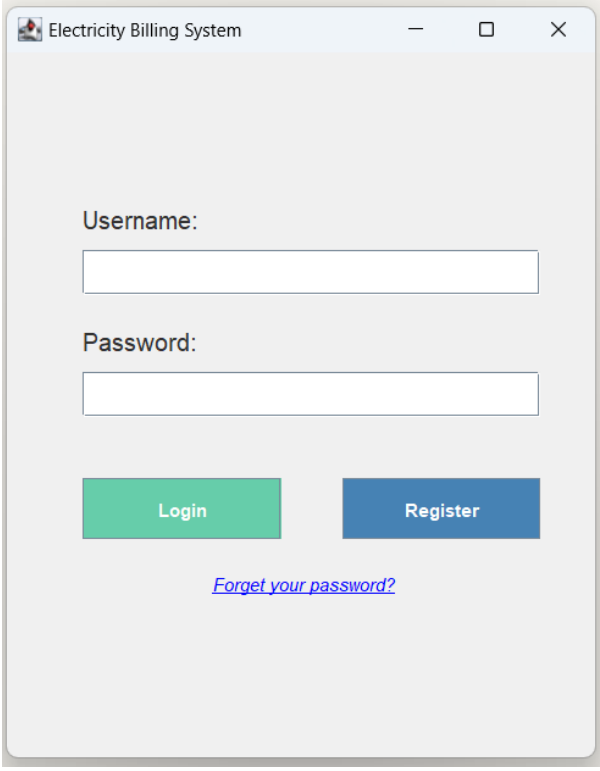
1. **Manage Customer Accounts**:
   This component allows staff to manage customer accounts by enabling them to view account details and update its information. Staff can search for accounts for easier and efficient access to customer accounts.
2. **Track Overdue Bills**:
   This component enables staff to identify, verify and handle overdue bills within the system. It generates a list of accounts with overdue bills updated by the system based on the pay date. Staff can review the customer's account and take necessary actions such as setting a new due date for the overdue bill and notify the customer.
3. **Usage Monitoring**:
   This component allows staff to review and analyze customer usage data, such as total number of customers, total number of bills generated, and total number of bills paid. It helps staff to identify trends and irregularities based on the data received.

Overall, the staff subsystem enhances operational efficiencies and customer service by providing staff with capabilities to manage customer-related process. These three components work seamlessly to allow staff to perform their tasks with ease and precision, improving customer satisfaction and supporting the overall success of the program.



3.5 Main Screens

The Login Screen for the Electricity Billing System serves as the entry point for all users, including customers, support administrators, staff, and utility providers, to access the application. It ensures secure and authorized access while maintaining a user-friendly interface. After successful authentication, users are directed to their respective interfaces, tailored to their roles and permissions.

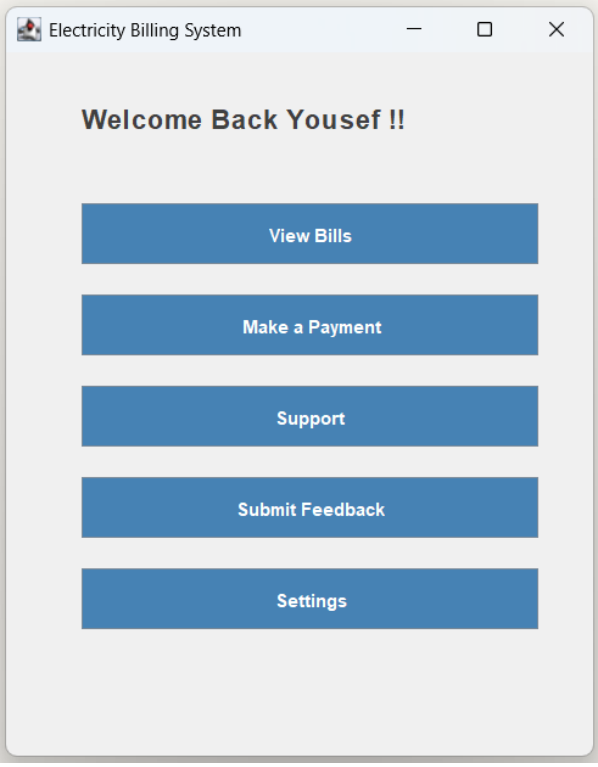


3.6 Subsystem 1 Screens: Customer

The main screen will be a one-stop dashboard where customers can manage all the tasks related to electricity. It will have easy access to the most used features like payment, bill display, profile update, feedback, and issue reporting. The design has focused on user convenience and operational efficiency.

Main screen

This is the first screen customers encounter upon logging into their accounts. It features multiple buttons, each providing access to a specific service, ensuring a user-friendly experience. The displayed name, such as "Yousef," dynamically updates to reflect the logged-in customer's identity, adding a personalized touch to the interface.



- **Customer-Specific Welcome Message**
  The screen displays a personalized message, such as "Welcome Back Yousef!". This feature confirms successful login and enhances the user experience by providing a warm and tailored greeting.
- **View Bills**
  The "View Bills" button allows customers to view a comprehensive list of all their electricity bills. The list includes essential details such as the billing balance, due date, and total amount due.
- **Make a Payment**
  The "Make a Payment" button allows customers to pay their electricity bills easily. Upon clicking the button, it redirects the user to a payment interface where they can see the outstanding balance of their bills, select a mode of payment, and make the payment seamlessly.
- **Feedback**
  The "Submit Feedback" button helps customers to provide feedback regarding the experience of using the electricity service system. Upon clicking, users are taken to a feedback form

where they rate and make comments or give suggestions about their experiences. This will help the Support Admin improve on the quality of service based on the input provided by the customers.

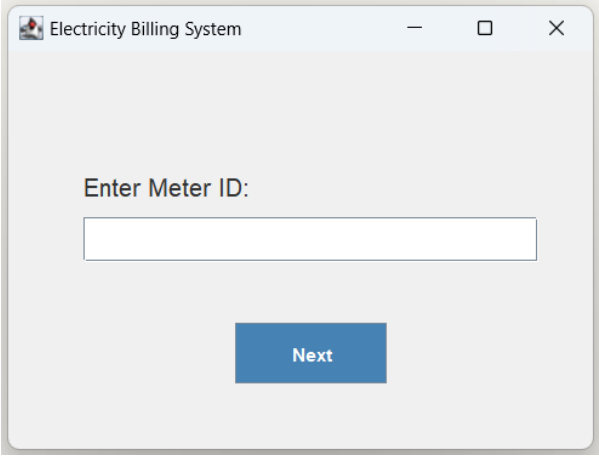
- **Support**
  The "Submit Issues" button allows the customer to report a problem or an issue regarding the electricity service, such as errors in billing or malfunctioning of the meter. It redirects the users to the form for submitting an issue, where they can explain the problem with attachments. This helps in timely resolution of the reported issue.

Registration screen

The **"Enter Meter ID" Screen** is a crucial step in linking the user's account to their specific electricity meter. Users are prompted to input their unique Meter ID and click "Next." The system validates the entered Meter ID to ensure accuracy:

- **If Valid:** The user proceeds to the next registration step, such as entering personal details.
- **If Invalid or Already Registered:** An error message is displayed, preventing further progress.

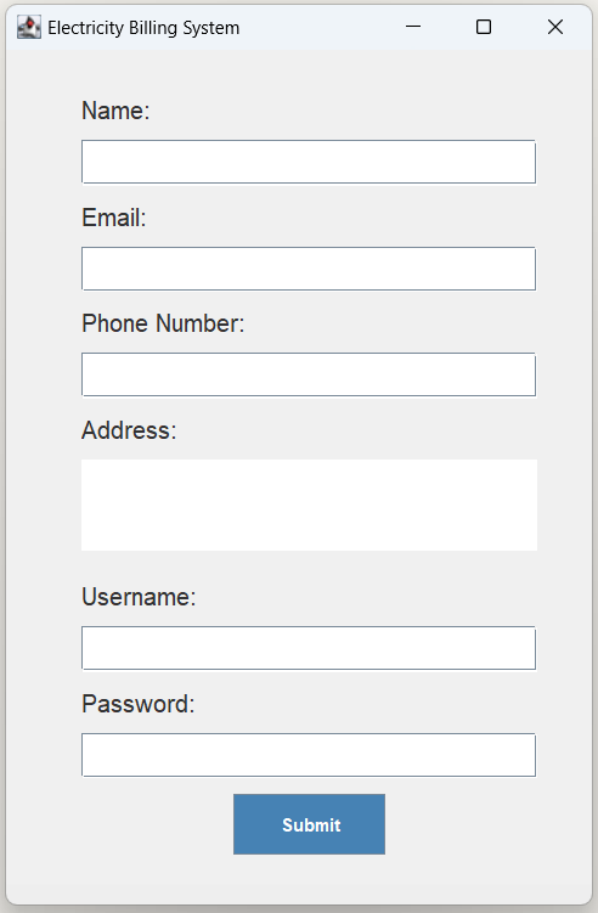
This step ensures accurate billing and proper association of accounts with the correct meter, forming the foundation for reliable service.



Once the user clicks "Next" with a valid Meter ID, they are redirected to the **Electricity Billing System Registration Form**. This form is designed to onboard new users by collecting mandatory details, including:

- Name
- Email
- Phone Number
- Address
- Username
- Password

After completing the required fields, users click the "Submit" button to finalize account creation. This registration process ensures security and proper access control, allowing only authorized users to manage electricity bill processing and account-related tasks efficiently.

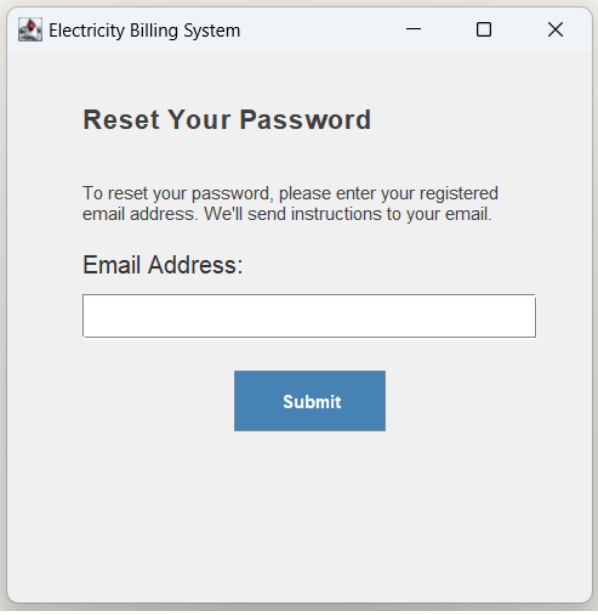


Reset password screen

The **"Reset Your Password"** feature of the Electricity Billing System offers a straightforward and secure mechanism for users to regain access to their accounts in case of forgotten passwords.

- **User Input:** Users enter the email address linked to their registered account in a simple and intuitive interface.
- **Action:** Upon clicking the "Submit" button, the system triggers the password reset process.
- **System Response:** An email is sent to the provided address containing detailed instructions, typically including a reset link or a verification code, to facilitate password recovery.

This feature ensures user convenience by enabling seamless account recovery while maintaining robust security measures to protect sensitive account information.



Report issue screen

This interface allows customers to easily report problems or issues related to their electricity service.

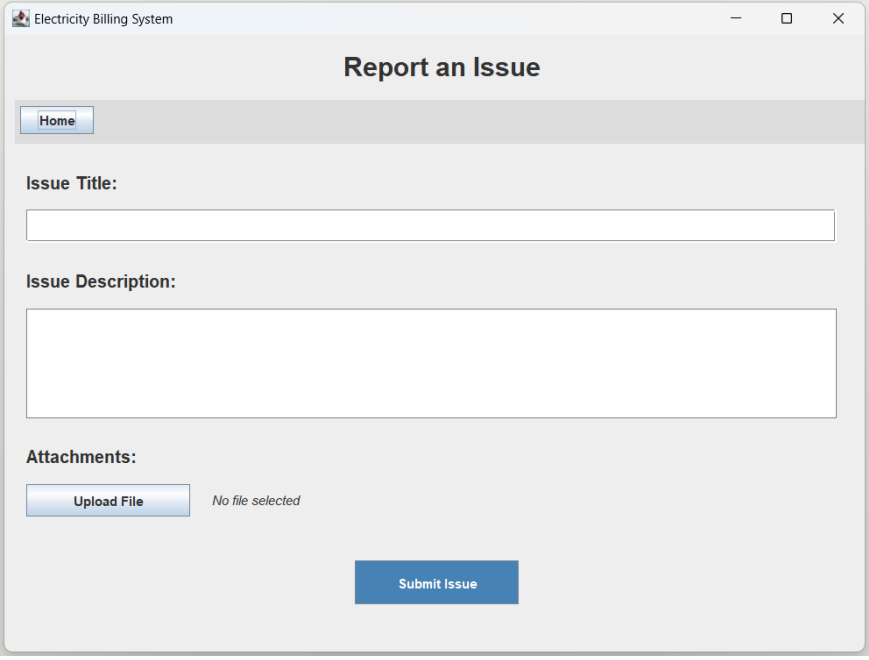
Navigate Home Button: This button takes the customer back to the main home screen of the system.

Issue Title: A text field where the customer can enter a brief title or subject for their issue.

Issue Description: A text area where the customers can provide a detailed description of the problem they are facing. This could include information like the nature of the problem, when it started, and any relevant symptoms.

Attachments: This section allows the customer to upload any supporting documents or images related to their issue. Upload file button triggers the file upload dialog, allowing to select files from the computer with message indicates if no file has been selected.

Submit Button: Once the user has entered the necessary information, they can click this button to submit the issue report.



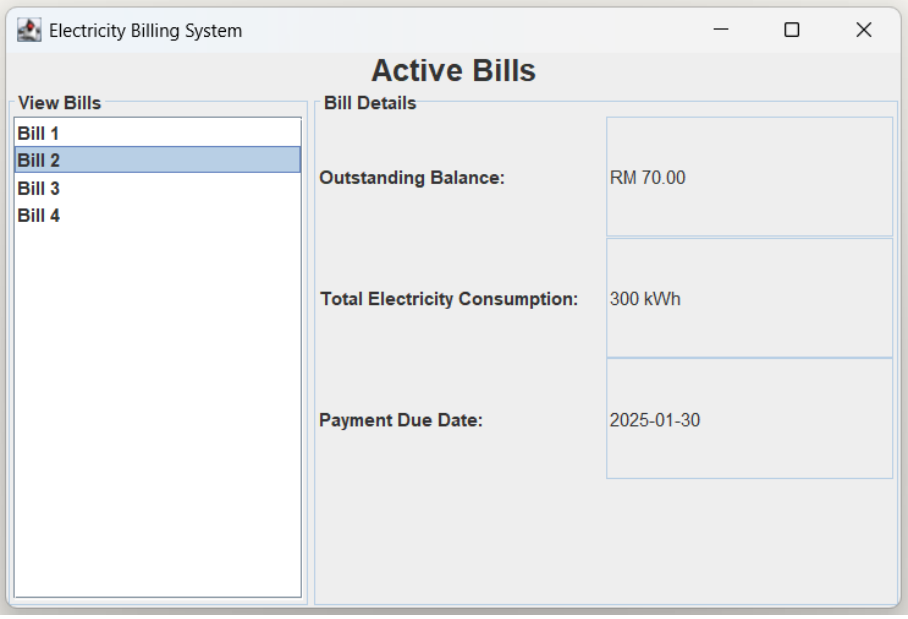
View bill details screen

This screen displays a list of the user's active electricity bills and provides detailed information about the selected bill.

View Bills: A list displaying all active bills associated with the user's account. Each bill is represented by a numbered entry (e.g., "Bill 1," "Bill 2").

Bill Details: A section that displays detailed information about the selected bill from the "View Bills" list. This includes:

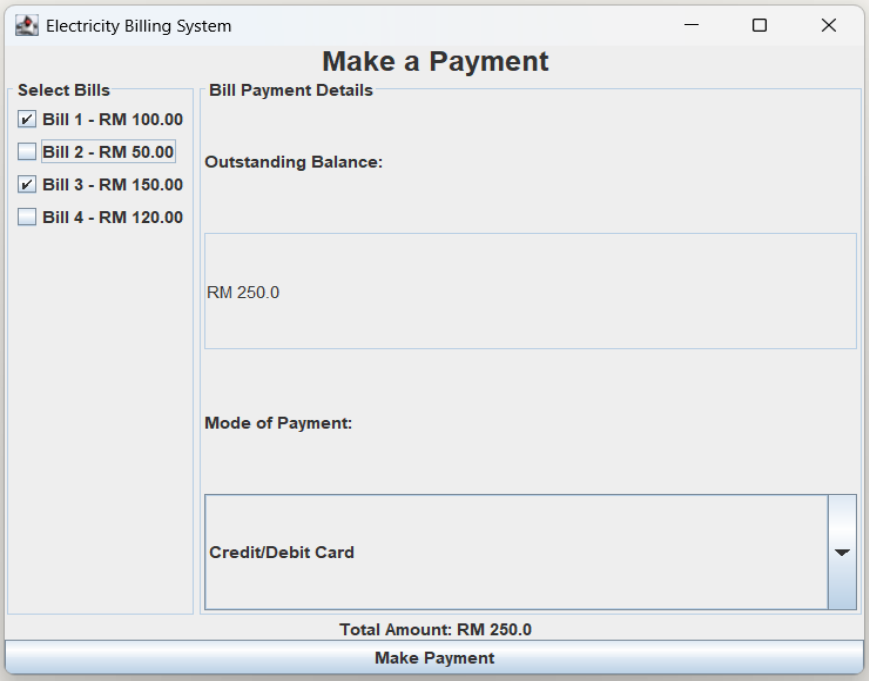
- Outstanding Balance: The amount to be paid for the selected bill.
- Total Electricity Consumption: The total amount of electricity consumed during the billing period.
- Payment Due Date: The date by which the payment for the selected bill is due.



Make a payment screen

This screen enables users to select bills for payment and proceed with the payment process.

- Select Bills: A list displaying all active bills associated with the customer's account. Each bill is presented with a checkbox to select and its corresponding amount. Customers can select multiple bills for payment.
- Outstanding Balance: Displays the total amount due for the selected bills.
- Mode of Payment: A dropdown menu allowing users to select their preferred payment method (e.g., Credit/Debit Card, Bank Transfer).
- Total Amount: Displays the total amount to be paid, which matches the "Outstanding Balance" of selected bills.



3.7 Subsystem 2 Screens: Utility Provider

Main Screen

The initial screen of the Utility Provider interface provides an organized and user-friendly environment for the utility provider to efficiently manage finance-related tasks within the Electricity Billing System. Below are the key elements visible on this screen:

- **Welcome Message**:

The Utility Provider is greeted with a welcoming message upon successful login.

- **Meter Readings and Tariffs:**

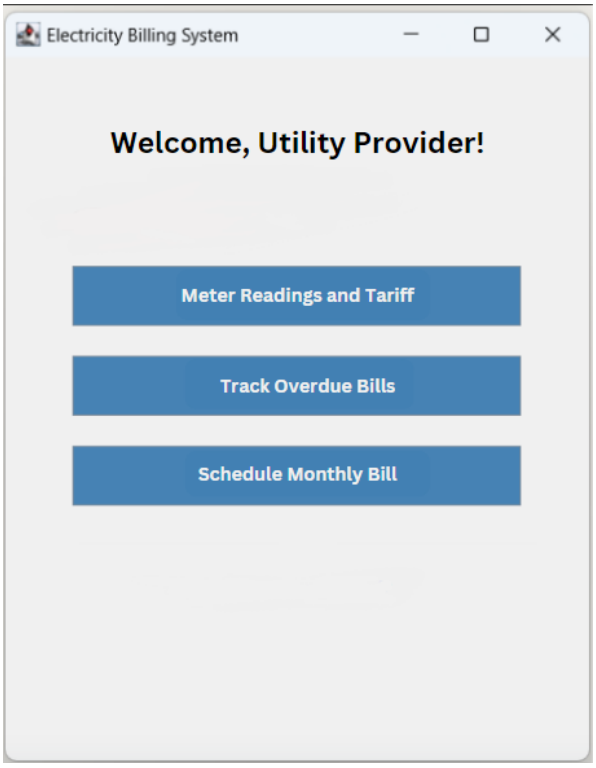
This option allows Utility Provider to check the current rates for the meter readings and tariffs, and to update the data values. This helps the staff be able to calculate the bill to accurately charge the customers based on the rates.

- **Track Overdue Bills:**

This option helps the Utility Provider check which customer has outstanding fees. The Utility Provider can choose to give them one more week as the final deadline or disconnect the electricity. This helps the Utility Provider to deal with customers with overdue fees efficiently.

- **Schedule Monthly Bill:**

This option lets the Utility Provider schedule a date on every month to generate the bill for each customer. This helps the Utility Provider to automate the billing process, with great accuracy and efficiency.

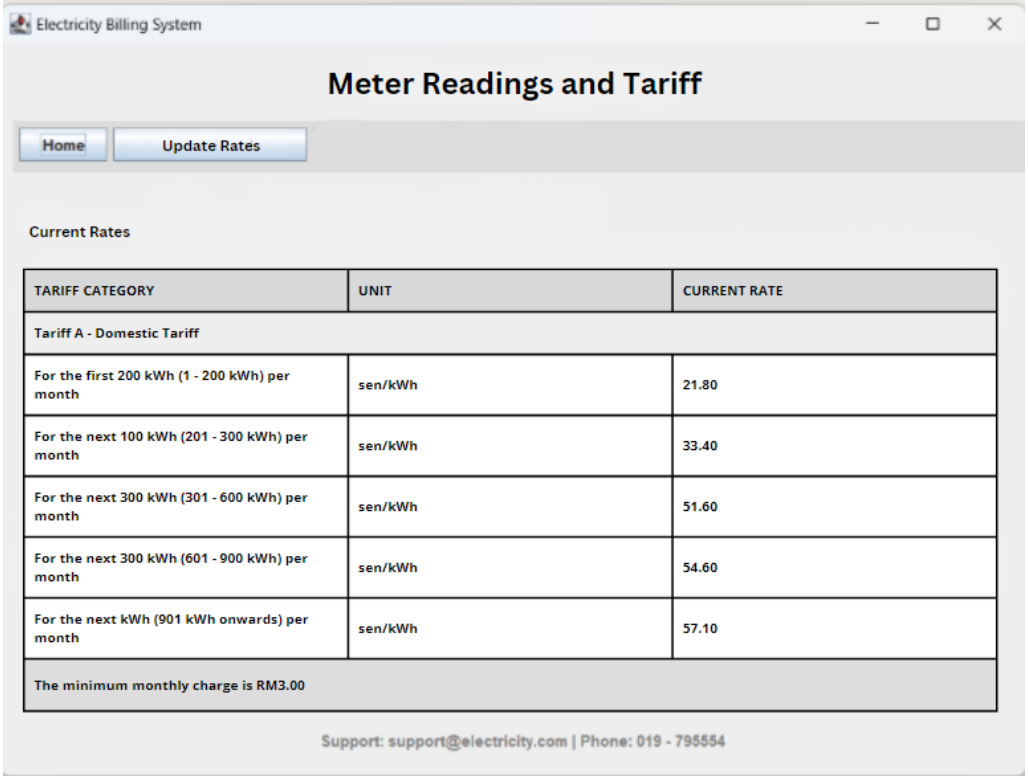


Meter Readings and Tariff screen

The Meter Readings and Tariff page is accessible by clicking the "Meter Readings and Tariff" button, it lets the utility provider check the current tariff values to make sure the data is accurate.
The column displays data such as:
Tariff category: The category of tariff.
Unit: The unit of tariff.
Current Rate: The current rate of tariff which the customer should be charged.

Navigation Bar:
The navigation bar provides quick access to various sections within the system:

- **Home**: Redirects the Utility provider to the main screen.
- **Update Rates**: Brings the Utility provider to the Update Rates screen.



Update Rates screen

The "Update Rates" screen prompts the utility provider to key-in the new tariff rates, there is a drop-down menu to let them choose which tariff category to update.

Track Overdue Bills Screen

The "Track Overdue Bills" screen is designed to help utility providers monitor, verify, and take necessary actions to manage overdue bills in the system. It ensures that overdue payments are identified and handled effectively.

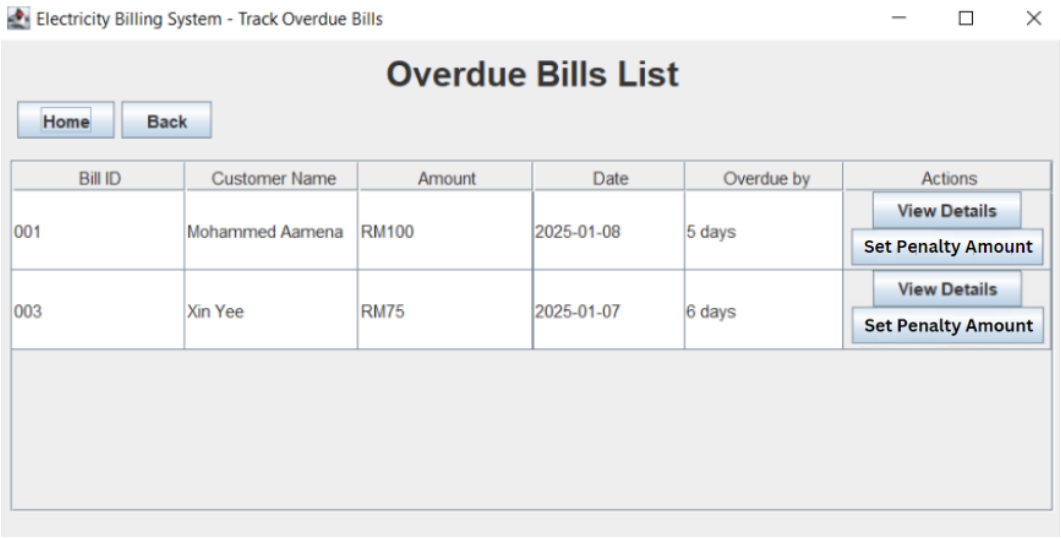
Home Button:

The "Home" button allows the utility provider to go back to the main screen.

Overdue Bills List:

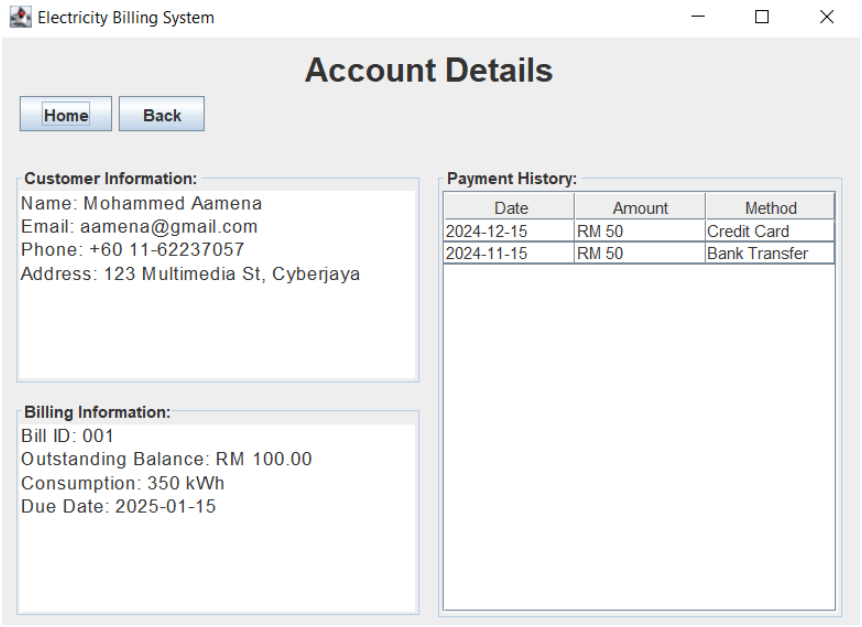
The table displays a list of overdue bills with columns such as:

- **Bill ID:** Unique identifier for each bill.
- **Customer Name:** The name of the account holder.
- **Amount:** The total amount due for the bill.
- **Date:** The date when the bill is generated.
- **Overdue by:** Indicates how long the bill has been overdue by.
- **Actions:** A "View Details" button to view a more detailed information of the bill details, and a "Set Penalty Amount" button for the utility provider to set a penalty for the customer and disconnect their electricity.



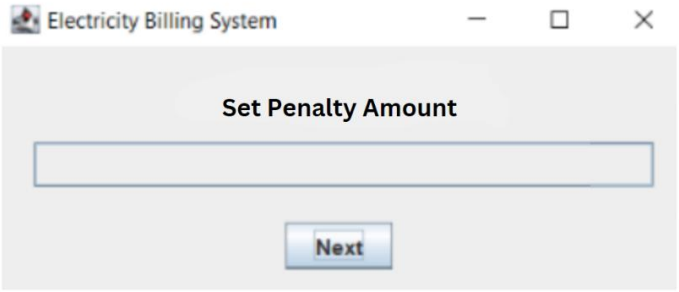
View Details Screen

The "View Details" screen provides a detailed description of the customer information, their billing information and payment histories. Utility providers can review and verify the necessary information.



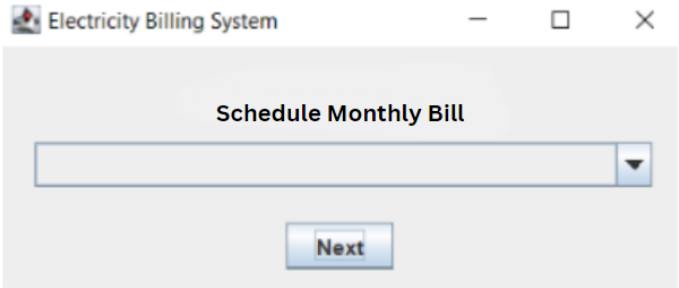
Set Penalty Screen

The “Set Penalty" screen prompts the utility provider to key-in the penalty amount of the customer. This will disconnect the electricity for the customer until they pay the penalty.



Schedule Monthly Bill screen

The “Schedule Monthly Bill” screen lets the utility provider to set a scheduled date to generate the monthly bill automatically, there is a drop-down menu with a calendar to assist them with scheduling.



3.8 Subsystem 3 Screens: Support Admin

Main Screen

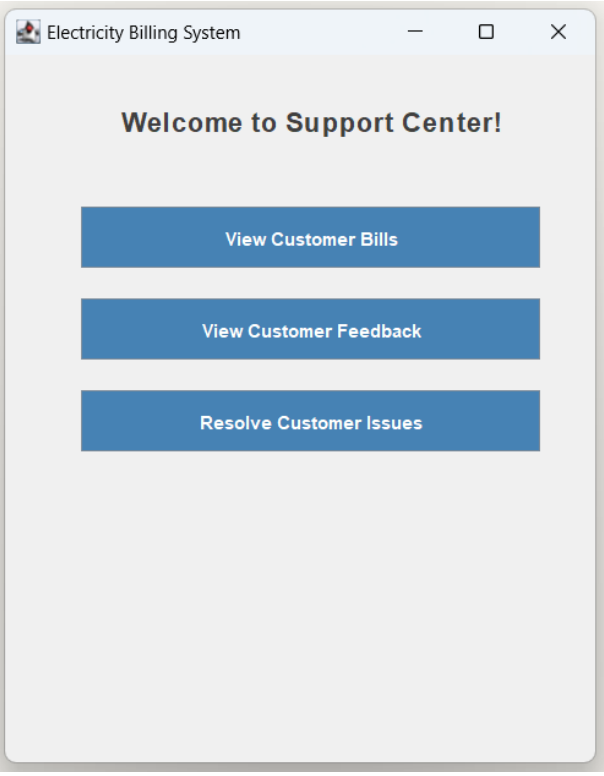
The initial screen of the Support Admin interface provides an organized and user-friendly environment for Support Administrators to efficiently manage customer-related tasks within the Electricity Billing System. Below are the key elements visible on this screen:

Welcome Message ("Welcome to Support Center!"): This greeting establishes a friendly and approachable tone for the interface, signalling the start of the Support Admin's session.

View Customer Bills: By selecting this option, Support Admins gain immediate access to customer billing details. This functionality is essential for troubleshooting billing discrepancies, answering customer inquiries, or providing clarifications on bill charges.

View Customer Feedback: This option allows Support Admins to review customer feedback, including complaints, suggestions, and compliments. Access to this information is critical for understanding customer satisfaction, identifying recurring issues, and prioritizing areas for service enhancement.

Resolve Customer Issues: Clicking this button directs Support Admins to a section dedicated to issue resolution. This includes features for tracking open issues, communicating with customers, managing resolution workflows, and accessing a knowledge base of common solutions.



View Customer Bill Screen

The primary function of the Bill Management Screen is to allow Support Admins to search, view, and manage customer bills in a streamlined interface. This screen displays a list of customer bills and offers the ability to filter and search for specific bills based on various criteria.

Navigation Bar: The navigation bar provides access to various sections within the system:

- **Home**: Takes the Support Admin to the main dashboard or central hub of the system.
- **Customer Feedback**: Directs to a section where customer feedback is managed.
- **Customer Issues**: The current screen where the list of customer issues is displayed.

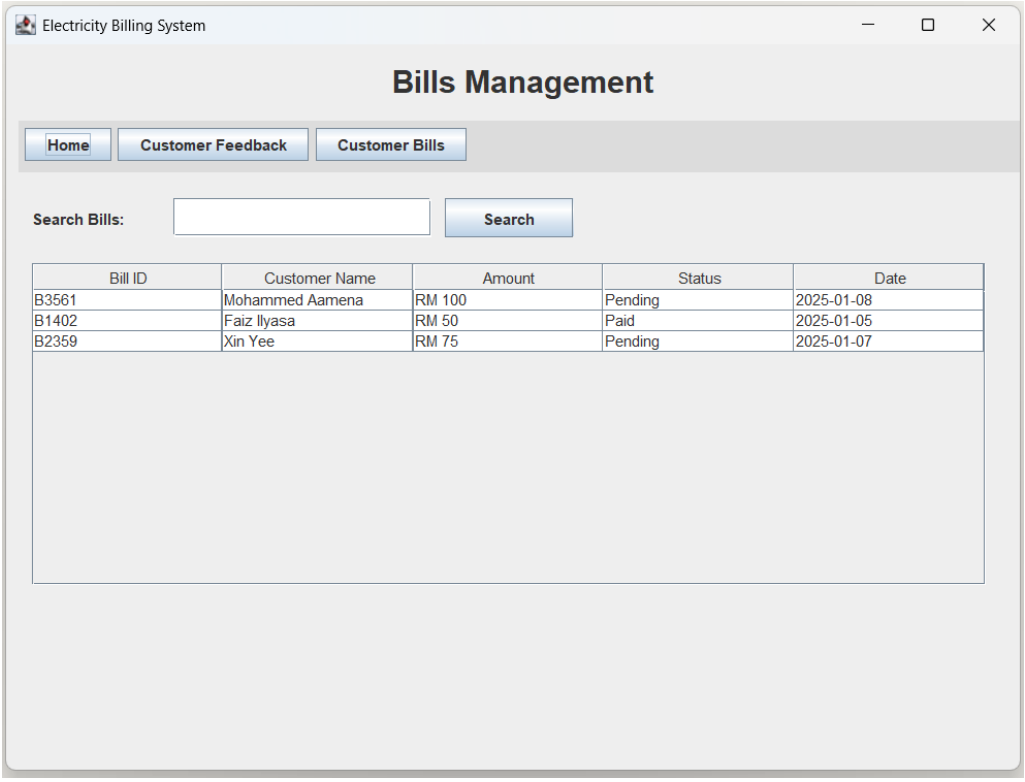
Search Functionality:

- **"Search Bills" Textbox:** Allows Support Admins to enter search criteria, such as customer name or bill ID, to quickly locate specific bills.
- **"Search" Button:** Initiates the search process, filtering the displayed bills based on the entered criteria.

Bill Information List:

- **Bill ID**: A unique identifier for each bill.
- **Customer Name**: The name of the customer associated with the bill.
- **Amount**: The total amount due for the bill.
- **Status**: Indicates the status of the bill (e.g., "Pending" if payment is not received, "Paid" if payment has been made).
- **Date**: The date when the bill was generated.

Clicking on an element on the list for a specific bill would navigate the Support Admin to the **"Bill Details"** screen, where they can view detailed information about the selected bill, including customer details, payment history, and billing details.



View Bill Details Screen

The "Bill Details" screen is accessible from the "View Customer Bills" section by the Support Admin. It provides a comprehensive view of the customer's billing and payment history, ensuring

that the Support Admin has all the relevant information required for resolving issues or assisting with customer inquiries.

The "Bill Details" screen includes the following sections:

Customer Information

This section presents essential customer details, including:

- Customer Name: Full name of the customer.
- Email Address: Contact email for communication.
- Phone Number: Customer's phone number for support contact.
- Customer Address: Address of the customer for record-keeping and contact purposes.

These details help the Support Admin identify the customer and contact them if necessary.

Billing Information

This section provides critical billing data, such as:

- Bill ID: Unique identifier for the bill.
- Outstanding Balance: The total amount owed by the customer.
- Total Electricity Consumption: The electricity usage during the billing period, measured in kilowatt-hours (kWh).
- Due Date: The date in which the payment is due.

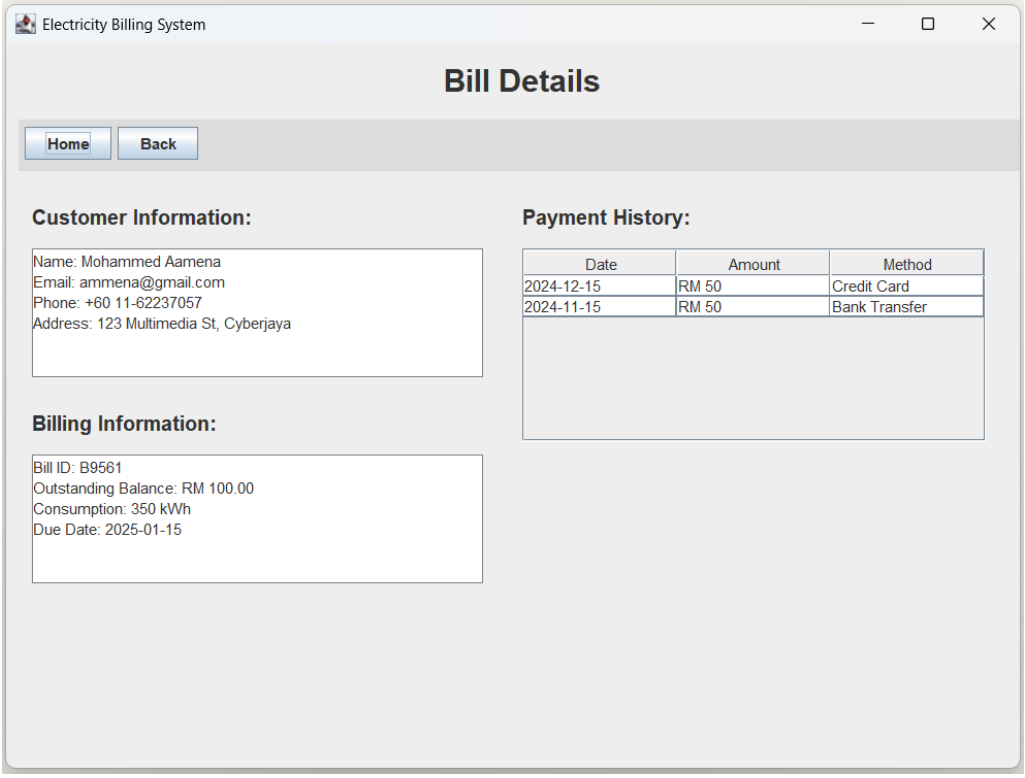
This section is essential for understanding the customer's financial obligations and tracking their billing cycle.

Payment History

Displays a chronological list of payments made toward this bill, which includes:

- Payment Date: Date when each payment was made.
- Amount Paid: Amount paid by the customer for each payment.
- Payment Method: Mode of payment used (e.g., credit card, bank transfer, e-wallet).

Payment history will provide the Support Admin with details on the status of previous payments to identify if any discrepancy or payment is missed.



View Customer Issues Screen

The major role of the Customer Issues Management Screen would be the centralization of all views for the Support Admin about issues reported by customers. Therefore, enable the Support Admin to trace the running issues together with Support Tickets on this screen in a move aimed at taking swift action towards their resolution.

Navigation Bar

The navigation bar provides quick access to various sections within the system:

- **Home**: Redirects the Support Admin to the main dashboard.
- **Customer Feedback**: Leads to a section where customer feedback, including complaints and suggestions, is managed and reviewed.
- **Customer Bills**: Takes the Support Admin to the screen for managing and reviewing customer billing information.

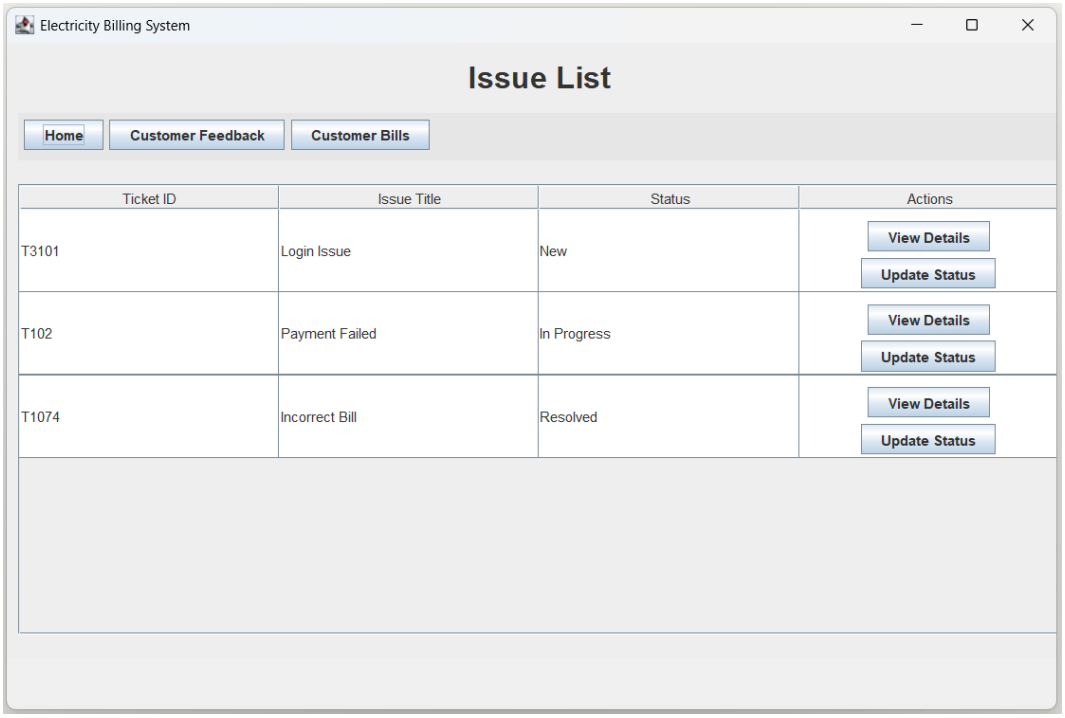
Issue List

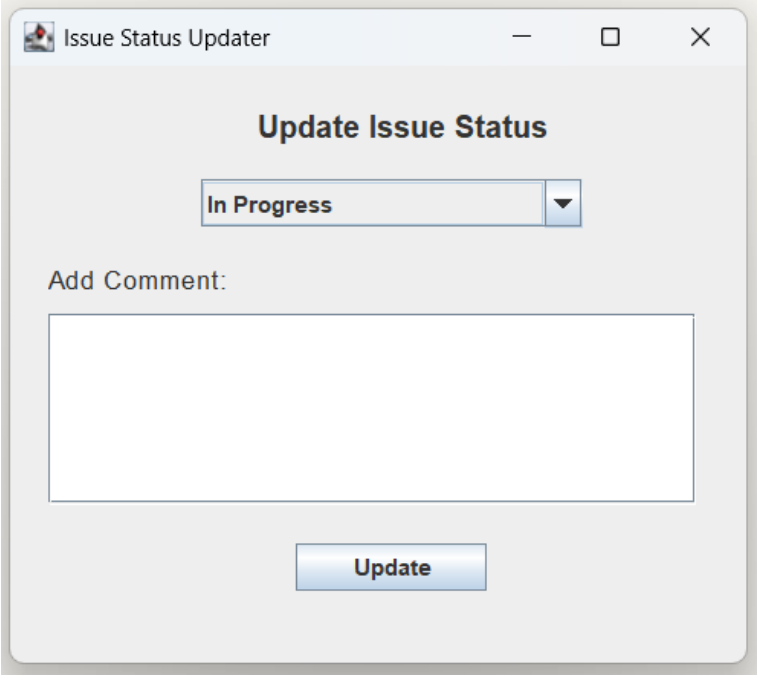
Presents a list of customer issues, displaying the following key columns:

- **Ticket ID**: A unique identifier assigned to each support issue.
- **Issue Title**: A brief about what the issue reported by the customer.
- **Status**: Indicates the status of the issue, such as "In Progress," or "Resolved."

Provides options for Support Admins to interact with each issue:

- **View Details**: Opens a detailed view of the specific issue, providing in-depth information, the full issue description, and any relevant attachments.
- **Update Status**: Allows the Support Admin to modify the status of the issue (e.g., updating it from "In Progress" to "Resolved") and add comments.

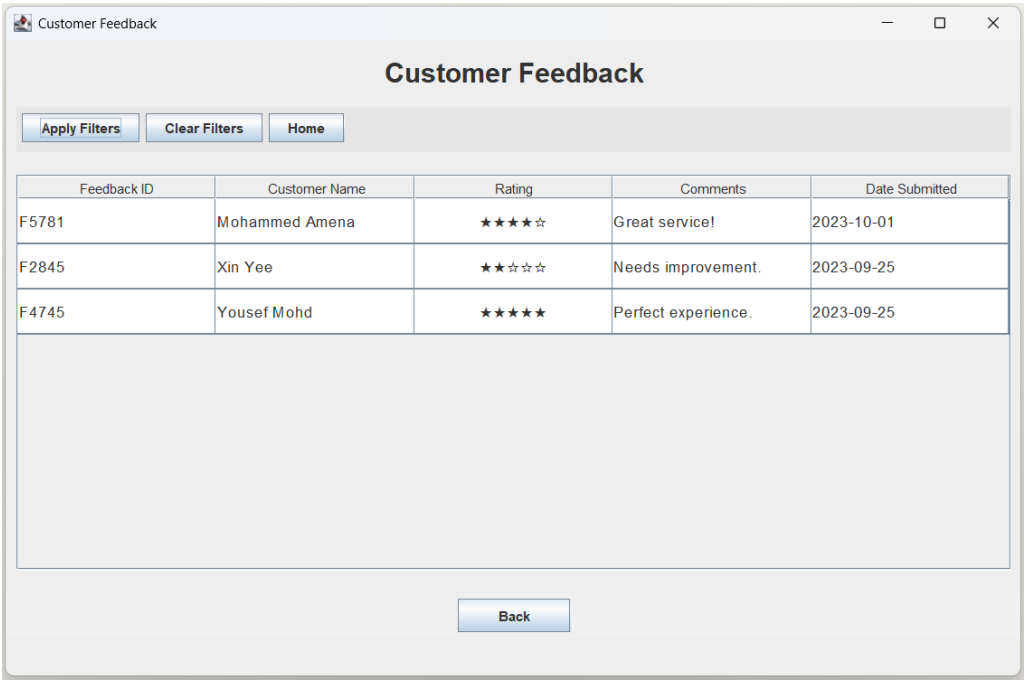




View Customer Feedback

This screen provides a centralized view of all customer feedback submitted by customer, enabling efficient analysis and management of customer sentiment and service quality.

- **Navigation:** The "Back" button allows the user to navigate to the previous screen in the application.
- **Apply Filters:** The "Apply Filters" button enables users to filter the displayed feedback data based on criteria such as:
  Date range (e.g., "Last month," "This year")
  Customer name (partial or exact match)
  Rating (e.g., "Positive," "Negative," "Neutral")
- **Clear Filters:** The "Clear Filters" button removes all applied filters, displaying the complete set of customer feedback.
- **Feedback List:** The screen presents a view of customer feedback entries, including:
  Feedback ID: A unique identifier for each feedback entry.
  Customer Name: The name or identifier of the customer who submitted the feedback.
  Rating: A visual representation of the customer's rating (e.g., star rating).
  Comments: The customer's written feedback or comments.
  Date Submitted: The date and time when the feedback was submitted.

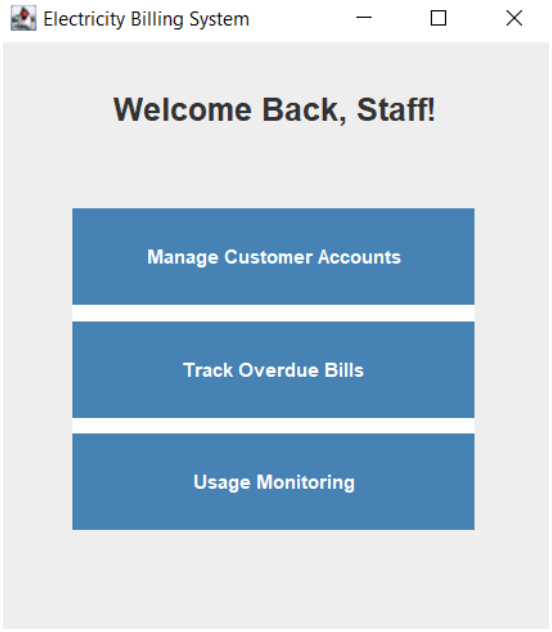


3.9 Subsystem 4 Screens: Staff

Main Screen

The main screen of the Staff interface allows the Staff to navigate between three tasks in the Electrical Billing System. It provides an intuitive and welcoming interface designed to simplify navigation. Below are the features of the main staff screen.

Welcome Message:
The staff is greeted with a welcoming message upon successful login as a staff.
Manage Customer Accounts:
The "Manage Customer Accounts" button takes staff to the customer account management page where they can view and update customer information.
Track Overdue Bills:
The "Track Overdue Bills" button navigates staff to overdue bills page to review and manage overdue bills.
Usage Monitoring:
The "Usage Monitoring" button provides access to monitor customer electricity usage trends.



Manage Customer Accounts Screen

The "Manage Customer Accounts" screen allows staff to manage and maintain a database of customer accounts with detailed information. This interface ensures that all customer records are easily accessible and modifiable.

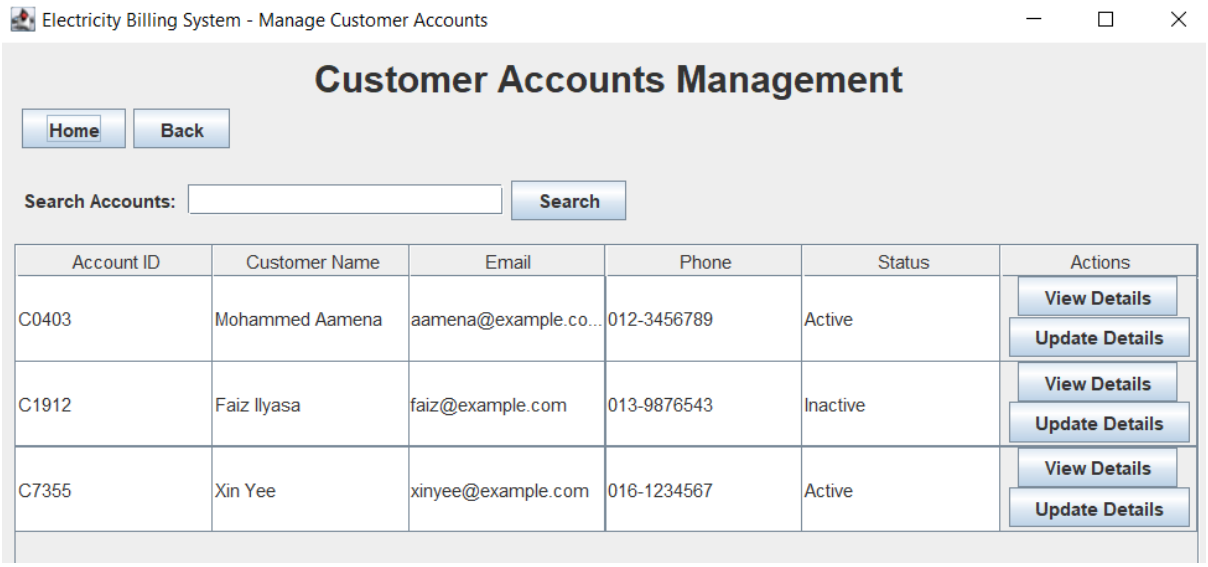
Home Button:
The "Home" button allows the staff to go back to the main screen.

Search Bar:
The staff can search for a specific customer account using their account ID or name. The screen will be filtered based on the search.

Customer Accounts List:
The table displays a list of customer accounts with columns such as:

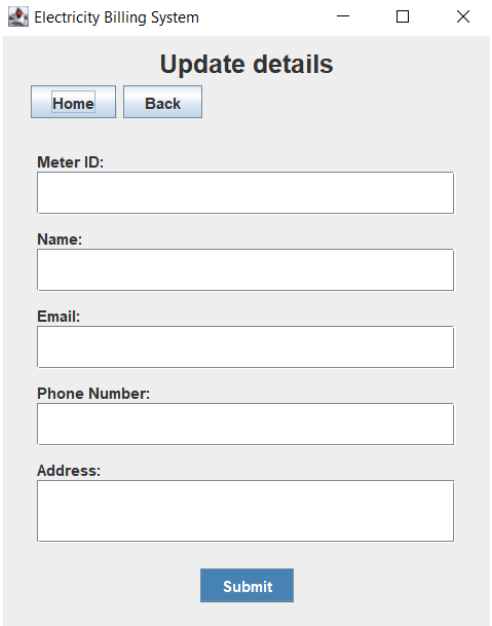
- **Account ID:** Unique identifier for each customer account.

- **Customer Name:** The name of the account holder.
- **Email:** The customer's email address.
- **Phone:** The customer's phone number.
- **Status:** Indicates the status of the customer account.
- **Actions:** A "View Details" button to view a more detailed information of the customer account, and an "Update Details" button for the staff to update the customer account details.



Update Details Screen

The staff can update the information of the customer account by clicking the "Update Details" button. The staff can then input the new details such as meter ID, name, email, phone number, and address of the customer.



Track Overdue Bills Screen

The "Track Overdue Bills" screen is designed to help staff monitor, verify, and take necessary actions to manage overdue bills in the system. It ensures that overdue payments are identified and handled effectively.

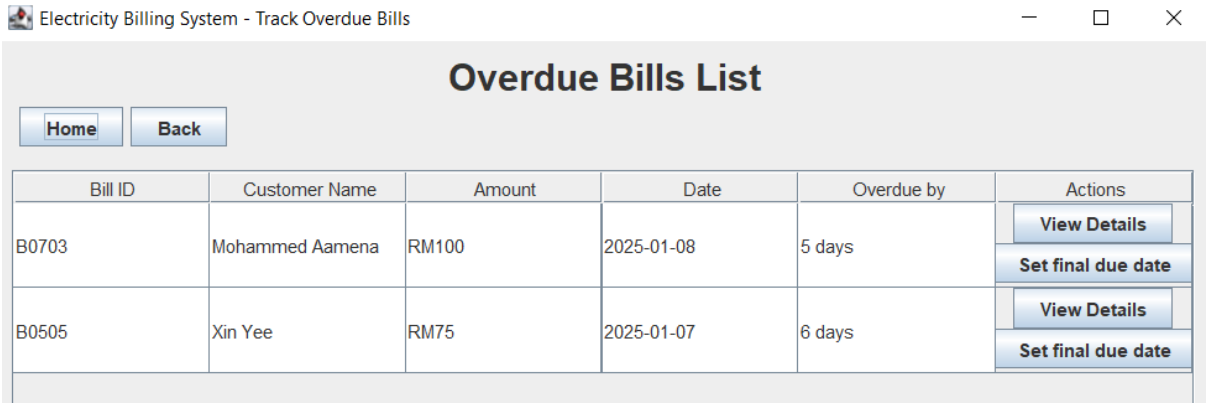
Home Button:

The "Home" button allows the staff to go back to the main screen.

Overdue Bills List:

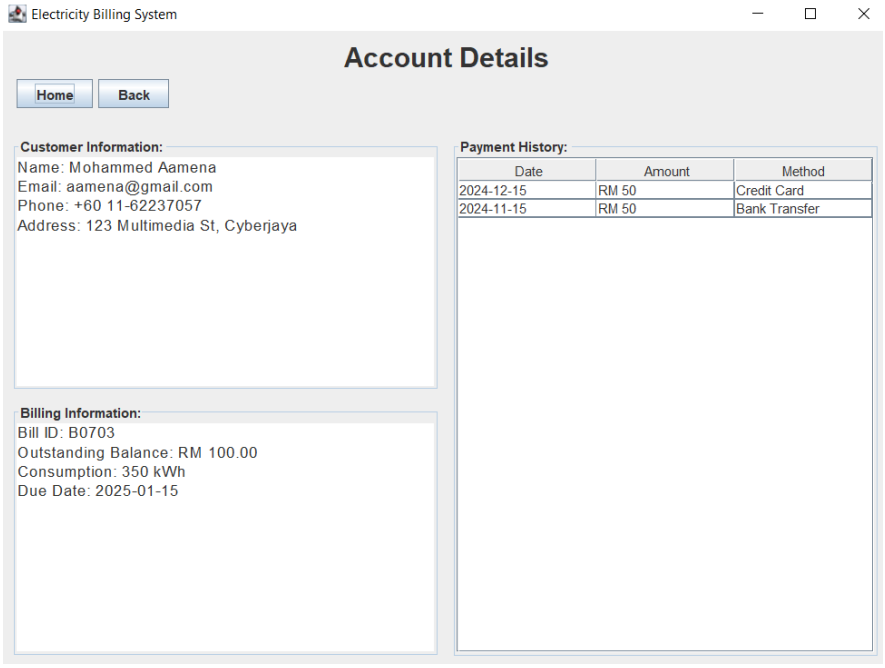
The table displays a list of overdue bills with columns such as:

- **Bill ID:** Unique identifier for each bill.
- **Customer Name:** The name of the account holder.
- **Amount:** The total amount due for the bill.
- **Date:** The date when the bill is generated.
- **Overdue by:** Indicates how long the bill has been overdue by.
- **Actions:** A "View Details" button to view a more detailed information of the bill details, and a "Set Final Due Date" button for the staff to set a new due date for the customer to pay their overdue bill.

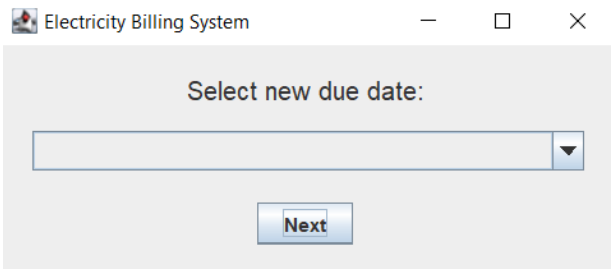


View Details Screen

The "View Details" screen is accessible from the "Manage Customer Accounts" section by the staff. It provides a detailed description of the customer information, their billing information and payment histories. Staff can review and verify the necessary information.



- **Set Final Due Date Screen**
  The "Set Final Due Date" screen prompts the staff to key-in a new due date from a drop-down menu. This will set the final due date for the customer to pay their overdue bill.

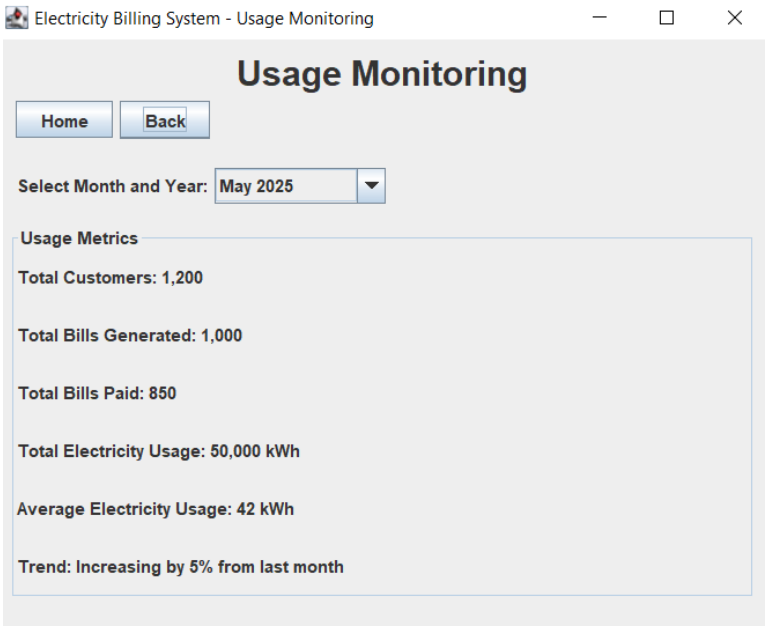


- **Usage Monitoring Screen**
  The "Usage Monitoring" screen offers the staff to track and analyze electricity usage trends for the overall system. This helps staff identify irregularities or patterns in usage.

  Home Button:
  The "Home" button allows the staff to go back to the main screen.

  Month and Year Filter:
  This allows the staff to filter the analytics based on a specific month and year.

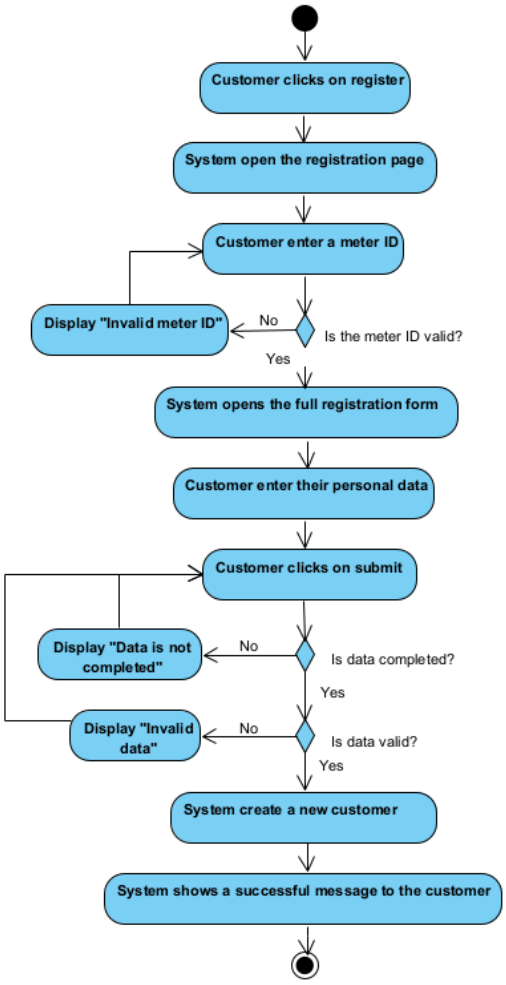
  Usage Metrics:
  The usage metrics section shows a breakdown of important analytics for the staff to view such as total customers, total bills generated, total bills paid, total electricity usage, average electricity usage, and trend.



3.10 Main Components

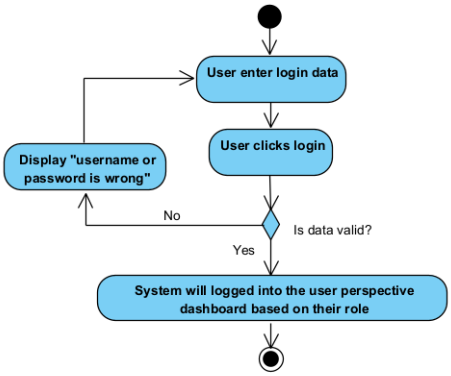
3.10.1 Component 1: System Registration

The activity diagram illustrates the workflow for the System Registration process. The process begins when the customer clicks on the "Register" button, initiating the opening of the registration page. The customer is then prompted to enter their Meter ID. The system validates the entered Meter ID. If valid, the full registration form is displayed, requiring the customer to enter their personal data. Upon clicking "Submit," the system validates the entered data for completeness and correctness. If all validations are successful, a new customer account is created, and a success message is displayed to the customer.



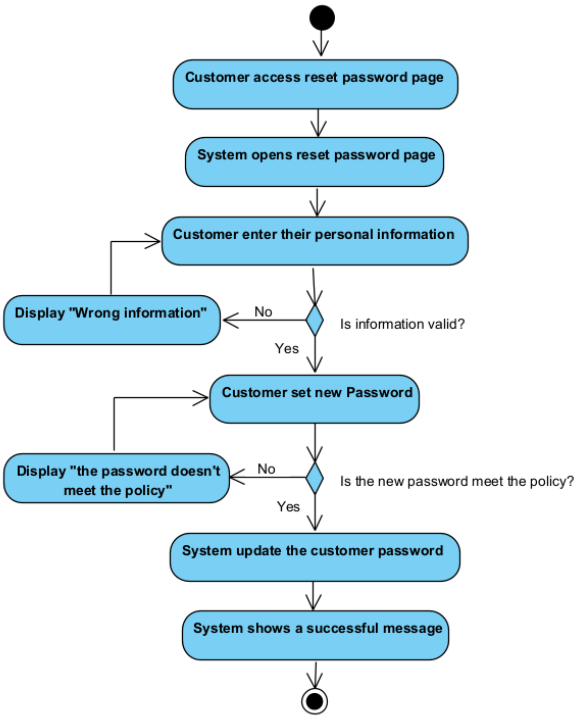
3.10.2 Component 2: Log in to the System

The activity diagram illustrates the user login process. The process begins with the user entering their login data, which typically includes their username and password. Once the user clicks the "Login" button, the system validates the provided data. If the data is invalid (e.g., incorrect username or password), the system displays an error message. However, if the data is valid, the system successfully logs in the user and redirects them to their respective dashboard based on their assigned role within the system.



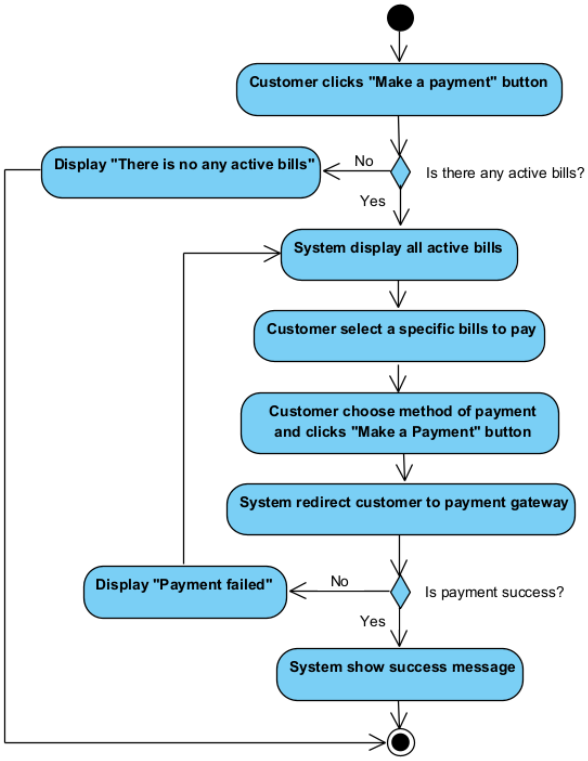
3.10.3 Component 3: Reset Password

The activity diagram depicts the process for resetting a forgotten password. The sequence begins with the customer initiating the process by accessing the "Reset Password" page. The system then presents the customer with a form to enter their personal information for verification. Following the entry of personal information, the system validates the provided details. If the information is deemed valid, the system proceeds to the stage where the customer sets a new password. Subsequently, the system enforces password policies, ensuring the new password meets the established security criteria. Upon successful validation of the new password, the system updates the customer's password in the database. The process concludes with the system displaying a success message to the customer, confirming the successful completion of the password reset.



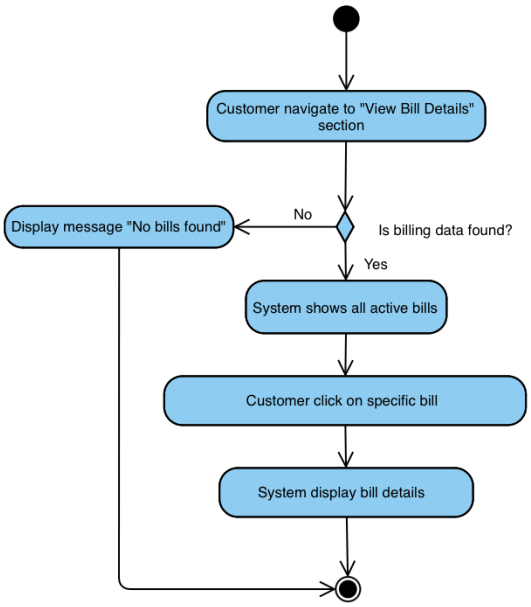
3.10.4 Component 4: Make a Payment and Receive Payment Receipt

The activity diagram illustrates the workflow for making a payment within the Electricity Billing System. Initiated by the customer clicking the "Make a Payment" button, the system first checks for the presence of any active bills. If no active bills are found, the system displays a message informing the customer. Conversely, if active bills exist, the system presents the customer with a list of active bills for selection. Subsequently, the customer chooses the desired bill and payment method and confirms the payment. The system then redirects the customer to the selected payment gateway to complete the transaction. Upon completion, the system verifies the payment status. If the payment is successful, a success message is displayed; otherwise, an error message is presented.



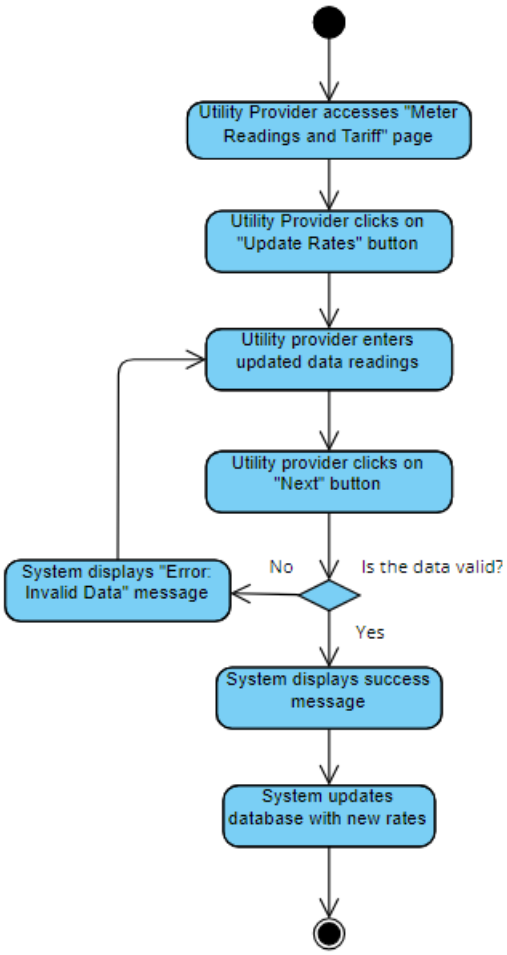
3.10.5 Component 5: View Bill Details

The activity diagram illustrates the process of viewing bill details for a customer. The process begins when the customer navigates to the "View Bill Details" section. The system then checks if there are any active bills associated with the customer. If no bills are found, the system displays a message indicating "No bills found." If active bills are found, the system displays a list of all active bills to the customer. The customer then selects a specific bill from the list. Finally, the system retrieves and displays the detailed information of the selected bill to the customer.



3.10.6 Component 6: Update Meter Reading and Tariff

The activity diagram shows the process of updating the meter readings and tariff rates for the utility provider. The process begins when the utility provider accesses the “Meter Readings and Tariff" page. Then, the system will present them with a form to update the rates data. Once the new data is entered by the utility provider, the system will check with the database to validate the data entered. If the data is invalid, an error message will be shown, otherwise the system will display a success message and update the database with the new rates.



3.10.7 Component 7: Track Overdue Bills

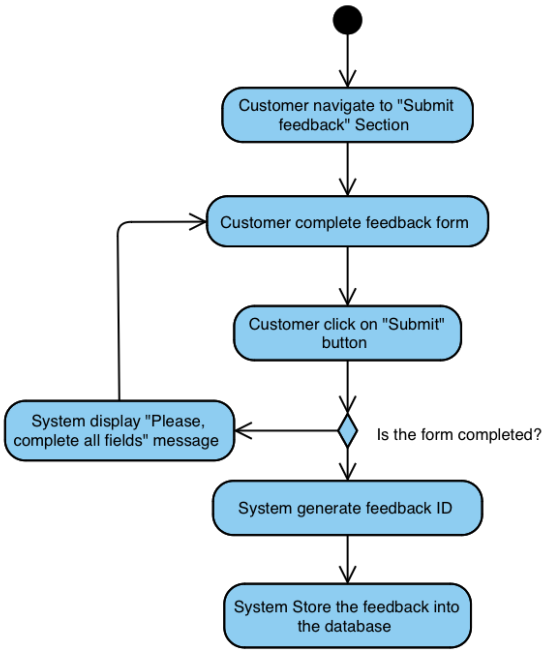
The activity diagram illustrates the tracking of overdue bills for the utility provider. The process begins when the utility provider accesses the “Track Overdue Bills" page. Then, the system will first check if there are any overdue bills. If there are no overdue bills, the system will display “No overdue bills found” and end the process. If there are overdue bills, the system displays them and the utility provider can verify them. The utility provider can then proceed to set a penalty amount to the customers who have overdue bills passed the final due date set by staff. If they have not passed the final due date, the system will display “Final due date not yet passed”. The utility provider will then enter the penalty amount for the system to save into the database and the customer will be notified.

3.10.8 Component 8: Generate Monthly Bills

The activity diagram shown illustrate the process of generating and delivering the electricity bills. The process begins when the utility provider schedules the date for bill generation within the system. Subsequently, the system receives the customer data from the database, including consumption details and relevant account information. The system calculates the bill amount based on the customer's consumption and applicable tariffs. The system generates a detailed bill and saves it into the database. The customer then receives a notification about the bill.

3.10.9 Component 9: Submit Feedback

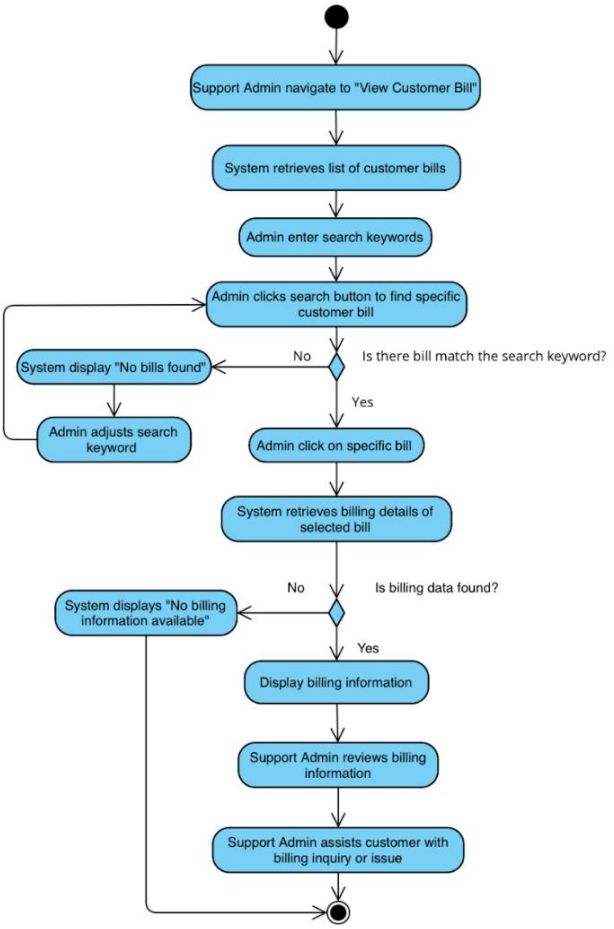
The activity diagram outlines the process of submitting customer feedback within the Electricity Billing System. The process begins when the customer navigates to the designated feedback submission section. Subsequently, the customer is required to complete the feedback form by providing the necessary information. Upon clicking the "Submit" button, the system validates the form for completeness. If any required fields are missing, the system displays an error message, prompting the customer to complete the form. If the form is complete, the system generates a unique identifier for the submitted feedback. Finally, the system stores the feedback, along with the associated identifier, within the system's database.



3.10.10 Component 10: View Customer Bills

The activity diagram depicts the workflow for the Support Admin when viewing and managing customer bills within the Electricity Billing System. The process commences with the Support Admin navigating to the "View Customer Bill" section. Subsequently, the system retrieves a comprehensive list of customer bills. The admin then proceeds to enter search

keywords to locate a specific customer bill and initiates the search by clicking the designated button. If no bills match the search criteria, the system displays a notification indicating "No bills found," prompting the admin to refine the search keywords. If a match is identified, the admin selects the specific bill. The system then retrieves the detailed billing information pertaining to the selected bill. If no billing information is available for the selected bill, the system displays a notification indicating "No billing information available" Conversely, if billing data is retrieved, the system presents the billing information to the Support Admin. Finally, the Support Admin reviews the billing information and proceeds to assist the customer with any inquiries or issues related to their bill.

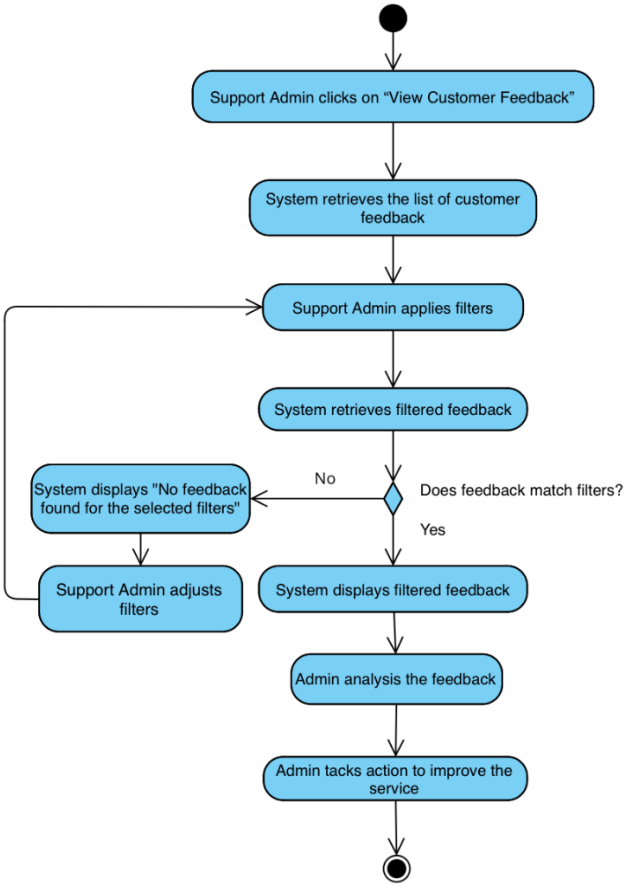


3.10.11 Component 11: View Customer Feedback and Analysis

The activity diagram illustrates the workflow for a Support Admin when viewing and analyzing customer feedback within the Electricity Billing System. The process begins with the Support Admin selecting the "View Customer Feedback" option. The system then retrieves and displays a comprehensive list of all submitted customer feedback.

The Support Admin applies filters to narrow down the feedback based on specific criteria, such as star rating, keywords, or comments. If no feedback matches the selected filters, the system displays a message stating, "No feedback found for the selected filters," prompting the Admin to modify the filter criteria.

If matching feedback entries are found, the system presents the filtered list to the Admin. The admin analyzes this feedback to identify trends, areas for improvement, and potential service issues. Based on the analysis, the admin takes appropriate actions to enhance the electricity system services.



3.10.12 Component 12: Submit and Resolve Customer Issue

The activity diagram below describes the workflow followed in the Electricity Billing System for addressing customer issues. The process starts with a customer going to the Support part of the site and finding the reporting form. The customer fills out the form, providing most everything that needs to be provided, as in what the issue is, time, date, and time of occurrence, and supporting information including any attachment that might help. When completed, it could be submitted via clicking at the bottom of the form the 'submit' button.

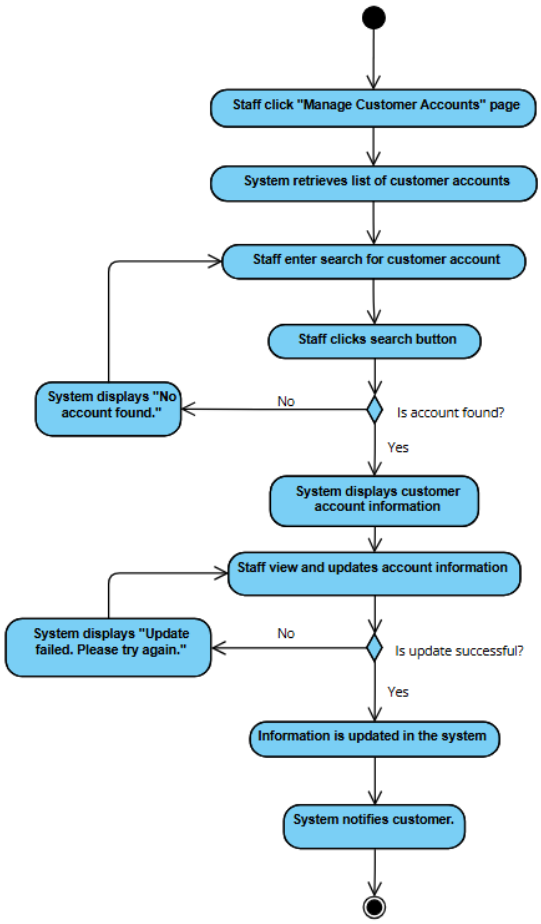
After submission, the system validates the form's completeness. Depending on the result of validation, if some required fields were indicated empty or they showed invalid data, an error message will appear alerting the customer to rectify this before submitting it again. If validation turns out to be successful, the system will identify the ticket uniquely to allow tracking of the issue and classify it as 'In Progress'.

At the same time, the system sends a confirmation message to the customer, acknowledging him for submitting the issue, along with the ticket ID which serves as reference to it. The system then assigns this particular issue to the support administrator who handles it. The administrator will look through the issue details, ascertain whatever needs to be done to resolve it, and then take necessary action. Such actions are troubleshooting, if need be, liaising with other departments, or even contacting the customer directly.

Once the issue is resolved, the respective administrator will change the system status of the issue to reflect that it is completed. The system then alerts the user that their request has been resolved, providing a summary or follow-up, if applicable. The process is then closed for resolution of issues pertaining to the Electricity Billing System.

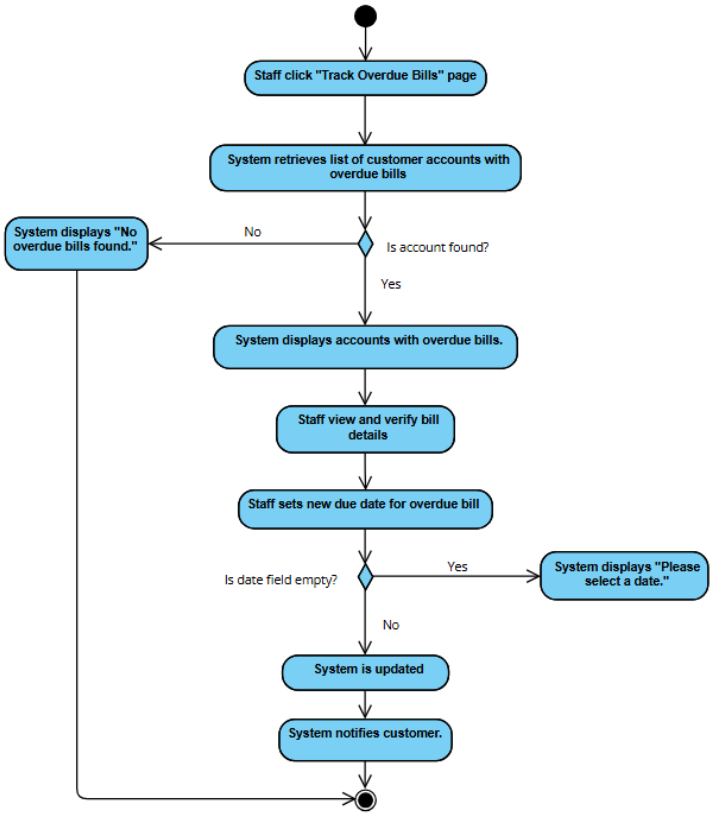
3.10.13 Component 13: Manage Customer Accounts

The staff can navigate to the “Manage Customer Accounts” page from the staff’s main menu. Once entered, the system will retrieve the list of customers accounts which will be displayed on the screen. For ease of managing accounts, staff can search for a specific account via the search field. The system checks if the search is valid and will display an error message if no accounts is found such as “No account found”. The staff can then view and update the customer account to make necessary changes. The system will validate the input and displays “Update failed. Please try again.” if update is unsuccessful. After the system has been updated, a notification will be sent to the customer.



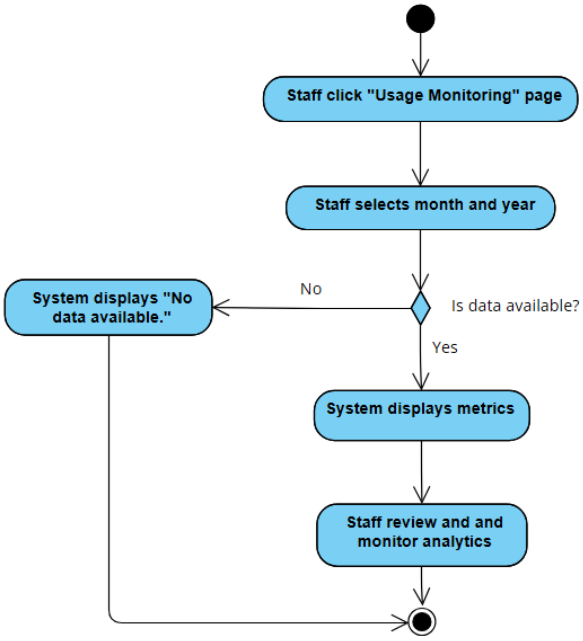
3.10.14 Component 14: Track Overdue Bills

Staff can access the “Track Overdue Bills” page from the staff’s main menu. The system will automatically retrieve a list of customer accounts with overdue bills. If no accounts with overdue bills are found, the system will display “No overdue bills found”. If there are overdue bills, the staff can view the bill details to monitor and verify overdue bills. The staff is also able to set a new due date for the overdue bill. The system will validate the date and if the date is invalid, it will display “Please select a valid date”. Upon a valid date, the system will be updated, and it will send the customer a notification.



3.10.15 Component 15: Usage Monitoring

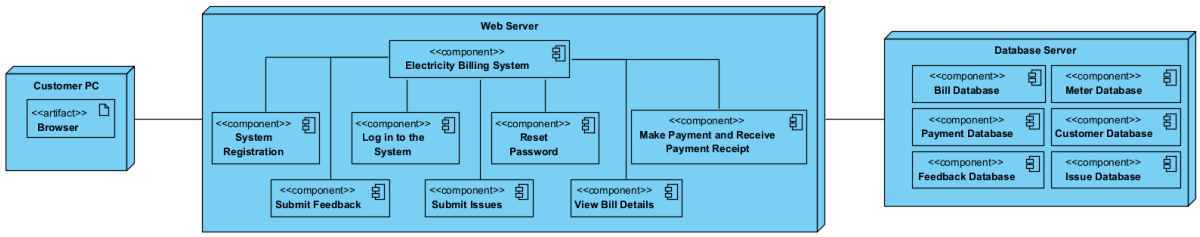
The staff can select the “Usage Monitoring” page from the staff’s main menu. The staff can then filter the usage metrics by month and year. If no data is available, the system will display “No data available”. If data is available, the system will display the metrics which the staff can review and monitor.



3.11 Deployment Diagram

3.11.1 Customer Deployment Diagram

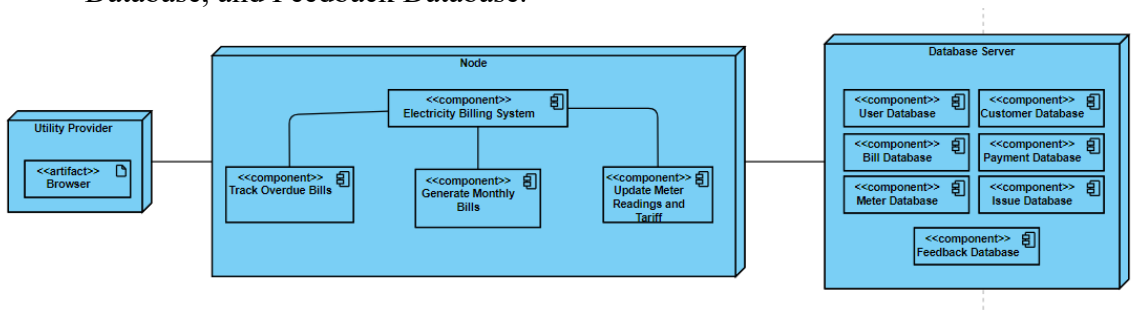
This deployment diagram illustrates the architecture of the Electricity Billing System. The system is comprised of a Web Server hosting the core application, which encompasses components such as System Registration, Log in to the System, Reset Password, Make Payment and Receive Payment Receipt, Submit Feedback, Submit Issues and View Bill Details. These components interact with a suite of databases, including Bill Database, Meter Database, Payment Database, Customer Database, Feedback Database, and Issue Database, for data storage and retrieval. The system is accessed by Customer PCs equipped with web browsers, enabling customers to interact with the application and perform various tasks such as registration, login, payment, feedback submission, and issue reporting.



3.11.2 *Utility Provider*

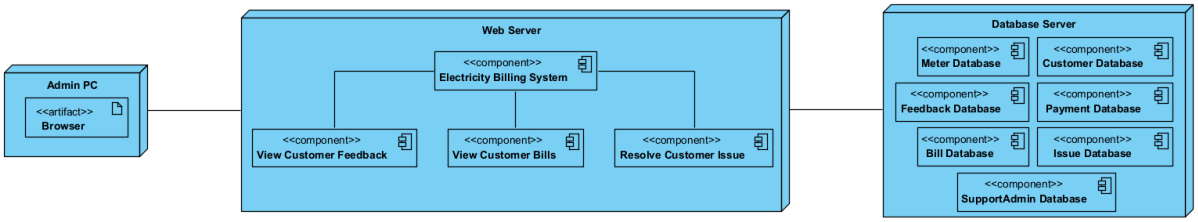
The Utility Provider Deployment Diagram shows components that can be accessed by the Utility Provider. When the utility provider logs into their PC, they can access three components which are Track Overdue Bills, Generate Monthly Bills, and Update Meter Readings and Tariffs. These components are supported by databases such as Utility Provider

Database, Customer Database, Bill Database, Payment Database, Meter Database, Issue Database, and Feedback Database.



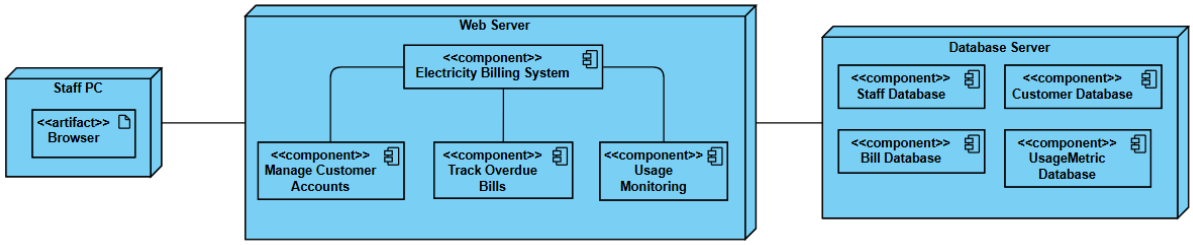
3.11.3 Support Admin

The Support Admin Deployment Diagram illustrates the architecture of the Electricity Billing System with a focus on the components accessible by support admins. The system is built around a central Web Server that hosts the application, which includes components such as View Customer Feedback, View Customer Bills, and Resolve Customer Issues. These components interface with a set of databases that store essential data for the system's operation. The databases include the Customer Database, Meter Database, Feedback Database, Payment Database, Bill Database, Issue Database, and SupportAdmin Database to retrieve the necessary data. The entire system can be accessed via Admin PCs with web browsers so that the members of administration can interface with the application to do their administrative work.



3.11.4 Staff

The Staff Deployment Diagram shows that when a staff member uses their PC to log into the Electricity Billing System, they can access three components which are Manage Customer Accounts, Track Overdue Bills, and Usage Monitoring. Each of these components are linked to their corresponding databases such as Staff Database, Customer Database, Bill Database, and UsageMetric Database to retrieve necessary data.



4 Implementation

4.1 Development Environment

4.1.1 Software Models

```python
import uuid
from django.db import models
from django.contrib.auth.models import AbstractUser, Group
from django.core.validators import MinValueValidator, MaxValueValidator, RegexValidator, EmailValidator
from django.core.exceptions import ValidationError


class User(AbstractUser):
    """
    Custom User model with role-based access control.
    """
    ROLE_CHOICES = [
        ('CUSTOMER', 'Customer'),
        ('SUPPORT_ADMIN', 'Support Admin'),
        ('STAFF', 'Staff'),
        ('UTILITY_PROVIDER', 'Utility Provider'),
    ]

    role = models.CharField(max_length=20, choices=ROLE_CHOICES, default='CUSTOMER')
    phone_number = models.CharField(
        max_length=13,
        blank=True,
        null=True,
        validators=[RegexValidator(regex=r'^\+?1?\d{9,15}$', message="Phone number must be in the format '+999999999'.")]
    )
    address = models.CharField(max_length=120, blank=True, null=True)

    def __str__(self):
        return f"{self.username} ({self.get_role_display()})"

    def assign_role(self, role_name):
        """
        Assigns a role to the user and adds them to the corresponding group.
        """
        if role_name not in dict(self.ROLE_CHOICES):
            raise ValueError(f"Invalid role name: {role_name}")
        group, created = Group.objects.get_or_create(name=role_name)
        self.groups.add(group)
        self.save()
```

```python
class BaseProfile(models.Model):
    """
    Abstract base model for all profile types.
    """
    user = models.OneToOneField(User, on_delete=models.CASCADE, related_name="%(class)s_profile")
    #identifier = models.UUIDField(default=uuid.uuid4, editable=False, unique=True)

    class Meta:
        abstract = True

    def __str__(self):
        return f"{self.__class__.__name__}: {self.user.username}"


class Customer(BaseProfile):
    customer_id = models.CharField(max_length=10, unique=True, null=True, blank=True)  # Ensure it's added
    customer_name = models.CharField(max_length=255)
    customer_address = models.CharField(max_length=255, blank=True, null=True)
    customer_email = models.EmailField(
        max_length=255,
        blank=True,
        null=True,
        validators=[EmailValidator(message="Enter a valid email address.")]
    )
    customer_phone = models.CharField(
        max_length=13,
        blank=True,
        null=True,
        validators=[RegexValidator(regex=r'^\+?1?\d{9,15}$', message="Phone number must be in the format '+999999999'.")]
    )
    meter_id = models.CharField(max_length=6, blank=True, null=True)

    class Meta:
        verbose_name = "Customer"
        verbose_name_plural = "Customers"

    def __str__(self):
        return f"{self.customer_id} - {self.customer_name}"
```

```python
class UtilityProvider(BaseProfile):
    """
    Model for Utility Provider profiles.
    """
    up_name = models.CharField(max_length=100)
    up_phone = models.CharField(
        max_length=15,
        validators=[RegexValidator(regex=r'^\+?1?\d{9,15}$', message="Phone number must be in the format '+999999999'.")]
    )
    up_email = models.EmailField(validators=[EmailValidator(message="Enter a valid email address.")])

    class Meta:
        verbose_name = "Utility Provider"
        verbose_name_plural = "Utility Providers"


class SupportAdmin(BaseProfile):
    """
    Model for Support Admin profiles.
    """
    admin_id = models.CharField(max_length=6, unique=True, blank=True, null=True)  # Unique admin ID
    admin_name = models.CharField(max_length=100)
    admin_phone = models.CharField(
        max_length=15,
        validators=[RegexValidator(regex=r'^\+?1?\d{9,15}$', message="Phone number must be in the format '+999999999'.")]
    )
    admin_email = models.EmailField(validators=[EmailValidator(message="Enter a valid email address.")])

    class Meta:
        verbose_name = "Support Admin"
        verbose_name_plural = "Support Admins"


class Staff(BaseProfile):
    """
    Model for Staff profiles.
    """
    staff_name = models.CharField(max_length=100)
    staff_phone = models.CharField(
        max_length=15,
        validators=[RegexValidator(regex=r'^\+?1?\d{9,15}$', message="Phone number must be in the format '+999999999'.")]
    )
    staff_email = models.EmailField(validators=[EmailValidator(message="Enter a valid email address.")])

    class Meta:
        verbose_name = "Staff"
        verbose_name_plural = "Staff"


class Meter(models.Model):
    """
    Model for Meter details.
    """
    meter_id = models.CharField(
        primary_key=True,
        max_length=6,
        validators=[RegexValidator(regex=r'^M\d{5}$', message="Meter ID must be in the format MXXXXX.")]
    )
    meter_reading = models.IntegerField()
    tariff_rate = models.DecimalField(max_digits=10, decimal_places=2)
    customer = models.OneToOneField(Customer, on_delete=models.SET_NULL, null=True, related_name="meter")

    def __str__(self):
        return f"Meter {self.meter_id}"

    class Meta:
        verbose_name = "Meter"
        verbose_name_plural = "Meters"
        unique_together = ("meter_id", "customer")

```

```python
class Bill(models.Model):
    """
    Model for Bills.
    """
    STATUS_CHOICES = [
        (True, 'Paid'),
        (False, 'Unpaid')
    ]

    bill_id = models.CharField(primary_key=True, max_length=6)
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    amount = models.DecimalField(max_digits=10, decimal_places=2)
    due_date = models.DateField()
    paid = models.BooleanField(choices=STATUS_CHOICES, default=False)
    creation_date = models.DateField(auto_now_add=True)
    penalty_fee = models.DecimalField(max_digits=10, decimal_places=2, default=0.00)

    def __str__(self):
        return f"Bill {self.bill_id} (Customer: {self.customer.customer_name})"

    class Meta:
        verbose_name = "Bill"
        verbose_name_plural = "Bills"


class Payment(models.Model):
    """
    Model for Payments.
    """
    PAYMENT_METHODS = [
        ('TnG', 'Touch n Go'),
        ('FPX', 'FPX'),
        ('Credit Card', 'Credit Card')
    ]

    payment_id = models.CharField(primary_key=True, max_length=6)
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    payment_date = models.DateField(auto_now_add=True)
    payment_method = models.CharField(max_length=15, choices=PAYMENT_METHODS)
    amount = models.DecimalField(max_digits=10, decimal_places=2)
```

```
194         def __str__(self):
195             return f"Payment {self.payment_id} (Customer: {self.customer.customer_name})"
196 
197         class Meta:
198             verbose_name = "Payment"
199             verbose_name_plural = "Payments"
200 
201 
202     class Issue(models.Model):
203         """
204         Model for Customer Issues.
205         """
206         STATUS_CHOICES = [
207             ('open', 'Open'),
208             ('in_progress', 'In Progress'),
209             ('resolved', 'Resolved'),
210         ]
211 
212         issue_id = models.CharField(primary_key=True, max_length=6)
213         customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
214         title = models.CharField(max_length=200)
215         description = models.TextField(max_length=300)
216         status = models.CharField(max_length=15, choices=STATUS_CHOICES, default='open')
217         ticket_id = models.CharField(max_length=6, unique=True)
218         support_admin = models.ForeignKey(SupportAdmin, on_delete=models.SET_NULL, null=True, blank=True)
219 
220         def __str__(self):
221             return f"Issue {self.issue_id} (Customer: {self.customer.customer_name})"
222 
223         class Meta:
224             verbose_name = "Issue"
225             verbose_name_plural = "Issues"
226 
```

```python
class Feedback(models.Model):
    """
    Model for Customer Feedback.
    """
    feedback_id = models.CharField(primary_key=True, max_length=6)
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    rating = models.IntegerField(validators=[MinValueValidator(1), MaxValueValidator(5)])
    comment = models.CharField(max_length=200)
    feedback_date = models.DateField(auto_now_add=True)

    def __str__(self):
        return f"Feedback {self.feedback_id} (Customer: {self.customer.customer_name})"

    class Meta:
        verbose_name = "Feedback"
        verbose_name_plural = "Feedbacks"


class Item(models.Model):
    item_id = models.CharField(primary_key=True, max_length=10)
    item_name = models.TextField()
    item_description = models.TextField(null=True, blank=True)

    def __str__(self):
        return f"Item {self.item_id}: {self.item_name}"

```

```python
class UsageMetrics(models.Model):
    total_customers = models.IntegerField(default=0)
    total_bills = models.IntegerField(default=0)
    total_usage = models.DecimalField(max_digits=10, decimal_places=2, default=0.0)
    avg_usage = models.DecimalField(max_digits=10, decimal_places=2, default=0.0)
    last_updated = models.DateTimeField(auto_now=True)

    def __str__(self):
        return f"Usage Metrics (Last Updated: {self.last_updated})"
```

4.1.2 Software Views

```python
from random import randint
from django.shortcuts import render, redirect

# Create your views here.
from django.http import Http404, HttpRequest
from django.template import RequestContext ##--
from django.http import HttpRequest
from datetime import datetime

from django.contrib.auth.decorators import login_required
from django.contrib import messages
from django.shortcuts import render, get_object_or_404
from django.contrib.auth.decorators import login_required, user_passes_test
from django.contrib.auth import authenticate, login as auth_login
from django.shortcuts import render, redirect
from app.models import User, Customer, Staff, SupportAdmin, UtilityProvider
from .forms import BootstrapAuthenticationForm
from django.contrib.auth import login as auth_login
from .forms import PasswordChangeForm
from django.contrib.auth import update_session_auth_hash
from django.contrib.auth.forms import AuthenticationForm
from django.http import HttpResponse
from .models import Customer, UtilityProvider, SupportAdmin, Staff
from django.db.models.signals import post_save
from django.dispatch import receiver
from .models import Customer, User
from django.contrib.auth import views as auth_views

from .models import (
    Customer, Bill, Payment,
    Issue, Feedback, SupportAdmin,
    UtilityProvider, Staff, Meter
)
```

```python
from .models import (
    Customer, Bill, Payment,
    Issue, Feedback, SupportAdmin,
    UtilityProvider, Staff, Meter
)

from .forms import RegistrationForm, RegistrationStep2Form, ResetPasswordForm

# Session keys
CUSTOMER_SESSION_KEY = 'customer_id'
ADMIN_SESSION_KEY = 'admin_id'
STAFF_SESSION_KEY = 'staff_id'
UTILITY_SESSION_KEY = 'utility_id'

# ========================
# Common Helper Functions
# ========================
def get_current_user(request):
    """Returns the logged-in user based on session"""
    if CUSTOMER_SESSION_KEY in request.session:
        return Customer.objects.get(customer_id=request.session[CUSTOMER_SESSION_KEY])
    if ADMIN_SESSION_KEY in request.session:
        return SupportAdmin.objects.get(admin_id=request.session[ADMIN_SESSION_KEY])
    if STAFF_SESSION_KEY in request.session:
        return Staff.objects.get(staff_id=request.session[STAFF_SESSION_KEY])
    if UTILITY_SESSION_KEY in request.session:
        return UtilityProvider.objects.get(utility_id=request.session[UTILITY_SESSION_KEY])
    return None
```

```python
def role_based_redirect(request):
    if request.user.is_authenticated:
        role = getattr(request.user, 'role', None)  # Safely get the role attribute
        if role:
            role = role.upper()  # Make sure it's in uppercase
            if role == 'CUSTOMER':
                return redirect('customer_dashboard')
            elif role == 'SUPPORT_ADMIN':
                return redirect('admin_dashboard')
            elif role == 'STAFF':
                return redirect('staff_dashboard')
            elif role == 'UTILITY_PROVIDER':
                return redirect('utility_dashboard')

    # If the user is not authenticated or the role is invalid, redirect to the home page
    return redirect('home')


def migrate_users():
    for customer in Customer.objects.all():
        User.objects.create(
            username=customer.username,
            password=customer.password,  # Store securely in production!
            role="CUSTOMER",
            email=customer.customer_email,
            phone_number=customer.customer_phone,
            address=customer.customer_address,
        )
```

```
    for staff in Staff.objects.all():
        User.objects.create(
            username=staff.username,
            password=staff.password,
            role="STAFF",
            email=staff.staff_email,
            phone_number=staff.staff_phone,
        )

    for admin in SupportAdmin.objects.all():
        User.objects.create(
            username=admin.username,
            password=admin.password,
            role="ADMIN",
            email=admin.admin_email,
            phone_number=admin.admin_phone,
        )

    for provider in UtilityProvider.objects.all():
        User.objects.create(
            username=provider.username,
            password=provider.password,
            role="UTILITY_PROVIDER",
            email=provider.up_email,
            phone_number=provider.up_phone,
        )
    print("User migration completed.")

```

```python
# ========================
# Authentication Views
# ========================
def login(request):
    if request.method == 'POST':
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            username = form.cleaned_data['username']
            password = form.cleaned_data['password']
            user = authenticate(username=username, password=password)

            if user is not None:
                # Log the user in
                auth_login(request, user)

                # Redirect based on the role
                return role_based_redirect(request)  # Uses the existing helper function
            else:
                return HttpResponse('Invalid username or password', status=401)
        else:
            return HttpResponse('Invalid form data', status=400)

    form = AuthenticationForm()
    return render(request, 'app/login.html', {'form': form})



def logout(request):
    if request.method == "GET":
        logout(request)
        return redirect('home')

```

```
152  def home(request):
153      """Renders the home page."""
154      assert isinstance(request, HttpRequest)
155
156      # Render the same page regardless of login status
157      return render(
158          request,
159          'app/index.html',
160          {
161              'title': 'Home Page',
162              'year': datetime.now().year,
163          }
164      )
165
166  def contact(request):
167      """Renders the contact page."""
168      assert isinstance(request, HttpRequest)
169      return render(
170          request,
171          'app/contact.html',
172          {
173              'title':'Contact',
174
175              'year':datetime.now().year,
176          }
177      )
```

```python
def about(request):
    """Renders the about page."""
    assert isinstance(request, HttpRequest)
    return render(
        request,
        'app/about.html',
        {
            'title':'Electricity Billing System',
            'message':'This application processes ...',
            'year':datetime.now().year,
        }
    )


# ========================
# Menu View
# ========================
@login_required
def menu(request):
    user = get_current_user(request)
    if not user:
        return redirect('login')

    context = {
        'title': 'Main Menu',
        'is_customer': isinstance(user, Customer),
        'is_support_admin': isinstance(user, SupportAdmin),
        'is_staff': isinstance(user, Staff),
        'is_utility': isinstance(user, UtilityProvider),
        'user': user,
        'year': datetime.now().year,
    }
    return render(request, 'app/menu.html', context)
```

```
213  def registration(request):
214      """
215      Step 1: Validate meter ID and initiate registration process.
216      """
217      if request.method == 'POST':
218          meter_id = request.POST.get('meter_id')
219
220          # Validate Meter ID
221          try:
222              # Check if meter ID exists and is not already assigned to a customer
223              meter = Meter.objects.get(meter_id=meter_id)
224              if hasattr(meter, 'customer'):
225                  return render(request, 'customer/registration.html', {
226                      'error': 'This meter is already registered to a customer.'
227                  })
228
229              # Store meter ID in session for step 2
230              request.session['meter_id'] = meter_id
231              return redirect('registration_step2')
232
233          except Meter.DoesNotExist:
234              return render(request, 'customer/registration.html', {
235                  'error': 'Invalid meter ID. Please check and try again.'
236              })
237
238      return render(request, 'customer/registration.html')
239
```

```python
def registration_step2(request):
    """
    Step 2: Complete customer registration.
    """
    meter_id = request.session.get('meter_id')
    if not meter_id:
        return redirect('registration')  # Redirect if no meter ID in session

    if request.method == 'POST':
        form = RegistrationStep2Form(request.POST)
        if form.is_valid():
            # Create Customer
            customer = Customer.objects.create(
                customer_id=f"C{Customer.objects.count() + 1:04}",  # Auto-generate customer ID
                username=form.cleaned_data['username'],
                password=form.cleaned_data['password'],  # Note: Hash password in production
                customer_name=form.cleaned_data['customer_name'],
                customer_address=form.cleaned_data['customer_address'],
                customer_phone=form.cleaned_data['customer_phone'],
                customer_email=form.cleaned_data['customer_email'],
                meter_id=meter_id
            )

            # Log the user in (optional, if using Django's auth system)
            # login(request, customer)

            # Clear session data
            del request.session['meter_id']

            return redirect('dashboard')
    else:
        form = RegistrationStep2Form()

    return render(request, 'customer/registration_step2.html', {'form': form})

```

```python
def reset_password(request):
    """
    Handle password reset requests.
    """
    if request.method == 'POST':
        form = ResetPasswordForm(request.POST)
        if form.is_valid():
            email = form.cleaned_data['email']
            try:
                customer = Customer.objects.get(customer_email=email)
                # Send password reset email (implement email logic here)
                # Example: send_reset_email(customer)
                return redirect('password_reset_sent')
            except Customer.DoesNotExist:
                form.add_error('email', 'No account found with this email address.')
    else:
        form = ResetPasswordForm()

    return render(request, 'customer/reset_password.html', {'form': form})

```

```python
# ---------------------------------------------
# Customer Dashboard Views
# ---------------------------------------------
def is_customer(user):
    return user.is_authenticated and user.role == 'CUSTOMER'

@login_required
@user_passes_test(is_customer)
def customer_dashboard(request):
    try:
        # Get the customer profile of the logged-in user
        customer_profile = Customer.objects.get(user=request.user)
    except Customer.DoesNotExist:
        messages.error(request, "Customer profile does not exist.")
        return redirect('home')  # Or redirect to a different page, like registration

    # Pass the customer profile to the template
    return render(request, 'app/customer_dashboard.html', {'customer': customer_profile})




@login_required
def view_bills(request):
    # Retrieve bills for the logged-in customer
    customer = request.user.customer_profile
    bills = Bill.objects.filter(customer=customer)
    return render(request, 'bills.html', {'bills': bills})

```

```python
@login_required
def make_payment(request, bill_id):
    customer = get_current_user(request)
    if not customer or not isinstance(customer, Customer):
        return redirect('login')

    bill = get_object_or_404(Bill, bill_id=bill_id)
    if request.method == 'POST':
        Payment.objects.create(
            payment_id=f"P{Payment.objects.count()+1:05}",
            customer=customer,
            payment_date=datetime.now().date(),
            payment_method=request.POST.get('method'),
            amount=bill.amount
        )
        bill.paid = True
        bill.save()
        return redirect('payment_confirmation', bill_id=bill_id)

    return render(request, 'customer/make_payment.html', {'bill': bill})
##

@login_required
def payment_receipt(request, bill_id):
    bill = Bill.objects.get(id=bill_id)
    payment = Payment.objects.get(bill=bill)
    return render(request, 'customer/payment_receipt.html', {'payment': payment})
```

```python
@login_required
def submit_feedback(request):
    if request.method == 'POST':
        rating = request.POST.get('rating')
        comments = request.POST.get('comments')
        Feedback.objects.create(user=request.user, rating=rating, comments=comments)
        return redirect('dashboard')
    return render(request, 'customer/submit_feedback.html')

@login_required
def submit_issue(request):
    if request.method == 'POST':
        title = request.POST.get('issue_title')
        description = request.POST.get('issue_description')
        attachments = request.FILES.get('attachments')

        Issue.objects.create(
            user=request.user,
            title=title,
            description=description,
            attachments=attachments
        )
        return redirect('dashboard')
    return render(request, 'customer/submit_issue.html')

@login_required
def change_password(request):
    if request.method == 'POST':
        form = PasswordChangeForm(request.POST)
        if form.is_valid():
            old_password = form.cleaned_data['old_password']
            new_password = form.cleaned_data['new_password']

            # Verify old password
            if request.user.check_password(old_password):
                request.user.set_password(new_password)
                request.user.save()
                update_session_auth_hash(request, request.user)  # Keep the user logged in
                return redirect('customer_dashboard')  # Redirect to dashboard
            else:
                form.add_error('old_password', 'Incorrect old password.')
    else:
        form = PasswordChangeForm()

    return render(request, 'app/change_password.html', {'form': form})
```

```
# Check if the user is a Support Admin
def is_support_admin(user):
    return user.role == 'SUPPORT_ADMIN'

# View for the Support Admin Dashboard
@user_passes_test(is_support_admin)
def admin_dashboard(request):
   return render(request, 'app/admin_dashboard.html')
   ''' try:
        # Get the support admin profile for the logged-in user
        support_admin_profile = SupportAdmin.objects.get(user=request.user)
    except SupportAdmin.DoesNotExist:
        messages.error(request, "Support admin profile does not exist.")
        return redirect('home')  # Or redirect to another page like registration

    try:
        # Get the relevant data for the dashboard
        issues = support_admin_profile.get_assigned_issues()
        feedback = support_admin_profile.get_feedback()
    except Exception as e:
        messages.error(request, f"An error occurred while fetching dashboard data: {str(e)}")
        return redirect('home')

    # Pass the profile data to the template
    return render(request, 'app/admin_dashboard.html', {
        'support_admin': support_admin_profile,
        'issues': issues,
        'feedback': feedback,
    })'''
```

```python
class CustomAdminLoginView(auth_views.LoginView):
    template_name = 'admin/login.html'

    def dispatch(self, request, *args, **kwargs):
        if request.user.is_authenticated:
            if request.user.is_staff:
                return redirect('/admin/')  # Redirect to Django admin if staff
            else:
                return redirect('admin_dashboard')  # Redirect to custom admin dashboard
        return super().dispatch(request, *args, **kwargs)


@login_required
@user_passes_test(is_support_admin)
def view_bills_admin(request):
    bills = Bill.objects.all().order_by('-due_date')
    for bill in bills:
        print(f"Bill ID: {bill.bill_id}")  # Debug line
    return render(request, 'app/admin_viewBill.html', {'bills': bills})


@login_required
@user_passes_test(is_support_admin)
def bill_details(request, bill_id):
    # Fetch the bill details
    bill = get_object_or_404(Bill, bill_id=bill_id)
    return render(request, 'app/bill_details.html', {'bill': bill})


@login_required
def customer_issues(request):
        # Fetch all issues without login required
    issues = Issue.objects.all()
    return render(request, 'app/admin_viewIssue.html', {'issues': issues})
```

```python
@login_required
def update_issue_status(request, issue_id):
    admin = get_current_user(request)
    if not admin or not isinstance(admin, SupportAdmin):
        return redirect('login')

    issue = get_object_or_404(Issue, issue_id=issue_id)
    if request.method == 'POST':
        issue.status = request.POST.get('status')
        issue.save()
        return redirect('manage_issues')

    return render(request, 'support/update_issue.html', {'issue': issue})

@login_required
@user_passes_test(is_support_admin)
def view_feedback(request):
    # Fetch the feedbacks from the database
    feedbacks = Feedback.objects.all().order_by('-feedback_date')

    # Render the feedbacks in the template
    return render(request, 'app/admin_viewFeedback.html', {'feedbacks': feedbacks})
```

```python
# ========================
# Staff Views
# ========================
def staff_dashboard(request):
    # Get the staff profile of the logged-in user
    staff_profile = Staff.objects.get(user=request.user)

    # Pass the staff profile to the template
    return render(request, 'app/staff_dashboard.html', {'staff': staff_profile})

def utility_dashboard(request):
    # Get the utility provider profile of the logged-in user
    utility_provider_profile = UtilityProvider.objects.get(user=request.user)

    # Pass the utility provider profile to the template
    return render(request, 'app/utility_dashboard.html', {'utility_provider': utility_provider_profile})

@receiver(post_save, sender=User)
def create_customer_profile(sender, instance, created, **kwargs):
    if created and instance.role == 'CUSTOMER':
        Customer.objects.create(user=instance)

@receiver(post_save, sender=User)
def save_customer_profile(sender, instance, **kwargs):
    if instance.role == 'CUSTOMER':
        instance.customer_profile.save()


def generate_unique_ticket_id():
    while True:
        # Generate a new ticket_id (you can customize this)
        ticket_id = f"T{str(randint(10000, 99999))}"
        if not Issue.objects.filter(ticket_id=ticket_id).exists():
            return ticket_id


def create_issue(customer, title, description, status):
    ticket_id = generate_unique_ticket_id()  # Generate unique ticket_id
    Issue.objects.create(
        customer=customer,
        title=title,
        description=description,
        status=status,
        ticket_id=ticket_id
    )
    print(f"Issue created with ticket_id: {ticket_id}")

```

```python
# ========================
# Utility Provider Views
# ========================
def utility_dashboard(request):
    overdue_count = Bill.objects.filter(paid=False, due_date__lt=timezone.now()).count()
    total_customers = Meter.objects.count()
    latest_tariff = Tariff.objects.latest('updated_at')

    context = {
        'overdue_count': overdue_count,
        'total_customers': total_customers,
        'latest_tariff': latest_tariff,
    }
    return render(request, 'app/utility_dashboard.html', context)

def meter_readings(request):
    tariffs = Tariff.objects.all()
    return render(request, "app/meter_readings.html", {"tariffs": tariffs})

@login_required
def update_tariff(request, category=None):
    if request.method == "POST":
        if not category:
            category = request.POST.get("category")

        new_rate = request.POST.get("new_rate")

        if not category or not new_rate:
            messages.error(request, "All fields are required.")
            return redirect("meter_readings")

        try:
            category = int(category)
            new_rate = float(new_rate)
```

```
            if new_rate <= 0:
                messages.error(request, "Tariff rate must be a positive number.")
                return redirect("meter_readings")

            # Retrieve and update the tariff
            tariff = get_object_or_404(Tariff, category=category)
            tariff.rate = new_rate
            tariff.save()

            messages.success(request, f"Tariff for category {category} updated successfully!")

        except ValueError:
            messages.error(request, "Invalid input. Please enter a valid number.")

    return redirect("meter_readings")

@login_required
def overdue_bills(request):
    overdue_bills = Bill.objects.filter(paid=False, due_date__lt=timezone.now()).select_related('customer')
    return render(request, 'app/overdue_bills.html', {'overdue_bills': overdue_bills})

def account_details(request, bill_id):
    bill = get_object_or_404(Bill, bill_id=bill_id)
    return render(request, 'app/account_details.html', {'bill': bill})

def set_penalty(request, bill_id):
    bill = get_object_or_404(Bill, bill_id=bill_id)
    if request.method == 'POST':
        penalty_amount = request.POST.get('penalty_amount')
        if 200 <= float(penalty_amount) <= 500:
            bill.penalty_fee = penalty_amount
            bill.save()
            return redirect('overdue_bills')
        else:
            return render(request, 'app/set_penalty.html', {'bill': bill, 'error': "Penalty must be between RM 200 and RM 500"})
    return render(request, 'app/set_penalty.html', {'bill': bill})
```

```
@login_required
def schedule_monthly_bills(request):
    error_message = None

    if request.method == 'POST':
        date_str = request.POST.get('schedule_date')

        if not date_str:
            error_message = 'Invalid date.'
        else:
            try:
                schedule_date = datetime.strptime(date_str, '%Y-%m-%d').date()
                today = timezone.now().date()
                max_date = today + timedelta(days=30)

                if schedule_date <= today or schedule_date > max_date or schedule_date.year != today.year:
                    error_message = 'Invalid date.'
                else:
                    ScheduledBilling.objects.create(schedule_date=schedule_date)
                    return render(request, 'app/schedule_success.html', {'schedule_date': schedule_date})

            except ValueError:
                error_message = 'Invalid date.'

    return render(request, 'app/schedule_monthly_bills.html', {'error_message': error_message})

@login_required
def bill_generation_success(request):
    return render(request, 'app/bill_generation_success.html')
```

```python
# ========================
# Utility Provider Views
# ========================
def utility_dashboard(request):
    provider = get_current_user(request)
    if not provider or not isinstance(provider, UtilityProvider):
        return redirect('login')

    return render(request, 'utility/dashboard.html', {'provider': provider})

# ========================
# Staff Views
# ========================
def staff_dashboard(request):
    # Get the staff profile of the logged-in user
    staff_profile = Staff.objects.get(user=request.user)

    # Pass the staff profile to the template
    return render(request, 'app/staff_dashboard.html', {'staff': staff_profile})

def utility_dashboard(request):
    # Get the utility provider profile of the logged-in user
    utility_provider_profile = UtilityProvider.objects.get(user=request.user)

    # Pass the utility provider profile to the template
    return render(request, 'app/utility_dashboard.html', {'utility_provider': utility_provider_profile})

@receiver(post_save, sender=User)
def create_customer_profile(sender, instance, created, **kwargs):
    if created and instance.role == 'CUSTOMER':
        Customer.objects.create(user=instance)
```

```python
@receiver(post_save, sender=User)
def save_customer_profile(sender, instance, **kwargs):
    if instance.role == 'CUSTOMER':
        instance.customer_profile.save()


def generate_unique_ticket_id():
    while True:
        # Generate a new ticket_id (you can customize this)
        ticket_id = f"T{str(randint(10000, 99999))}"
        if not Issue.objects.filter(ticket_id=ticket_id).exists():
            return ticket_id


def create_issue(customer, title, description, status):
    ticket_id = generate_unique_ticket_id()  # Generate unique ticket_id
    Issue.objects.create(
        customer=customer,
        title=title,
        description=description,
        status=status,
        ticket_id=ticket_id
    )
    print(f"Issue created with ticket_id: {ticket_id}")

```

```python
# ========================
# Staff Views
# ========================
def is_staff(user):
    return user.is_authenticated and user.role == 'STAFF'

@login_required
@user_passes_test(is_staff)
def staff_dashboard(request):
    try:
        # Get the staff profile of the logged-in user
        staff_profile = Staff.objects.get(user=request.user)
    except Staff.DoesNotExist:
        messages.error(request, "Staff profile does not exist.")
        return redirect('home')  # Or redirect to a different page, like registration

    # Pass the customer profile to the template
    return render(request, 'app/staff_dashboard.html', {'customer': staff_profile})

@login_required
@user_passes_test(is_staff)
def manage_customer_accounts(request):
    customers = Customer.objects.all()
    return render(request, 'app/manage_customer_accounts.html', {'customers': customers})

@login_required
@user_passes_test(is_staff)
def staff_viewCustomer(request, customer_id
    customer = get_object_or_404(Customer, customer_id=customer_id)
    bills = Bill.objects.filter(customer=customer)  # Fetch bills for this customer

    return render(request, 'app/staff_viewCustomer.html', {'customer': customer, 'bills': bills})
```

```python
@login_required
@user_passes_test(is_staff)
def staff_updateCustomer(request, customer_id):
    customer = get_object_or_404(Customer, customer_id=customer_id)

    if request.method == 'POST':
        form = CustomerForm(request.POST, instance=customer)
        if form.is_valid():
            form.save()
            # Add success message
            messages.success(request, f"Details has been successfully updated!")
    else:
        form = CustomerForm(instance=customer)

    return render(request, 'app/staff_updateCustomer.html', {'form': form, 'customer': customer})


@login_required
@user_passes_test(is_staff)
def staff_viewBill(request, bill_id):
    bill = get_object_or_404(Bill, bill_id=bill_id)
    return render(request, 'app/staff_viewBill.html', {'bill': bill})

@login_required
@user_passes_test(is_staff)
def staff_setDueDate(request, bill_id):
    # Fetch the bill by its bill_id
    bill = get_object_or_404(Bill, bill_id=bill_id)

    if request.method == 'POST':
        # Get the due date from the form
        due_date = request.POST.get('due_date')

        # Update the due date of the bill
        bill.due_date = due_date
        bill.save()

        # Add success message
        messages.success(request, f"Due date for Bill {bill.bill_id} has been successfully updated!")


    # Render the template with the bill data
    return render(request, 'app/staff_setDueDate.html', {'bill': bill})
```

```python
@login_required
@user_passes_test(is_staff)
def track_overdue_bills(request):
    today = date.today()
    overdue_bills = Bill.objects.filter(due_date__lt=today, paid=False)

    if not overdue_bills.exists():  # Use .exists() instead of checking `if not overdue_bills`
        message = "No overdue bills found"
    else:
        message = None

    return render(request, 'app/track_overdue_bills.html', {'overdue_bills': overdue_bills, 'message': message})



@login_required
@user_passes_test(is_staff)
def usage_monitoring(request):
    month_year = request.GET.get('month', None)

    if month_year:
        # Extract year and month from the selected value (yyyy-mm)
        year, month = map(int, month_year.split('-'))
        # Filter usage metrics by the selected month and year
        usage_metrics = UsageMetrics.objects.filter(
            last_updated__year=year,
            last_updated__month=month
        ).first()  # Get the first matching record
    else:
        usage_metrics = UsageMetrics.objects.latest('last_updated')  # Get the most recent metrics

    return render(request, 'app/usage_monitoring.html', {'usage_metrics': usage_metrics})
```

```python
def update_usage_metrics():
    # Recalculate metrics
    total_customers = Customer.objects.count()
    total_bills = Bill.objects.count()
    total_usage = Bill.objects.aggregate(total_usage=Sum('usage'))['total_usage']  # Assuming usage is represented by 'amount'
    avg_usage = total_usage / total_bills if total_bills else 0
    today = date.today()

    # Update or create the UsageMetrics object
    metrics, created = UsageMetrics.objects.update_or_create(
        defaults={
            'total_customers': total_customers,
            'total_bills': total_bills,
            'total_usage': total_usage,
            'avg_usage': avg_usage,
        },
        last_updated=today  # Always set the last updated time
    )



@receiver(post_save, sender=User)
def create_staff_profile(sender, instance, created, **kwargs):
    if created and instance.is_staff:  # Ensure only staff users get a profile
        Staff.objects.create(user=instance)

@receiver(post_save, sender=User)
def save_staff_profile(sender, instance, **kwargs):
    if hasattr(instance, 'staff_profile'):
        instance.staff_profile.save()

```

4.2 Software Integration

To integrate all subsystems within the billing system, we need to ensure that each component works smoothly with the others, sharing data, managing workflows, and ensuring consistency across the system.

Some of the main files used for integration is as follows:

File	Description
billing.db	Database file to store all application data for the operation of the system
settings.py	Contains all settings for the Django application which includes database configurations, installed apps, middleware settings, static file managements, and templates configuration
urls.py	Defines URL patterns for the Django application to map each URL to specific view
forms.py	Contains form classes to handle user input

4.3 Database

Our system database contains multiple tables managing different aspects of the system. Below is a structured description of the key tables:

1. **Customer** – Stores customer details, including login credentials and contact information.
    - customer_id, username, password, customer_name, customer_address, customer_email, customer_phone, meter_id
2. **UtilityProvider** – Holds utility provider details.
    - utility_id, username, password, up_name, up_phone, up_email
3. **SupportAdmin** – Stores support administrator details.
    - admin_id, username, password, admin_name, admin_phone, admin_email
4. **Staff** – Manages staff member details.
    - staff_id, username, password, staff_name, staff_phone, staff_email
5. **Issue** – Records customer issues and support tickets.
    - issue_id, customer_id, title, description, status, ticket_id
6. **Payment** – Stores payment transactions.
    - payment_id, customer_id, payment_date, payment_method, amount
7. **Feedback** – Manages customer feedback and ratings.
    - feedback_id, customer_id, rating, comment, feedback_date
8. **Meter** – Holds meter readings and tariff rates.
    - meter_id, customer_id, meter_reading, tariff_rate
9. **Bill** – Stores billing details, including due amounts and penalties.
    - bill_id, customer_id, amount, due_date, paid, creation_date, penalty_fee

```
...
Feedback F00001 (Customer: Mohammed Yousef)
Feedback F00002 (Customer: Faiz_24)
Feedback F00003 (Customer: ChhC3)
>>>
```

```
...
Issue  (Customer: Mohammed Yousef)
Issue I22002 (Customer: Faiz_24)
Issue I33003 (Customer: ChhC3)
Issue I22001 (Customer: Mohammed Yousef)
>>>
```

```
...
Customer ID: C00001
Name: Mohammed Yousef
Address: Persiaran Sepang, Cyber11, 6300 Cyberjaya, Selangor
Email: yysuf@gmail.com
Phone: +60162238871
Meter ID: M001
--------------------------------------------------
Customer ID: C00002
Name: Faiz_24
Address: Persiaran Sepang, Cyberjaya, Selangor
Email: ezzieff@outlook.com
Phone: +60123456789
Meter ID: M00112
--------------------------------------------------
Customer ID: C00003
Name: ChhC3
Address: Jalan ABC, KL
Email: chcheng@gmail.com
Phone: +60123456789
Meter ID: M00113
--------------------------------------------------
Customer ID: C00004
Name: Alex89
Address: Jalan XYZ, Johor Bahru
Email: alex.johnson@gmail.com
Phone: +60123456789
Meter ID: M00114
--------------------------------------------------
Customer ID: C00005
Name: LinaM
Address: Bandar Baru, Melaka
Email: lina.mendez@yahoo.com
Phone: +60123456789
Meter ID: M00115
--------------------------------------------------
Customer ID: C00006
Name: RaviK
Address: Jalan 123, Penang
Email: ravi.kumar@hotmail.com
Phone: +60123456789
Meter ID: M00116
```

```
<Bill: Bill B39680 (Customer: Mohammed Yousef)>
Created Bill: B39680 for Customer: Mohammed Yousef with Amount: 67.4
<Bill: Bill B55850 (Customer: Faiz_24)>
Created Bill: B55850 for Customer: Faiz_24 with Amount: 60.0
<Bill: Bill B45810 (Customer: ChhC3)>
Created Bill: B45810 for Customer: ChhC3 with Amount: 132.2
<Bill: Bill B12345 (Customer: Alex89)>
Created Bill: B12345 for Customer: Alex89 with Amount: 100.0
<Bill: Bill B78901 (Customer: LinaM)>
Created Bill: B78901 for Customer: LinaM with Amount: 85.5
<Bill: Bill B23456 (Customer: RaviK)>
Created Bill: B23456 for Customer: RaviK with Amount: 45.0
>>>
```

```
Admin ID: A4456
Username: nznaf12
Password (plaintext): Naf_2569
Admin Name: Nafis Zahran
Admin Phone: +6099611561
Admin Email: naf.zahran@gmail.com
----------------------------------------


Admin ID: A4568
Username: leecc
Password (plaintext): LcLC_11
Admin Name: Lee Chong
Admin Phone: +6062636982
Admin Email: lee.chong@gmail.com
----------------------------------------


Admin ID: A6459
Username: irh_wa
Password (plaintext): irir1315
Admin Name: Irham Waseem
Admin Phone: +6057866693
Admin Email: irwaseem@outlook.com
----------------------------------------
>>> for user in users:
...     print(f"User ID: {user['id']}, Username: {user['username']}, Email: {user['email']}, Role: {user['rol
e']}")
...
User ID: 1, Username: ImY1l, Email: yysuf@gmail.com, Role: CUSTOMER
User ID: 2, Username: Faiz_24, Email: ezzieff@outlook.com, Role: CUSTOMER
User ID: 3, Username: ChhC3, Email: chcheng@gmail.com, Role: CUSTOMER
User ID: 4, Username: Alex89, Email: alex.johnson@gmail.com, Role: CUSTOMER
User ID: 5, Username: LinaM, Email: lina.mendez@yahoo.com, Role: CUSTOMER
User ID: 6, Username: RaviK, Email: ravi.kumar@hotmail.com, Role: CUSTOMER
>>>
```

5 Testing

5.1 Testing Strategy

Our testing strategy is done after creating the user profiles, we then create the modules classes that are needed in our system. Once all the modules are available, we then test the actor's functions with the related modules starting with admin, parent, healthcare professionals and insurance panels. We perform the system testing and identify the error that occurred and improve the system.

5.2 Test Data

1. System Registration

Test Scenario	Input	Expected Output	Screen
Valid Meter ID	Meter ID: M63350	Redirect to Registration page	Registration page 1
Valid Registration	Name: Muhammed Faiz, Email: ezzieff@outlook.com, Phone: +60122462882, Address: Persiaran Bestari, Cyberjaya, 6300 Cyberjaya, Selangor, Username: Faiz_24, Password: ZxZ445	Success Message: "Registration successful"	Registration page
Invalid Meter ID	Meter ID: M6785	Error: "Meter ID is incorrect"	Registration page 1
Duplicated Meter ID	Meter ID: M67850	Error: "Meter ID used by another account"	Registration page 1
Incomplete Information	Meter ID: M6335, Name: (Missing), Email: chcheng@gmail.com, Phone: +60193973839, Address: (Missing), Username: ChhC3, Password: CCc22a	Error: "Please complete all required fields or correct invalid entries"	Registration page

2. Log in to the System

Test Scenario	Input	Expected Output	Screen
Valid Login	Username: ImY1l, Password: Yym226	Redirect to Customer main page	Main page
Invalid Username	Username: ImY10, Password: Yym226	Error: "Invalid username or password"	Login page
Invalid Password	Username: ImY1l, Password: ym2266	Error: "Invalid username or password"	Login page
Empty Fields	Username: ImY10, Password: (Empty)	Error: "Please enter your username and password"	Login page

3. Reset Password

Test Scenario	Input	Expected Output	Screen
Valid Reset Password	Email: yysuf@gmail.com, New Password: ImY77	Success: "Password reset successfully"	Reset Password Page
Invalid Email	Email: amy47@gmail.com, New Password: Amy1	Error: "No account found with this email"	Reset Password Page
Invalid Password	Email: chcheng@gmail.com.com, New Password: ImY11il7877	Error: "Invalid password, try another one"	Reset Password Page

4. View Bill Details

Test Scenario	Input	Expected Output	Screen
Valid Bill Details	Select Bill ID: B39680	Display Bill Details: - Customer Information: Name: Mohammed Aamena Email: ammena@gmail.com Phone: +60 11-62237057 Address: 123 Multimedia st, Cyberjaya - Billing Information: Bill ID: B9561 Outstanding Balance: RM 100.00 Consumption: 350 kWh, Due Date: 2025-01-15 - Payment History	Bills Details Page
No Bill Data	Customer ID: C99990 (No bills available)	Message: "No billing information available"	View Bills Page
Invalid Bill ID	Bill ID: R4455	Error: "Invalid Bill ID"	View Bills Page

5. Make a payment and Receive Payment Receipt

Test Scenario	Input	Expected Output	Screen
Successful Payment	Method: TnG Amount: RM 50.00	Successful payment, Thank you!	Payment Page
Invalid Amount	Method: TnG Amount: RM 200.00 (The total amount of the bill is RM 67.00)	Invalid amount, try again	Payment Page
No Method Chosen	Method: - Amount: RM 50.00	An error has occurred: no method has been selected	Payment Page

6. Submit and Analysis Customer Feedback

6.1 Customer

Test Scenario	Input	Expected Output	Screen
Incomplete Feedback Form	Customer ID: C33650, Rating: 5, Feedback: "Great service!"	Success Message: "Your feedback has been submitted"	Feedback Form Page
Incomplete Feedback Form	Customer ID: C24890, Rating: (Missing), Feedback: "Great service!	Error: "Please complete all required fields"	Feedback Form Page

6.2 Support Admin

Test Scenario	Input	Expected Output	Screen
View All Feedback	Click on “View Customer Feedback” without filter applied	Display All Feedback	Customer Feedback Page
Filter Feedback	Filter by Rating: 5	Display Filtered Feedback	Customer Feedback Page
No Feedback Found	Filter by Rating: 1	Message: "No feedback found for the selected filters"	Customer Feedback Page

7. Submit and Resolve Customer Issues

7.1 Customer

Test Scenario	Input	Expected Output	Screen
Valid issue submission	Customer ID: C33650, Title: "Payment Issue", Description: "I can’t make any payment; every time I try to pay it says I’m blocked from paying", Attachments: No file attached	Ticket ID: T02580, Status: "In Progress"	Support Page
Incomplete Issue Form	Customer ID: C24890, Title: (Missing), Description: "When I try to see my bills the app shuts down", Attachments: issue.pdf	Error: "Please complete all required fields"	Support Form Page
Invalid Attachment Format	Customer ID: C33650, Title: "Payment Issue", Description: "I can’t make any payment; every time I try to pay it says I’m blocked from paying", Attachments: issue.exe	Error: "Invalid file format. Please upload images or documents only."	Support Form Page
Large Attachment Size	Customer ID: C33650, Title: "Payment Issue", Description: "I can’t make any payment; every time I try to pay it says I’m blocked from paying",	Error: "File size exceeds the limit. Please upload files smaller than 5MB."	Support Form Page

	Attachments: largefile.zip (10MB+)		

7.2 Support Admin

Test Scenario	Input	Expected Output	Screen
View Issue Details	Ticket ID: T02580	Display Ticket Details: Customer ID: C3365, Issue Description: I can't make any payment; every time I try to pay it says I'm blocked from paying, Current Status: In progress	Issue Detail Page
Update Issue Status	Ticket ID: T87560, Status: "Pending", Resolution Comment: "Investigating payment issue"	Success: "Ticket status updated successfully"	Update status Page

8. Track Overdue Bills

Test Scenario	Input	Expected Output	Screen
View details	-	Display account details	View Details Page
Empty due date field	-	Error: "Please enter a valid date."	Set Due Date Page
Invalid due date field	5th May 1997	Error: "Please enter a valid date."	Set Due Date Page
Valid due date field	7th March 2025	Success: "Due date set successfully!"	Set Due Date Page

9. Update Meter Reading and Tariffs

Test Scenario	Input	Expected Output	Screen
View current rates	-	Display current rates	Meter Readings and Tariff page
Empty rate field	-	Error: "Please choose a valid rate value"	Update Tariff Rate page
Invalid rate value	7000	Error: "Please choose a valid rate value"	Update Tariff Rate page
Valid rate value	34.80	Success: "New rate values set successfully!"	Update Tariff Rate page

10. Generate Monthly Bills

Test Scenario	Input	Expected Output	Screen
Invalid date field	5th May 1997	Error: "Please choose a valid date."	Schedule Monthly Bill page
Empty date field	-	Error: "Please choose a valid date."	Schedule Monthly Bill page

Valid date field	7th March 2025	Success: "Monthly billing date set successfully!"	Schedule Monthly Bill page

11. Manage Customer Accounts

Test Scenario	Input	Expected Output	Screen
Invalid search account	Search accounts: C03300	Error: "No accounts found."	Manage Customer Account Page
Valid search account	Search accounts: C04030	Display account	Manage Customer Account Page
View details	-	Display account details	View Details Page
Empty update details fields	Meter ID: M73550 Name: Faiz Ilyasa Email: Phone number: 23 Address: 123 Multimedia St, Cyberjaya	Error: "Update failed. Please try again."	Update Details page
Valid update details	Meter ID: M73550 Name: Faiz Ilyasa Email: Phone number: 0122462882 Address: 123 Multimedia St, Cyberjaya	Success: "Details updated successfully!"	Update Details page

12. Usage Monitoring

Test Scenario	Input	Expected Output	Screen
Empty month and year field	-	Error: "Please select valid month and year."	Usage Monitoring Page
Invalid month and year field	March 2030	Error: "No usage metrics found."	Usage Monitoring Page
Valid month and year field	December 2024	Display usage metrics	Usage Monitoring Page

5.3 Acceptance Testing

5.3.1 Customer

Criteria	Fulfilled?	Remarks
Log in to the System	Yes	
View Dashboard	Yes	

Date Checking	Yes	

Date tested : 28/1/2025
% Complete : 100%
Tested by : Mohammed Yousef Mohammed Abdulkarem
Verified by : Dr. Palanichamy Naveen

5.3.2 Utility Provider

Criteria	Fulfilled	Remarks
Set Penalty Amount	Yes	
Schedule Monthly Bills	Yes	

Date tested : 28/1/2025
% Complete : 100%
Tested by : Chua Cheng Zong
Verified by : Dr. Palanichamy Naveen

5.3.3 Support Admin

Criteria	Fulfilled?	Remarks
Update Status	Yes	
Analysis Feedbacks	Yes	
Manage Bills	Yes	

Date tested : 4/2/2025
% Complete : 100%
Tested by : Mohammed Aamena Mohammed Abdulkarem
Verified by : Dr. Palanichamy Naveen

5.3.4 Staff

Criteria	Fulfilled	Remarks
Set Due Date	Yes	
View Bill Details	Yes	

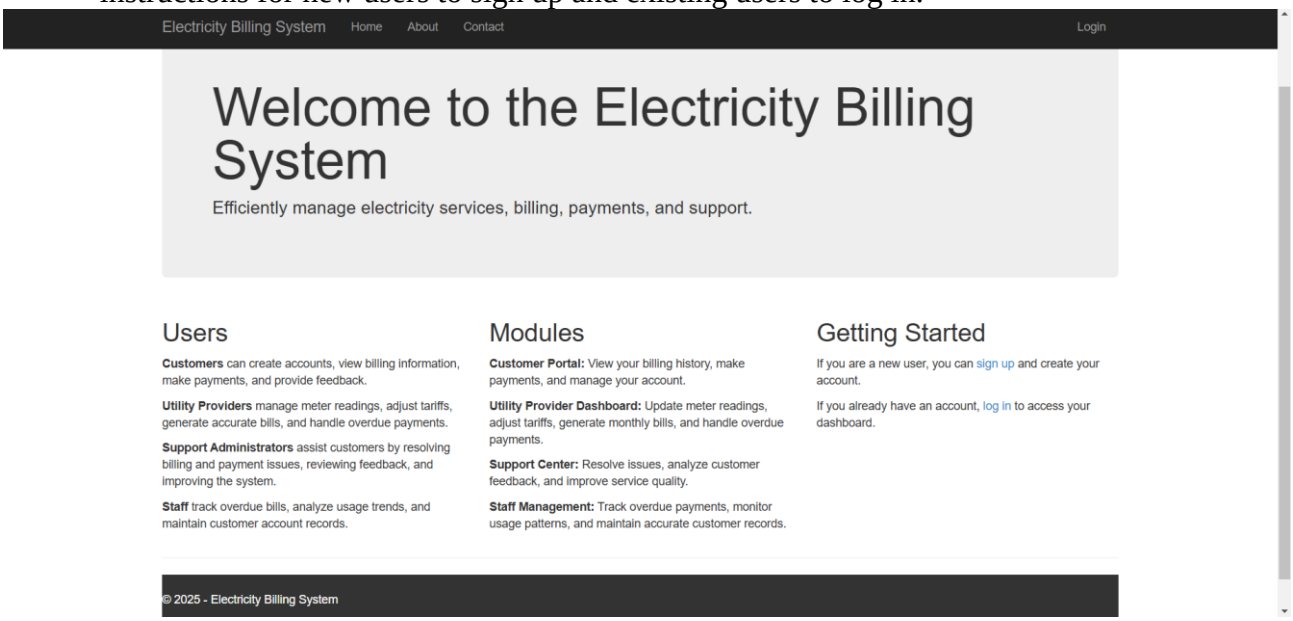
Date tested : 4/2/2025
% Complete : 100%
Tested by : Muhammad Faiz bin Ilyasa
Verified by : Dr. Palanichamy Naveen

6 Sample Screens

6.1 Main Screen

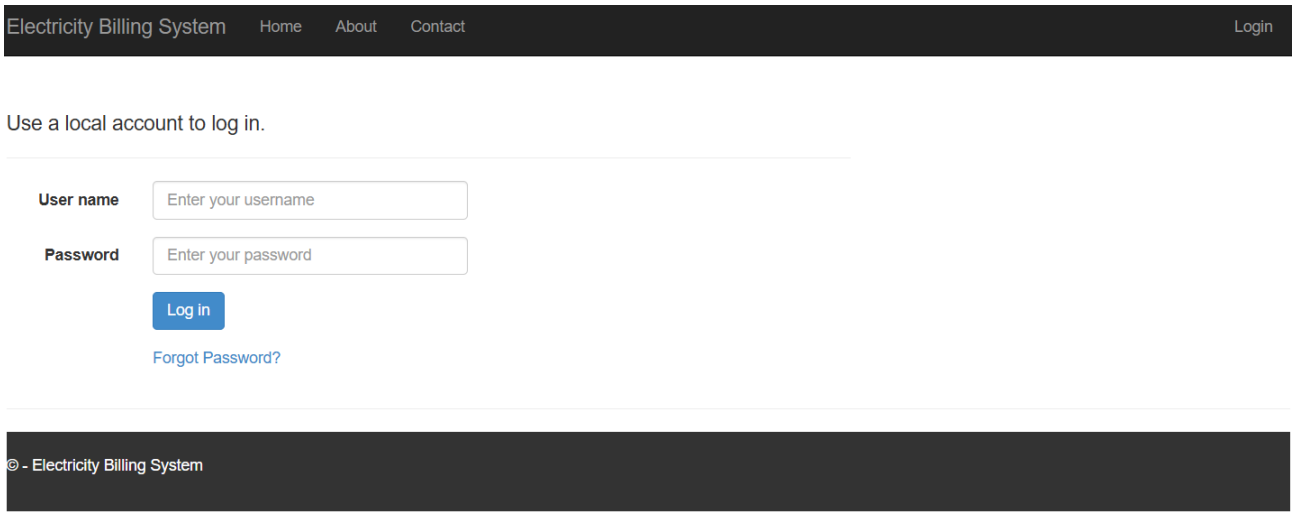
6.1.1 Home Screen

The home screen of the Electricity Billing System serves as an introductory interface for users before logging in, providing an overview of the system's functionalities. It includes a navigation bar with links to the Home, About, Contact, and Login pages. A welcome message highlights the system's purpose of managing electricity services, billing, payments, and support. The page is structured into three sections: Users, which outlines different roles and their responsibilities; Modules, which describes key system features such as the Customer Portal and Utility Provider Dashboard; and Getting Started, which provides instructions for new users to sign up and existing users to log in.



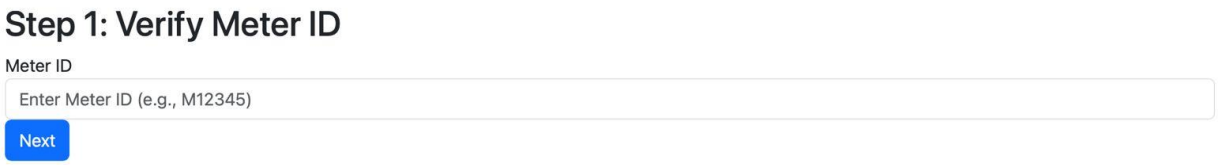
6.1.2 Log in Screen

The Login Screen of the Electricity Billing System functions as the primary access point for all user categories, including customers, support administrators, staff, and utility providers. It facilitates secure and authorized access while maintaining an intuitive user interface. Upon successful authentication, users are directed to their respective dashboards, customized according to their roles and permissions.



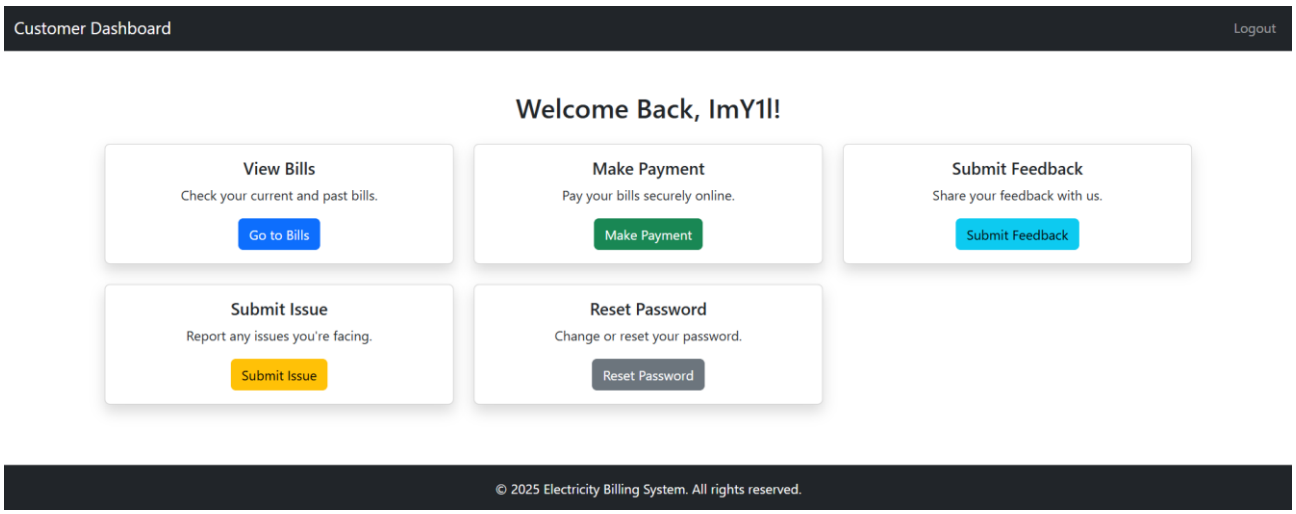
6.1.3 Subsystem 1 Screens: Customer

- **Registration Screen**
  The "Enter Meter ID" screen is displayed first, it's a critical role in linking a user's account to their designated electricity meter. Users are required to input their unique Meter ID before proceeding by selecting "Next." The system validates the provided Meter ID for accuracy before directing the user to the Electricity Billing System Registration Form.



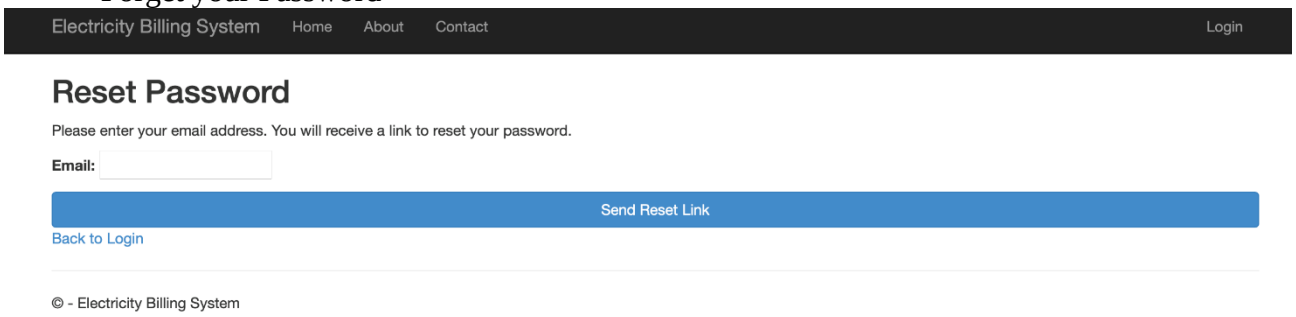
  This registration form collects essential user details as part of the onboarding process. Upon completing the required fields, users finalize account creation by selecting the "Submit" button. This registration procedure ensures security and proper access management, allowing only authorized individuals to handle electricity bill processing and related account activities efficiently.

- **Main Customer Screen**
  For customers, this is the initial screen encountered upon logging in. It incorporates multiple buttons, each granting access to specific services, thereby enhancing user experience. Additionally, the system dynamically updates and displays the logged-in customer's username, such as "ImY1l" contributing to a more personalized interface.

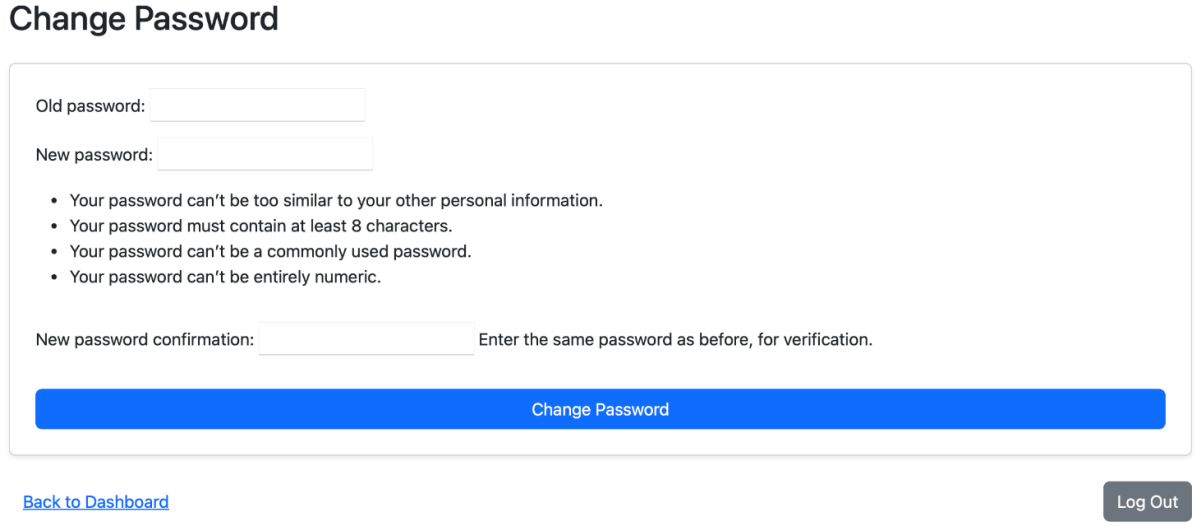


- **Reset Password Screen**
  The system includes a "Reset Your Password" feature, providing a secure and efficient mechanism for users to recover access to their accounts in case of forgotten credentials. This functionality enhances user convenience by enabling seamless account recovery while upholding stringent security protocols to safeguard sensitive information.

- Forget your Password



- Desire to change the password



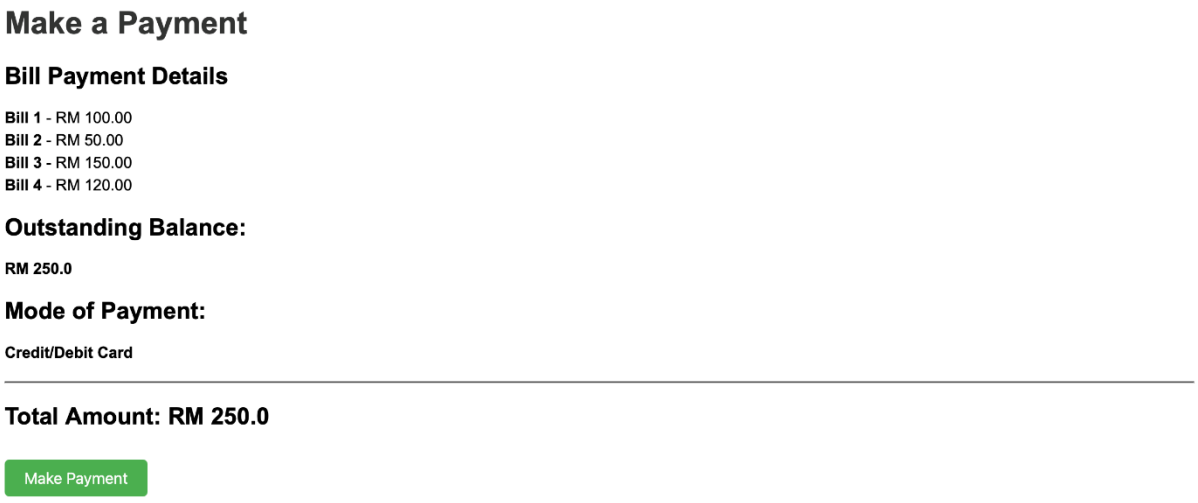
- **View Bills Screen**

  A dedicated screen presents users with a comprehensive list of their active electricity bills, offering detailed insights into each selected bill.

- **Make a Payment**

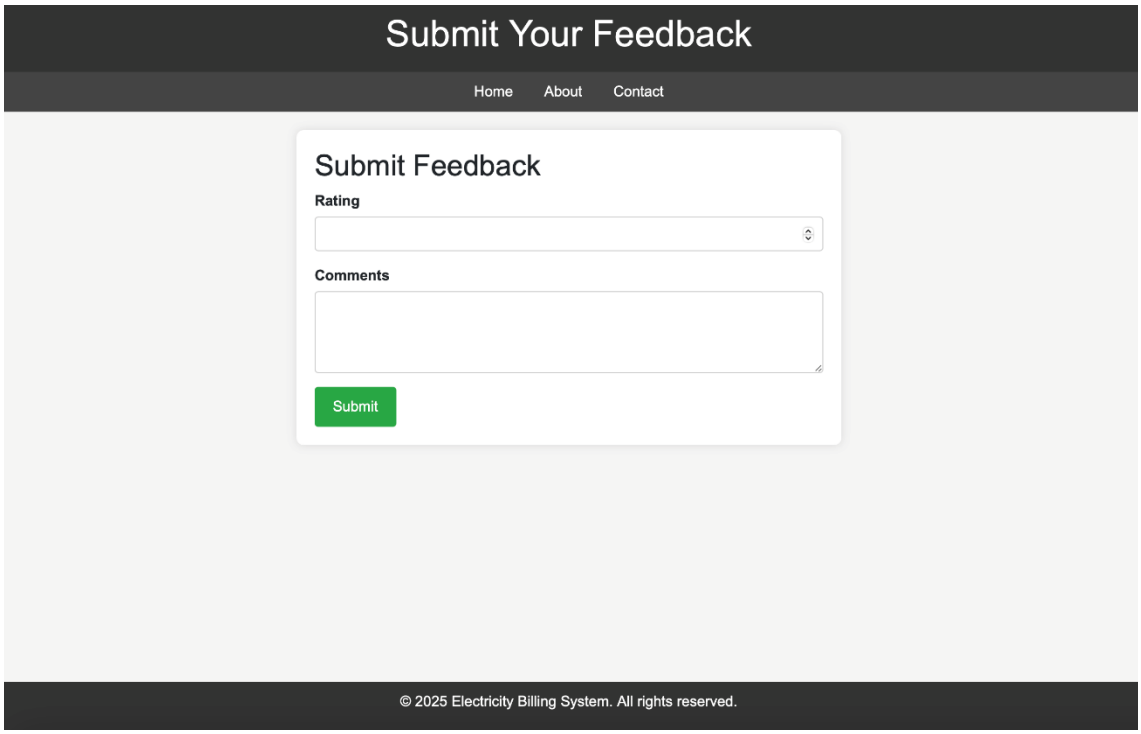
  Another interface facilitates bill selection and payment processing, ensuring a streamlined transaction experience.

-

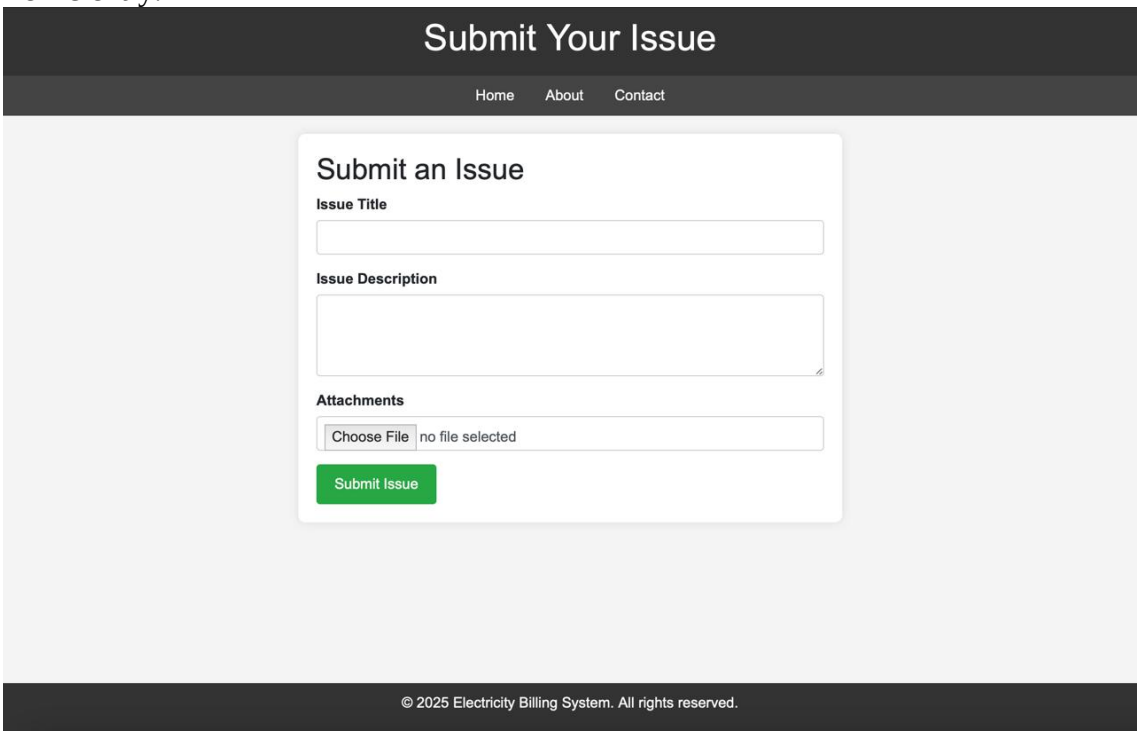


- **Submit Feedback**

  This interface allows customers to submit feedback and comment on issues related to the system efficiently.

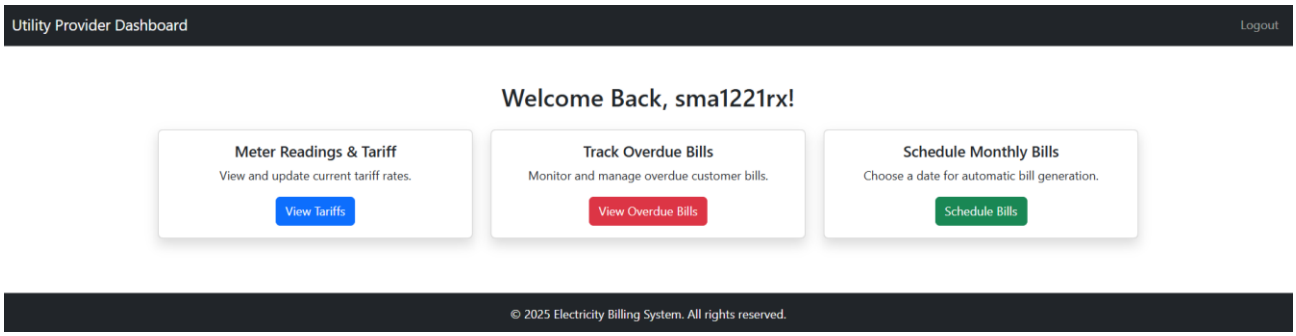


- **Report an Issue**
  Furthermore, the interface allows customers to report issues related to their electricity service efficiently.



6.1.4 Subsystem 2 Screens: Utility Provider

Main Screen



The initial screen of the Utility Provider interface provides an organized and user-friendly environment for the utility provider to efficiently manage finance-related tasks within the Electricity Billing System. Below are the key elements visible on this screen:

- **Welcome Message**:

The Utility Provider is greeted with a welcoming message upon successful login.

- **Meter Readings and Tariffs:**

This option allows Utility Provider to check the current rates for the meter readings and tariffs, and to update the data values. This helps the staff be able to calculate the bill to accurately charge the customers based on the rates.

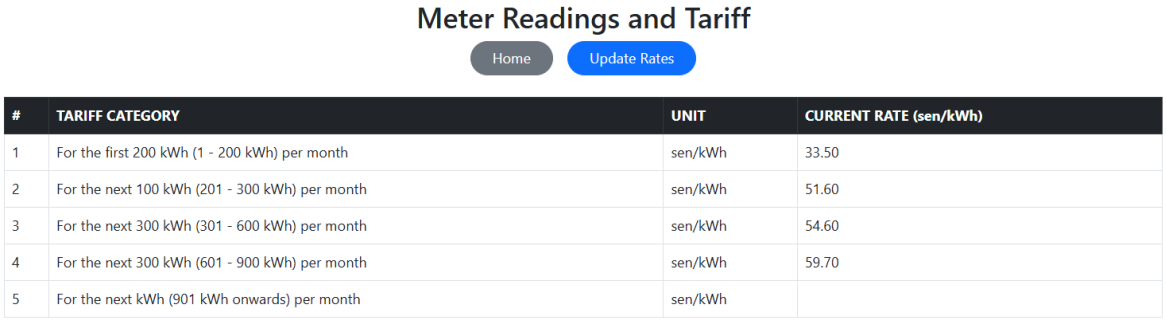
- **Track Overdue Bills:**

This option helps the Utility Provider check which customer has outstanding fees. The Utility Provider can choose to give them one more week as the final deadline or disconnect the electricity. This helps the Utility Provider to deal with customers with overdue fees efficiently.

- **Schedule Monthly Bill:**

This option lets the Utility Provider schedule a date on every month to generate the bill for each customer. This helps the Utility Provider to automate the billing process, with great accuracy and efficiency.

Meter Readings and Tariff screen



#	TARIFF CATEGORY	UNIT	CURRENT RATE (sen/kWh)
1	For the first 200 kWh (1 - 200 kWh) per month	sen/kWh	33.50
2	For the next 100 kWh (201 - 300 kWh) per month	sen/kWh	51.60
3	For the next 300 kWh (301 - 600 kWh) per month	sen/kWh	54.60
4	For the next 300 kWh (601 - 900 kWh) per month	sen/kWh	59.70
5	For the next kWh (901 kWh onwards) per month	sen/kWh	

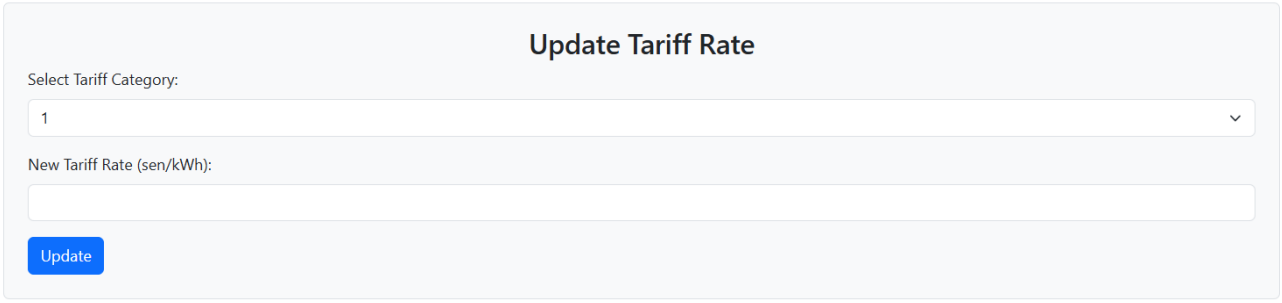
The Meter Readings and Tariff page is accessible by clicking the “View Tariff” button, it lets the utility provider check the current tariff values to make sure the data is accurate.
The column displays data such as:
Tariff category: The category of tariff.

Unit: The unit of tariff.
Current Rate: The current rate of tariff which the customer should be charged.

Navigation Bar:
The navigation bar provides quick access to various sections within the system:

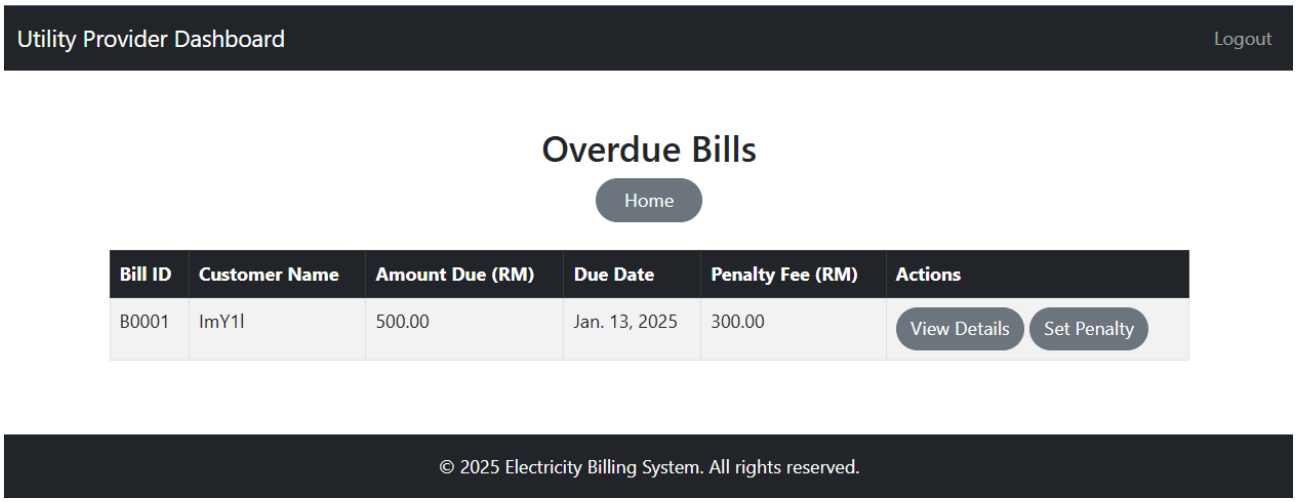
- **Home**: Redirects the Utility provider to the main screen.
- **Update Rates**: Brings the Utility provider to the Update Rates screen.

Update Rates screen



The "Update Rates" screen prompts the utility provider to key-in the new tariff rates, there is a drop-down menu to let them choose which tariff category to update.

Track Overdue Bills Screen



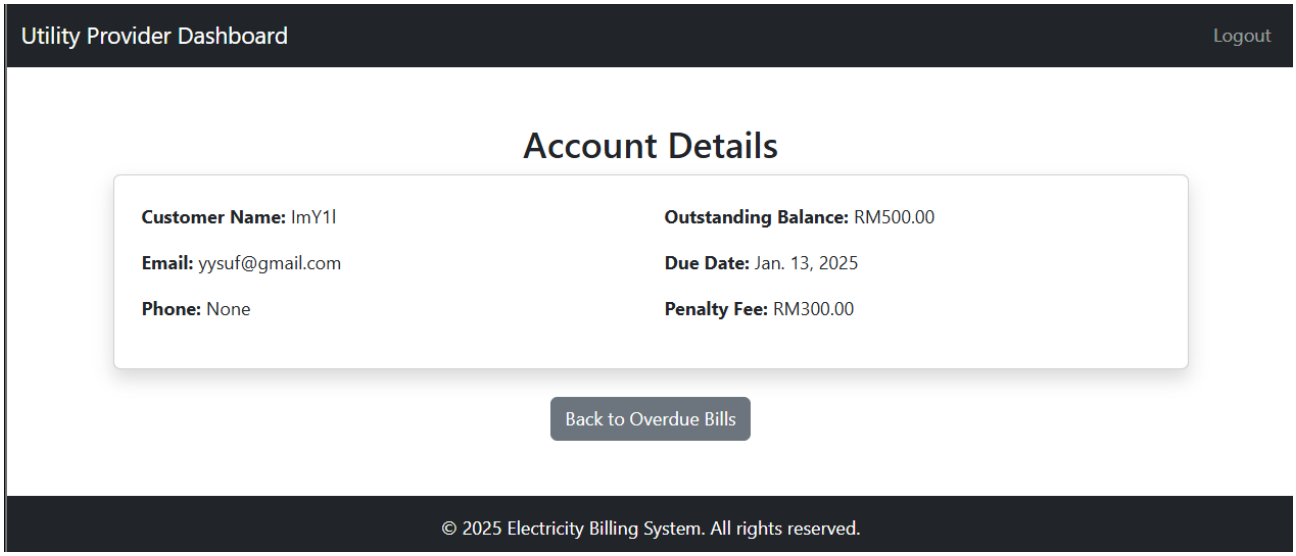
The "Track Overdue Bills" screen is designed to help utility providers monitor, verify, and take necessary actions to manage overdue bills in the system. It ensures that overdue payments are identified and handled effectively.

Home Button:
The "Home" button allows the utility provider to go back to the main screen.
Overdue Bills List:
The table displays a list of overdue bills with columns such as:

- **Bill ID:** Unique identifier for each bill.

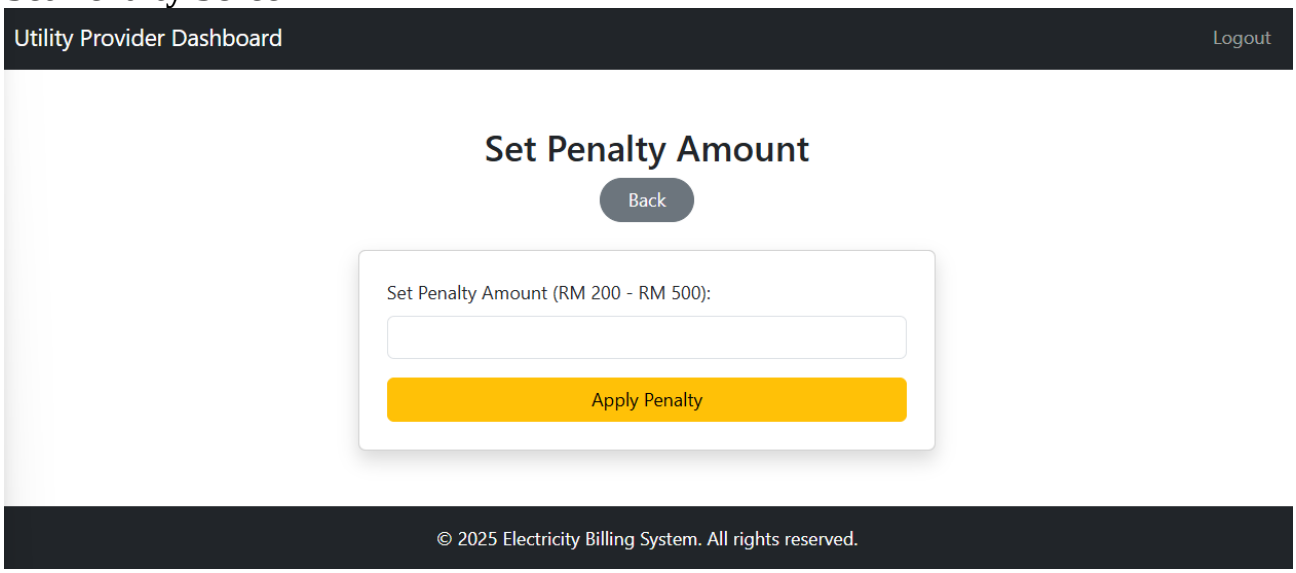
- **Customer Name:** The name of the account holder.
- **Amount:** The total amount due for the bill.
- **Date:** The date when the bill is generated.
- **Overdue by:** Indicates how long the bill has been overdue by.
- **Actions:** A "View Details" button to view a more detailed information of the bill details, and a "Set Penalty Amount" button for the utility provider to set a penalty for the customer and disconnect their electricity.

View Details Screen



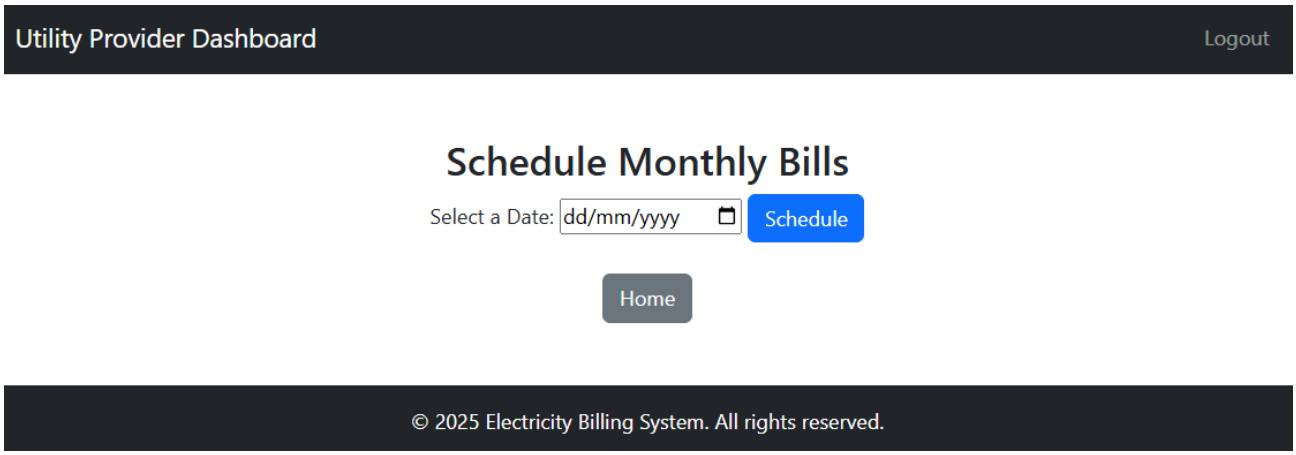
The "View Details" screen provides a detailed description of the customer information, their billing information and payment histories. Utility providers can review and verify the necessary information.

Set Penalty Screen



The "Set Penalty" screen prompts the utility provider to key-in the penalty amount of the customer. This will disconnect the electricity for the customer until they pay the penalty.

Schedule Monthly Bill screen

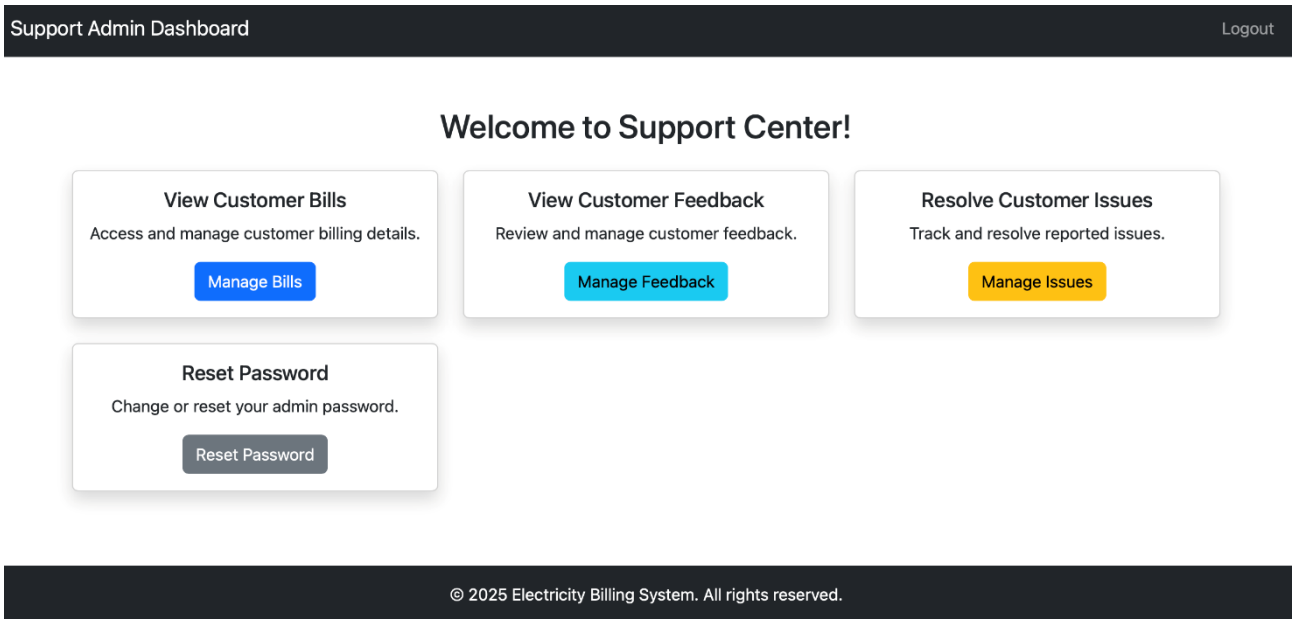


The “Schedule Monthly Bill” screen lets the utility provider to set a scheduled date to generate the monthly bill automatically, there is a drop-down menu with a calendar to assist them with scheduling.

6.1.5 Subsystem 3 Screens: Support Admin

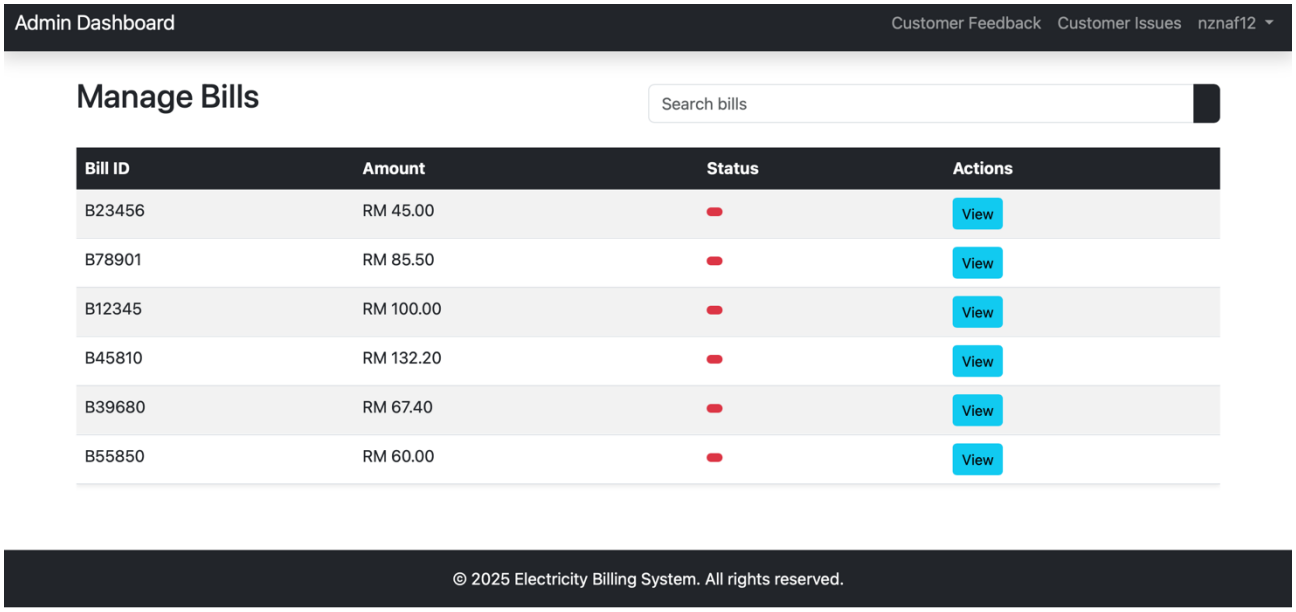
- **Main Support Admin Screen**

  The initial screen of the Support Admin interface in the Electricity Billing System provides an organized and user-friendly environment for Support Administrators to efficiently manage customer-related tasks. The **Welcome Message ("Welcome to Support Center!")** establishes a professional and approachable tone, marking the beginning of the Support Administrator's session.



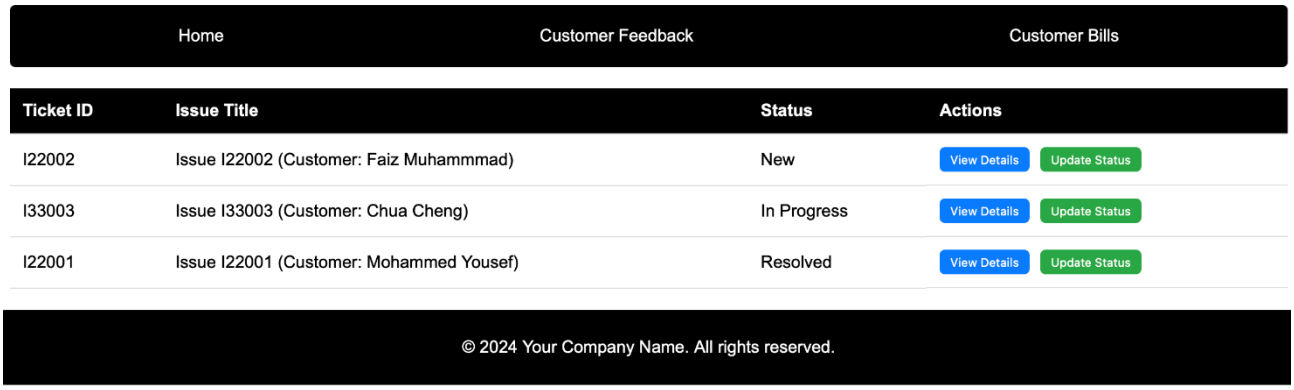
- **View Customer Bill Screen**

  The **View Customer Bills** feature grants Support Administrators immediate access to customer billing records. This functionality is essential for addressing billing discrepancies, responding to inquiries, and providing clarifications regarding bill charges. It allows Support Administrators to efficiently search, view, and manage customer bills. The interface displays a list of bills and includes filtering and search functionalities. Selecting a specific bill navigates the administrator to the **Bill Details Screen**, where they can access comprehensive information, including customer details, payment history, and billing records.



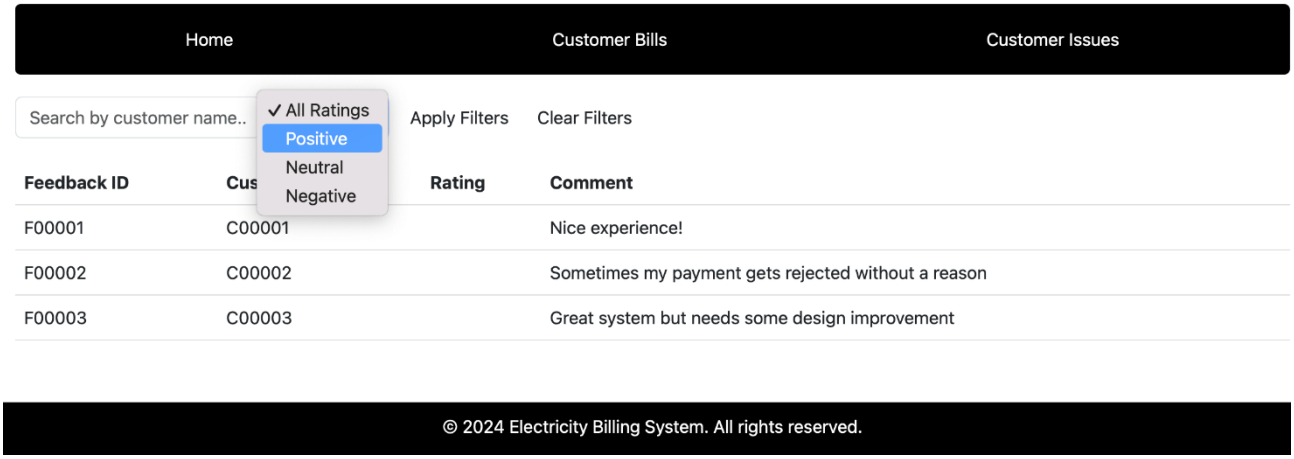
- **View Customer Issues Screen**

  The Customer Issues Management Screen serves as a centralized platform for Support Administrators to monitor all reported issues. This interface enables administrators to track ongoing cases and manage support tickets, facilitating timely resolution. The Resolve Customer Issues button directs Support Administrators to a section dedicated to issue resolution. This area includes features for tracking ongoing cases, communicating with customers, managing resolution workflows, and accessing a knowledge base of common solutions.



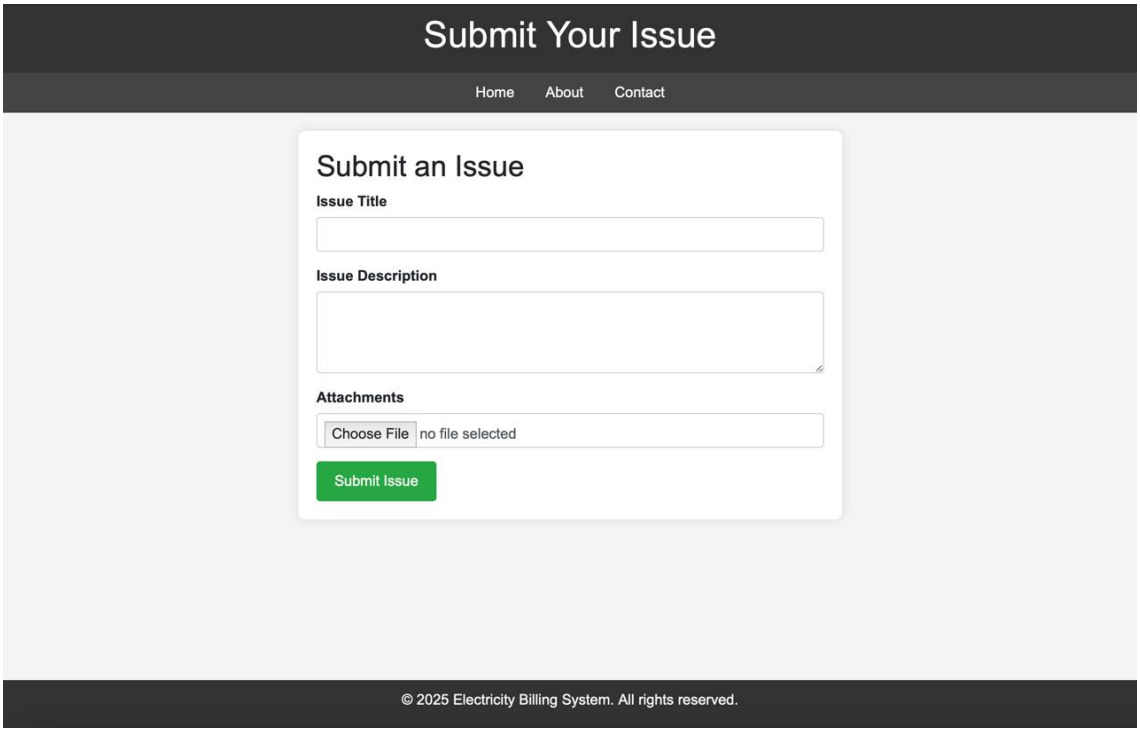
- **View Customer Feedback Screen**

  The View Customer Feedback option enables Support Administrators to review customer feedback, including complaints, suggestions, and commendations. Access to this information is crucial for assessing customer satisfaction, identifying recurring concerns, and prioritizing service improvements. The Customer Feedback Management Screen provides a comprehensive view of all customer feedback submissions. This feature supports systematic analysis and enhances service quality by enabling efficient management of customer sentiment.



- **Report an Issue**

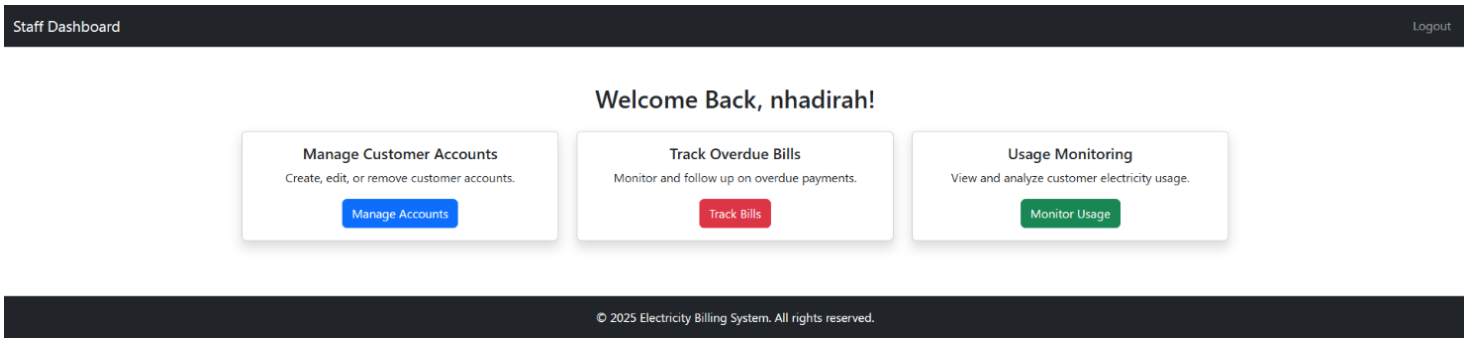
  Furthermore, the interface allows customers to report issues related to their electricity service efficiently.


Submit Your Issue
Home
About
Contact
Submit an Issue
Issue Title
Issue Description
Attachments
Choose File
no file selected
Submit Issue
© 2025 Electricity Billing System. All rights reserved.

6.1.6 Subsystem 4 Screens: Staff

- **Main Staff Screen**
  The main screen of the Staff interface allows the Staff to navigate between three tasks in the Electrical Billing System. It provides an intuitive and welcoming interface designed to simplify navigation.

  Welcome Message:
  The staff is greeted with a welcoming message upon successful login as a staff.
  Manage Customer Accounts:
  The "Manage Customer Accounts" button takes staff to the customer account management page where they can view and update customer information.
  Track Overdue Bills:
  The "Track Overdue Bills" button navigates staff to overdue bills page to review and manage overdue bills.
  Usage Monitoring:
  The "Usage Monitoring" button provides access to monitor customer electricity usage trends.



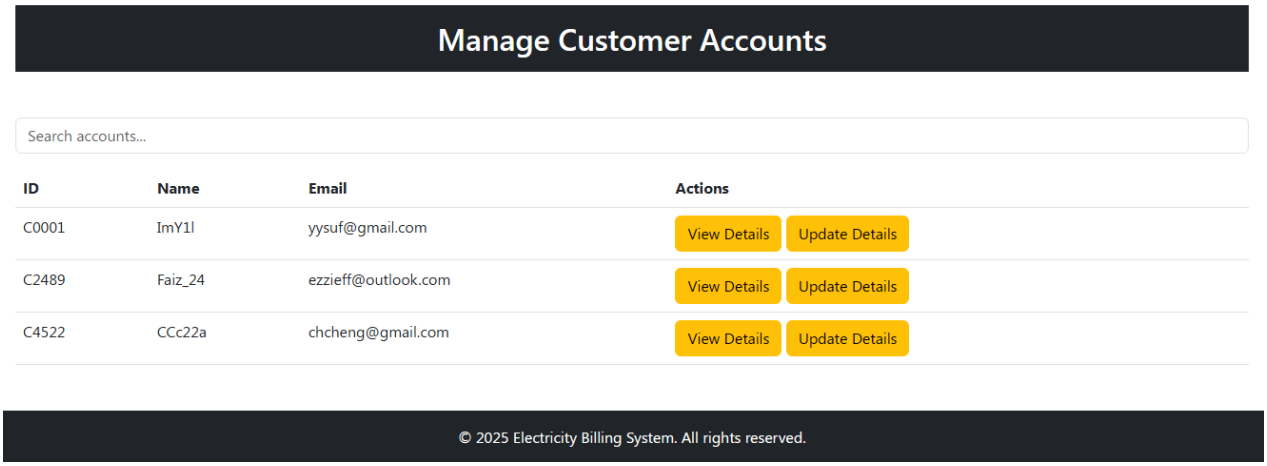
- **Manage Customer Accounts Screen**
  The "Manage Customer Accounts" screen allows staff to manage and maintain a database of customer accounts with detailed information. This interface ensures that all customer records are easily accessible and modifiable.

  Search Bar:
  The staff can search for a specific customer account using their account ID or name. The screen will be filtered based on the search.

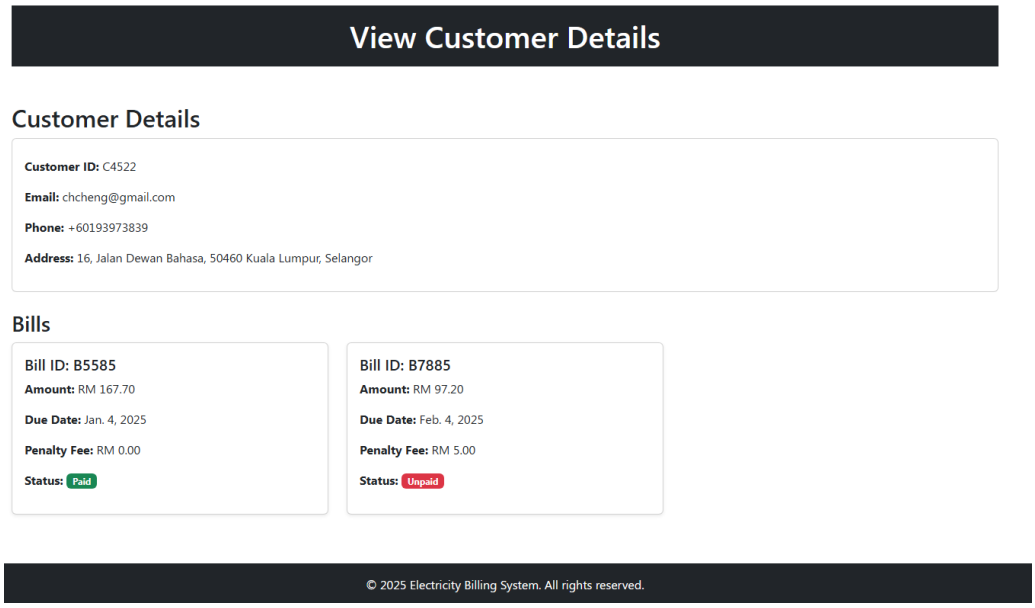
  Customer Accounts List:
  The table displays a list of customer accounts with columns such as:

  - **Account ID:** Unique identifier for each customer account.
  - **Customer Name:** The name of the account holder.
  - **Email:** The customer's email address.
  - **Phone:** The customer's phone number.
  - **Status:** Indicates the status of the customer account.

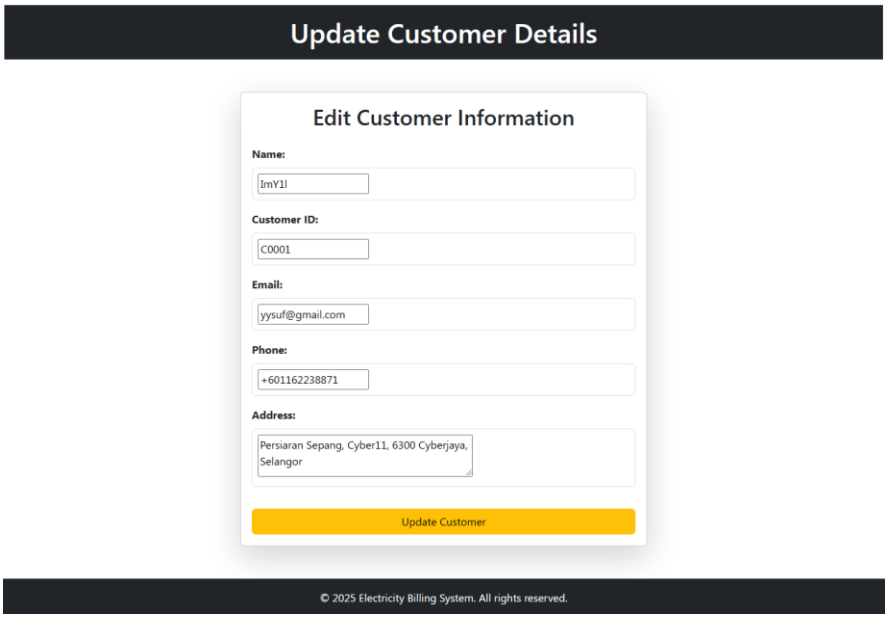
Actions: A "View Details" button to view a more detailed information of the customer account, and an "Update Details" button for the staff to update the customer account details.



- **View Customer Details Screen**
  The "View Customer Details" screen is accessible from the "Manage Customer Accounts" section by the staff. It provides a detailed description of the customer information and their billing information. Staff can review and verify the necessary information.



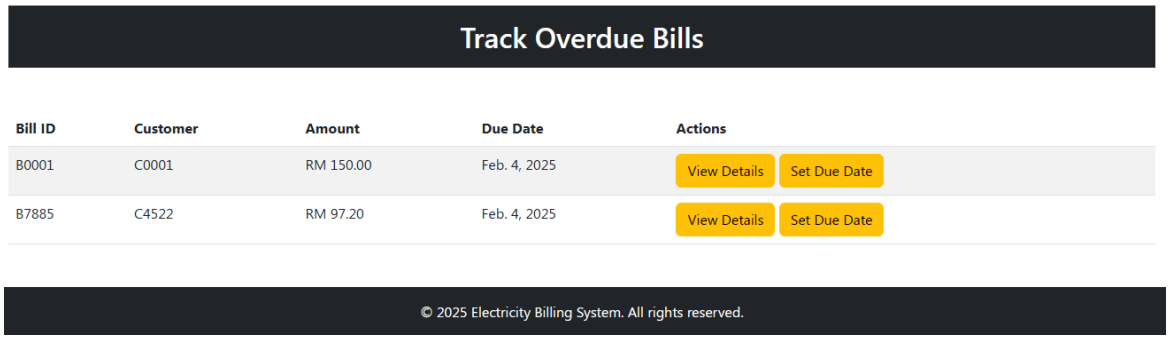
- **Update Details Screen**
  The staff can update the information of the customer account by clicking the "Update Customer" button. The staff can then input the new details such as meter ID, name, email, phone number, and address of the customer.



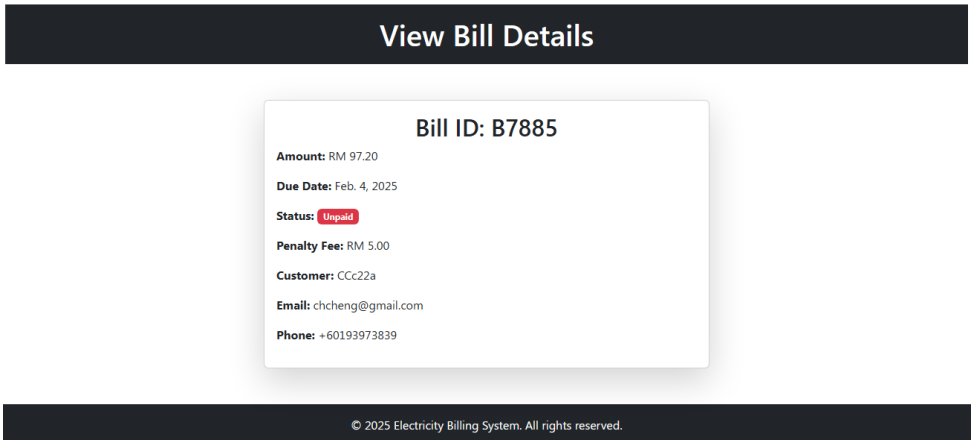
- **Track Overdue Bills Screen**
  The "Track Overdue Bills" screen is designed to help staff monitor, verify, and take necessary actions to manage overdue bills in the system. It ensures that overdue payments are identified and handled effectively.

  Overdue Bills List:
  The table displays a list of overdue bills with columns such as:

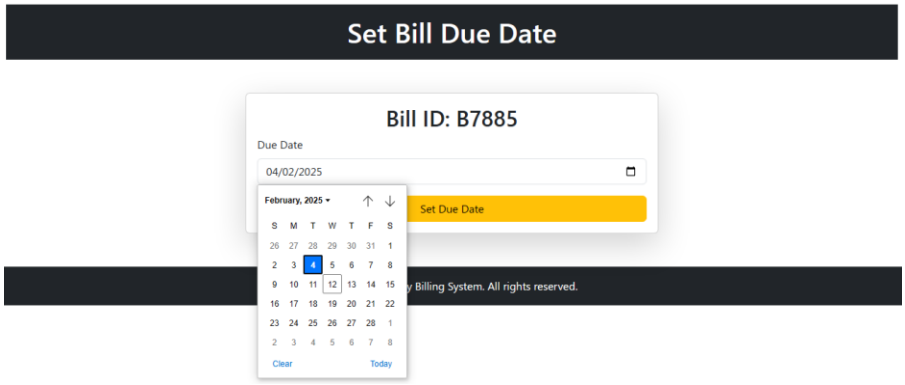
  - **Bill ID:** Unique identifier for each bill.
  - **Customer Name:** The name of the account holder.
  - **Amount:** The total amount due for the bill.
  - **Date:** The date when the bill is generated.
  - **Overdue by:** Indicates how long the bill has been overdue by.
  - **Actions:** A "View Details" button to view a more detailed information of the bill details, and a "Set Due Date" button for the staff to set a new due date for the customer to pay their overdue bill.



- **View Bill Details Screen**
  The "View Bill Details" screen is accessible from the "Track Overdue Bills" section by the staff. It provides a detailed description of the bill details. Staff can review and verify the necessary information.



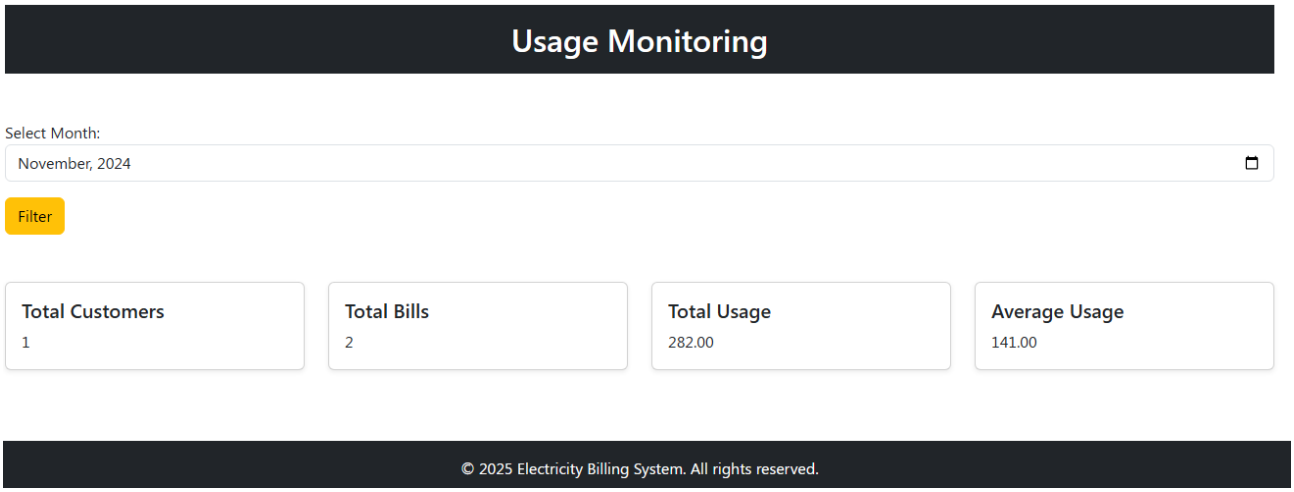
- **Set Due Date Screen**
  The "Set Due Date" screen prompts the staff to key-in a new due date from a drop-down menu. This will set the final due date for the customer to pay their overdue bill.



- **Usage Monitoring Screen**
  The "Usage Monitoring" screen offers the staff to track and analyze electricity usage trends for the overall system. This helps staff identify irregularities or patterns in usage.

  Month and Year Filter:
  This allows the staff to filter the analytics based on a specific month and year.

  Usage Metrics:
  The usage metrics section shows a breakdown of important analytics for the staff to view such as total customers, total bills generated, total electricity usage, and average electricity usage.


Usage Monitoring
Select Month:
November, 2024
Filter
Total Customers
1
Total Bills
2
Total Usage
282.00
Average Usage
141.00
© 2025 Electricity Billing System. All rights reserved.

7 Conclusion

6.1 Completion of Software

We have completed our software. We successfully implemented features specified for every actor such as **Manage User Profiles, Handle Customer Billing, and Assist User Feedback**. Additionally, the system supports functionalities for **Viewing and Managing Electricity Bills** and **Resolving Customer Issues**. However, certain components remain incomplete due to time constraints and technical challenges.

For **customers**, we implemented essential features including **Report Issues, Submit Feedback, Make Payments, and View Bill Details**. The system also includes **dynamic user authentication and account management**, ensuring a personalized and secure experience.

For **Staff subsystem**, we implemented essential features including **View Customer Details, Update Details, View Bill,** and **Set Due Date**.

6.2 Software Quality Assurance

To ensure software quality, we prioritized understanding system requirements and potential challenges. We addressed architectural design flaws early in development to enhance system reliability. A comprehensive project plan was devised, outlining quality assurance strategies and change management processes. Additionally, we documented software deviations and implemented corrective measures in a structured manner. Collaborative problem-solving efforts were undertaken to resolve issues efficiently, ensuring that the final software meets high-quality standards.

6.3 Group Collaboration

Collaboration played a crucial role in the development of the project. Team members maintained constant communication through group chats and meetings, allowing us to review progress, troubleshoot issues, and exchange research findings. This collaborative approach proved essential in debugging, refining system functionalities, and improving overall project quality.

6.4 Problem Encountered

Several challenges arose during the implementation of the **Electricity Billing System**, impacting progress and system completeness. The most significant difficulties included:

- Time constraints, particularly during coding and debugging phases.
- Complexity of integrating multiple functionalities, such as secure payment processing and real-time bill updates.
- Technical challenges related to Flask, due to limited prior experience with the framework and the need to reference external documentation extensively.
- Syntax errors and debugging difficulties, which led to delays and frequent trial-and-error approaches.
- Limited exposure to Python, Flask, and Django, which increased the learning curve and slowed implementation.

8 User Guide

1. Unzip the project files.
2. Run Windows Command Prompt and go to the folder “myproject”.
3. Run the virtual environment with the command “env\Scripts\activate”.
4. Run Visual Studio Code with the command “code .”.
5. Edit the file “env\pyvenv.cfg” and update the paths for the programs accordingly (depends on your installation).
6. In Visual Studio Code, select from the menu ‘View -> Command Palette’ and select the Python interpreter with “(‘env’)” .
7. Select from the menu ‘Terminal -> New Terminal’.
8. If there is an error message containing “… is not digitally signed …”, enter the command “SetExecutionPolicy -ExecutionPolicy RemoteSigned -Scope Process” at the prompt and then enter “env\Scripts\activate” to start the virtual environment.
9. In the terminal, enter the command “python manage.py runserver” to run the application.
10. Ctrl-click “http://127.0.0.1:8000/” and view the application in the browser.
11. Login with the user name “Faiz_24” and password “ZxZ445” to see the customer menu.
12. Login with the user name “nznaf12” and password “Naf_2569” to see the support admin menu.
13. Login with the user name “sma1221rx” and password “ newpassword123 ” to see the utility provider menu.
14. Login with the user name “nhadirah” and password “Skskrara73” to see the staff menu.