System Documentation Individual Report

for

Electricity Billing System

Version 3.0

Tutorial Section: **T10L**

Group No.: **2**

Mohammed Yousef Mohammed Abdulkarem 1221305727

Date: **12th February 2025**

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Mohammed Yousef Mohammed Abdulkarem	Generated the Project Group & Plan, System Overview, Scenario-Based Models, Class Models, and Behavioural & Flow Models.	8/12/2024
2.0	Mohammed Yousef Mohammed Abdulkarem	Created Data Design Diagrams, Architecture Design Diagrams, Interface Design Diagrams, Component Design Diagrams and Deployment Design Diagrams.	12/1/2025
3.0	Mohammed Yousef Mohammed Abdulkarem	Implement the specification to an actual web app.	12/2/2025

1 System Overview

1.1 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.

1.2 Use Cases

Actor	Use Cases
Customer	System Registration Log in to the System Reset Password Make a Payment and Receive Payment Receipt View Bill Details

1.3 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

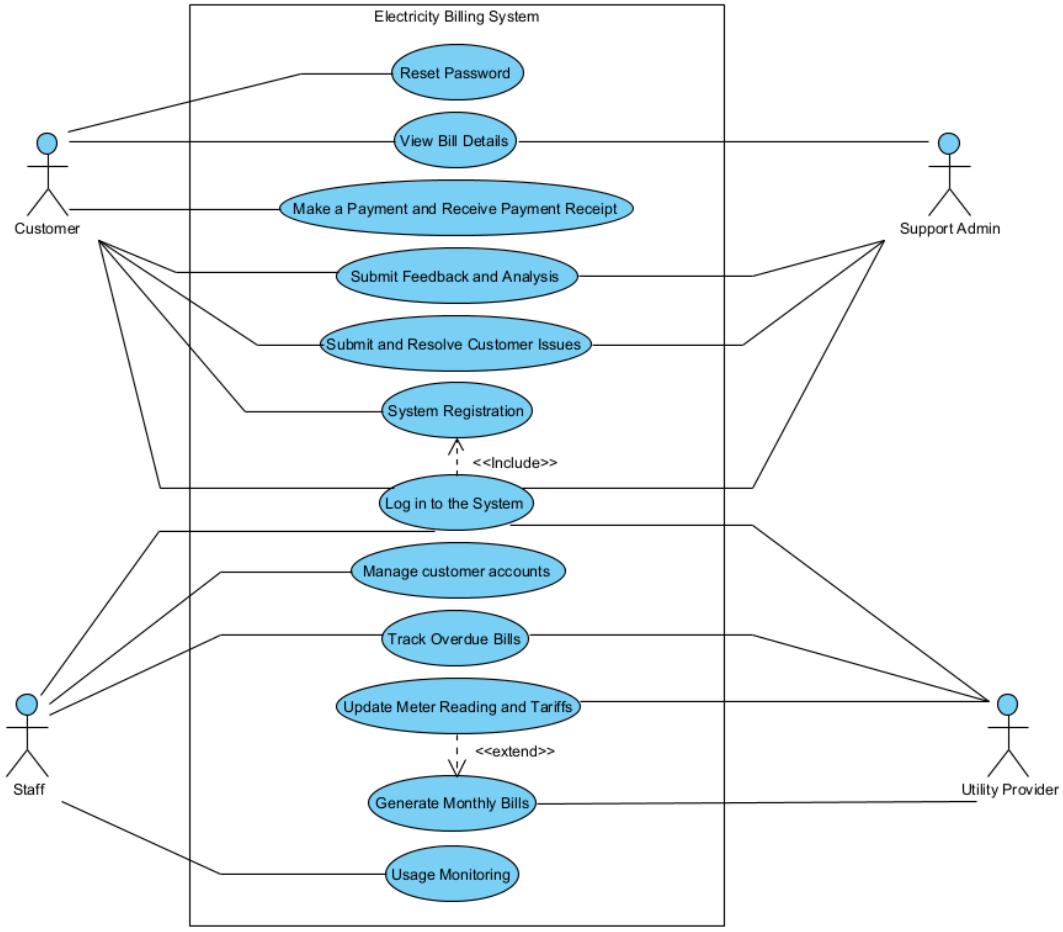
Microsoft Access, a robust and scalable relational database system, will manage all system data effectively. It ensures secure storage, fast retrieval, and the capability to scale as the system grows, supporting seamless operations and reliability.

Ongoing Maintenance and Support

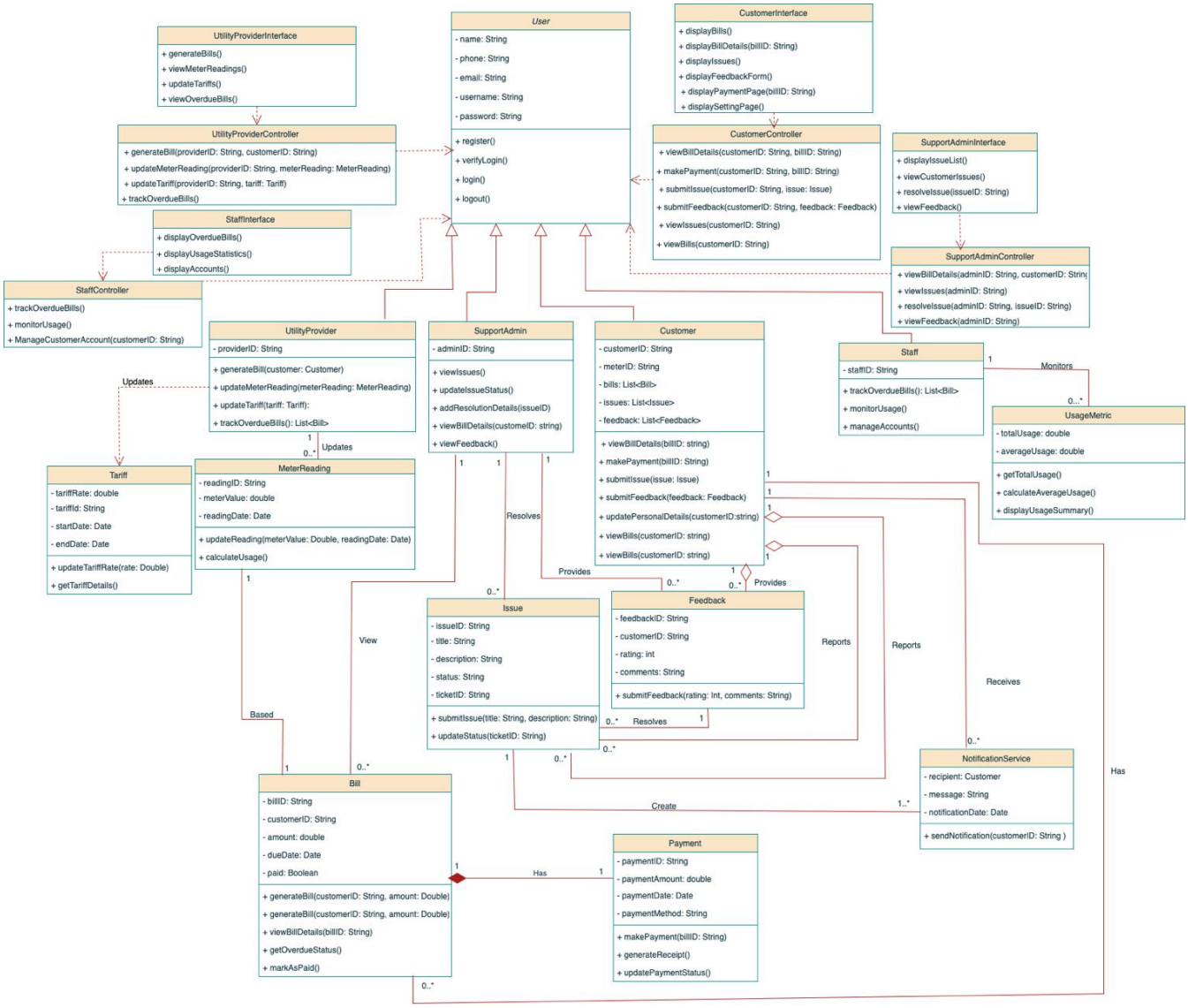
Continuous system updates, performance monitoring, and regular maintenance will be essential to address emerging needs, improve functionality and support best system performance.

2 Requirements

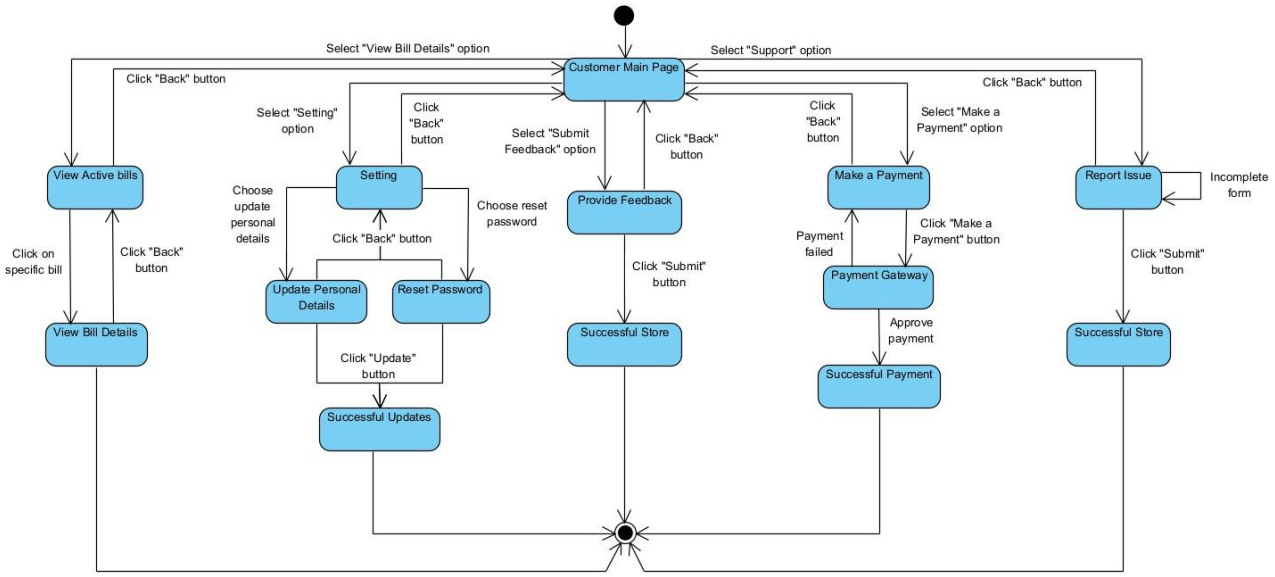
2.1 Use Case Diagram



2.2 Class Diagrams / ERD



2.3 State Diagrams



3 Design

3.1 Use Cases

3.1.1 Use Case 1: System Registration

Actors: Customer
Description:

The system allows new customers to register by providing their meter ID to connect the account with and their personal information. Then the system will verify the provided information by sending a code to the customer's email it also checks with the system database for any duplication (e.g., email, meter number). The Customer account is created and stored in the system database if no duplication has been found. Upon successful registration, the user can easily access the system by logging in to the system.

Preconditions: N/A
Main Flow:

1. The customer navigates to the homepage and selects the "New Customer Registration" option.
2. The customer enters their meter ID.
3. The customer clicks the "Next" button, so the system validates the provided ID.
4. The customer enters the required details, including:
   - Name
   - Email
   - Phone number
   - Address
   - Username
   - Password
5. The customer clicks the "Register" button, which submits the registration form.
6. The system validates the provided information to ensure they are complete and correctly formatted.
7. The system checks for any duplicate accounts based on the email or meter number.
8. If no duplicate records are found, the system creates a user account and stores the customer information in the database.

Postconditions:

- A new customer account is successfully created and securely stored in the system database.

Alternative Flows:

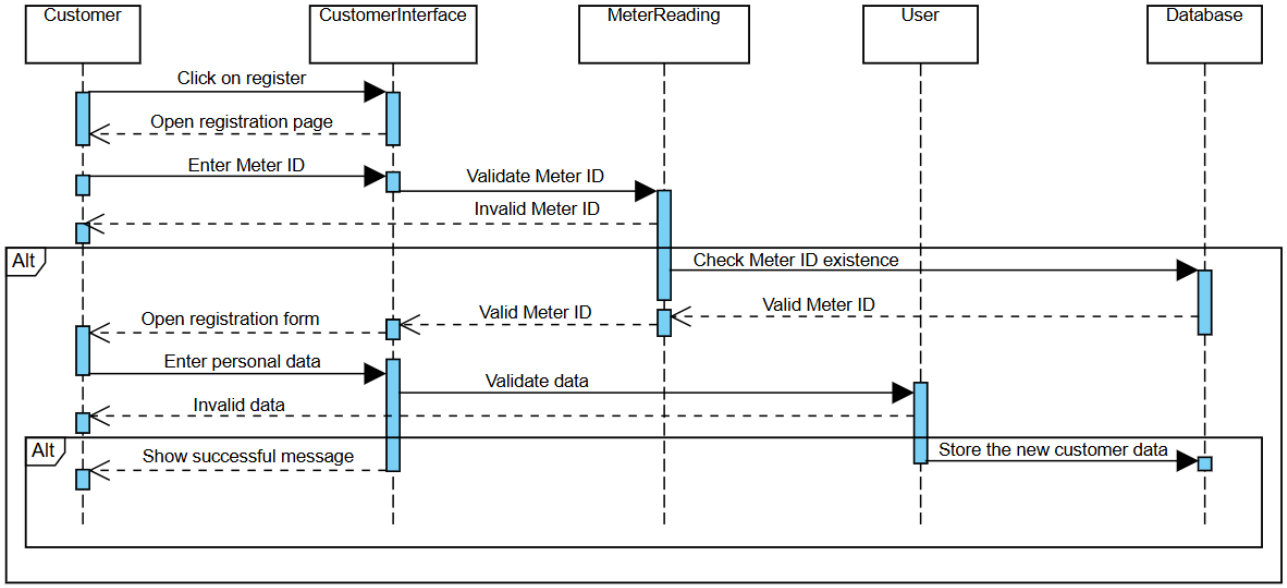
- **Incomplete or Invalid Information:**
  - If required fields are incomplete or contain invalid data, the system displays an error message "Please complete all required fields or correct invalid entries".
  - The system marked the fields requiring correction for the customer’s attention.
  - The customer updates the information and resubmits the registration form.

- **Duplicate Account Detection:**
  - If a duplicate account is detected based on the provided email or meter number, the system displays a message such as "An account with this email or meter ID already exists".
  - The customer is prompted to provide unique information again or use a different email or meter number to proceed.

Assumptions:

- Users have a stable internet connection while attempting to log in.
- The system is operational during the customer’s registration attempt.

The customer provides accurate and verifiable personal or organizational details for registration.



3.1.2 Use Case 2: Log in to the System

Actors: Customer, Staff, Support Admin, and Utility Provider.
Description:
This use case enables the user to log into the system using valid credentials. The system will open the respective System dashboard based on the user role, ensuring that users interact only with the needed functionalities based on their role.
Preconditions:

- The system and its database must be operational.
- The user must possess a registered account.

Main Flow:

1. The user accesses the system's login page.
2. The user inputs their registered username and password into the corresponding fields.
3. The user submits the login form by clicking the "Login" button.
4. The system verifies the provided credentials against the database.
5. If the credentials are correct. The user is redirected to their personalized dashboard or homepage, where features and options are tailored to their role.

Postconditions:

- The user successfully accesses the system with functionality appropriate to their role.
- A record of the login is stored in the system for tracking purposes.

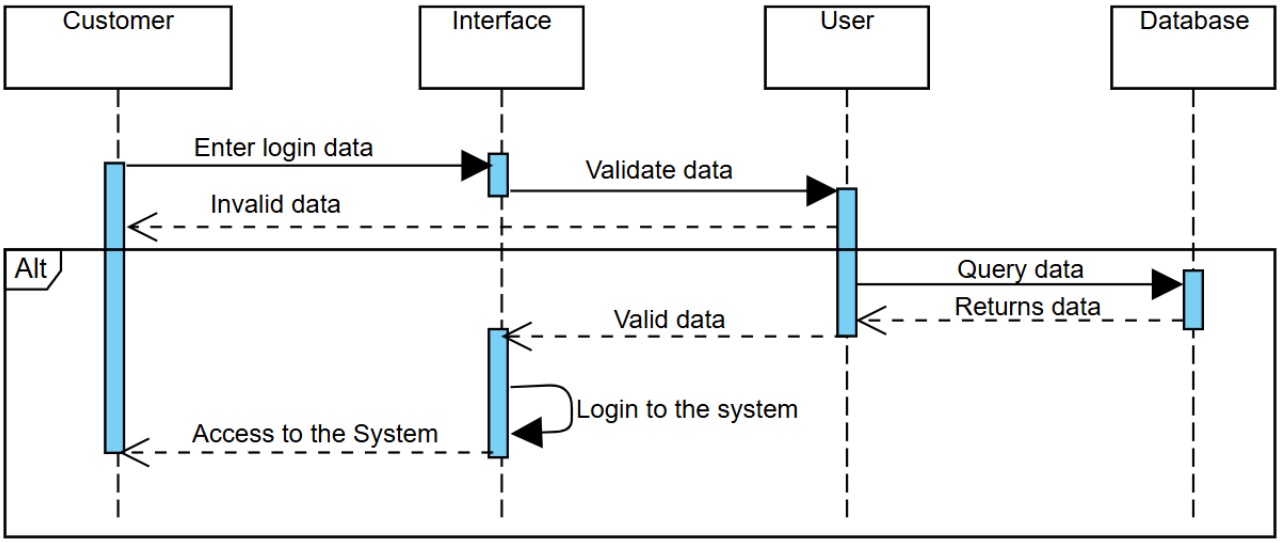
Alternative Flows:

- **Invalid Credentials:**
  If the entered username or password is incorrect:
  - The system returns an error "Invalid username or password."
  - The user must re-enter their credentials.
- **Forgotten Password:**
  - Click the "Forgot Password" option.
  - Provide their registered email to receive password reset instructions.
  - Follow the reset process and attempt to log in again.

Assumptions:

- Users have a stable internet connection while attempting to log in.
- The authentication system and database are functioning correctly.

Role-based permissions for users are accurately configured within the system.



3.1.3 Use Case 3: Reset Password

Actors: Customer

Description:

This use case enables the customer to reset their password either if they forget it or prefer to. The system will ask for the customer's email, so it sends the reset instructions to it. Further on the database will be updated so the customer can use their new password.

Preconditions:

- The system and its database must be operational.
- The user must possess a registered account.

Main Flow:

1. The customer selects the forget password option or navigates to the settings and selects the reset password option.
2. The system will ask the user to provide their email address, so it can send the instructions to it.
3. The customer will follow the instructions and set their new password.
4. The system will update the old password and store the new one in the database.

Postconditions:

- The user successfully updated their password.
- A record of the operation is stored in the system for tracking purposes.

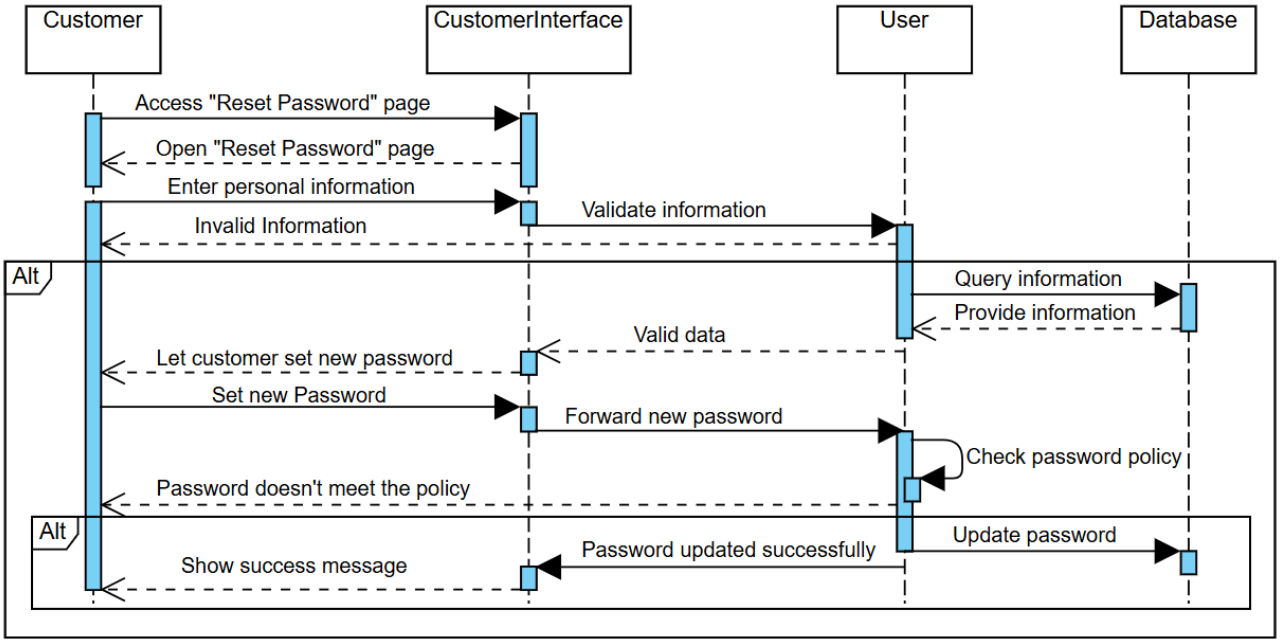
Alternative Flows:

- **If the new password matches the old one:**
  - The system will display an error message “Your new password can’t be the same as the old one”.
  - The customer must enter another password.

Assumptions:

- Customers have a stable internet connection while resetting their password.

The customer provides a different password than their old one.



3.1.4 Use Case 4: Make a Payment and Receive Payment Receipt

Actors: Customer

Description:

This use case describes the process of making a payment for electricity bills and receiving a digital receipt.

Preconditions:

- The customer must be registered and logged into the system.
- The customer must have at least one active bill.

Main Flow:

1. The customer navigates to “Make a Payment” section.
2. The system retrieves a list of all the customer's active bills and displays them to the customer.
3. The customer selects specific bills to pay.
4. The customer selects the preferred payment method (e.g., credit card, bank transfer).
5. The customer clicks the "Make Payment" button to proceed.
6. The system redirects the customer to the selected payment gateway.
7. The customer completes the payment transaction through the payment gateway.
8. Upon successful payment confirmation from the gateway, the system updates the status of the selected bills to "Paid".
9. The system generates and displays a digital receipt to the customer, confirming the successful payment

Postcondition:

- The selected bills are marked as "Paid" in the system.
- The customer receives a digital receipt confirming the payment.

Alternative Flow:

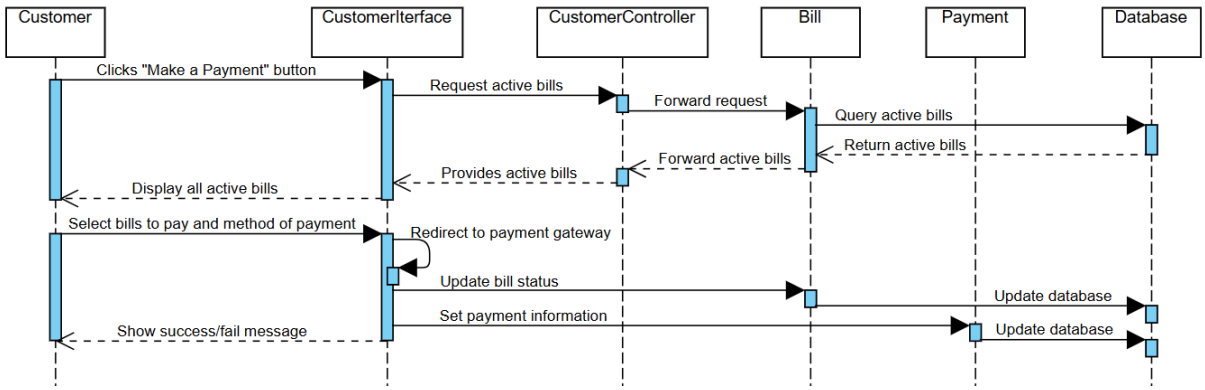
Payment Failure:

- If the payment fails at the gateway, the system displays an error message to the customer.
- The bill status remains unchanged.

Assumptions:

- The customer is able to access the system with a stable internet connection.
- The payment gateway integration is stable and reliable.
- The system can handle various payment methods and currencies.

The system ensures the security and confidentiality of customer payment information.



3.1.5 Use Case 5: View Bill Details

Actors: Customer

Description:

This use case allows the customer to view detailed electricity billing information, such as outstanding balances, consumption records, and payment history, through a user-friendly interface.

Preconditions:

- The customer must have a registered account.
- The customer must be logged into the system.
- Relevant billing data must exist and be accessible in the system.

Main Flow:

1. The customer navigates to the "View Bills" section in the system interface.
2. The customer clicks on the specific bill for which they want to view details.
3. The system retrieves billing details based on the selected bill.
4. The customer reviews the displayed billing information:
   - Outstanding balance.
   - Total electricity consumption.
   - Payment due date.

Postconditions:

The customer successfully views their bill.

Alternative Flows:

No bill data found

The system sending the customer a message, "No billing information available." in case of no billing data available at its end.

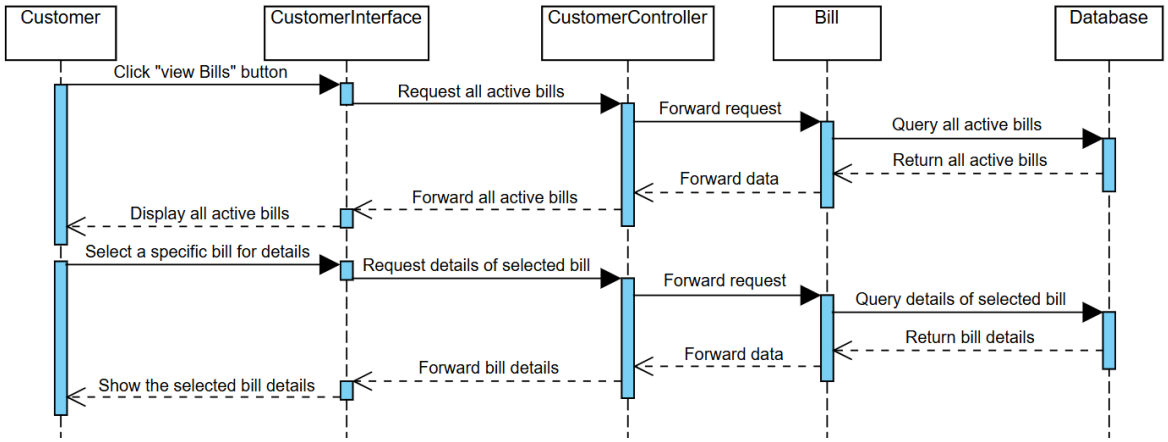
Error Retrieving Bill from System

If there is an error during bill retrieve the system displays a message (e.g., "Unable to retrieve bill details. Please try again later.").

The error is logged, and the customer arise issue to the support administrator for resolution.

Assumptions:

- The customer is able to access the system with a stable internet connection.
- Billing data is updated in real-time or at regular intervals for accuracy.
- The customer account is active, authorized, and not subject to any restrictions.



3.2 Data Dictionary

- *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.

- *Bill Tabel*

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.

- *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.

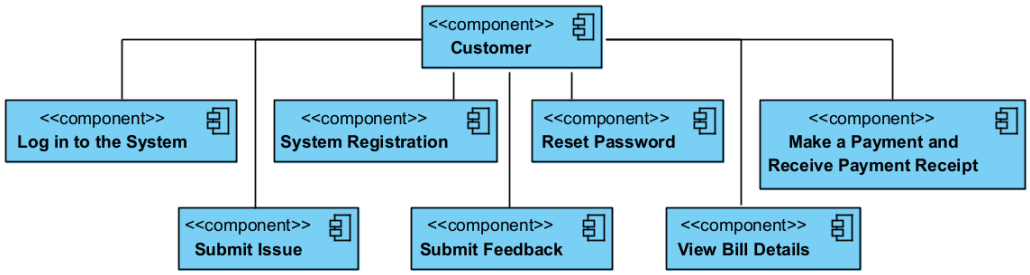
- *Meter Table*

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.

3.3 Subsystem Architecture

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.



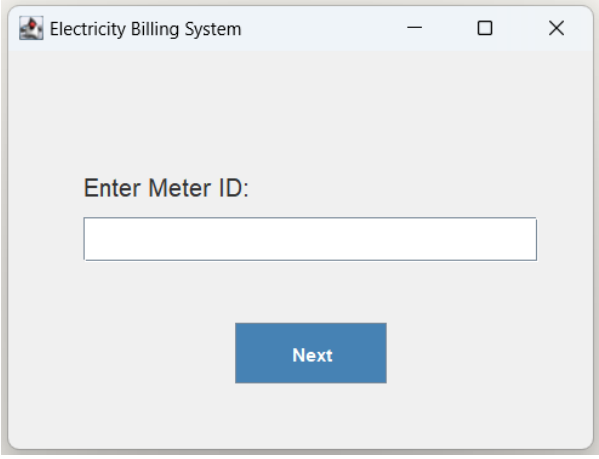
3.4 Subsystem Screens

3.4.1 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.

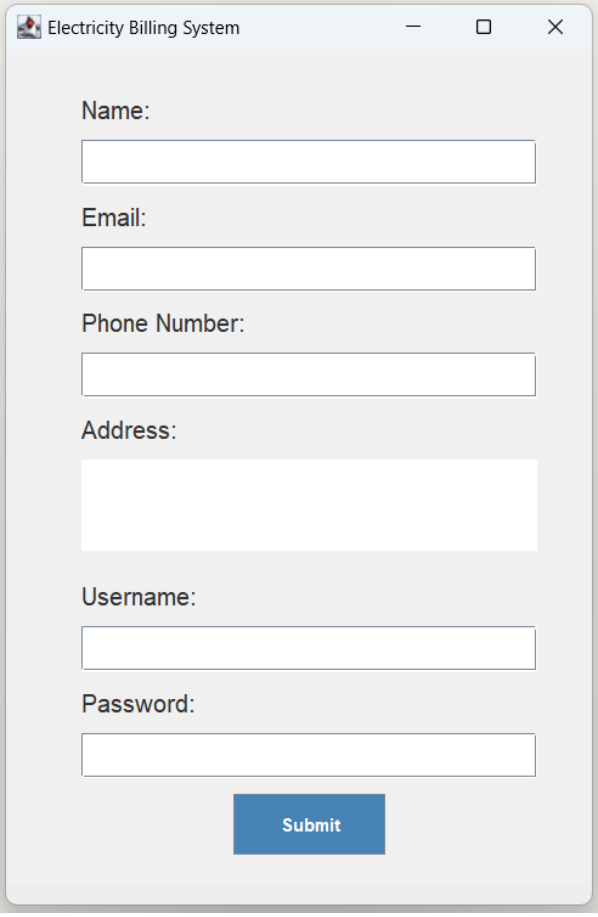
This step ensures accurate billing and proper association of accounts with the correct meter, forming the foundation for reliable service.



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:

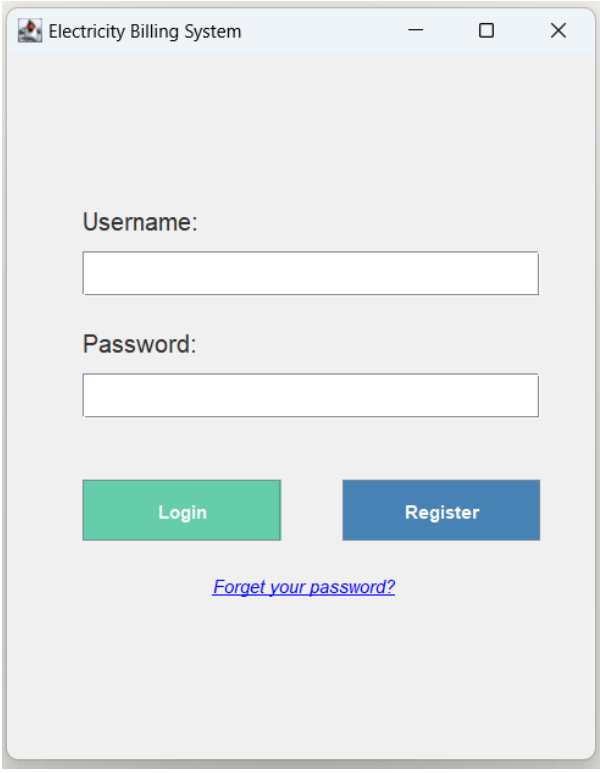
- Name
- Email
- Phone Number
- Address
- Username
- Password

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.



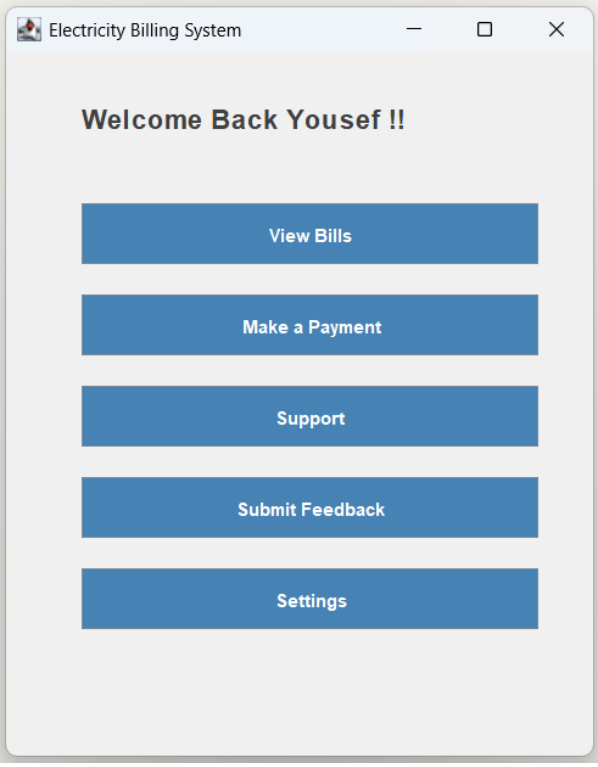
3.4.2 Log in Screen

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.



3.4.3 Main Customer 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.



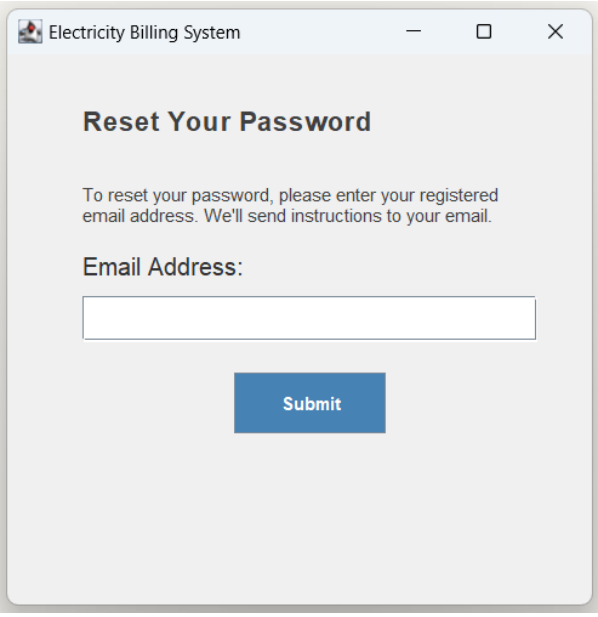
- **Customer-Specific Welcome Message**
  The screen displays a personalized message, such as "Welcome Back Yousef!". This feature confirms successful login and enhances the user experience by providing a warm and tailored greeting.
- **View Bills**
  The "View Bills" button allows customers to view 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.

3.4.4 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.



3.4.5 Report Issue Screen

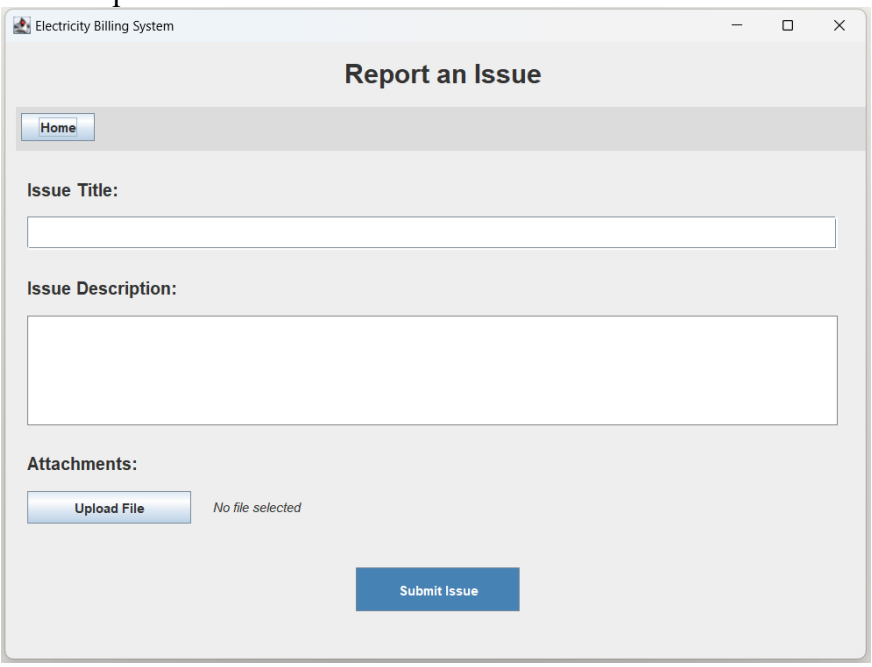
This interface allows customers to easily report problems or issues related to their electricity service.

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.

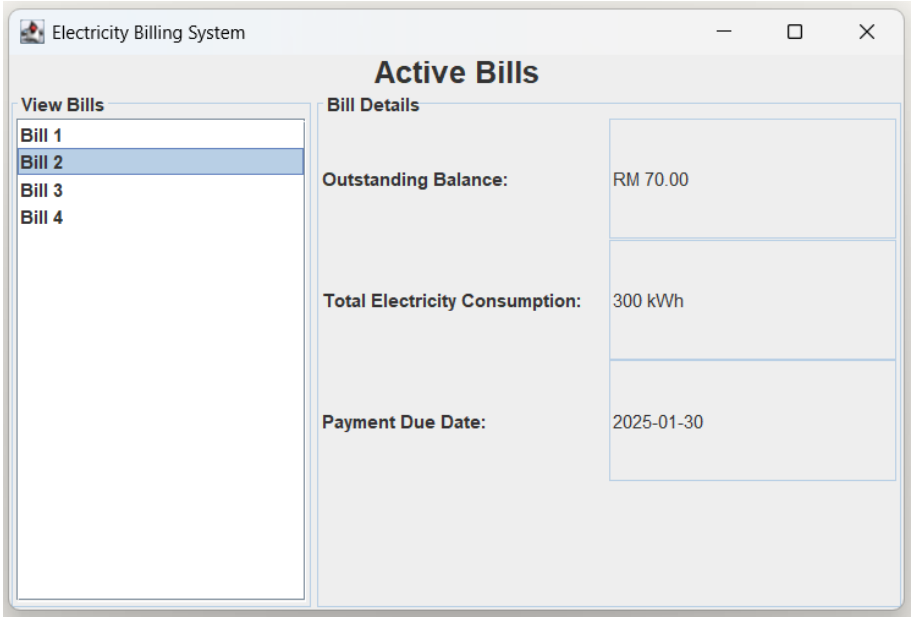
Attachments: This section allows the customer to upload any supporting documents or images related to their issue. Upload file button triggers the file upload dialog, allowing to select files from the computer with message indicates if no file has been selected.
Submit Button: Once the user has entered the necessary information, they can click this button to submit the issue report.



3.4.6 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:

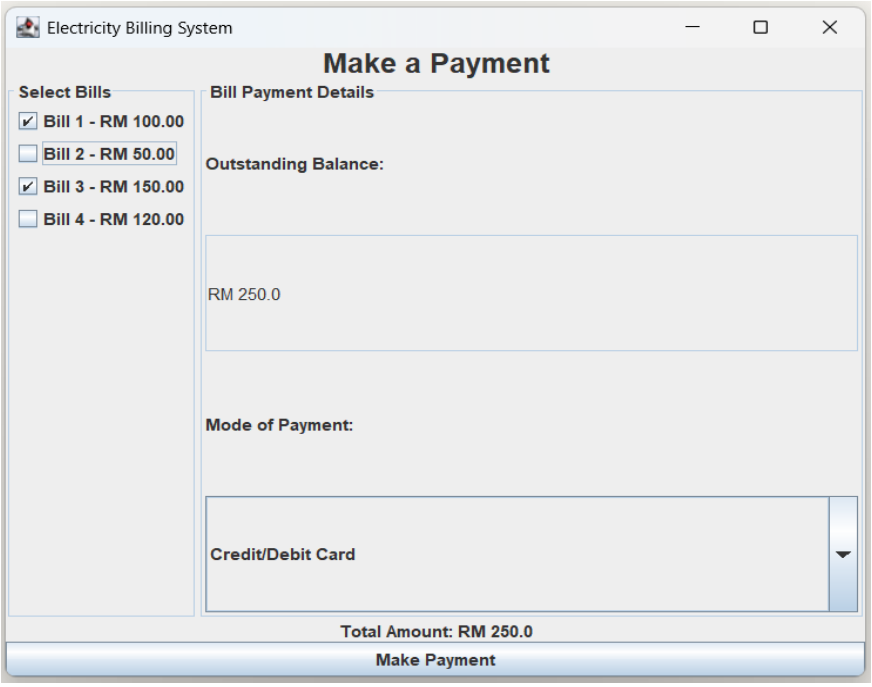
- Outstanding Balance: The amount to be paid for the selected bill.
- Total Electricity Consumption: The total amount of electricity consumed during the billing period.
- Payment Due Date: The date by which the payment for the selected bill is due.



3.4.7 Make a Payment Screen

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

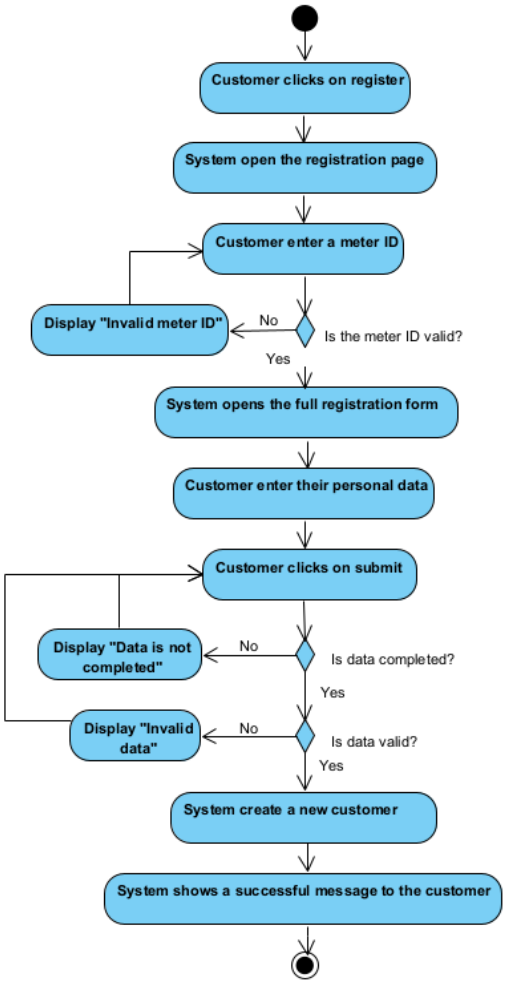
- Select Bills: A list displaying all active bills associated with the customer's account. Each bill is presented with a checkbox to select and its corresponding amount. Customers can select multiple bills for payment.
- Outstanding Balance: Displays the total amount due for the selected bills.
- Mode 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.



3.5 Subsystem Components

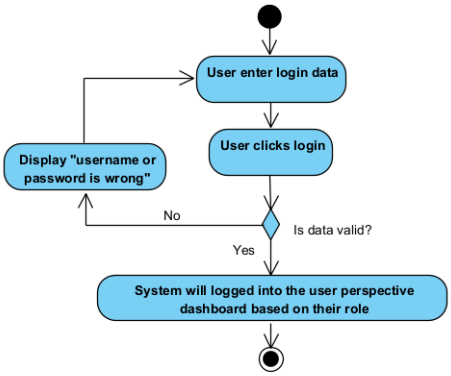
3.5.1 Component 1: System Registration

The activity diagram illustrates in *Figure xx* 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.



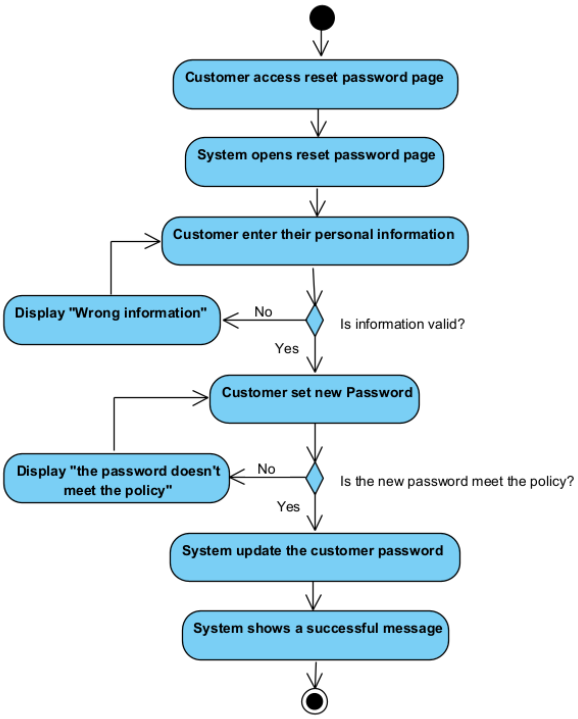
3.5.2 Component 2: Log in to the System

The activity diagram in *Figure xx* 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.



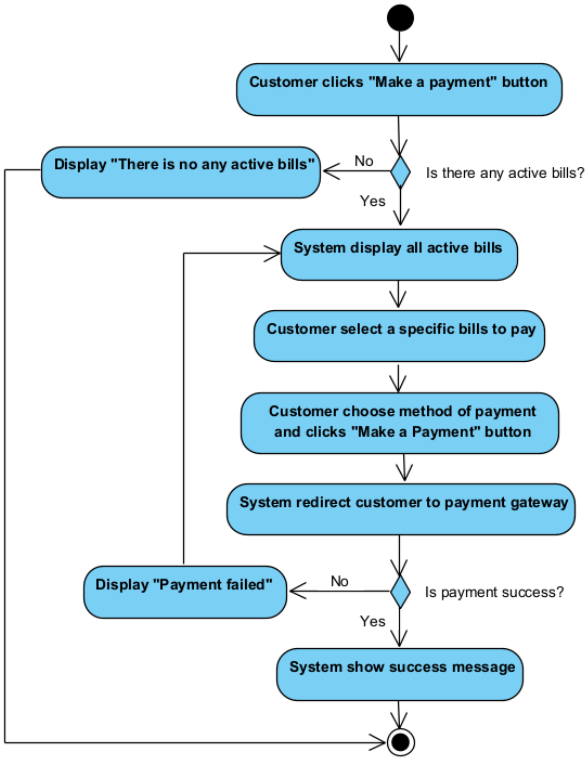
3.5.3 Component 3: Reset Password

The activity diagram in *Figure 56* 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.



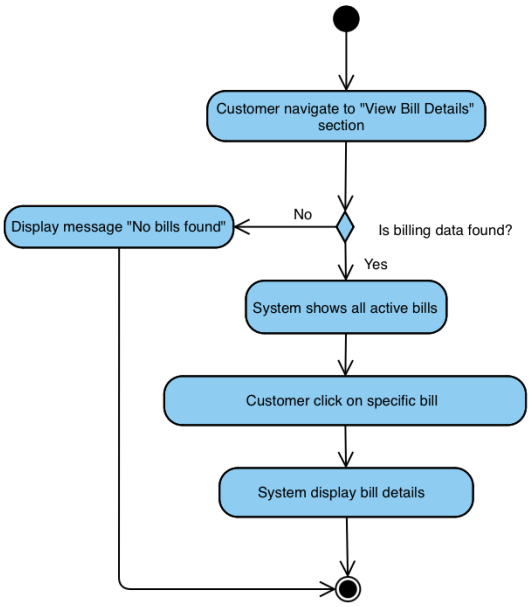
3.5.4 Component 4: Make a Payment and Receive Payment Receipt

The activity diagram in *Figure xx* 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.



3.5.5 Component 5: View Bill Details

The activity diagram shown in *Figure xx* 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.


Customer navigate to "View Bill Details" section
Is billing data found?
No
Display message "No bills found"
Yes
System shows all active bills
Customer click on specific bill
System display bill details

4 Implementation

4.5 Development Environment

- 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)
```

```python
    def __str__(self):
        return f"Payment {self.payment_id} (Customer: {self.customer.customer_name})"

    class Meta:
        verbose_name = "Payment"
        verbose_name_plural = "Payments"


class Issue(models.Model):
    """
    Model for Customer Issues.
    """
    STATUS_CHOICES = [
        ('open', 'Open'),
        ('in_progress', 'In Progress'),
        ('resolved', 'Resolved'),
    ]

    issue_id = models.CharField(primary_key=True, max_length=6)
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    description = models.TextField(max_length=300)
    status = models.CharField(max_length=15, choices=STATUS_CHOICES, default='open')
    ticket_id = models.CharField(max_length=6, unique=True)
    support_admin = models.ForeignKey(SupportAdmin, on_delete=models.SET_NULL, null=True, blank=True)

    def __str__(self):
        return f"Issue {self.issue_id} (Customer: {self.customer.customer_name})"

    class Meta:
        verbose_name = "Issue"
        verbose_name_plural = "Issues"

```

```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}"

```

- 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')


```

```python
def home(request):
    """Renders the home page."""
    assert isinstance(request, HttpRequest)

    # Render the same page regardless of login status
    return render(
        request,
        'app/index.html',
        {
            'title': 'Home Page',
            'year': datetime.now().year,
        }
    )

def contact(request):
    """Renders the contact page."""
    assert isinstance(request, HttpRequest)
    return render(
        request,
        'app/contact.html',
        {
            'title':'Contact',

            'year':datetime.now().year,
        }
    )
```

```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)
```

```python
def registration(request):
    """
    Step 1: Validate meter ID and initiate registration process.
    """
    if request.method == 'POST':
        meter_id = request.POST.get('meter_id')

        # Validate Meter ID
        try:
            # Check if meter ID exists and is not already assigned to a customer
            meter = Meter.objects.get(meter_id=meter_id)
            if hasattr(meter, 'customer'):
                return render(request, 'customer/registration.html', {
                    'error': 'This meter is already registered to a customer.'
                })

            # Store meter ID in session for step 2
            request.session['meter_id'] = meter_id
            return redirect('registration_step2')

        except Meter.DoesNotExist:
            return render(request, 'customer/registration.html', {
                'error': 'Invalid meter ID. Please check and try again.'
            })

    return render(request, 'customer/registration.html')

```

```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})

# ---------------------------------------------
# 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})
```

```python
# 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}")

```

4.6 Main Program Codes

Application	Files
Customer Dashboard	models.py views.py customer_dashboard.html

View Bill Details	models.py views.py bill_detaills.html
System Registration	models.py views.py registration_1.html registration_2.html
Submit Feedback	models.py views.py submitFeedback.html
Report an Issue	models.py views.py submit_Issue.html
Make a Payment	models.py views.py makePayment.html

customer_dashboard.html

```html
{% extends 'base.html' %}
{% block title %}Customer Dashboard{% endblock %}
{% block content %}

<!-- Navigation Bar -->
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">Customer Dashboard</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav"
        aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav ms-auto">
                <li class="nav-item">
                    <a class="nav-link" href="{% url 'logout' %}">Logout</a>
                </li>
            </ul>
        </div>
    </div>
</nav>

<!-- Main Content -->
<div class="container mt-5">
    <h2 class="text-center mb-4">Welcome Back, {{ user.username }}!</h2>
    <div class="row">
        <!-- View Bills Card -->
        <div class="col-md-4 mb-4">
            <div class="card shadow">
                <div class="card-body text-center">
                    <h5 class="card-title">View Bills</h5>
                    <p class="card-text">Check your current and past bills.</p>
                    <a href="{% url 'view_bills' %}" class="btn btn-primary">Go to Bills</a>
                </div>
            </div>
        </div>
```

```html
        <!-- Make Payment Card -->
        <div class="col-md-4 mb-4">
            <div class="card shadow">
                <div class="card-body text-center">
                    <h5 class="card-title">Make Payment</h5>
                    <p class="card-text">Pay your bills securely online.</p>
                    <a href="{% url 'make_payment' %}" class="btn btn-success">Make Payment</a>
                </div>
            </div>
        </div>

        <!-- Submit Feedback Card -->
        <div class="col-md-4 mb-4">
            <div class="card shadow">
                <div class="card-body text-center">
                    <h5 class="card-title">Submit Feedback</h5>
                    <p class="card-text">Share your feedback with us.</p>
                    <a href="{% url 'submit_feedback' %}" class="btn btn-info">Submit Feedback</a>
                </div>
            </div>
        </div>

        <!-- Submit Issue Card -->
        <div class="col-md-4 mb-4">
            <div class="card shadow">
                <div class="card-body text-center">
                    <h5 class="card-title">Submit Issue</h5>
                    <p class="card-text">Report any issues you're facing.</p>
                    <a href="{% url 'submit_issue' %}" class="btn btn-warning">Submit Issue</a>
                </div>
            </div>
        </div>

        <!-- Reset Password Card -->
        <div class="col-md-4 mb-4">
            <div class="card shadow">
                <div class="card-body text-center">
                    <h5 class="card-title">Reset Password</h5>
                    <p class="card-text">Change or reset your password.</p>
                    <a href="{% url 'change_password' %}" class="btn btn-secondary">Reset Password</a>
                </div>
            </div>
        </div>
    </div>
</div>

<!-- Footer -->
<footer class="bg-dark text-white text-center py-3 mt-5">
    <div class="container">
        <p class="mb-0">&copy; 2025 Electricity Billing System. All rights reserved.</p>
    </div>
</footer>

{% endblock %}
```

- **bill_details.html**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Bill Details</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        .bill-details {
            max-width: 600px;
            margin: auto;
            padding: 20px;
            border: 1px solid #ddd;
            border-radius: 8px;
        }
        .bill-details h2 {
            text-align: center;
            color: #333;
        }
        .bill-details table {
            width: 100%;
            border-collapse: collapse;
            margin-top: 20px;
        }
        .bill-details table, .bill-details th, .bill-details td {
            border: 1px solid #ddd;
        }
        .bill-details th, .bill-details td {
            padding: 10px;
            text-align: left;
        }
        .bill-details th {
            background-color: #f4f4f4;
        }
    </style>
</head>
```

```html
<body>
    <div class="bill-details">

        <h3>Customer Information:</h3>
        <p><strong>Name:</strong> Mohammed Aarnena</p>
        <p><strong>Email:</strong> ammena@gmail.com</p>
        <p><strong>Phone:</strong> +60 11-62237057</p>
        <p><strong>Address:</strong> 123 Multimedia St, Cyberjaya</p>

        <h3>Payment History:</h3>
        <table>
            <thead>
                <tr>
                    <th>Date</th>
                    <th>Amount</th>
                    <th>Method</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>2024-12-15</td>
                    <td>RM 50</td>
                    <td>Credit Card</td>
                </tr>
                <tr>
                    <td>2024-11-15</td>
                    <td>RM 50</td>
                    <td>Bank Transfer</td>
                </tr>
            </tbody>
        </table>

        <h3>Billing Information:</h3>
        <p><strong>Bill ID:</strong> B9561</p>
        <p><strong>Outstanding Balance:</strong> RM 100.00</p>
        <p><strong>Consumption:</strong> 350 kWh</p>
        <p><strong>Due Date:</strong> 2025-01-15</p>
    </div>
</body>
</html>
```

- **registration1.html**

```html
{% extends "app/layout.html" %}

{% block content %}
<div class="container">
    <h2>Step 1: Verify Meter ID</h2>

    {% if form.errors %}
    <div class="alert alert-danger">
        {% for field, errors in form.errors.items %}
            {% for error in errors %}
                <p>{{ error }}</p>
            {% endfor %}
        {% endfor %}
    </div>
    {% endif %}

    <form method="post">
        {% csrf_token %}
        <div class="form-group">
            <label>{{ form.meter_id.label }}</label>
            {{ form.meter_id }}
            <small class="text-muted">Example: M12345</small>
        </div>
        <button type="submit" class="btn btn-primary">Next</button>
    </form>
</div>
{% endblock %}
```

- **registration_2.html**

```html
{% extends "app/layout.html" %}

{% block content %}
<div class="container">
    <h2>Step 2: Complete Registration</h2>
    <p>Please fill out the form below to complete your registration.</p>

    <!-- Display form errors if any -->
    {% if form.errors %}
    <div class="alert alert-danger">
        <strong>Error!</strong> Please correct the following:
        <ul>
            {% for field, errors in form.errors.items %}
                {% for error in errors %}
                    <li>{{ field|title }}: {{ error }}</li>
                {% endfor %}
            {% endfor %}
        </ul>
    </div>
    {% endif %}

    <!-- Registration Form -->
    <form method="post" class="form-horizontal">
        {% csrf_token %}

        <!-- Username -->
        <div class="form-group">
            <label for="{{ form.username.id_for_label }}" class="col-sm-2 control-label">Username:</label>
            <div class="col-sm-10">
                {{ form.username }}
                <small class="form-text text-muted">
                    Username can only contain letters, numbers, and underscores.
                </small>
            </div>
        </div>

        <!-- Password -->
        <div class="form-group">
            <label for="{{ form.password.id_for_label }}" class="col-sm-2 control-label">Password:</label>
            <div class="col-sm-10">
                {{ form.password }}
                <small class="form-text text-muted">
                    Choose a strong password.
                </small>
            </div>
        </div>

        <!-- Full Name -->
        <div class="form-group">
            <label for="{{ form.customer_name.id_for_label }}" class="col-sm-2 control-label">Full Name:</label>
            <div class="col-sm-10">
                {{ form.customer_name }}
            </div>
        </div>
```

```
            <!-- Full Name -->
            <div class="form-group">
                <label for="{{ form.customer_name.id_for_label }}" class="col-sm-2 control-label">Full Name:</label>
                <div class="col-sm-10">
                    {{ form.customer_name }}
                </div>
            </div>

            <!-- Address -->
            <div class="form-group">
                <label for="{{ form.customer_address.id_for_label }}" class="col-sm-2 control-label">Address:</label>
                <div class="col-sm-10">
                    {{ form.customer_address }}
                </div>
            </div>

            <!-- Phone Number -->
            <div class="form-group">
                <label for="{{ form.customer_phone.id_for_label }}" class="col-sm-2 control-label">Phone:</label>
                <div class="col-sm-10">
                    {{ form.customer_phone }}
                    <small class="form-text text-muted">
                        Format: +60123456789 or 012-3456789
                    </small>
                </div>
            </div>

            <!-- Email -->
            <div class="form-group">
                <label for="{{ form.customer_email.id_for_label }}" class="col-sm-2 control-label">Email:</label>
                <div class="col-sm-10">
                    {{ form.customer_email }}
                </div>
            </div>

            <!-- Submit Button -->
            <div class="form-group">
                <div class="col-sm-offset-2 col-sm-10">
                    <button type="submit" class="btn btn-primary">Register</button>
                </div>
            </div>
        </form>
    </div>
    {% endblock %}
```

- **submitFeedback.html**

```
{% extends 'base.html' %}
{% block title %}Submit Feedback{% endblock %}
{% block content %}
<h2>Submit Feedback</h2>
<form method="post">
    {% csrf_token %}
    <div class="form-group">
        <label for="rating">Rating</label>
        <input type="number" class="form-control" id="rating" name="rating" min="1" max="5" required>
    </div>
    <div class="form-group">
        <label for="comments">Comments</label>
        <textarea class="form-control" id="comments" name="comments" rows="3" required></textarea>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
{% endblock %}
```

- **submitIssue.html**

```
{% extends 'base.html' %}
{% block title %}Submit Issue{% endblock %}
{% block content %}
<h2>Submit Issue</h2>
<form method="post" enctype="multipart/form-data">
    {% csrf_token %}
    <div class="form-group">
        <label for="issue_title">Issue Title</label>
        <input type="text" class="form-control" id="issue_title" name="issue_title" required>
    </div>
    <div class="form-group">
        <label for="issue_description">Issue Description</label>
        <textarea class="form-control" id="issue_description" name="issue_description" rows="3" required></textarea>
    </div>
    <div class="form-group">
        <label for="attachments">Attachments</label>
        <input type="file" class="form-control" id="attachments" name="attachments">
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
{% endblock %}
```

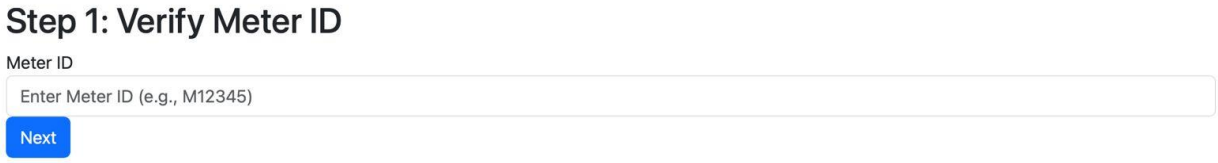
- **makePayment.html**

```html
{% extends 'base.html' %}
{% block title %}Make Payment{% endblock %}
{% block content %}
<h2>Make Payment</h2>
<form method="post">
    {% csrf_token %}
    <div class="form-group">
        <label for="bill_id">Select Bill</label>
        <select class="form-control" id="bill_id" name="bill_id" required>
            {% for bill in bills %}
            <option value="{{ bill.id }}">Bill {{ bill.id }} - {{ bill.amount }}</option>
            {% endfor %}
        </select>
    </div>
    <div class="form-group">
        <label for="payment_method">Payment Method</label>
        <select class="form-control" id="payment_method" name="payment_method" required>
            <option value="credit_card">Credit Card</option>
            <option value="bank_transfer">Bank Transfer</option>
        </select>
    </div>
    <button type="submit" class="btn btn-primary">Pay Now</button>
</form>
{% endblock %}
```

4.7 Sample Screens

4.7.1 Registration Screen

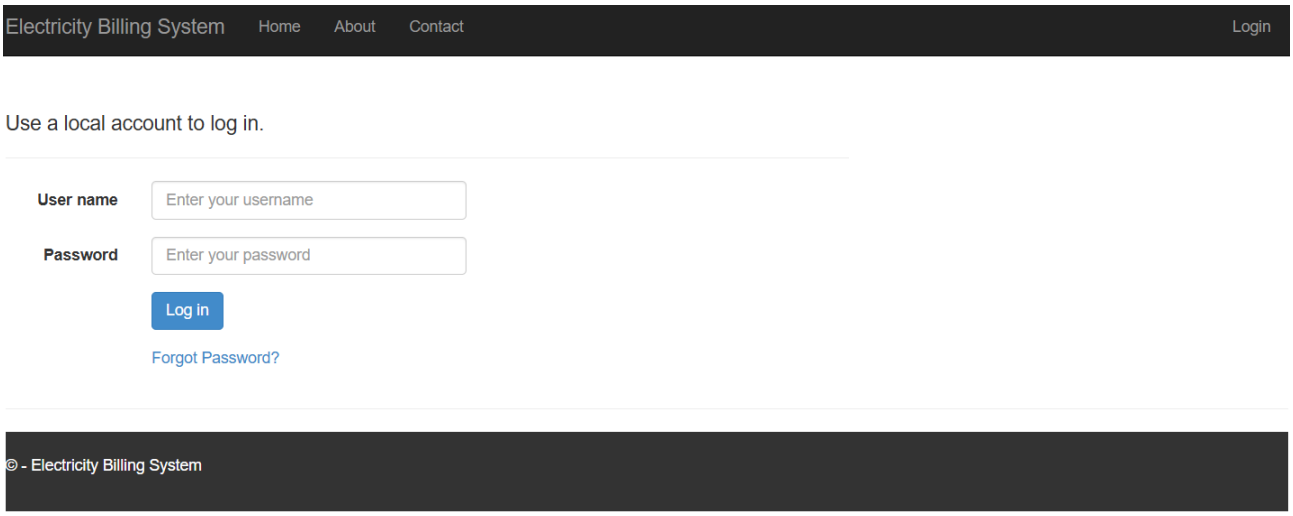
The "Enter Meter ID" screen is displayed first, it’s a critical role in linking a user’s account to their designated electricity meter. Users are required to input their unique Meter ID before proceeding by selecting "Next." The system validates the provided Meter ID for accuracy before directing the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

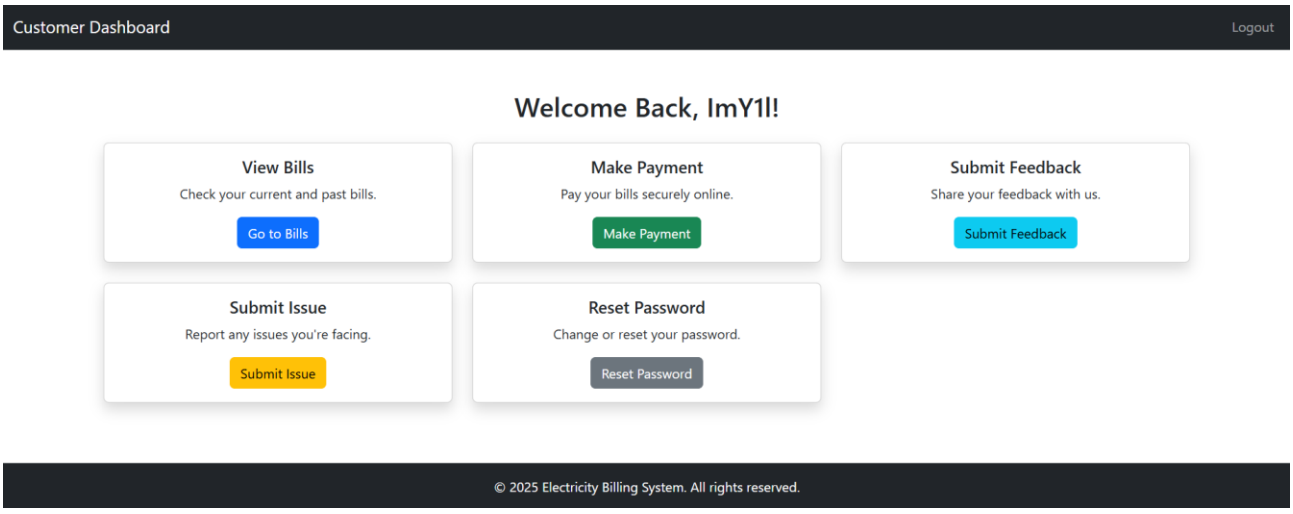
4.7.2 Log in Screen

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.



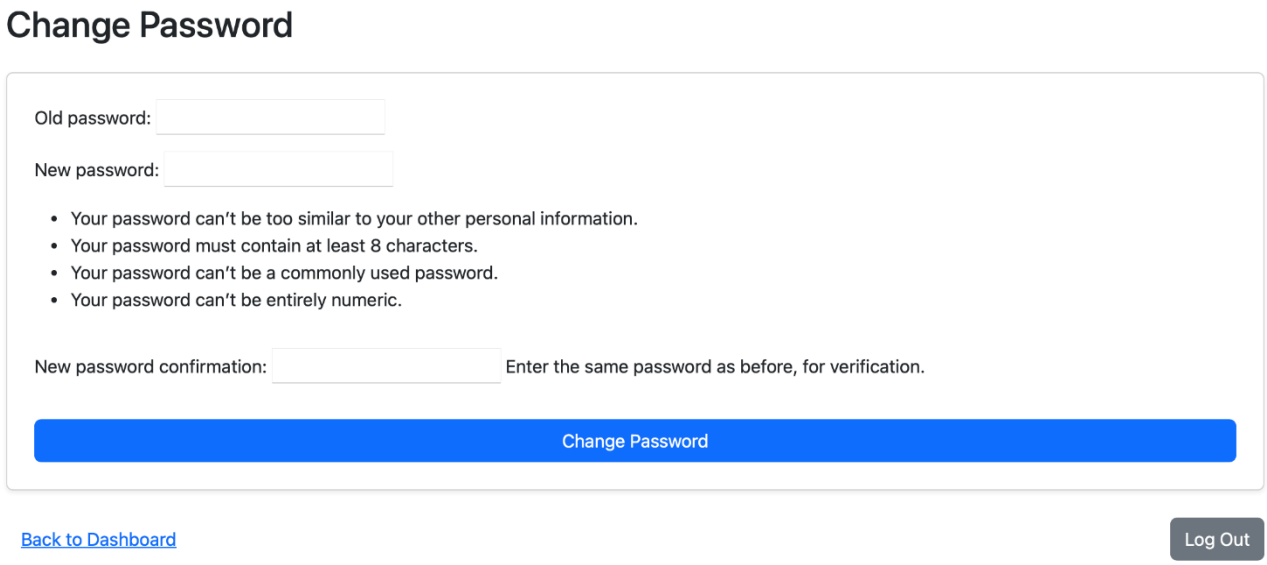
4.7.3 Main Customer Screen

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 "Yousef," contributing to a more personalized interface.



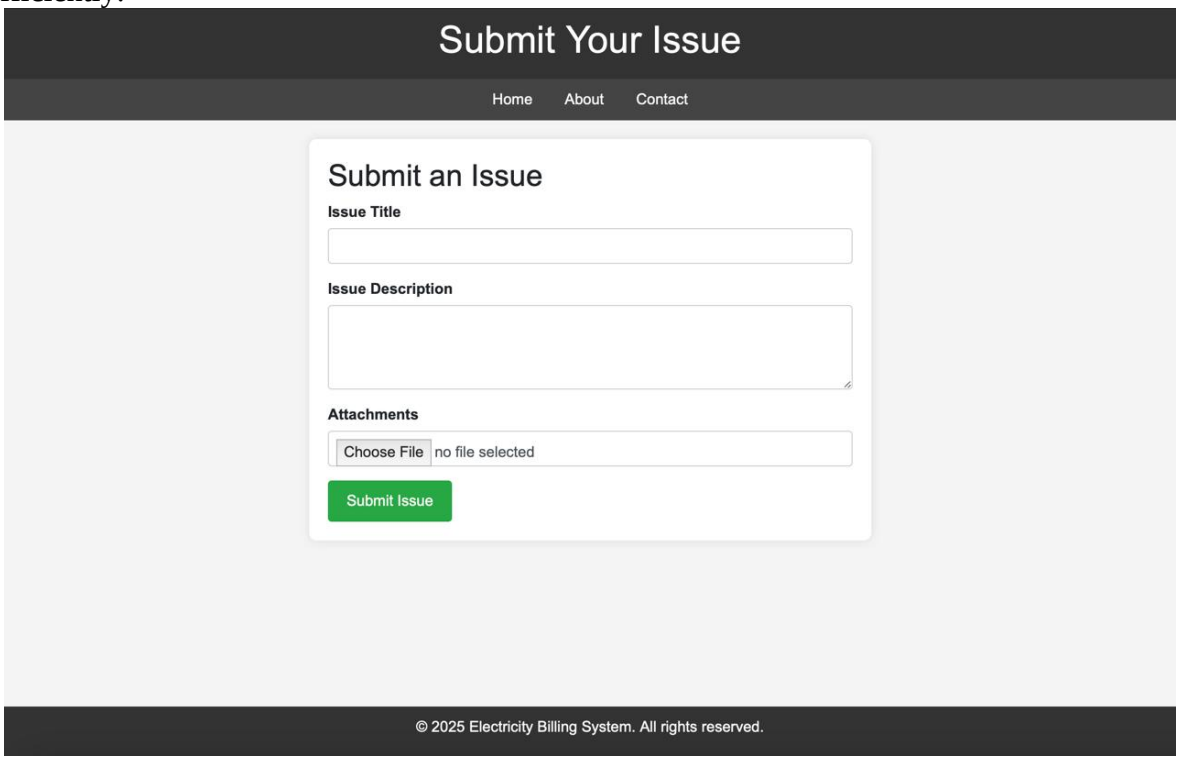
4.7.4 Reset Password Screen

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.



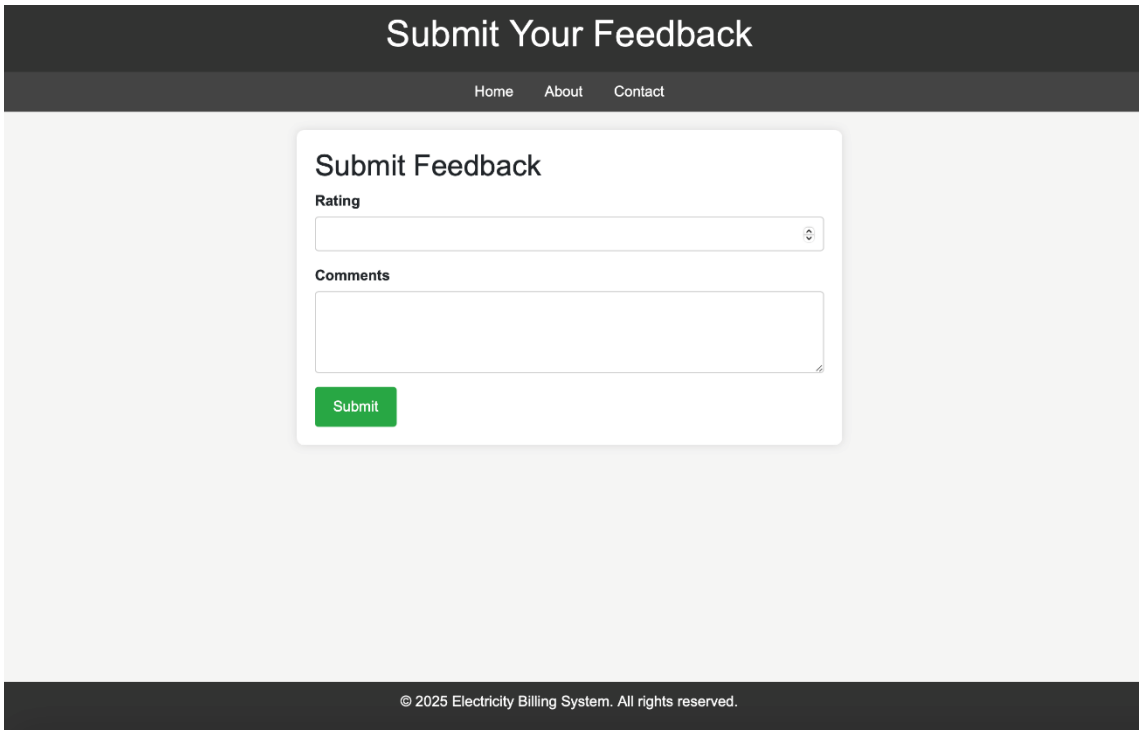
4.7.5 Report an Issue Screen

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



4.7.6 Submit Feedback

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



4.7.7 View Bills Screen

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

4.7.8 Make a Payment Screen

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

Make a Payment

Bill Payment Details

Bill 1 - RM 100.00
Bill 2 - RM 50.00
Bill 3 - RM 150.00
Bill 4 - RM 120.00

Outstanding Balance:

RM 250.0

Mode of Payment:

Credit/Debit Card

Total Amount: RM 250.0

Make Payment

5 Testing

5.5 Test Data

5.5.1 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

5.5.2 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

5.5.3 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

5.5.4 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

5.5.5 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

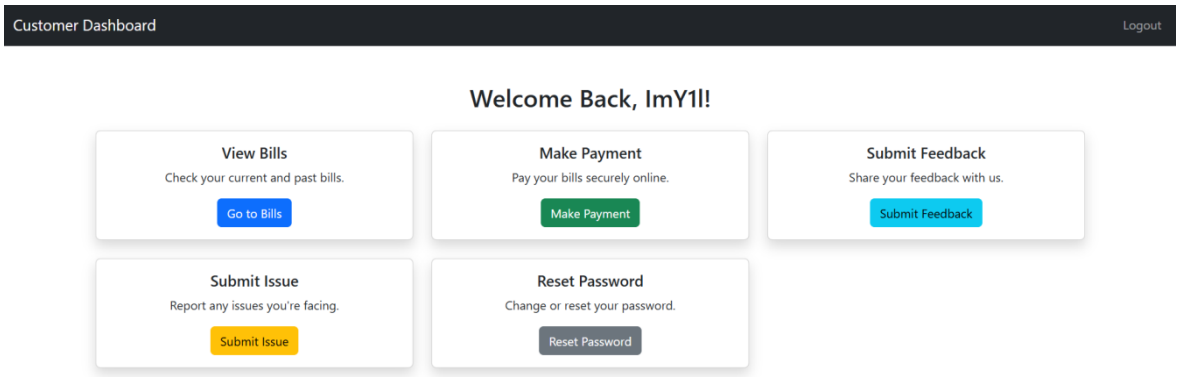
5.6 Acceptance Testing: Customer

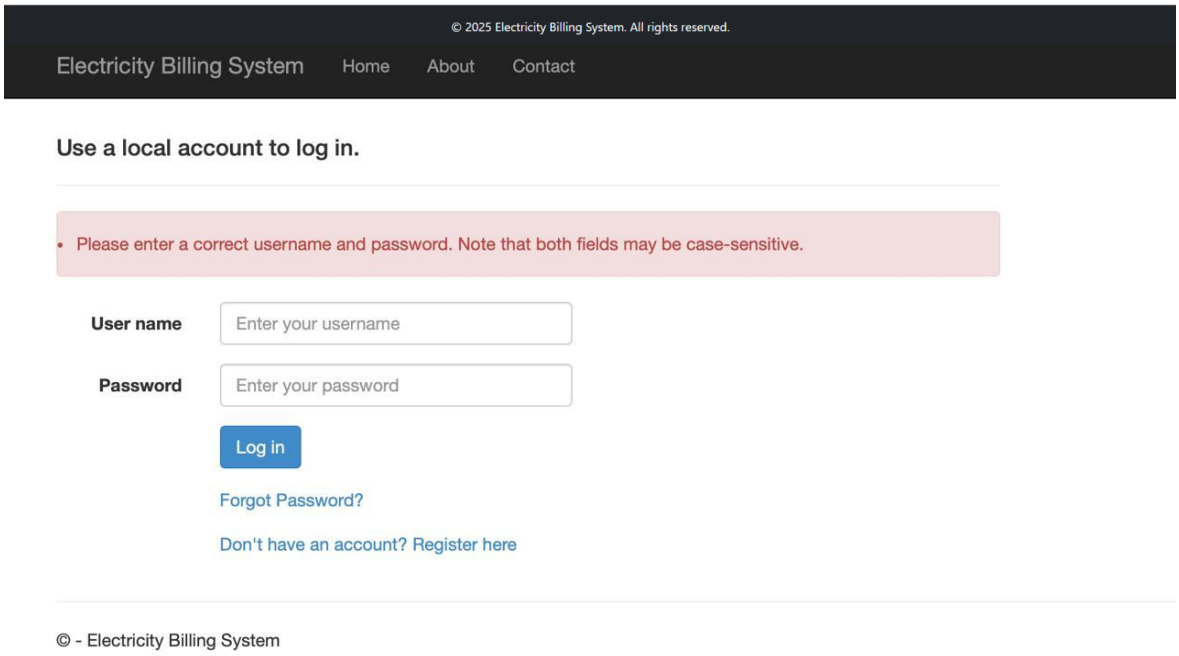
Criteria	Fulfilled?	Remarks
Log in to the System	Yes	

View Dashboard	Yes	
Data Checking	Yes	

Date tested : 28/2/2025
% Complete : 100%
Tested by : Mohammed Yousef Mohammed Abdulkarem
Verified by : Dr. Palanichamy Naveen

5.7 Test Results





6 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.

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.