Introduction

Electricity Billing System is a software-based application.

i. This project aims at serving the department of electricity by computerizing thebilling system.
ii. It mainly focuses on the calculation of units consumed during the specified timeand the money to be charged by the electricity offices.
iii. This computerized system will make the overall billing system easy, accessible,comfortable, and effective for consumers.

- To design the billing system more service oriented and simple, the following featureshave been implemented in the project. The application has high speed of performance with accuracy and efficiency.
- The software provides facility of data sharing, it does not require any staff as in the conventional system. Once it is installed on the system only the meter readings are to be given by the admin where customer can view all details, it has the provision of security restriction.
- The electricity billing software calculates the units consumed by the customer and makes bills, it requires small storage for installation and functioning. There is provisionfor debugging if any problem is encountered in the system.
- The system excludes the need of maintaining paper electricity bill, administrator does not have to keep a manual track of the users, users can pay the amount without visiting the office. Thus, it saves human efforts and resources.

1.1 Objective

- We, the owners of our project, respect all the customers and make them
- Happy with our services.
- The main aim of our projects is to satisfy customers by saving their time by payment
- Processes, maintaining records, and allowing the customer to view their records and
- permitted them to update their details.
- The firm handles all the work manually, which is very tedious and mismatched.

The objectives of our project are as follows:

- ❖ Keep the information of Customers. Manage customer info well for success
  Manage customer information effectively and efficiently. Maintain comprehensive customer records.
- ❖ To keep the information of consuming unit energy of current month.
  Maintain accurate records of energy consumption for the current month.Record energy consumption for the current month.
- ❖ To maintain detailed records of energy consumption from the previous month.
- ❖ To consistently compute monthly energy consumption units with precision and regularity.
- ❖ To systematically generate bills incorporating penalties and rental charges with meticulous attention to detail and accuracy.
- ❖ To optimize efficiency through the implementation of an online payment system, thereby streamlining processes and saving valuable time.

1.2 Problem Definition

- ♦ The existing manual system is fraught with numerous shortcomings that pose significant obstacles to both operational efficacy and customer contentment. Chief among these drawbacks is the laborious and time-intensive nature of manually maintaining bills, which demands substantial human effort and resources. The meticulous recording and preservation of information by hand not only consume extensive time but also introduce a heightened risk of errors and inaccuracies, thereby jeopardizing the integrity of the data.

  Moreover, the traditional approach necessitates periodic visits by staff to customers' premises for bill dissemination and payment collection, exacerbating inefficiencies and inconveniences inherent in the system. This repetitive process not only strains human resources but also imposes burdens on customers, who must allocate time for these visits regularly.
- ♦ Recognizing the pressing need for reform, there exists a compelling imperative for a transition towards automation to streamline operations and bolster overall efficiency. Embracing computerization presents an opportunity to minimize redundant tasks, consolidate information, and eliminate repetitive data entry processes. By leveraging technology, businesses can mitigate the labor-intensive nature of the current system and deliver services with greater promptness and precision, thereby enhancing customer satisfaction.

- ♦ The integration of automation holds transformative potential, promising to revolutionize the operational landscape and usher in an era of heightened efficiency and customer-centricity. By embracing this shift, organizations can realize manifold benefits, including enhanced productivity, reduced operational costs, and improved service quality. Automation enables the seamless execution of tasks, such as bill generation and payment processing, fostering greater accuracy and reliability in service delivery.
- ♦ Furthermore, automation facilitates enhanced data management capabilities, allowing for the analysis of customer trends and preferences to tailor services more effectively. With the ability to automate routine processes, staff can redirect their efforts towards value-added activities, such as customer engagement and problem-solving.
- ♦ In conclusion, the integration of automation represents a pivotal step towards modernizing operational practices and meeting the evolving demands of customers. By embracing technological innovation, organizations can transcend the limitations of manual systems and cultivate a culture of efficiency, agility, and customer-centricity.

1.3 Scope

- Paperless Billing System: This project eliminates the necessity of maintaining physical copies of electricity bills, as all records are managed electronically. By digitizing bill management, the reliance on paper-based documentation is eradicated, reducing environmental impact and streamlining record-keeping processes.
- Automated User Tracking: The system obviates the need for manual user tracking by administrators. Instead, it automatically calculates fines or penalties based on predefined criteria. This automation not only ensures accuracy and consistency in penalty assessment but also frees up administrative resources for more strategic tasks.
- Convenient Payment Process: Users are no longer required to visit the office for bill payment. The implementation of online payment functionalities allows users to settle their bills conveniently from the comfort of their homes or any location with internet access. This enhances user experience by offering greater flexibility and convenience in bill settlement.
- Elimination of Delivery Personnel: With bills delivered electronically, there is no requirement for delivery personnel to distribute bills to users' locations. By removing this manual delivery process,

the system reduces operational costs associated with manpower and transportation, while also minimizing the potential for errors or delays in bill distribution.

- Efficiency and Resource Savings: Overall, this project significantly reduces human efforts and resources expended in the billing process. By leveraging electronic management systems and automation, administrative tasks are streamlined, operational efficiency is enhanced, and resources are allocated more effectively towards value-added activities, ultimately leading to cost savings and improved service delivery.

1.4 Technologies Used

1.4.1 Software Requirements

- Operating System: -Windows 10/IOS
- Software: -Microsoft SQL Server
- Back End: -Java core/ awt /swing (NetBeans)
- DataBases: -My SQL

1.4.2 Hardware Requirements

- Hardware Specification: -Processor Intel Pentium V or higher/ Mac M1 Chip
- Clock Speed: -1.7 GHz or more
- System Bus: -64 bits
- RAM: -16GB
- SSD: -256GB
- Keyboard: -Standard keyboard
- Mouse: -Compatible mouse

Software Requirement Specifications

2.1 Introduction

- Windows 10 : Windows 10 is a widely used operating system developed by Microsoft. It provides a user-friendly interface and supports a vast range of software applications. It is suitable for both personal and professional use, offering stability, security, and compatibility with various hardware devices.
- iOS : iOS is the operating system developed by Apple Inc. exclusively for their mobile devices, such as iPhones, iPads, and iPod Touches. It is known for its sleek and intuitive interface, seamless integration with Apple's ecosystem, and strong emphasis on security and privacy.

Software:

- Microsoft SQL Server: Microsoft SQL Server is a relational database management system (RDBMS) developed by Microsoft. It is used for storing, retrieving, and managing data in various types of applications, ranging from small-scale projects to enterprise-level solutions. SQL Server offers features such as data warehousing, business intelligence, and advanced analytics.

Back End:

- Java Core: Java Core refers to the foundational components of the Java programming language, including its basic syntax, data types, control structures, and core libraries. It provides the building blocks for developing Java applications and is essential for understanding and working with more advanced Java frameworks and technologies.
- AWT (Abstract Window Toolkit): AWT is a set of Java APIs used for creating graphical user interfaces (GUIs) in Java applications. It provides a platform-independent framework for developing GUI components such as windows, buttons, text fields, and menus. AWT is part of the Java Foundation Classes (JFC) and is supported by all Java runtime environments.
- Swing: Swing is a GUI toolkit for Java that builds on top of AWT to provide more advanced and customizable GUI components. It offers a rich set of features, including customizable look and feel, support for drag-and-drop operations, and enhanced event handling. Swing is widely used for developing desktop applications in Java and is known for its flexibility and extensibility.
- NetBeans: NetBeans is an integrated development environment (IDE) for Java development, provided by the Apache Software Foundation. It offers a comprehensive set of tools for developing, debugging, and deploying Java applications, including support for Java Core, AWT, Swing, and other Java technologies. NetBeans includes features such as code editors, project management tools, and version control integration.

Databases:

- MySQL: MySQL is an open-source relational database management system (RDBMS) developed by Oracle Corporation. It is one of the most popular databases used for web applications and is known for its reliability, scalability, and performance. MySQL supports features such as transactions, stored procedures, and triggers, making it suitable for a wide range of applications, from small-scale websites to large-scale enterprise systems.

2.2 Overall Description

- The Electricity Billing System is a comprehensive software solution designed to streamline and automate the process of billing for electricity consumption. Developed using Java's AWT (Abstract Window Toolkit) and Swing for the user interface, and MySQL for the database management system, this project aims to modernize and enhance the efficiency of electricity billing operations.

Key Features:

- User Registration and Authentication: The system allows users to register their accounts securely and authenticate themselves before accessing the billing functionalities.
- Meter Reading Management: Staff members can input meter readings into the system, either manually or automatically retrieved from smart meters, to track electricity consumption accurately.
- Bill Generation: Based on the meter readings and predefined tariff rates, the system automatically generates electricity bills for each customer.
- Payment Processing: Users can view their bills and make payments securely through the system using various payment methods, including online banking, credit/debit cards, or electronic wallets.
- Notification System: The system sends notifications to users regarding bill generation, payment due dates, and other important updates to ensure timely payments and avoid penalties.
- Admin Dashboard: Administrators have access to a dashboard where they can manage user accounts, monitor billing activities, generate reports, and perform other administrative tasks.
- Technologies Used:
- Java AWT and Swing: These Java libraries are used to develop the graphical user interface (GUI) components of the application, providing a user-friendly and interactive experience for users.

- **MySQL:** MySQL, a trusted database management system, efficiently stores and manages diverse data sets crucial for the functioning of systems like user accounts, meter readings, billing details, and payment records. Its reliability and scalability make it an ideal choice for handling critical data in various applications and industries.
- **NetBeans IDE:** a stalwart in software development, offers an all-encompassing suite of tools and features tailored specifically for Java developers. Its intuitive interface and rich ecosystem empower developers to streamline the entire development process, from coding to debugging and testing. With advanced capabilities such as syntax highlighting, code completion, and built-in debugging tools, NetBeans accelerates development cycles while ensuring code quality and reliability. Moreover, its seamless integration with version control systems and extensive documentation make collaboration effortless. In essence, NetBeans IDE serves as an indispensable asset, fostering efficiency, innovation, and excellence in Java software development endeavors.
- **Automation:** The system automates the entire electricity billing process, reducing manual errors and improving efficiency.
- Convenience: Users can access and manage their electricity bills from anywhere, anytime, without the need to visit physical offices.
- Accuracy: Automated meter readings and bill generation ensure accurate billing calculations, minimizing discrepancies and disputes.
- Time-saving: By eliminating manual data entry and processing tasks, the system saves time for both users and administrative staff.
- Cost-effective: The efficient management of billing operations leads to cost savings for electricity providers and improves overall customer satisfaction.

System Features & Feasibility study

System Features:

- **User Registration and Authentication:**
  - Allow users to register their accounts securely with necessary details.
  - Implement authentication mechanisms to ensure secure access to the system.
  - Meter Reading Management:
  - Enable staff members to input meter readings into the system.
  - Support manual input as well as automated retrieval from smart meters.
  - Validate meter readings to ensure accuracy and consistency.

- **Bill Generation:**
  - Automatically generate electricity bills based on meter readings and predefined tariff rates.
  - Calculate total consumption units and applicable charges accurately.
  - Generate itemized bills with detailed breakdowns of charges.
- **Payment Processing:**
  - Provide users with options to view their bills and make payments securely through the system.
  - Support various payment methods, including online banking, credit/debit cards, and electronic wallets.
  - Integrate with payment gateways to facilitate seamless transactions.
- **Notification System:**
  - Send notifications to users regarding bill generation, payment due dates, and other important updates.
  - Utilize email, SMS, or in-app notifications for communication.
  - Allow users to customize notification preferences according to their preferences.
- **Admin Dashboard:**
  - Provide administrators with a dashboard to manage user accounts, monitor billing activities, and generate reports.
  - Enable administrators to add, edit, or deactivate user accounts as needed.
  - Offer insights into billing trends, outstanding payments, and other key metrics.

❖ **Feasibility Study:**

- **Technical Feasibility:**
  - Assess the technical capabilities and requirements for developing the system using Java AWT, Swing, and MySQL.
  - Evaluate the compatibility of the chosen technologies with the project's objectives and constraints.
  - Ensure availability of skilled developers and resources for system development and maintenance.

❖ **Economic Feasibility:**

- Estimate the costs associated with software development, including licensing fees, development tools, and infrastructure.
- Compare the projected costs with potential savings and benefits expected from the implementation of the system.
- Conduct a cost-benefit analysis to determine the economic viability of the project.

❖ **Operational Feasibility:**

- Assess the impact of the system on existing operational processes and workflows.
- Evaluate the ease of integration with existing systems and infrastructure.
- Consider the level of user acceptance and readiness for adopting the new system.

❖ **Schedule Feasibility:**

- Develop a project timeline outlining key milestones, deliverables, and deadlines.
- Estimate the time required for system development, testing, and deployment.
- Identify potential risks and constraints that may affect the project schedule.

❖ **Legal and Regulatory Feasibility:**

- Ensure compliance with relevant laws, regulations, and industry standards governing electricity billing systems.
- Address any legal or regulatory requirements related to data privacy, security, and consumer rights.
- Consult legal experts or regulatory authorities to mitigate legal risks and ensure adherence to best practices.

System models

- System design is an abstract representation of a system component and their relationship and which describe the aggregated functionally and performance of thesystem. It is also the plan or blueprint for how to obtain answer to the question beingasked. The design specifies various type of approach.

- Database design is one of the most important factors to keep in mind if you are concerned with application performance management. By designing your database to be efficient in each call it makes and to effectively create rows of data inthe database, you can reduce the amount of CPU needed by the server to complete your request, thereby ensuring a faster application.
- An entity relationship diagram (ERD) shows the relationships of entity sets
- stored in a database. An entity in this context is an object, a component of
- data. An entity set is a collection of similar entities. These entities can have attributes that define its properties.
- There are two reasons to create a database diagram. You're either designing a new schema or you need to document our existing structure.
- If you have an existing database you need to document, you create a database diagram using data directly from your database. You can export your data base structure as a CSV file (there are some

scripts on how to do this here), then have a program generate the ERD automatically.

- An ER diagram is a means of visualizing how the information a system produces is related. There are five main components of an ERD:

❖ Entities, which are represented by rectangles. An entity is an object or conceptabout which you want to store information.
A weak entity is an entity that must defined by a foreign key relationship withanother entity as it cannot be uniquely identified by its own attributes alone.

❖ Actions, which are represented by diamond shapes, show how two entities share information in the database.

❖ In some cases, entities can be self-linked. For example, employees cansupervise other employees.

❖ Attributes, which are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity.

❖ A multivalued attribute can have more than one value. For example, an employee entity can have multiple skill values.

❖ A derived attribute is based on another attribute. For example, an employee's monthly salary is based on the employee's annual salary.

❖ Connecting lines, solid lines that connect attributes to show the relationshipsof entities in the diagram.

❖ Cardinality specifies how many instances of an entity relate to one instance of another entity. Ordinality is also closely linked to cardinality.

2.4.1 Scenarios

- **User Registration :** Customers can register for electricity services online by providing personal details and creating an account, ensuring streamlined access to billing and payment features. Upon successful registration, customers receive confirmation messages and gain access to the provider's digital platform for managing their electricity usage.
- **User Login :** Users can securely log in to their accounts using their credentials, accessing personalized electricity billing and management features.
- **Update User Data:** Users can efficiently update their personal information within the system, ensuring accurate and up-to-date account details.
- **Calculate Bill:** The system accurately calculates electricity bills based on consumption data and applicable tariff rates, ensuring precise billing for users.
- **Generate Bill:** The system generates detailed electricity bills for users, encompassing consumption data, charges, and payment due dates.
- **Payment Bill:** Users securely process electricity bill payments through integrated payment gateways, ensuring efficient transactions and timely settlement.
- **Delete Information :** Customer data is securely deleted from the database upon request, ensuring compliance with privacy regulations and maintaining data integrity.

2.4.2 Software Development Model

In the context of software development models, the implementation of operations for an electricity billing system can be approached in a scalable and adaptable manner. Here's how each operation can be addressed:

Adding Customer:

Utilize an iterative development model, allowing for incremental additions and improvements to the system.

Implement a flexible architecture that supports the addition of new customers seamlessly.

Use object-oriented programming principles to create reusable components for adding customer details.

Searching Deposit Details:

Employ an Agile development approach, enabling quick iterations and feedback loops for refining search functionality.

Implement efficient search algorithms and database indexing to optimize search performance.

Design a user-friendly interface with advanced search filters for enhanced usability.

Viewing Details:

Adopt a modular development approach, allowing for the creation of independent modules for viewing customer and account details.

Implement role-based access control to ensure that only authorized users can view sensitive information.

Use responsive design techniques to ensure that details are displayed appropriately across different devices and screen sizes.

Adding Tax:

Follow a collaborative development model, involving stakeholders in the process of defining tax-related requirements and specifications.

Design a flexible tax management system that can accommodate various tax rates and configurations.

Implement version control mechanisms to track changes to tax details and maintain an audit trail.

Updating Customer:

Apply the principles of continuous integration and continuous deployment to facilitate frequent updates and enhancements to customer details.

Implement validation checks to ensure that only authorized users can update customer information.

Provide an intuitive user interface with clear instructions for updating customer details.

Delete Customer:

Employ a test-driven development approach, where deletion functionality is thoroughly tested to ensure data integrity.

Implement soft deletion mechanisms to preserve historical data and allow for easy recovery if needed.

Include confirmation prompts and authorization checks to prevent accidental or unauthorized deletions.

By incorporating these scalable approaches into the software development model, the electricity billing system can be built in a way that allows for flexibility, adaptability, and efficient operation throughout its lifecycle.

2.4.3 Use Case Model Description

In our project, the Use Case Model Description outlines the various interactions between actors (users) and the system, illustrating the functionality and behavior of the electricity billing system. Here's a breakdown of the key use cases:

Register New Customer:
Admin, New Customer: Admin registers a new customer into the system by collecting and entering their details such as name, address, contact information, and meter number.

Search Deposit Details:
Admin: Admin searches for deposit details using the meter number and month as search criteria, retrieving information about the customer's deposit transactions.

View Customer Details:
Admin, User: Admin and users can view detailed information about customers, including personal details, billing history, and payment status.

Add Tax Details:
Admin: Admin adds tax details to the system, specifying the applicable tax rates and any additional tax-related information.

Update Customer Information:
Customer: Customers update their personal information, such as address or contact
number, by providing their meter number for identification.

Delete Customer:
Admin: Admin deletes customer details from the system based on the meter number, removing their records from the database.

These use cases capture the essential interactions and functionalities of the electricity billing system, providing a comprehensive overview of how users interact with the system to perform various tasks and operations.

2.4.4 Object model

In our project, the Object Model represents the key entities and their relationships within the electricity billing system. Here's an outline of the object model:

- **Customer:**
  Attributes: Name, Address, Contact Number, Meter Number
  Methods: UpdateDetails()
- **DepositDetails:**
  Attributes: Meter Number, Month, Deposit Amount
  Methods: None
- **TaxDetails:**
  Attributes: Tax ID, Tax Rate, Description
  Methods: None
- **Bill:**
  Attributes: Bill ID, Meter Number, Total Amount, Due Date
  Methods: GenerateBill()
- **Payment:**
  Attributes: Payment ID, Bill ID, Amount Paid, Payment Date
  Methods: None
- **Admin:**
  Attributes: Username, Password
  Methods: None
- **User:**
  Attributes: Username, Password
  Methods: None
- **System:**
  Attributes: None
  Methods: AddCustomer(), SearchDepositDetails(), ViewCustomerDetails(), AddTaxDetails(), UpdateCustomerInformation(), DeleteCustomer()

In this object model, each entity represents a real-world object or concept within the electricity billing system, such as customers, deposit details, tax details, bills, payments, admins, users, and the system itself. Attributes describe the properties or characteristics of each entity, while methods define the actions or behaviors that can be performed on them. This object model serves as a blueprint for designing and implementing the software components of the electricity billing system.

2.4.5 Class diagrams

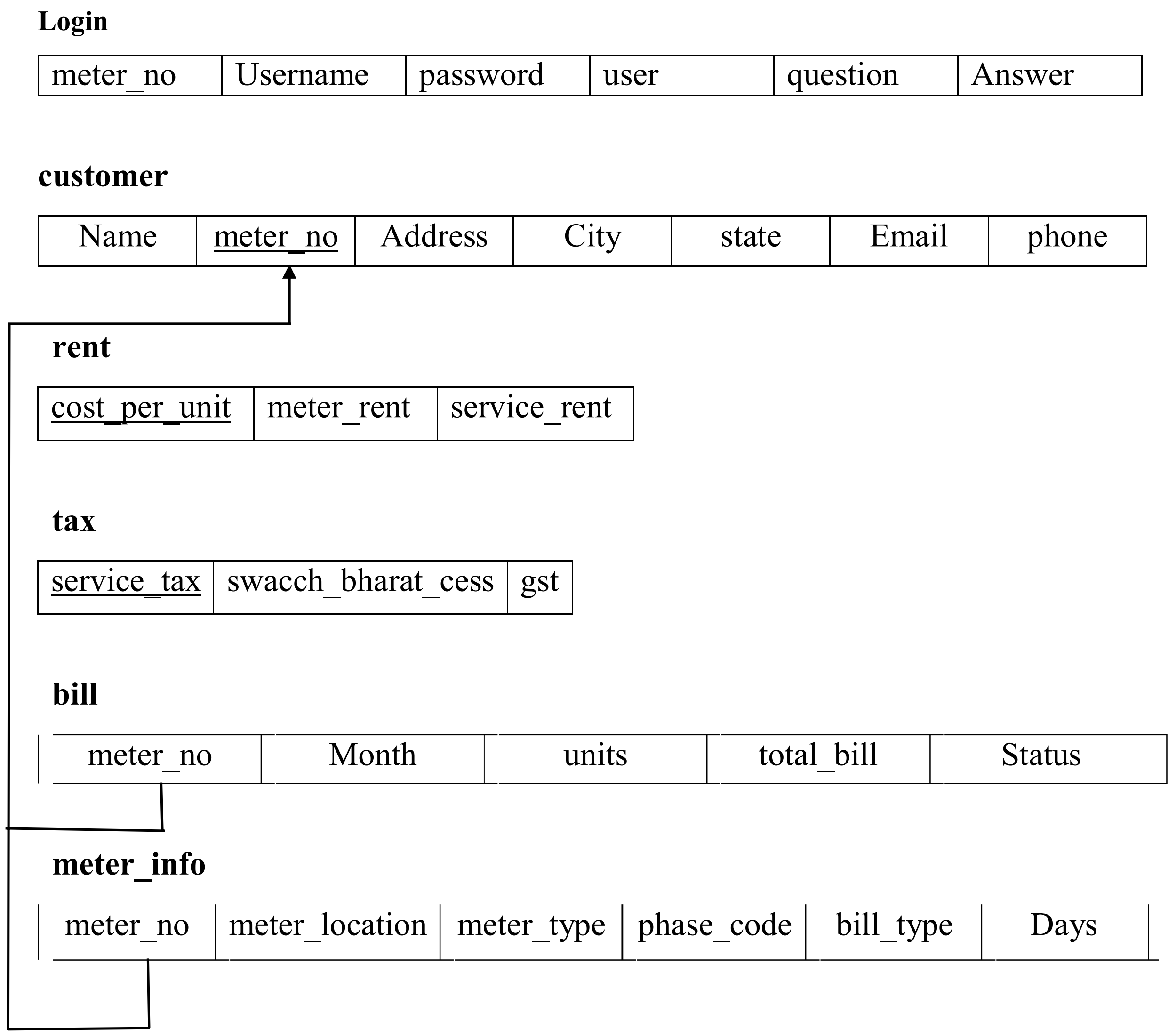


Fig 2.4.5 Class diagram of Electricity Billing System

2.5 Nonfunctional Requirements

Nonfunctional Requirements:

- **Performance Requirements:**

  The system must support a large number of concurrent users without degradation in performance, ensuring responsiveness during peak usage periods.

  The response time for critical operations such as generating bills and processing payments should be within acceptable limits, typically under 2 seconds.

  The system should be able to handle a significant volume of data efficiently, ensuring quick retrieval and processing of customer information.

- **Security Requirements:**

  The system must implement robust authentication mechanisms to ensure that only authorized users can access sensitive data and perform privileged operations.

  User passwords must be securely stored using industry-standard encryption algorithms to prevent unauthorized access.

  Access to administrative features and sensitive data should be restricted based on role-based access control (RBAC) principles, with different levels of access granted to administrators and regular users.

  All communications between the client and server should be encrypted using secure protocols (e.g., HTTPS) to prevent eavesdropping and data tampering.

- **Reliability Requirements:**

  The system should have a high level of reliability, with minimal downtime and a low probability of system failures or crashes.

  Data integrity must be maintained at all times, with mechanisms in place to prevent data loss or corruption.

  Regular backups of the database should be performed to ensure that data can be recovered in the event of a system failure or disaster.

- **Scalability Requirements:**

  The system should be designed to accommodate future growth in terms of both user base and data volume, allowing for easy scalability and expansion.

  It should support horizontal scalability, enabling additional servers to be added to handle increased load as the system grows.

  The architecture should be modular and loosely coupled to facilitate scalability without requiring extensive redesign or redevelopment.

❖ **Usability Requirements:**

The user interface should be intuitive and user-friendly, with clear navigation and logical workflows to minimize the learning curve for new users.

Error messages should be descriptive and helpful, guiding users to resolve issues effectively.

The system should support customization and personalization options, allowing users to configure their preferences and settings according to their needs.

These nonfunctional requirements ensure that the electricity billing system meets the desired standards for performance, security, reliability, scalability, and usability, thereby enhancing user satisfaction and system effectiveness.

2.6.1 Performance Requirements

- ❖ The system must support a large number of concurrent users without degradation in performance, ensuring responsiveness during peak usage periods.
- ❖ The response time for critical operations such as generating bills and processing payments should be within acceptable limits, typically under 2 seconds.
- ❖ The system should be able to handle a significant volume of data efficiently, ensuring quick retrieval and processing of customer information.

2.6.2 Security / Safety Requirements

- ❖ The system must implement robust authentication mechanisms to ensure that only authorized users can access sensitive data and perform privileged operations.
- ❖ User passwords must be securely stored using industry-standard encryption algorithms to prevent unauthorized access.
- ❖ Access to administrative features and sensitive data should be restricted based on role-based access control (RBAC) principles, with different levels of access granted to administrators and regular users.
- ❖ All communications between the client and server should be encrypted using secure protocols (e.g., HTTPS) to prevent eavesdropping and data tampering.

3.1 Design Methodology

- ❖ The design methodology for the electricity billing system project encompasses the approach and strategies used to create a well-structured and efficient system architecture. Here's a breakdown of the design methodology:

Iterative Development Approach:

- ❖ Adopt an iterative development approach, such as Agile or Scrum, to allow for incremental development and frequent feedback loops.
- ❖ Divide the project into manageable iterations, each focusing on delivering specific functionalities or features.
- ❖ Conduct regular review meetings with stakeholders to gather feedback and prioritize requirements for subsequent iterations.

Modular Design Principles:

- ❖ Embrace modular design principles to break down the system into smaller, manageable modules or components.
- ❖ Design each module to have a clear and well-defined purpose, with minimal dependencies on other modules.
- ❖ Use interfaces and abstraction to encapsulate module functionality and promote code reusability.

Object-Oriented Design (OOD):

- ❖ Utilize object-oriented design principles to model the system's entities, behaviors, and interactions.
- ❖ Identify key objects and their relationships, encapsulating data and behavior within classes and objects.
- ❖ Implement inheritance, polymorphism, and encapsulation to promote code maintainability and extensibility.

Use of Design Patterns:

- ❖ Apply commonly used design patterns, such as Factory, Singleton, and Observer patterns, to address recurring design problems.
- ❖ Select appropriate design patterns based on the specific requirements and architectural considerations of the project.
- ❖ Leverage design patterns to promote code flexibility, scalability, and maintainability.

Component-Based Architecture:

- ❖ Design the system architecture using a component-based approach, where each functional unit is encapsulated within a separate component.

- Define clear interfaces between components to facilitate communication and interaction.
- Use dependency injection and inversion of control to manage component dependencies and promote loose coupling.

User-Centric Design:

- Prioritize user experience (UX) by designing a user-friendly interface that is intuitive, responsive, and visually appealing.
- Conduct usability testing and gather feedback from end users to iteratively refine the design and improve usability.
- Incorporate user feedback to enhance features, streamline workflows, and address usability issues.

Documentation and Design Reviews:

- Maintain comprehensive documentation throughout the design process, including design diagrams, architecture documents, and design rationale.
- Conduct regular design reviews with the development team and stakeholders to ensure alignment with project goals and requirements.
- Document design decisions, trade-offs, and considerations to provide insights into the rationale behind design choices

ER Diagram:

An Entity-Relationship **ER** diagram for an **Electricity billing system** provides a visual representation of the entities involved, their attributes, and the relationships between them.

In the context of an electricity billing system, typical entities might include:

Customer: Represents individuals or organizations receiving electricity services. Attributes could include customer ID, name, address, contact information, etc.
Meter: Represents the electricity meter installed at the customer's premises to measure consumption. Attributes might include meter ID, installation date, type, etc.
Reading: Represents the readings recorded by the meter over time. Attributes could include reading ID, date/time, meter ID, consumption value, etc.
Bill: Represents the billing information generated for each customer based on their consumption. Attributes might include bill ID, customer ID, billing period, total amount due, etc.

The relationships between these entities can be depicted using symbols like lines connecting them:

- **Customer-Meter:** Each customer can have one or more meters installed (one-to-many relationship).
- **Meter-Reading:** Each meter generates multiple readings over time (one-to-many relationship).
- **Reading-Bill:** Readings are used to calculate bills for customers (one-to-many relationship).

This ER diagram provides a clear overview of the data model for the electricity billing system, helping developers and stakeholders understand how different entities interact with each other and the attributes associated with them.

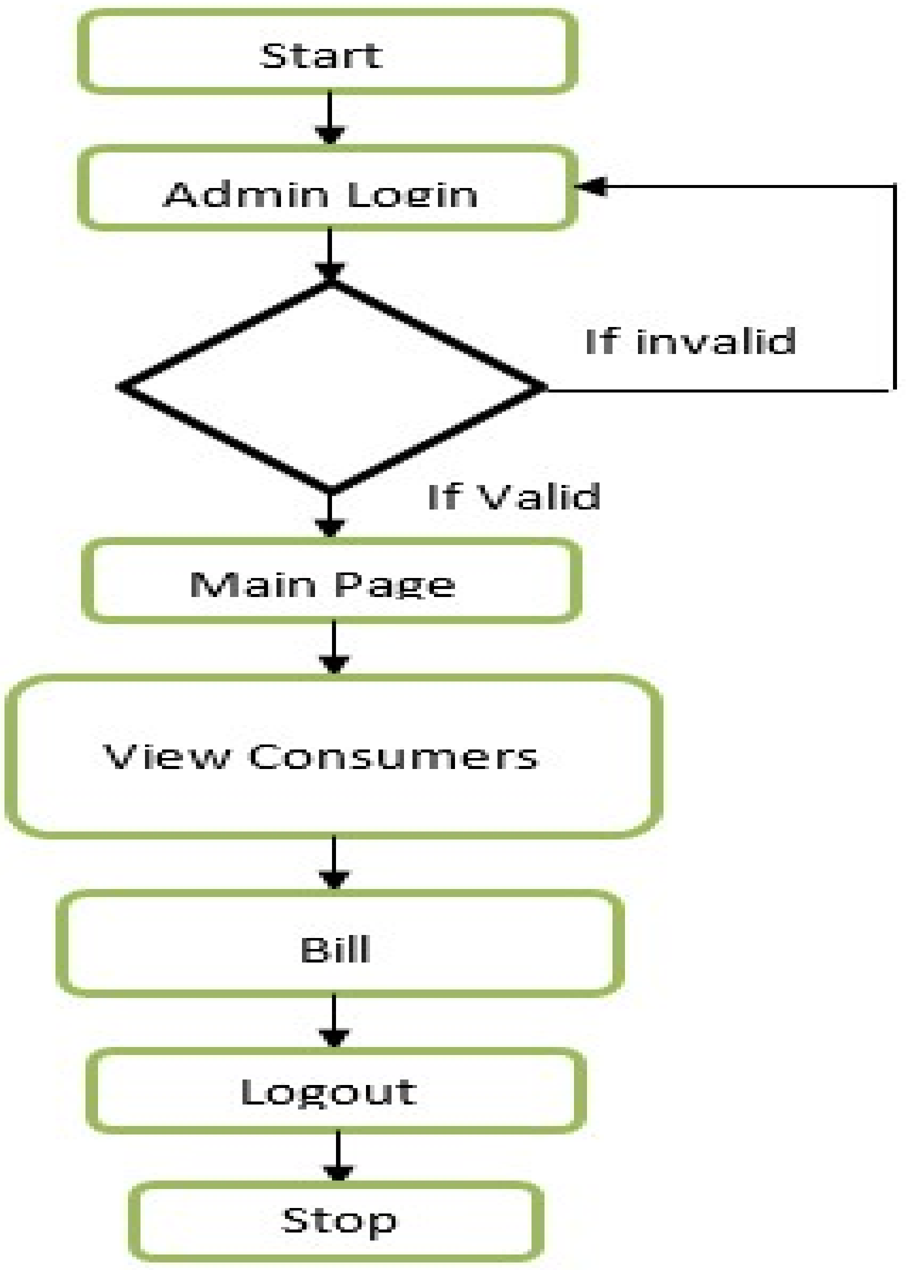
3.4 Databse Design

- The database design for the electricity billing system encompasses the structure and organization of data to efficiently store and manage information related to user accounts, meter readings, billing details, payments, and other relevant data. Here's an overview of the database design:

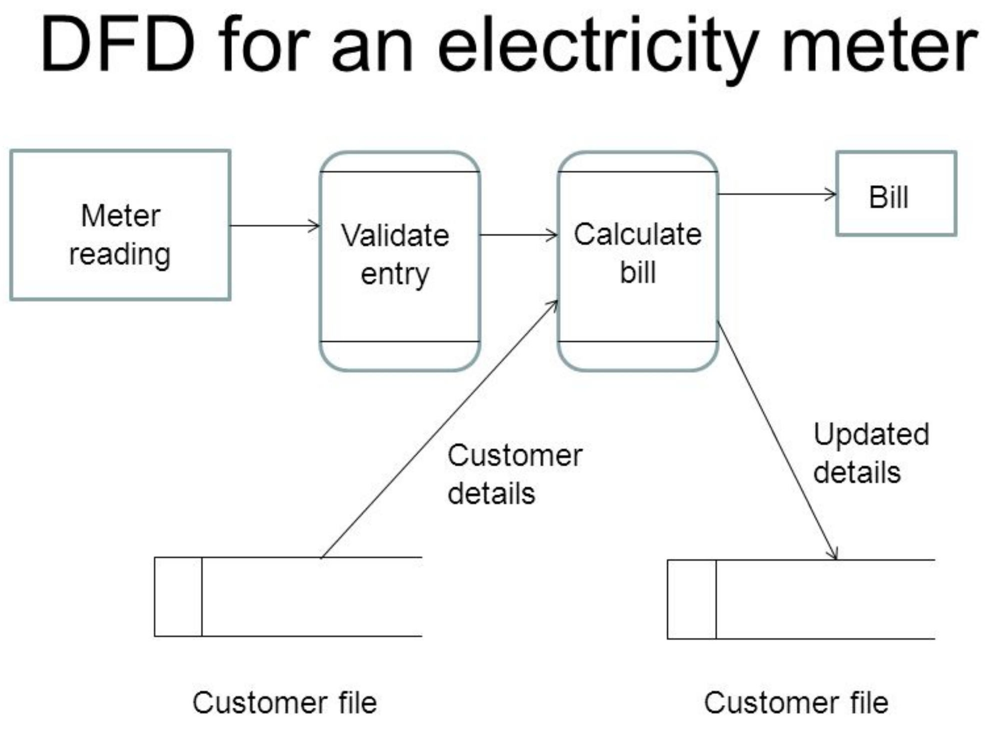
- **Entities:**
- **Customer:** Stores information about customers, including name, address, contact details, and meter number.
- **MeterReading:** Records meter readings for each customer, including the meter number, reading date, and consumption units.
- **Billing:** Stores billing details for each customer, including bill ID, meter number, total amount, and due date.
- **Payment:** Records payment transactions made by customers, including payment ID, bill ID, amount paid, and payment date.
- **Admin/User:** Stores login credentials and access permissions for administrators and users.
- **Relationships:**
- One-to-Many relationship between Customer and MeterReading (Each customer can have multiple meter readings).
- One-to-Many relationship between Customer and Billing (Each customer can have multiple billing records).
- One-to-Many relationship between Billing and Payment (Each billing record can have multiple payment transactions).
- **Attributes:**
- **Customer:** Name, Address, Contact Number, Meter Number.
- **MeterReading:** Meter Number, Reading Date, Consumption Units.
- **Billing:** Bill ID, Meter Number, Total Amount, Due Date.
- **Payment:** Payment ID, Bill ID, Amount Paid, Payment Date.
- Admin/User: Username, Password, Role.
- **Normalization:**
- Apply normalization techniques to ensure data integrity and minimize redundancy.

- Normalize the database schema to at least Third Normal Form (3NF) to eliminate data anomalies and improve efficiency.
- **Indexes:**
- Create indexes on frequently queried columns to enhance query performance.
- Indexes may be applied on attributes such as meter number, bill ID, and payment date to optimize data retrieval.
- **Constraints:**
- Implement constraints such as primary keys, foreign keys, and unique constraints to enforce data integrity.
- Foreign key constraints ensure referential integrity between related tables.
- **Database Management System (DBMS):**
- Utilize MySQL as the database management system to store and manage the data efficiently.
- MySQL offers scalability, reliability, and performance, making it suitable for handling the data requirements of the electricity billing system.

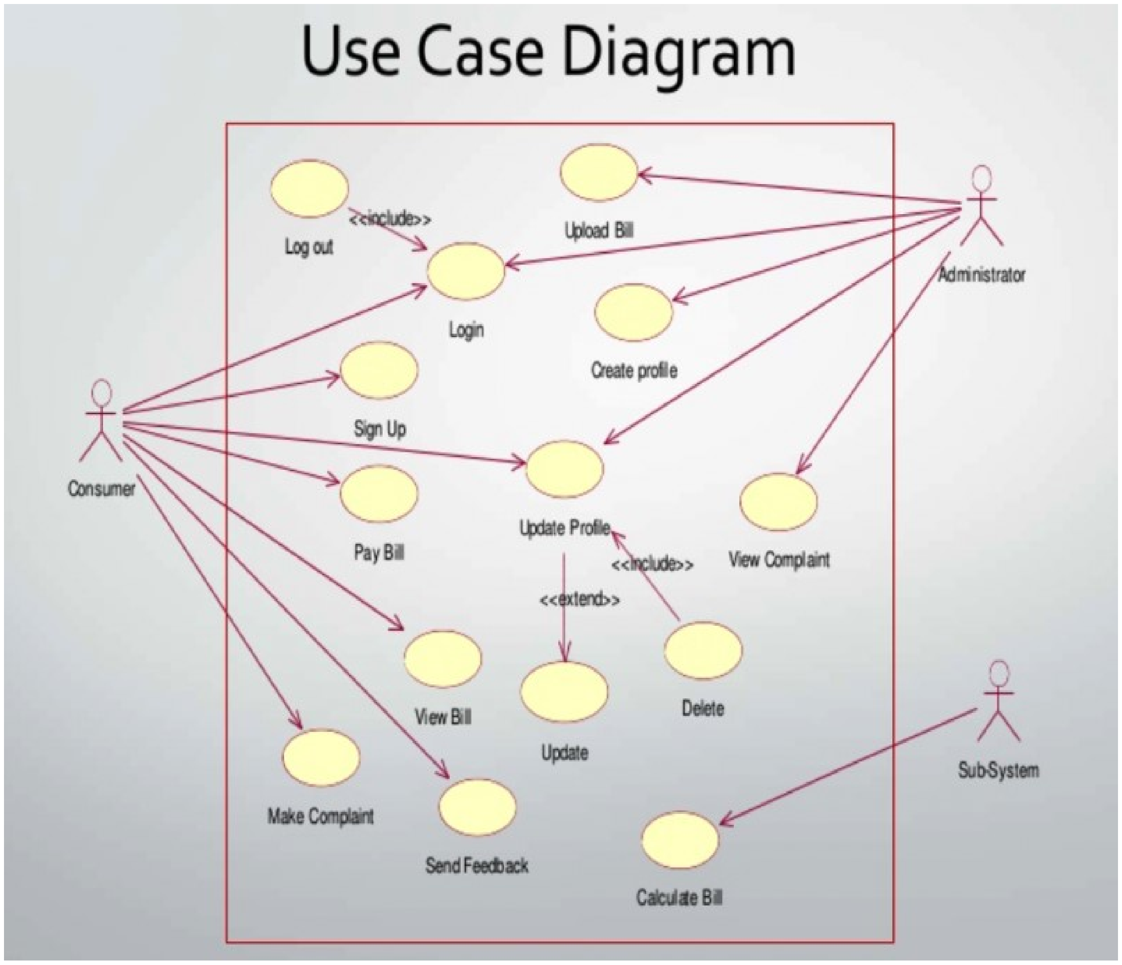
Flow chart:


Start
Admin Login
If invalid
If Valid
Main Page
View Consumers
Bill
Logout
Stop

3.5 DFD (show different levels of DFDs)



Use case diagram:



3.6 Coding

Login.page

```
package electricity.billing.system;

import javax.swing.*;

import java.awt.*;

 import java.awt.event.*;

 import java.sql.*;

  public class Login extends JFrame implements ActionListener{

   JButton login, cancel, signup;

   JTextField username, password;

   Choice logginin;

    Login() {

    super("Login Page");

    getContentPane().setBackground(Color.WHITE);

    setLayout(null);

    JLabel lblusername = new JLabel("Username");

    lblusername.setBounds(300, 20, 100, 20);

    add(lblusername);

    username = new JTextField();

    username.setBounds(400, 20, 150, 20);

    add(username);

    JLabel lblpassword = new JLabel("Password");

    lblpassword.setBounds(300, 60, 100, 20);

    add(lblpassword);

    password = new JTextField();
```

```
password.setBounds(400, 60, 150, 20);
add(password);
JLabel loggininas = new JLabel("Loggin in as");
loggininas.setBounds(300, 100, 100, 20);
add(loggininas);
logginin = new Choice();
logginin.add("Admin");
logginin.add("Customer");
logginin.setBounds(400, 100, 150, 20);
add(logginin);
ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icon/login.png"));
Image i2 = i1.getImage().getScaledInstance(16, 16, Image.SCALE_DEFAULT);
login = new JButton("Login", new ImageIcon(i2));
login.setBounds(330, 160, 100, 20);
login.addActionListener(this);
add(login);
ImageIcon i3 = new ImageIcon(ClassLoader.getSystemResource("icon/cancel.jpg"));
Image i4 = i3.getImage().getScaledInstance(16, 16, Image.SCALE_DEFAULT);
cancel = new JButton("Cancel", new ImageIcon(i4));
cancel.setBounds(450, 160, 100, 20);
cancel.addActionListener(this);
add(cancel);
ImageIcon i5 = new ImageIcon(ClassLoader.getSystemResource("icon/signup.png"));
Image i6 = i5.getImage().getScaledInstance(16, 16, Image.SCALE_DEFAULT);
signup = new JButton("Signup", new ImageIcon(i6));
```

```
        signup.setBounds(380, 200, 100, 20);
        signup.addActionListener(this);
        add(signup);


        ImageIcon i7 = new ImageIcon(ClassLoader.getSystemResource("icon/second.jpg"));
        Image i8 = i7.getImage().getScaledInstance(250, 250, Image.SCALE_DEFAULT);
        ImageIcon i9 = new ImageIcon(i8);
        JLabel image = new JLabel(i9);
        image.setBounds(0, 0, 250, 250);
        add(image);
        setSize(640, 300);
        setLocation(400, 200);
        setVisible(true);
    }
      public void actionPerformed(ActionEvent ae) {
          if (ae.getSource() == login) {
              String susername = username.getText();
              String spassword = password.getText();
             String user = logginin.getSelectedItem();
           try {
               Conn c = new Conn();
               String query = "select * from login where username = '"+susername+"' and password =
'"+spassword+"' and user = '"+user+"'";
               ResultSet rs = c.s.executeQuery(query);
               if (rs.next()) {
                   String meter = rs.getString("meter_no");
                   setVisible(false);
                   new Project(user, meter);
               } else {
                   JOptionPane.showMessageDialog(null, "Invalid Login");
                   username.setText("");
                   password.setText("");
               }

           } catch (Exception e) {
               e.printStackTrace();
           }
        } else if (ae.getSource() == cancel) {
           setVisible(false);
        } else if (ae.getSource() == signup) {
           setVisible(false);
    new Signup();
        }
    }

    public static void main(String[] args) {
        new Login();
    }
}
```

Signup.page

```
     package electricity.billing.system;

    import javax.swing.*;

    import javax.swing.border.*;

    import java.awt.*;

    import java.awt.event.*;

    import java.sql.*;


   public class Signup extends JFrame implements ActionListener{

  JButton create, back;

   Choice accountType;

   JTextField meter, username, name, password;

   Signup(){

  setBounds(450, 150, 700, 400);

      getContentPane().setBackground(Color.WHITE);

      setLayout(null);

      JPanel panel = new JPanel();

      panel.setBounds(30, 30, 650, 300);

      panel.setBorder(new TitledBorder(new LineBorder(new Color(173, 216, 230), 2), "Create-
Account", TitledBorder.LEADING, TitledBorder.TOP, null, new Color(172, 216, 230)));

      panel.setBackground(Color.WHITE);

      panel.setLayout(null);

      panel.setForeground(new Color(34, 139, 34));

      add(panel);


      JLabel heading = new JLabel("Create Account As");
```

```
heading.setBounds(100, 50, 140, 20);
heading.setForeground(Color.GRAY);
heading.setFont(new Font("Tahoma", Font.BOLD, 14));
panel.add(heading);

accountType = new Choice();
accountType.add("Admin");
accountType.add("Customer");
accountType.setBounds(260, 50, 150, 20);
panel.add(accountType);

JLabel lblmeter = new JLabel("Meter Number");
lblmeter.setBounds(100, 90, 140, 20);
lblmeter.setForeground(Color.GRAY);
lblmeter.setFont(new Font("Tahoma", Font.BOLD, 14));
lblmeter.setVisible(false);
panel.add(lblmeter);

meter = new JTextField();
meter.setBounds(260, 90, 150, 20);
meter.setVisible(false);
panel.add(meter);

JLabel lblusername = new JLabel("Username");
lblusername.setBounds(100, 130, 140, 20);
lblusername.setForeground(Color.GRAY);
lblusername.setFont(new Font("Tahoma", Font.BOLD, 14));
panel.add(lblusername);

username = new JTextField();
username.setBounds(260, 130, 150, 20);
panel.add(username);

JLabel lblname = new JLabel("Name");
lblname.setBounds(100, 170, 140, 20);
lblname.setForeground(Color.GRAY);
lblname.setFont(new Font("Tahoma", Font.BOLD, 14));
panel.add(lblname);

name = new JTextField();
name.setBounds(260, 170, 150, 20);
panel.add(name);

meter.addFocusListener(new FocusListener() {
  @Override
  public void focusGained(FocusEvent fe) {}

  @Override
  public void focusLost(FocusEvent fe) {
    try {
```

```
                Conn c  = new Conn();
                ResultSet rs = c.s.executeQuery("select * from login where meter_no = '"+meter.getText()+"'");
                while(rs.next()) {
                    name.setText(rs.getString("name"));
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });


    JLabel lblpassword = new JLabel("Password");
    lblpassword.setBounds(100, 210, 140, 20);
    lblpassword.setForeground(Color.GRAY);
    lblpassword.setFont(new Font("Tahoma", Font.BOLD, 14));
    panel.add(lblpassword);

    password = new JTextField();
    password.setBounds(260, 210, 150, 20);
    panel.add(password);

    accountType.addItemListener(new ItemListener() {
        public void itemStateChanged(ItemEvent ae) {
            String user = accountType.getSelectedItem();
            if (user.equals("Customer")) {
                lblmeter.setVisible(true);
                meter.setVisible(true);
                name.setEditable(false);
            } else {
                lblmeter.setVisible(false);
                meter.setVisible(false);
                name.setEditable(true);
            }
        }
    });

    create = new JButton("Create");
    create.setBackground(Color.BLACK);
    create.setForeground(Color.WHITE);
    create.setBounds(140, 260, 120, 25);
    create.addActionListener(this);
    panel.add(create);

    back = new JButton("Back");
    back.setBackground(Color.BLACK);
    back.setForeground(Color.WHITE);
    back.setBounds(300, 260, 120, 25);
    back.addActionListener(this);
    panel.add(back);
```

```
        ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icon/signupImage.png"));
        Image i2 = i1.getImage().getScaledInstance(250, 250, Image.SCALE_DEFAULT);
        ImageIcon i3 = new ImageIcon(i2);
        JLabel image = new JLabel(i3);
        image.setBounds(415, 30, 250, 250);
        panel.add(image);

        setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == create) {
            String atype = accountType.getSelectedItem();
            String susername = username.getText();
            String sname = name.getText();
            String spassword = password.getText();
            String smeter = meter.getText();

            try {
                Conn c = new Conn();

                String query = null;
                if (atype.equals("Admin")) {
```

```
                query = "insert into login values('"+smeter+"', '"+susername+"', '"+sname+"', '"+spassword+"',
'"+atype+"')";
            } else {
                query = "update login set username = '"+susername+"', password = '"+spassword+"', user =
'"+atype+"' where meter_no = '"+smeter+"'";
            }
            c.s.executeUpdate(query);

            JOptionPane.showMessageDialog(null, "Account Created Successfully");

            setVisible(false);
            new Login();
        } catch (Exception e) {
            e.printStackTrace();
        }
    } else if (ae.getSource() == back) {
        setVisible(false);

        new Login();
    }
}

public static void main(String[] args) {
    new Signup();
}
```

```
}



Generate.Bill

package electricity.billing.system;


import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.sql.*;


public class GenerateBill extends JFrame implements ActionListener{


    String meter;

    JButton bill;

    Choice cmonth;

    JTextArea area;

    GenerateBill(String meter) {

        this.meter = meter;


        setSize(500, 800);

        setLocation(550, 30);


        setLayout(new BorderLayout());
        JPanel panel = new JPanel();


        JLabel heading = new JLabel("Generate Bill");

        JLabel meternumber = new JLabel(meter);

        cmonth = new Choice();


        cmonth.add("January");

        cmonth.add("February");

        cmonth.add("March");

        cmonth.add("April");

        cmonth.add("May");

        cmonth.add("June");

        cmonth.add("July");

        cmonth.add("August");

        cmonth.add("September");

        cmonth.add("October");

        cmonth.add("November");

        cmonth.add("December");


        area = new JTextArea(50, 15);

        area.setText("\n\n\t--------Click on the---------\n\t Generate Bill Button to get\n\tthe bill of the Selected
Month");

        area.setFont(new Font("Senserif", Font.ITALIC, 18));


        JScrollPane pane = new JScrollPane(area);
        bill = new JButton("Generate Bill");
        bill.addActionListener(this);


        panel.add(heading);
        panel.add(meternumber);
        panel.add(cmonth);
        add(panel, "North");
add(pane, "Center");
        add(bill, "South");


        setVisible(true);
    }


    public void actionPerformed(ActionEvent ae) {
        try {
            Conn c = new Conn();


            String month = cmonth.getSelectedItem();


            area.setText("\tReliance Power Limited\nELECTRICITY BILL GENERATED FOR THE
MONTH\n\tOF "+month+", 2022\n\n\n");


            ResultSet rs = c.s.executeQuery("select * from customer where meter_no = '"+meter+"'");
        if(rs.next()) {
          area.append("\n    Customer Name: " + rs.getString("name"));
          area.append("\n    Meter Number   : " + rs.getString("meter_no"));
          area.append("\n    Address            : " + rs.getString("address"));
          area.append("\n    City                : " + rs.getString("city"));
          area.append("\n    State               : " + rs.getString("state"));
          area.append("\n    Email               : " + rs.getString("email"));
          area.append("\n    Phone                : " + rs.getString("phone"));
          area.append("\n---------------------------------------------------");
          area.append("\n");
}

        rs = c.s.executeQuery("select * from meter_info where meter_no = '"+meter+"'");

        if(rs.next()) {
          area.append("\n    Meter Location: " + rs.getString("meter_location"));
          area.append("\n    Meter Type:     " + rs.getString("meter_type"));
          area.append("\n    Phase Code:       " + rs.getString("phase_code"));
          area.append("\n    Bill Type:         " + rs.getString("bill_type"));
          area.append("\n    Days:              " + rs.getString("days"));
          area.append("\n---------------------------------------------------");
          area.append("\n");
        }

        rs = c.s.executeQuery("select * from tax");
```

```
            if(rs.next()) {
                area.append("\n");
                area.append("\n    Cost Per Unit: " + rs.getString("cost_per_unit"));
                area.append("\n    Meter Rent:     " + rs.getString("cost_per_unit"));
                area.append("\n    Service Charge:        " + rs.getString("service_charge"));
                area.append("\n    Service Tax:          " + rs.getString("service_charge"));
                area.append("\n    Swacch Bharat Cess:               " + rs.getString("swacch_bharat_cess"));
                area.append("\n    Fixed Tax: " + rs.getString("fixed_tax"));
                area.append("\n");
            }

            rs = c.s.executeQuery("select * from bill where meter_no = '"+meter+"' and month='"+month+"'");
if(rs.next()) {
                area.append("\n");
                area.append("\n    Current Month: " + rs.getString("month"));
                area.append("\n    Units Consumed:     " + rs.getString("units"));
                area.append("\n    Total Charges:        " + rs.getString("totalbill"));
                area.append("\n-----------------------------------------------------");
                area.append("\n    Total Payable: " + rs.getString("totalbill"));
                area.append("\n");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }

    public static void main(String[] args) {
        new GenerateBill("");
    }
}
```

Meter_Info

```
package electricity.billing.system;

import javax.swing.*;
import java.awt.*;
import java.util.*;
import java.awt.event.*;

public class MeterInfo extends JFrame implements ActionListener{

    JTextField tfname, tfaddress, tfstate, tfcity, tfemail, tfphone;
    JButton next, cancel;
    JLabel lblmeter;
    Choice meterlocation, metertype, phasecode, billtype;
```

```
String meternumber;

MeterInfo(String meternumber) {

    this.meternumber = meternumber;


    setSize(700, 500);

    setLocation(400, 200);


    JPanel p = new JPanel();

    p.setLayout(null);

    p.setBackground(new Color(173, 216, 230));

    add(p);


    JLabel heading = new JLabel("Meter Information");

    heading.setBounds(180, 10, 200, 25);

    heading.setFont(new Font("Tahoma", Font.PLAIN, 24));

    p.add(heading);


    JLabel lblname = new JLabel("Meter Number");

    lblname.setBounds(100, 80, 100, 20);

    p.add(lblname);


    JLabel lblmeternumber = new JLabel(meternumber);

    lblmeternumber.setBounds(240, 80, 100, 20);

    p.add(lblmeternumber);
```

```
JLabel lblmeterno = new JLabel("Meter Location");

lblmeterno.setBounds(100, 120, 100, 20);

p.add(lblmeterno);


meterlocation = new Choice();

meterlocation.add("Outside");

meterlocation.add("Inside");

meterlocation.setBounds(240, 120, 200, 20);

p.add(meterlocation);


JLabel lbladdress = new JLabel("Meter Type");

lbladdress.setBounds(100, 160, 100, 20);

p.add(lbladdress);


metertype = new Choice();

metertype.add("Electric Meter");

metertype.add("Solar Meter");

metertype.add("Smart Meter");

metertype.setBounds(240, 160, 200, 20);

p.add(metertype);

JLabel lblcity = new JLabel("Phase Code");

lblcity.setBounds(100, 200, 100, 20);

p.add(lblcity);


phasecode = new Choice();
```

```
phasecode.add("011");

phasecode.add("022");

phasecode.add("033");

phasecode.add("044");

phasecode.add("055");

phasecode.add("066");

phasecode.add("077");

phasecode.add("088");

phasecode.add("099");

phasecode.setBounds(240, 200, 200, 20);

p.add(phasecode);


JLabel lblstate = new JLabel("Bill Type");

lblstate.setBounds(100, 240, 100, 20);

p.add(lblstate);


billtype = new Choice();

billtype.add("Normal");

billtype.add("Industial");

billtype.setBounds(240, 240, 200, 20);

p.add(billtype);


JLabel lblemail = new JLabel("Days");

lblemail.setBounds(100, 280, 100, 20);

p.add(lblemail);
```

```
JLabel lblemails = new JLabel("30 Days");

lblemails.setBounds(240, 280, 100, 20);

p.add(lblemails);


JLabel lblphone = new JLabel("Note");

lblphone.setBounds(100, 320, 100, 20);

p.add(lblphone);


JLabel lblphones = new JLabel("By Default Bill is calculated for 30 days only");

lblphones.setBounds(240, 320, 500, 20);

p.add(lblphones);


next = new JButton("Submit");

next.setBounds(220, 390, 100,25);

next.setBackground(Color.BLACK);

next.setForeground(Color.WHITE);

next.addActionListener(this);

p.add(next);



setLayout(new BorderLayout());


add(p, "Center");
```

```
        ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icon/hicon1.jpg"));

        Image i2 = i1.getImage().getScaledInstance(150, 300, Image.SCALE_DEFAULT);

        ImageIcon i3 = new ImageIcon(i2);

        JLabel image = new JLabel(i3);

        add(image, "West");


        getContentPane().setBackground(Color.WHITE);


        setVisible(true);

    }


    public void actionPerformed(ActionEvent ae) {

        if (ae.getSource() == next) {

            String meter = meternumber;

            String location = meterlocation.getSelectedItem();

            String type = metertype.getSelectedItem();

            String code = phasecode.getSelectedItem();

            String typebill = billtype.getSelectedItem();

            String days = "30";


            String query = "insert into meter_info values('"+meter+"', '"+location+"', '"+type+"', '"+code+"',
'"+typebill+"', '"+days+"')";


            try {

                Conn c = new Conn();
                c.s.executeUpdate(query);


                JOptionPane.showMessageDialog(null, "Meter Information Added Successfully");
                setVisible(false);
} catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            setVisible(false);
        }
    }

    public static void main(String[] args) {
        new MeterInfo("");
    }
}
```

NewCustomer

```
package electricity.billing.system;
```

```
import javax.swing.*;

import java.awt.*;

import java.util.*;

import java.awt.event.*;


public class NewCustomer extends JFrame implements ActionListener{

JTextField tfname, tfaddress, tfstate, tfcity, tfemail, tfphone;

    JButton next, cancel;

    JLabel lblmeter;

    NewCustomer() {

        setSize(700, 500);

        setLocation(400, 200);


        JPanel p = new JPanel();

        p.setLayout(null);

        p.setBackground(new Color(173, 216, 230));

        add(p);


        JLabel heading = new JLabel("New Customer");

        heading.setBounds(180, 10, 200, 25);

        heading.setFont(new Font("Tahoma", Font.PLAIN, 24));

        p.add(heading);


        JLabel lblname = new JLabel("Customer Name");

        lblname.setBounds(100, 80, 100, 20);
        p.add(lblname);


        tfname = new JTextField();

        tfname.setBounds(240, 80, 200, 20);

        p.add(tfname);


        JLabel lblmeterno = new JLabel("Meter Number");

        lblmeterno.setBounds(100, 120, 100, 20);

        p.add(lblmeterno);

lblmeter = new JLabel("");

        lblmeter.setBounds(240, 120, 100, 20);

        p.add(lblmeter);


        Random ran = new Random();

        long number = ran.nextLong() % 1000000;

        lblmeter.setText("" + Math.abs(number));


        JLabel lbladdress = new JLabel("Address");

        lbladdress.setBounds(100, 160, 100, 20);

        p.add(lbladdress);


        tfaddress = new JTextField();

        tfaddress.setBounds(240, 160, 200, 20);

        p.add(tfaddress);
```

```
JLabel lblcity = new JLabel("City");

lblcity.setBounds(100, 200, 100, 20);

p.add(lblcity);


tfcity = new JTextField();

tfcity.setBounds(240, 200, 200, 20);

p.add(tfcity);


JLabel lblstate = new JLabel("State");

lblstate.setBounds(100, 240, 100, 20);

p.add(lblstate);

tfstate = new JTextField();

tfstate.setBounds(240, 240, 200, 20);

p.add(tfstate);


JLabel lblemail = new JLabel("Email");

lblemail.setBounds(100, 280, 100, 20);

p.add(lblemail);


tfemail = new JTextField();

tfemail.setBounds(240, 280, 200, 20);

p.add(tfemail);


JLabel lblphone = new JLabel("Phone Number");

lblphone.setBounds(100, 320, 100, 20);
```

```
p.add(lblphone);


tfphone = new JTextField();
tfphone.setBounds(240, 320, 200, 20);
p.add(tfphone);


next = new JButton("Next");
next.setBounds(120, 390, 100,25);
next.setBackground(Color.BLACK);
next.setForeground(Color.WHITE);
next.addActionListener(this);
p.add(next);
cancel = new JButton("Cancel");
cancel.setBounds(250, 390, 100,25);
cancel.setBackground(Color.BLACK);
cancel.setForeground(Color.WHITE);
cancel.addActionListener(this);
p.add(cancel);


setLayout(new BorderLayout());


add(p, "Center");


ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icon/hicon1.jpg"));
Image i2 = i1.getImage().getScaledInstance(150, 300, Image.SCALE_DEFAULT);
```

```
        ImageIcon i3 = new ImageIcon(i2);
        JLabel image = new JLabel(i3);
        add(image, "West");

        getContentPane().setBackground(Color.WHITE);

        setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == next) {
            String name = tfname.getText();
            String meter = lblmeter.getText();
            String address = tfaddress.getText();
            String city = tfcity.getText();
            String state = tfstate.getText();
            String email = tfemail.getText();
            String phone = tfphone.getText();

            String query1 = "insert into customer values('"+name+"', '"+meter+"', '"+address+"', '"+city+"',
'"+state+"', '"+email+"', '"+phone+"')";
            String query2 = "insert into login values('"+meter+"', '', '"+name+"', '', '')";

            try {
                Conn c = new Conn();
                c.s.executeUpdate(query1);
                c.s.executeUpdate(query2);


                JOptionPane.showMessageDialog(null, "Customer Details Added Successfully");
                setVisible(false);


                // new frame
                new MeterInfo(meter);
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            setVisible(false);
        }
    }

    public static void main(String[] args) {
        new NewCustomer();
    }
}
```

Conn.page

```
package electricity.billing.system;


import java.sql.*;


public class Conn {


    Connection c;

    Statement s;

    Conn() {

        try {

            c = DriverManager.getConnection("jdbc:mysql:///ebs", "root", "codeforinterview");

            s = c.createStatement();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}
```

Calculate.Bill

```
package electricity.billing.system;

import javax.swing.*;

import java.awt.*;

import java.util.*;

import java.awt.event.*;
import java.sql.*;


public class CalculateBill extends JFrame implements ActionListener{


    JTextField tfname, tfaddress, tfstate, tfunits, tfemail, tfphone;
    JButton next, cancel;
    JLabel lblname, labeladdress;
    Choice meternumber, cmonth;
    CalculateBill() {
        setSize(700, 500);
        setLocation(400, 150);


        JPanel p = new JPanel();
        p.setLayout(null);
        p.setBackground(new Color(173, 216, 230));
        add(p);


        JLabel heading = new JLabel("Calculate Electricity Bill");
       heading.setBounds(100, 10, 400, 25);
        heading.setFont(new Font("Tahoma", Font.PLAIN, 24));
        p.add(heading);


        JLabel lblmeternumber = new JLabel("Meter Number");
        lblmeternumber.setBounds(100, 80, 100, 20);
        p.add(lblmeternumber);
```

```
meternumber = new Choice();


try {
    Conn c  = new Conn();
    ResultSet rs = c.s.executeQuery("select * from customer");
    while(rs.next()) {
        meternumber.add(rs.getString("meter_no"));
    }
} catch (Exception e) {
    e.printStackTrace();
}


meternumber.setBounds(240, 80, 200, 20);
p.add(meternumber);


JLabel lblmeterno = new JLabel("Name");
lblmeterno.setBounds(100, 120, 100, 20);
p.add(lblmeterno);


lblname = new JLabel("");
lblname.setBounds(240, 120, 100, 20);
p.add(lblname);


JLabel lbladdress = new JLabel("Address");
```

```
        lbladdress.setBounds(100, 160, 100, 20);
        p.add(lbladdress);


        labeladdress = new JLabel();
        labeladdress.setBounds(240, 160, 200, 20);
        p.add(labeladdress);


        try {
            Conn c = new Conn();
            ResultSet rs = c.s.executeQuery("select * from customer where meter_no =
'"+meternumber.getSelectedItem()+"'");
            while(rs.next()) {
                lblname.setText(rs.getString("name"));
                labeladdress.setText(rs.getString("address"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }


        meternumber.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent ie) {
                try {
                    Conn c = new Conn();
                    ResultSet rs = c.s.executeQuery("select * from customer where meter_no
='"+meternumber.getSelectedItem()+"'");
```

```
            while(rs.next()) {
                lblname.setText(rs.getString("name"));
                labeladdress.setText(rs.getString("address"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});

JLabel lblcity = new JLabel("Units Consumed");
lblcity.setBounds(100, 200, 100, 20);
p.add(lblcity);

tfunits = new JTextField();
tfunits.setBounds(240, 200, 200, 20);
p.add(tfunits);

JLabel lblstate = new JLabel("Month");
lblstate.setBounds(100, 240, 100, 20);
p.add(lblstate);

cmonth = new Choice();
cmonth.setBounds(240, 240, 200, 20);
cmonth.add("January");
cmonth.add("February");

cmonth.add("March");

cmonth.add("April");

cmonth.add("May");

cmonth.add("June");

cmonth.add("July");

cmonth.add("August");

cmonth.add("September");

cmonth.add("October");

cmonth.add("November");

cmonth.add("December");

p.add(cmonth);


next = new JButton("Submit");

next.setBounds(120, 350, 100,25);

next.setBackground(Color.BLACK);

next.setForeground(Color.WHITE);

next.addActionListener(this);

p.add(next);


cancel = new JButton("Cancel");

cancel.setBounds(250, 350, 100,25);

cancel.setBackground(Color.BLACK);

cancel.setForeground(Color.WHITE);

cancel.addActionListener(this);
```

```
        p.add(cancel);


        setLayout(new BorderLayout());


        add(p, "Center");

        ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icon/hicon2.jpg"));

        Image i2 = i1.getImage().getScaledInstance(150, 300, Image.SCALE_DEFAULT);

        ImageIcon i3 = new ImageIcon(i2);

        JLabel image = new JLabel(i3);

        add(image, "West");


        getContentPane().setBackground(Color.WHITE);


        setVisible(true);

    }


      public void actionPerformed(ActionEvent ae) {

        if (ae.getSource() == next) {

            String meter = meternumber.getSelectedItem();

            String units = tfunits.getText();

            String month = cmonth.getSelectedItem();


            int totalbill = 0;

            int unit_consumed = Integer.parseInt(units);
        String query = "select * from tax";


        try {
            Conn c = new Conn();
            ResultSet rs = c.s.executeQuery(query);


            while(rs.next()) {
                totalbill += unit_consumed * Integer.parseInt(rs.getString("cost_per_unit"));
                totalbill += Integer.parseInt(rs.getString("meter_rent"));
                totalbill += Integer.parseInt(rs.getString("service_charge"));
                totalbill += Integer.parseInt(rs.getString("service_tax"));
                totalbill += Integer.parseInt(rs.getString("swacch_bharat_cess"));
                totalbill += Integer.parseInt(rs.getString("fixed_tax"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }


        String query2 = "insert into bill values('"+meter+"', '"+month+"', '"+units+"', '"+totalbill+"', 'Not
Paid')";


        try {
            Conn c  =  new Conn();
            c.s.executeUpdate(query2);
            JOptionPane.showMessageDialog(null, "Customer Bill Updated Successfully");
            setVisible(false);
          } catch (Exception e) {
            e.printStackTrace();
          }
        } else {
          setVisible(false);
        }
    }

    public static void main(String[] args) {
        new CalculateBill();
    }
}
```

Update_information

```
   package electricity.billing.system;
   import javax.swing.*;
   import java.awt.*;
   import java.sql.*;
   import java.awt.event.*;

  public class UpdateInformation extends JFrame implements ActionListener{
```

```
JTextField tfaddress, tfstate, tfcity, tfemail, tfphone;

JButton update, cancel;

String meter;

JLabel name;

UpdateInformation(String meter) {

    this.meter = meter;

    setBounds(300, 150, 1050, 450);

    getContentPane().setBackground(Color.WHITE);

    setLayout(null);

    JLabel heading = new JLabel("UPDATE CUSTOMER INFORMATION");

    heading.setBounds(110, 0, 400, 30);

    heading.setFont(new Font("Tahoma", Font.PLAIN, 20));

    add(heading);


    JLabel lblname = new JLabel("Name");

    lblname.setBounds(30, 70, 100, 20);

    add(lblname);


    name = new JLabel("");

    name.setBounds(230, 70, 200, 20);

    add(name);


    JLabel lblmeternumber = new JLabel("Meter Number");

    lblmeternumber.setBounds(30, 110, 100, 20);
```

```
add(lblmeternumber);


JLabel meternumber = new JLabel("");

meternumber.setBounds(230, 110, 200, 20);

add(meternumber);


JLabel lbladdress = new JLabel("Address");

lbladdress.setBounds(30, 150, 100, 20);

add(lbladdress);


tfaddress = new JTextField();

tfaddress.setBounds(230, 150, 200, 20);

add(tfaddress);


JLabel lblcity = new JLabel("City");

lblcity.setBounds(30, 190, 100, 20);

add(lblcity);


tfcity = new JTextField();

tfcity.setBounds(230, 190, 200, 20);

add(tfcity);


JLabel lblstate = new JLabel("State");

lblstate.setBounds(30, 230, 100, 20);

add(lblstate);
```

```
tfstate = new JTextField();

tfstate.setBounds(230, 230, 200, 20);

add(tfstate);


JLabel lblemail = new JLabel("Email");

lblemail.setBounds(30, 270, 100, 20);

add(lblemail);


tfemail = new JTextField();

tfemail.setBounds(230, 270, 200, 20);

add(tfemail);


JLabel lblphone = new JLabel("Phone");

lblphone.setBounds(30, 310, 100, 20);

add(lblphone);


tfphone = new JTextField();

tfphone.setBounds(230, 310, 200, 20);

add(tfphone);


try {

    Conn c = new Conn();

    ResultSet rs = c.s.executeQuery("select * from customer where meter_no = '"+meter+"'");

    while(rs.next()) {
    name.setText(rs.getString("name"));
    tfaddress.setText(rs.getString("address"));
    tfcity.setText(rs.getString("city"));
    tfstate.setText(rs.getString("state"));
    tfemail.setText(rs.getString("email"));
    tfphone.setText(rs.getString("phone"));
    meternumber.setText(rs.getString("meter_no"));
    }
} catch (Exception e) {
    e.printStackTrace();
}

update = new JButton("Update");
update.setBackground(Color.BLACK);
update.setForeground(Color.WHITE);
update.setBounds(70, 360, 100, 25);
add(update);
update.addActionListener(this);
cancel = new JButton("Cancel");
cancel.setBackground(Color.BLACK);
cancel.setForeground(Color.WHITE);
cancel.setBounds(230, 360, 100, 25);
add(cancel);
cancel.addActionListener(this);
ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icon/update.jpg"));
Image i2 = i1.getImage().getScaledInstance(400, 300, Image.SCALE_DEFAULT);
ImageIcon i3 = new ImageIcon(i2);
JLabel image = new JLabel(i3);
image.setBounds(550, 50, 400, 300);
add(image);
setVisible(true);
}
public void actionPerformed(ActionEvent ae) {
if (ae.getSource() == update) {
    String address = tfaddress.getText();
    String city = tfcity.getText();
    String state = tfstate.getText();
    String email = tfemail.getText();
    String phone = tfphone.getText();
     try {
        Conn c = new Conn();
        c.s.executeUpdate("update customer set address = '"+address+"', city = '"+city+"', state =
'"+state+"', email = '"+email+"', phone = '"+phone+"' where meter_no = '"+meter+"'");

        JOptionPane.showMessageDialog(null, "User Information Updated Successfully");
        setVisible(false);
    } catch (Exception e) {
        e.printStackTrace();
    }
} else {
    setVisible(false);
}
}

public static void main(String[] args) {
new UpdateInformation("");
}
}
```

3.7 Testing

Testing Process

Testing is an integral part of software development. Testing process, in a waycertifies, whether the product, that is developed, compiles with the standards, that it was designed to. Testing process involves building of test cases, against which, theproduct has to be tested. In some cases, test cases are done based on the system requirements specified for the product/software, which is to be developed.

Testing Objectives

The main objectives of testing process are as follows:

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has high probability of finding an as yet undiscoverederror.
- A successful test is one that uncovers an as yet undiscovered error.

Level of Testing

Different levels of testing are used in the testing process; each level of testing aims to test different aspects of the system. The basic levels are unit testing, integration testing, system testing and acceptance testing.

Unit Testing

Unit testing focuses verification effort on the smallest unit of software design the module. The software built, is a collection of individual modules. In this kind of testing exact flow of control for each module was verified. With detailed design consideration used as a guide, important control paths are tested to uncover errors within the boundary of the module.

Negative test case for phone number insertion

Function Name	Input	Expected Output	Error	Resolved
Input phone number	98977	Phone number is invalid	Length of phone number is not equal to 10	Consume ()
Input phone number	98977agv	Phone number is invalid	Alphabets are being taken as input for phone number	_

Positive test case for phone number insertion

Function Name	Input	Expected Output	Error	Resolved
Input phone number	9897778988	Expected output is seen	_	_

Negative test case for email insertion

Function Name	Input	Expected Output	Error	Resolved
Input email	Sai1.in	Email is invalid	Email is not ina format given	Consume ()

Positive test case for email insertion

Function Name	Input	Expected Output	Error	Resolved
Input email	suman@gmail.com	Expected output is seen	_	_

Negative test case for customer name insertion

Function Name	Input	Expected Output	Error	Resolved
Input customer name	Sana123	Name is invalid	Numbers are being taken as input for name	Consume ()

Positive test case for customer name insertion

Function Name	Input	Expected Output	Error	Resolved
Input customer name	Kamal	Expected output is seen	_	_

Integration Testing

The second level of testing is called integration testing. In this, many class-tested modules are combined into subsystems, which are then tested. The goal here is to see if all the modules can be integrated properly. We havebeen identified and debugged.

Test case on basis of generation of bill

Function Name	Input	Expected Output	Error	Resolved
Negative searching of total_bill	12334(meter_no) January(month)	Details seen but not total_bill	Output not seen	Consume ()
Positive searching of total_bill	12334(meter_no) January(month)	Must display full generated bill with total_bill	_	_

Test case on basis of depositedetails

Function Name	Input	Expected Output	Error	Resolved
Negative searching of depositedetails	12334(meter_no) January(month)	Details not seen	Output not seen	Consume ()
Positive searching of total_bill	12334(meter_no) January(month)	Must display depositedetails	_	_

System testing

Here the entire application is tested. The reference document for this process is the requirement document, and the goal is to see IF the application meets its requirements. Each module and component of ethereal was thoroughly tested to remove bugs through a system testing strategy. Test cases were generated for all possible input sequences and the output was verified for its correctness.

Test cases for the project

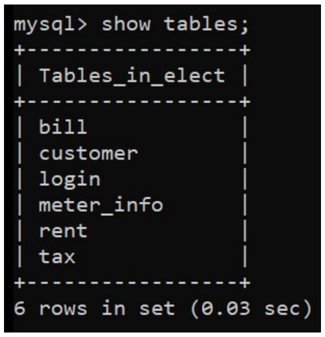
Steps	Action	Expected output
Step1 choice	The screen appears whenthe users run the program. 1.If admin login 2.If customer login	A page with differentmenu's appears. 1.Admin panel opens and 2.Customer panel opens
Step 2	The screen appears whenthe admin logs in and selects any one of the menus from the click ofthe mouse.	A window for adding new customer, insertingtax, calculate bill, view deposit details etc
Selection 1	❖ New Customer ❖ Customer Details ❖ Deposit Details ❖ Calculate Bill ❖ Tax Details ❖ Delete Customer	

Step 2.1	The screen appears whenthe customer login and selects any one of the menus from the click of the mouse	A window for generatingbill, update customer details, view details, generating bill
Selection 2	❖ Update Details ❖ View Details	
Selection 2a	❖ Generate Bill	
Selection 2b	❖ Pay Bill ❖ Bill Details	

Tables:

The given below table is a snapshot of backend view of the localhost and thestructures of the tables present in Electricity Billing System. The tables present are login, customer, tax, bill, meter_info.

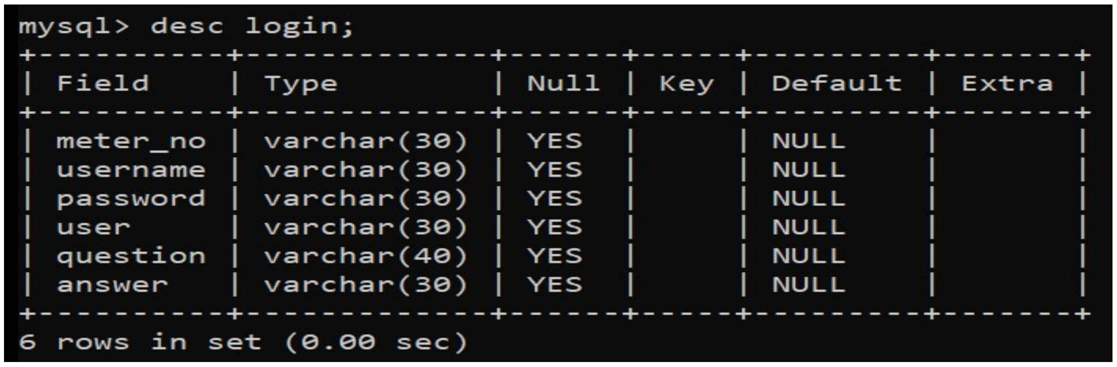
- ✓ The login is used to store the details of login's admin and customer withmeter_no.
- ✓ The customer is used to store details of customer.
- ✓ The tax is used to store tax values.
- ✓ The rent is used to store rent values.
- ✓ The bill is used to store details of bill of meter.
- ✓ The meter_info is used to store information of meter placed.



```
mysql> show tables;
+-----------------+
| Tables_in_elect |
+-----------------+
| bill            |
| customer        |
| login           |
| meter_info      |
| rent            |
| tax             |
+-----------------+
6 rows in set (0.03 sec)
```

List of tables

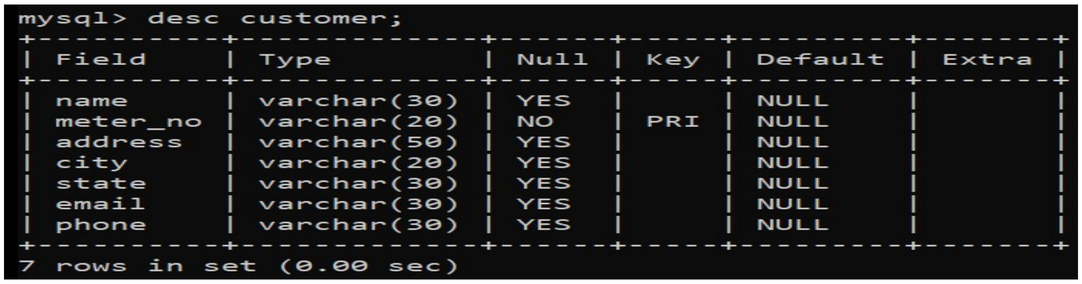
Login Table:



```
mysql> desc login;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| meter_no | varchar(30) | YES  |     | NULL    |       |
| username | varchar(30) | YES  |     | NULL    |       |
| password | varchar(30) | YES  |     | NULL    |       |
| user     | varchar(30) | YES  |     | NULL    |       |
| question | varchar(40) | YES  |     | NULL    |       |
| answer   | varchar(30) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

login table description

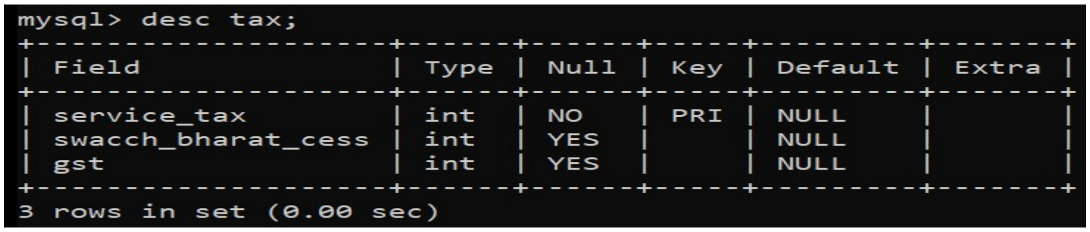
Customer Table:



```
mysql> desc customer;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| name     | varchar(30) | YES  |     | NULL    |       |
| meter_no | varchar(20) | NO   | PRI | NULL    |       |
| address  | varchar(50) | YES  |     | NULL    |       |
| city     | varchar(20) | YES  |     | NULL    |       |
| state    | varchar(30) | YES  |     | NULL    |       |
| email    | varchar(30) | YES  |     | NULL    |       |
| phone    | varchar(30) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Customer table description

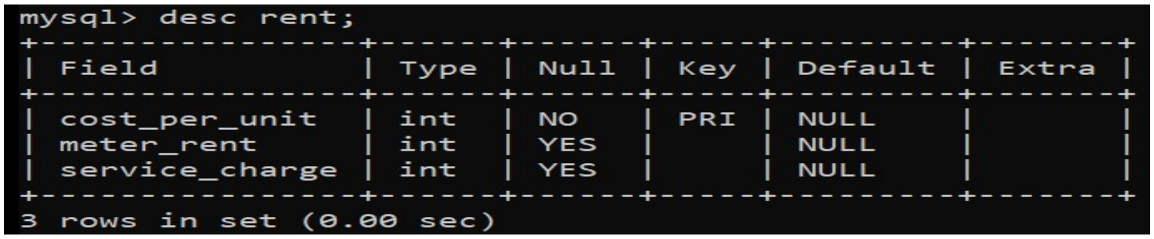
Tax Table:



```
mysql> desc tax;
+-------------------+------+------+-----+---------+-------+
| Field             | Type | Null | Key | Default | Extra |
+-------------------+------+------+-----+---------+-------+
| service_tax       | int  | NO   | PRI | NULL    |       |
| swacch_bharat_cess| int  | YES  |     | NULL    |       |
| gst               | int  | YES  |     | NULL    |       |
+-------------------+------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Tax table description

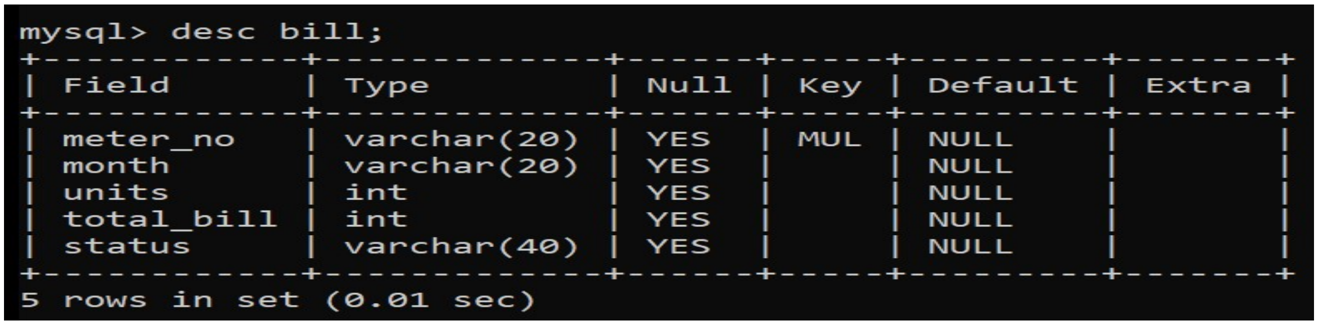
Rent Table:



```
mysql> desc rent;
+----------------+------+------+-----+---------+-------+
| Field          | Type | Null | Key | Default | Extra |
+----------------+------+------+-----+---------+-------+
| cost_per_unit  | int  | NO   | PRI | NULL    |       |
| meter_rent     | int  | YES  |     | NULL    |       |
| service_charge | int  | YES  |     | NULL    |       |
+----------------+------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Rent table description

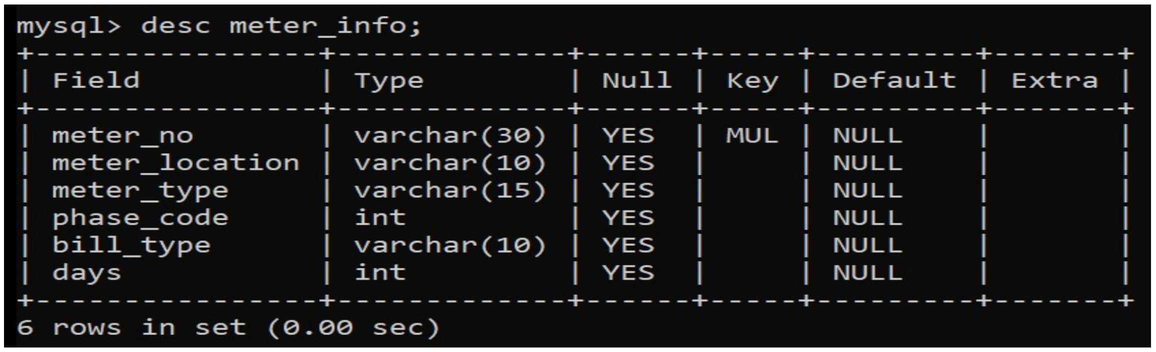
Bill Table:



```
mysql> desc bill;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| meter_no   | varchar(20) | YES  | MUL | NULL    |       |
| month      | varchar(20) | YES  |     | NULL    |       |
| units      | int         | YES  |     | NULL    |       |
| total_bill | int         | YES  |     | NULL    |       |
| status     | varchar(40) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

Bill table description

Meter info table:

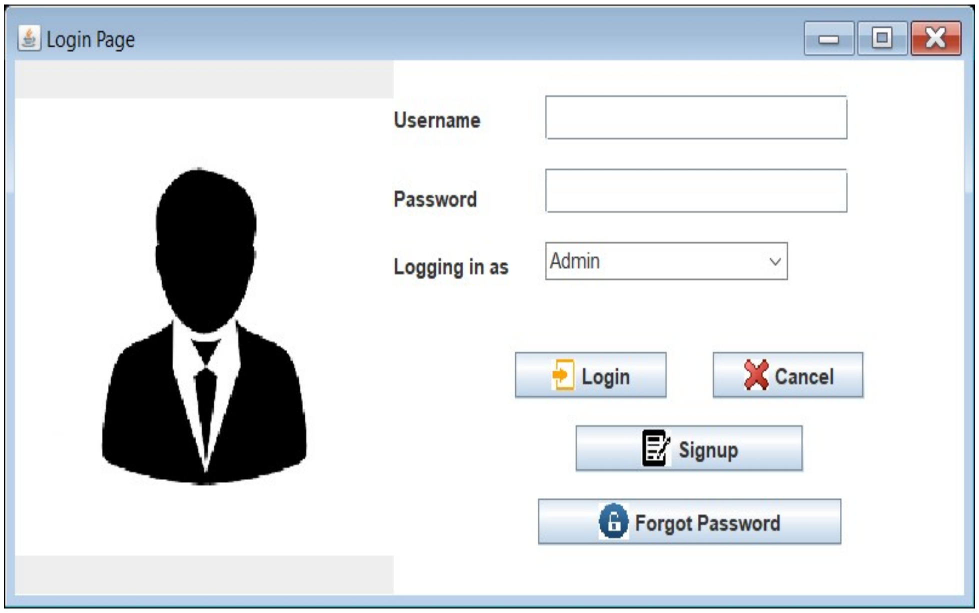


```
mysql> desc meter_info;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| meter_no       | varchar(30) | YES  | MUL | NULL    |       |
| meter_location | varchar(10) | YES  |     | NULL    |       |
| meter_type     | varchar(15) | YES  |     | NULL    |       |
| phase_code     | int         | YES  |     | NULL    |       |
| bill_type      | varchar(10) | YES  |     | NULL    |       |
| days           | int         | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

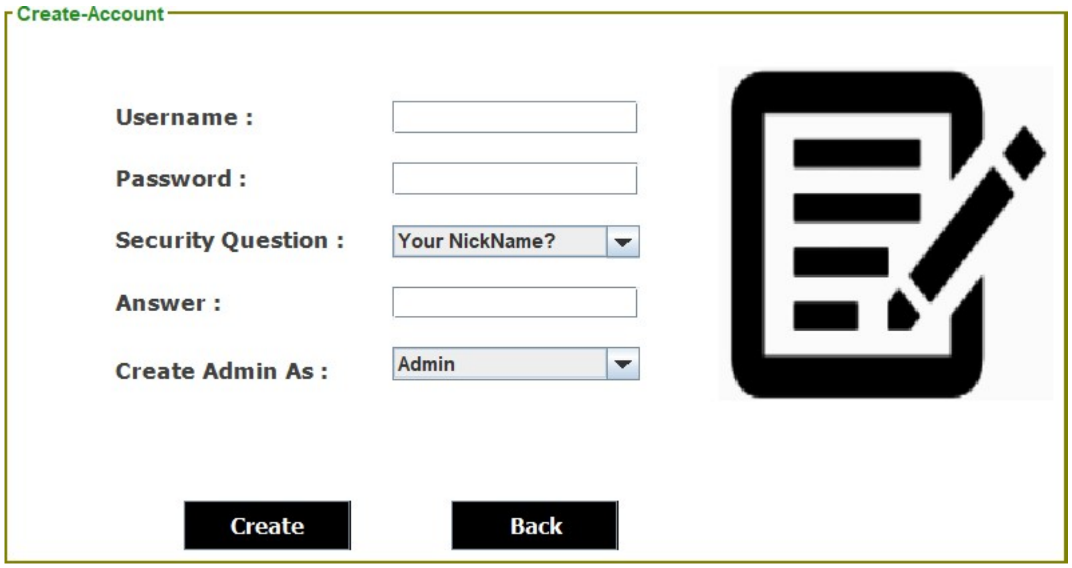
Meter info table description

SNAPSHOTS:

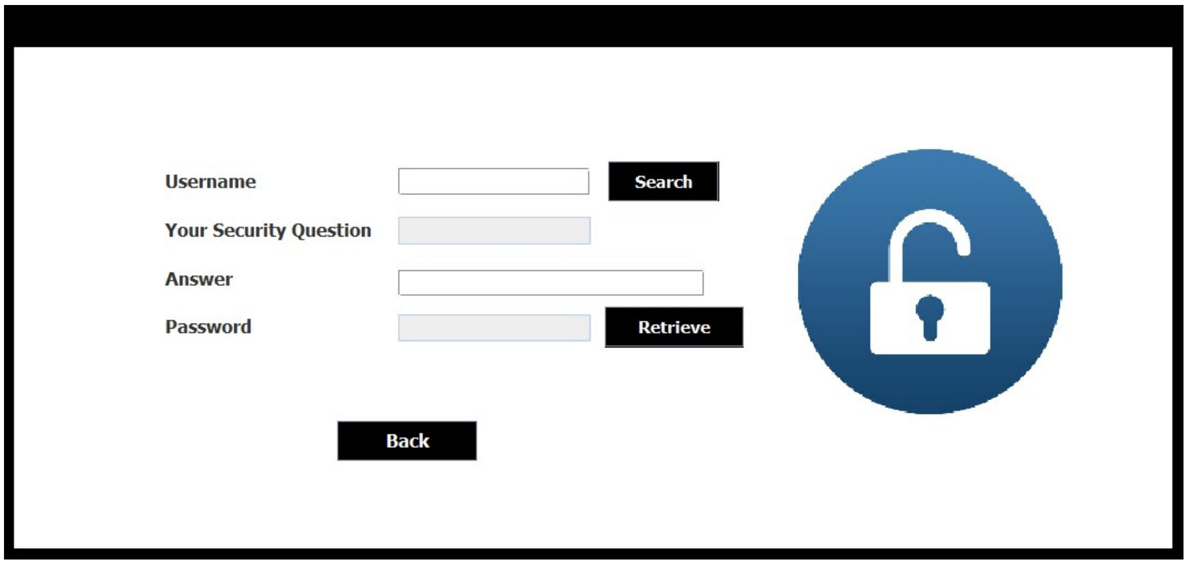
Splash page of Electricity Billing System



Login Page



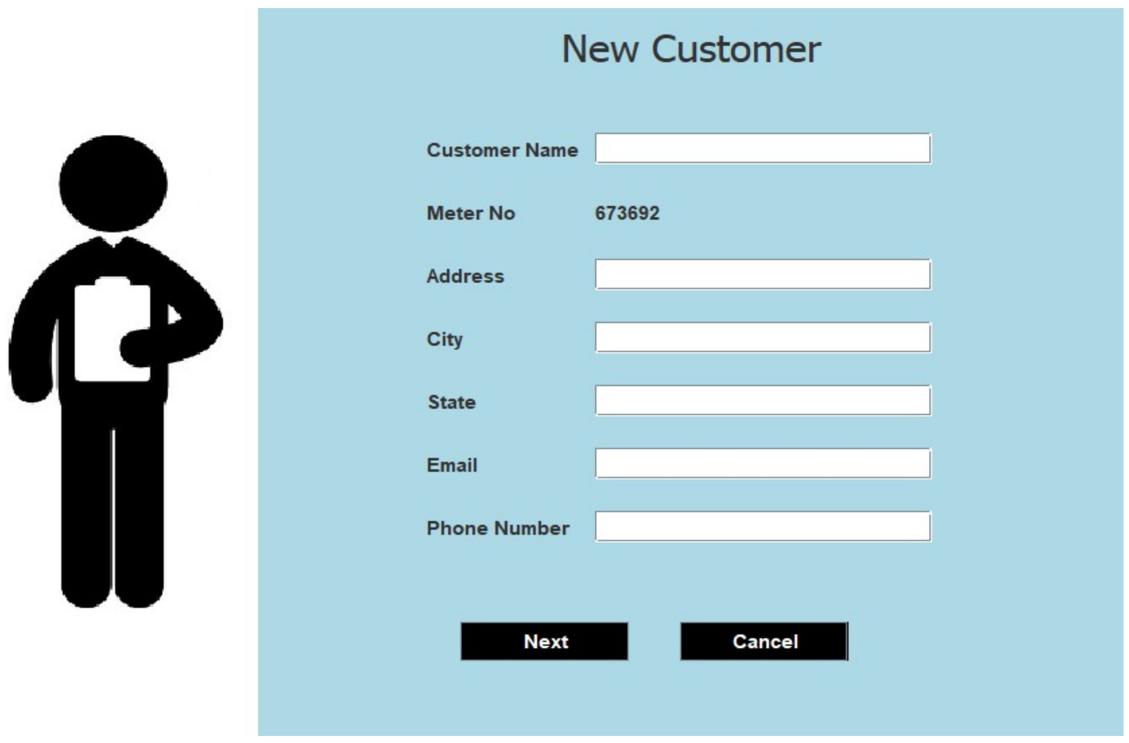
Signup page



Forget Password page

Master
Utility
About
Logout

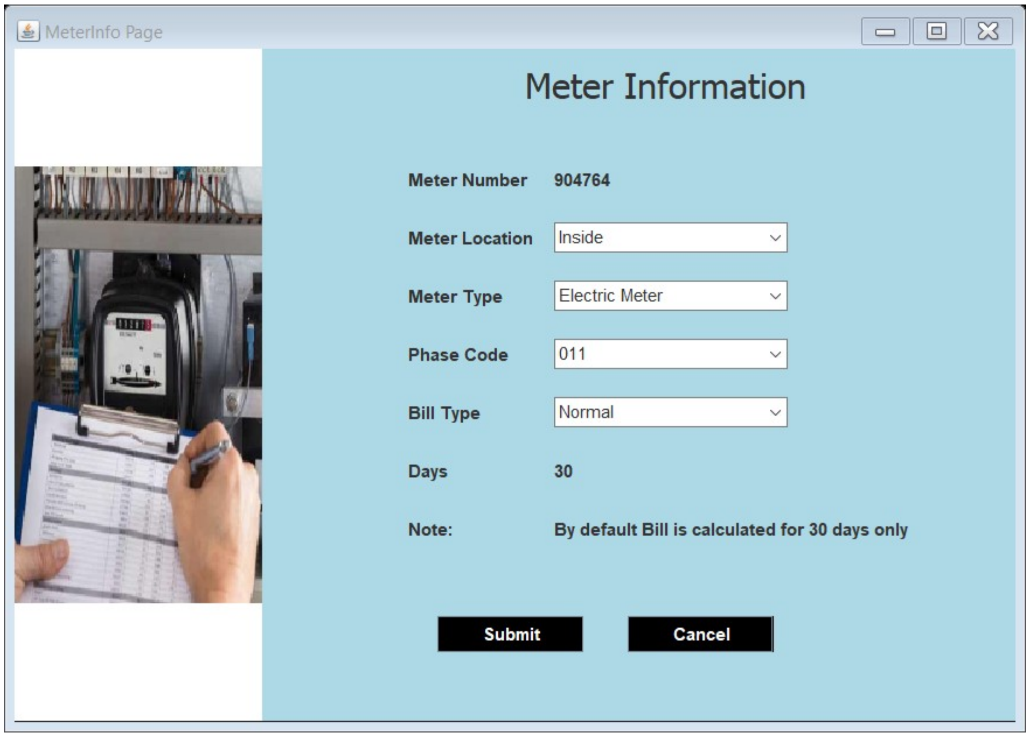
Admin home page



New Customer

Customer Name	
Meter No	673692
Address	
City	
State	
Email	
Phone Number	

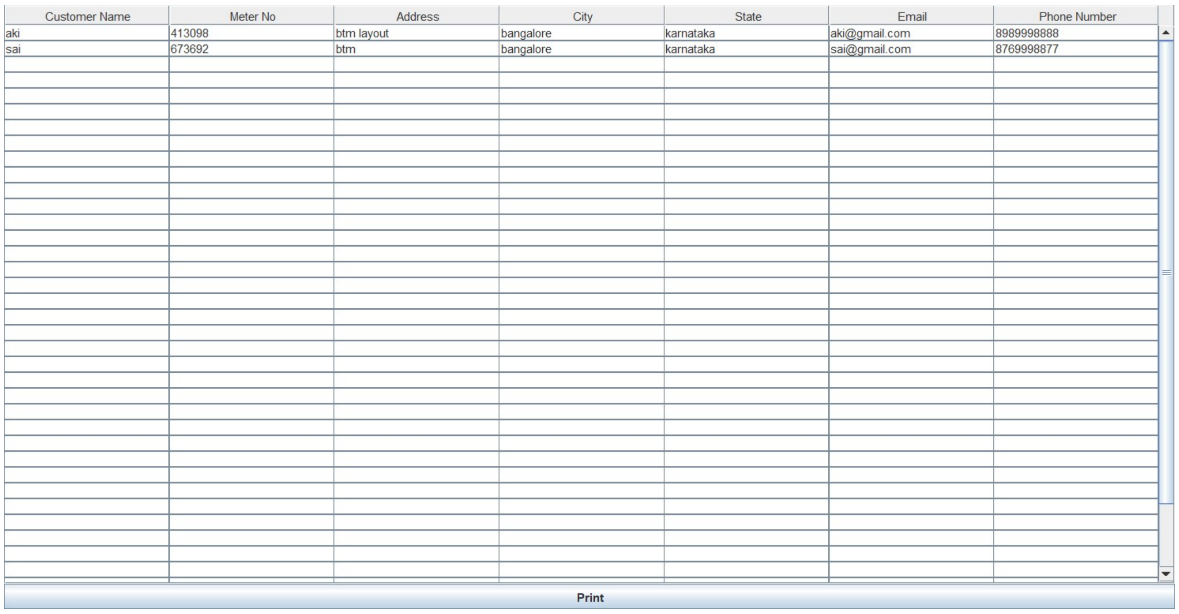
New customer page



Meter Information

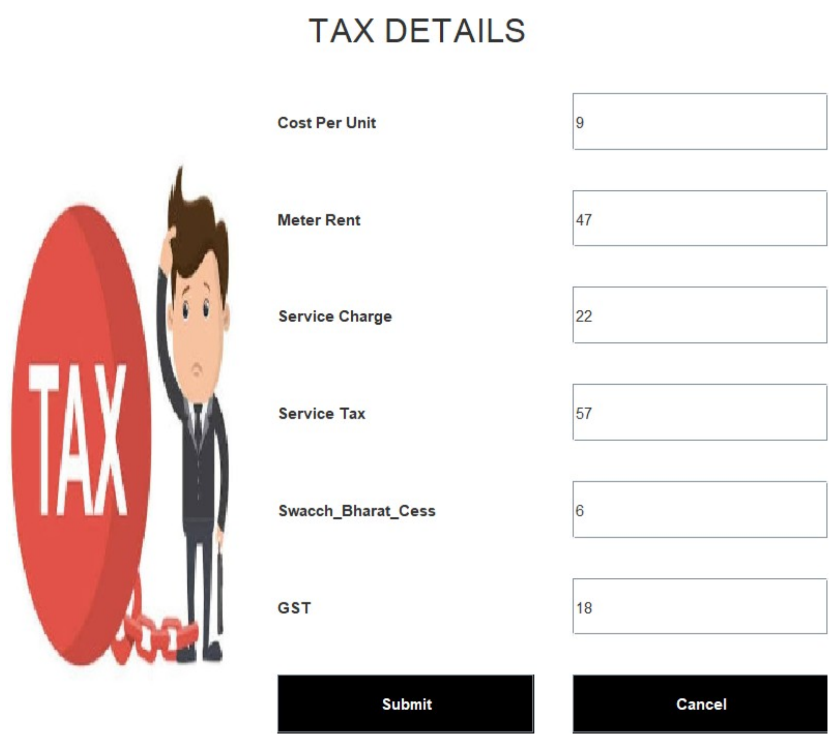
Meter Number	904764
Meter Location	Inside
Meter Type	Electric Meter
Phase Code	011
Bill Type	Normal
Days	30
Note:	By default Bill is calculated for 30 days only

Meter Info page



Customer Name	Meter No	Address	City	State	Email	Phone Number
aki	413098	btm layout	bangalore	karnataka	aki@gmail.com	8989998888
sai	673692	btm	bangalore	karnataka	sai@gmail.com	8769998877

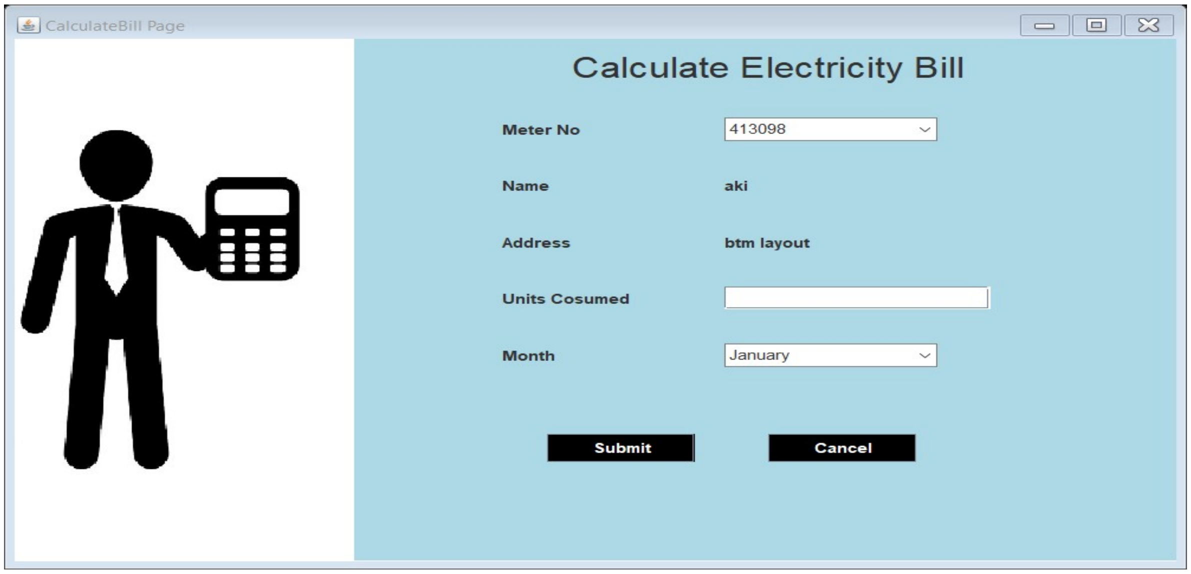
Customer Details page



TAX DETAILS

Cost Per Unit	9
Meter Rent	47
Service Charge	22
Service Tax	57
Swacch_Bharat_Cess	6
GST	18

Tax Details page

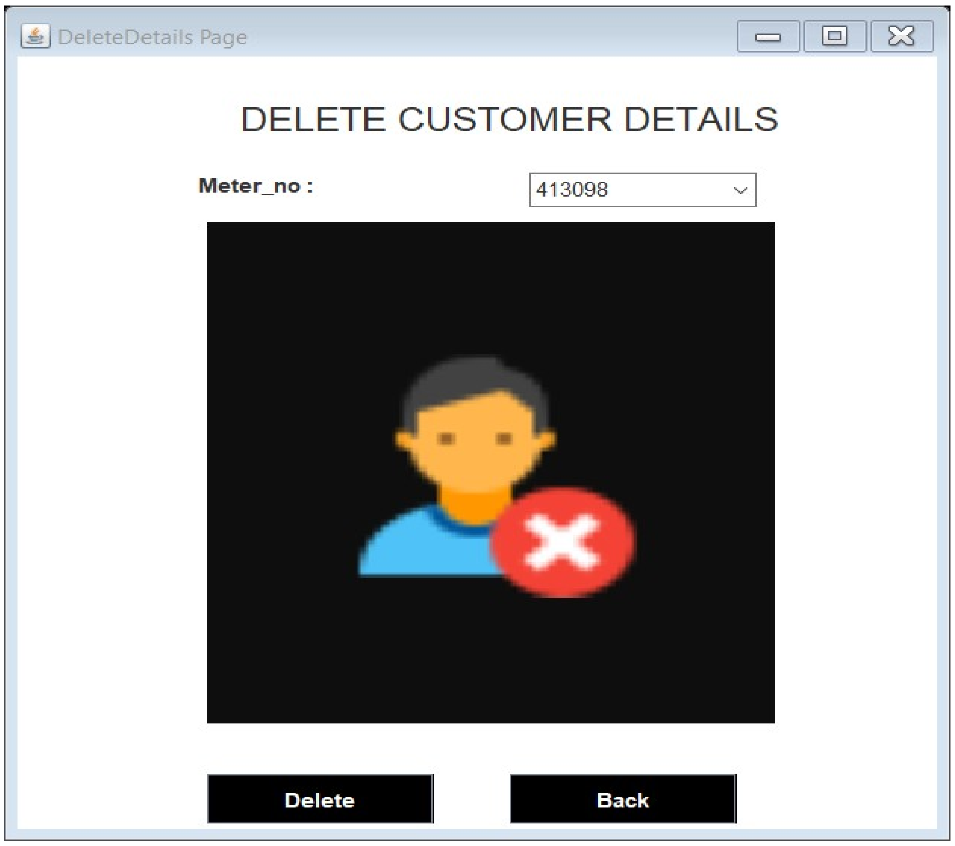


CalculateBill Page

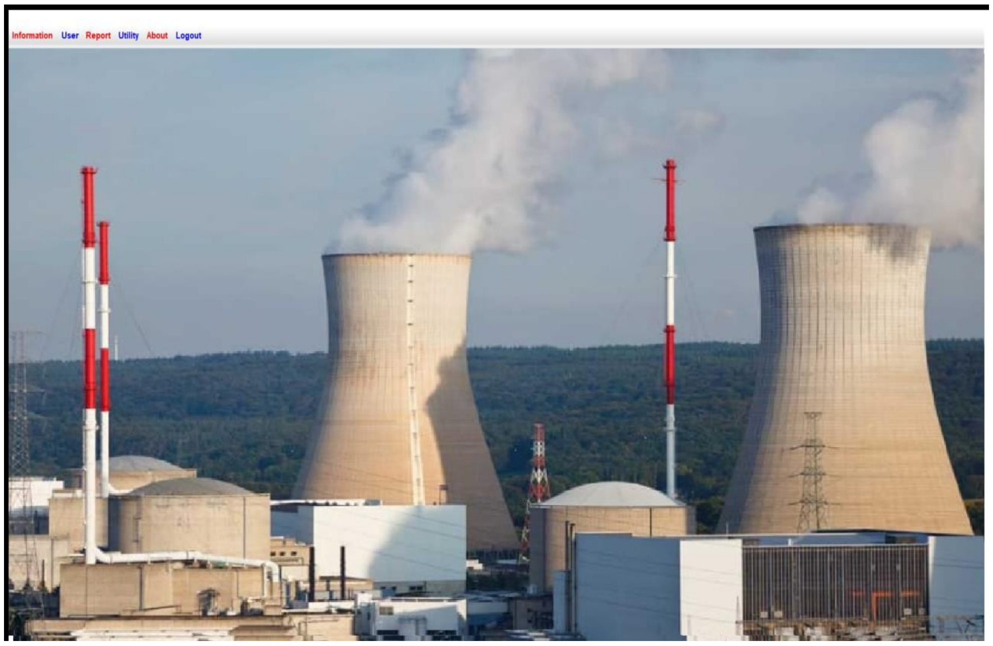
Calculate Electricity Bill

Meter No	413098
Name	aki
Address	btm layout
Units Cosumed	
Month	January

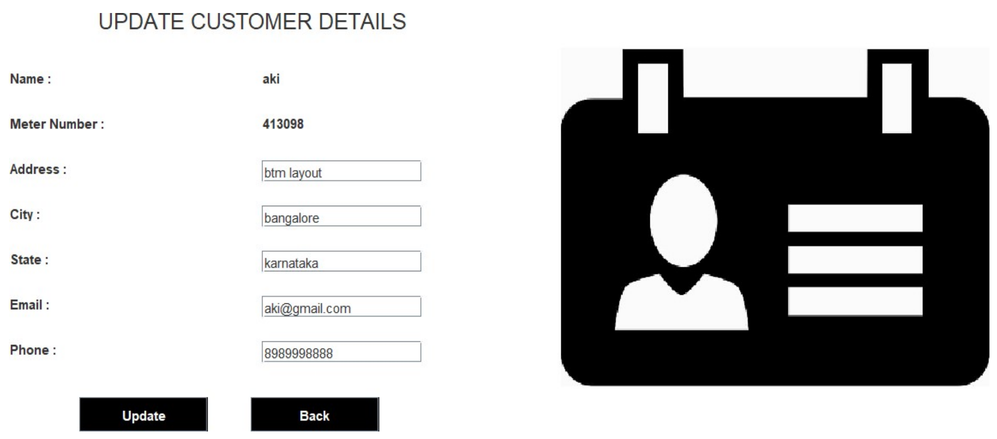
Calculate Bill page



Delete Customer page

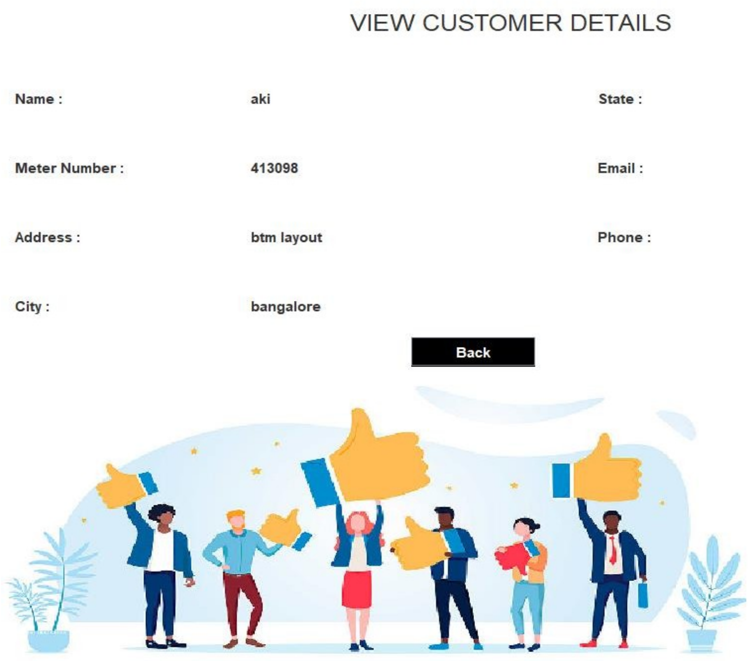


Customer Home page


UPDATE CUSTOMER DETAILS

Name :	aki
Meter Number :	413098
Address :	btm layout
City :	bangalore
State :	karnataka
Email :	aki@gmail.com
Phone :	8989998888

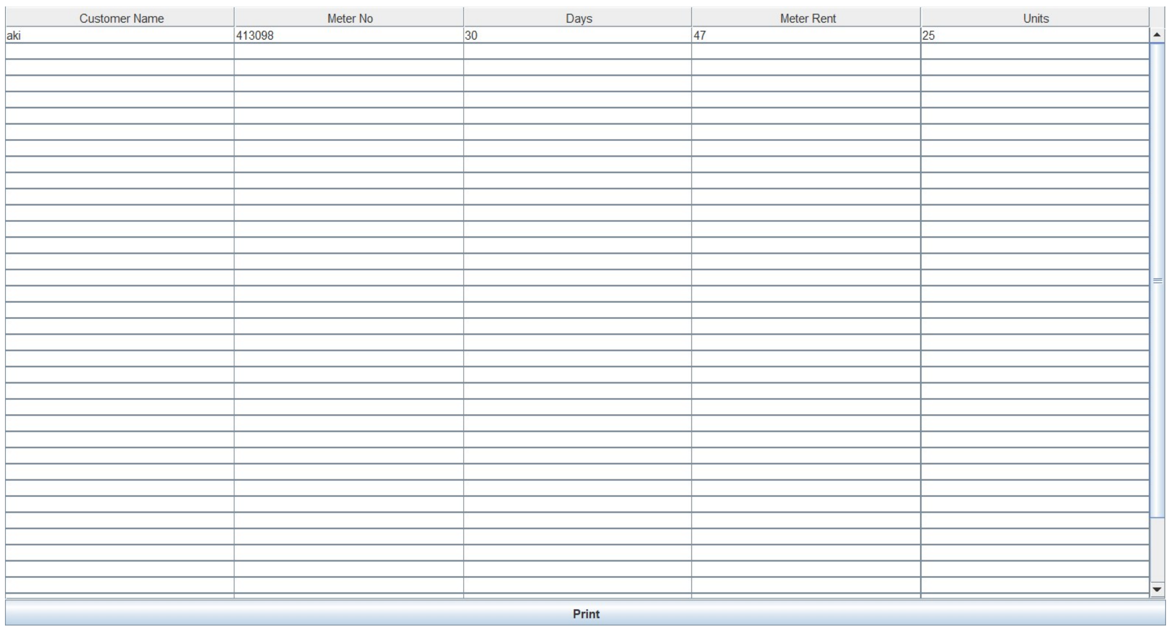
Update customer details page



VIEW CUSTOMER DETAILS

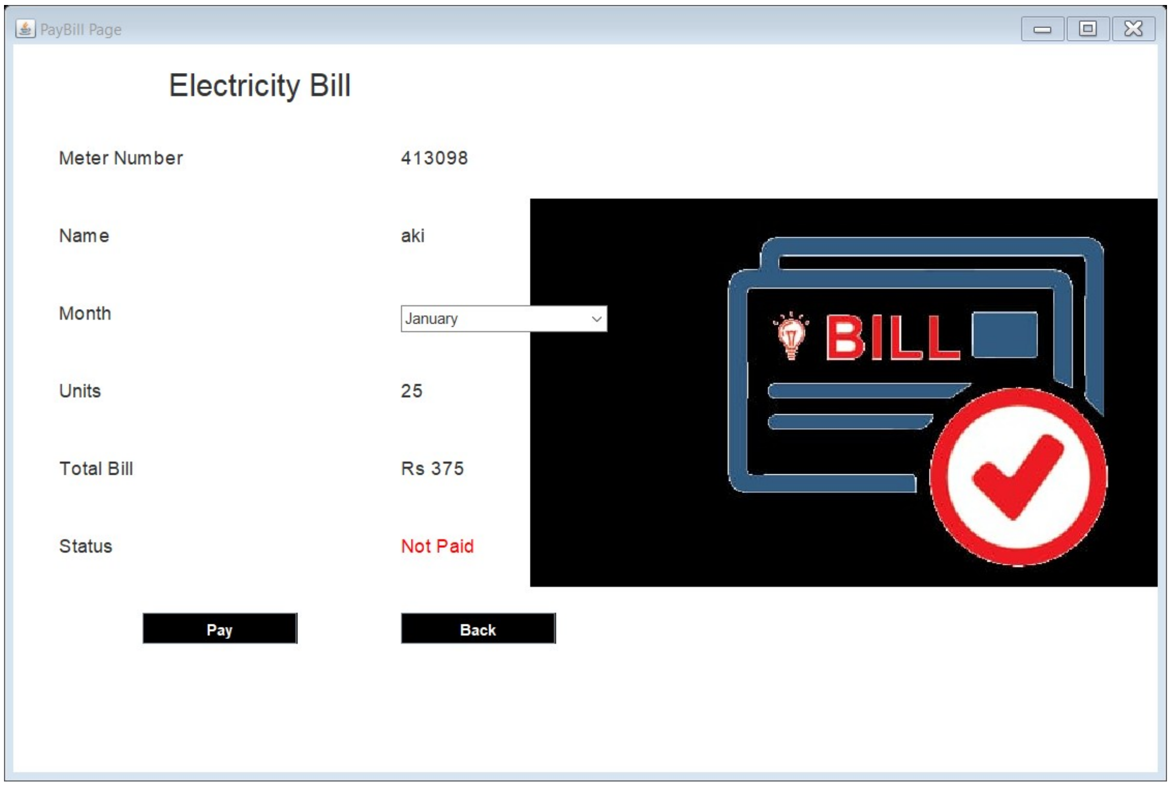
Name :	aki	State :	karnataka
Meter Number :	413098	Email :	aki@gmail.com
Address :	btm layout	Phone :	8989998888
City :	bangalore		

View Customer Details page



Customer Name	Meter No	Days	Meter Rent	Units
aki	413098	30	47	25

Query page



PayBill Page

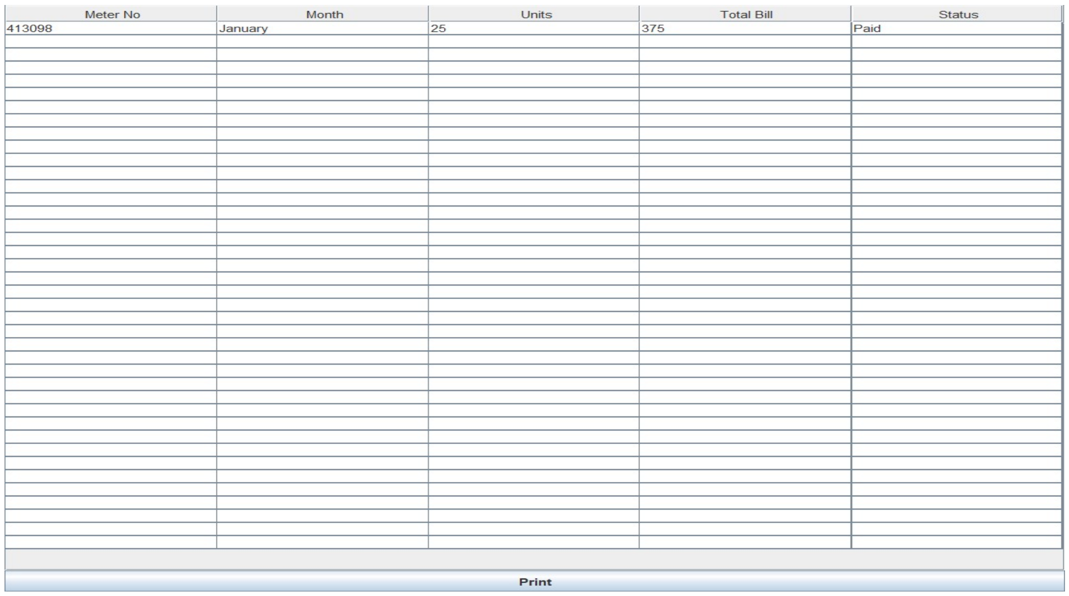
Electricity Bill

Meter Number	413098
Name	aki
Month	January
Units	25
Total Bill	Rs 375
Status	Not Paid

Pay Bill page

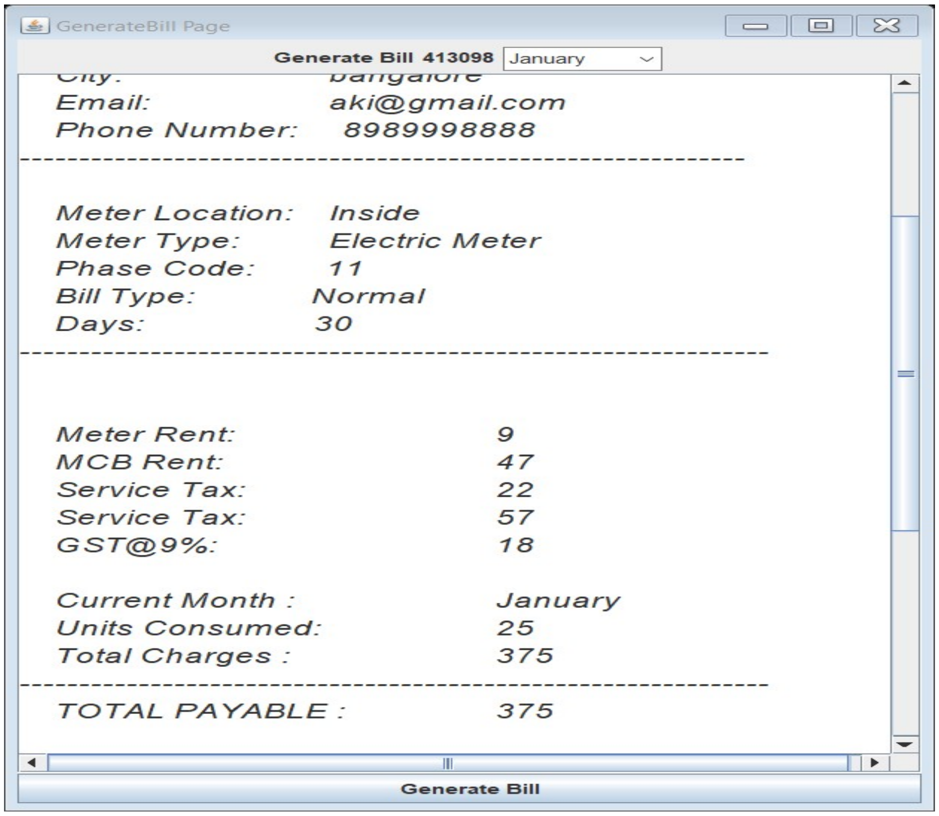
- ◦ Paytm Wallet
  ◦ **Rs. 0.00**
- ◦ No Items in Your Bag
- ◦ Log In /Sign Up
- image

Paytm page



Meter No	Month	Units	Total Bill	Status
413098	January	25	375	Paid

Bill Details page



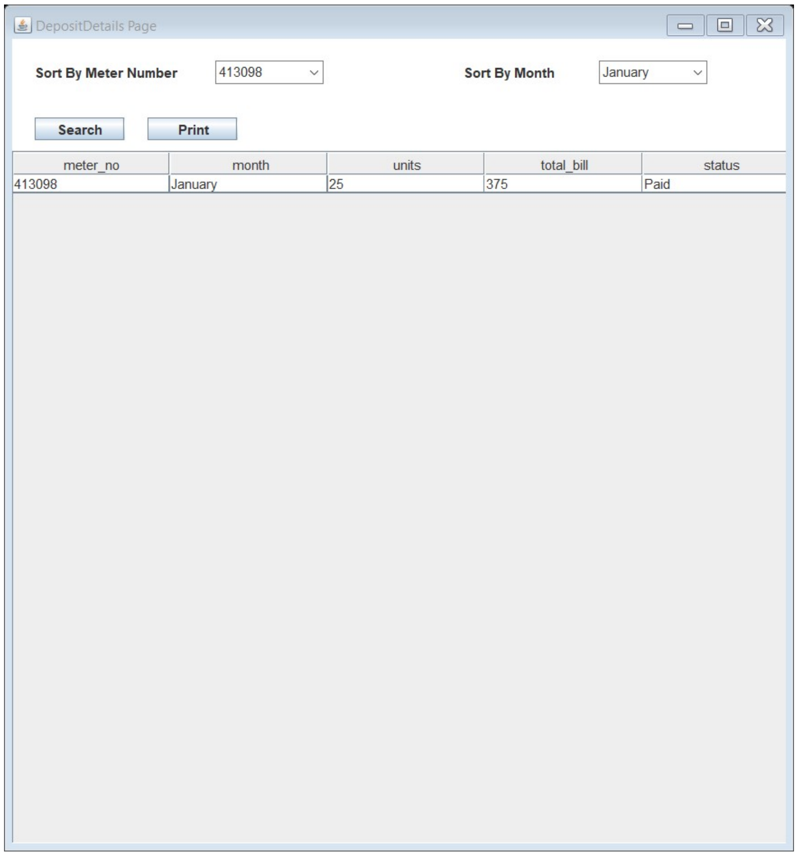
```
    City:               bangalore
    Email:              aki@gmail.com
    Phone Number:        8989998888
-----------------------------------------------------

    Meter Location:     Inside
    Meter Type:         Electric Meter
    Phase Code:         11
    Bill Type:          Normal
    Days:               30
-------------------------------------------------------


    Meter Rent:                     9
    MCB Rent:                       47
    Service Tax:                    22
    Service Tax:                    57
    GST@9%:                         18

    Current Month :                 January
    Units Consumed:                 25
    Total Charges :                 375
-------------------------------------------------------
    TOTAL PAYABLE :                 375
```

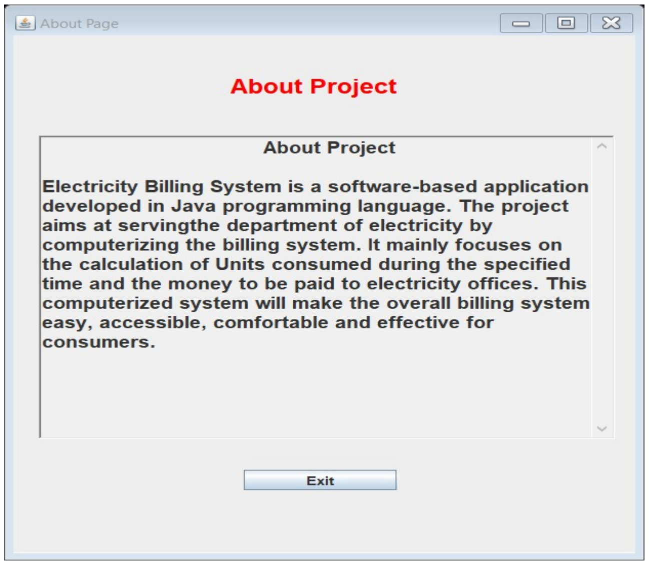
Generate Bill page



Sort By Meter Number 413098 Sort By Month January

meter_no	month	units	total_bill	status
413098	January	25	375	Paid

Deposit Details page



About Page

About Project

About Project

Electricity Billing System is a software-based application developed in Java programming language. The project aims at servingthe department of electricity by computerizing the billing system. It mainly focuses on the calculation of Units consumed during the specified time and the money to be paid to electricity offices. This computerized system will make the overall billing system easy, accessible, comfortable and effective for consumers.

About page

4.1CONCLUSION

After all the hard work is done for electricity bill management system is here.It is a software which helps the user to work with the billing cycles, paying bills, managing different DETAILS under which are working etc.

This software reduces the amount of manual data entry and gives greater efficiency. The User Interface of it is very friendly and can be easily used by anyone.

It also decreases the amount of time taken to write details and other modules.

4.1 Future Scope

The electricity billing system project holds significant potential for future enhancements and expansions. Here are some avenues for future scope:

Integration with Smart Meters:
Incorporate integration with smart meters to automate the collection of meter readings, reducing manual intervention and improving accuracy.
Mobile Application Development:
Develop a mobile application for users to conveniently access their account details, view bills, and make payments on-the-go, enhancing user experience and accessibility.
Predictive Analytics for Energy Consumption:
Implement predictive analytics algorithms to forecast future energy consumption trends based on historical data, enabling better resource planning and optimization.
Real-time Billing and Notifications:
Enable real-time billing generation and notifications to users, providing timely updates on consumption, bills, and payment reminders.
Energy Efficiency Recommendations:
Offer personalized energy efficiency recommendations to users based on their consumption patterns, promoting sustainable energy practices.
Integration with Renewable Energy Sources:
Integrate support for renewable energy sources such as solar or wind power, allowing users to track their usage and savings from renewable energy generation.

Advanced Reporting and Analytics:

Enhance reporting capabilities with advanced analytics and visualization tools, enabling stakeholders to gain deeper insights into energy consumption patterns and trends.

API Integration for Third-party Services:

Integrate with third-party services such as online payment gateways or energy management systems through APIs, expanding the system's functionality and interoperability.

Multi-language Support and Localization:

Implement multi-language support and localization features to cater to users from diverse linguistic backgrounds and geographical regions.

Regulatory Compliance Enhancements:

Stay updated with regulatory requirements and standards related to energy billing and consumption, ensuring compliance and adherence to industry regulations.

REFERENCES

- ❖ Book
- ❖ YouTube
- ❖ W3School