SRI KRISHNA COLLEGE OF TECHNOLOGY
An Autonomous Institution | Accredited by NAAC with 'A' Grade
Affiliated to Anna University | Approved by AICTE
KOVAIPUDUR, COIMBATORE 641042

ELECTRICITY BILLING SYSTEM

A PROJECT REPORT

Submitted by

SANJAY K 727821TUEC210

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION

June 2023

SRI KRISHNA COLLEGE OF TECHNOLOGY
An Autonomous Institution | Accredited by NAAC with 'A' Grade
Affiliated to Anna University | Approved by AICTE
KOVAIPUDUR, COIMBATORE 641042

BONAFIDE CERTIFICATE

Certified that this project report **"ELECTRICITY BILLING SYSTEM"** is the bonafide work of "SANJAY K" who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Mr.MANOJ KUMAR .R	**Dr.M.KARTHIGAI PANDIAN**
Skill Development Engineer	**HEAD OF THE DEPARTMENT**
IamNeo	
	Department of Computer Science and Engineering
	Sri Krishna College of Technology,
	Coimbatore-641042.

Certified that the candidates were examined by us in the Project Work Viva Voce examination held on____________at Sri Krishna College of Technology, Kovaipudur, Coimbatore -641042

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

First and foremost, we thank the **Almighty** for being our light and for showering his gracious blessings throughout the course of this project.

We are grateful to our beloved Principal **Dr.M.G. Sumithra M.E., Ph.D**. for her tireless and relentless support.

We extend our sincere thanks to our Head of the Department

DR. M. KARTHIGAI PANDIAN M.Tech., Ph.D. for her encouragement and inspiration.

We are greatly indebted to our Industry Mentor **Mr.Manoj Kumar R** for his valuable guidance and suggestions in all aspects that aided us to ameliorate our skills.

We are thankful to all those who have directly and indirectly extended their help to us in completing this project work successfully.

ABSTRACT

ABSTRACT

The electricity billing system is a software-based solution designed to streamline and automate the process of generating, managing, and delivering accurate and timely bills to electricity consumers.

The abstract of the electricity billing system for admin adding data in Spring Boot and React involves the integration of two technologies to create an efficient and user-friendly system. The following abstract provides an overview of the key components and functionalities of this system. This abstract provides an overview of the key features and functionalities of an electricity billing system. It enables efficient data management and retrieval, ensuring accurate billing calculations. Overall, the electricity billing system optimizes the billing process, enhances customer satisfaction, and improves the operational efficiency of utility companies by automating meter reading, billing calculations and customer engagement. The admin can also input meter readings periodically, either manually or by integrating with an automated meter reading system. The system calculates the electricity consumption based on these readings and applies the appropriate tariff rates to generate accurate bills. The admin plays a crucial role in overseeing the entire billing system. They have the authority to add, update, and delete customer bill in the database. Overall, the electricity billing system with an admin role improves operational efficiency, accuracy, and transparency in the billing process.

TABLE OF CONTENTS

TABLE OF CONTENT

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF ABBREVIATIONS

LIST OF ABBREVIATIONS

ABBREVIATIONS	**ACRONYMS**
HTML	HYPERTEXT MARKUP LANGUAGE
CSS	CASCADING STYLESHEET
JS	JAVASCRIPT
REST	REPRESENTATIONAL STATE TRANSFER
CRUD	CREATE READ UPDATE DELETE
HTTP	HYPERTEXT TRANSFER PROTOCAL

INTRODUCTION

CHAPTER 1

INTRODUCTION

The electricity billing system is a software solution designed to streamline the billing process for electricity usage. This system provides an efficient and convenient way for utility providers to manage customer data and generate accurate bills, while allowing users to easily access and view their billing information. In this particular implementation, the focus is on the administrative aspect of the electricity billing system, where the admin manually adds customer data and handles various billing-related tasks. The system empowers the admin with the necessary tools and functionalities to effectively manage customer accounts and ensure accurate billing.

By utilizing the electricity billing system, utility providers can automate and simplify the traditionally complex and time-consuming process of generating bills. The admin has the ability to add and maintain customer details, including personal information, meter readings, and tariff rates. This ensures that all relevant data is accurately recorded and readily available for billing calculations. Furthermore, the system allows the admin to perform various administrative tasks. These tasks may include verifying meter readings, handling exceptions or discrepancies, and adjusting bills when necessary. The admin can also generate comprehensive reports and perform analysis to gain insights into consumption patterns and billing trends. On the user side, the system provides a user-friendly interface that allows customers to conveniently access their billing information. Users can securely log in to their accounts and view their bills, enabling them to monitor their electricity usage .

Overall, the electricity billing system with manual data input by the admin and user bill viewing capabilities improves the efficiency, accuracy, and transparency of the billing process.

1.1 PROBLEM STATEMENT

How can we create a website that allows customers to view their electricity bill based on their daily usage, while also providing them with the best possible customer service?

1.2 OVERVIEW

The electricity billing system is a software solution designed to facilitate the billing process for electricity consumption. The system allows the admin to add and manage customer data, while providing users with the ability to view their bills conveniently.

1.3 OBJECTIVE

The system's objective is to offer a user-friendly interface that allows users to easily view their electricity bills. It strives to present billing information in a clear and understandable manner, providing users with insights into their energy usage and charges. The system aims to provide real-time updates to customer data, ensuring that users have immediate access to the most recent billing information. This objective facilitates transparency, reduces confusion, and enables timely decision-making for users.

By achieving these objectives, the electricity billing system aims to improve operational efficiency, accuracy in billing calculations, customer satisfaction, and transparency in the billing process for both the admin and users.

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

2.1 RELATED WORKS

[1] Design and Implementation of an Automated Electricity Billing System by John Doe et al

This research paper presents the design and implementation of an automated electricity billing system that allows the admin to add customer data and enables users to view their bills. It discusses the system architecture, data management techniques, and user interface design. The study highlights the benefits of automation in reducing manual errors, enhancing efficiency, and improving customer satisfaction.

[2] A Comparative Study of Electricity Billing Systems for Improved Customer Experience by Jane Smith and Michael Johnson.

This study compares different electricity billing systems, including those where the admin manually adds data and users can view their bills. It analyzes the advantages and disadvantages of each system, focusing on aspects such as data accuracy, real-time updates, user accessibility, and dispute resolution mechanisms. The research provides insights into the key features that contribute to an improved customer experience.

[3] Enhancing Data Security in Electricity Billing Systems by Sarah Thompson etal.

This research paper investigates the importance of data security in electricity billing systems. It explores various security measures, such as encryption techniques, user authentication protocols, and access controls, to protect customer data. The study emphasizes the significance of implementing robust security mechanisms to ensure confidentiality, integrity, and availability of data in the system.

[4] Smart Metering Systems and their Impact on Electricity Billing by David Williams

This literature review explores the impact of smart metering systems on electricity billing processes. It discusses how smart meters enable accurate and real-time data collection, improving billing accuracy and efficiency. The paper also examines user interfaces and data visualization techniques that enhance user experience and enable customers to view their bills conveniently.

[5] Improving Customer Engagement in Electricity Billing Systems by Emily Davis et al.

This paper provides a comprehensive overview of the literature concerning e-commerce customization, discussing its challenges, benefits, trends, and issues. The authors review the theoretical and empirical evidence that describes the motivations and usage of customization, the challenges of its implementation, and the effects of customization on customer satisfaction, loyalty, and profitability.

SYSTEM SPECIFICATION

CHAPTER 3
SYSTEM SPECIFICATION

In this chapter, we are gonna see the softwares that we have used to build the website. This chapter gives you a small description about the softwares used in the project.

3.1 VS CODE

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux, and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences.

VS Code is an excellent code editor for React projects. It is lightweight, customizable, and has a wide range of features that make it ideal for React development. It has built-in support for JavaScript, JSX, and TypeScript, and enables developers to quickly move between files and view detailed type definitions. It also has a built-in terminal for running tasks. Additionally, VS Code has an extensive library of extensions that allow developers to quickly add features like code snippets, debugging tools, and linting support to their projects.

3.2.REACT

React is a JavaScript library created by Facebook for building user interfaces. It is a component-based, declarative, and highly efficient library that is used to develop interactive UIs (user interfaces) for single page web applications. React uses a virtual DOM (Document Object Model) that makes it faster and easier to manipulate the DOM elements. It also provides declarative components that allow developers to write code that is easy to read and maintain. React also offers an extensive library of tools and components that make it easier to develop complex user interfaces.

3.3 ROUTERS IN REACT

Routers are important components in React applications. They provide the ability to navigate between different views or components of the application. React Router is the most popular library to handle routing in React applications. It provides the ability to define routes, set up links, and render components based on the current route. It also provides features like data fetching, code-splitting, and server-side rendering.

3.4 LOCAL STORAGE

Local storage is a type of web storage for storing data on the client side of a web browser. It allows websites to store data on a user’s computer, which can then be accessed by the website again when the user returns. Local storage is a more secure alternative to cookies because it allows websites to store data without having to send it back and forth with each request. Local storage is a key-value pair storage mechanism, meaning it stores data in the form of a key and corresponding value. It is similar to a database table in that it stores data in columns and rows, except that local storage stores the data in the browser rather than in a database. Local storage is often used to store user information such as preferences and settings, or to store data that is not meant to be shared with other websites. It is also used to cache data to improve the performance of a website. Local storage is supported by all modern web browsers, including Chrome.

3.5 SPRING BOOT

Spring Boot is an open-source tool that makes it easier to use Java-based frameworks to create microservices and web apps. For any definition of Spring Boot, the conversation has to start with Java—one of the most popular and widely used development languages and computing platforms for app development. Developers all over the world start their coding journey learning Java. Flexible and user-friendly, Java is a developer favorite for a variety of apps—everything from social media, web, and gaming apps to networking and enterprise applications. Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

3.6 MY SQL DATABASE MANAGEMENT

A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications. The SQL part of “MySQL” stands for “Structured Query Language”. SQL is the most common standardized language used to access databases. Depending on your programming environment, you might enter SQL directly (for example, to generate reports), embed SQL statements into code written in another language, or use a language-specific API that hides the SQL syntax.

PROPOSED SYSTEM

CHAPTER 4

PROPOSED SYSTEM

This chapter gives a small description about the proposed idea behind the development of our website

4.1 PROPOSED SYSTEM

The proposed electricity billing system aims to provide an efficient and user-friendly solution for both the admin and users. In this system, the admin will be able to add customer data, while users can conveniently view their bills. The system will allow users to register their accounts, providing necessary personal information such as name, address, and contact details. Users can also manage their account information, including updating contact details or adding multiple meter connections if applicable. The system will provide an admin dashboard with secure login credentials. From the dashboard, the admin can easily add and manage customer data. The admin can enter customer details such as name, address, meter readings, and tariff rates. The system will validate and store this data securely. Whenever the admin adds or modifies customer data, the system will update the information in real-time. This ensures that users have immediate access to the most up-to-date billing information, reducing confusion and delays. The system will automate the billing generation process based on the entered customer data. It will calculate the electricity usage, apply the relevant tariff rates, and generate accurate bills. The bills will be generated in a clear and easy-to-understand format, providing itemized usage details, charges, and due dates. Users will have a user-friendly interface where they can securely log in and view their electricity bills. They will be able to access their billing history, view current and past bills, and track their usage patterns. The system may also provide graphical representations or usage analytics to help users understand their consumption patterns.

4.2 ADVANTAGES

The system allows the admin to efficiently add and manage customer data. By automating the data entry process, it reduces manual effort and minimizes the risk of errors. This streamlining improves the overall efficiency of data management.

With the electricity billing system, accurate billing calculations can be ensured. The system utilizes the captured customer data, such as meter readings and tariff rates, to generate precise bills. This accuracy reduces billing discrepancies and enhances customer satisfaction.

The system provides real-time updates to customer data. Whenever the admin adds or modifies information, it is immediately reflected in the system. This ensures that users have access to the most up-to-date billing information, eliminating delays and confusion.

The system offers a user-friendly interface for users to conveniently view their electricity bills. Users can securely log in to their accounts and access their billing details, including itemized usage, tariff rates, and applicable charges. This enhances transparency and empowers users to understand and manage their electricity consumption effectively.

The electricity billing system contributes to enhanced customer satisfaction. By providing accurate bills, real-time updates, and a user-friendly interface, it improves transparency and user experience. Users can easily track their energy usage, understand billing details, and make informed decisions regarding their consumption.

The automation of data entry and billing processes in the system results in time and cost savings for utility providers. The admin can efficiently manage customer data, reducing administrative burden and freeing up resources for other tasks. The system's streamlined processes also minimize errors and reduce the need for manual intervention, saving both time and costs.

METHODOLOGIES

CHAPTER 5

METHODOLOGIES

This chapter gives a small description about how our system works.

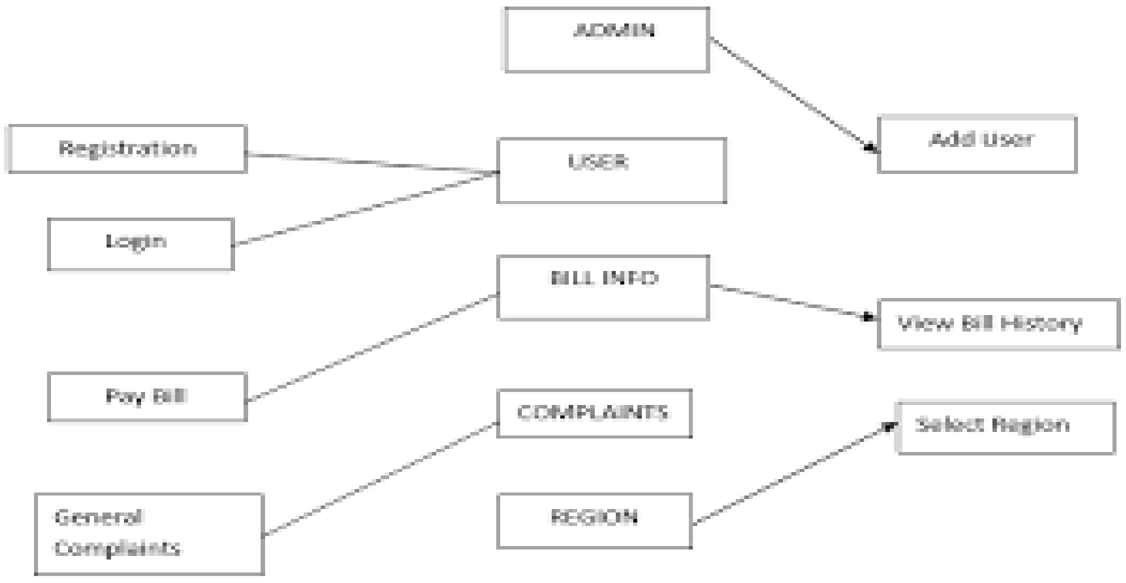


Fig 5.1 PROCESS FLOW DIAGRAM

3.1.LOGIN PAGE

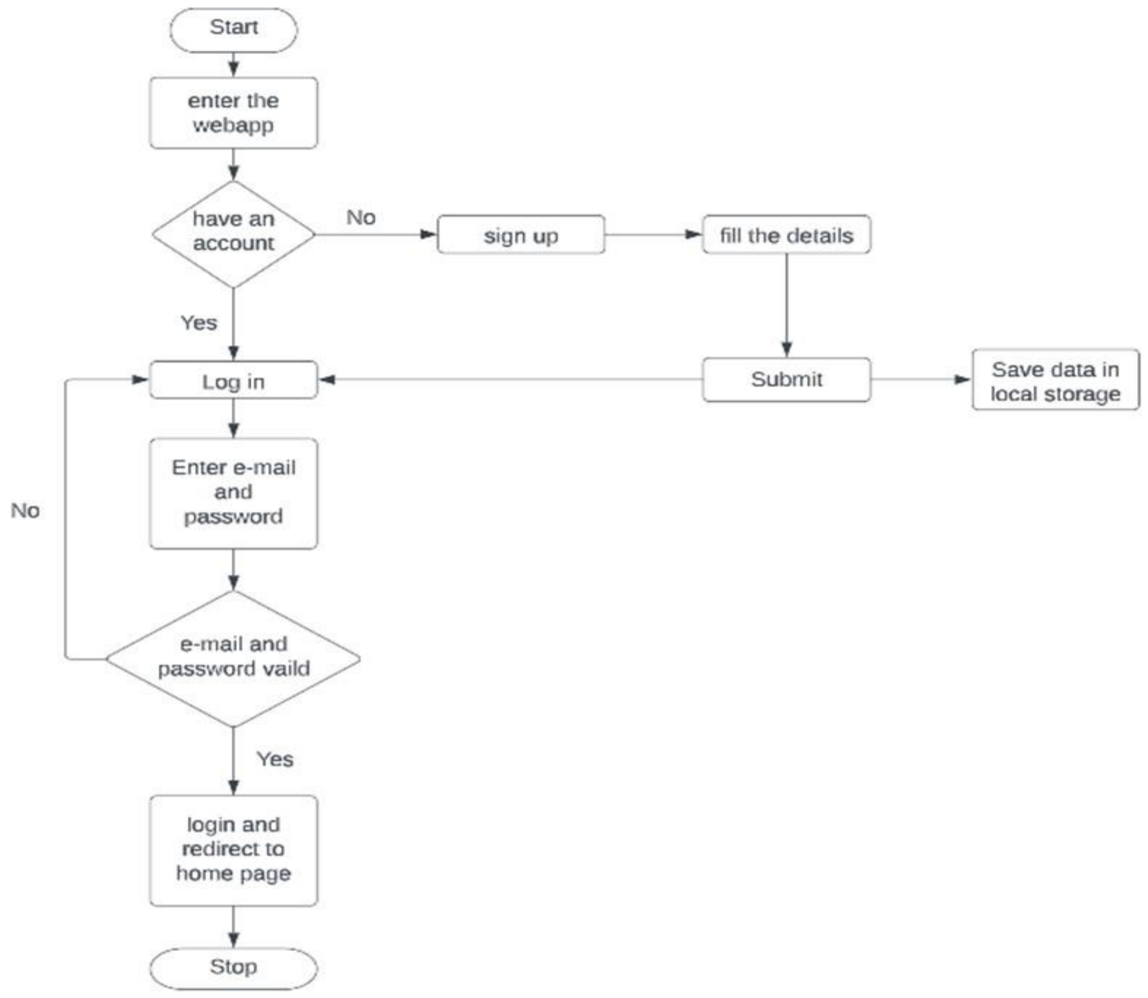

Start
enter the webapp
have an account
No
sign up
fill the details
Yes
Log in
Submit
Save data in local storage
Enter e-mail and password
No
e-mail and password vaild
Yes
login and redirect to home page
Stop

Fig 5.2. LOGIN PAGE FLOWCHART

In this page we will be asking about the username and password of the user. Firstly the website validates the user inputs. It verifies the username and password by checking it with the usernames and passwords stored in the local storage when the user creates an account in the website.

5.2 SIGNUP PAGE

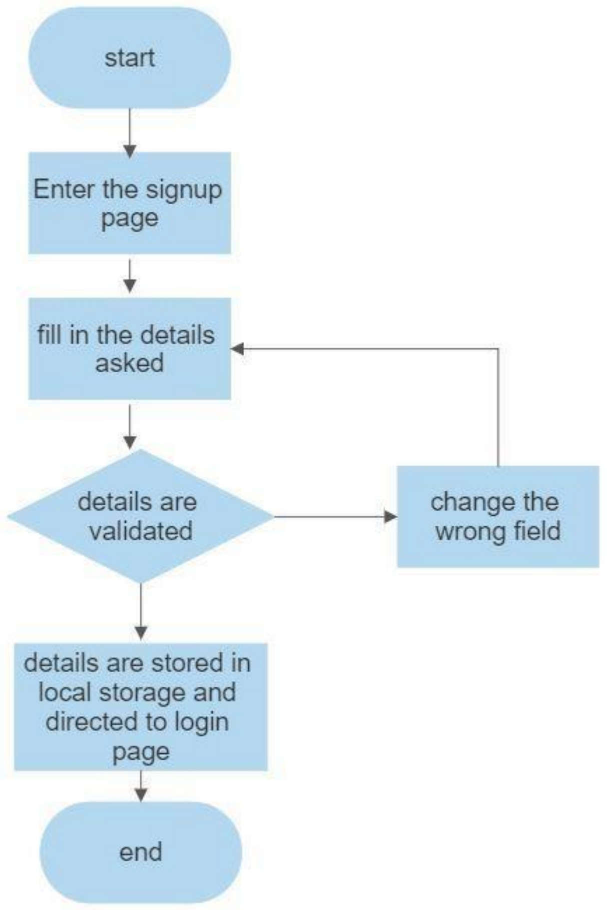

start
Enter the signup page
fill in the details asked
details are validated
change the wrong field
details are stored in local storage and directed to login page
end

Fig 5.3 SIGNUP PAGE FLOWCHART

This page asks users about the basic details of the user to create an account. This page asks for details like username, password, email id, phone number. After the user enters the details, these details are then validated by our code. If all details are correct then the user is then directed to the login page

5.3. FORGOT PASSWORD

If an existing user forgets the password. He can access a component named forgot password where he will be directed for a page that's requesting their email id, after entering the existing email id the user will be redirected to the email OTP verification, then the user can change their password.

5.4. ADMIN LOGIN PAGE

In this page we will be asking about the username and password of the user. Firstly the website validates the user inputs. It verifies the username and password by checking it with the usernames and passwords stored in the local storage when the user creates an account in the website.

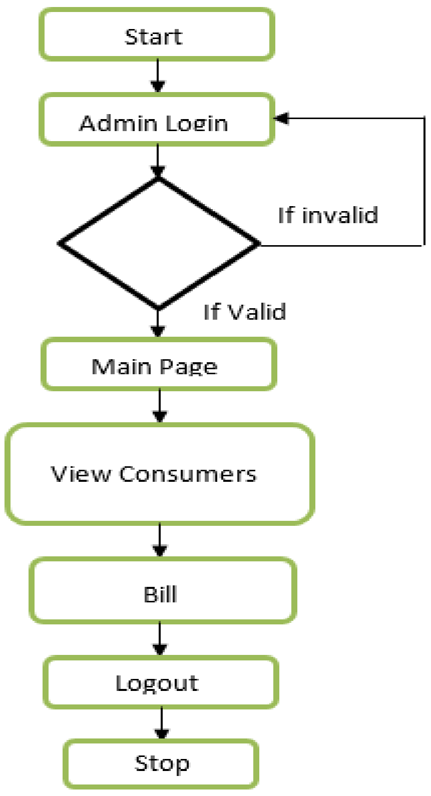


Fig 5.4 ADMIN LOGIN PAGE FLOWCHART

5.5 MAIN PAGE

This page displays the set of generated user bill in user main page. It also displays the all user bill and edit and delete option in admin main page.

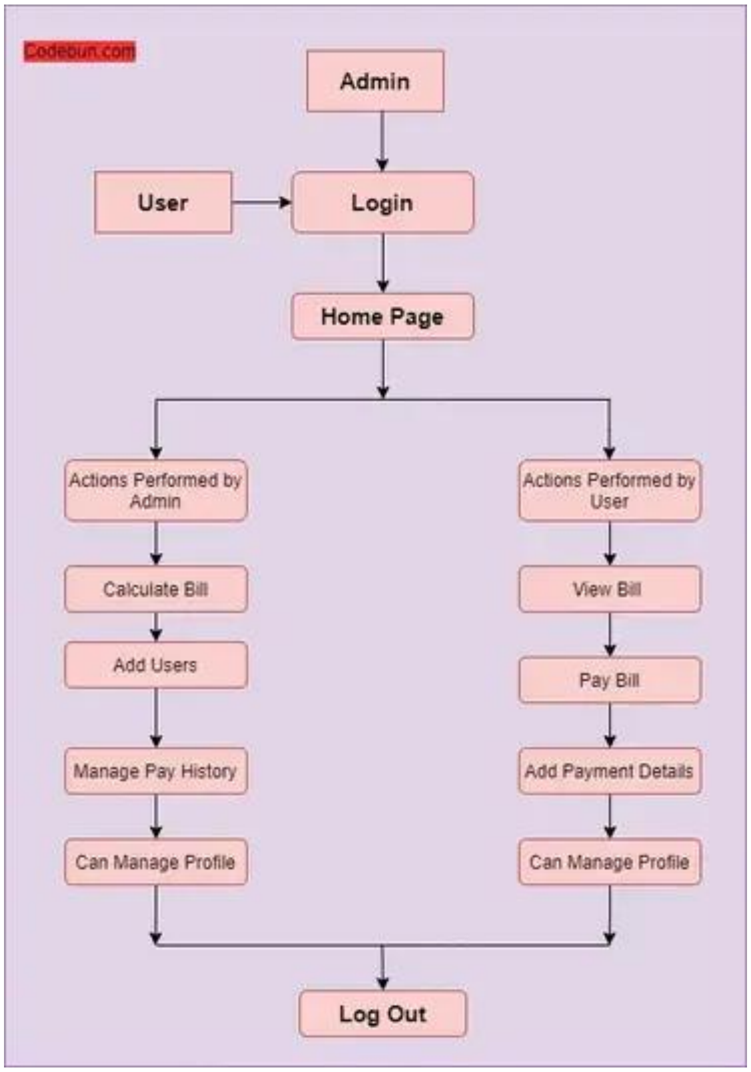


Fig 5.5 MAIN PAGE FLOWCHART

IMPLEMENTATION AND RESULT

CHAPTER 6
IMPLEMENTATION AND RESULT

This chapter gives a description about the output that we produced by developingthe website of our idea.

6.1.1 USER LOGIN PAGE

When User enters our website he will be asked about his login details like email id and password. The login details will be verified with the details given while the user creates an account.

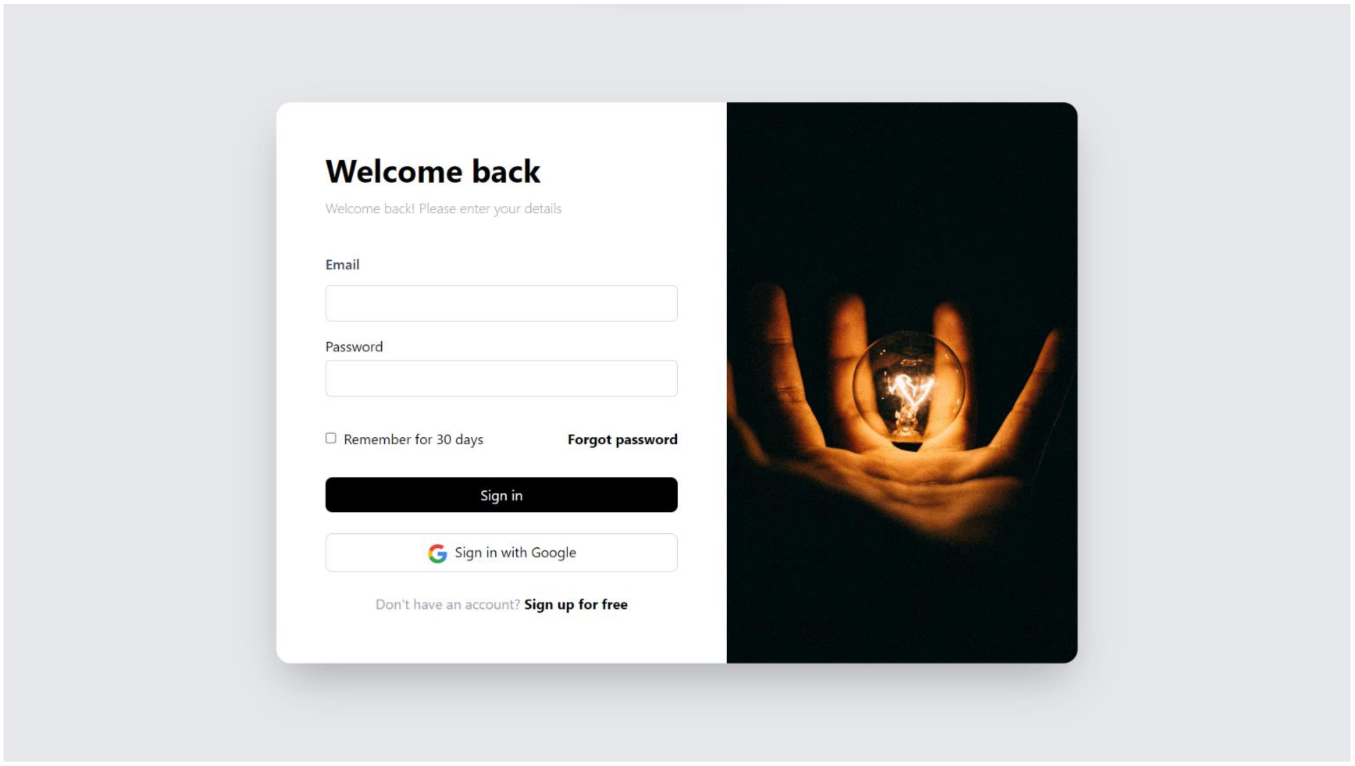


Fig 6.1 USER LOGIN PAGE

6.1.2. ADMIN LOGIN PAGE

When Admin enters our website. He will be asked about his login details like email id and password. The login details will be verified with the details given while the admin have an default account.

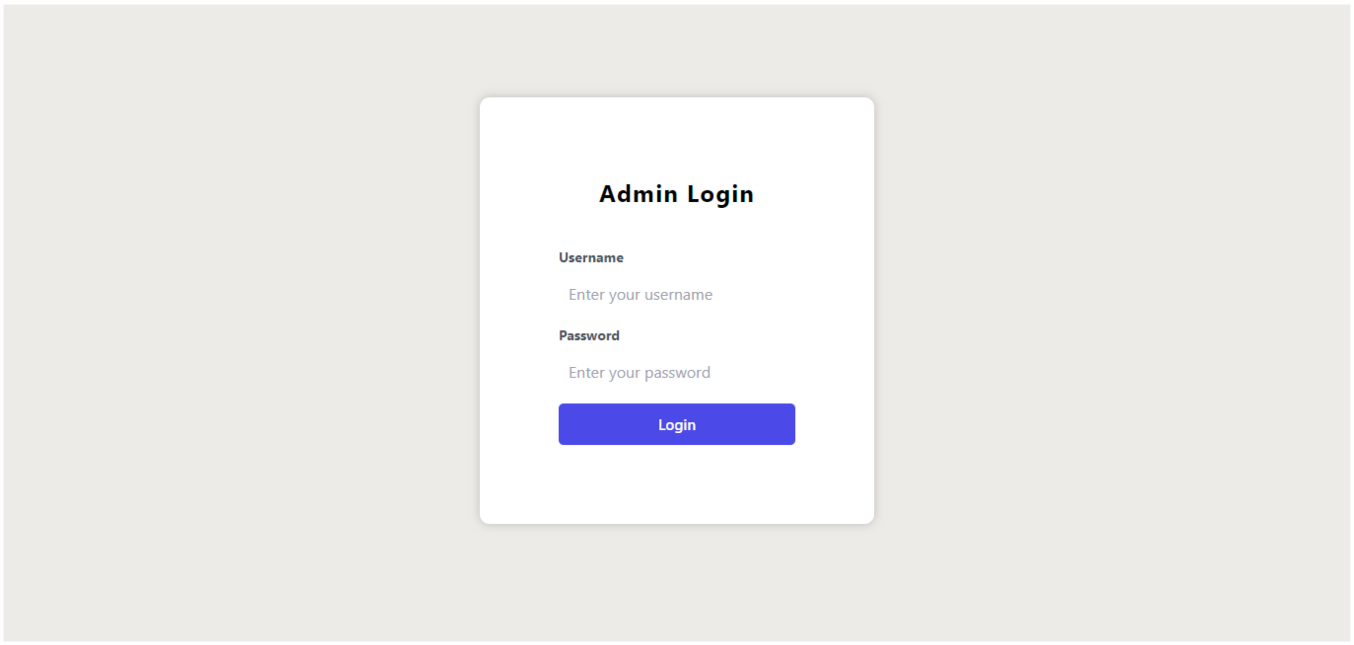


Fig 6.2 ADMIN LOGIN PAGE

6.1.3 SIGN UP PAGE

If a user doesn’t have an account on the website, User can use a component named create new account available in the login page. When the user clicks on that he will be redirected to the signup page. In sign up he should fill up his email id, password and phone number. These inputs will be validated.

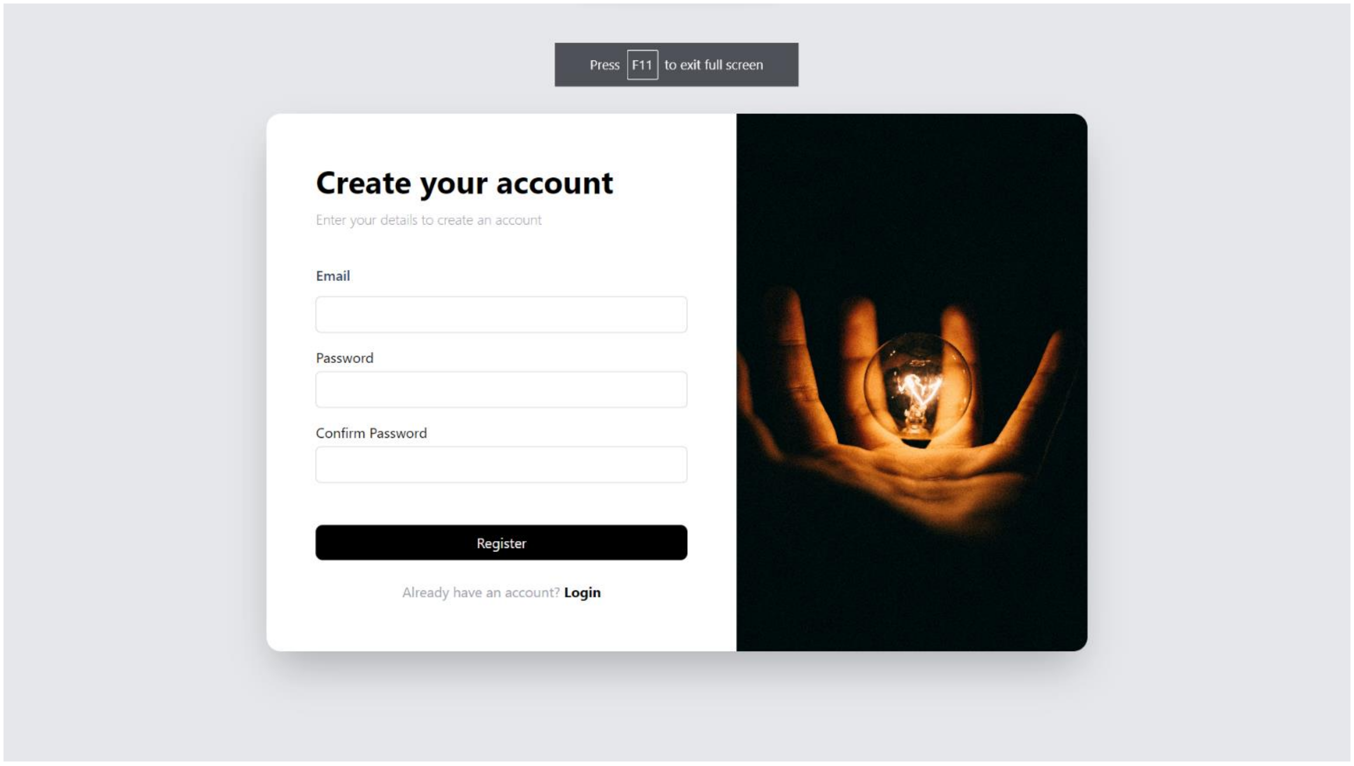
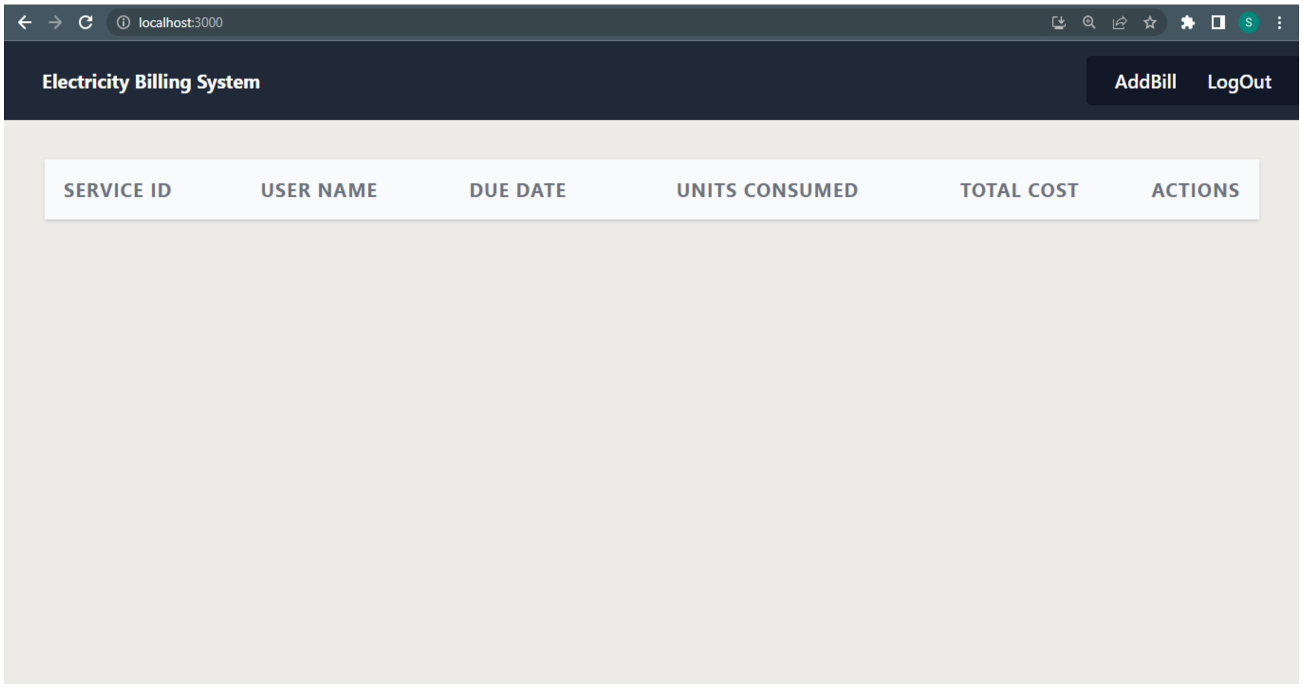

Press F11 to exit full screen
Create your account
Enter your details to create an account
Email
Password
Confirm Password
Register
Already have an account? Login

Fig 6.3 SIGNUP PAGE

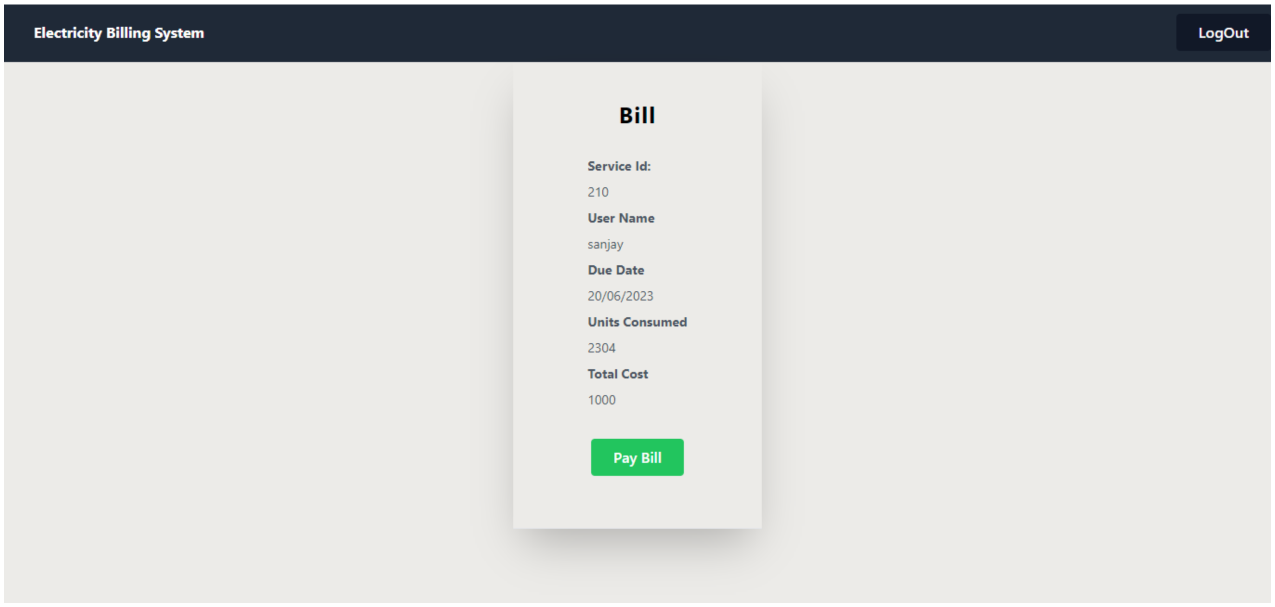
6.1.4 MAIN PAGE

This page displays is used for accessing the admin to added bill for user in to edit and delete the bill .



SERVICE ID	USER NAME	DUE DATE	UNITS CONSUMED	TOTAL COST	ACTIONS

Fig 6.4 HOME PAGE



Bill

Service Id:
210
User Name
sanjay
Due Date
20/06/2023
Units Consumed
2304
Total Cost
1000

Fig 6.5 BILL GENERATE PAGE

6.1.5 ADD NEW BILL PAGE

This page displays the admin to add the bill for user to view their bill and pay their bill.

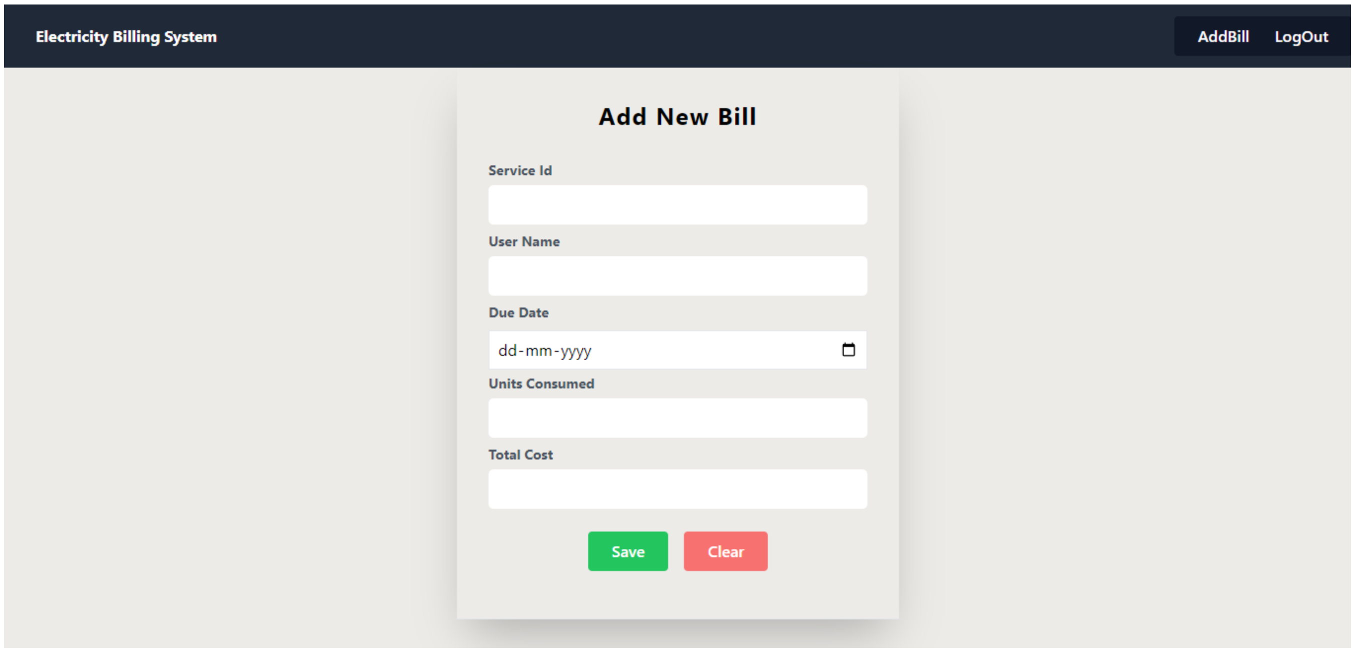


Fig 6.6 ADD NEW BILL PAGE

6.1.6 UPDATE BILL PAGE

This page displays the access for admin to update the bill for the user of electricity bill of user to view the bill.

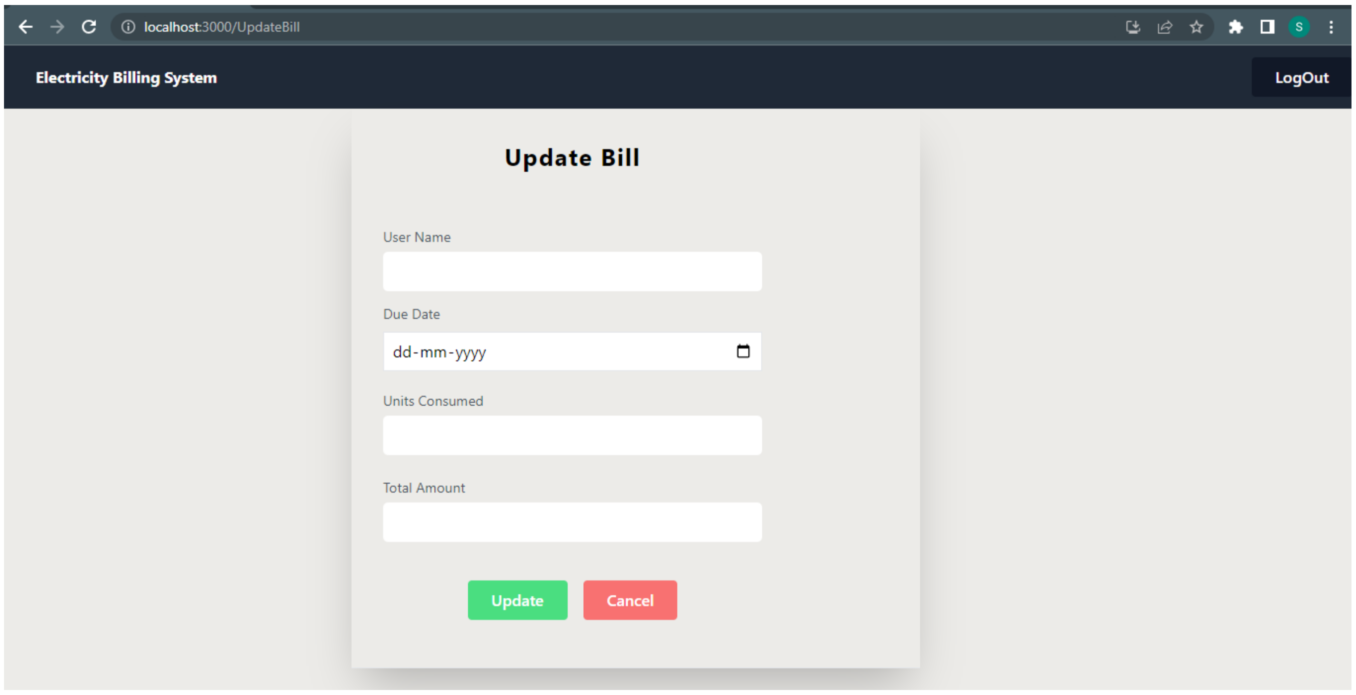


Fig 6.7 UPDATE BILL PAGE

6.1.7 CODING FRONTEND

LOGIN PAGE:

```
const import React from "react";
import GoogleLogo from '../user/google.svg';
import LoginImage from '../user/login.jpg';
import * as Yup from 'yup';
import { useForm } from 'react-hook-form';
import { yupResolver } from '@hookform/resolvers/yup';
export default function Loginuser() {
    const validationSchema = Yup.object().shape({
      email: Yup.string()
        .required('Email is required')
        .email('Email is invalid'),
      password: Yup.string()
        .required('Password is required')
        .min(6, 'Password must be at least 6 characters')
        .max(40, 'Password must not exceed 40 characters'),
    });
  const {
    register,
    handleSubmit,
    formState: { errors }
  } = useForm({
    resolver: yupResolver(validationSchema)
  });
  const onSubmit = (data) => {
    console.log(data.email);
    console.log(data.password);
  };
```

```
 return (
   <div id="background" className="flex items-center justify-center min-h-screen bg-gray-
200">
     <div id="form=box" className="relative flex flex-col m-6 space-y-8 bg-white shadow-2xl
rounded-2xl md:flex-row md:space-y-0">
       <form
         id="form"
         noValidate
         className="flex flex-col justify-center p-8 md:p-14"
         onSubmit={handleSubmit(onSubmit)}
       >
         <div
           id="welcome-text"
           className="flex flex-col"
         >
           <span className="mb-3 text-4xl font-bold">Welcome back</span>
           <span className="font-light text-gray-400 mb-8">Welcome back! Please enter your
details</span>
         </div>
         <div id="email" className="py-2">
           <span className="block text-md font-medium text-slate-700 mb-2">Email</span>
           <input
             autoComplete='off'
             type="text"
             className="peer w-full p-2
             border border-gray-300
             rounded-md"
             {... register('email')}
           />
```

```
<p className="mt-1 invisible peer-invalid:visible text-red-600 border-gray-800">{errors.email?.message}</p>
     </div>
     <div id="password" className="py-1">
      <span className="font-normal mb-2 text-md">Password</span>
      <input
       autoComplete="off"
       type="text"
       className="peer w-full p-2 mt-1 border border-gray-300 rounded-md placeholder:font-light placeholder:text-gray-500 focus:placeholder-gray-500 invalid:border-red-800 invalid:placeholder-red-500 focus:invalid:border-red-800"
id="pass-field"
       name="pass-field"
       required={true}
       {... register('password')}
      />
      <p className="mt-2 invisible peer-invalid:visible text-red-600 text-sm">{errors.password?.message}</p>
     </div>
     <div className="flex justify-between w-full py-4 mt-2">
      <div className="mr-24">
       <input
        type="checkbox"
        name="checkbox"
        id="checkbox"
        className="mr-2 accent-black"
       />
       <span className="text-md">Remember for 30 days</span>
      </div>
```

```
 <a className="font-bold text-base" href="/forgotpass">Forgot password</a>
          </div>
          <button
            type="submit"
            id="sign-in-button"
            className="w-full bg-black text-white p-2 rounded-lg mb-6 mt-4 hover:bg-white
hover:text-black hover:border hover:border-gray-300"
          >
            Sign in
          </button>
          <button
            id="google-button"
            className="w-full border border-gray-300 text-md p-2 rounded-lg mb-6 hover:bg-black
hover:text-white"
          >
            <img
              src={GoogleLogo}
              alt="Google svg"
              className="w-6 h-6 inline mr-2"
            />
            Sign in with Google
          </button>
<div className="text-center text-gray-400">
            Don't have an account?
            <a className="font-bold text-black" href="/signup">  Sign up for free</a>
          </div>
        </form>
        <div id="image" className="relative">
          <img
            src={LoginImage}
            alt='Login img'
            className="w-[400px] h-full hidden rounded-r-2xl md:block object-cover"
          />
        </div>
      </div>
    </div>
   )
}
```

HOME PAGE:

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import {useNavigate} from "react-router-dom";
import Navbar from './Navbar';
const Home = () => {
 const navigate=useNavigate();
const handleDelete=(serviceid)=>{
    let result=window.confirm('Are you sure to delete this record ?');
    if(serviceid){
        axios.delete(`http://localhost:8080/electricity/deleteDetails/${serviceid}`)
        .then(resp=>{
            alert(resp.data.data)
            console.log(resp.data)
        })
        .catch(error=>{
            console.log(error)
        })
    }
}
```

```
const [data, setData] = useState([]);
  useEffect(() => {
    axios.get('http://127.0.0.1:8080/electricity/getall')
      .then(response => {
        setData(response.data);
      })
      .catch(error => {
        console.log(error);
      });
  }, []);

  return (
    <>
    <Navbar/>
    <div className="container mx-auto my-8">
      <div className="flex shadow border-b">
        <table className="min-w-full">
          <thead className="bg-gray-50">
            <tr>
              <th className="text-left font-bold text-gray-500 uppercase tracking-wider py-3 px-
4 ">
                Service Id
              </th>
              <th className="text-left font-bold text-gray-500 uppercase tracking-wider py-3 px-
4">
                User Name
              </th>
```

```
<th className="text-left font-bold text-gray-500 uppercase tracking-wider py-3 px-4">
                Due Date
              </th>
              <th className="text-center font-bold text-gray-500 uppercase tracking-wider py-3
px-2">
                Units Consumed
              </th>
              <th className="text-right font-bold text-gray-500 uppercase tracking-wider py-3 px-
2">
                Total Cost
              </th>
 <th className=" text-right font-bold text-gray-500 uppercase tracking-wider py-3 px-4">
                Actions
              </th>
            </tr>
          </thead>
          <tbody className="bg-white">
            {data.map(user => (
              <tr key={user.id}>
                <td className="text-left px-6 py-4 whitespace-nowrap">
                  <div className="text-sm text-gray-500">{user.serviceid}</div>
                </td>
                <td className="text-left px-6 py-4 whitespace-nowrap">
                  <div className="text-sm text-gray-500">{user.username}</div>
                </td>
                <td className="text-left px-6 py-4 whitespace-nowrap">
                  <div className="text-sm text-gray-500">{user.duedate}</div>
                </td>
```

```
  <td className="text-right px-6 py-4 whitespace-nowrap">
                  <div className="text-sm text-gray-500">{user.unitsconsumed}</div>
                </td>
                <td className="text-right px-2 py-4 whitespace-nowrap">
                  <div className="text-sm text-gray-500">{user.totalamount}</div>
                </td>
                <td className="text-right px-6 py-4 whitespace-nowrap font-medium text-sm">
                  <a onClick={()=> navigate("/UpdateBill")} className="text-indigo-600
hover:text-indigo-600 px-4 hover:cursor-pointer">Edit</a>
                  <a onClick={e=>handleDelete(user.serviceid)} className= " text-indigo-600
hover:text-indigo-600 px-4 hover:cursor-pointer">Delete</a>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
    </>
  );
};

export default Home;
```

6.2 CODING BACKEND

ELECTRICITY CONTROLLER:

```
package com.example.demo.controller;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.model.Electricity;
import com.example.demo.services.ElectricityService;
@CrossOrigin
@RequestMapping("/electricity")
@RestController


public class ElectricityController {
    @Autowired
    ElectricityService ls;
    @PostMapping("/save")
    public Electricity saveinfo(@RequestBody Electricity l)
    {
            return ls.create(l);
    }
    @GetMapping("/getall")
    public List<Electricity> getall()
    {
            return ls.getalldetails();
    }
@GetMapping(value = "/get/{serviceid}")
    public Optional<Electricity> getDetails(@PathVariable("serviceid") int serviceid)
    {
            return ls.getDetails(serviceid);
    }
    @PutMapping(value = "/updatedetails/{serviceid}")
    public String updateDetailsById(@RequestBody Electricity
em,@PathVariable("serviceid") int serviceid)
    {
            return ls.updateDetails(em,serviceid);
    }
    @DeleteMapping(value = "/deleteDetails/{serviceid}")
    public String deleteDetails(@PathVariable("serviceid") int serviceid)
    {
            ls.deleteDetails(serviceid);
            return "Service with ID " + serviceid + " is deleted";
}
```

ELECTRICITY SERVICE:

```
package com.example.demo.services;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.data.domain.Sort.Direction;
import org.springframework.stereotype.Service;
import com.example.demo.model.Electricity;
import com.example.demo.repository.ElectricityRepo;
@Service
public class ElectricityServiceImpl implements ElectricityService {
    @Autowired
    ElectricityRepo lr;
@Override
    public Electricity create(Electricity e)
    {
            return lr.save(e);
    }
    @Override
   public List<Electricity>getalldetails()
   {
    return lr.findAll();
   }
}

@Override
   public Optional<Electricity> getDetails(int serviceid)
    {
            return lr.findById(serviceid);
    }
    @Override
   public String updateDetails(Electricity i,int serviceid)
    {
    Electricity e = lr.findById(serviceid).orElse(null);
              if(e != null){

               return "The details of the  ID was updated";
              }
              else{
               return "The ID is not present in the database to update the value";
              }
             }
    @Override
    public void deleteDetails(int serviceid)
    {
            lr.deleteById(serviceid);
  }
```

ELECTRICITY REPOSITORY:

```
    package com.example.demo.repository;
    import org.springframework.data.jpa.repository.JpaRepository;
    import com.example.demo.model.Electricity;
    public interface ElectricityRepo extends JpaRepository<Electricity, Integer>{
}
```

CONCLUSION

CHAPTER 7
CONCLUSION

This chapter tells about the conclusion that anyone can drive from the project and the learning we learnt by taking over this project.

7.1 CONCLUSION

The development of an electricity billing system where the admin can add data and users can view their bills brings numerous benefits to utility providers and customers alike. This system streamlines data management, ensures accurate billing calculations, provides real-time updates, and offers a user-friendly interface for users to access their billing information. By implementing robust security measures, the system safeguards customer data and enhances trust. The methodologies involved, such as requirement gathering, system design, user interface design, database design, development, and testing, enable the successful implementation of such a system. Overall, this electricity billing system improves efficiency, transparency, and customer satisfaction, making it a valuable tool for both the admin and users in managing and monitoring electricity consumption.

7.2 FUTURE SCOPE

Customized Integration with Smart Metering: Integrating the system with smart metering technology can enable automatic meter reading and real-time data synchronization. This integration would eliminate the need for manual meter readings and provide users with more accurate and up-to-date billing information.

Energy Consumption Analytics: Adding analytics features to the system can provide users with insights into their energy consumption patterns. Users can access visualizations and reports that highlight peak usage periods, energy-saving recommendations, and comparisons with similar households. This information can encourage energy conservation and efficiency.

Automated Billing Notifications: Implementing automated billing notifications can alert users about upcoming due dates and payment reminders. Users can receive notifications through email, SMS, or mobile app notifications, ensuring timely bill payments and reducing the chances of late fees or disconnections.

Integration with Online Payment Gateways: Integrating the system with popular online payment gateways would enable users to conveniently pay their bills directly through the system. This integration would streamline the payment process and provide users with various secure payment options.

REFERENCES

REFERENCES

1. Mandal, J. K., Sarkar, B., & Jha, S. (2014). Review on electricity billing system and development of billing software.
2. Babu, B. R., & Reddy, G. V. (2018). Electricity billing system using blockchain technology.
3. Alotaibi, S., & Khalifa, O. O. (2018). An intelligent electricity billing system using artificial neural network.
4. Hosseinzadeh. M, & Mohamadian. M (2012). An innovative approach for electricity billing system based on power line communication.
5. Tahir, M., Shah, T. A., & Javaid, N. (2015). A secure electricity billing system for smart grid using wireless sensor network.