بِسم الله الرَحمن الرَحيم

Palestine Polytechnic University

College of Information Technology and Computer Engineering

Graduation Project

"Website for Hebron Electric Power Company"

Prepared By

Arwa Arafah Leen Maraqa

Supervised By

Ezdehar Jawabrah

This project was presented to fulfill the requirements of the graduation project in the specialization of Computer Science at the College of Information Technology and Computer Engineering.

2022/2023

Acknowledgement

We are humbled and grateful to Allah for giving us the strength and patience to complete this work, and a huge thanks to our supervisor Mrs.Ezdehar for the insights and guidance she provides. Also, it is our pleasure to work on such a project for our college including all faculty members. Finally, thanks to the company members, we appreciate the support for promoting the realization of the full potential.

Abstract

The aim of Hebron Electricity Company is to provide the best electrical services to the community. To achieve this vision, we have developed a website that aims to facilitate and enhance the customer experience in dealing with the company. The website enables subscribers of Hebron Electricity Company to access a variety of electric services. Customers can conveniently request services such as meter inspection, pole relocation, and updating their personal information, among others. Additionally, customers can view details of their past and current requests, as well as check the amounts due from their bills, allowing them to monitor and track the status of their requests and ensure their proper execution with minimal effort.

Furthermore, the website offers employees a platform to review and process customer requests online, streamlining the response and handling of inquiries. Employees can also leverage the website to disseminate the latest news and reports. The electronic services provided by Hebron Electricity Company deliver convenience and ease to customers in their electricity-related procedures. Ultimately, the website contributes to the improvement of electronic services and fosters stronger customer-company relationships.

تهدف شركة كهرباء الخليل إلى توفير أفضل الخدمات الكهربائية للمجتمع. ولتحقيق هذه الرؤية، قمنا بتطوير موقع إلكتروني يهدف إلى تسهيل وتحسين تجربة العملاء في التعامل مع الشركة. يتيح الموقع للمشتركين في شركة كهرباء الخليل الوصول إلى مجموعة متنوعة من الخدمات الإلكترونية. يمكن للعملاء الآن تقديم طلبات الخدمة المختلفة مثل طلب فحص العداد ونقل الأعمدة وتغيير بياناتهم الشخصية والمزيد. أيضا أصبح بإمكان العملاء عرض تفاصيل الطلبات السابقة والحالية ومعرفة المبالغ المستحقة من الفواتير، مما يتيح لهم مراقبة ومتابعة حالة طلباتهم وضمان تنفيذها بشكل صحيح وبأقل جهد ممكن.

بالإضافة إلى ذلك، يوفر الموقع للموظفين وسيلة لمراجعة هذه الطلبات والمعاملات عبر الإنترنت، مما يسهل ويسرع عملية المعالجة والاستجابة لطلبات العملاء. كما يمكن للموظفين استخدام الموقع لنشر آخر الأخبار والتقارير وتبادل المعلومات الهامة. وتعتبر خدمات شركة كهرباء الخليل الإلكترونية مفيدة للعملاء حيث توفر لهم الراحة والسهولة في إجراءاتهم المتعلقة بالكهرباء. كما ساهم موقعنا الاكتروني في تحسين الخدمات الإلكترونية وتعزيز العلاقة بين العملاء والشركة.

Table of contents:

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
HEPCo	Hebron Electric Power Company
CSE	Customer Service Employee
PRE	Public Relations Employee
ITDE	IT Department Employee
JWT	JSON Web Token
MVC	Model View Controller

Chapter 1: Introduction

1.1 Overview

In this chapter, we provide an overview of our project, including a detailed description, key performance indicators, problem statement, and the final results of the project.

1.2 Description of the system

The project entails the development of a website for HEPCo, the primary electricity provider serving residents of Hebron and Halhul. The website aims to inform citizens about the available services offered by the company and provide them with the convenience of placing orders online. This eliminates the need for citizens to physically visit the company's premises, as they can now access and manage their service requests through the website. Furthermore, the website enables citizens to view their installment plans and invoices, offering them easy access to their payment information.

1.3 Problem Statement

1.3.1 Description of the current website

The current website of HEPCo is one of the sites that does not allow the customer to interact with the site. The customer can only see the services provided by the company, but she/he cannot manage to get a service or contact with the company

1.3.2 Suggested Improvements

1. The customer will be able to send requests through the website, such as:
    a. electricity meter inspection.
    b. change the payment mechanism from bill to card.
    c. street lighting installation.
2. The Customer will be able to view his previous installments and bills.
3. The Customer will be able to track his request status.

1.4 The Significance and Motivation

1. There is no official website for the customers to deal with the company.
2. HEPCo is the main provider of electricity to the citizens of Hebron, so it needs a website to facilitate communication between customers and the company.
3. Spreading the culture of online services in the community.
4. Improve and develop the services provided to the customers.
5. It will be easier for the customers to submit a request for any service they need and follow up on the status of their request online.

1.5 Aims and Objectives

1. Provide a platform for customers to get the services provided by HEPCo via the website, which saves so much time and effort.
2. Provide a platform for PRE to publish news and advertisements instead of posting them on social media.
3. Saving time and effort for the CSE, so that they can easily review customers' complaints and requests via the Internet.
4. Made a new design with a user-friendly interface.
5. Give customers an easy way to track their installments, bills and requests.
6. Facilitate the management of customers data and save their privacy.

1.6 Context Diagram

Context models are utilized to depict the operational environment of a system, revealing the elements existing beyond the system's boundaries. The model showcases the interconnections between the electricity system and the adjacent systems in the project, as depicted in Figure 1.1.

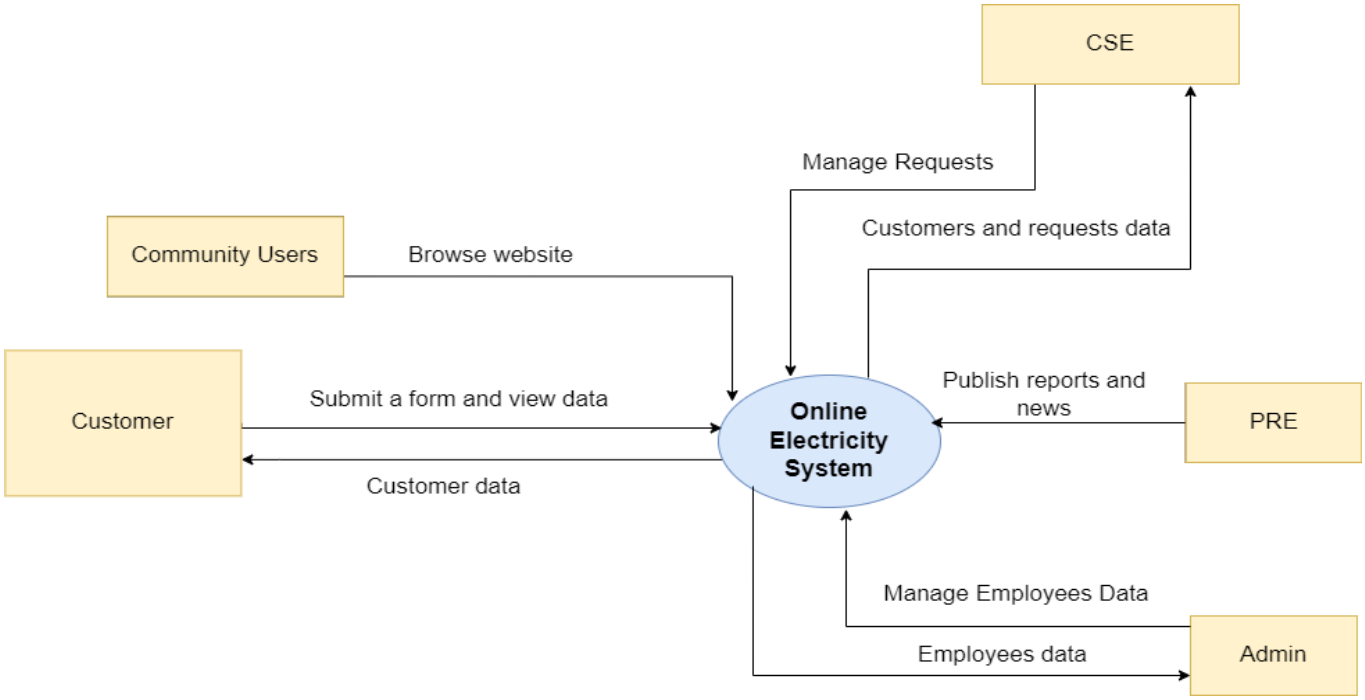

Figure 1.1 Context diagram

1.7 Project Scope

Our aim is to produce a new proposed comprehensive website for the Hebron Electricity Power Company (HEPCo) that will be beneficial to both the citizens and company employees. As a result, that will increase the usage and need for the website for the citizens, at the same time leads to efficient communication between people and employees serving them the required functionality.

1.8 Implementation Alternatives

The primary goal of Hebron Electricity Company is to offer services to a wide range of citizens. To fulfill this objective, we developed a Custom-Coded website from scratch specifically to meet the company's needs, the platform includes frequently used features to facilitate the process for citizens to apply for any service remotely from the website.After careful consideration, we concluded that an interactive web application based on transactions would be superior to the mobile application in fulfilling all the essential requirements necessary for the system.

1.9 Project Scheduling

Table 1.1 Project tasks

Activity/Details	Time Spent
System definition and planning	3 weeks
Analyze system requirements	6 weeks
System design	4 weeks
System implementation	3 weeks
System testing	2 weeks
Documenting	Along the working period

Chapter 2: Functional and non-functional requirements

2.1 Overview

In this chapter, we will discuss the functional requirements for the end-user's side, the employee's side and technical admin aspects of the project. Additionally, we will address the non-functional requirements that are crucial for the project's implementation.

2.2 Functional Requirements

Functional requirements are typically defined and documented in collaboration with electricity company employees. They help to ensure that the software system meets the needs and provides the desired level of functionality and usability. The main actors of the system and their requirements are as follows.

2.2.1 Admin

The Admin is basically the IT Department Employee (ITDE):

1. create a new employee.
2. archive an employee.

2.2.2 Public Relationships Employee

1. Create advertisements and news.
2. Create reports.

2.2.3 Customer

1. The customer will be able to submit one of the main requests, which are as follows:
    a. check the electricity meter.
    b. change the subscription type from bill to card.
    c. install street lighting.
    d. modify the type of property from commercial to home.
2. Submit a request to change the subscription status from temporary to permanent.
3. Submit a request to transfer the poles and networks opposing construction and property.
4. Submit a request to modify the beneficiary's data (Tenant's name).

5. Submit one of the secondary requests, which are as follows:
    a. solve the problem of weak electricity.
    b. maintenance of equipment related to electrical installations.
    c. reduce the old debt installments.
    d. objection to the estimated amount of consumption during the period of failure of the electricity meter.
6. View his previous requests.
7. View his previous bills.
8. View his previous installments.
9. View all his subscriptions and the information about each subscription.

2.2.4 Customer Services Department Employees (CSE)

1. View all requests.
2. Search for the request by the customer's name.
3. Change request status.
4. View subscriber information.

2.3 Non-Functional Requirements

Product Requirements:

1. The system is available for all customers who will be able to apply for any request.
2. The system provides high levels of security and privacy by using encryption algorithms and time-dedicated tokens.
3. The system must obtain the best user experience, so users can deal with it easily .
4. The ability to log in to the system with a valid username and password.

2.4 Technical admin's functional requirements tables

Admin creates new employee

Table 2.5.1 Admin creates new employee

Requirement	Create a new employee with his/her information in the system
Actor	Admin
Objective	Admin can create an employee and fill all the required fields, the employee will be given a password for the login process.
Precondition	Must be specified in the system as admin
Scenario	Admin clicks "new employee" button on the employees' profiles section and add the employee's data (e.g: password, name, role)
Exceptions	No internet connection

Admin view all employees' data

Table 2.5.2 Admin view all employees' data

Requirement	Admin view all employees' data
Actors	Admin
Objective	Admin views a list of current employees and their data including role, name and phone number.
Precondition	Must be specified in the system as admin
Scenario	Admin clicks on the 'employees' button on the sidebar menu then a table of all employees data will appear from the table.
Exceptions	No internet connection

Admin archives an employee

Table 2.5.3 Admin archives an employee

Requirement	Archive an employee which will prevent them from appearing in the admin interface
Actors	Admin
Objective	Admin archives an employee with their corresponding data
Precondition	Must be specified in the system as admin
Scenario	Admin clicks on 'Archive' button located in each row of the employees data table, then the archived employee will disappear.
Exceptions	No internet connection

2.6 Public Relationships Employee's functional requirements tables (PRE)

PRE creates a new advertisement or company related news

Table 2.6.1 PRE creates an advertisement

Requirement	Create an advertisement or news
Actors	PRE
Objective	PRE creates an advertisement of a specific topic, or company related news
Precondition	Must be specified in the system as a PRE
Scenario	PRE specifies the advertisement or news body, picture and date
Exceptions	No internet connection

PRE creates a new report

Table 2.6.2 PRE creates an annual report

Requirement	Create an annual report
Actors	PRE
Objective	PRE creates a report of an issue
Precondition	Must be specified in the system as a PRE
Scenario	PRE specifies the report body, employee signature and date
Exceptions	No internet connection

2.7 Customer functional requirements tables

Submit the main request

Table 2.7.1 Submit the main request

Requirement	Submit one of the main requests that were mentioned on section 2.2.3
Actor	Customer
Objective	Create a new request for one of the main request
Precondition	The customer must be logged in the system
Scenario	1. Click on the services on the home page in the header section. 2. Select one of the main requests. 3. Fill the form with the required details for example applicant's name, applicant's phone number, address and service ID. 4. Click submit.
Exceptions	1. No internet connection 2. The Customer does not fill in all the required details.

View previous installments

Table 2.7.2 View previous installments

Requirement	View previous installments
Actor	Customer
Objective	Enable the Customer to view previous installments
Precondition	The customer must be logged in the system
Scenario	Click on my installments on the home page in the header section
Exceptions	No internet connection

View previous requests

Table 2.7.3 View previous requests

Requirement	View previous requests
Actor	Customer
Objective	Enable the Customer to view his previous requests
Precondition	The customer must be logged in the system
Scenario	Click on my requests on the home page in the header section
Exceptions	No internet connection

Submit a request to change the subscription status from temporary to permanent.

Table 2.7.4 Change subscription status

Requirement	Change the subscription status from temporary to permanent
Actor	Customer
Objective	Send a request to HEPCo to inform them that the customer needs to change the subscription status from temporary to permanent
Precondition	The customer must be logged in the system
Scenario	1. Click on the services on the home page in the header section 2. Select change the subscription status from temporary to permanent 3. Fill the form with the required details for example applicant's name, applicant's phone number, address clarification of the reason for the request, service ID, electrical engineer name and electrical engineer phone number. 4. Click submit.
Exceptions	1. No internet connection 2. The Customer does not fill in all the required details.

Applying for transfer of poles and networks opposing construction and property

Table 2.7.5 Transfer poles and networks

Requirement	Transfer the poles and networks opposing construction and property
Actor	Customer
Objective	Send a request to HEPCo to inform them that the customer needs to transfer the poles and networks opposing construction and property
Precondition	The customer must be logged in the system
Scenario	1. Click on the services on the home page in the header section 2. Click on the request for transferring poles and networks 3. Fill the form with the required details for example applicant's name, applicant's phone number, clarification of the reason for the request, service ID, address, image showing the problem and footprint. 4. Click submit.
Exceptions	1. No internet connection 2. The Customer does not fill in all the required details.

Request for modifying beneficiary's data

Table 2.7.6 Request for modifying beneficiary's data

Requirement	Modify the beneficiary's data
Actor	Customer
Objective	Send a request to HEPCo to inform them that the customer needs to modify the beneficiary's data
Precondition	The customer must be logged in the system
Scenario	1. Click on the services on the home page in the header section 2. Click on the request for transferring poles and networks 3. Fill the form with the required details for example applicant's name, applicant's phone number, service ID, address, beneficiary's name, customer ID image and beneficiary's ID image 4. Click submit.
Exceptions	1. No internet connection 2. The Customer does not fill in all the required details.

View previous bills

Table 2.7.7 View previous bills

Requirement	View previous bills
Actor	Customer
Objective	Enable the Customer to view his previous bills
Precondition	The customer must be logged in the system
Scenario	Click on my bills on the home page in the header section
Exceptions	No internet connection

Submit a request for one of the secondary requests

Table 2.7.8 Submit the secondary request

Requirement	Submit one of the secondary requests that were mentioned on 2.3.3
Actor	Customer
Objective	Create a request for one of the secondary requests
Precondition	The customer must be logged in the system
Scenario	1. Click on the services on the home page in the header section. 2. Select one of the main requests. 3. Fill the form with the required details for example applicant's name, applicant's phone number, address, clarification of the reason for the request and service ID. 4. Click submit.
Exceptions	The Customer does not fill in all the required details.

View all subscriptions and the information about each subscription

Table 2.7.9 View all subscriptions

Requirement	view all subscriptions
Actor	Customer
Objective	Enable the Customer view all subscriptions
Precondition	The customer must be logged in the system
Scenario	Click on my subscriptions on the home page in the header section
Exceptions	No internet connection

2.8 CSE functional requirements tables

View all requests

Table 2.8.1 View all requests

Requirement	View all requests
Actor	CSE
Objective	Enable the CSE to review customer requests
Precondition	Must be specified in the system as CSE
Scenario	Click on the ‘requests’ button on the sidebar menu
Exception	No internet connection.

Search for the request by the customer's name.

Table 2.8.2 Search for a request

Requirement	Search for the request
Actor	CSE
Objective	Enable the CSE to search and filter customer requests
Precondition	Must be specified in the system as CSE
Scenario	Click on customer requests The CSE enters the customer name to search for his requests
Exception	No internet connection.

Update the status of the request

Table 2.8.3 Update request status

Requirement	CSE is able to track the request status and update it constantly
Actor	CSE
Objective	Enable the CSE to update the status of the request
Precondition	Must be specified in the system as CSE
Scenario	Click on request status input that appears on the requests table as a column, then change the status from new to pending or rejected
Exceptions	No internet connection.

View all subscribers.

Table 2.8.4 View all subscribers

Requirement	View all subscribers in the system and their data
Actor	CSE
Objective	Enable the CSE to view a list of all customers data and their subscriptions details .
Precondition	Must be specified in the system as CSE
Scenario	Click on ‘customers’ button on the side bar menu.
Exceptions	No internet connection.

2.9 State Diagram

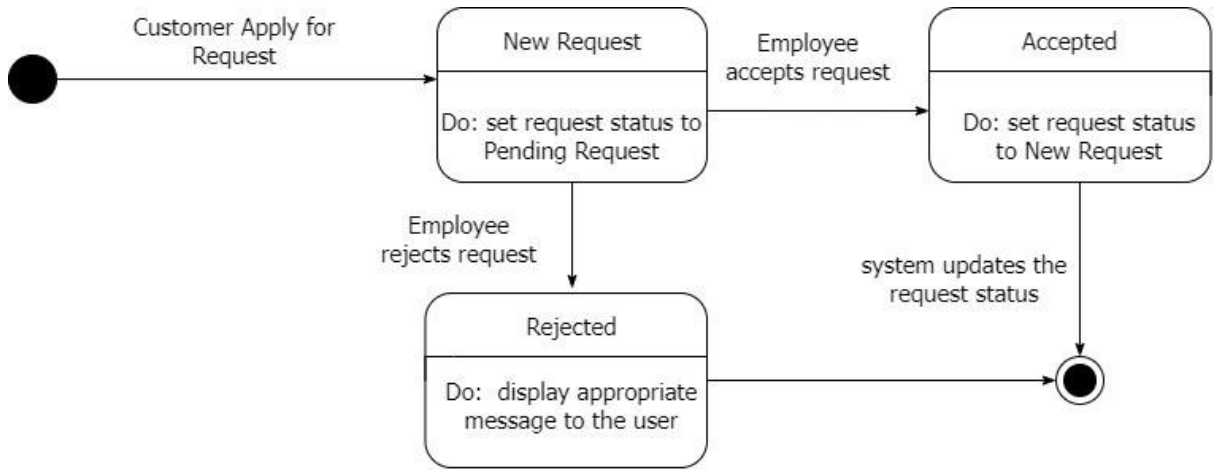

Figure 2.9 State diagram

2.10 Class Diagram

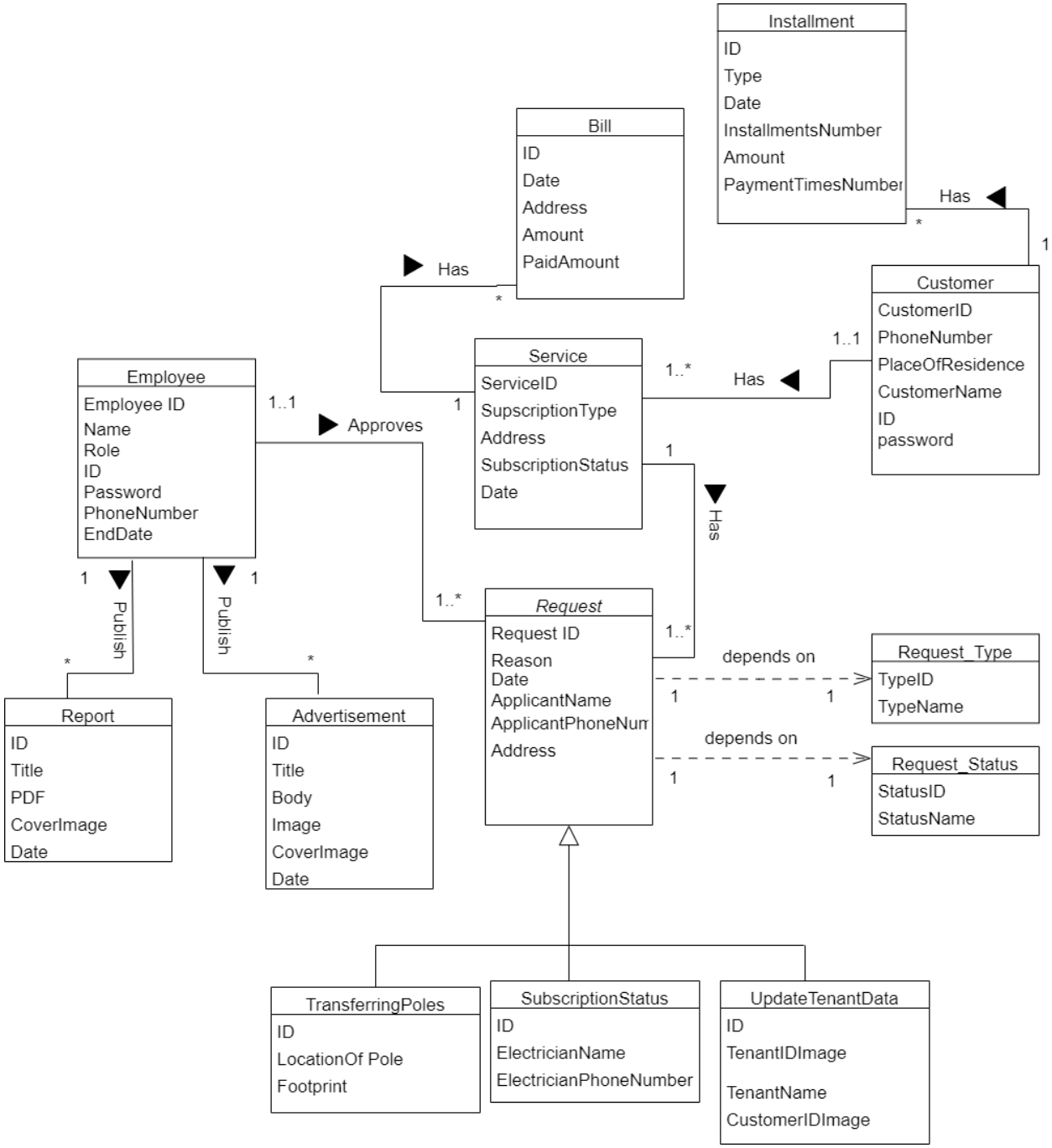

Figure 2.10: Class diagram

2.11 Use Case Diagram

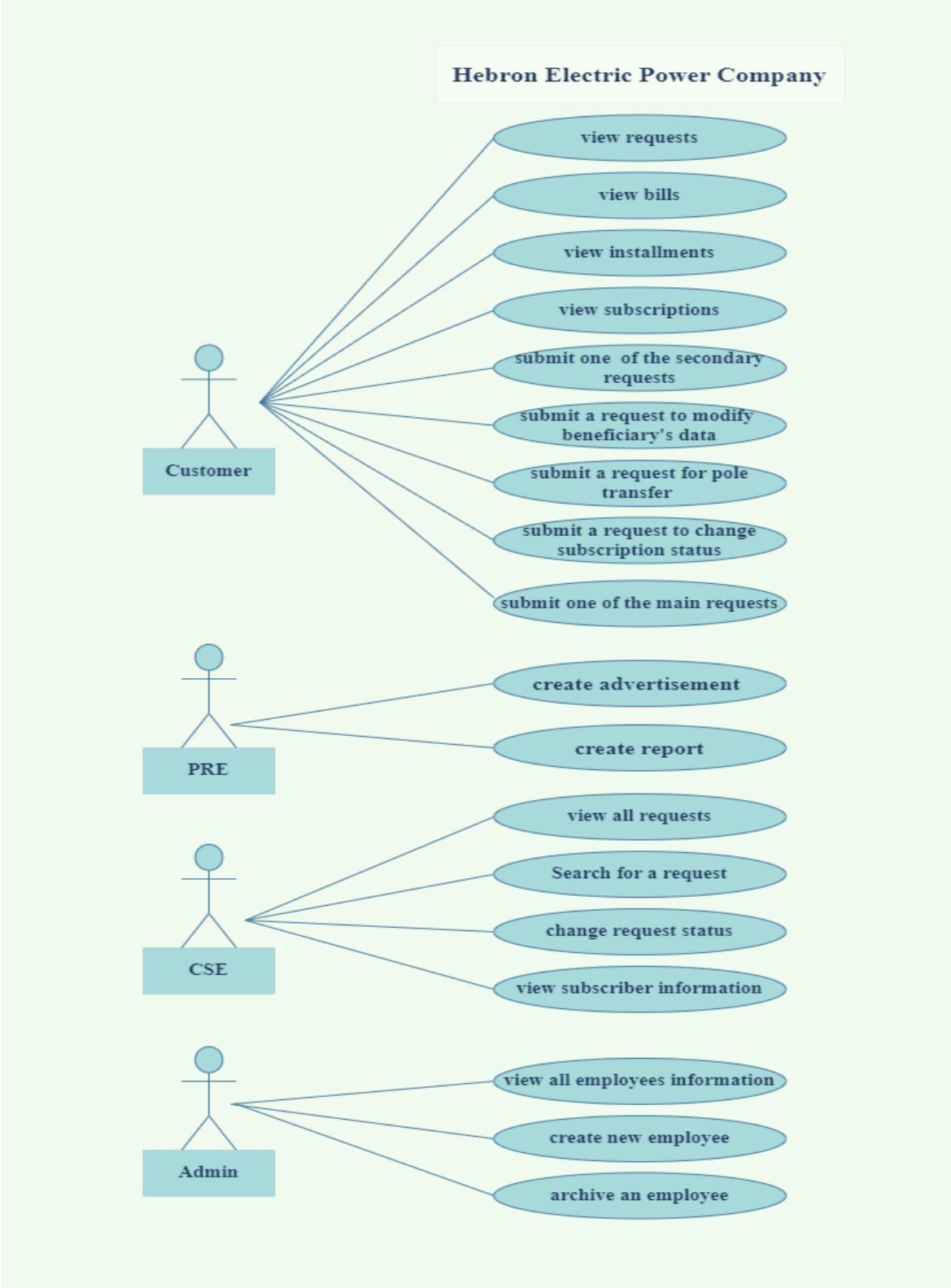

Figure 2.11 Use case diagram.

2.12 Sequence Diagram

2.12.1 Sequence Diagram For Accepting/Rejecting a Customer's request

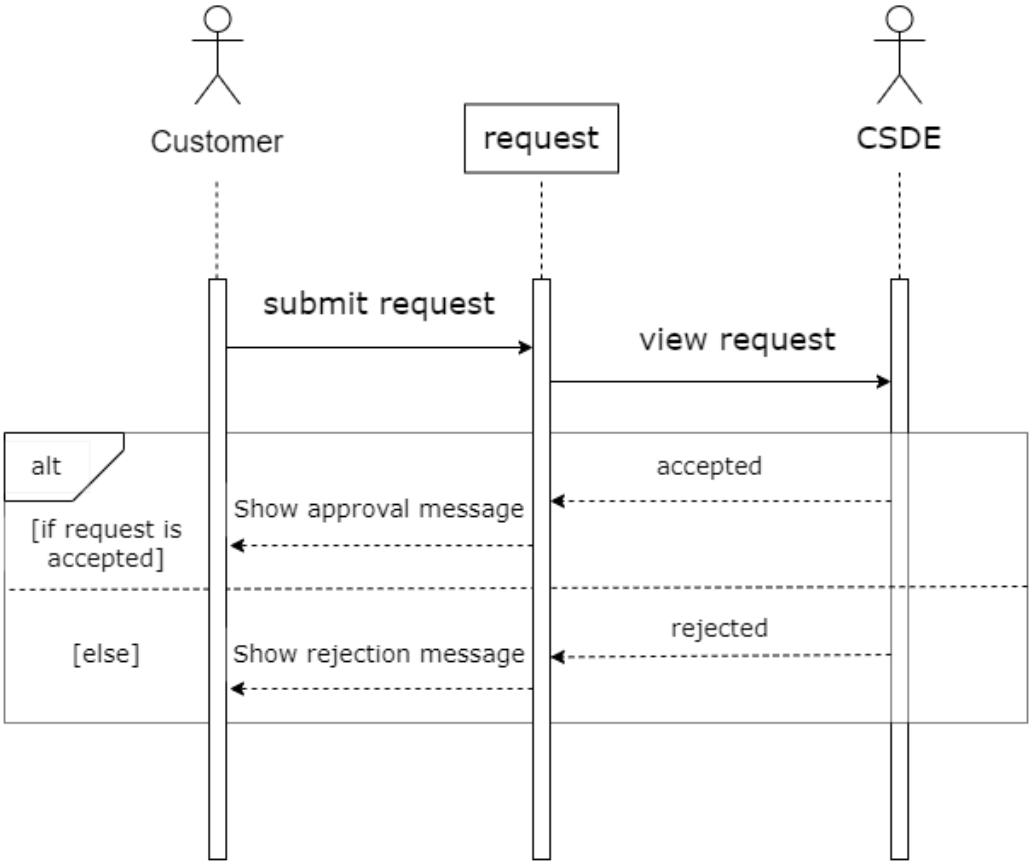

Figure 2.12.1 Sequence diagram for accepting/rejecting a customer's request.

2.12.2 Sequence Diagram For Transferring Poles request

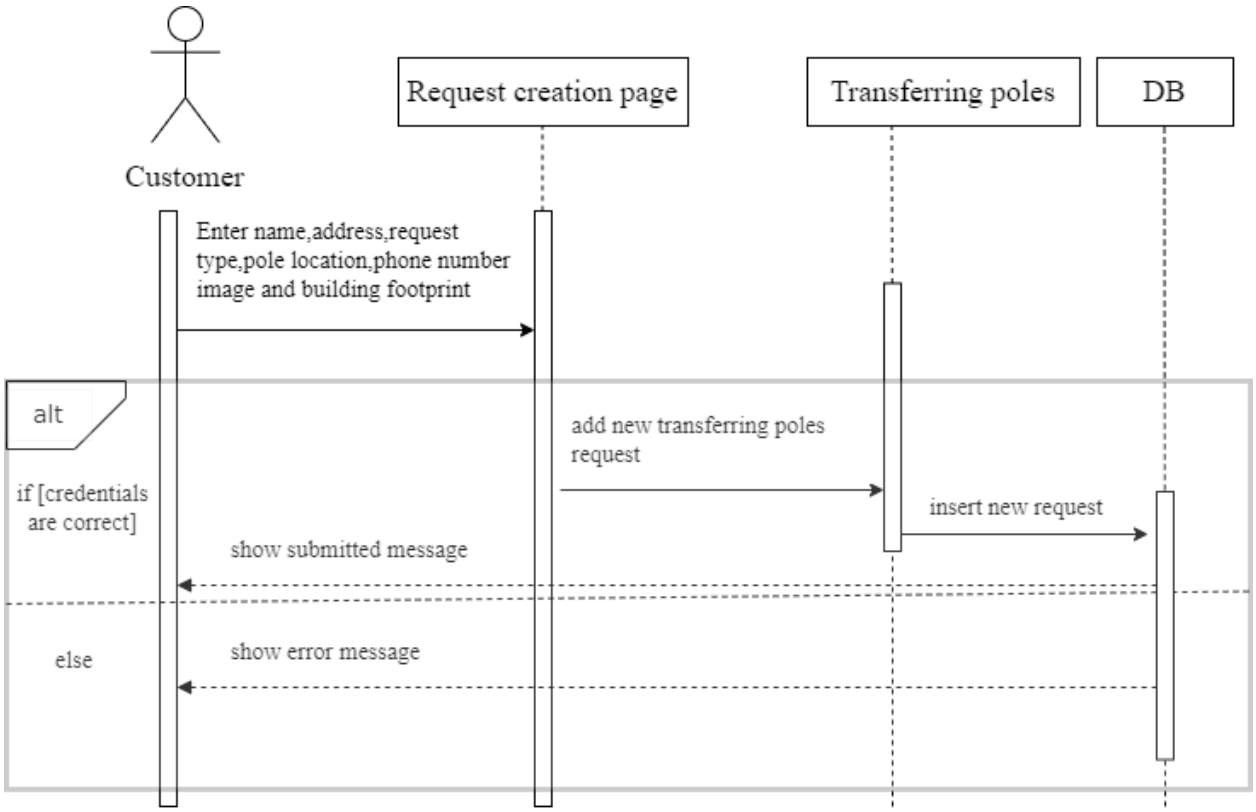

Figure 2.12.2 Sequence diagram for transferring poles request.

2.12.3 Sequence Diagram of View Subscriptions request

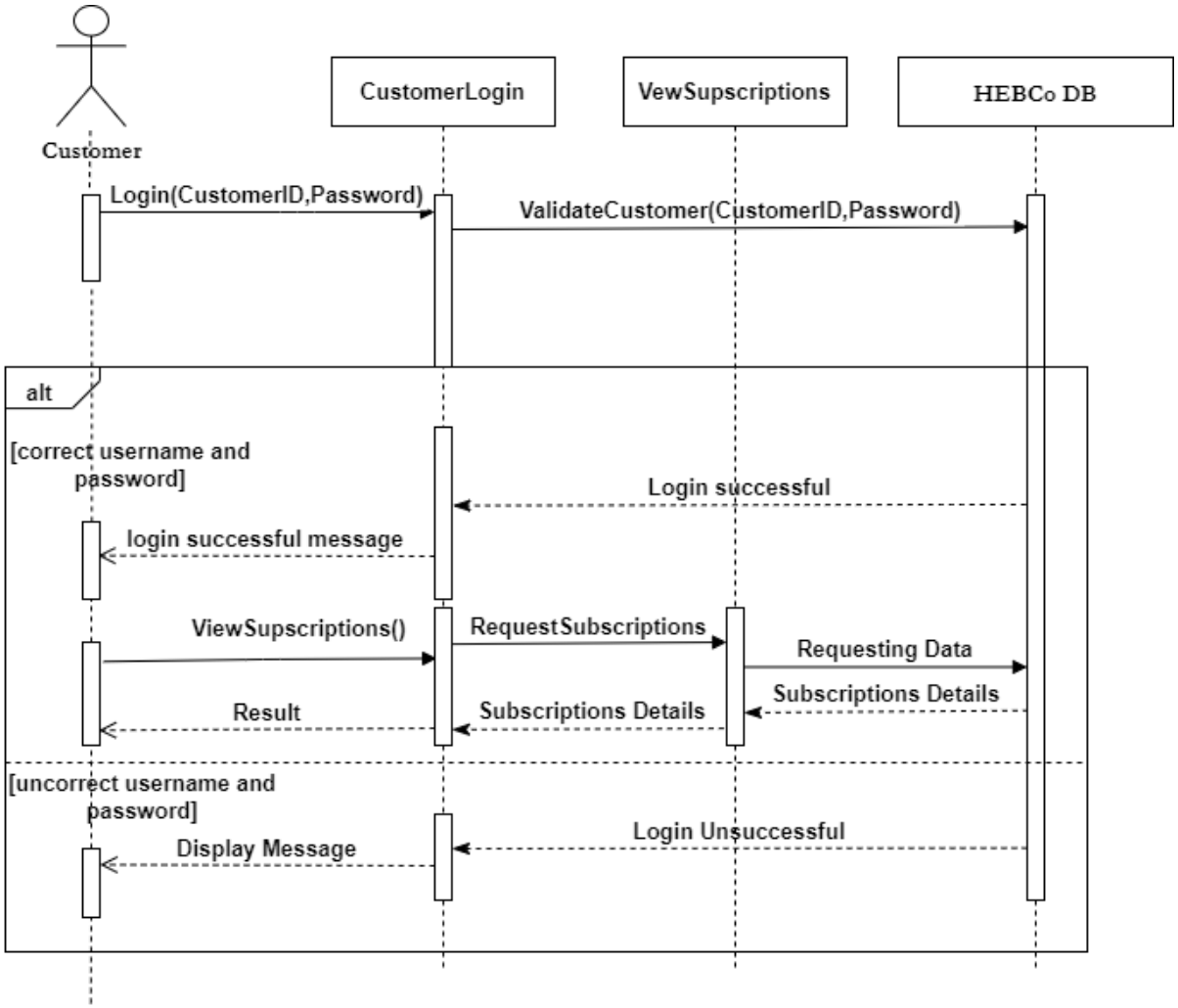

Figure 2.12.3 Sequence diagram of view subscriptions request.

Chapter 3: System Design

3.1 Overview

In this chapter, we will discuss the system architecture and design pattern it uses, database tables, normalized form of the relationship between tables, and a view of the intefaces.

3.2 Introduction

This chapter will focus on discussing the System model and its various components, as well as the architecture employed in our project. We will also explore an alternative architecture. Additionally, we will delve into the database section, covering topics such as mapping and designing. The significance of this chapter lies in its ability to provide programmers with a comprehensive understanding of the system's components. By doing so, it facilitates programming tasks and enhances comprehension of the intercommunication among different project elements.

3.3 Design Alternatives

As a web application it is considered to be a transaction-based application which might use layered architectural pattern .

- **Layered advantages**

  It organizes the subsystems into layers each of which provide a set of services to the layer above it, it is recommended when making updates and adding new facilities and for multi-level security.

- **Layered disadvantages**

  Separation between layers is often difficult. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

- **Why choose MVC?**

  The model and view components operate independently, ensuring a clear separation between data presentation and data logic. This separation greatly simplifies the development of complex applications. In case the model encounters an error, the controller takes charge by instructing the view to generate an appropriate presentation for the error, which is then delivered to the user.

3.4 MVC Architectural Pattern

MVC stands for Model-View-Controller, which is a software architectural pattern commonly used in web development. It separates the application's logic and components into three interconnected components as shown in figure 3.4.1.

3.4.1 The system architecture (MVC) components

Main components are:

1. View.
2. Model.
3. Controller.

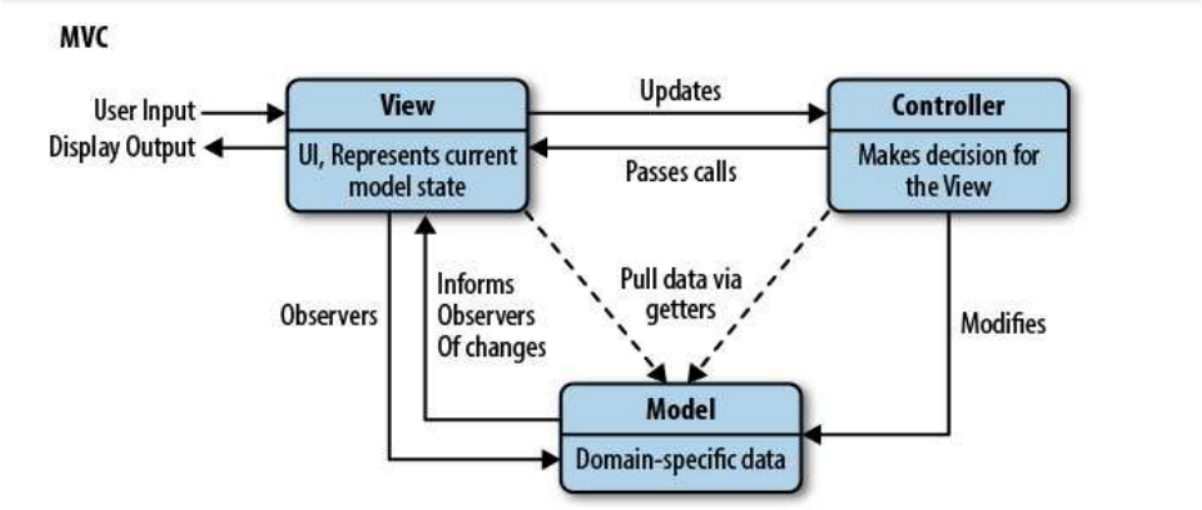

Figure 3.4.1 The system architecture MVC components.adopted from [1]

3.4.2 General Description

When a user requests a specific page from the server, the server forwards all the request details to the controller. The DB controller manages the client's request and retrieves relevant information from the model based on the request. The model, responsible for interacting with the database, focuses solely on data manipulation and doesn't handle user requests. Once the model provides its response to the controller, the controller interacts with the view to display the data to the user. After rendering the page, the controller receives the final presentation and sends it back to the user.

3.5 Model Objects Contains

The following are the main models in the class diagram Figure 2.10

- Customer
- Service
- Advertisement
- Report
- Request
- Employee
- UpdateTenantData
- SubscriptionStatus
- TransferringPoles
- Request_Status
- Request_Type
- Installment
- Bill

3.6 Database Logical Mapping

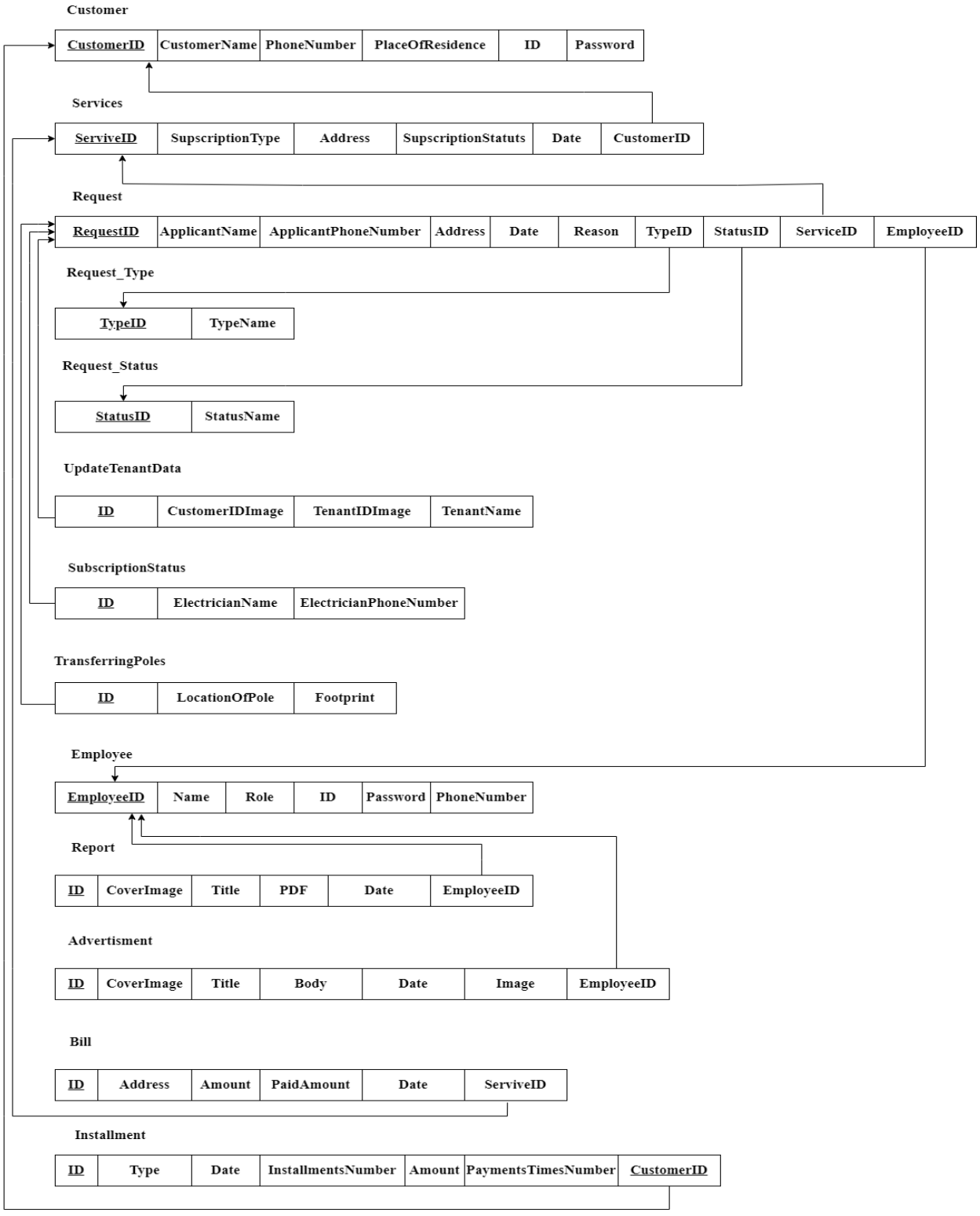

Figure 3.6 Database logical mapping

3.7 Class diagram tables description

1. Customer:
    a. CustomerID: unique integer of length 11, auto-increment for all Customers (PK).
    b. CustomerName: varchar data type of length 40 and not-null.
    c. PhoneNumber: unique integer whose length is 10 and not null
    d. PlaceOfResidence: varchar data type of length 80 and not-null.
    e. ID: unique integer data type of length 9 ,not null.
    f. Password: text data type, not null.
2. Service:
    a. ServicesID: unique integer of length 11 ,auto increment for all services(PK).
    b. SubscriptionType: character data type of length 4, not-null.
    c. Address: varchar data type of length 80, not-null.
    d. SubscriptionStatus: character data type of length 9, not null
    e. CustomerID: unique integer of length 9 (FK references CustomerID in Customer table)
3. Advertisement:
    a. ID: unique integer of length 11, auto increment for all advertisements(PK).
    b. Title: varchar data type of length 100, not-null.
    c. Body: varchar data type of length 250, not-null.
    d. Image: text data type, not null.
    e. CoverImage: text data type, not null.
    f. Date: date (auto generated by sequelize).
    g. EmployeeID: unique integer of length 11 (FK references EmployeeID in Employee table)
4. Report:
    a. ID: unique integer of length 11, auto increment for all reports(PK).
    b. Title: varchar data type of length 100, not-null.
    c. PDF: blob data type, not-null.
    d. CoverImage: text data type, not null.
    e. Date: date (auto generated by sequelize)
    f. EmployeeID: FK references EmployeeID in Employee table.

5. Request:
    a. RequestID: unique integer of length 11, auto increment for all requests (PK).
    b. Reason: varchar data type of length 100.
    c. ApplicantName: varchar data type of length 50, not null.
    d. ApplicantPhoneNumber: integer data type of length 10, not null.
    e. Address: varchar data type of length 80, not null.
    f. EmployeeID: FK references EmployeeID in employee table, represents the employee who tracks and constantly updates the status of the request.
    g. ServiceID: integer, FK references ServiceID in Service table, represents the service that the request belongs to.
    h. TypeID: integer FK references TypeID in Request_Type table, represents the type of the request.
    i. StatusID: integer FK references StatusID in Request_Status table, represents the status of the request.

6.Employee:

    a. EmployeeID: unique integer, not-null, auto increment for all employees (PK).
    b. Name: varchar of length 40, not-null.
    c. Role: varchar of length 5, not-null.
    d. ID: unique integer of length 9, not null.
    e. Password : text data type, not null .
    f. PhoneNumber: unique integer of length 10, not null.

7. UpdateTenantData:

    a. ID: integer of length 11 (PK, FK references RequestID in Request table).
    b. CustomerIDImage: text data type, not null .
    c. TenantIDImage: text data type, not null .
    d. TenantName: varchar data type of length 60, not-null.

8. SubscriptionStatus:

a. ID: integer of length 11 (PK, FK references RequestID in Request table).
b. ElectricianName: varchar data type of length 40, not-null.
c. ElectricianPhoneNumber: integer of length 10,not-null .

9. TransferringPoles:

a. ID: integer of length 11 (PK, FK references RequestID in Request table).
b. LocationOfPole: text data type, not null .
c. Footprint: text data type, not null .

10. Request_Type:

a. TypeID: integer of length 11, auto increment (PK).
b. TypeName: varchar data type of length 40.

11. Request_Status:

a. StatusID: integer of length 11, auto increment (PK).
b. StatusName: varchar data type of length 10.

12. Installment :

a. ID: unique integer of length 11, auto increment (PK).
b. Type: varchar data type of length 30, not null.
c. Date: Date data type, auto generated by sequelize.
d. InstallmentNumber: integer data type of length 2, not null.
e. Amount: Float(8,4) data type specifies a float column with a precision of 8(total number of digits) and a scale of 4(number of digits after the decimal point), not null.
f. PaymentTimesNumber: integer data type of length 2, not null.
g. CustomerID: integer data type of length 11 (FK references CustomerID in Customer table)

13. Bill:

a. ID: unique integer of length 11, auto increment (PK).
b. Date: Date data type, auto generated by sequelize.
c. Address: varchar data type of length 80, not-null.
d. Amount: Float(8,4) data type specifies a float column with a precision of 8 and a scale of 4 (number of digits after the decimal point), not null.
e. PaidAmount: Float(8,4) data type specifies a float column with a precision of 8 and a scale of 4 (number of digits after the decimal point), not null.
f. ServiceID: integer of length 11 (FK references ServiceID in Service table).

3.8 Database tables

Customers

Table 3.7.1 Customer table description.

Attribute	Type	Length	PK	FK	Null	unique
CustomerID	int	11	yes			yes
CustomerName	varchar	40				
PhoneNumber	int	10				yes
PlaceOfResidence	varchar	80				
ID	int	9				yes
Password	varchar	250			yes	

Services

Table 3.7.2 Services table description.

Attribute	Type	Length	PK	FK	Null	unique
ServiceID	int	11	yes			yes
SubscriptionType	char	4				
Address	varchar	80				
SubscriptionStatus	char	9				
Date	date					
CustomerID	int	11		yes		

Advertisement

Table 3.7.3 Advertisement table description

Attribute	Type	Length	PK	FK	Null	unique
ID	int	11	yes			yes
Title	varchar	100				
Body	varchar	250				
Date	date					
Image	text					
CoverImage	text					
EmployeeID	int	11		yes		

Report

Table 3.7.4 Report table description.

Attribute	Type	Length	PK	FK	Null	unique
ID	int	11	yes			yes
Title	varchar	100				
PDF	blob					
CoverImage	text					
EmployeeID	int	11		yes		yes
Date	date					

Requests

Table 3.7.5 Request table description

Attribute	Type	Length	PK	FK	Null	unique
requestID	int	11	yes			yes
TypeID	int	11		yes		
StatusID	int	11		yes		
Address	varchar	80				
Reason	varchar	100			yes	
Date	date					
EmployeeID	int	11		yes	yes	yes
ServiceID	int	11		yes		yes
ApplicantName	varchar	50				
ApplicantPhoneNumber	int	10				

UpdateTenantData

Table 3.7.6 UpdateTenantData table description.

Attribute	Type	Length	PK	FK	Null	unique
ID	int	11	yes	yes		yes
CustomerIDImage	text					
TenantIDImage	text					
TenantName	varchar	60			yes	

SubscriptionStatus

Table 3.7.7 SubscriptionStatus table description.

Attribute	Type	Length	PK	FK	Null	unique
ID	int	11	yes	yes		yes
ElectricianName	varchar	40				
ElectricianPhoneNumber	int	10				yes

TransferringPoles

Table 3.7.8 TransferringPoles table description.

Attribute	Type	Length	PK	FK	Null	unique
ID	int	11	yes	yes		yes
Footprint	text					
LocationOfPole	text					

Employee

Table 3.7.9 Employee table description.

Attribute	Type	Length	PK	FK	Null	unique
EmployeeID	int	11	yes			yes
EmployeeName	varchar	40				
Role	varchar	5				
ID	int	9				yes
Password	varchar	250				
PhoneNumber	int	10				
EndDate	Date				yes	

Request_Type

Table 3.7.10 Request_Type table description.

Attribute	Type	Length	PK	FK	Null	unique
TypeID	int	11	yes			yes
TypeName	varchar	40				

Request_Status

Table 3.7.11 Request_Status table description.

Attribute	Type	Length	PK	FK	Null	unique
StatusID	int	11	yes			yes
StatusName	varchar	10				

Installment

Table 3.7.12 Installment table description.

Attribute	Type	Length	PK	FK	Null	unique
ID	int	11	yes			yes
Type	varchar	30				
Date	date					
InstallmentNumber	int	2				
Amount	float	8				
PaymentTimesNumber	int	2				
CustomerID	int	11		yes		

Bill

Table 3.7.13 Bill table description.

Attribute	Type	Length	PK	FK	Null	unique
ID	int	11	yes			yes
Date	date					
Address	varchar	80				
Amount	float	8				
PaidAmount	float	8				
ServiceID	int	11		yes		

3.9 User Interface

Login page

Login requires entering the correct ID and password. Only subscribed individuals can log in and access their electrical services information. This ensures that the login process remains exclusive to registered users.

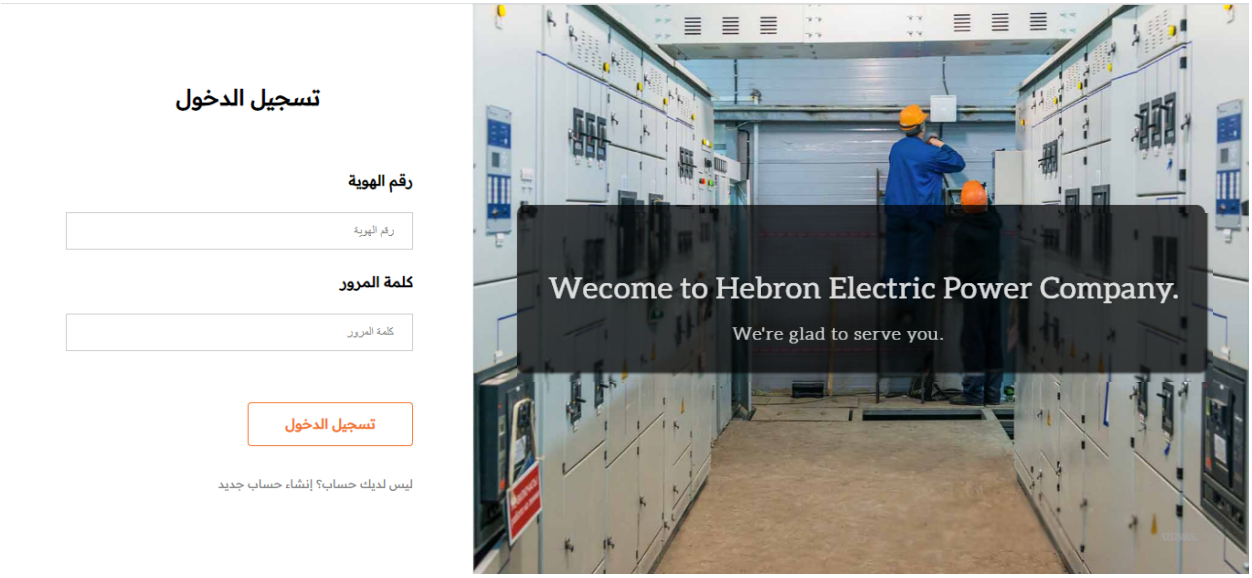

Figure 3.9.1 Login page

Services page

The services page offers a variety of requests that subscribers can easily submit online. These requests include changing payment methods from a bill to a prepaid card, installing street lighting and other essential services.

Figure 3.9.2 Services page

CSE dashboard

As shown in Figure 3.9.3, the CSE Dashboard enables customer services employees to view and manage subscriber information and requests.

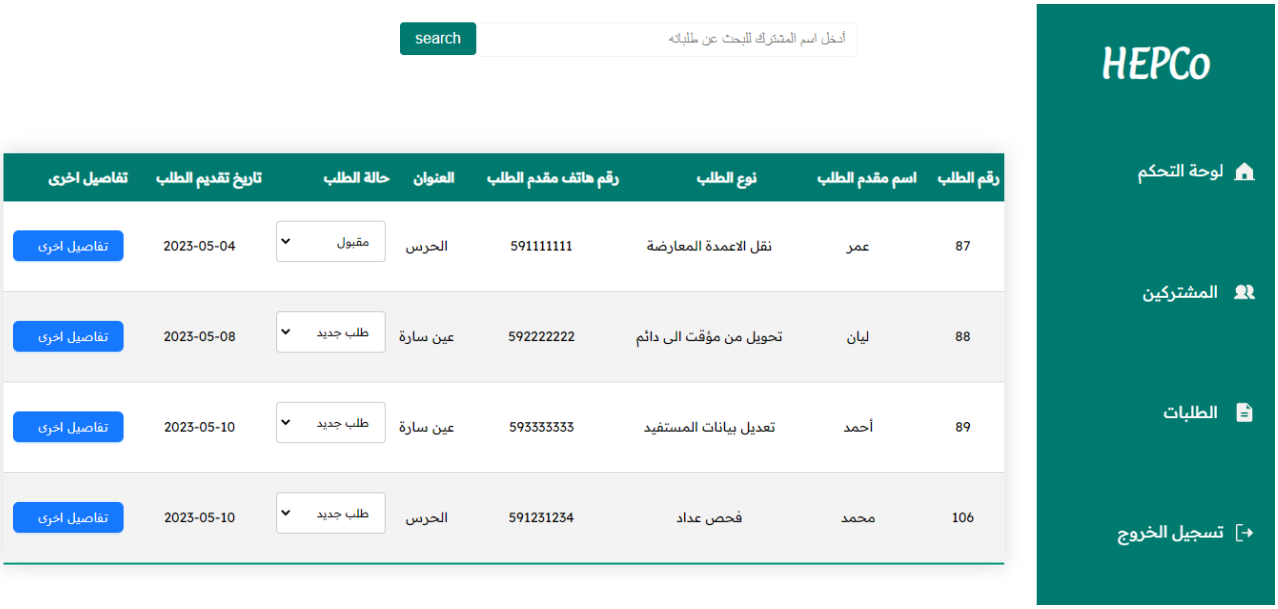

Figure 3.9.3 CSE dashboard

PRE dashboard

As shown in figure 3.9.4, the PRE Dashboard provides the functionality to add news articles and upload annual reports. The PR team can easily add new articles and upload annual reports through this user-friendly interface.

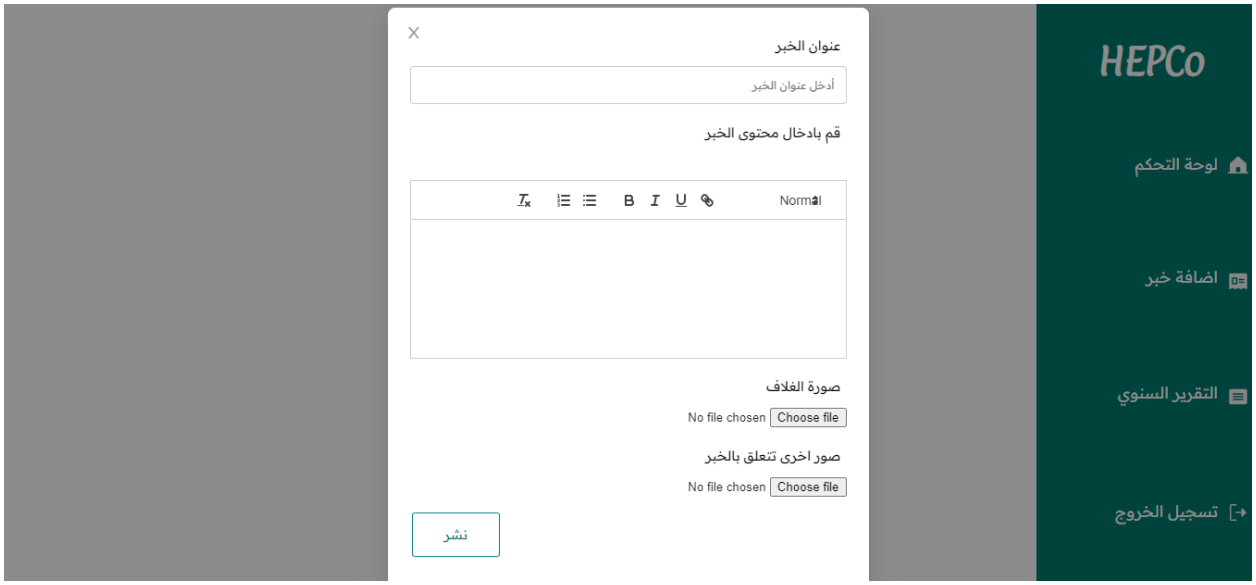

Figure 3.9.4 PRE dashboard

Admin dashboard

The admin has the ability to add a new employee by entering their relevant information, such as name, position, and contact details. Additionally, the admin can view a list of all employees. Furthermore, the admin can perform employee archiving, maintaining a record of past and inactive employees.

Figure 3.9.5 Admin dashboard

View submitted requests

Once logged in, subscribers can easily track their submitted requests and their respective statuses. They also have access to their subscription details, including bills and installment information. This user-friendly dashboard enables subscribers to stay informed about their interactions with HEPCo.

رقم الطلب	رقم الخدمة	العنوان	نوع الطلب	حالة الطلب	تاريخ تقديم الطلب
87	1	الحرس بجانب بنك فلسطين	نقل الاعمدة المعارضة	مقبول	2023-05-04
88	2	عين سارة	تحويل من مؤقت الى دائم	طلب جديد	2023-05-08
89	1	الحرس بجانب بنك فلسطين	تعديل بيانات المستفيد	طلب جديد	2023-05-10

Figure 3.9.6 View submitted requests

Request form for changing payment method from bill to prepaid card

Figure 3.9.7 illustrates a request form that enables subscribers to submit a request for changing their payment method from bill to card.

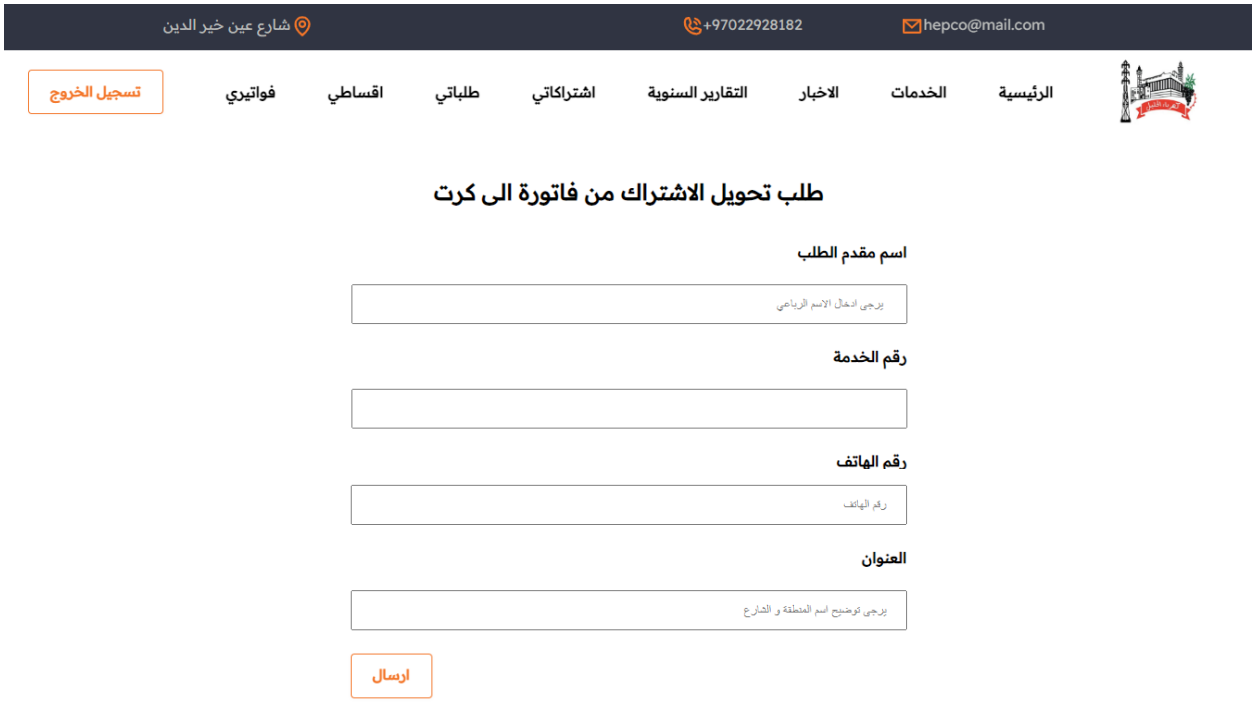

Figure 3.9.7 Request form for changing the payment method from bill to card

Chapter 4: Software Demonstration

4.1 Introduction

In this chapter, we will discuss how the system was built. The system implementation stage, which is the preparatory stage of the system to the practical stage, and then start programming and building the system. We will learn about the tools and programs we used while developing our system.

4.2 Software environment and tools

There are many helpful tools and requests we used in developing our software including:

- **Visual studio code** : The platform that is used to program frontend and backend.
- **Postman** : Software development for testing backend API's.
- **Draw.io** : Drawing all diagrams needed in demonstrating the system architecture.
- **phpMyAdmin:** MySQL Relational database.
- **XAMPP:** a software distribution which provides the Apache web server, MySQL database and Php all in one package. Which provides a solution for setting up a local web server.

- **React.js framework :** a powerful JavaScript framework used for building interactive user interfaces in web applications. It follows a component-based approach, allowing developers to create reusable UI components and efficiently manage state changes. React.js leverages a virtual DOM, enabling efficient rendering and updating of UI elements. It offers a declarative syntax and extensive ecosystem of libraries, making it a popular choice for front-end development.

- **Node.js framework** : a runtime environment built on JavaScript that allows developers to execute code on the server side. It is highly useful for building scalable and efficient network applications. Node.js utilizes an event-driven, non-blocking I/O model, which makes it well-suited for handling concurrent requests and real-time applications. It is

commonly used for creating APIs, server-side applications and building high-performance web applications.

- **GitHub:** Constantly pushing our work to the cloud, it was very helpful because my SSD in the laptop has crashed due to huge load and electricity hits while developing.
- **Google Drive:** sharing all documents with our team, editing and commenting is helpful.
- **npm Docs** : largest software registry to share and borrow packages including react.js.

In Addition to a promise-based node.js ORM tool named Sequelize for creating, updating and synchronizing the database with using node and express .

4.3 System programming

The web application we have developed is centered around transactions and supports multiple interfaces tailored to the specific user roles: customer, admin, and employee. Customers can submit requests, which are then reviewed and approved or denied by the Customer Support Executive (CSE). The Public Relations Executive (PRE) is tasked with publishing news and advertisements related to the company. The admin panel, managed by the IT Department Executive (ITDE), allows for adding new employees and maintaining an archive of existing employees.

4.3.1 System structure

Figure 4.3.1.1 shows the basic backend folders in the project that represent some of the system architecture components, previously mentioned in section 3.4 .

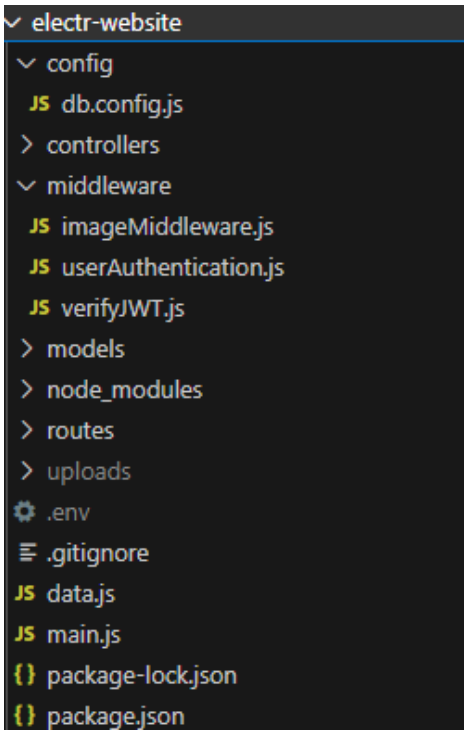

Figure 4.3.1.1 Project folders

1. Controllers folder: Each controller consists of multiple methods utilized in the routes folder and carries out operations on models, such as create, update, delete, and review.

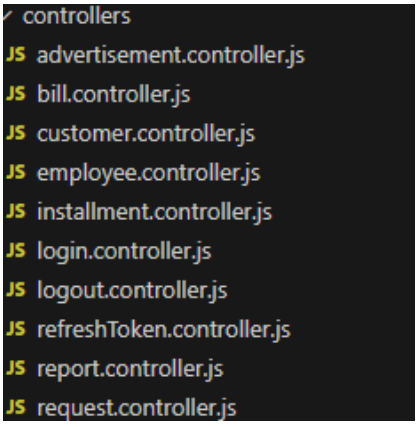

Figure 4.3.1.2 Controllers folder

2. Routes folder: Every route encapsulates all the APIs (endpoints) related to a specific class. It utilizes the functions and methods imported from the controller. Subsequently, it provides an appropriate response to the frontend API that requested the particular route.

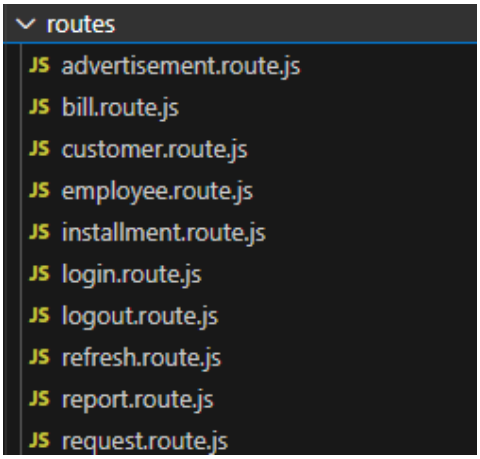

Figure 4.3.1.3 Routes folder

3. Models folder: They represent the entirety of the system's database tables. These tables are constructed using the sequelize Node.js ORM, which defines the table's name, fields, attributes, indexes, and handles validations and relationships between models.

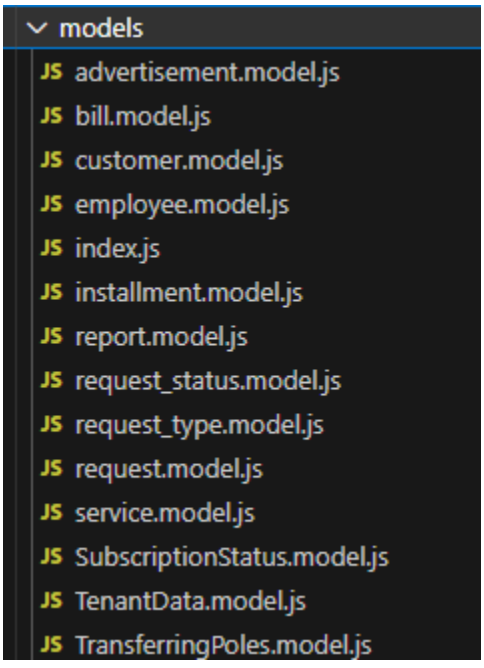

Figure 4.3.1.4 Models folder

4.4 Key fragments of the code

In this section we will be showing you snippets of react and nodeJS code.

4.4.1 Credential validation code

Login validation code consists of two stages, the first stage is to authenticate the user using the middleware function userAuthentication, the second stage is where we include access and refresh tokens to improve security in the subsequent call back function, the stages are shown in figure 4.4.1 as follows.

```
router.post('/', userAuthentication, async(req, res, next) => {        You, yesterday • storing refresh token …

    try {
        if (req.employee) {
            const accessToken = await loginController.generateAccessToken(req.employee.EmployeeID);
            console.log(accessToken);
            const refreshToken = await loginController.generateRefreshToken(req.employee.EmployeeID)
            console.log({ "role": req.employee.role, "employeeID": req.employee.EmployeeID });
            /* http only means not available to js more secure than storing refresh Token on local storage
             or another cookie that is available to javascript*/

            res.cookie('jwt', refreshToken, { httpOnly: true, sameSite: 'None', secure: true, maxAge: 30 * 24 * 60
                //send accessToken as json to frontend
            res.status(200).json({
                success: true,
                message: 'Successfully logged in',
                token: accessToken,
                // refresToken: refreshToken,
                userId: req.employee.EmployeeID,
                role: req.employee.role
            });
        } else if (req.customer) {
            try {
                await loginController.customerLogin(req, res);
                console.log('customer login');

            } catch (err) {
                console.log(err);
            }
        } else {
            res.status(401).json({ 'message': 'Invalid Credentials' });
        }
```

Figure 4.4.1 Authentication

4.4.2 Create a request API

Figure 4.4.2 shows a post route for creating a request of any type in the request route folder, and calling the controller for some other types as shown below.

```
router.post('/create', upload.fields([      arwa arafeh, 2 weeks ago • login …
    { name: 'reason', maxCount: 1 },
    { name: 'serviceID', maxCount: 1 },
    { name: 'appType', maxCount: 1 },
    { name: 'EmployeeID', maxCount: 1 },
    { name: 'appStatus', maxCount: 1 },
    { name: 'beneficiaryName', maxCount: 1 },
    { name: 'electricianName', maxCount: 1 },
    { name: 'electricianPhoneNumber', maxCount: 1 },
    { name: 'footPrint', maxCount: 1 },
    { name: 'locationImage', maxCount: 1 },
    { name: 'userIDImage', maxCount: 1 },
    { name: 'beneficiaryIDImage', maxCount: 1 },
    { name: 'applicantName', maxCount: 1 },
    { name: 'applicantPhoneNumber', maxCount: 1 },
    { name: 'applicantAddress', maxCount: 1 },
]), async(req, res, next) => {

    try {
        console.log(req.files);
        console.log(req.body);

        await requestController.create(req, res);
        const id = await db.sequelize.query(`SELECT RequestID FROM requests ORDER BY createdAt DESC LIMIT 1;`)
        console.log(id[0][0].RequestID);
        if (req.body.appType == 'تعديل بيانات المستفيد') {
            await requestController.tenantDataModification(req, res, id);
        }

        if (req.body.appType == 'نقل الاعمدة المعارضة') {
            await requestController.transferringPoles(req, res, id);

        }

        if (req.body.appType == 'تحويل من مؤقت الى دائم') {
            await requestController.propertyTypeModification(req, res, id);
        }

        // Send a response
        res.json({ message: 'Your Request has been sent successfully' });
    } catch (e) {
        res.status(500).json({ message: 'An error occurred.' });
    }
});
```

Figure 4.4.2 Create a request Endpoint

4.4.3 Employee Model

Figure 4.4.3 shows the employee model we created with sequelize, where we can add validations, indexes and other attributes without having to deal with the database, changes will automatically be saved.

```
// This line just to get the autocomplete working      You, now • Uncommitted changes
let s = new Sequelize({ dialect: 'mysql' });

const Employee = (sequelize) => {
    s = sequelize;
    return s.define("employee", {
        EmployeeID: {
            type: DataTypes.INTEGER,
            autoIncrement: true,
            primaryKey: true,
            allowNull: false,
        },
        PhoneNumber: {
            type: DataTypes.INTEGER,
            allowNull: false
        },
        EmployeeName: {
            type: DataTypes.STRING(30), //varchar 30
            allowNull: false
        },
        role: {
            type: DataTypes.STRING(30),
            allowNull: false
        },
        id: {
            type: DataTypes.INTEGER,
            allowNull: false,
        },

        password: {
            type: DataTypes.STRING(30),
        }
    }, {
        timestamps: false,
        tableName: 'employee',
        indexes: [
            // Define an index on the 'id' field
            {
                unique: true,
                fields: ['id']
            }
        ]
    });
};
```

Figure 4.4.3 Employee model

4.4.4 View requests and update request status

The code in Figure 4.4.4 enables CSE to view a list of requests and update the status of each request. It fetches requests from an API endpoint and uses a PUT request to modify the status. The user interface is then updated to reflect the changes made to the request status.

```
const Applications = () => {
  const [applications, setApplications] = useState([]);
  const empNum = localStorage.getItem("userId");
  const fetchApplications = async () => {
    const response = await fetch("http://localhost:5000/api/request");
    const allApplications = await response.json();
    setApplications(allApplications);
  };
  const handleStatusUpdate = async (id, newStatus) => {
    try {
      const response = await fetch(`http://localhost:5000/api/request`, {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          id: id,
          status: newStatus,
          empNum: empNum,
        }),
      });
      const updatedApplication = await response.json();
      setApplications(
        applications.map((app) =>
          app.RequestID === id
            ? { ...app, request_status: { StatusName: newStatus } }
            : app
        )
      );
    } catch (error) {
      console.error(error);
    }
  };
```

Figure 4.4.4 View requests and update request status

4.4.5 Retrieve and display customer bills

The code in the snippet retrieves and displays customer bills. It utilizes the fetchBills function to make an authorized API request to retrieve the bills associated with the authenticated user. The fetched data is then stored in the customerBills state variable using setCustomerBills. The code is executed when the component mounts using the useEffect hook.

```
const View_Bills = () => {
  const [customerBills, setCustomerBills] = useState([]);
  const userId = localStorage.getItem("userId");
  const token = localStorage.getItem("token");
  const fetchBills = async () => {
    const response = await fetch
    ("http://localhost:5000/api/customers/bills", {
      headers: {
        Authorization: `Bearer ${token}`,
        userid: userId,
      },
    });
    const jsonData = await response.json();
    setCustomerBills(jsonData);
  };
  useEffect(() => {
    fetchBills();
  }, []);
```

Figure 4.4.5 Retrieve and display customer bills

4.4.6 Admin dashboard for employee management and archiving

The code in Figure 4.4.6 implements an Admin Dashboard for managing and archiving employees. It retrieves employee data from an API and shows only active employees without an end date. The archiveEmployee function enables the admin to archive an employee by updating their end date through a PUT request. The user interface is then updated accordingly.

```
const Applications = () => {
  const [applications, setApplications] = useState([]);
  const empNum = localStorage.getItem("userId");
  const fetchApplications = async () => {
    const response = await fetch("http://localhost:5000/api/request");
    const allApplications = await response.json();
    setApplications(allApplications);
  };
  const handleStatusUpdate = async (id, newStatus) => {
    try {
      const response = await fetch(`http://localhost:5000/api/request`, {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          id: id,
          status: newStatus,
          empNum: empNum,
        }),
      });
      const updatedApplication = await response.json();
      setApplications(
        applications.map((app) =>
          app.RequestID === id
            ? { ...app, request_status: { StatusName: newStatus } }
            : app
        )
      );
    } catch (error) {
      console.error(error);
    }
  };
```

Figure 4.4.6 Admin dashboard for employee management and archiving

Chapter 5 : Testing

5.1 Introduction

In the stage of testing, we make sure that the system works correctly without any issues, and we also make sure that all the requirements of the project are fulfilled, and that the system performs effectively. The stage of testing comes after the design and implementation of the system.

5.2 Validation

All information entered in all fields in the request are tested to ensure that the data entered by the user matches all conditions as follows:

- Ensure that the user data is stored in the database during the login process.
- The process will not be executed if the users entered a file type that is not allowed.
- The process will not be executed if wrong data is entered.

5.3 API retesting

The system units were fully tested. The result of the examination was successful. The following tables review the testing we have done:

5.3.1 Frontend testing

Front end testing is a crucial part of the project to ensure that the UI components work as intended, providing a seamless and intuitive user experience. It involves verifying the handling of messages received from the back end. As well as, front end testing helps confirm that users are prompted to enter all the required information in the expected format. This validation ensures data integrity and improves the overall user experience.

Table 5.3.1.1 Login process

#	case	input	Expected output	Obtained output	Pass/ Fail
1	Correct and complete information	Id :**987654321** Password: **GH%$#123**	Successfully Logged In	As Expected	Pass
2	Wrong password	Id :**987654321** Password: **GHs$#123**	Incorrect Password	As Expected	Pass
3	Wrong id and password	Id :**987654321** Password: **GHs$#123**	Invalid Login Credentials	As Expected	Pass
	Correct id but no password exist	Id: **987654321** Password: **NULL**	No password found. Please sign up	As Expected	Pass

In Figure 5.1.3.1, when the customer entered an incorrect password, an error message was displayed, indicating that the password provided was invalid. Similarly, in Figure 5.1.3.2, when the customer's id is not associated with a password, an error message was displayed, indicating that the user must sign up to the system.

Figure 5.3.1.1 Incorrect password

Figure 5.3.1.2 Account not found

Table 5.3.1.2 Add new employee

#	case	input	Expected output	Obtained output	Pass/ Fail
1	Correct and complete information	employeeName: ahmad Id:123451234 PhoneNumber: 567654321 Password: GH%$#123 Role:CSE	Successfully added	As Expected	Pass
2	Account Already Exists	Id:123451234	An account with the provided identification number already exists	As Expected	Pass

The Figure 5.3.1.3. shows an admin trying to add a new employee but an error occurs due to incomplete or inaccurate data entry. The error message indicates that certain required fields were not properly filled out. Furthermore, in the Figure 5.3.1.4 the admin is attempting to add an employee who already exists in the system

Figure 5.3.1.3 Incomplete data entry while adding a new employee

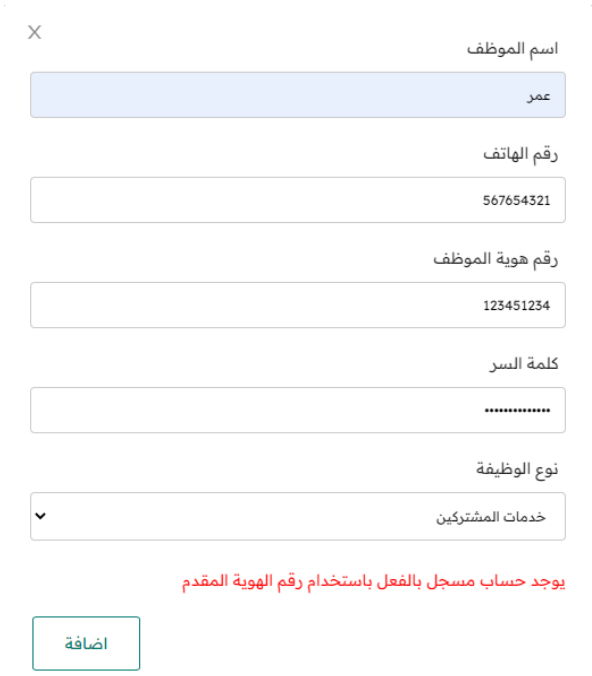

Figure 5.3.1.4 Admin attempt to add existing employee

Table 5.3.1.3 Sign up process

#	case	input	Expected output	Obtained output	Pass/ Fail
1	Correct and complete information	**Id:223456789 PhoneNumber:591234567 Password: GH%$#123**	Successfully Signed up	As Expected	Pass
2	Account Already Exists	**Id:223456789**	An account with the provided identification number already exists	As Expected	Pass
3	create an account for Non-Subscriber	**Id:223123123**	You cannot create an account if you are not subscribed to the company	As Expected	Pass

5.3.2 Backend testing

We have tested each route in the backend using Postman application before integrating them with the frontend.
Figure 5.3.2.1 shows the testing process of the request route , specifically when the user submits a form for transferring electrical poles away from a certain location.

Table 5.3.2.1 Poles transfer request

#	case	input	Expected output	Obtained output	Pass/ Fail
1	Correct and complete information	Reason: نقل عامودين العامود الاول يشكل خطر على اصحاب المنزل والعامود الثاني يعارض لورشة بناء serviceID: 4 Footprint: footprint.jpg locationImage: imageLocation.jpg applicantAddress: شارع الجامعة applicantName: نيرمين اسعد applicantPhoneNumber: 598476398	Your request has been sent successfully	As Expected	Pass
2	Wrong image type	Reason: نقل عامودين العامود الاول يشكل خطر على اصحاب المنزل والعامود الثاني يعارض لورشة بناء serviceID: 4 Footprint: footprint.png locationImage: imageLocation.png applicantAddress: شارع الجامعة applicantName: نيرمين اسعد applicantPhoneNumber: 598476398	Invalid file type only jpeg /jpg images are allowed	As Expected	Pass

Table 5.3.3 tests whether the access token was expired or typed incorrectly, the figures below demonstrate the process using the Postman application for testing the backend routes , the user login and an access token is generated, a refresh token will be stored in a cookie as in figure 5.3.2.1.

Table 5.3.2.2 Testing JWT

#	case	input	Expected output	Obtained output	Pass/ Fail
1	Valid authentication headers	Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySUQiOjIsImlhdCI6MTY4MzM5ODc4MiwiZXhwIjoxNjgzNDAyMzgyfQ.TLtil595gBL15zwa85VIiJEgRHbimDrn6JBsbc8Uekg	Bring all services for the customer	As Expected	Pass
2	Access token is expired	Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySUQiOjIsImlhdCI6MTY4MzM5ODc4MiwiZXhwIjoxNjgzNDAyMzgyfQ.TLtil595gBL15zwa85VIiJEgRHbimDrn6JBsbc8Uekg	Error: forbidden	403	Pass
3	Access token not provided	Token:	Error: Unauthorized	401	Pass

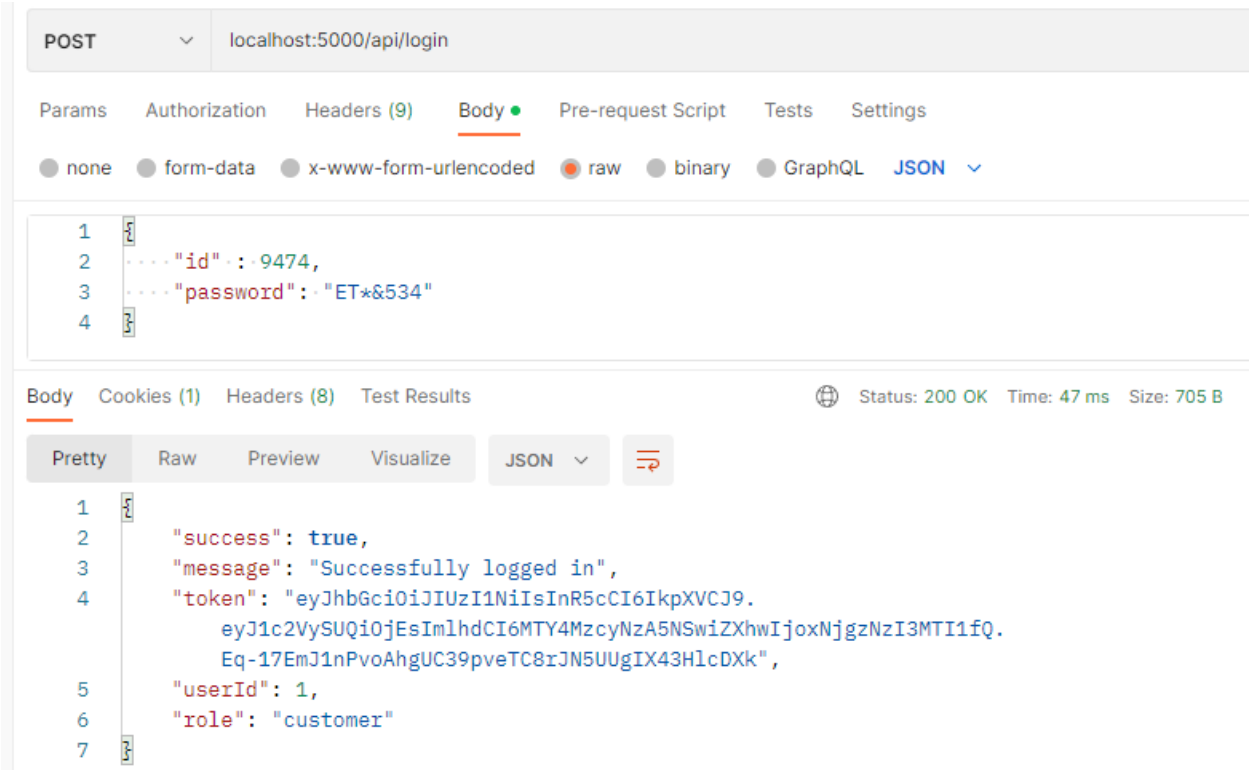

Figure 5.3.2.1

User access token

Then after applying the middleware that verifies the token in the api that brings all the services we get the successful response as in figure 5.3.2.2

If the token was expired we get the result forbidden , and if the token was now provided in the authorization headers we get the result unauthorized.

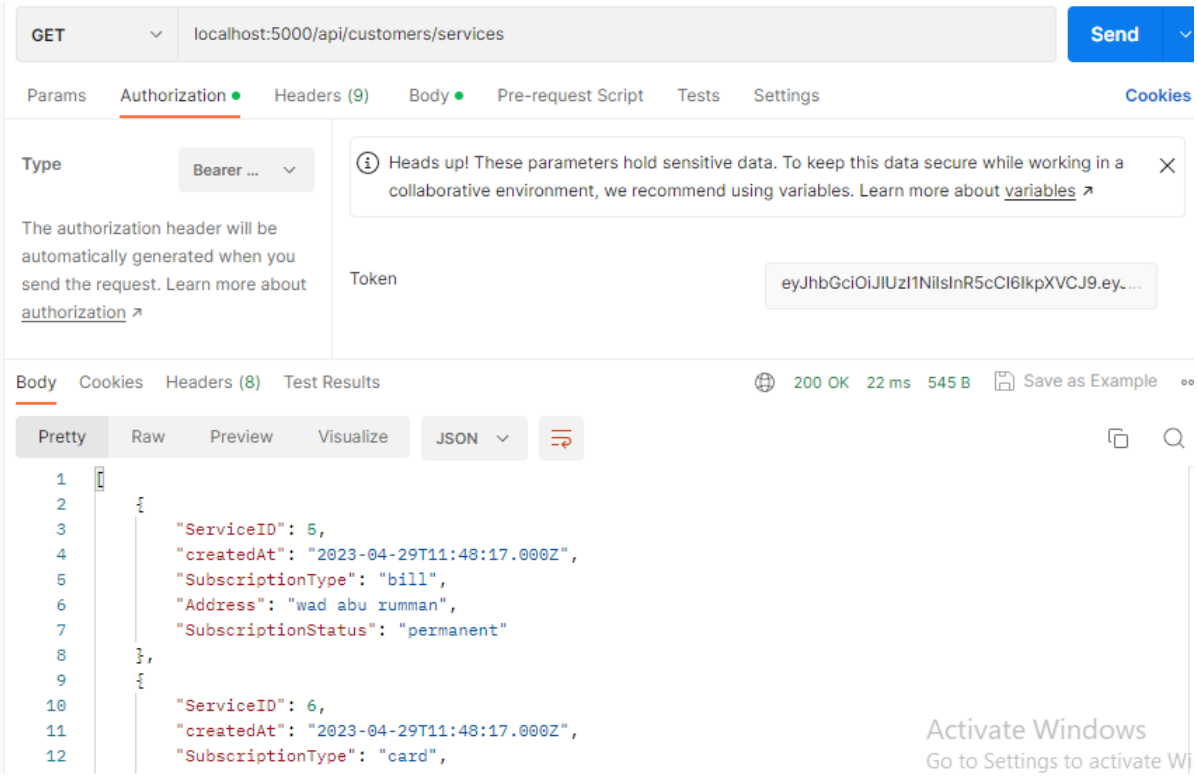

Figure 5.3.2.2: Verify JWT

Now, after passing access token check, here comes the refresh token functionality that generates a new pair of access and refresh tokens. And after the refresh token expires the user must log in again, figure 5.3.2.3 the user is logging in again to generate a new access and refresh token which improves the security and protects the login process from XSS, CSRF attacks.

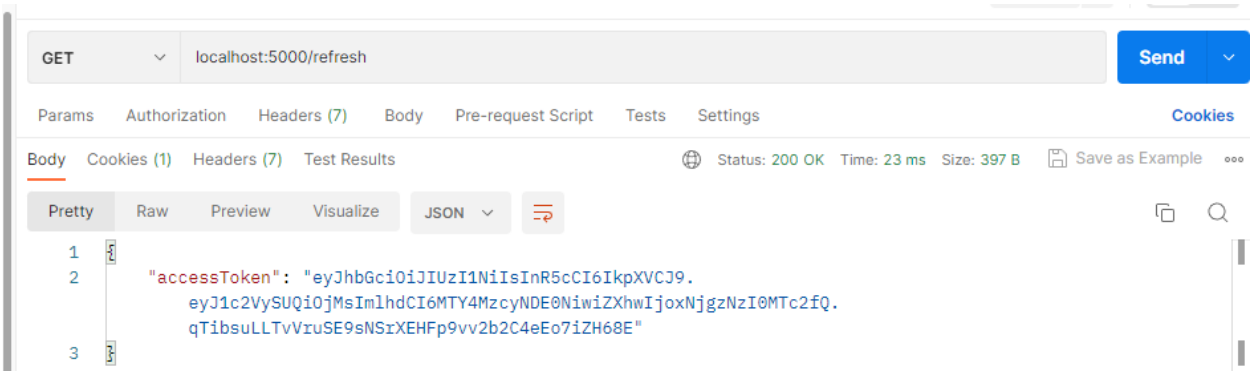

Figure 5.3.2.3: Generating a new token

In Addition, figure 5.3.2.4 shows the process when an employee creates an advertisement .

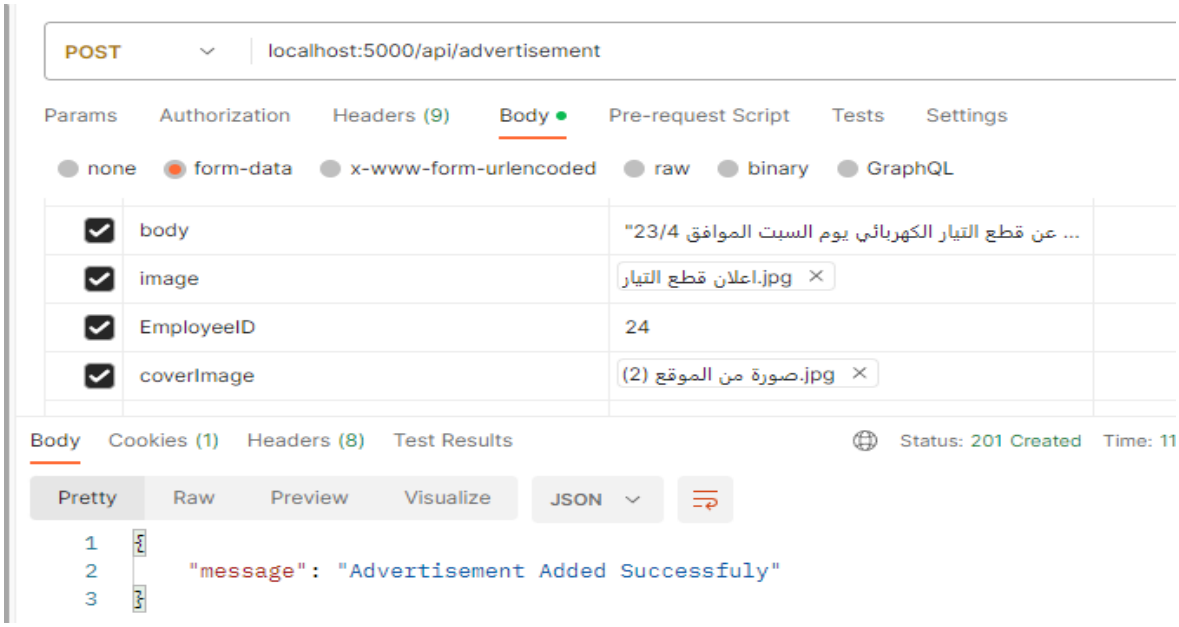

Figure 5.3.2.4: Create an advertisement

Finally, figures 5.3.2.5 and 5.3.2.6 represent the create request endpoint testing on two types of requests .

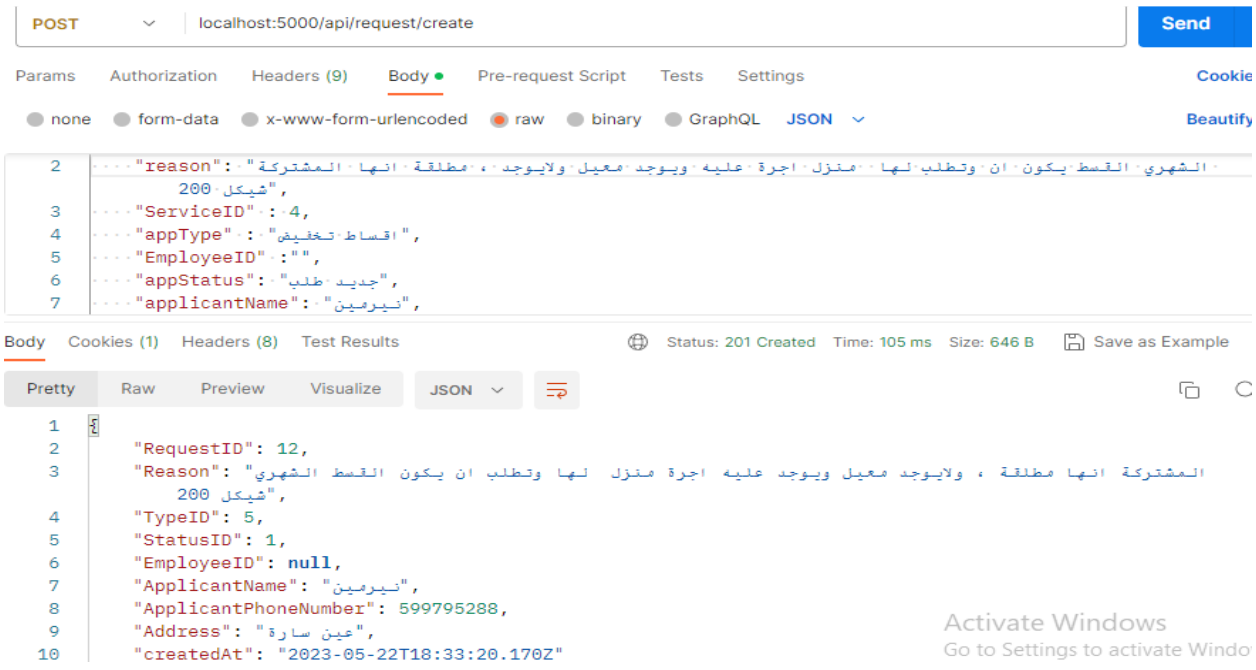

Figure 5.3.2.5: Reduction of installments request

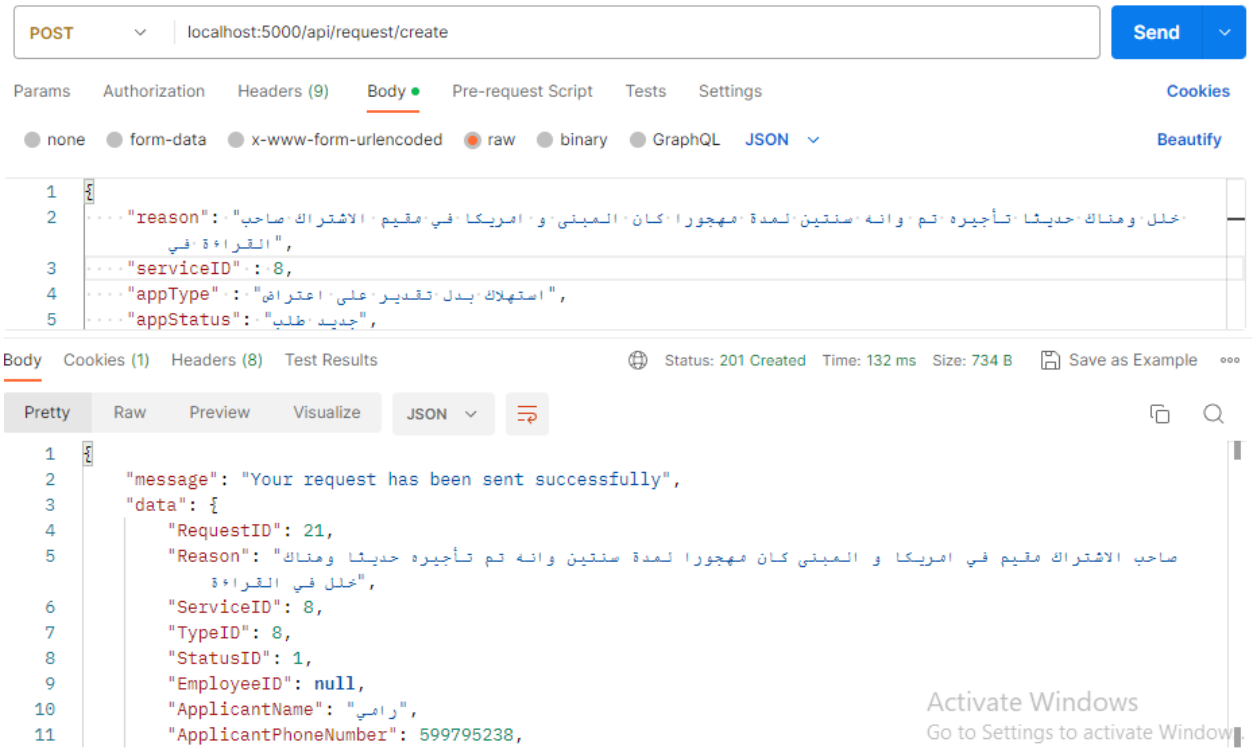

Figure 5.3.2.6: Objection on consumption amount

The request api testing appearing in figure 5.3.2.6 results in the request being stored into the database as shown in figure 5.3.2.7.

RequestID	Reason	createdAt	ServiceID	TypeID	StatusID	EmployeeID	ApplicantName	ApplicantPhoneNumber	Address
	الا...								
21	صاحب الاشتراك مقيم في امريكا و المبنى كان مهجورا ل...	2023-05-23	8	8	1	NULL	رامي	599795238	شارع الحدل

Figure 5.3.2.7: Request stored in the database

Chapter 6: Conclusion

6.1 Conclusion

In conclusion, the development of the website for HEPCo will have a significant impact and benefit for the organization. While the website has not been utilized yet, we are optimistic about the positive impact it will have on improving operations and enhancing customer satisfaction. Once the website is launched and put into action, it is expected to provide customers with a convenient and user-friendly platform to engage with HEPCo services. The website will enable customers to easily submit service requests, access important information, and manage their accounts, thereby enhancing their overall experience.

Internally, the website will streamline processes for employees, allowing them to efficiently handle customer inquiries, access relevant data, and collaborate effectively. This will result in improved productivity and the ability to provide prompt and reliable services to customers.

We are excited about the potential of the website to strengthen the relationship between HEPCo and its customers. It will facilitate better communication, transparency, and responsiveness, ultimately enhancing customer satisfaction and loyalty.

6.2 Recommendations

We developed a web-application that facilitates the process of communication between the customers and the company. We recommend all customers to take advantage of our system, they will benefit from knowing details about their subscriptions, applying for requests, and keep updated by the status of their request. This will make the process easier and flexible than before.

We recommend working on implementing the system at the company for at least one semester, to specify the problems that result from using the system and working to solve them.

6.3 Future work

In the future, we are looking forward to adding many features to the current system such as:

- Adding a notification feature whenever the employee updates the status of the request.
- Log in using one time authentication code sent by phone number.
- Adding a live chatting feature that facilitates the communication between the customer and the employees.
- Send emails that contains the title and body of the news using a cloud-based service.
- Use a cloud-based storage service like AWS S3 to store uploaded files.

References

[1] Sommerville, Ian. Software Engineering, 9/E. Pearson Education India, 2011

Sites used in developing the system:

Node.js. (n.d.). Node.js. https://nodejs.org/en

React – A JavaScript library for building user interfaces. (n.d.). React. https://legacy.reactjs.org/

Sequelize. (n.d.). Feature-rich ORM for Modern TypeScript & JavaScript. https://sequelize.org/