|  |  |
| --- | --- |
|  | **Cognizant Academy**  **Power Consumer Management System**  **EU NA IDE Mentorship – Business Aligned Project**  **Case Study Specification**  **Version 1.0** |
| |  |  |  |  | | --- | --- | --- | --- | |  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** | | **Name** | Khaleelullah Hussaini Syed |  |  | | **Role** | Trainer |  |  | | **Signature** | t-syed8 |  |  | | **Date** | 26 October 2022 |  |  | |
|  |

Table of Contents

[1.0 Important Instructions 3](#_Toc118566814)

[2.0 Introduction 4](#_Toc118566815)

[2.1 Purpose of this document 4](#_Toc118566816)

[2.2 Project Overview 4](#_Toc118566817)

[2.3 Scope 4](#_Toc118566818)

[3.0 Use Case Diagram 6](#_Toc118566819)

[4.0 System Architecture Diagram 7](#_Toc118566820)

[5.0 Development Phases 7](#_Toc118566821)

[6.0 System Requirements 8](#_Toc118566822)

[**6.1.1** **Module – Loan Plans** 8](#_Toc118566823)

[**6.1.2** **Module – Loan Applications** 11](#_Toc118566824)

[**6.1.3** **Module – EMI Management** 15](#_Toc118566825)

[**6.1.4** **Module – EMI Extensions Management** 18](#_Toc118566826)

[7.0 Deployment requirements 21](#_Toc118566827)

[8.0 Design Considerations 21](#_Toc118566828)

[9.0 Reference learning 23](#_Toc118566829)

[10.0 Project Templates 24](#_Toc118566830)

[11.0 Change Log 24](#_Toc118566831)

# Important Instructions

1. Associate must adhere to the Design Considerations specific to each Technolgy Track.
2. Associate must not submit project with compile-time or build-time errors.
3. Being a Full-Stack Developer Project, you must focus on ALL layers of the application development.
4. Unit Testing is Mandatory, and we expect a code coverage of 100%. Use Unit testing and Mocking Frameworks wherever applicable.
5. All the Microservices, Client Application, DB Scripts, have to be packaged together in a single ZIP file. Associate must submit the solution file in ZIP format only.
6. If backend has to be set up manually, appropriate DB scripts have to be provided along with the solution ZIP file.
7. A READ ME has to be provided with steps to execute the submitted solution, the Launch URLs of the Microservices in cloud must be specified.

(Importantly, the READ ME should contain the steps to execute DB scripts, the LAUNCH URL of the application)

1. Follow coding best practices while implementing the solution. Use appropriate design patterns wherever applicable.
2. You are supposed to use an In-memory database or code level data as specified, for the Microservices that should be deployed in cloud. No Physical database is suggested for Microservice.

# Introduction

## Purpose of this document

The purpose of the software requirement document is to systematically capture requirements for the project and the system “Power Consumer Management System” that has to be developed. Both functional and non-functional requirements are captured in this document. It also serves as the input for the project scoping.

The scope of this document is limited to addressing the requirements from a user, quality, and non-functional perspective.

High Level Design considerations are also specificed wherever applicable, however the detailed design considerations have to be strictly adhered to during implementation.

## Project Overview

The \*\*Power Consumer Management System (PCMS)\*\* is designed to manage customer and plan information efficiently. It allows users to register, search for plans based on their location, and track their usage while enabling administrators to approve registrations and manage plans effectively. The system generates monthly billing based on user consumption and provides an intuitive interface for both customers and administrators. The project aims to streamline the management of power consumer data, enhance user experience, and ensure efficient billing and plan management.

## Scope

Below are the modules that needs to be developed part of the Project:

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Req. Name** | **Req. Description** |
| REQ\_01 | **Registration module** | * User is able to register himself on application * Admins are able to approve or reject new registrations. * By using Username and Password user can login into system only if his registration is approved by admin. |
| REQ\_02 | **Plan & Applications module** | * Admin will be able to create a new plan in the system * The users and admin can view the existing plans present in the system based on location. * users can submit a application * A admin will be able to view the applications & approve or reject a application |
| REQ\_03 | **Usage and Bill generation module** | * User able to view the usage and bill. So, that they can pay based on usage. |

# Use Case Diagram

The following use case diagram shows various users of the system and their responsibilities.

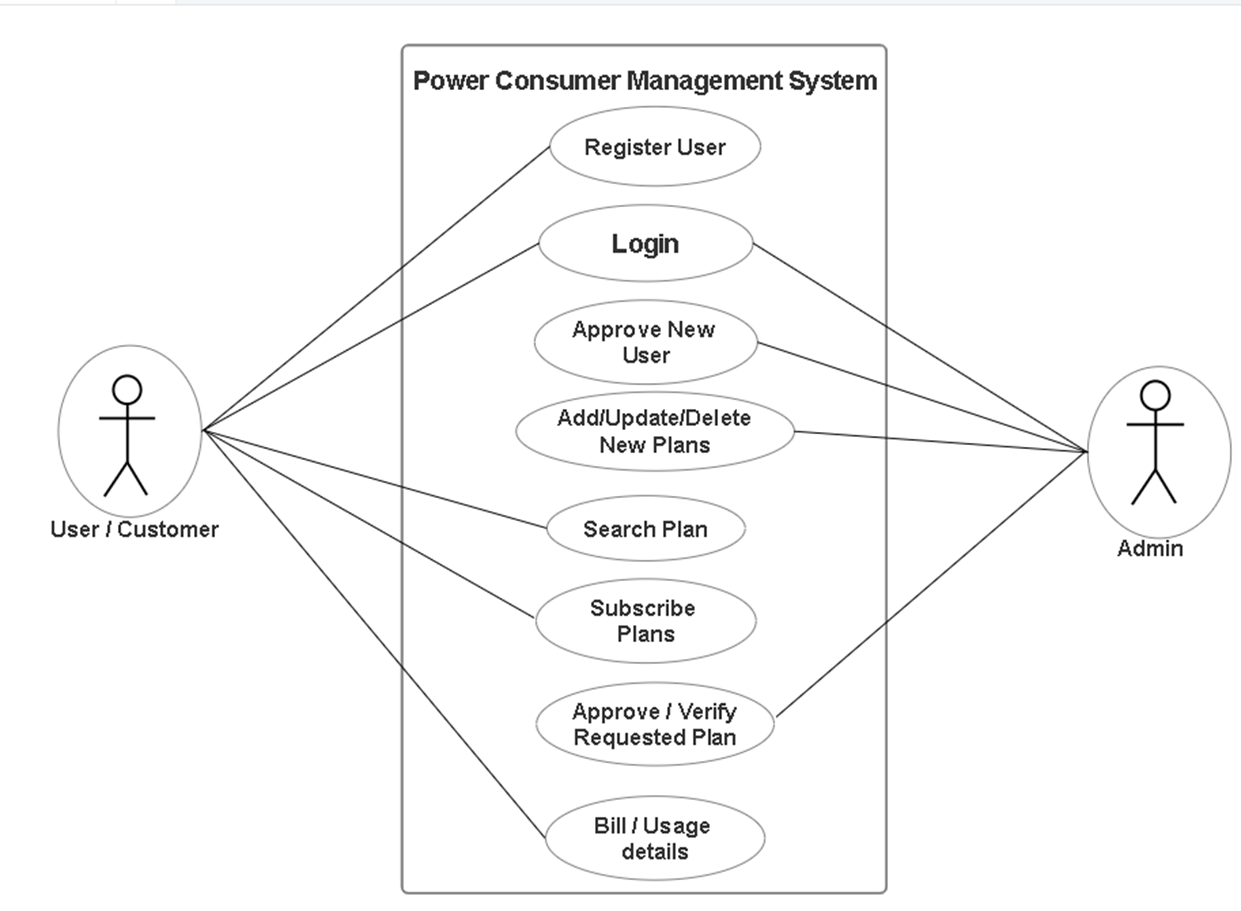


Figure 1 : Use case diagram

# System Architecture Diagram

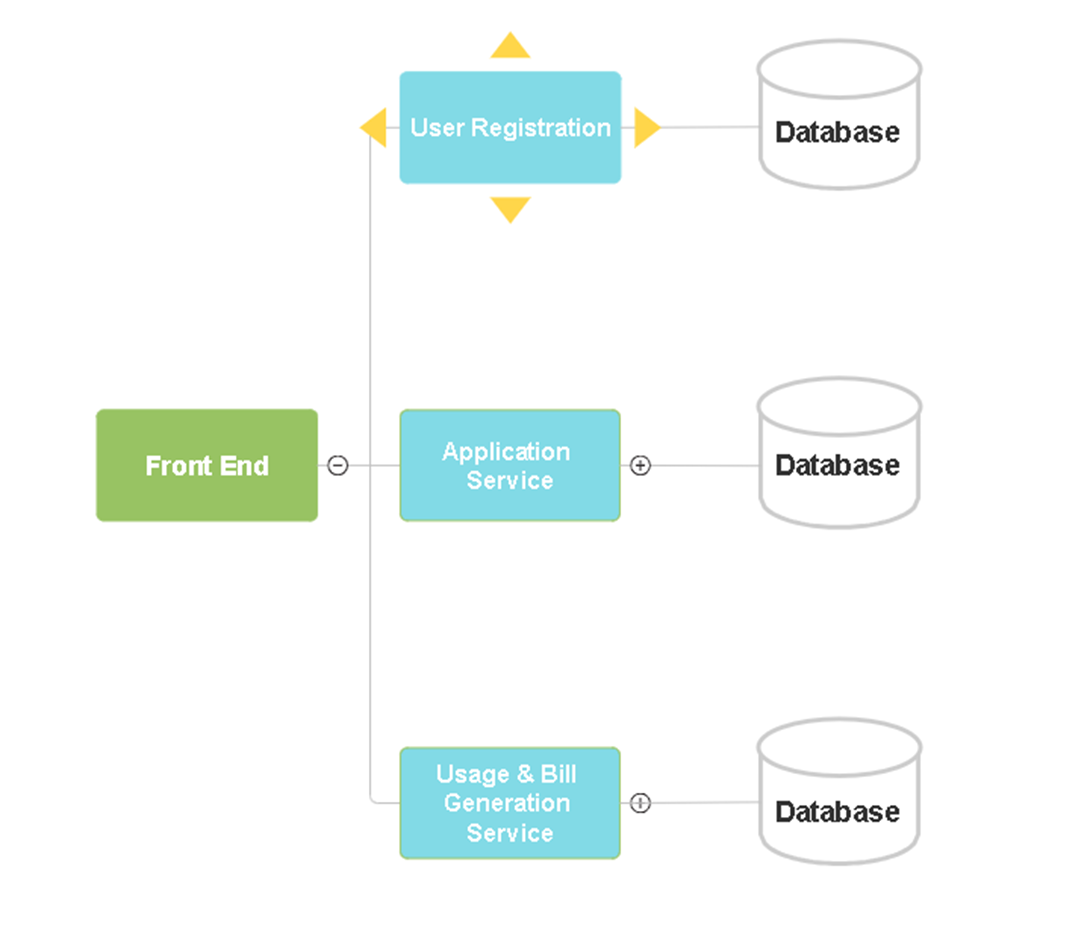
+

Figure 2 : Application Architecture Diagram

# Development Phases

* The application will be developed in 2 phase.
* Each phase will have 3 stages followed by a review at the end.
* The phase-1 output will be unit tested core business logic of the application.
* In phase-2 the output will be a functional application with micro-service and the Front end.
* Each stage of the development phase must be completed alongside the learning milestone

# System Requirements

A diagram of a computer

Description automatically generated

**ER Diagram for case study**

### **Module – User Registrations**

Using the User Registrations module admin and users can perform the following operations.

1. Users can register themselves on the system.
2. The admin can view the details of users.
3. Admin can approve or reject the registrations.

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

A screenshot of a computer screen

Description automatically generated

**Figure 3 : ER Diagram – Users Registrations**

1. Enforce the following constraints on the database apart from primary key, foreign key and unique keys
   1. MailId should be unique otherwise throw error
   2. Password should contain atleast one special character and length should be more than 6 characters.
   3. Status of application by default should be NEW and admin can change it to Approve or Reject only.
   4. Admin need to provide rejection comment if he is rejecting any user.

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Insert a new User.
   2. Return a list of Newly registered Users.
   3. Update users for approve and reject.

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
   1. Insert a new User.
   2. Return a list of Newly registered Users.
   3. Update users for approve and reject.

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. All exceptions in the micro-service must be handled and logged using a logging library
3. Create the following end-points and test them using postman and export the requests into a json file.

Table 2 : Register User - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/user/register |
| **Request Type** | POST |
| **User Role** | user |
| **Trigger** | Front end |
| **Description** | Endpoint for users to register 2themselves by entering their details. |
| **Inputs** | Full Name, Email, Phone Number, SSN (for individuals), Address, Zip Code, Password, etc. |
| **Outputs** |  |

Table 3 : Register User - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/user/update/{id} |
| **Request Type** | PUT |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | Admin can approve or reject new user registration, by updating status. |
| **Inputs** | UserDTO with status updated |
| **Outputs** |  |

Table 4 : Register User - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/user/{id} |
| **Request Type** | GET |
| **User Role** | Customers and Admin |
| **Trigger** | Front end |
| **Description** | Get details for a specific user by ID. Admins can access all users; individuals can access their own. |
| **Inputs** |  |
| **Outputs** |  |

Table 5 : Loan Plans - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/user/getAllUsers |
| **Request Type** | GET |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | Retrieve a list of all users. |
| **Inputs** |  |
| **Outputs** |  |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. **Registration Component**: Handles the user registration process.
   1. Form for collecting user details (e.g., username, email, password).
   2. Form validation for required fields, password strength, and email format.
   3. Error handling for failed registrations (e.g., duplicate email, server errors).
   4. Success message upon successful registration.
2. **Login Component**: Manages user login.
   1. Form for entering login credentials (e.g., username or email, password).
   2. Form validation (e.g., required fields).
   3. Error handling for incorrect credentials.
   4. Option to redirect to registration page if the user doesn't have an account.

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro-services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

### **Module – Plan & Applications Module**

This module will provide the following functionalities to the application users

1. Admin is responsible for creations of plans
2. Users are able to apply for those Plans
3. Admin is able to approve or reject user request.
4. Admin can able to update or delete plans.

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

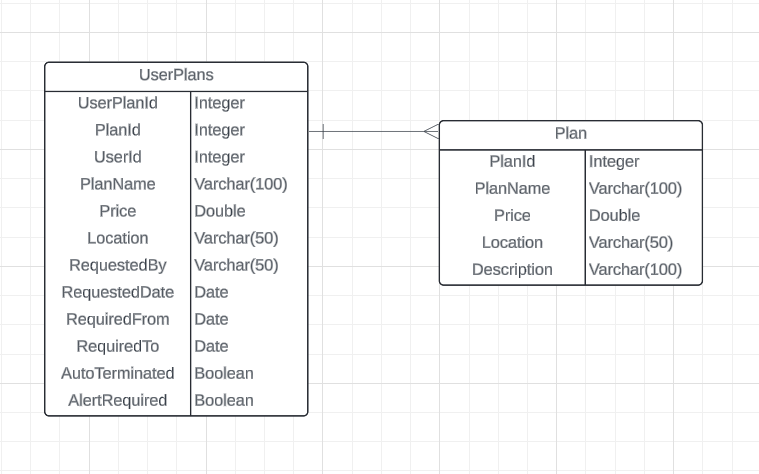


Figure 4 : ER Diagram – Plan & Applications Module

1. Apply the following constraints apart from primary keys and foreign keys on the database
   1. Not Null Constraints :UserPlanId, PlanId, UserId, and PlanName in the UserPlans table should not be null.
   2. PlanId, PlanName in the Plan table should not be null.
   3. Default Values: For AutoTerminated and AlertRequired in the UserPlans table, default values should be set to false.
   4. Check Constraints: Price values should be greater than 0.
   5. Date Constraints: RequiredFrom should always be earlier than RequiredTo.

Note: Seed the data for document types as Bank statement, Payslip, ITR, SaleDeed, Medical certificate, ID proof, Address proof.

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
4. Create repositories or data access objects (DAO) for accessing the database

**Methods:**

* findByUserId(int userId): Retrieve plans for a specific user.
* findActivePlans(): Retrieve all active plans where AutoTerminated is false.

**Plan Repository**

**Methods:**

* **findByLocation(String location):** Retrieve plans available in a specific location.
* **findPlansByPriceRange(double minPrice, double maxPrice):** Retrieve plans within a specified price range.

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Implement the following business operations:

* Plan Management
* Create, update, and delete plans in the system.
* Validate that Price is positive before adding a plan.
* User Plan Assignment
* Assign plans to users based on requirements.
* Set up alerts for AlertRequired cases.
* Auto-Termination Check
* If the current date is beyond RequiredTo, mark AutoTerminated as true.

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 7 : User Plan - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/user-plan/assign |
| **Request Type** | POST |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | Assign a plan to user. |
| **Inputs** | User ID,Plan ID,Requested Date, Status |
| **Outputs** |  |

Table 8 : User Plan - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/user-plan/{userId} |
| **Request Type** | GET |
| **User Role** | Admin and customers |
| **Trigger** | Front end |
| **Description** | Get all plans associated with a specific user. |
| **Inputs** |  |
| **Outputs** |  |

Table 9 :User Plan- Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/user-plan/update-status/{id} |
| **Request Type** | PUT |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | Update the status of a user’s plan |
| **Inputs** |  |
| **Outputs** |  |

Table 10 : Plan - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/plan/create |
| **Request Type** | POST |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | Create a new plan |
| **Inputs** | Plan Name, Description, Location, Price |
| **Outputs** |  |

Table 11 : Plan - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/plan/update/{id} |
| **Request Type** | PUT |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | Update plan details |
| **Inputs** |  |

Table 12 : Plan - Endpoint - 6

|  |  |
| --- | --- |
| **URL** | /api/plans/location/{location} |
| **Request Type** | GET |
| **User Role** | Admin and customers |
| **Trigger** | Front end |
| **Description** | Get all plans available for a specific location. |
| **Inputs** |  |

Table 13 : Plan - Endpoint - 7

|  |  |
| --- | --- |
| **URL** | /api/plans/delete/{id} |
| **Request Type** | DELETE |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | Delete a specific plan. |
| **Inputs** |  |

**Stage: Font-end design**

Create the following components as per the specification provided below.

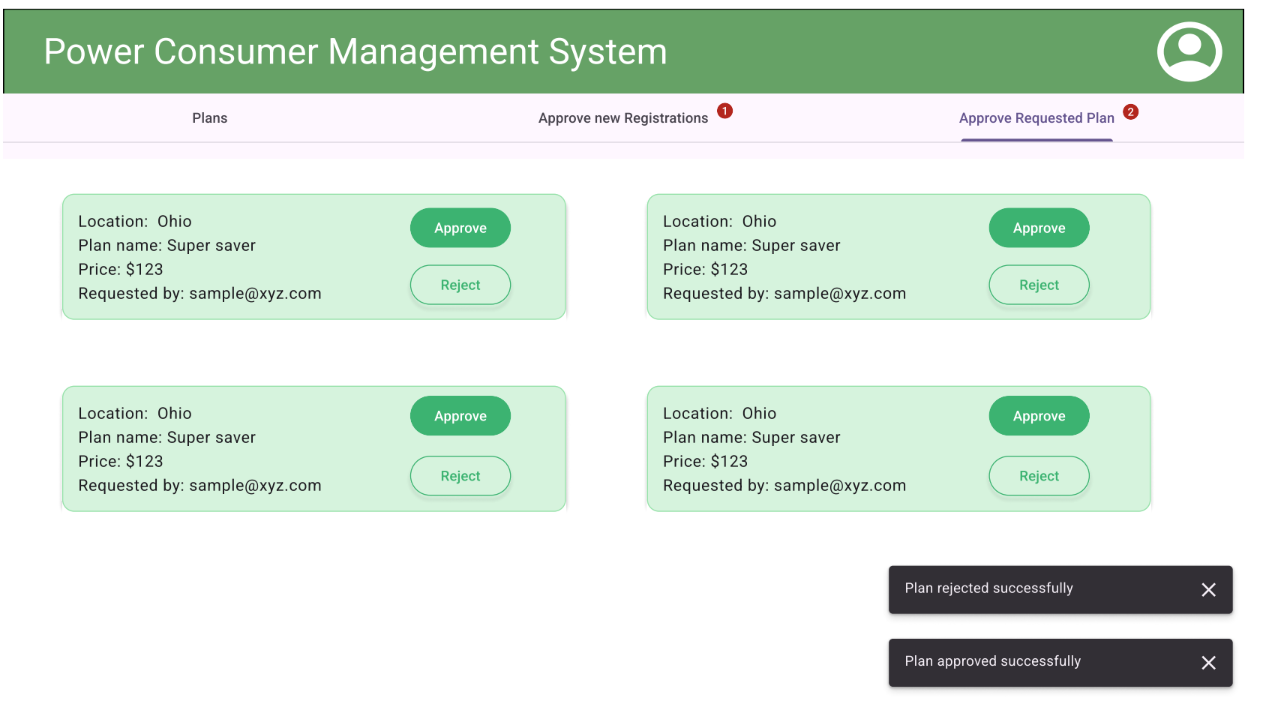
1. **Admin Add/Update/Delete Component**

A screenshot of a computer

Description automatically generated

* 1. Create a component where admin can search any plan based on the input provided.
  2. Add new Plan Button in order to add new plan.
  3. For each of the plan created there should be edit and delete functionality.

1. **Admin User Requested Plan Approval Component**



* 1. Panel which list out all the request plan by user.
  2. For each of the plan raised for there should be approve and request button
  3. Show appropriate toaster message.

**3.User Search and Apply Plan**

1. User can search the plan based on the input
2. User can raise request for plan based on the location.

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

### **Module – Usage and Bill Generation**

This module will allow the features to the users

1. System is able to show the details of usage
2. System is also able to show amount as per usage
3. The users are able to see the usage and bill amount.

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

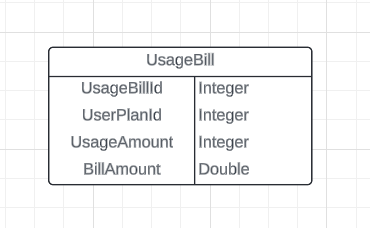


Figure 5 : ER Diagram – Usage and Bill Generation

1. Apart from primary and foreing keys implement the following additional constraints
   1. UsageAmount should be greater than or equal to 0.
   2. BillAmount should be greater than or equal to 0.
   3. BillAmount should be calculated based on usage.

Note: Seed data into the payment methods table as Card/NetBanking/UPI respectively

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. findByUserPlanId(int userPlanId): Retrieve all bills associated with a specific user plan.
   2. findBillsAboveAmount(double amount): Retrieve bills with amounts greater than a specified value.
   3. findUsageByDateRange(Date from, Date to): Retrieve usage data within a specified date range (if a date field is added later).

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Implement the following business operations:

* Bill Generation
* Calculate the BillAmount based on UsageAmount and the rate per unit (e.g., per kWh). The rate could be dynamically fetched from the associated Plan or set as a configuration parameter.
* If UsageAmount exceeds a certain threshold, apply a surcharge or discount, if applicable.
* Usage Tracking
* Track the monthly or periodic usage for each user and generate a corresponding bill.
* Store previous bills and allow retrieval for audit purposes.
* Notification Service
* Send alerts if a bill exceeds a certain amount or if there is an unusual spike in usage.

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All the exceptions must be handled and logged using a logging library.
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 14: Usage & Bill Management - End point - 1

|  |  |
| --- | --- |
| **URL** | /api/usage-bill/{userId} |
| **Request Type** | GET |
| **User Role** | Admin and customers |
| **Trigger** | Front ends |
| **Description** | Get usage and bill details for a user. |
| **Inputs** | userplanId |
| **Outputs** |  |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. **Display Usage and Bill Component**
2. Develop a component to be used by bank managers which contains a form to create a new EMI plan.
3. EMI status should be automatically taken as OnGoing
4. Once all the details are validated, user should be able to get an acknowledgement on submission of form.

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro-services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

Design Considerations

Java specific design considerations are attached here. These design specifications, technology features have to be strictly adhered to.



Refer this link for the coding standards. <https://cognizantonline.sharepoint.com/:w:/r/sites/GTP-Solutions/Gencsharepath/Shared%20Documents/Internship2020/FSE/Coding%20standards/Effective%20coding%20standards.docx?d=w6430574d9db5478bbbe37c25b16e68e2&csf=1&web=1&e=84lTVf>

|  |  |
| --- | --- |
| **Category** | **Rule** |
| Database | Table names in database must be pascal cased and plural. All primary keys must be named as Pk\_<table>. All foreign keys must be named as FK\_<PrimaryKeyTable>\_<ForeignKeyTable> |
| Database | Column names must be pascal cased. Multi-word column must be split using \_ (underscore) |
| Coding | Follow pascal casing for naming classes, interfaces, methods, properties and other public members |
| Coding | Use camel casing for method parameter name, backing fields for properties and private variables. Consts must be capitalized |
| Coding | All exceptions must be handled and logged using a logging library |
| Coding | For communication between micro-services use the HttpClient class available in .Net and Java |
| Unit testing | Each method in services classes in business logic must be unit tested using nUnit/jUnit |
| Unit testing | Use a mocking library to mock the repositories while performing tests for business logic layer |
| Code Coverage | Should be minimum 90% |
| Front end(Angular) | Use pascal casing for the component names |
| Front end(Angular) | Create all components and data services in Angular/React project in dedicated folders |
| GitHub | Create ONLY Private Repositories.  No password should be stored.  DO NOT Mention in the Profile that You work for Cognizant |

**UI GUILDELINES**

header & footer - #006400  
navigation tab active underline color - #006400  
primary button(all buttons)  - #3CB371  
Secondary button border and text color(all buttons) - #3CB371  
card edit button border and text color - #3CB371  
Badge(tab new req registration and approval count) - #B3261E;  
Toaster background color - #322F35;  
Toaster text color - #F5EFF7;  
Dropdown list color(all drop down options) - #F3EDF7 Progress circle blue color - #32ADE6;  
Progress circle background - #000000 with 20% opacity;

Registration from header text color - #006400

# Reference learning

Please go through all of these k-point videos for

Microservices deployment into Azure Kubernetes Service.

|  |
| --- |
| [AzureWithCICD-1](https://cognizant.kpoint.com/app/video/gcc-19532393-d4e0-4fd9-8a0c-80ecbdb349d3) |
| [AzureWithCICD-2](https://cognizant.kpoint.com/app/video/gcc-6633a958-ab72-4c69-b926-fe832e4b56a1) |
| [AzureWithCICD-3](https://cognizant.kpoint.com/app/video/gcc-553eb186-c1cf-448e-96fc-a96fe37b2e6a) |
| [AzureWithCICD-4](https://cognizant.kpoint.com/app/video/gcc-fad7d4af-d651-4501-99c6-2785190670c2) |

**Other References:**

|  |  |
| --- | --- |
| Java 8 Parallel Programming | <https://dzone.com/articles/parallel-and-asynchronous-programming-in-java-8> |
| RestTemplate | [https://dzone.com/articles/Microservices-communication-feign-as-rest-client](https://dzone.com/articles/microservices-communication-feign-as-rest-client) |
| Postman | [https://dzone.com/articles/centralized-documentation-in-Microservice-spring-b](https://dzone.com/articles/centralized-documentation-in-microservice-spring-b) |
| ECL Emma Code Coverage | <https://www.eclipse.org/community/eclipse_newsletter/2015/august/article1.php> |
| Lombok Logging | <https://javabydeveloper.com/lombok-slf4j-examples/> |
| Spring Security | <https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world> |
| H2 In-memory Database | <https://dzone.com/articles/spring-data-jpa-with-an-embedded-database-and-spring-boot>  <https://www.baeldung.com/spring-boot-h2-database> |
| AppInsights logging | <https://www.codeproject.com/Tips/1044948/Logging-with-ApplicationInsights> |
| Error response in WebApi | <https://stackoverflow.com/questions/10732644/best-practice-to-return-errors-in-asp-net-web-api> |
| Read content from CSV | <https://stackoverflow.com/questions/26790477/read-csv-to-list-of-objects> |
| Access app settings key from appSettings.json in .Net core application | <https://www.c-sharpcorner.com/article/reading-values-from-appsettings-json-in-asp-net-core/>  <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-3.1> |

# Project Templates









# Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Changes Made | | | |
| V1.0.0 | Initial baseline created on 05-November-2022 by Khaleelullah Hussaini Syed | | | |
|  |  | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
|  |  |  |  |
|  |  |