

[Github](https://github.com/obaidu11ah/)[linkedin](https://www.linkedin.com/in/obaid-ullah-381791266?utm_source=share&utm_campaign=share_via&utm_content=profile&utm_medium=android_app)

NATIONALUNIVERSITY OF MODERN LANGUAGES

ISLAMABAD



Service ease platform

**Service Ease Platform (Desktop Application)**



Submitted By: **Obaid Ullah**

Department: **BS Information Technology**

Session: 20**23** – 20**27**

[**Github-Project-Link**](https://github.com/obaidu11ah/CarRental-Software)

**Table of Contents**

[1. Introduction: 4](#_Toc27445)

[➢ Objective: 4](#_Toc27446)

[➢ Problem Statement: 4](#_Toc27447)

[➢ Solution: 4](#_Toc27448)

[➢ Project Description: 4](#_Toc27449)

[➢ Key Features: 5](#_Toc27450)

[➢ Use Cases: 5](#_Toc27451)

[➢ Advantages: 5](#_Toc27452)

[➢ Potential Challenges: 6](#_Toc27453)

[➢ Conclusion: 6](#_Toc27454)

[2. Entities and Their Relationships (ERD Overview): 6](#_Toc27455)

[3. Business Rules for ERD (Entity Relationship Diagram): 9](#_Toc27456)

[➢ Users to Services (One-to-Many) 9](#_Toc27457)

[➢ Users to Booking (as Customer) (One-to-Many) 9](#_Toc27458)

[➢ Services to Booking (One-to-Many) 9](#_Toc27459)

[➢ Services to ServiceAudit (One-to-Many) 9](#_Toc27460)

[ERD (Entity Relationship Diagram) of ServiceEase: 10](#_Toc27461)

[4. Business Rules for EERD (Enhanced ERD) Of ServiceEase 11](#_Toc27462)

[➢ User Roles (Specialization/Generalization) 11](#_Toc27463)

[➢ Service Availability (Participation Constraint) 11](#_Toc27464)

[➢ Audit Mechanism (Optional Participation) 11](#_Toc27465)

[➢ Booking Status (Domain Constraint) 11](#_Toc27466)

[EERD (Enhanced ERD) of ServiceEase 12](#_Toc27467)

[5. SQL Queries, Triggers, and Performance Optimization 13](#_Toc27468)

[➢ Database Initialization 13](#_Toc27469)

[➢ SQL Table Definitions 13](#_Toc27470)

[➢ Indexing for Query Optimization 13](#_Toc27471)

[➢ Trigger-Based Audit Logging 14](#_Toc27472)

[➢ Transactions & Concurrency Control: 15](#_Toc27473)

[➢ Secure Login & Validation Procedures 18](#_Toc27474)

[➢ CheckLoginFunc: 18](#_Toc27475)

[ LOGIN FUNCTION: 19](#_Toc27476)

[➢ ValidateServiceProvider 19](#_Toc27477)

[➢ ValidateCustomer: 19](#_Toc27478)

[➢ Analytical Views and Functions 20](#_Toc27479)

[➢ GetServicesByUser:{ For provider dashboard } 20](#_Toc27480)

[➢ GetServicesShow {To display all available services}: 21](#_Toc27481)

[➢ GetBookedServicesByProvider{ Show all bookings under a provider }: 21](#_Toc27482)

[➢ GetBookedServicesByCustomer: Show bookings for a specific customer 22](#_Toc27483)

[➢ For Admin Testing command: 22](#_Toc27484)

[➢ Backup & Recovery Strategy 22](#_Toc27485)

[➢ Backup creation: 24](#_Toc27486)

[➢ Security/authorization and Role-Based User Access 24](#_Toc27487)

[6. Summary of Recovery Techniques in SQL Server 24](#_Toc27488)

[GUI interface: 26](#_Toc27489)

[➢ Login Interface: 26](#_Toc27490)

[➢ Signup: 27](#_Toc27491)

[➢ Service Interface: 28](#_Toc27492)

[➢ Dashboard button on service provider interface: 28](#_Toc27493)

[➢ Edit Profile interface: 29](#_Toc27494)

[➢ Booking button Service Ease provider: 30](#_Toc27495)

[➢ Logout: 30](#_Toc27496)

[➢ Customer Interface: 31](#_Toc27497)

[➢ Payment interface 31](#_Toc27498)

[➢ Dashboard Customer Interface: 32](#_Toc27499)

[➢ Logout: 32](#_Toc27500)

[➢ Edit Profile customer: 33](#_Toc27501)

Project: Service Ease Provider Platform

# 1. Introduction:

## ➢ Objective:

ServiceEase is a user-friendly platform designed to connect service seekers with reliable service providers in various categories such as home maintenance, beauty and wellness, tutoring, event management, and more. The platform aims to streamline the process of finding, hiring, and reviewing service providers while ensuring quality and trust.

## ➢ Problem Statement:

Finding reliable and skilled service providers is often a time-consuming and challenging task. Many service seekers face issues such as lack of transparency, inconsistent pricing, and poor quality of services. Service providers, on the other hand, struggle to find a platform that helps them showcase their skills and connect with potential clients effectively.

## ➢ Solution:

ServiceEase addresses these challenges by offering a centralized platform that connects service seekers with verified and reviewed service providers. Through a seamless interface, the platform ensures transparency, consistency in pricing, and quality assurance. ServiceEase also empowers service providers by giving them a professional space to display their expertise and build their clientele.

## ➢ Project Description:

ServiceEase is a desktop-based application developed in Java using the **Swing GUI framework**, offering a modern graphical user interface for an intuitive and interactive user experience. The project was built **in Eclipse IDE with full integration of Microsoft SQL Server (SSMS) as the backend database.**

**JDBC (Java Database Connectivity)** was used to establish a reliable and secure connection between the Java application and SQL Server. This allowed seamless data transfer and operations such as registration, login authentication, service posting, booking management, and feedback storage.

Author : Obaid Ullah

The overall system architecture follows a modular and scalable design, focusing on user roles and clear separation of concerns.

## ➢ Key Features:

1. **User Registration:**

Separate registration paths for Service Seekers and Service Providers.

1. **Login System with Role Selection:**

Secure login with role-based redirection (using hashed passwords and validation).

1. **Service Posting and Management (Providers):**

Providers can post services with title, category, description, availability, price, and estimated duration.

1. **Booking System (Customers):**

Customers can search and book services. Booked services are stored and trackable.

1. **Dashboard and Profile Management:**

Both users have dashboards with role-specific features. Users can see their profile and manage bookings/services.

## ➢ Use Cases:

* A homeowner searching for a plumber to fix a leakage
* A student looking for a math tutor for weekend classes.
* An individual booking a makeup artist for a wedding.
* A company hiring an event manager for an upcoming conference.

## ➢ Advantages:

* Centralized platform for diverse service needs.
* Time-saving and easy-to-use interface.
* Enhanced trust through verified profiles and user reviews.

Author: Obaid Ullah

## ➢ Potential Challenges:

* Ensuring the authenticity of service providers.
* Handling disputes between users and service providers.
* Calability to accommodate a growing user base.

## ➢ Conclusion:

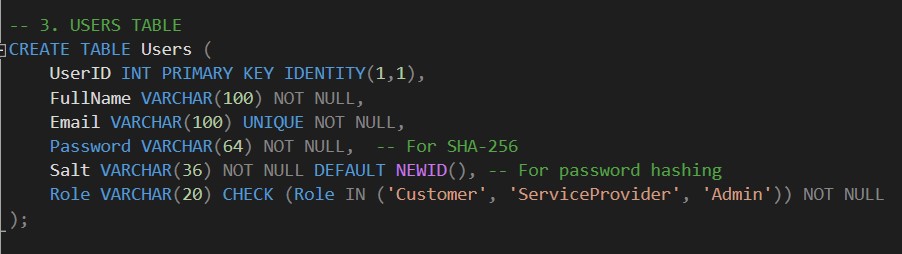
ServiceEase is a scalable and versatile platform designed to simplify the process of finding and managing services. By providing a centralized solution, it aims to create value for both service seekers and providers, fostering trust and convenience in the service industry.

# 2. Entities and Their Relationships (ERD Overview):

This section outlines the core database entities of the **ServiceEase Provider Platform**, their attributes, and how they are related. These entities are designed to maintain data integrity, enforce relational constraints, and support all core functionalities such as user management, service listings, bookings, and auditing.

* **Users Table – Base Table for All User Roles**

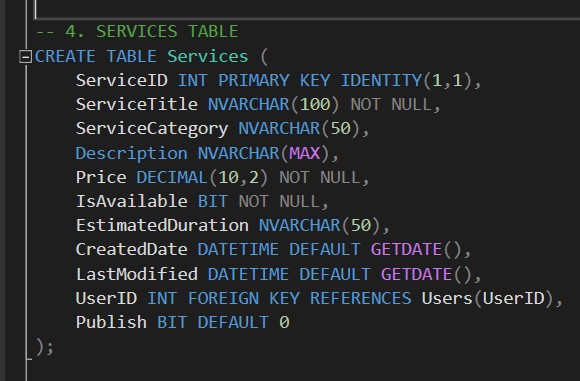
|  |  |
| --- | --- |
| **Attribute** | **Description** |
| **UserID (PK)** | Unique identifier for each user |
| **FullName** | Full name of the user |
| **Email** | Unique email address |
| **Password** | Securely stored hashed password (SHA-256) |
| **Role** | User type: Customer, ServiceProvider |



Author: Obaid Ullah

* **2.1. Services Table – Services Offered by Providers**

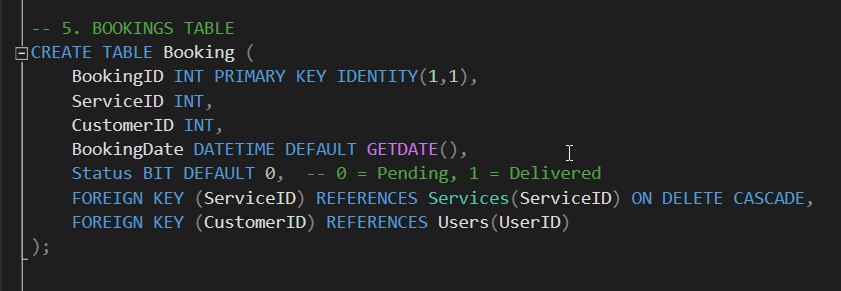
|  |  |
| --- | --- |
| **Attribute** | **Description** |
| **ServiceID (PK)** | Unique identifier for each service |
| **ServiceTitle** | Title of the service (e.g., "AC Installation") |
| **ServiceCategory** | Category of the service (e.g., "Home Appliances") |
| **Price** | Service cost in PKR |
| **IsAvailable** | Availability status: 1 for Available, 0 for Booked |
| **UserID (FK)** | Linked to the Service Provider (Users.UserID) |



* **Booking Table – Customer Bookings Record**

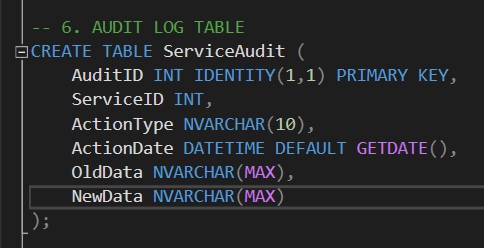
Author: Obaid Ullah

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| **BookingID (PK)** | Unique booking identifier |
| **ServiceID (FK)** | References the booked service (Services.ServiceID) |
| **CustomerID (FK)** | References the customer (Users.UserID) |
| **Status** | 0 = Pending, 1 = Delivered |



* **ServiceAudit Table – Logs for Service Changes**

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| **AuditID (PK)** | Unique log entry ID |
| **ServiceID (FK)** | The service that was modified (Services.ServiceID) |
| **ActionType** | Type of action: INSERT, UPDATE, DELETE |
| **OldData / NewData** | JSON-formatted old and new data for auditing changes |



# 3. Business Rules for ERD (Entity Relationship Diagram):

## ➢ Users to Services (One-to-Many)

* **Rule**: One **ServiceProvider** can offer **many services**, but each service belongs to exactly **one provider**.
* **Explanation**: This is a 1:N relationship. The UserID in the Services table is a **foreign key** referencing the primary key of the Users table.
* **Why?**: A service provider can list multiple services like plumbing, electrician work, etc., but each service is created by only one provider.

Author : Obaid Ullah

## ➢ Users to Booking (as Customer) (One-to-Many)

* **Rule**: A customer can make multiple bookings, but each booking is linked to only one customer.
* **Explanation**: This is also a 1:N relationship. The CustomerID in the Booking table is a **foreign key** referencing the Users table.
* **Why?**: A customer might book several different services, but each booking record belongs to a single customer.

## ➢ Services to Booking (One-to-Many)

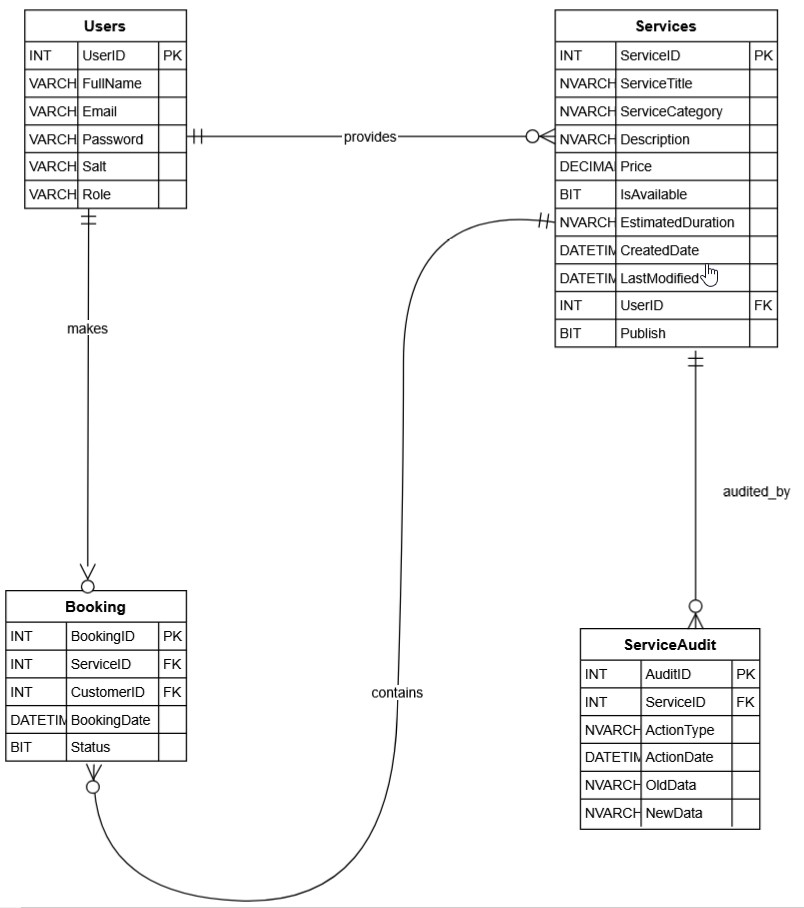
* **Rule**: A service can be booked many times, but each booking references only one service.
* **Explanation**: Another 1:N relationship. The ServiceID in the Booking table references the Services table.
* **Why?**: The same service, like “AC Repair,” can be booked by multiple users, but each booking is tied to one specific service.

## ➢ Services to ServiceAudit (One-to-Many)

* **Rule**: Each service can have multiple audit entries, but each audit entry is linked to a single service.
* **Explanation**: 1:N again. ServiceID in ServiceAudit is a foreign key referencing Services.
* **Why?**: Every update, delete, or insert event is logged for auditing. One service might be updated multiple times.

Author: Obaid Ullah

# ERD (Entity Relationship Diagram) of ServiceEase:



Author: Obaid Ullah

# 4. Business Rules for EERD (Enhanced ERD) Of ServiceEase

EERD adds advanced details like inheritance, constraints, participation, and generalization.

## ➢ User Roles (Specialization/Generalization)

* **Rule**: All roles (Customer, ServiceProvider, Admin) are specializations of a generic User.
* **Explanation**: This is a **generalization hierarchy**. Each role inherits base user attributes but also plays specific roles.
* **Why?**: Enables role-based access and behavior without duplicating user fields.

## ➢ Service Availability (Participation Constraint)

* **Rule**: A service must be linked to a valid, active service provider (total participation).
* **Explanation**: This is a **total participation constraint** — a service **must** belong to a user.
* **Why?**: Prevents orphan services from existing without an owner.

## ➢ Audit Mechanism (Optional Participation)

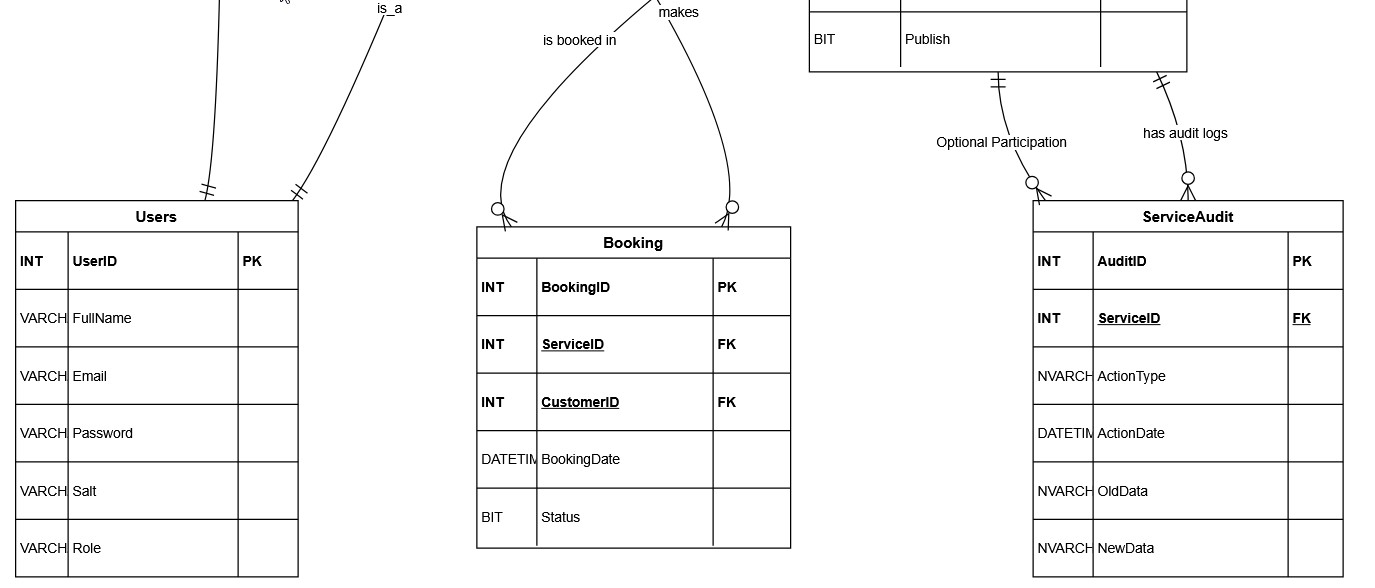
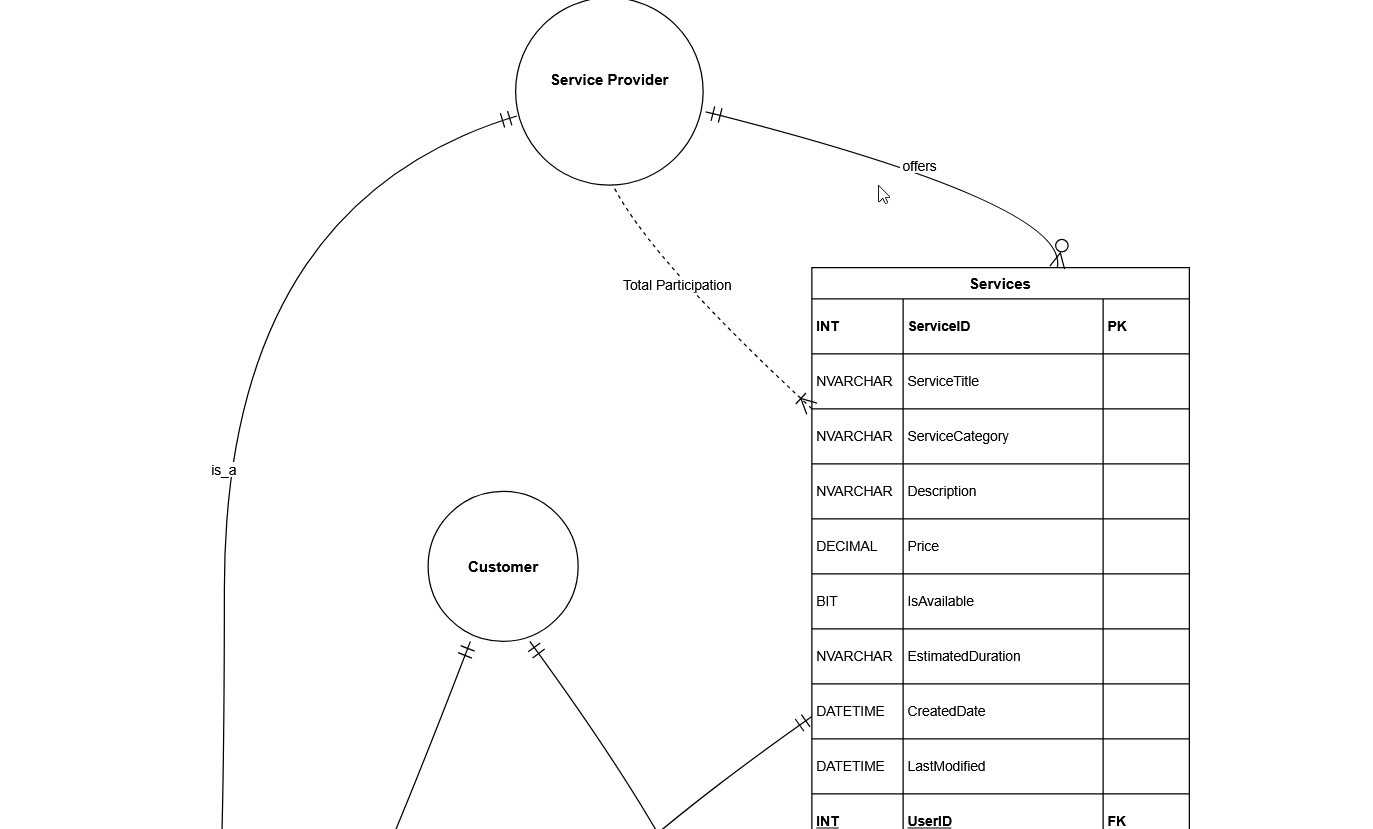
* **Rule**: Not every service will have an audit record unless modified.
* **Explanation**: This is a **partial participation** — a service may or may not have entries in ServiceAudit.
* **Why?**: Only updated, inserted, or deleted services are tracked.

## ➢ Booking Status (Domain Constraint)

* **Rule**: A booking can only be either Pending (0) or Delivered (1).
* **Explanation**: A **domain constraint** applied via a CHECK condition.
* **Why?**: Ensures clean data with limited, meaningful values.

Author: Obaid Ullah

# EERD (Enhanced ERD) of ServiceEase



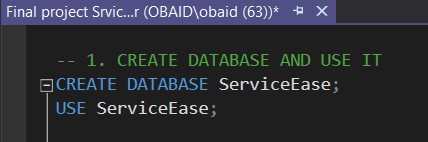
Author: Obaid Ullah

# 5. SQL Queries, Triggers, and Performance Optimization

This section highlights the structured SQL logic, indexing strategies, and data integrity mechanisms used in the **ServiceEase Provider Platform** to ensure secure, performant, and consistent operations across all modules.

## ➢ Database Initialization

The database was created and configured using:



## ➢ SQL Table Definitions

Normalized and relational tables were created for:

* Users
* Services
* Booking
* ServiceAudit

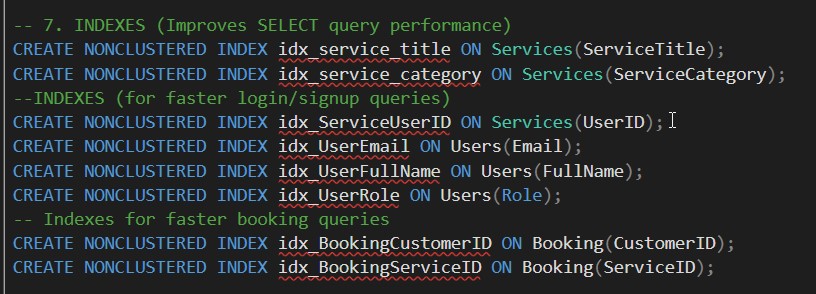
Foreign keys and data types were defined with constraints to ensure **referential integrity and domain validity** (e.g., CHECK constraint on user roles).

## ➢ Indexing for Query Optimization

To improve the performance of frequently executed SELECT, JOIN, and WHERE queries, several **non-clustered indexes** were created:

Author: Obaid Ullah

|  |  |  |
| --- | --- | --- |
| **Table** | **Indexed Columns** | **Purpose** |
| **Users** | Email, FullName, Role | Faster login, filtering, and searches |
| **Services** | ServiceTitle, ServiceCategory, UserID | Efficient service filtering and lookups |
| **Booking** | CustomerID, ServiceID | Optimized booking queries and dashboard views |



## ➢ Trigger-Based Audit Logging

The **trServiceAudit** trigger is a **DML (Data Manipulation Language) trigger** created on the Services table. It activates automatically **after every INSERT, UPDATE, or DELETE** operation on that table. This trigger plays a critical role in **auditing** and **monitoring changes** within the system.

➢ **How it Works:**

* The trigger uses inserted and deleted pseudo-tables to compare new and old data.
* Based on the action (Insert, Update, Delete), it identifies:

o **Action Type** ('INSERT', 'UPDATE', or 'DELETE') o **Old Data** (for updates/deletes) o **New Data** (for inserts/updates)

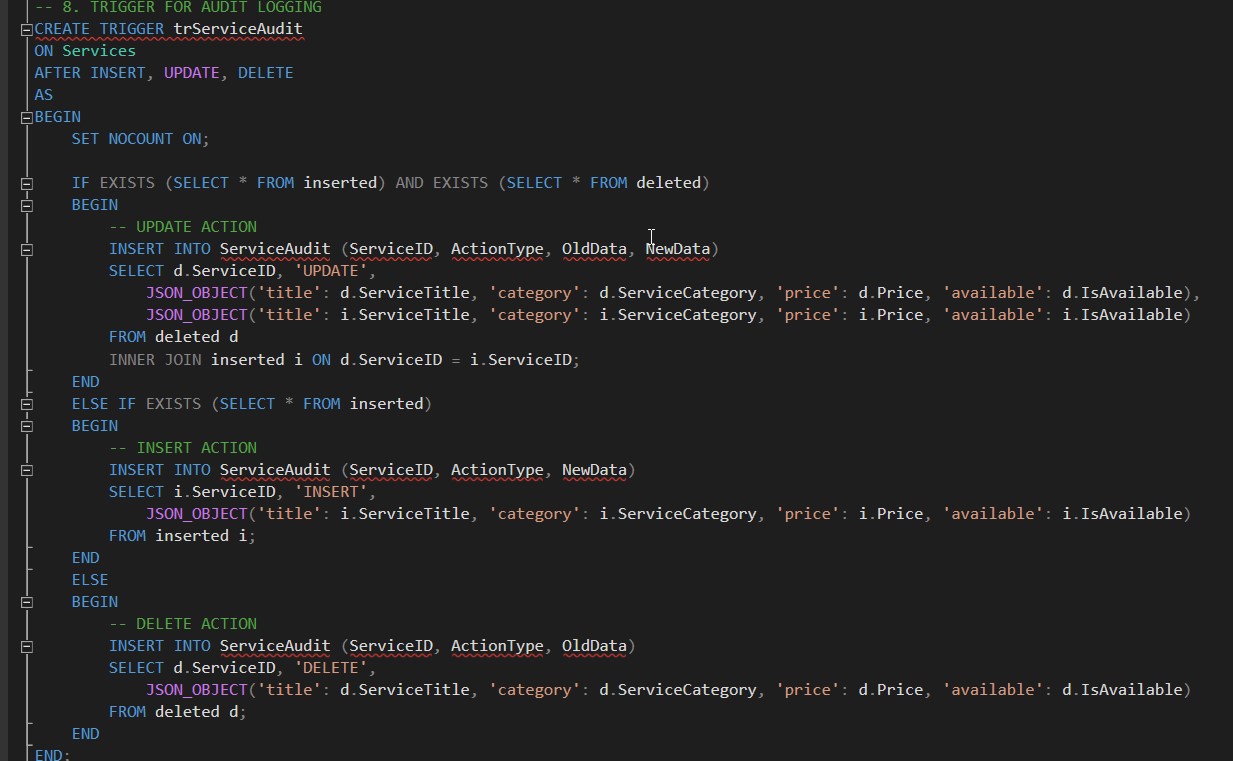
* It then **stores this information as JSON** into the ServiceAudit table using JSON\_OBJECT.

This enhances:

* **Data accountability**
* **Change tracking**
* **Regulatory compliance (audit trail)**

Why Storing data in JSON format helps in Keeping the **structure readable and compact,** Making it easier to **parse, extract, or report on changes** later**,** Supporting future **API or external auditing integrations**

Author: Obaid Ullah



## ➢ Transactions & Concurrency Control:

To ensure **safe and consistent booking operations**, the BookService stored procedure incorporates **robust concurrency control mechanisms** that protect against race conditions and ensure data integrity in multi-user environments.

➢ **How It Works:**

1. **Isolation Level: SERIALIZABLE** o This is the **strictest isolation level** in SQL Server.
   * It **prevents dirty reads, non-repeatable reads, and phantom reads**, meaning:
     + No user can read or modify the same data while another transaction is still processing it.
     + Ensures consistent results and avoids overlapping bookings.
2. **Explicit Transactions** o The procedure starts with BEGIN TRANSACTION and ends with either COMMIT or ROLLBACK.
   * This ensures **all operations (check, update, insert)** happen as a single unit:

Author: Obaid Ullah

* + - If **any part fails**, everything is rolled back (no partial booking).
    - If **all succeed**, the changes are permanently saved.

1. **Row-Level Locking: WITH (UPDLOCK)** o Applies a **row-level update lock** on the service being booked.
   * This ensures that **only one customer can read and lock that row for booking** at a time. o Other users attempting to book the same service must wait until the lock is released.

* **Benefits of This Approach:** 
  + **Prevents double-booking** of the same service by two customers simultaneously.
  + Ensures **data consistency and accuracy**, especially under heavy load or concurrent users.
  + Avoids issues like booking a service that has just become unavailable by another user.
* **Example:**

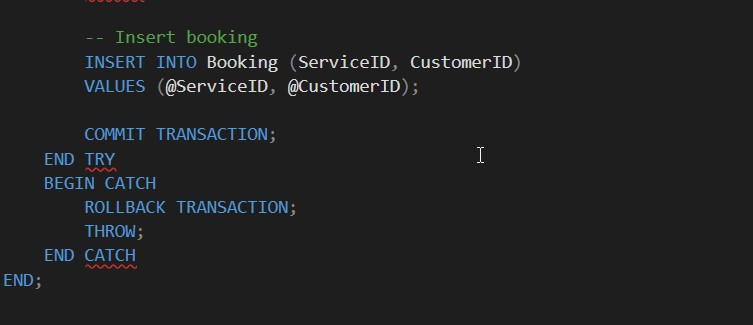
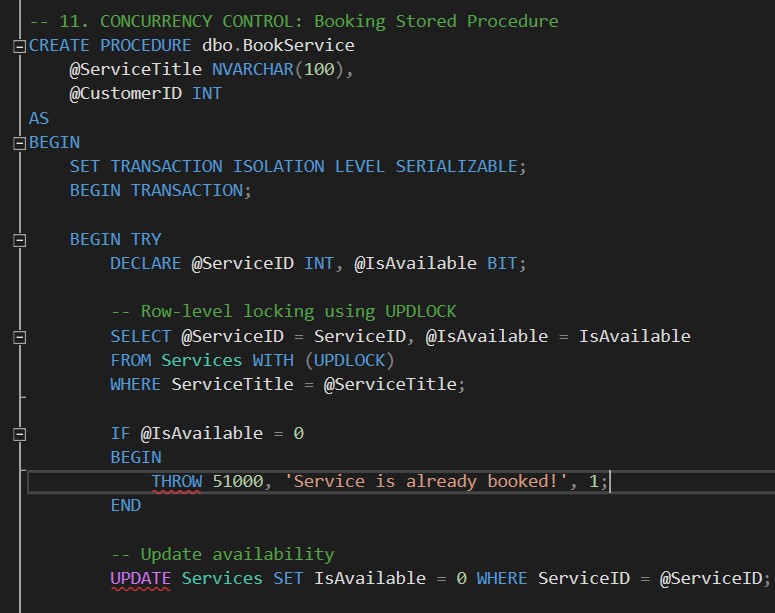
If two customers try to book the same service at the same time:

* + The first one locks the row using WITH (UPDLOCK) and proceeds.
  + The second one **waits** until the lock is released.
  + If the service becomes unavailable during that time, the second transaction will **fail safely**.

Author: Obaid Ullah

**Code:**

Author: Obaid Ullah



## Secure Login & Validation Procedures

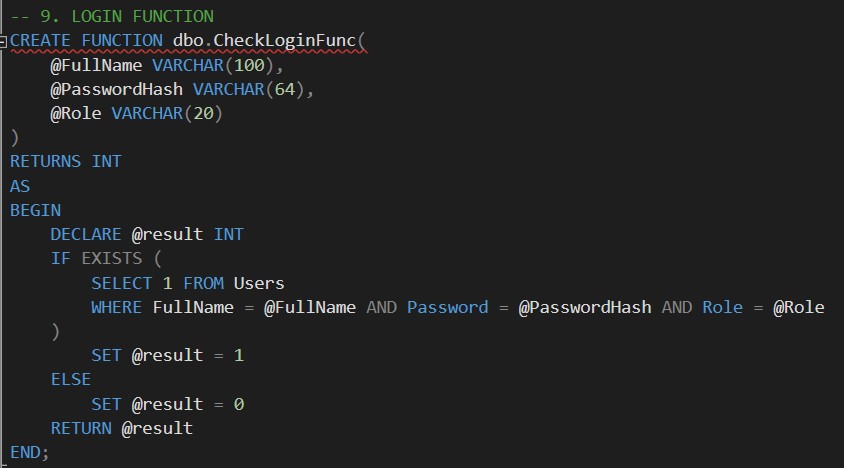
Custom **user-defined functions and stored procedures** are implemented to manage user authentication

|  |  |
| --- | --- |
| **Object Name** | **Description** |
| **CheckLoginFunc** | Function to store user identity |
| **ValidateCustomer** | Procedure for customer validation |
| **ValidateServiceProvider** | Procedure for provider validation |
|  |  |

*Passwords are hashed using SHA-256 and further secured using a unique Salt (NEWID()).*

## ➢ CheckLoginFunc:

Store Full Name, Password, Role from Sign Up interface in SSMs

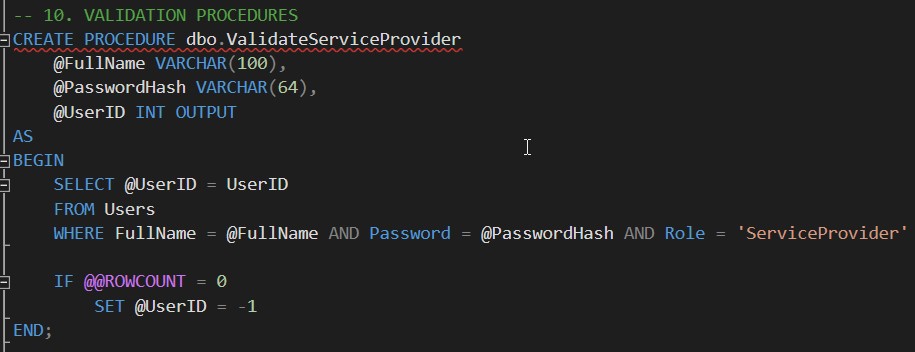


Author: Obaid Ullah

##  LOGIN FUNCTION:

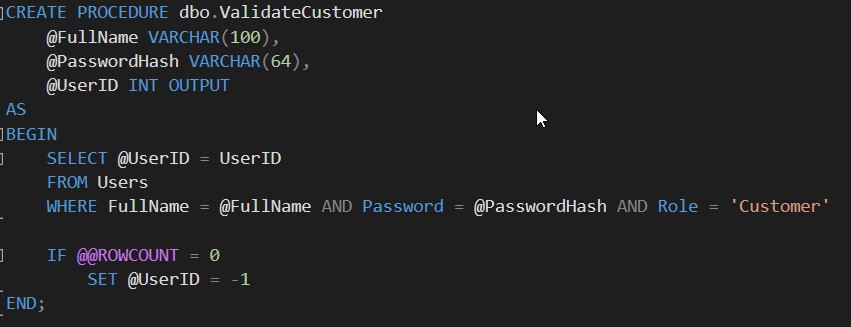
## ➢ ValidateServiceProvider

Match fullName, Password and role as service provider in Login Interdace.



## ➢ ValidateCustomer:

Match fullName, Password and role as Customer in Login Interdace.



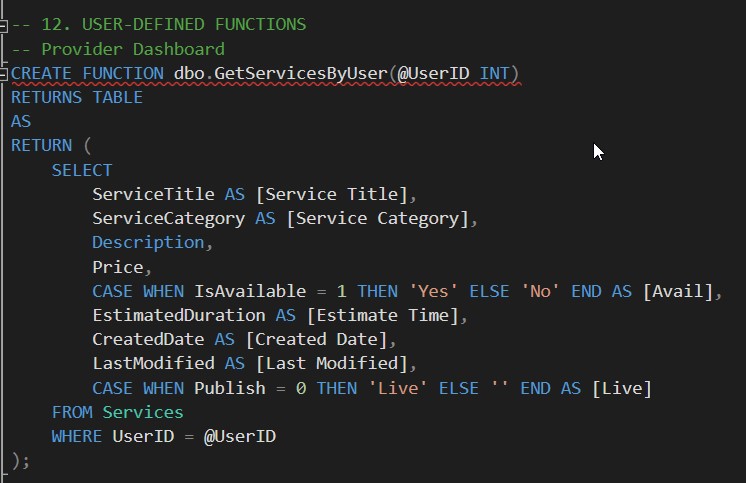
Author: Obaid Ullah

## Analytical Views and Functions

To support front-end dashboard functionality and modular querying, several **table-valued functions** were created:

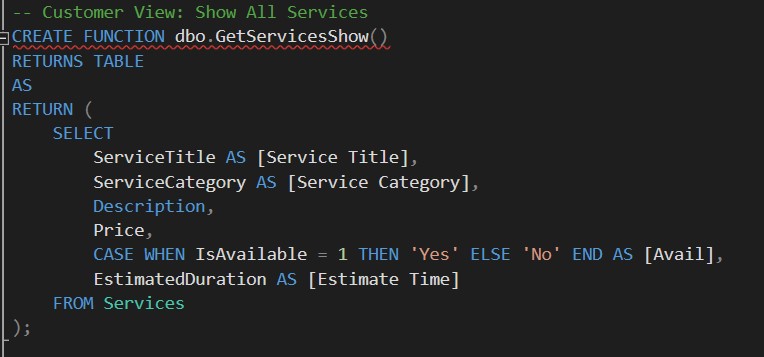
|  |  |
| --- | --- |
| **Function Name** | **Purpose** |
| **GetServicesByUser** | For provider dashboard |
| **GetServicesShow** | To display all available services |
| **GetBookedServicesByProvider** | Show all bookings under a provider |
| **GetBookedServicesByCustomer** | Show bookings for a specific customer |

## ➢ GetServicesByUser:{ For provider dashboard }

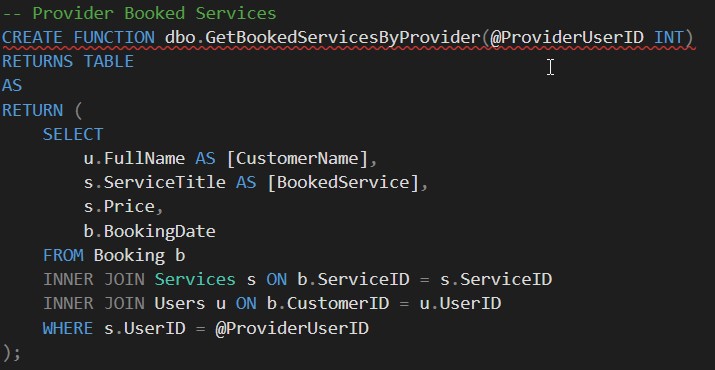


Author: Obaid Ullah

## GetServicesShow {To display all available services}:

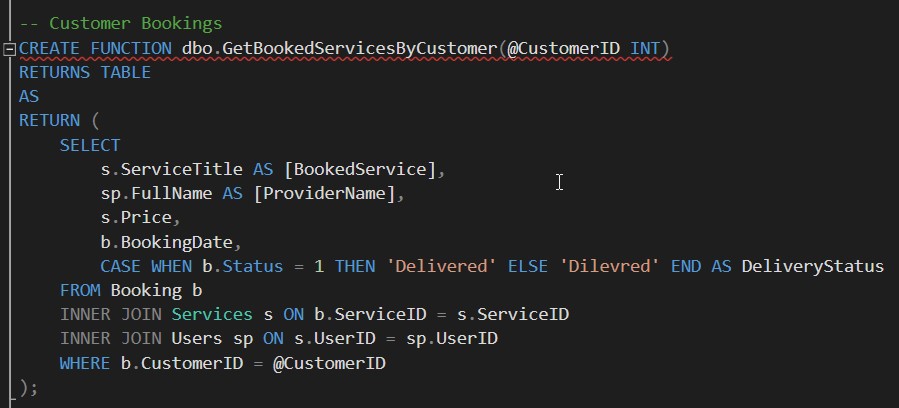


## ➢ GetBookedServicesByProvider{ Show all bookings under a provider }:

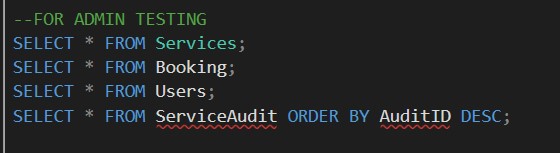


Author: Obaid Ullah

## GetBookedServicesByCustomer: Show bookings for a specific customer

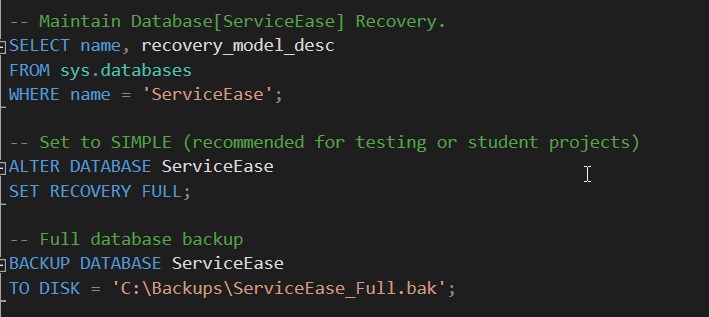


## ➢ For Admin Testing command:



## ➢ Backup & Recovery Strategy

A full backup was performed using the following SQL Server script:

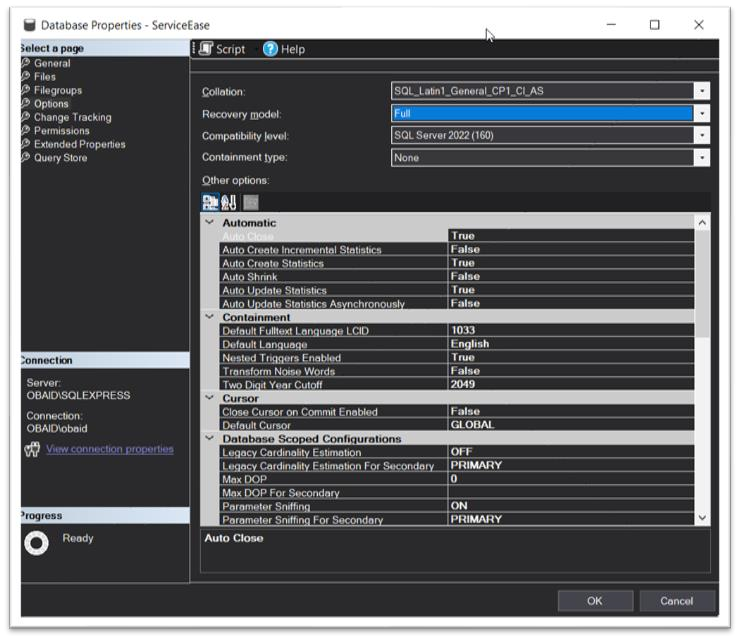


Author: Obaid Ullah

We configure the **Recovery Model** of the ServiceEase database using the graphical interface in SSMS, follow these steps:

1. **Open SQL Server Management Studio (SSMS).**
2. In the **Object Explorer**, expand the **Databases** node.
3. Right-click on the ServiceEase database and select **Properties**.
4. In the **Database Properties** window, click on the **Options** tab.
5. Locate the **Recovery Model** drop-down menu.
6. Select the desired recovery model — typically:

o Simple (for basic recovery without log backups), o or Full (for point-in-time recovery with transaction log backups).



Author: Obaid Ullah

## Backup creation:

The recovery model is set to FULL to ensure **point-in-time recovery** in case of data loss.

## ➢ Security/authorization and Role-Based User Access

* The login mechanism uses **hashed passwords (SHA-256)** combined with random

Salt values for extra security.

* Users are authenticated and verified based on:

o FullName, PasswordHash, and Role.

* Post-login access is **role-restricted** (Customer, ServiceProvider, Admin), ensuring that users only interact with features relevant to their responsibilities.

# 6. Summary of Recovery Techniques in SQL Server

SQL Server provides multiple recovery techniques to ensure data protection, minimize data loss, and support business continuity. These techniques are based on the database **recovery model** selected and include the following:

1. **Recovery Models**

SQL Server supports three types of recovery models, which determine how transaction logs are maintained and how recovery operations are performed:

Author: Obaid Ullah

* + **Simple Recovery Model** o Minimal logging; log space is reused automatically.
    - No transaction log backups are possible.
    - Suitable for development or small-scale applications.
    - Point-in-time recovery **not supported**.
  + **Full Recovery Model** o All transactions are fully logged.
    - Requires regular transaction log backups.
    - Supports **point-in-time recovery**, making it ideal for production systems with critical data.
  + **Bulk-Logged Recovery Model**
    - Minimizes logging for bulk operations (e.g., bulk inserts). o Supports recovery but with some limitations for point-in-time recovery.
    - A balance between performance and logging detail.

1. **Backup Types**

SQL Server uses various types of backups as part of its recovery strategy:

* + **Full Backup**

Captures the entire database at a point in time.

* + **Differential Backup**

Captures only the data changed since the last full backup.

* + **Transaction Log Backup**

Captures all transactions since the last log backup (available only in Full/BulkLogged recovery models).

1. **Restore Operations**

Using backups, SQL Server supports several types of restore operations:

* + **Database Restore**

Recover the entire database using a full backup.

Author: Obaid Ullah

* + **Point-in-Time Restore**

Restore the database to a specific moment using full + log backups (only in Full model).

* + **Page Restore**

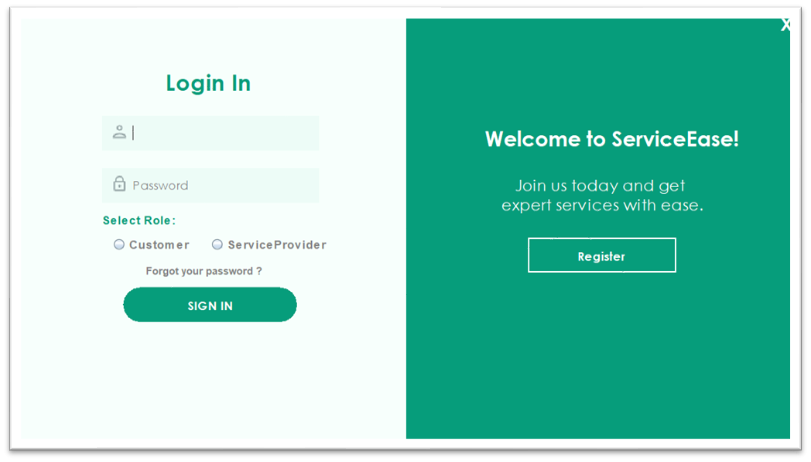
Recover individual corrupted pages without restoring the full database.

1. **Best Practices for Recovery** 
   * Always choose the appropriate recovery model based on the system’s requirements.
   * Schedule regular backups (full, differential, and log).
   * Store backups in secure, reliable, and accessible locations.
   * Test restore procedures periodically to ensure readiness.

# GUI interface:

## ➢ Login Interface:

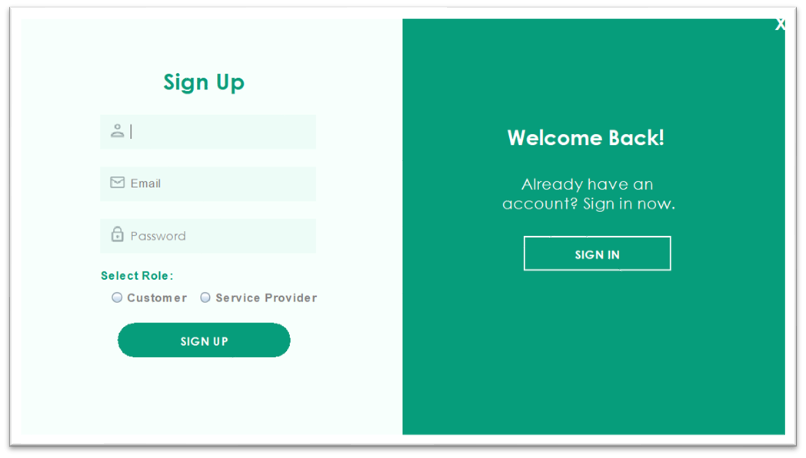
On the login screen, the user must first register by providing their details and selecting whether they are a Customer or a Service Provider by clicking Register button.



Author: Obaid Ullah

## ➢ Signup:

On the sign-up page, the user will enter their name, email, and password, select their role (Customer or Service Provider), and upon clicking the Sign-Up button, the entered data will be stored in SQL Server Management Studio (SSMS) using a user defined function.

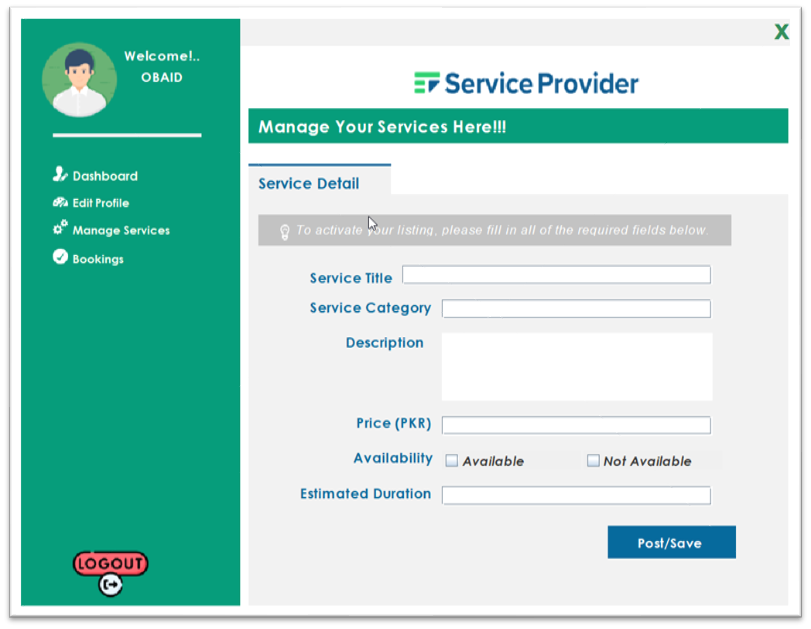


Author: Obaid Ullah

## Service Interface:

"In the Service Provider interface, the user can post services by filling out the entry fields.

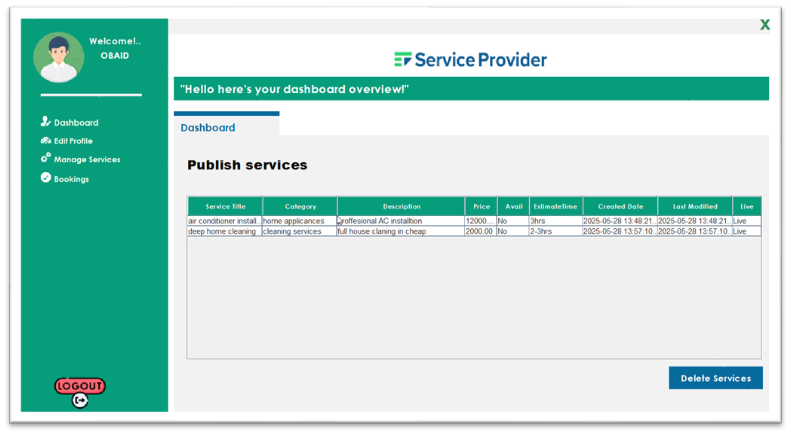
When the 'Post and Save' button is clicked, the data is stored in SQL Server Management Studio (SSMS) by calling a predefined SQL function."



## ➢ Dashboard button on service provider interface:

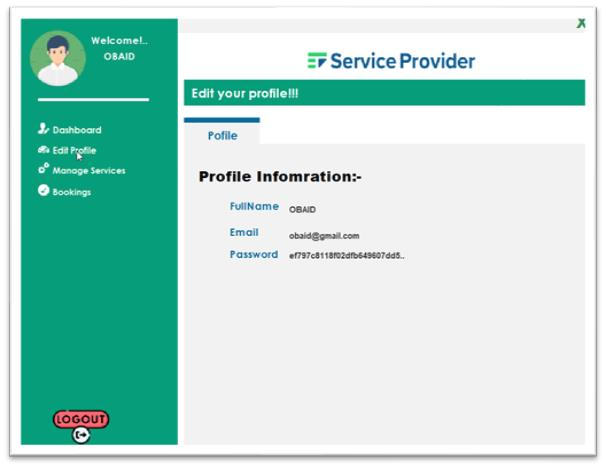
"In the Services section, when the user clicks the Dashboard button on the left side, they can view and track their posted services—whether they are live, pending update, or inactive. This is achieved by calling a SQL function , activity log in SQL Server that retrieves and displays the service status."

Author: Obaid Ullah



## ➢ Edit Profile interface:

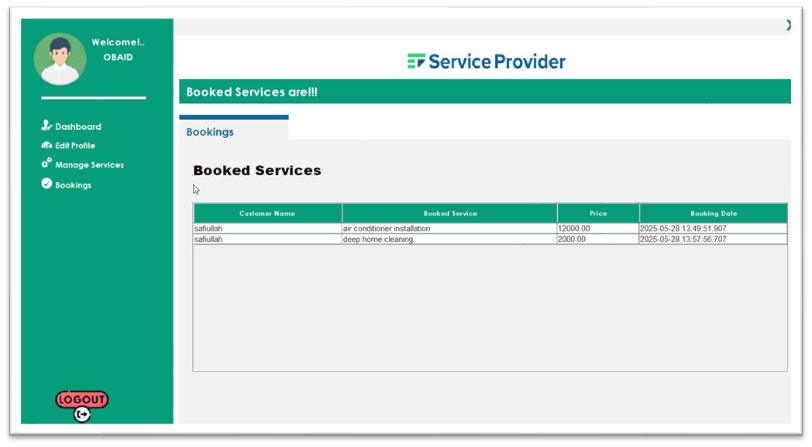
"In the Edit Profile section, the user can view their full name, email, password, and other details. If they have forgotten any information, it is retrieved from the database using a SQL service."



Author: Obaid Ullah

## Booking button Service Ease provider:

In the Booking section of the Service Provider interface, the provider can view all bookings made—who booked the service, when it was booked, and the related payment details."



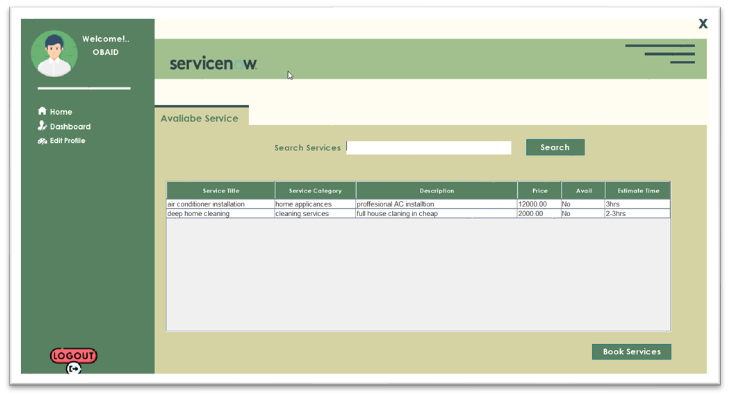
## ➢ Logout:

By pressing logout they move to login page.

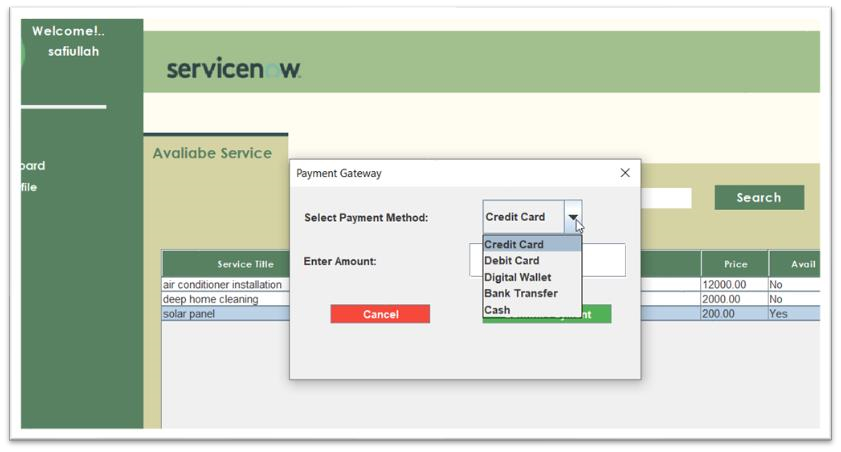
Author: Obaid Ullah

# Customer Interface:

"In the Customer interface, the user can search for available services and make bookings. During the booking process, the customer will also select a preferred payment method. Like debit card, bank transfer etc. "



## ➢ Payment interface



Author: Obaid Ullah

## Dashboard Customer Interface:

"In the Dashboard, the customer can check the delivery status of their bookings after making a payment. They can also view details such as which provider is assigned, the booking date, and all related activities through the Activity Log."



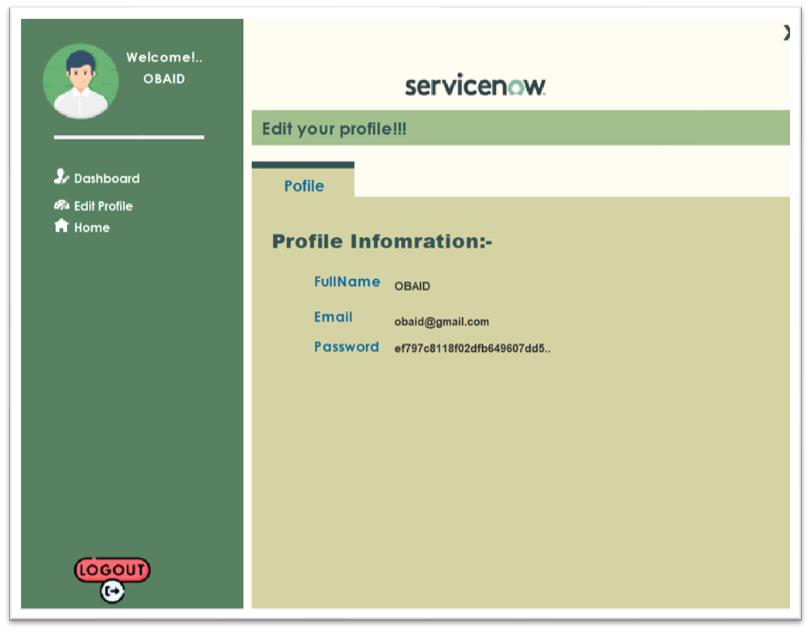
## ➢ Logout:

By clicking logout move to login interface.

Author: Obaid Ullah

## Edit Profile customer:

"In the Edit Profile section, the customer can view their full name, email, password, and other details. If they have forgotten any information they can easily view on that interface with the help from SQL service."



Author: Obaid Ullah