NATIONALUNIVERSITY OF MODERN LANGUAGES ISLAMABAD

Github linkedin

SERVICE EASE PLATFORM

Service Ease Platform (Desktop Application)

Submitted By: **Obaid Ullah**

Department: **BS Information Technology**

Session: 20**23** – 20**27**

GithubProjectLink

Table of Contents

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.

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.
2. **Login System with Role Selection:**
   Secure login with role-based redirection (using hashed passwords and validation).
3. **Service Posting and Management (Providers):**
   Providers can post services with title, category, description, availability, price, and estimated duration.
4. **Booking System (Customers):**
   Customers can search and book services. Booked services are stored and trackable.
5. **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.

Potential Challenges:

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

Conclusion:

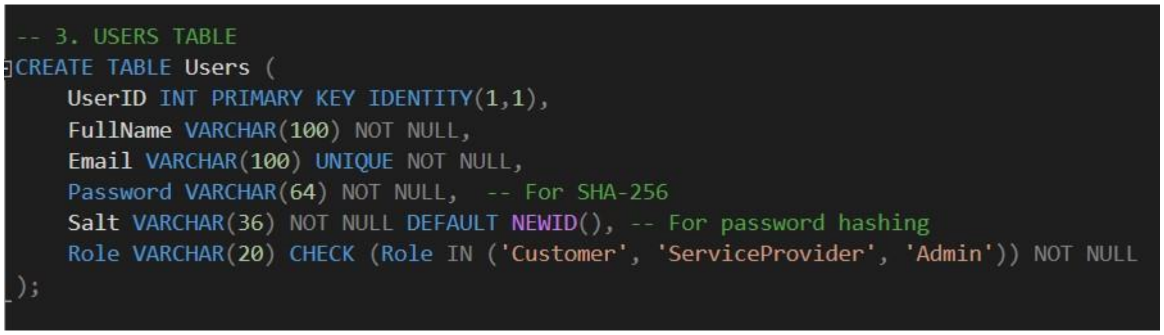
ServiceEase is a scalable and versatile platform designed to simplify the process of finding and managing services. By providing a centralized solution, it aims to create value for both service seekers and providers, fostering trust and convenience in the service industry.

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

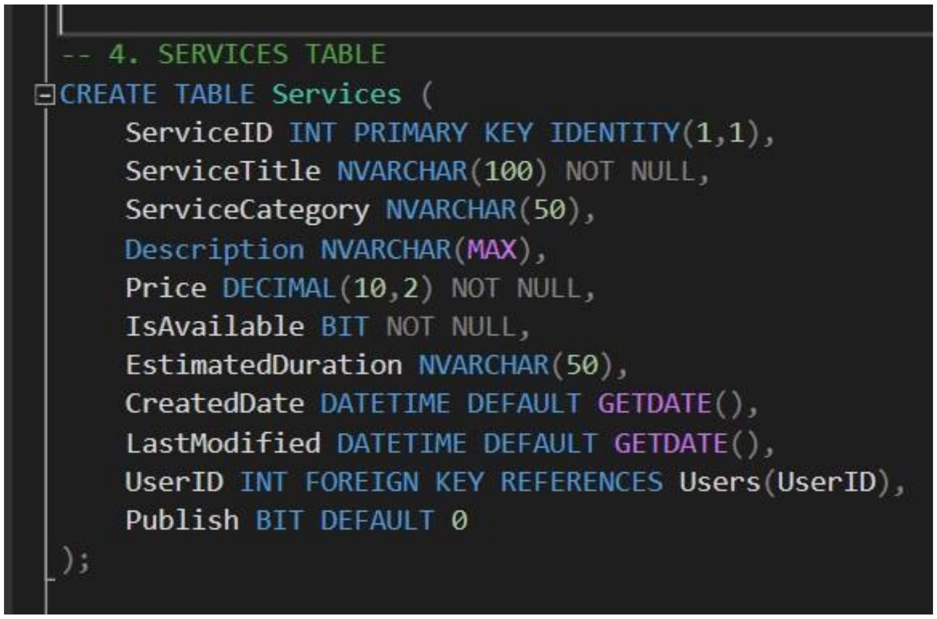
Attribute	Description
UserID (PK)	Unique identifier for each user
FullName	Full name of the user
Email	Unique email address
Password	Securely stored hashed password (SHA-256)
Role	User type: Customer, ServiceProvider



```sql
-- 3. USERS TABLE
CREATE TABLE Users (
    UserID INT PRIMARY KEY IDENTITY(1,1),
    FullName VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Password VARCHAR(64) NOT NULL,  -- For SHA-256
    Salt VARCHAR(36) NOT NULL DEFAULT NEWID(), -- For password hashing
    Role VARCHAR(20) CHECK (Role IN ('Customer', 'ServiceProvider', 'Admin')) NOT NULL
);
```

2.1. Services Table – Services Offered by Providers

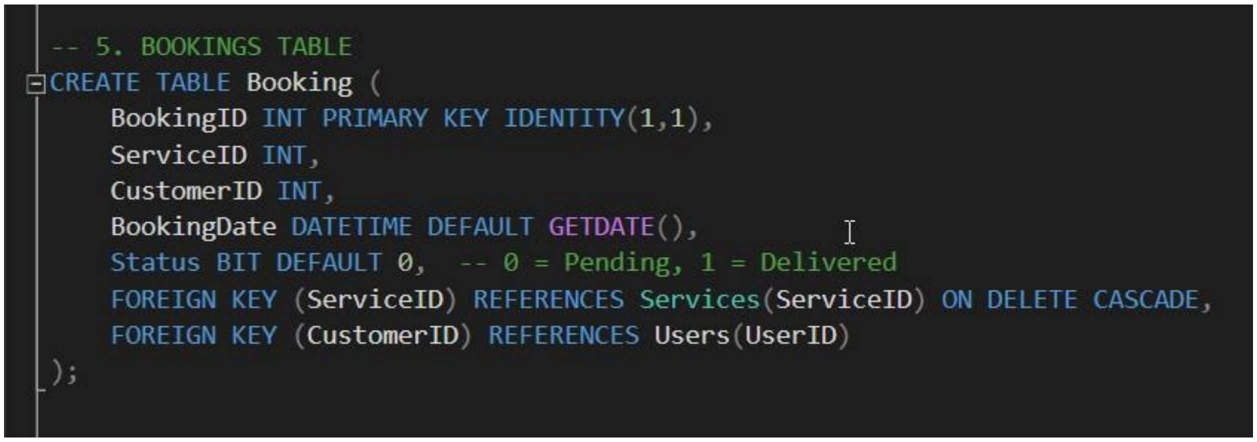
Attribute	Description
ServiceID (PK)	Unique identifier for each service
ServiceTitle	Title of the service (e.g., "AC Installation")
ServiceCategory	Category of the service (e.g., "Home Appliances")
Price	Service cost in PKR
IsAvailable	Availability status: 1 for Available, 0 for Booked
UserID (FK)	Linked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



```sql
-- 4. SERVICES TABLE
CREATE TABLE Services (
    ServiceID INT PRIMARY KEY IDENTITY(1,1),
    ServiceTitle NVARCHAR(100) NOT NULL,
    ServiceCategory NVARCHAR(50),
    Description NVARCHAR(MAX),
    Price DECIMAL(10,2) NOT NULL,
    IsAvailable BIT NOT NULL,
    EstimatedDuration NVARCHAR(50),
    CreatedDate DATETIME DEFAULT GETDATE(),
    LastModified DATETIME DEFAULT GETDATE(),
    UserID INT FOREIGN KEY REFERENCES Users(UserID),
    Publish BIT DEFAULT 0
);
```

Booking Table – Customer Bookings Record

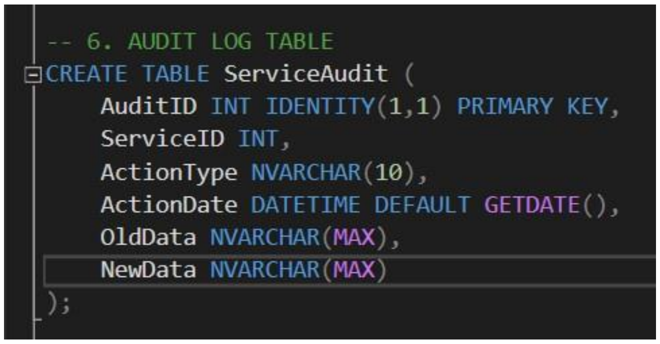
Attribute	Description
BookingID (PK)	Unique booking identifier
ServiceID (FK)	References the booked service (Services.ServiceID)
CustomerID (FK)	References the customer (Users.UserID)
Status	0 = Pending, 1 = Delivered



```
-- 5. BOOKINGS TABLE
CREATE TABLE Booking (
    BookingID INT PRIMARY KEY IDENTITY(1,1),
    ServiceID INT,
    CustomerID INT,
    BookingDate DATETIME DEFAULT GETDATE(),
    Status BIT DEFAULT 0,  -- 0 = Pending, 1 = Delivered
    FOREIGN KEY (ServiceID) REFERENCES Services(ServiceID) ON DELETE CASCADE,
    FOREIGN KEY (CustomerID) REFERENCES Users(UserID)
);
```

- **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



```
-- 6. AUDIT LOG TABLE
CREATE TABLE ServiceAudit (
    AuditID INT IDENTITY(1,1) PRIMARY KEY,
    ServiceID INT,
    ActionType NVARCHAR(10),
    ActionDate DATETIME DEFAULT GETDATE(),
    OldData NVARCHAR(MAX),
    NewData NVARCHAR(MAX)
);
```

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.

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.

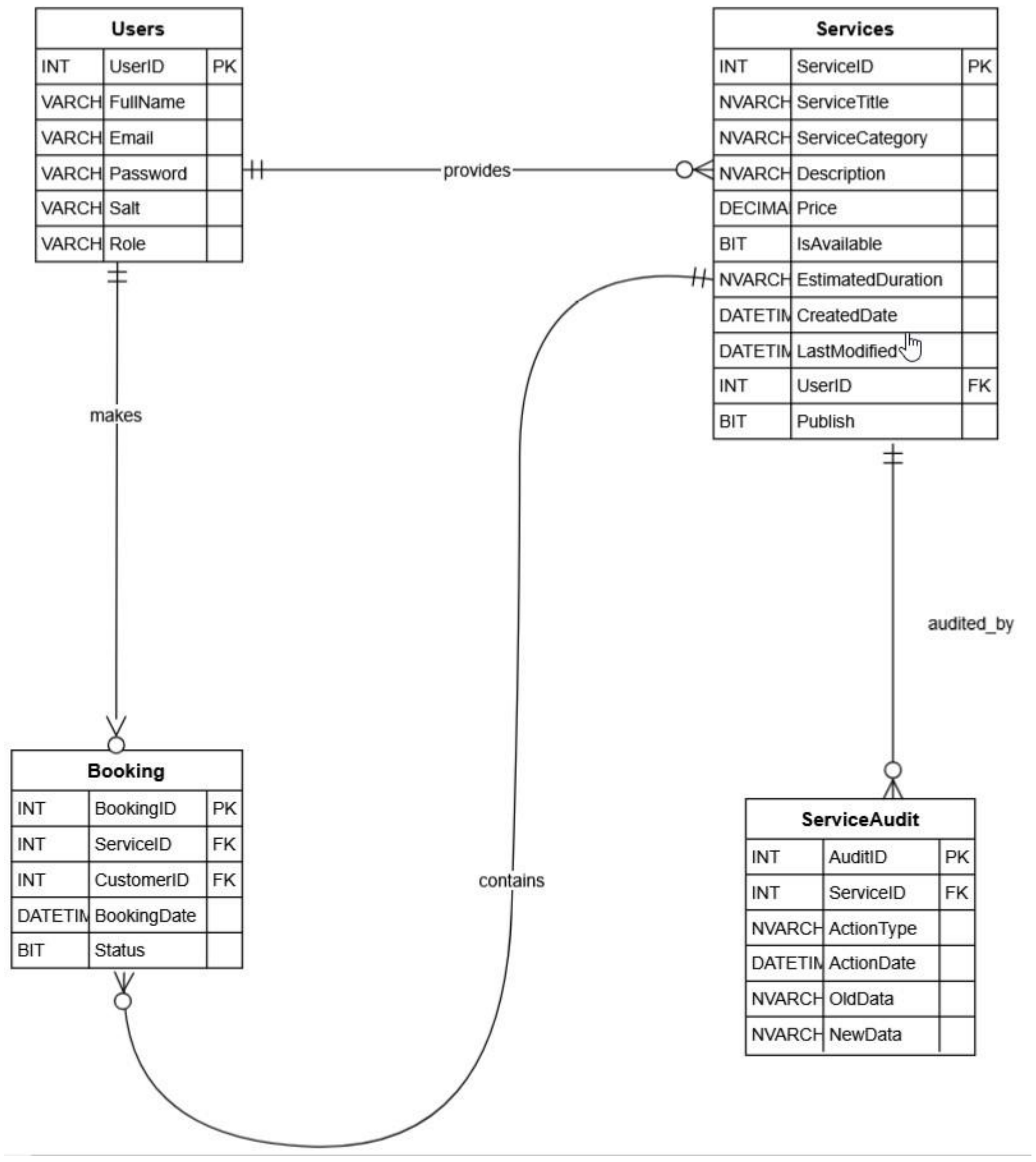
Services to Booking (One-to-Many)

- **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.

ERD (Entity Relationship Diagram) of ServiceEase:



Users		
INT	UserID	PK
VARCH	FullName	
VARCH	Email	
VARCH	Password	
VARCH	Salt	
VARCH	Role	

Services		
INT	ServiceID	PK
NVARCH	ServiceTitle	
NVARCH	ServiceCategory	
NVARCH	Description	
DECIMA	Price	
BIT	IsAvailable	
NVARCH	EstimatedDuration	
DATETIN	CreatedDate	
DATETIN	LastModified	
INT	UserID	FK
BIT	Publish	

Booking		
INT	BookingID	PK
INT	ServiceID	FK
INT	CustomerID	FK
DATETIN	BookingDate	
BIT	Status	

ServiceAudit		
INT	AuditID	PK
INT	ServiceID	FK
NVARCH	ActionType	
DATETIN	ActionDate	
NVARCH	OldData	
NVARCH	NewData	

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.

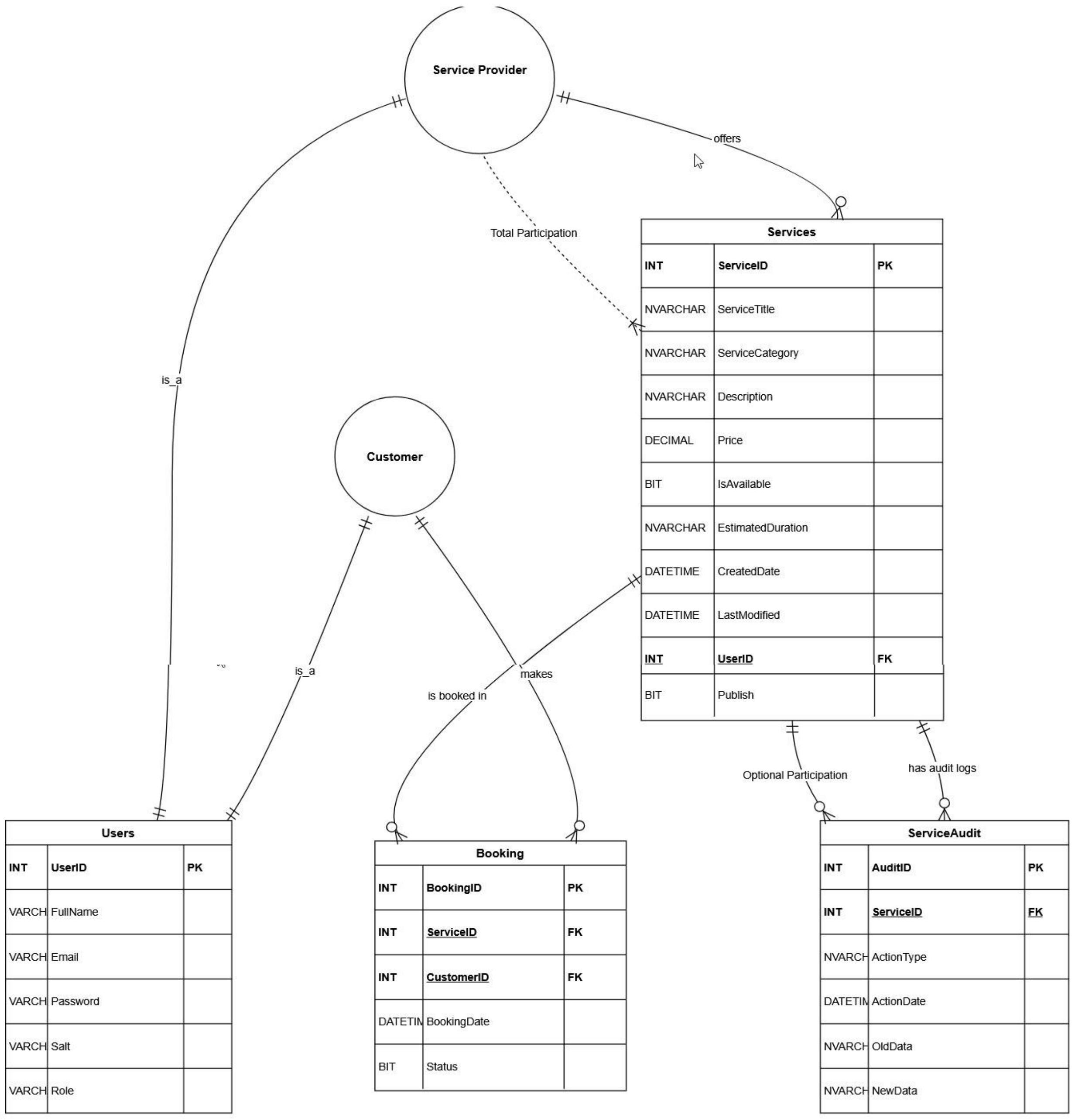
Audit Mechanism (Optional Participation)

- **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.

EERD (Enhanced ERD) of ServiceEase

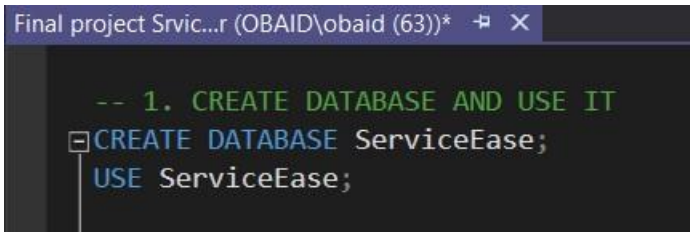


5. SQL Queries, Triggers, and Performance Optimization

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:



```
-- 1. CREATE DATABASE AND USE IT
CREATE DATABASE ServiceEase;
USE ServiceEase;
```

○ SQL Table Definitions

Normalized and relational tables were created for:

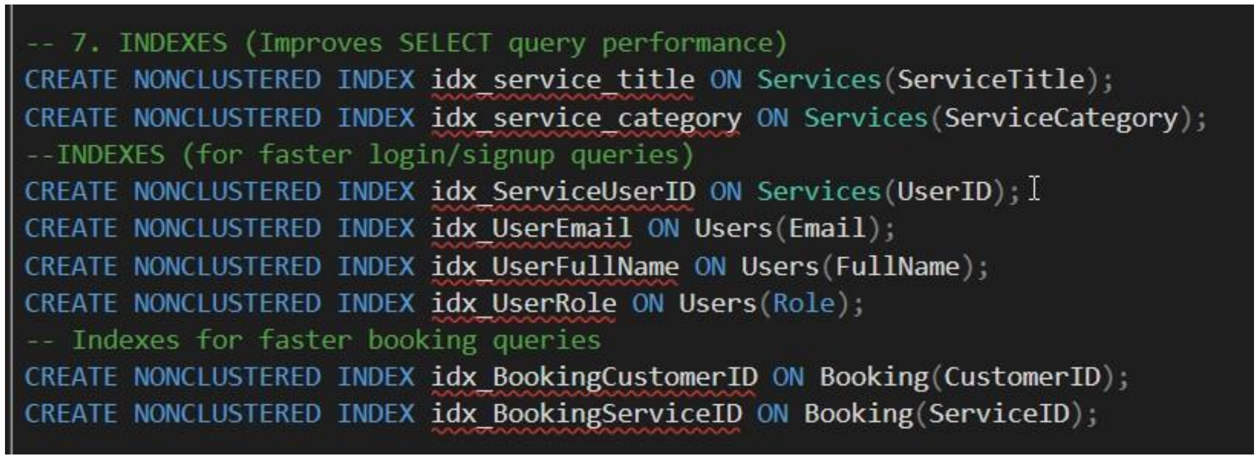
- Users
- Services
- Booking
- ServiceAudit

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:

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


```
-- 7. INDEXES (Improves SELECT query performance)
CREATE NONCLUSTERED INDEX idx_service_title ON Services(ServiceTitle);
CREATE NONCLUSTERED INDEX idx_service_category ON Services(ServiceCategory);
--INDEXES (for faster login/signup queries)
CREATE NONCLUSTERED INDEX idx_ServiceUserID ON Services(UserID);
CREATE NONCLUSTERED INDEX idx_UserEmail ON Users(Email);
CREATE NONCLUSTERED INDEX idx_UserFullName ON Users(FullName);
CREATE NONCLUSTERED INDEX idx_UserRole ON Users(Role);
-- Indexes for faster booking queries
CREATE NONCLUSTERED INDEX idx_BookingCustomerID ON Booking(CustomerID);
CREATE NONCLUSTERED INDEX idx_BookingServiceID ON Booking(ServiceID);
```

○ 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:

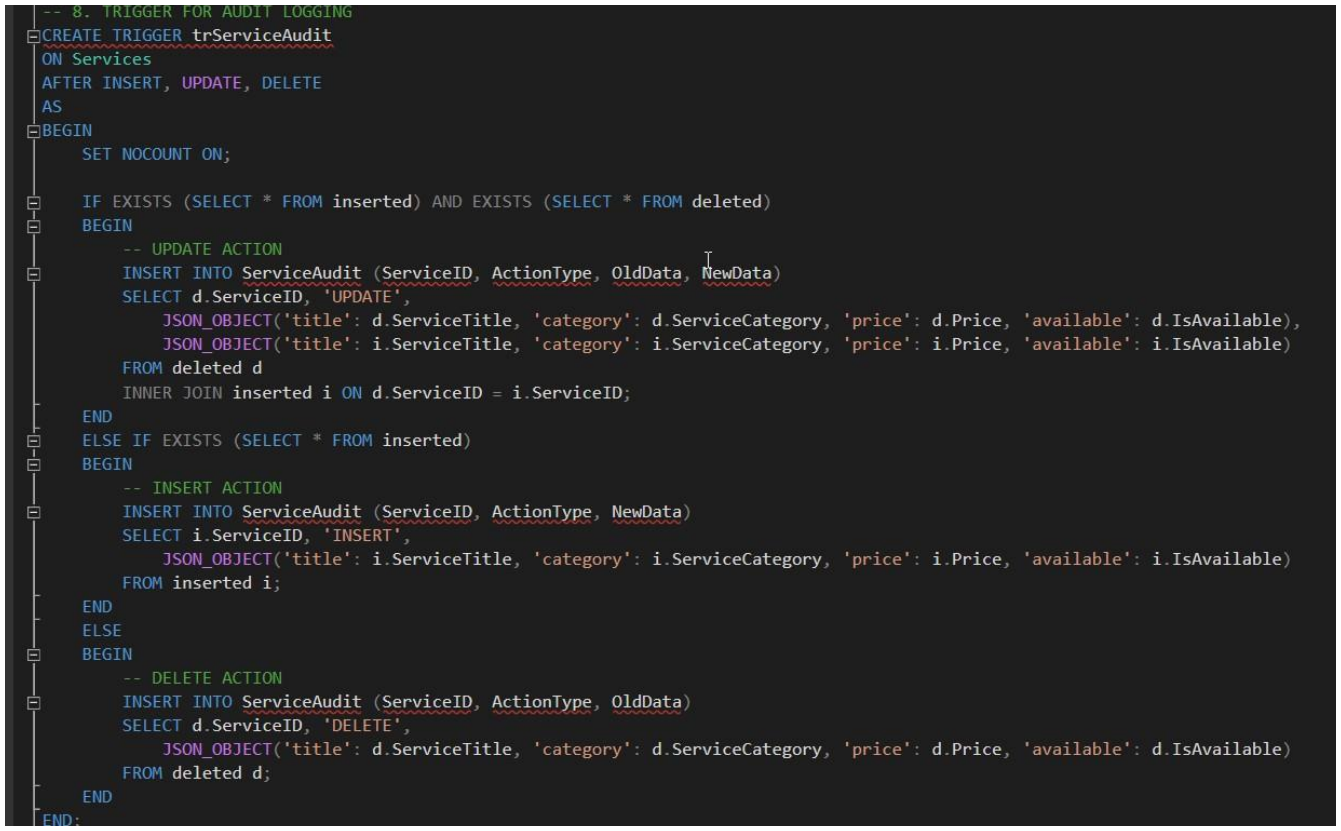
  ○ **Action Type** ('INSERT', 'UPDATE', or 'DELETE') ○ **Old Data** (for updates/deletes) ○ **New Data** (for inserts/updates)

- 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**



```sql
-- 8. TRIGGER FOR AUDIT LOGGING
CREATE TRIGGER trServiceAudit
ON Services
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        -- UPDATE ACTION
        INSERT INTO ServiceAudit (ServiceID, ActionType, OldData, NewData)
        SELECT d.ServiceID, 'UPDATE',
            JSON_OBJECT('title': d.ServiceTitle, 'category': d.ServiceCategory, 'price': d.Price, 'available': d.IsAvailable),
            JSON_OBJECT('title': i.ServiceTitle, 'category': i.ServiceCategory, 'price': i.Price, 'available': i.IsAvailable)
        FROM deleted d
        INNER JOIN inserted i ON d.ServiceID = i.ServiceID;
    END
    ELSE IF EXISTS (SELECT * FROM inserted)
    BEGIN
        -- INSERT ACTION
        INSERT INTO ServiceAudit (ServiceID, ActionType, NewData)
        SELECT i.ServiceID, 'INSERT',
            JSON_OBJECT('title': i.ServiceTitle, 'category': i.ServiceCategory, 'price': i.Price, 'available': i.IsAvailable)
        FROM inserted i;
    END
    ELSE
    BEGIN
        -- DELETE ACTION
        INSERT INTO ServiceAudit (ServiceID, ActionType, OldData)
        SELECT d.ServiceID, 'DELETE',
            JSON_OBJECT('title': d.ServiceTitle, 'category': d.ServiceCategory, 'price': d.Price, 'available': d.IsAvailable)
        FROM deleted d;
    END
END;
```

○ 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** ○ 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** ○ 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:

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

3. **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.
   - Other users attempting to book the same service must wait until the lock is released.

Benefits of This Approach:

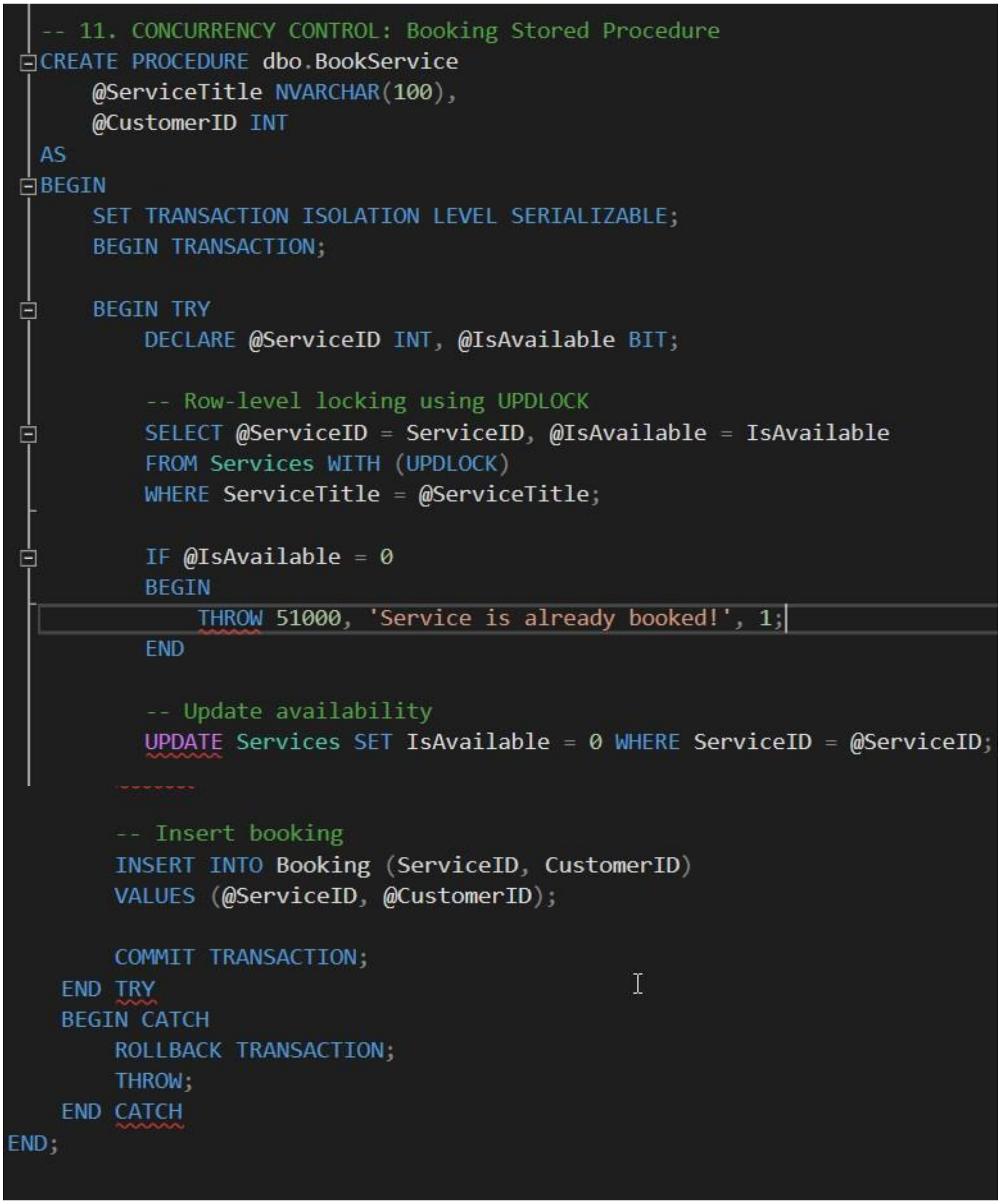
- **Prevents double-booking** of the same service by two customers simultaneously.
- Ensures **data consistency and accuracy**, especially under heavy load or concurrent users.
- Avoids issues like booking a service that has just become unavailable by another user.

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**.

Code:



```
-- 11. CONCURRENCY CONTROL: Booking Stored Procedure
CREATE PROCEDURE dbo.BookService
    @ServiceTitle NVARCHAR(100),
    @CustomerID INT
AS
BEGIN
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    BEGIN TRANSACTION;

    BEGIN TRY
        DECLARE @ServiceID INT, @IsAvailable BIT;

        -- Row-level locking using UPDLOCK
        SELECT @ServiceID = ServiceID, @IsAvailable = IsAvailable
        FROM Services WITH (UPDLOCK)
        WHERE ServiceTitle = @ServiceTitle;

        IF @IsAvailable = 0
        BEGIN
            THROW 51000, 'Service is already booked!', 1;
        END

        -- Update availability
        UPDATE Services SET IsAvailable = 0 WHERE ServiceID = @ServiceID;

      -- Insert booking
      INSERT INTO Booking (ServiceID, CustomerID)
      VALUES (@ServiceID, @CustomerID);

      COMMIT TRANSACTION;
  END TRY
  BEGIN CATCH
      ROLLBACK TRANSACTION;
      THROW;
  END CATCH
END;
```

o

Secure Login & Validation Procedures

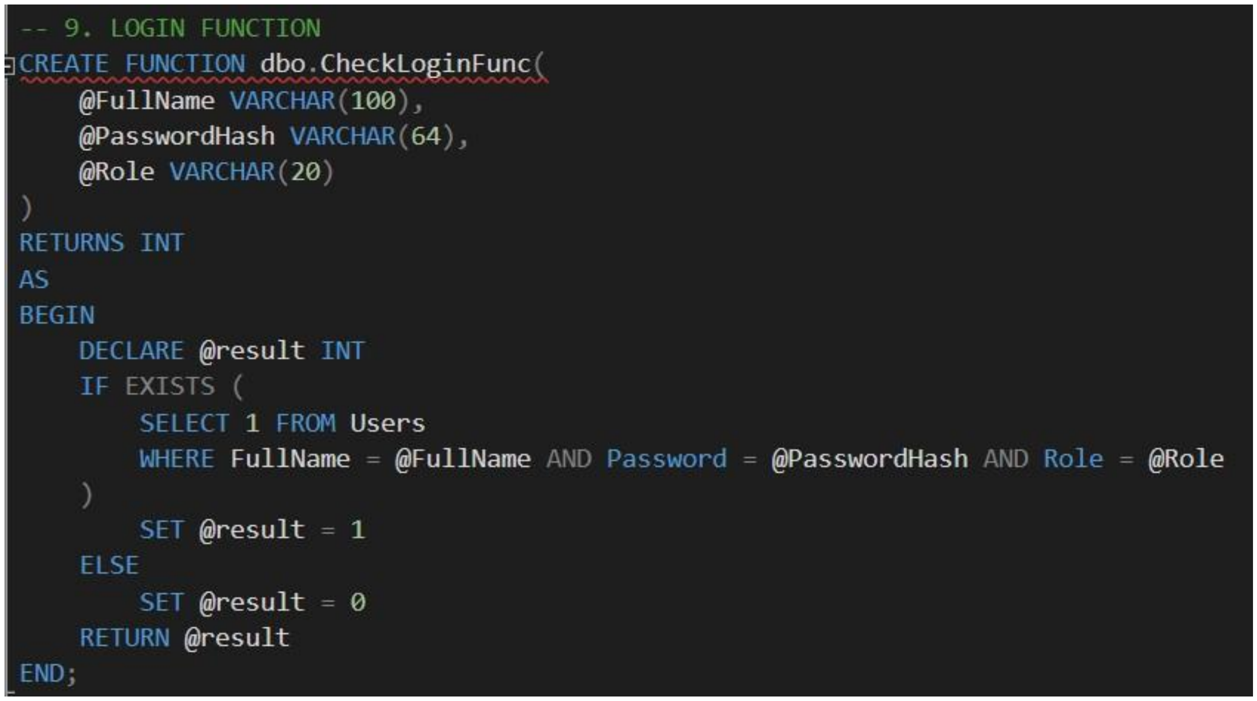
Custom **user-defined functions and stored procedures** are implemented to manage user authentication

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()).

o CheckLoginFunc:

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

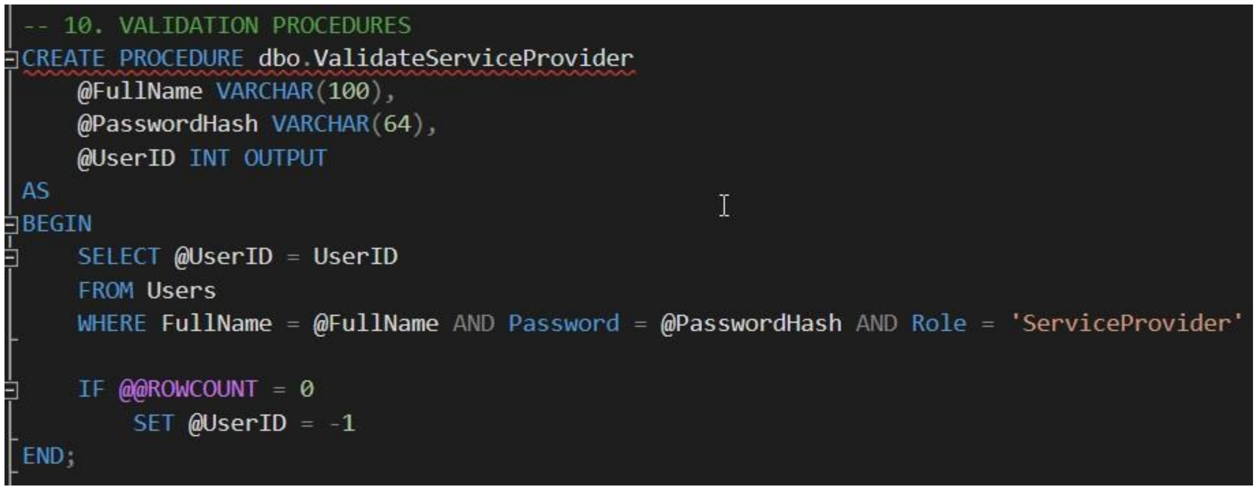


```
-- 9. LOGIN FUNCTION
CREATE FUNCTION dbo.CheckLoginFunc(
    @FullName VARCHAR(100),
    @PasswordHash VARCHAR(64),
    @Role VARCHAR(20)
)
RETURNS INT
AS
BEGIN
    DECLARE @result INT
    IF EXISTS (
        SELECT 1 FROM Users
        WHERE FullName = @FullName AND Password = @PasswordHash AND Role = @Role
    )
        SET @result = 1
    ELSE
        SET @result = 0
    RETURN @result
END;
```

⇨ LOGIN FUNCTION:

○ ValidateServiceProvider

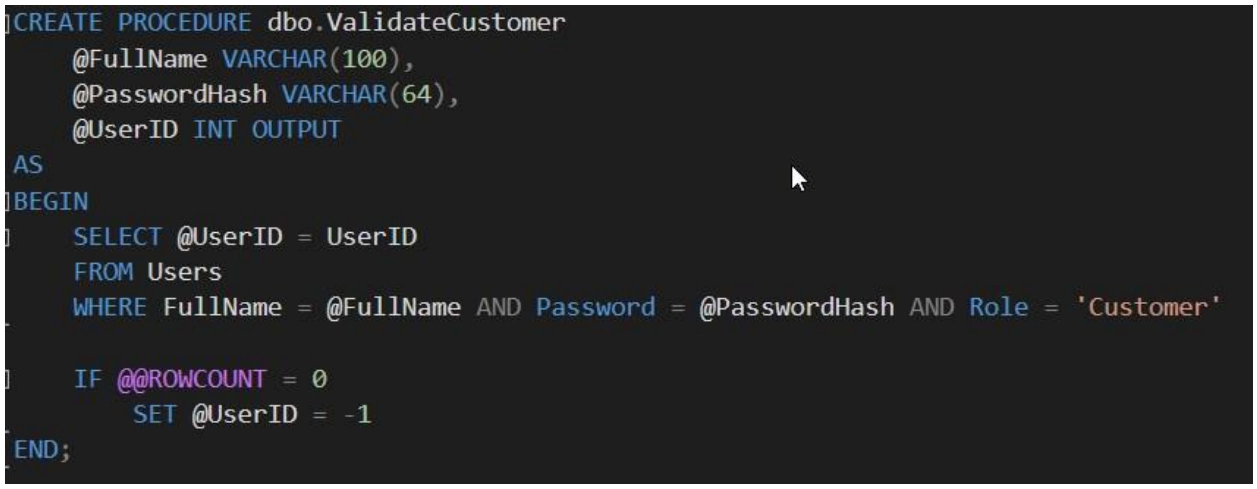
Match fullName, Password and role as service provider in Login Interdace.



```
-- 10. VALIDATION PROCEDURES
CREATE PROCEDURE dbo.ValidateServiceProvider
    @FullName VARCHAR(100),
    @PasswordHash VARCHAR(64),
    @UserID INT OUTPUT
AS
BEGIN
    SELECT @UserID = UserID
    FROM Users
    WHERE FullName = @FullName AND Password = @PasswordHash AND Role = 'ServiceProvider'

    IF @@ROWCOUNT = 0
        SET @UserID = -1
END;
```

○ ValidateCustomer:

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



```
CREATE PROCEDURE dbo.ValidateCustomer
    @FullName VARCHAR(100),
    @PasswordHash VARCHAR(64),
    @UserID INT OUTPUT
AS
BEGIN
    SELECT @UserID = UserID
    FROM Users
    WHERE FullName = @FullName AND Password = @PasswordHash AND Role = 'Customer'

    IF @@ROWCOUNT = 0
        SET @UserID = -1
END;
```

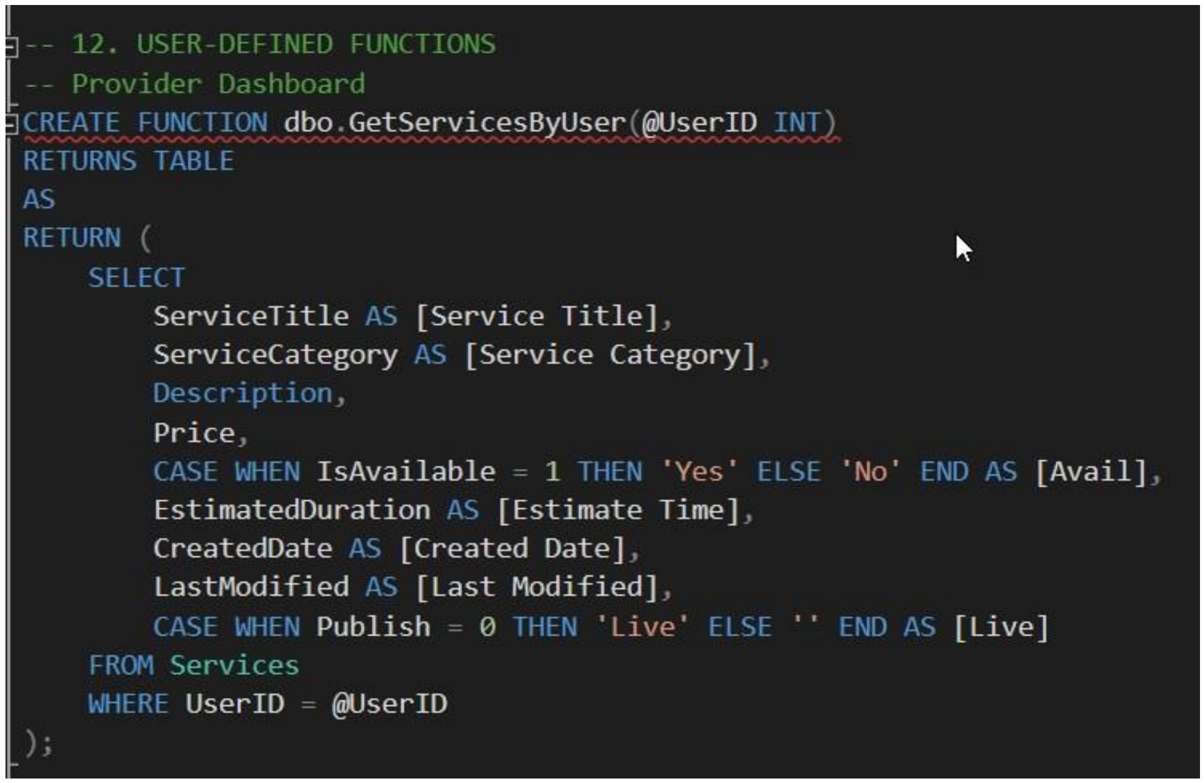
○

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

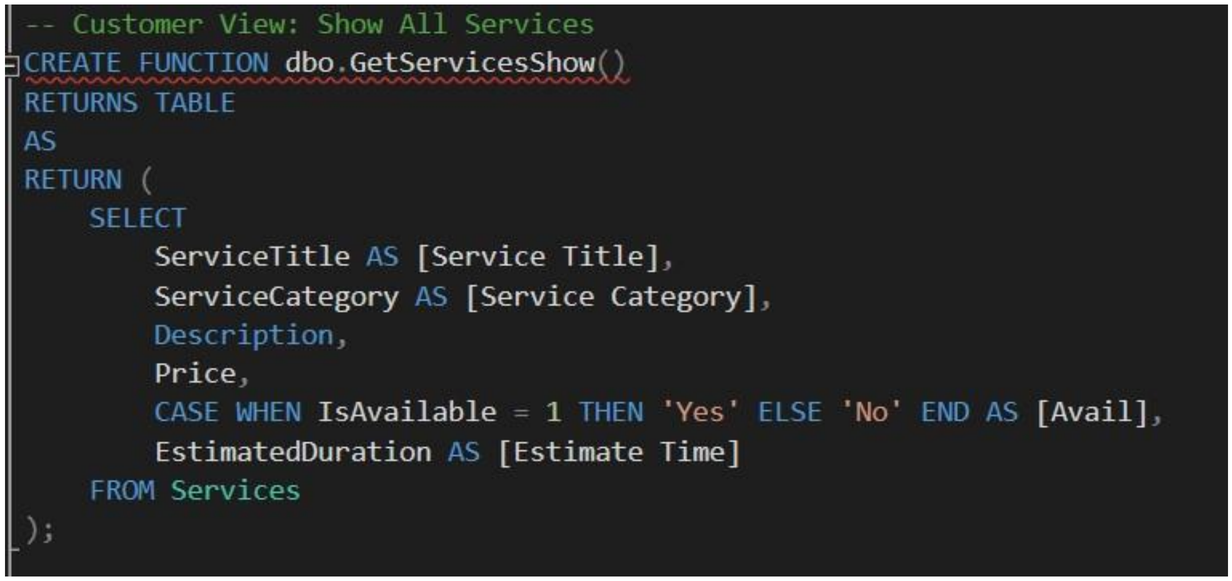
○ GetServicesByUser:{ For provider dashboard }



```
-- 12. USER-DEFINED FUNCTIONS
-- Provider Dashboard
CREATE FUNCTION dbo.GetServicesByUser(@UserID INT)
RETURNS TABLE
AS
RETURN (
    SELECT
        ServiceTitle AS [Service Title],
        ServiceCategory AS [Service Category],
        Description,
        Price,
        CASE WHEN IsAvailable = 1 THEN 'Yes' ELSE 'No' END AS [Avail],
        EstimatedDuration AS [Estimate Time],
        CreatedDate AS [Created Date],
        LastModified AS [Last Modified],
        CASE WHEN Publish = 0 THEN 'Live' ELSE '' END AS [Live]
    FROM Services
    WHERE UserID = @UserID
);
```

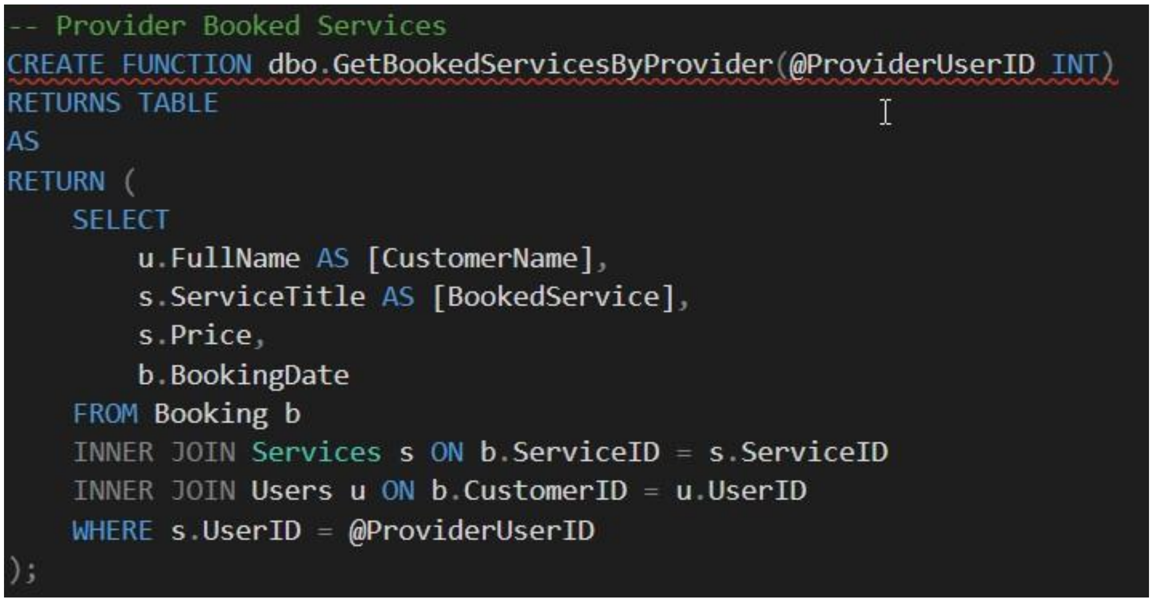
○

GetServicesShow {To display all available services}:



```sql
-- Customer View: Show All Services
CREATE FUNCTION dbo.GetServicesShow()
RETURNS TABLE
AS
RETURN (
    SELECT
        ServiceTitle AS [Service Title],
        ServiceCategory AS [Service Category],
        Description,
        Price,
        CASE WHEN IsAvailable = 1 THEN 'Yes' ELSE 'No' END AS [Avail],
        EstimatedDuration AS [Estimate Time]
    FROM Services
);
```

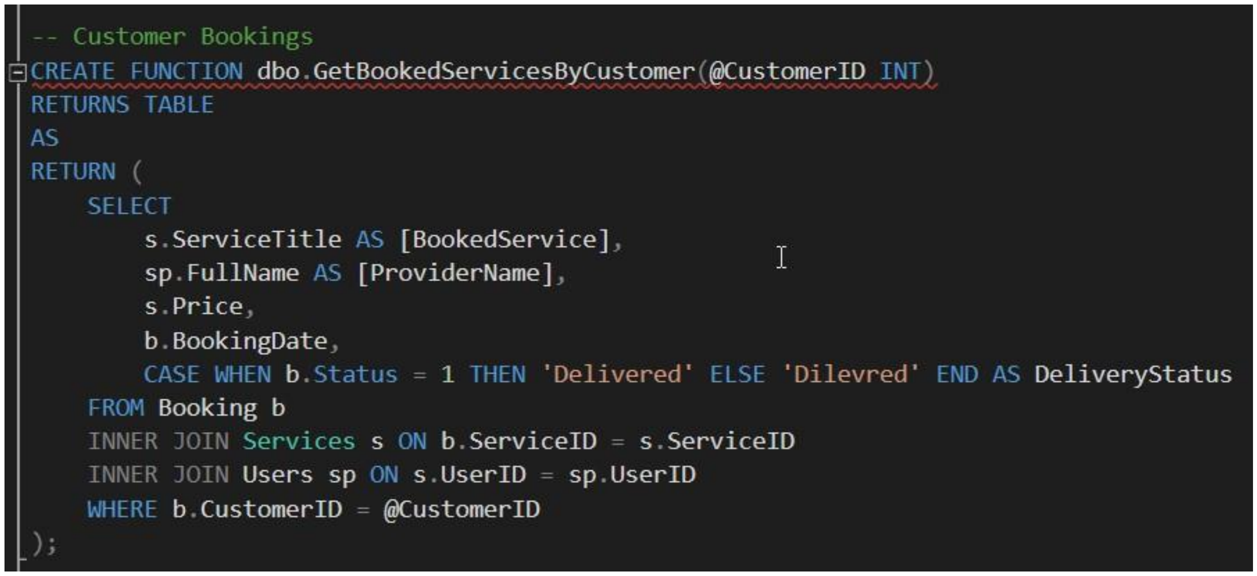
○ **GetBookedServicesByProvider{ Show all bookings under a provider }:**



```sql
-- Provider Booked Services
CREATE FUNCTION dbo.GetBookedServicesByProvider(@ProviderUserID INT)
RETURNS TABLE
AS
RETURN (
    SELECT
        u.FullName AS [CustomerName],
        s.ServiceTitle AS [BookedService],
        s.Price,
        b.BookingDate
    FROM Booking b
    INNER JOIN Services s ON b.ServiceID = s.ServiceID
    INNER JOIN Users u ON b.CustomerID = u.UserID
    WHERE s.UserID = @ProviderUserID
);
```

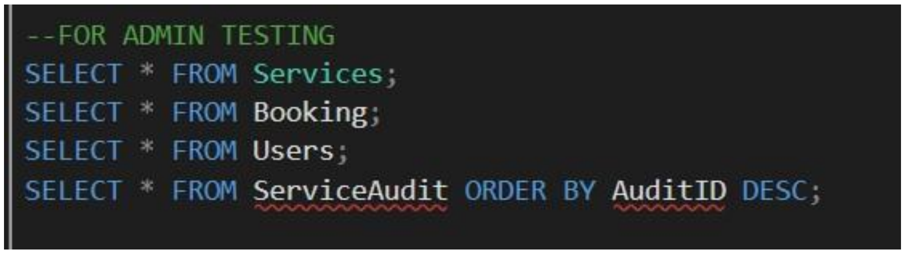
○

GetBookedServicesByCustomer: Show bookings for a specific customer



```
-- Customer Bookings
CREATE FUNCTION dbo.GetBookedServicesByCustomer(@CustomerID INT)
RETURNS TABLE
AS
RETURN (
    SELECT
        s.ServiceTitle AS [BookedService],
        sp.FullName AS [ProviderName],
        s.Price,
        b.BookingDate,
        CASE WHEN b.Status = 1 THEN 'Delivered' ELSE 'Dilevred' END AS DeliveryStatus
    FROM Booking b
    INNER JOIN Services s ON b.ServiceID = s.ServiceID
    INNER JOIN Users sp ON s.UserID = sp.UserID
    WHERE b.CustomerID = @CustomerID
);
```

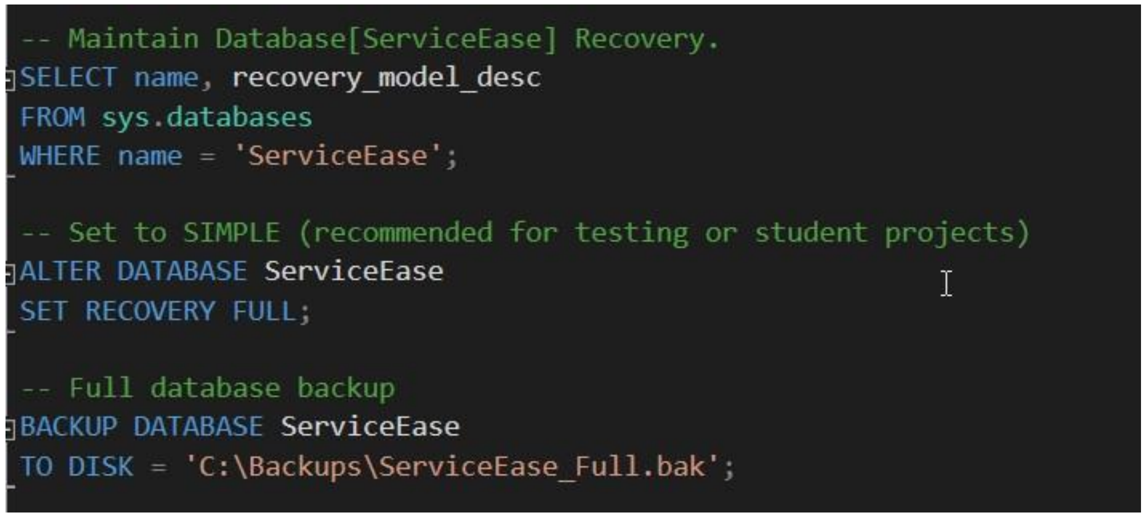
○ For Admin Testing command:



```
--FOR ADMIN TESTING
SELECT * FROM Services;
SELECT * FROM Booking;
SELECT * FROM Users;
SELECT * FROM ServiceAudit ORDER BY AuditID DESC;
```

○ Backup & Recovery Strategy

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

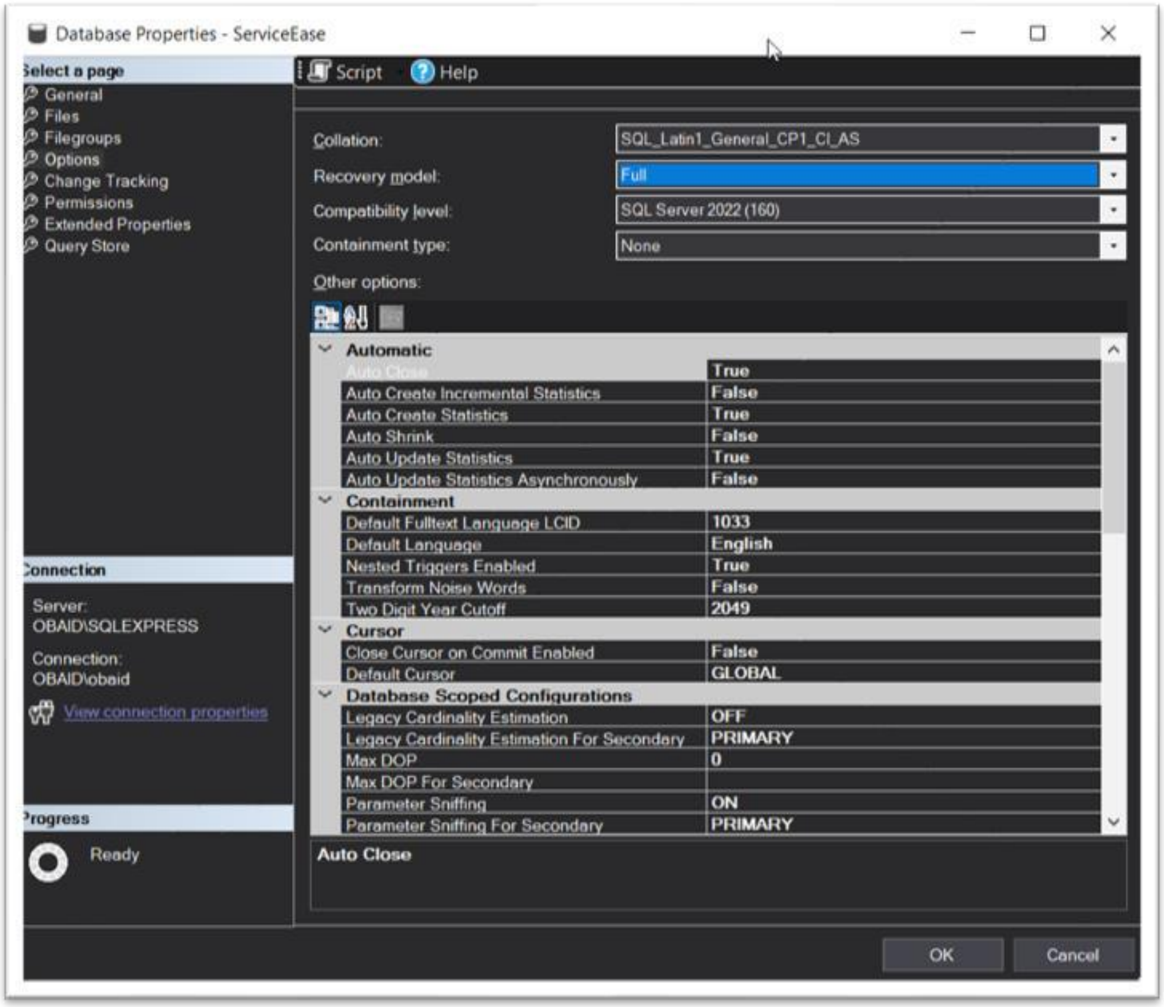


```
-- Maintain Database[ServiceEase] Recovery.
SELECT name, recovery_model_desc
FROM sys.databases
WHERE name = 'ServiceEase';

-- Set to SIMPLE (recommended for testing or student projects)
ALTER DATABASE ServiceEase
SET RECOVERY FULL;

-- Full database backup
BACKUP DATABASE ServiceEase
TO DISK = 'C:\Backups\ServiceEase_Full.bak';
```

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:
    - Simple (for basic recovery without log backups), o or Full (for

   point-in-time recovery with transaction log backups).



Server: OBAID\SQLEXPRESS

Connection: OBAID\obaid

Collation: SQL_Latin1_General_CP1_CI_AS

Recovery model: Full

Compatibility level: SQL Server 2022 (160)

Containment type: None

Other options:

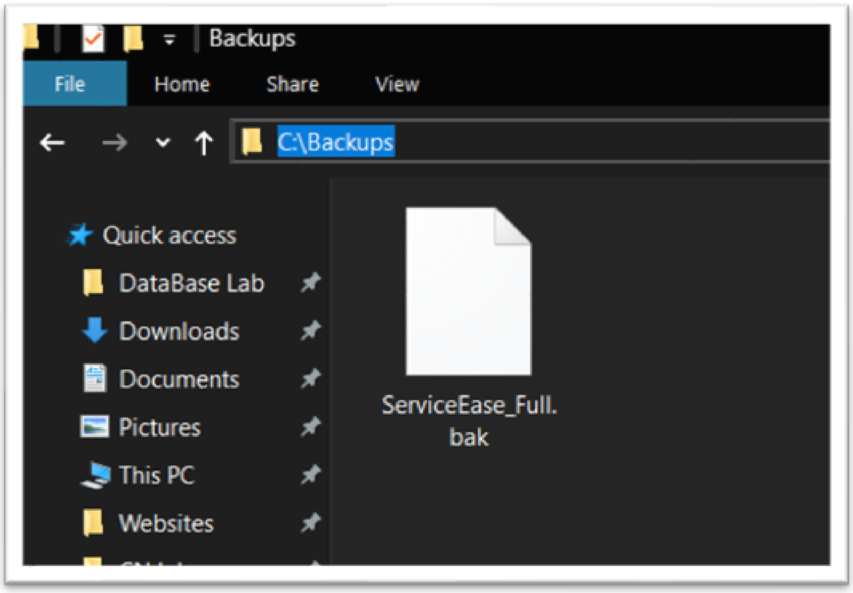
Automatic	
Auto Close	True
Auto Create Incremental Statistics	False
Auto Create Statistics	True
Auto Shrink	False
Auto Update Statistics	True
Auto Update Statistics Asynchronously	False
Containment	
Default Fulltext Language LCID	1033
Default Language	English
Nested Triggers Enabled	True
Transform Noise Words	False
Two Digit Year Cutoff	2049
Cursor	
Close Cursor on Commit Enabled	False
Default Cursor	GLOBAL
Database Scoped Configurations	
Legacy Cardinality Estimation	OFF
Legacy Cardinality Estimation For Secondary	PRIMARY
Max DOP	0
Max DOP For Secondary	
Parameter Sniffing	ON
Parameter Sniffing For Secondary	PRIMARY

Auto Close

o

Backup creation:

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



o

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:
  - 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:

- **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.

2. 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).

3. Restore Operations

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

- **Database Restore**
  Recover the entire database using a full backup.

- **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.

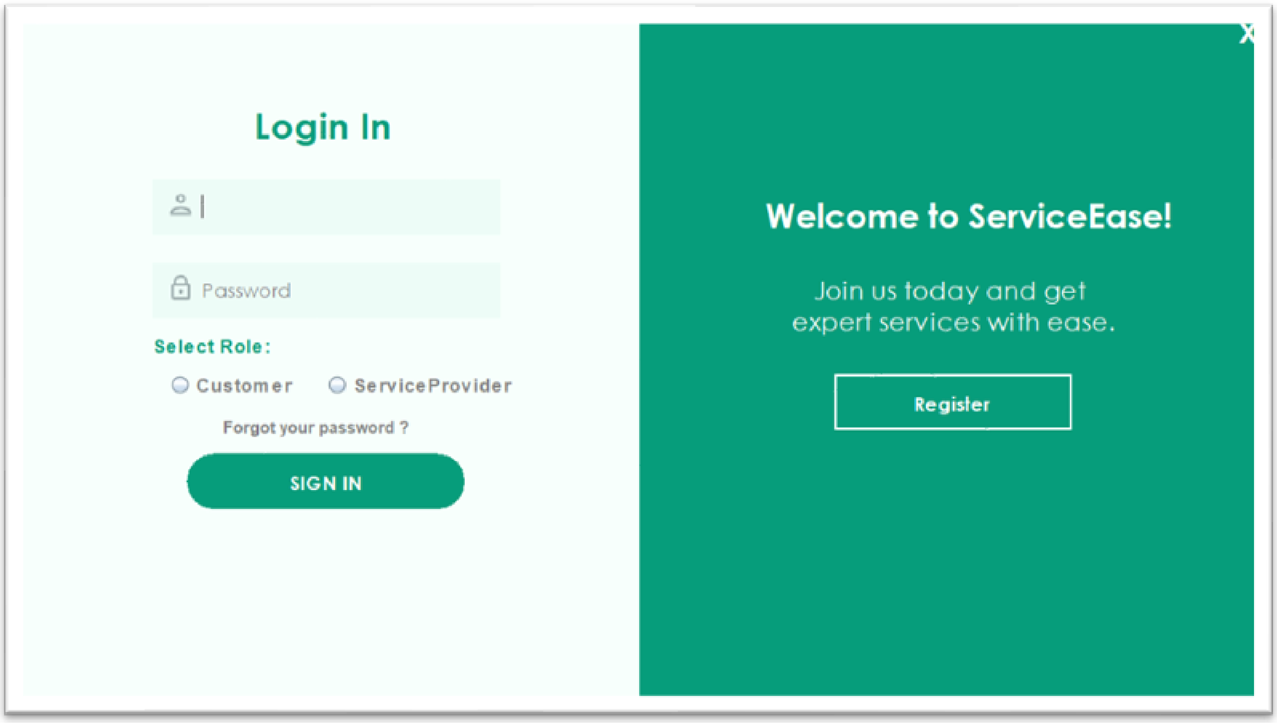
4. Best Practices for Recovery

- 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:

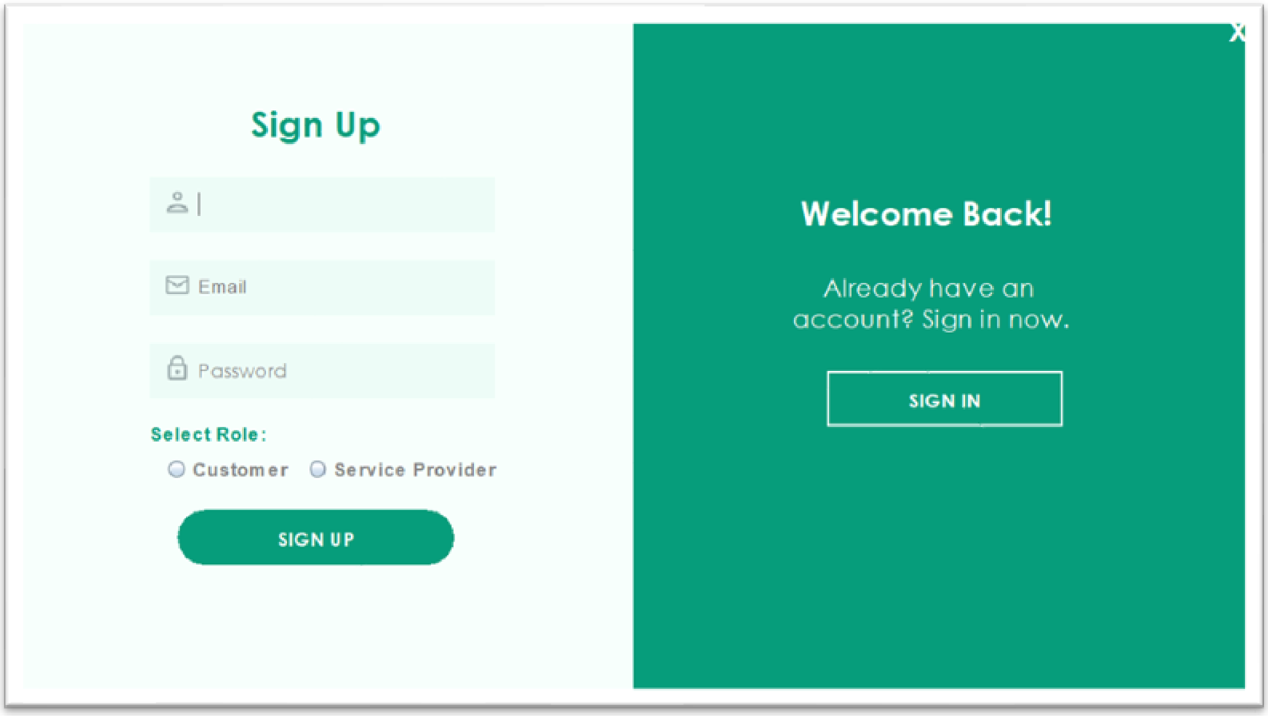
Login Interface:

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.



○ Signup:

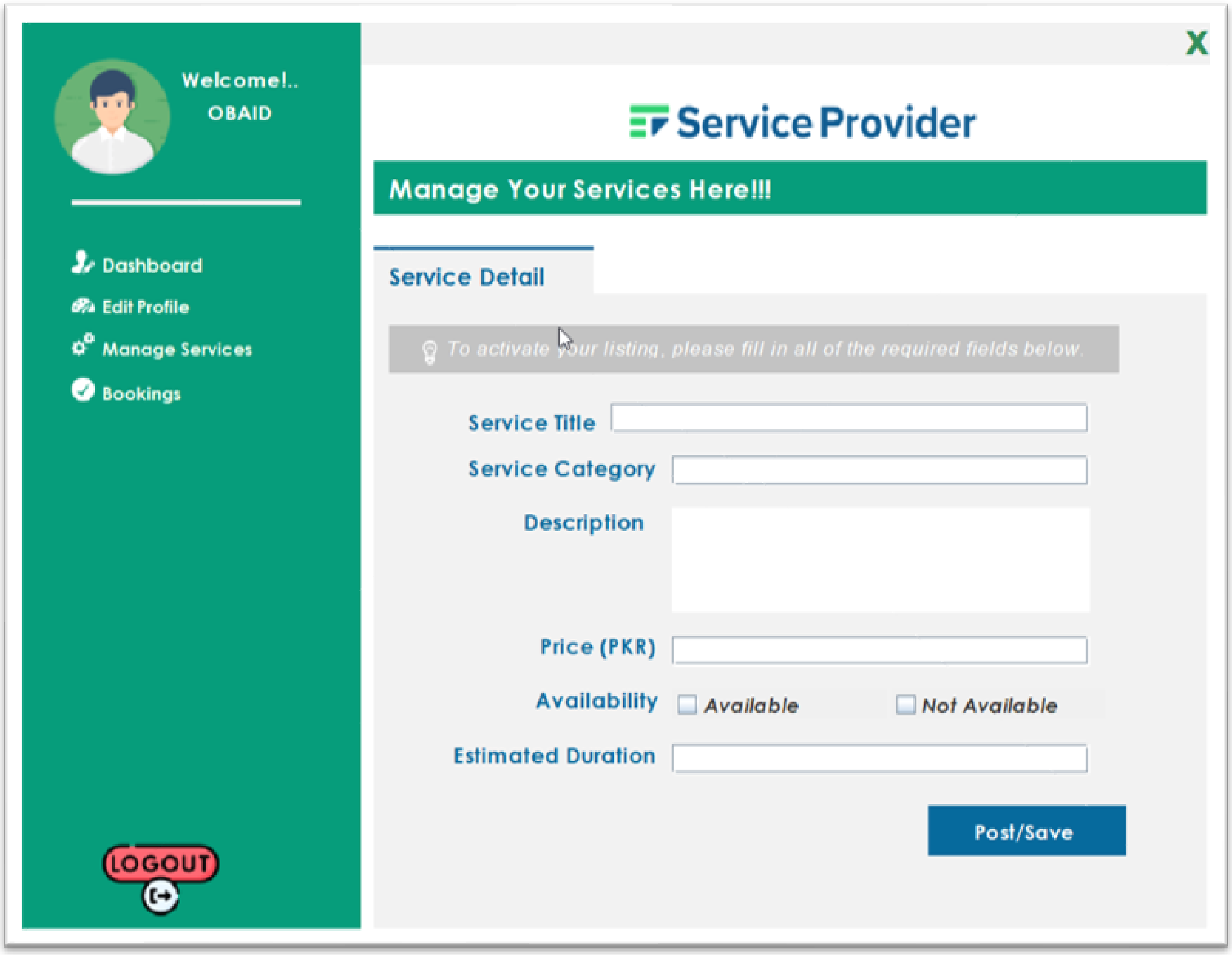
On the sign-up page, the user will enter their name, email, and password, select their role (Customer or Service Provider), and upon clicking the Sign-Up button, the entered data will be stored in SQL Server Management Studio (SSMS) using a user defined function.



○

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."

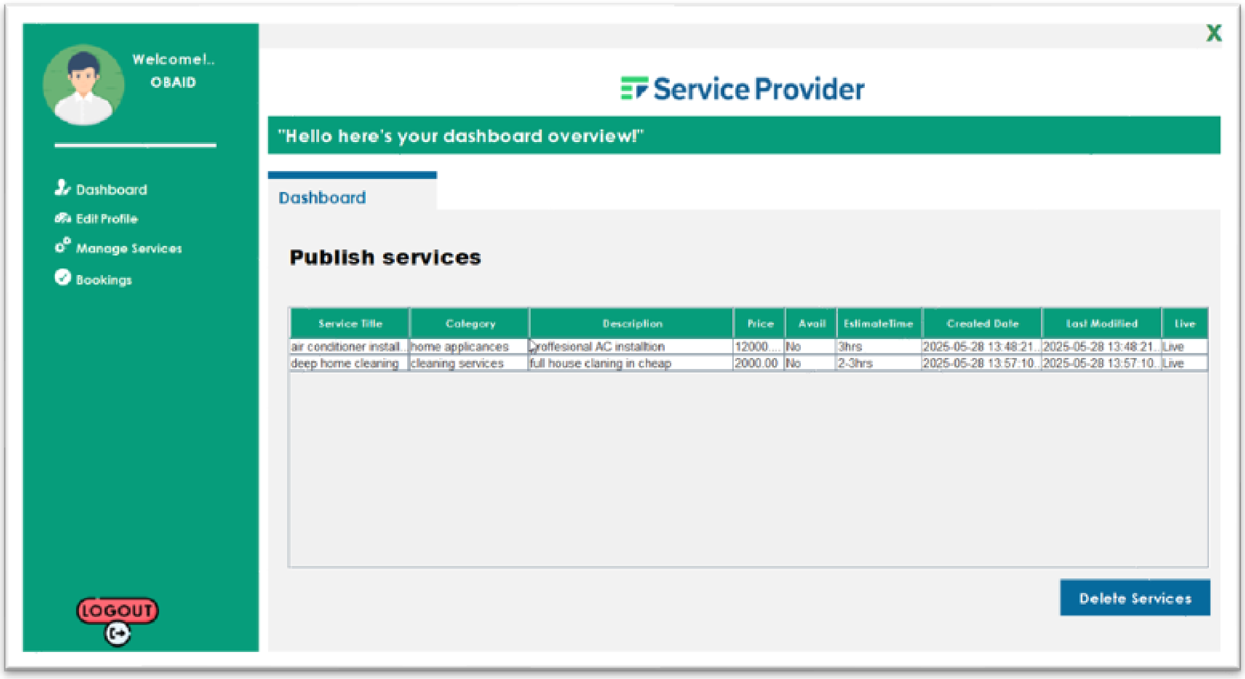


To activate your listing, please fill in all of the required fields below.

Service Title	
Service Category	
Description	
Price (PKR)	
Availability	☐ Available ☐ Not Available
Estimated Duration	

○ 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."



"Hello here's your dashboard overview!"

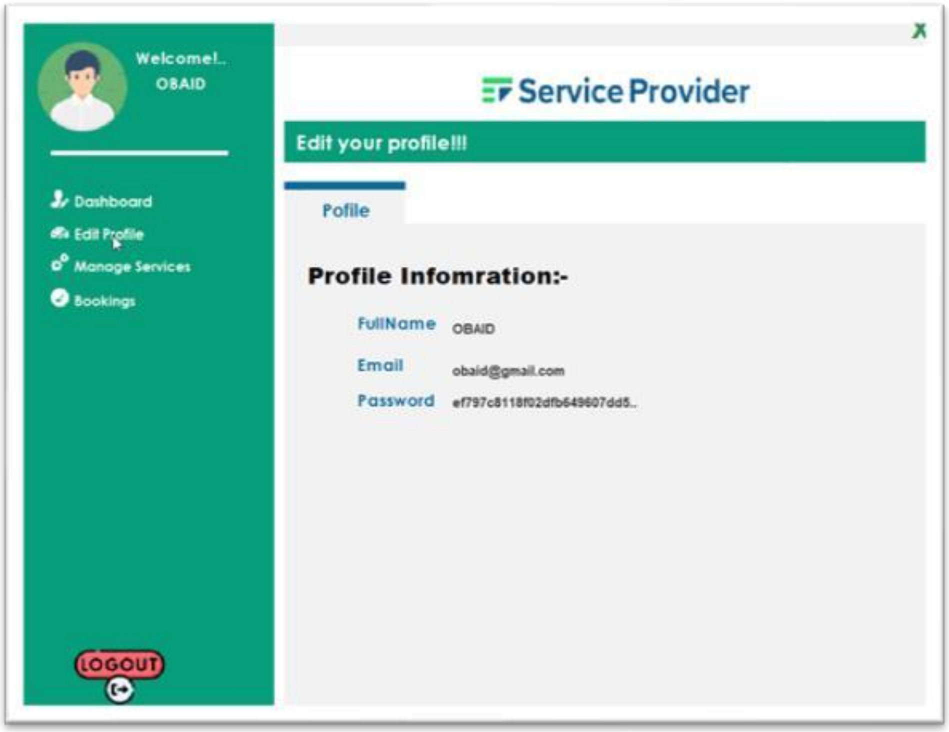
Dashboard

Publish services

Service Title	Category	Description	Price	Avail	EstimateTime	Created Date	Last Modified	Live
air conditioner install...	home applicances	proffesional AC installtion	12000....	No	3hrs	2025-05-28 13:48:21...	2025-05-28 13:48:21...	Live
deep home cleaning	cleaning services	full house claning in cheap	2000.00	No	2-3hrs	2025-05-28 13:57:10...	2025-05-28 13:57:10...	Live

○ 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."



Edit your profile!!!

Pofile

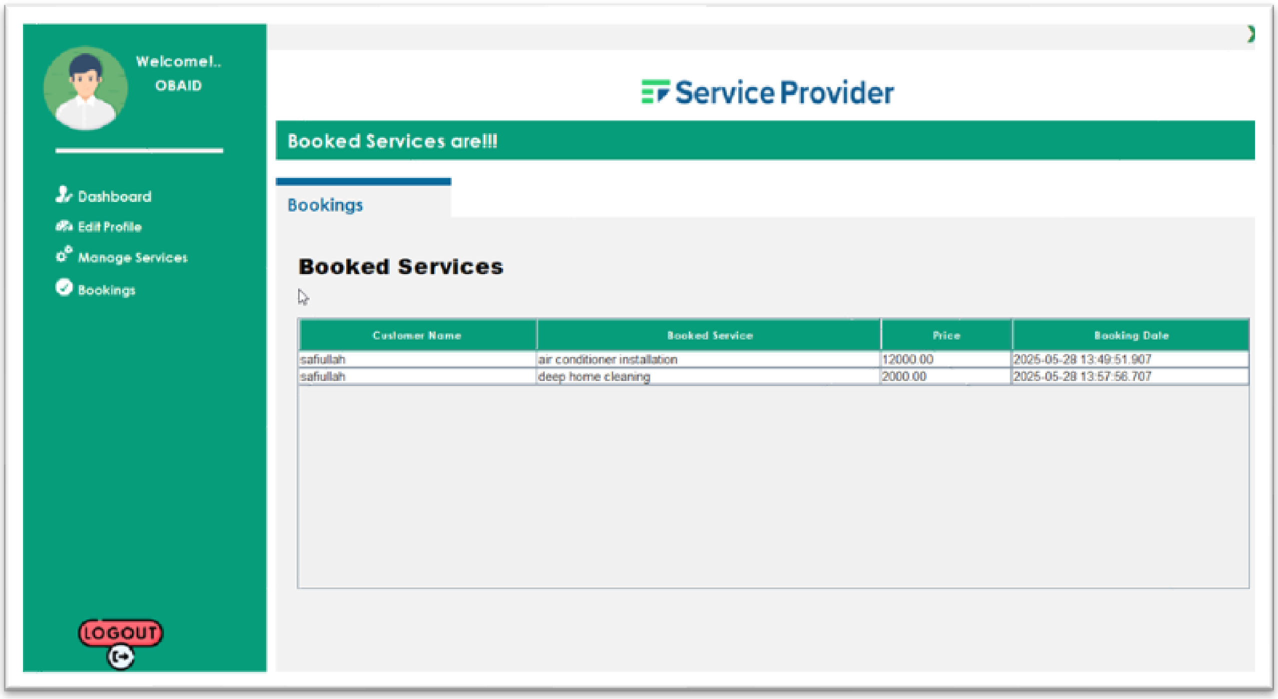
Profile Infomration:-

FullName	OBAID
Email	obaid@gmail.com
Password	ef797c8118f02dfb649607dd5..

o

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."



Booked Services

Customer Name	Booked Service	Price	Booking Date
safiullah	air conditioner installation	12000.00	2025-05-28 13:49:51.907
safiullah	deep home cleaning	2000.00	2025-05-28 13:57:56.707

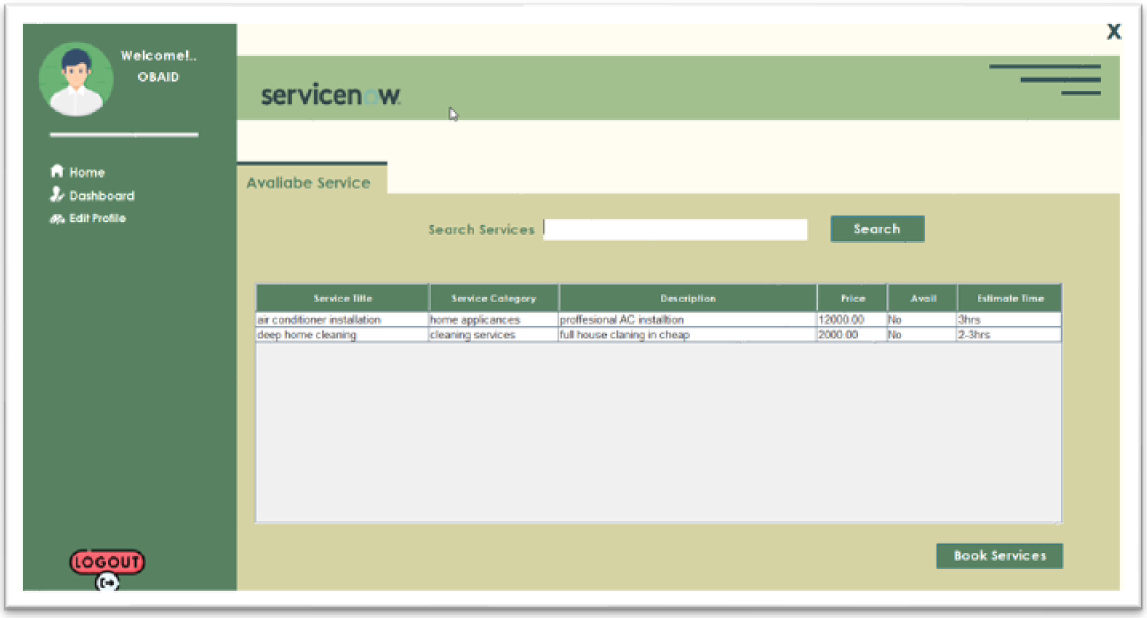
- **Logout:**

By pressing logout they move to login page.

○

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. "



Avaliabe Service

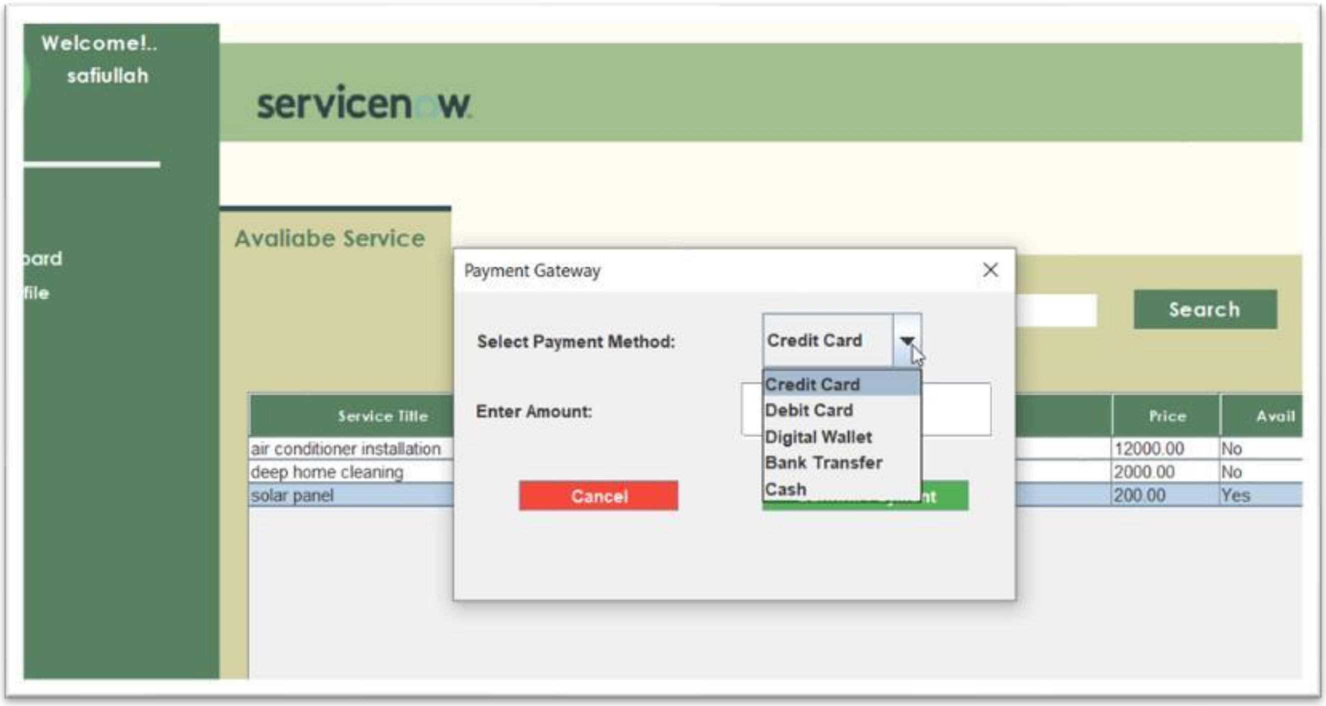
Search Services

Search

Service Title	Service Category	Description	Price	Avail	Estimate Time
air conditioner installation	home applicances	proffesional AC installtion	12000.00	No	3hrs
deep home cleaning	cleaning services	full house claning in cheap	2000.00	No	2-3hrs

Book Services

○ Payment interface



Avaliabe Service

Payment Gateway

Select Payment Method: Credit Card

Credit Card
Debit Card
Digital Wallet
Bank Transfer
Cash

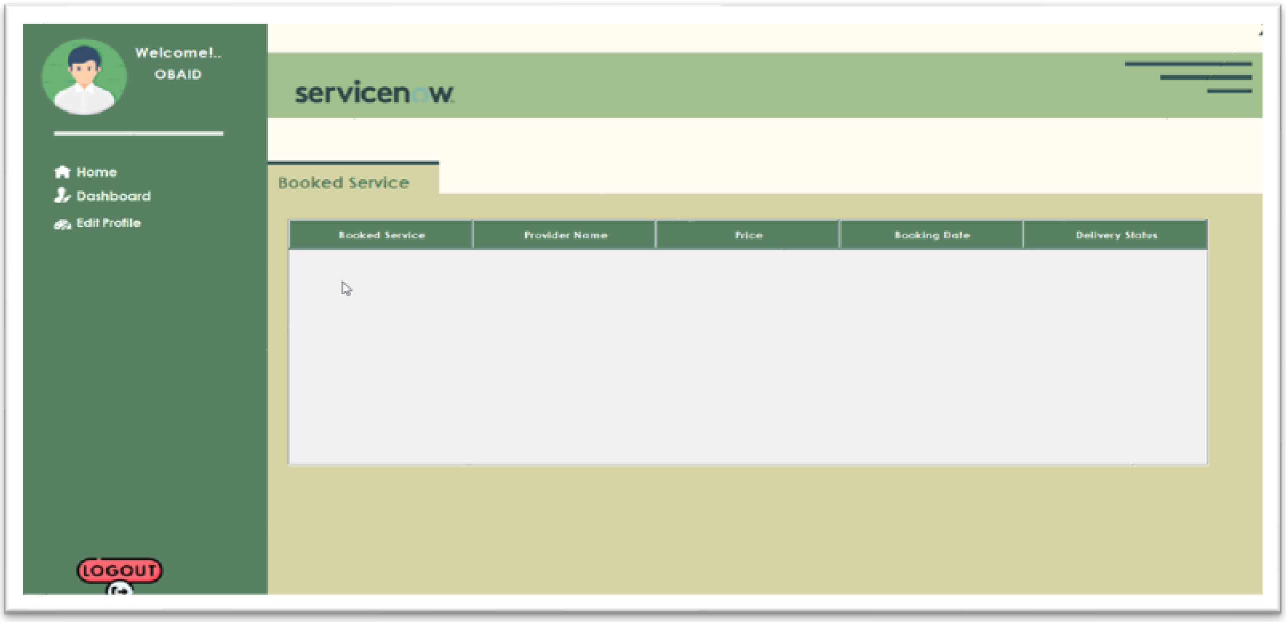
Enter Amount:

Service Title	Price	Avail
air conditioner installation	12000.00	No
deep home cleaning	2000.00	No
solar panel	200.00	Yes

○

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."



Booked Service	Provider Name	Price	Booking Date	Delivery Status

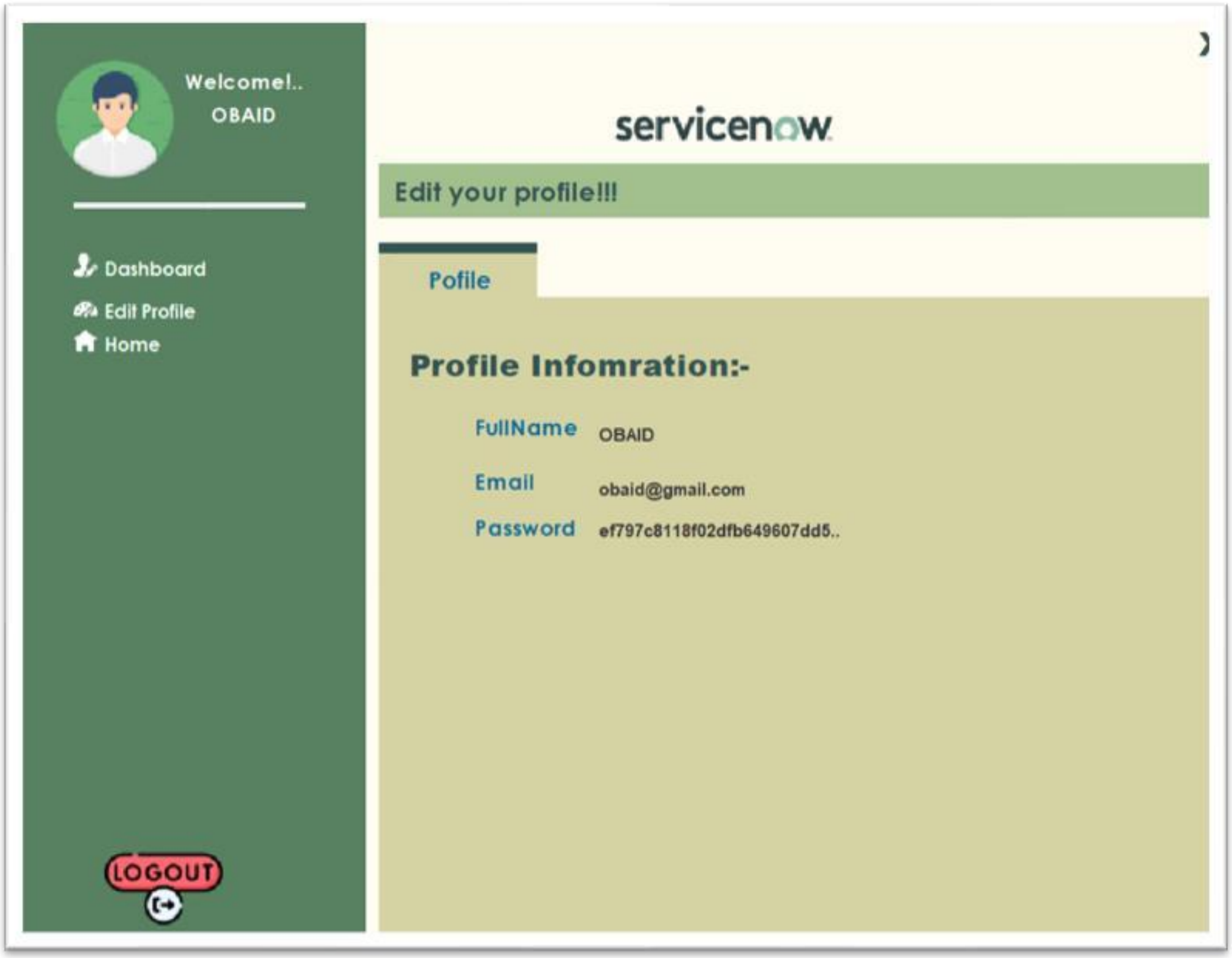
○ Logout:

By clicking logout move to login interface.

o

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."



Profile Infomration:-

FullName	OBAID
Email	obaid@gmail.com
Password	ef797c8118f02dfb649607dd5..