**Software Document**

**User Stories:**

| **Title** | As a Customer, I want deposit cash with receipt generation |
| --- | --- |
| Example: | As a bank customer, I want to deposit cash with receipt generation so that I have a record of my transaction. |
| Acceptance Criteria: | 1. Long as I'm logged into my account when I go to the deposit page I should be able to choose the account and input the amount to deposit.  2. Once I input the deposit amount and click on the "Deposit" button the system should process the transaction. Update my account balance.  3. Following a deposit the system should provide a receipt containing details such, as transaction ID, date and time account information, deposited amount and current balance.  4. The receipt should be downloadable in PDF format for reference.  5. In case of a failed deposit, for any reason (funds) the system should show an error message and ask me to retry or cancel the transaction. |
| Priority: | High |
| Estimate: | 3 story points |
| Notes: | This user story covers the basic functionality of depositing cash with receipt generation, which is essential for banking operations. The acceptance criteria outline the specific requirements and behaviors expected from the system to fulfill the user's needs effectively. |

**Project Planning:**

| **Project Name:** | **Banking System Development** |
| --- | --- |
| **Project Vision:** | Creating an easy-to-use banking system that offers clients effortless entry to services while maintaining security, dependability, and adherence, to industry standards. |
| **Project Objectives:** | 1. Create a system that can manage different banking functions like handling accounts processing financial transactions and providing account services.  2. Develop a user interface that lets customers smoothly navigate and carry out banking tasks on any device.  3. Put in place security protocols to safeguard customer information and deter access or fraudulent behaviour.  4. Adhere to applicable regulations and data privacy laws to uphold trust and reputation, with customers as well as regulatory bodies. |
| **Project Scope:** | -Managing Accounts; Setting up modifying and closing accounts.  -Handling Finances; Depositing, withdrawing, and moving money.  -Account Features; Checking account details, overseeing services establishing payments.  -Safety and Regulations; Putting in place verification methods encrypting data and following regulatory guidelines. |
| **Project Deliverables:** | 1. Set up a system that includes a database structure and APIs, for managing accounts and handling transactions.  2. Created design mockups and prototypes for the user interface.  3. Our authentication system is secure with two-factor authentication, in place.  4. Prepared compliance documentation that details requirements and the steps we've taken to comply with them.  5. Test plans and documentation to ensure quality assurance. |
| **Project Constraints:** | - Budget: $25000.  - Timeline: 16 weeks/ 4 months.  - Resource Availability: X developers, X testers, X designers.  - Technology Stack: ASP.NET, C#, SQL Server, Web technology (JavaScript, HTML,CSS3). etc. |
| **Project Stakeholders:** | - Product Owner: Hamdi  - Scrum Master: Hamdi  - Development Team: Hamdi  - Customers: [End-users of the banking system]  - Regulatory Authorities: [Relevant regulatory bodies] |
| **Project Risks:** | 1. Data Security Breaches: Risk of unauthorized access to customer data  2. Compliance Risks: Failure to comply with regulatory requirements.  3. Technology Risks: Issues with technology stack or third-party dependencies.  4. Scope Creep: Expansion of project scope beyond initial requirements. |
| **Success Criteria:** | 1. Completion of project deliverables within the specified timeline and budget  2. Positive feedback from end-users regarding usability and functionality.  3. Compliance with regulatory requirements and industry standards.  4. Minimal bugs and issues reported during testing and production. |
| **Project Milestones:** | - Sprint 1: Backend Development  - Sprint 2: User Interface Design  -Sprint 3: Authentication Implementation  - Sprint 4: Compliance Documentation  - Sprint 5: Testing and Quality Assurance  - Sprint 6: Deployment and Launch |

Objects Layer code

// Base class for Bank entity

public class Bank

{

public int BankID { get; set; }

public string BankName { get; set; }

public string Address { get; set; }

public string PhoneNumber { get; set; }

}

// Base class for Customer entity

public class Customer

{

public int CustomerID { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Address { get; set; }

public string Email { get; set; }

public string PhoneNumber { get; set; }

public string UserName { get; set; }

public string Password { get; set; }

}

// Base class for Account entity

public class Account

{

public Account()

{

//Initialize the List in the constructor

AccountHoldersList = new List<Customer>();

}

public int AccountID { get; set; }

public Customer AccountHolder { get; set; }

public List<Customer> AccountHoldersList { get; set; }//List of AccountHolders

public Bank Bank { get; set; }

public string AccountType { get; set; }

public decimal Balance { get; set; }

public Account(decimal initialBalance)

{

Balance = initialBalance;

}

public void Deposit(decimal amount)

{

if (amount <= 0)

{

throw new ArgumentException("Deposit amount must be positive.");

}

Balance += amount;

}

public void Withdraw(decimal amount)

{

if (amount <= 0)

{

throw new ArgumentException("Withdrawal amount must be positive.");

}

if (Balance < amount)

{

throw new InvalidOperationException("Insufficient funds.");

}

Balance -= amount;

}

}

// Derived class for Savings Account

public class SavingsAccount : Account

{

public decimal InterestRate { get; set; }

// Method to calculate interest

public decimal CalculateInterest()

{

// Implement interest calculation logic

return Balance \* InterestRate;

}

}

// Derived class for Checking Account

public class CheckingAccount : Account

{

public decimal OverdraftLimit { get; set; }

// Method to perform overdraft check

public bool IsOverdraft(decimal amount)

{

return (Balance - amount) < (-1 \* OverdraftLimit);

}

}

Data Layer code in code project its Name

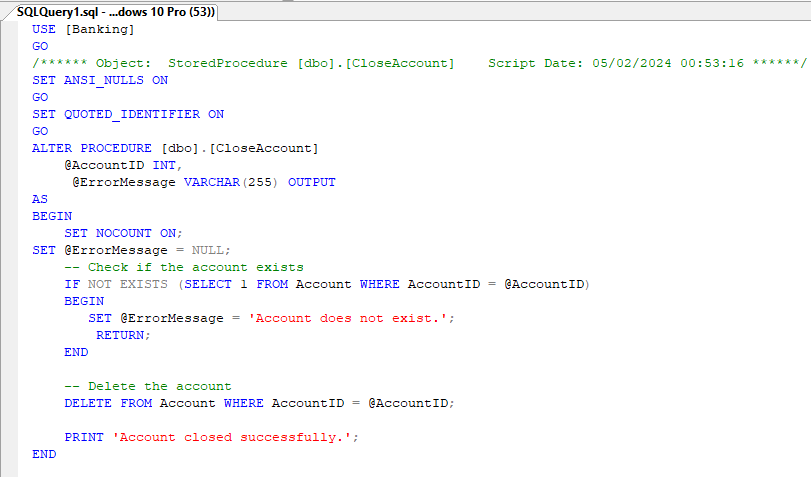
class DatabaseManager

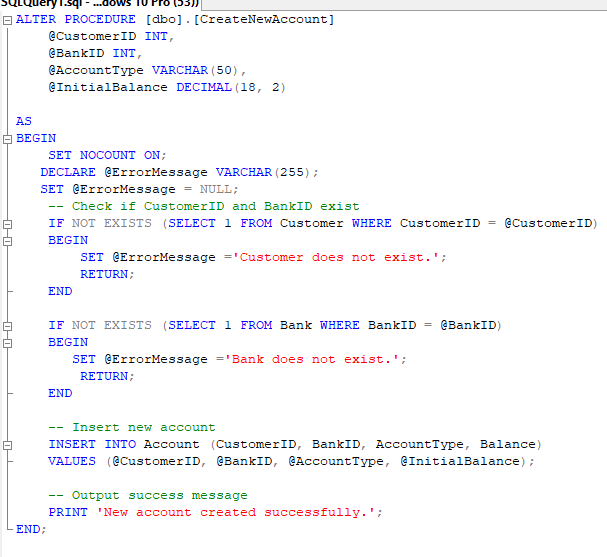
There are ASPX Pages and their code behind \*.CS code for each page in system you can see them in code project uploaded to GetHub.

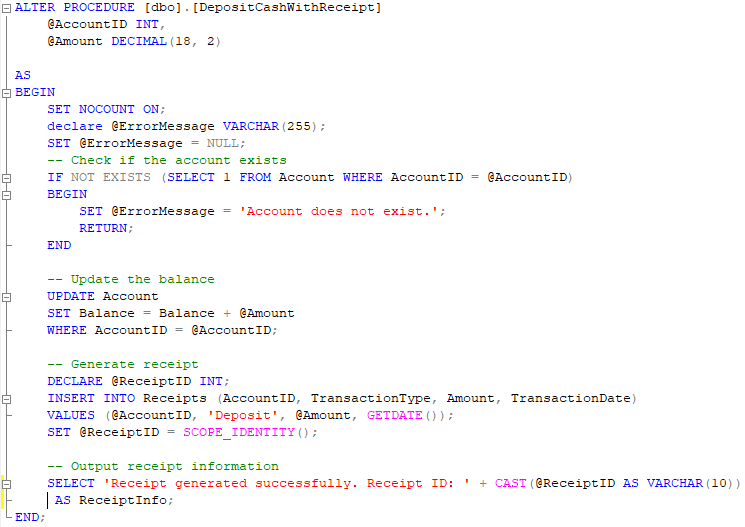
**Banking SQL Data Base**

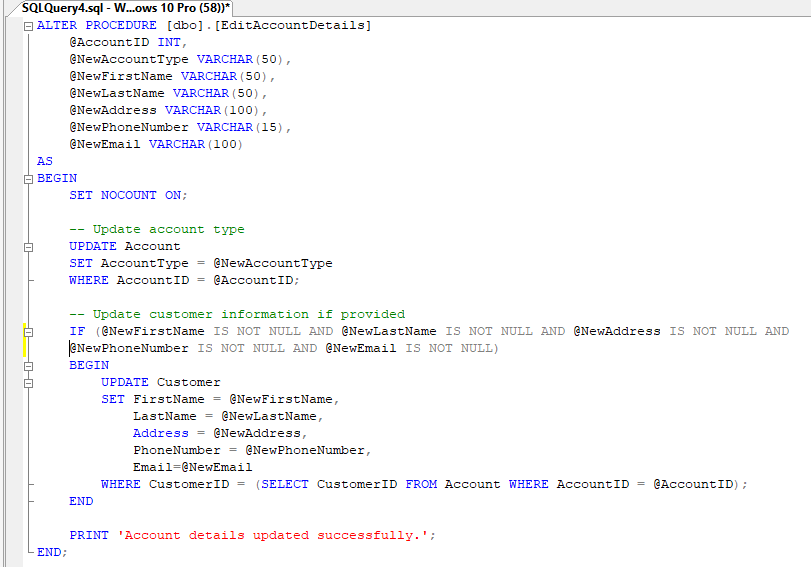
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

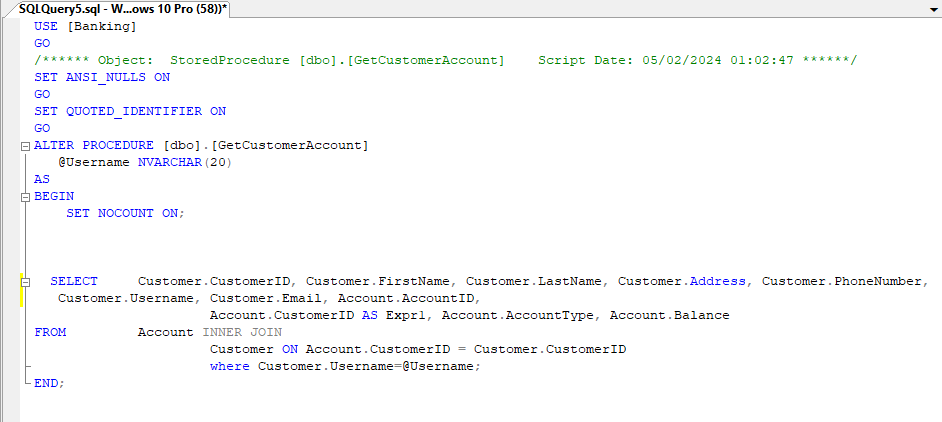
**Banking SQL Stored procedures:**

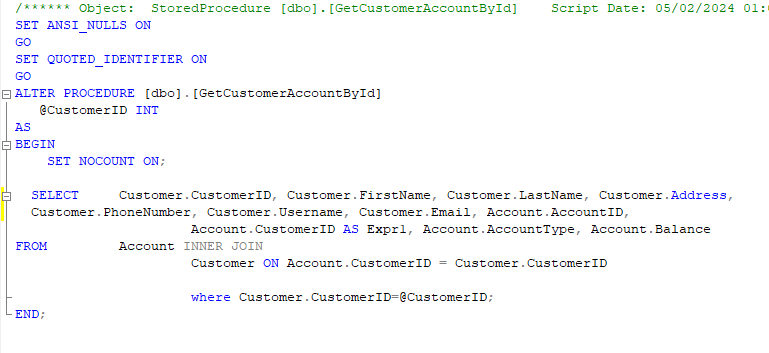


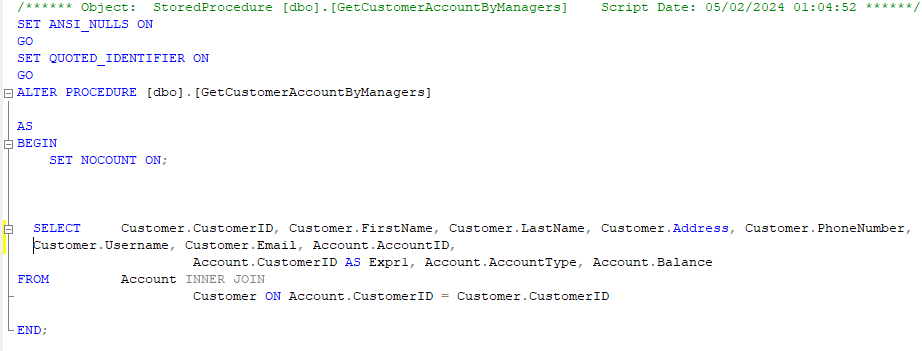


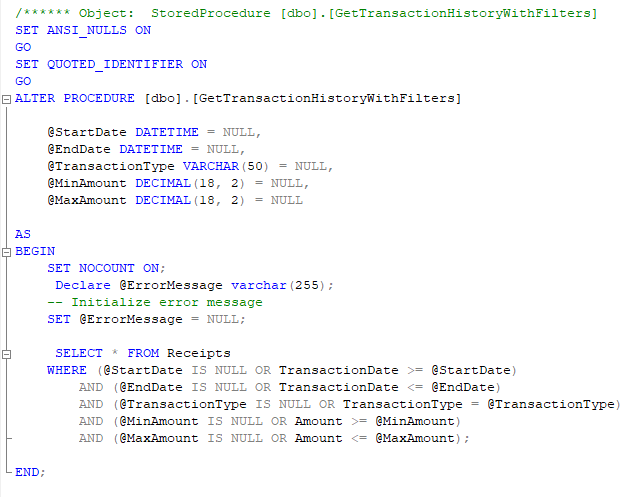


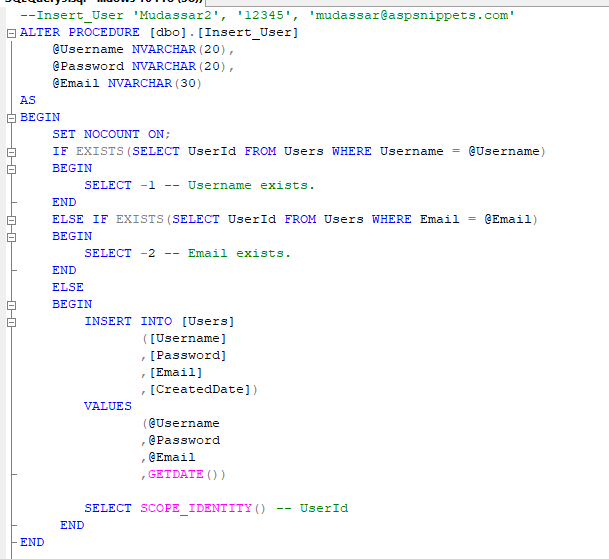


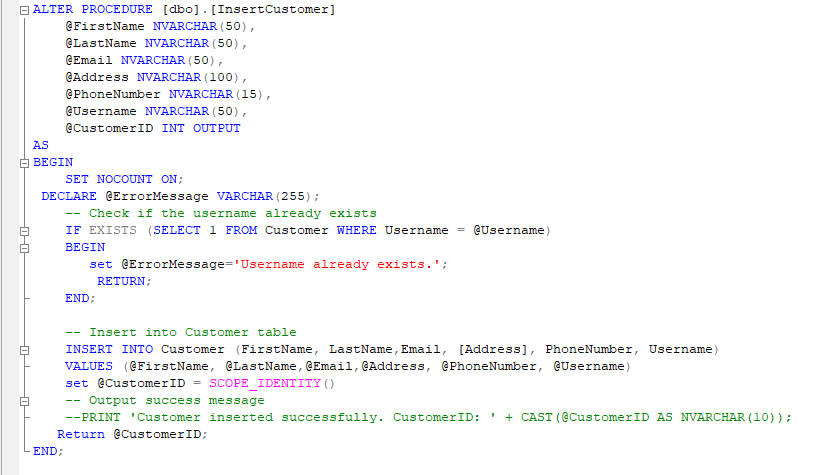












ALTER PROCEDURE [dbo].[TransferFundsWithinSameBank]

@SourceAccountID INT,

@DestinationAccountID INT,

@Amount DECIMAL(18, 2),

@ErrorMessage VARCHAR(255) OUTPUT

AS

BEGIN

SET NOCOUNT ON;

-- Initialize error message

SET @ErrorMessage = NULL;

--Check if source and destination accounts exist

IF NOT EXISTS (SELECT 1 FROM Account WHERE AccountID = @SourceAccountID)

BEGIN

SET @ErrorMessage = 'Source account does not exist.';

RETURN;

END;

IF NOT EXISTS (SELECT 1 FROM Account WHERE AccountID = @DestinationAccountID)

BEGIN

SET @ErrorMessage = 'Destination account does not exist.';

RETURN;

END;

-- Check if source account has sufficient balance

DECLARE @SourceBalance DECIMAL(18, 2);

SELECT @SourceBalance = Balance FROM Account WHERE AccountID = @SourceAccountID;

IF @SourceBalance < @Amount

BEGIN

SET @ErrorMessage = 'Insufficient balance in the source account.';

RETURN;

END;

-- Update source and destination account balances

BEGIN TRY

BEGIN TRANSACTION;

UPDATE Account

SET Balance = Balance - @Amount

WHERE AccountID = @SourceAccountID;

UPDATE Account

SET Balance = Balance + @Amount

WHERE AccountID = @DestinationAccountID;

COMMIT TRANSACTION;

SET @ErrorMessage = 'Funds transferred successfully.';

RETURN;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

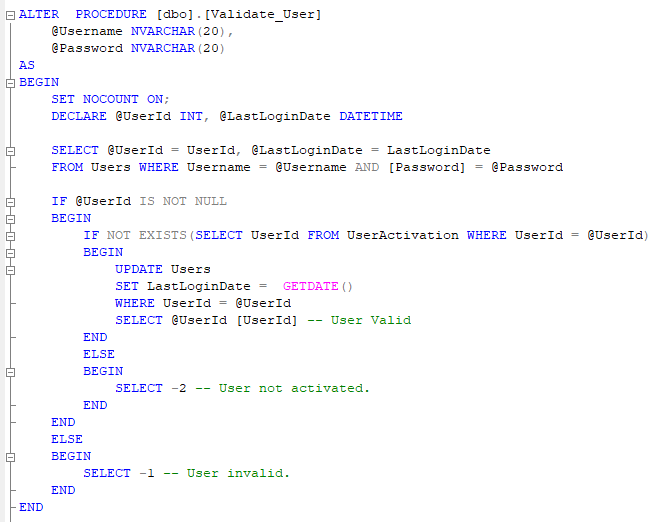
ROLLBACK TRANSACTION;

SET @ErrorMessage = 'An error occurred while transferring funds.';

RETURN;

END CATCH;

END;



ALTER PROCEDURE [dbo].[WithdrawCashWithRealTimeUpdate]

@AccountID INT,

@Amount DECIMAL(18, 2),

@ErrorMessage VARCHAR(255)OUTPUT

AS

BEGIN

SET NOCOUNT ON;

SET @ErrorMessage = NULL;

-- Check if the account exists

IF NOT EXISTS (SELECT 1 FROM Account WHERE AccountID = @AccountID)

BEGIN

SET @ErrorMessage ='Account does not exist.';

RETURN;

END;

-- Check if sufficient balance is available

DECLARE @CurrentBalance DECIMAL(18, 2);

SELECT @CurrentBalance = Balance FROM Account WHERE AccountID = @AccountID;

IF @CurrentBalance < @Amount

BEGIN

SET @ErrorMessage ='Insufficient balance.';

RETURN;

END;

-- Update the balance

UPDATE Account

SET Balance = Balance - @Amount

WHERE AccountID = @AccountID;

-- Generate receipt

DECLARE @ReceiptID INT;

INSERT INTO Receipts (AccountID, TransactionType, Amount, TransactionDate)

VALUES (@AccountID, 'Withdraw', @Amount, GETDATE());

SET @ReceiptID = SCOPE\_IDENTITY();

-- Output success message

PRINT 'Withdrawal successful. Updated balance: ' + CAST((@CurrentBalance - @Amount) AS VARCHAR(20));

END;