**Customer Banking**

**Application &**

**Backend API**

Table of Contents

[**Project Overview:** 4](#_Toc202714700)

[**Objective** 4](#_Toc202714701)

[**Technical Stack** 4](#_Toc202714702)

[**Solution Structure** 4](#_Toc202714703)

[**API Integration** 6](#_Toc202714704)

[**Signup Module** 7](#_Toc202714705)

[**1. Purpose** 7](#_Toc202714706)

[**2. Page Structure & Flow** 7](#_Toc202714707)

[**3. ViewModel: SignupViewModel.cs** 7](#_Toc202714708)

[**4. Request Model: SignupRequest.cs** 8](#_Toc202714709)

[**5. API Endpoint: POST /api/auth/signup** 8](#_Toc202714710)

[**6. Device Lock Logic (One Device per Customer)** 9](#_Toc202714711)

[**7. Response Model: SignupResponse.cs** 9](#_Toc202714712)

[**7. Summary** 10](#_Toc202714713)

[**Login Module** 11](#_Toc202714714)

[**1. Purpose** 11](#_Toc202714715)

[**2. Page Structure & Flow** 11](#_Toc202714716)

[**3. ViewModel: LoginViewModel.cs** 11](#_Toc202714717)

[**4. Request Model: LoginRequest.cs** 12](#_Toc202714718)

[**5. API Endpoint: POST /api/auth/login** 12](#_Toc202714719)

[**6. Device Lock Logic (Reinforced at Login)** 12](#_Toc202714720)

[**7. Response Model: LoginResponse.cs** 12](#_Toc202714721)

[**8. Summary** 13](#_Toc202714722)

[**Forgot Password Module** 14](#_Toc202714723)

[**1. Purpose (Outdated)** 14](#_Toc202714724)

[**2. Page Structure & Flow** 14](#_Toc202714725)

[**3. ViewModel: ForgotPasswordViewModel.cs** 14](#_Toc202714726)

[**4. Request Model (Ad-hoc, Inline)** 15](#_Toc202714727)

[**5. API Endpoints** 15](#_Toc202714728)

[**6. Security Gaps (Requires Refactoring)** 15](#_Toc202714729)

[**7. Response Models** 15](#_Toc202714730)

[**8. Summary** 16](#_Toc202714731)

[**Home Module Documentation** 17](#_Toc202714732)

[**1. Purpose** 17](#_Toc202714733)

[**2. Page Structure** 17](#_Toc202714734)

[**3. ViewModel: HomeViewModel.cs** 17](#_Toc202714735)

[**4. UI Composition** 18](#_Toc202714736)

[**5. API Integration** 18](#_Toc202714737)

[**6. Logout Functionality** 19](#_Toc202714738)

[**7. UX Highlights** 19](#_Toc202714739)

[**8. Summary** 20](#_Toc202714740)

[**Statement Page Module Documentation** 21](#_Toc202714741)

[**1. Purpose** 21](#_Toc202714742)

[**2. Page Structure** 21](#_Toc202714743)

[**3. Flow of Execution** 21](#_Toc202714744)

[**4. ViewModels** 21](#_Toc202714745)

[**5. API Integration** 22](#_Toc202714746)

[**6. PDF Export Logic** 23](#_Toc202714747)

[**7. Navigation Flow** 23](#_Toc202714748)

[**8. Summary** 24](#_Toc202714749)

[**Beneficiary Module Documentation** 25](#_Toc202714750)

[**1. Purpose** 25](#_Toc202714751)

[**2. Page Structure** 25](#_Toc202714752)

[**3. Flow of Execution** 25](#_Toc202714753)

[**4. ViewModels** 25](#_Toc202714754)

[**5. API Integration** 26](#_Toc202714755)

[**6. Module Logic Summary** 27](#_Toc202714756)

[**7. Security Considerations** 27](#_Toc202714757)

[**8. Summary** 27](#_Toc202714758)

[**Payments Module Documentation** 28](#_Toc202714759)

[**1. Purpose** 28](#_Toc202714760)

[**2. Page Flow Overview** 28](#_Toc202714761)

[**3. Page & ViewModel Breakdown** 28](#_Toc202714762)

[**4. Bugs & Technical Issues** 31](#_Toc202714763)

[**5. Summary** 31](#_Toc202714764)

[**6. Suggested Improvements** 31](#_Toc202714765)

[**Supporting Modules (Extended & Final)** 32](#_Toc202714766)

[**Purpose** 32](#_Toc202714767)

[**Final Coverage Includes:** 32](#_Toc202714768)

[**Details** 32](#_Toc202714769)

[**Final Suggestions Summary** 37](#_Toc202714770)

# **Project Overview:**

**Customer Banking Application & Backend API**

**Objective**

The goal of this project is to develop a secure, scalable, and modular **mobile banking application** using .NET MAUI for cross-platform compatibility and ASP.NET Core Web API for backend services. The system allows users to manage accounts, transfer funds, view statements, reset passwords securely, and much more — all while enforcing strong device-level authentication.

This solution ensures a professional-grade architecture to support real-world digital banking requirements for cooperative banks or small-scale financial institutions.

**Technical Stack**

|  |  |  |
| --- | --- | --- |
| Component | Technology | Description |
| Frontend | .NET MAUI | Cross-platform UI framework for Android, iOS, and Windows |
| Backend API | ASP.NET Core Web API (.NET 8) | RESTful API providing secure endpoints for all banking operations |
| Database | SQL Server | Central data storage for customer info, transactions, beneficiaries, etc. |
| Architecture | MVVM (Frontend), Layered (API) | Clear separation of UI, logic, and data access |
| Communication | HTTP REST APIs (JSON) | All API interactions use JSON-based payloads over HTTP |
| Authentication | Device-bound + PIN + OTP | Single-device login model + OTP verification for critical flows |
| Security | SHA-based hashing, OTP service | PINs/passwords are securely hashed, OTP ensures secure verification |
| PDF Export | PDFSharpCore | Statement export functionality on mobile |

**Solution Structure**

**1. Frontend: bank\_demo (MAUI App)**

Structured into logical folders to separate UI, ViewModels, services, and models.

**➤ Pages**

Contains all the XAML-based UI pages, grouped by functionality:

* **AuthenticationPages** – Login, Signup, Forgot Password
* **BeneficiaryPages** – Add, view, and manage beneficiaries
* **Fund\_Transfer** – Transfer money, enter amount, view receipt
* **StatementPages** – Account ledger, statement view
* **HomePages** – Navigation drawer, Home, About, Settings
* **Others** – QR code, support, contact, transaction history, profile, etc.

**➤ ViewModels**

All business logic and page state management is implemented here using the **MVVM pattern**:

* SignupViewModel, LoginViewModel, EnterAmountViewModel, StatementViewModel, etc.

**➤ Services**

Core backend communication and utilities:

* ApiService.cs – Unified API calling class using HttpClient
* OtpService.cs – Handles OTP generation and verification
* DBHelper.cs – Local SQL operations (if needed)
* SecurityHelper.cs – Password/PIN hashing utilities
* StatementPdfExporter.cs – Handles PDF generation of statements

**➤ Models (Services/API)**

Classes used for sending and receiving data via the API:

* LoginRequest, SignupRequest, StatementResponse, etc.

**2. Backend: BankingAPI (ASP.NET Web API)**

**➤ Controllers**

RESTful endpoints grouped logically:

* AuthController – Login, Signup, Forgot Password, Device Check
* BeneficiariesController – Add, view, delete beneficiaries
* TransactionController – Handle money transfers
* StatementController, CustomerLedgerController – Account ledger and statement endpoints

**➤ Models**

Simple POCOs (Plain Old C# Objects) for request/response:

* SignupRequest, LoginResponse, TransactionRequest, etc.

**➤ Utility & Helpers**

* SecurityHelper.cs – Hashing and verification
* OTP, password logic implemented on both frontend and backend sides.

**API Integration**

The mobile app communicates with the API through a service abstraction layer:

* **Service Used**: ApiService.cs
* **Communication Protocol**: HTTP over JSON
* **All API URLs**: Managed through BaseURL.cs
* **Security**:
  + All PINs/passwords are hashed before saving
  + Device-bound login prevents multi-device usage
  + OTP-based authentication (currently mocked) is used for signup and password reset
* **Response Parsing**: JSON responses are deserialized using System.Text.Json

All calls are wrapped in try/catch blocks for fail-safety and display appropriate alerts for errors.

# **Signup Module**

**1. Purpose**

The **Signup Module** allows a new or existing banking customer to securely register their device and create a login PIN. This process is **device-restricted**—meaning only one device per account can be active at any time. It also includes **OTP verification** to ensure authenticity.

The user must enter:

* Customer ID (numeric)
* A 4-digit secure PIN
* Confirmation of the PIN

**2. Page Structure & Flow**

**🔹 File: SignupPage.xaml**

The page provides an entry form with three fields:

* Customer ID (numeric only)
* 4-digit PIN (masked)
* Confirm PIN (masked)

It also includes:

* A **Signup** button bound to SignupCommand
* A **Login here** tap for redirecting to login, bound to LoginCommand

**🔹 Code-behind: SignupPage.xaml.cs**

Csharp

BindingContext = new SignupViewModel();

**3. ViewModel: SignupViewModel.cs**

Implements signup logic via **MVVM** using two commands:

**Properties:**

* CustomerId – integer input
* Pin / ConfirmPin – secure PINs
* SignupCommand – triggers validation & API call
* LoginCommand – navigates to login via AppShell.RecheckDeviceAsync()

**Logic Flow in ExecuteSignup:**

* Validates input fields:
  + CustomerId must be provided
  + PIN must be exactly 4 digits
  + PIN and ConfirmPin must match
* Calls **OTP Service**:
  + Sends OTP to registered mobile (mocked or actual)
  + Verifies OTP before proceeding
* Retrieves or generates **DeviceId** using Preferences
* Constructs SignupRequest
* Sends a **POST request** to the /api/auth/signup endpoint

**Handles backend response:**

* Displays success/failure alerts
* If account exists on another device, prompts user to override
* On success, stores device ID and proceeds to login context

**4. Request Model: SignupRequest.cs**

C#

public class SignupRequest

{

public int Id { get; set; } // Customer ID

public string Pin { get; set; } // 4-digit PIN (plain)

public string DeviceId { get; set; } // GUID or device-specific value

public bool ForceOverride { get; set; } = false;

}

**5. API Endpoint: POST /api/auth/signup**

**Location: AuthController.cs**

**Logic Flow:**

* Validates required fields: Id, Pin, DeviceId
* Checks if user exists in database using:

Sql

SELECT UserPassword, IMEINo FROM Customer WHERE Id = @Id

* Handles the following conditions:
  + If user doesn’t exist → returns failure
  + If already registered on a **different device**:
    - Returns failure unless ForceOverride == true
  + If already registered on **same device**:
    - Returns "Already registered, please login"
* Hashes the PIN using:

Csharp

SecurityHelper.HashPassword(Pin)

* Updates Customer table with hashed PIN and new IMEINo
* Returns a structured JSON SignupResponse

**6. Device Lock Logic (One Device per Customer)**

The system ensures **single-device login enforcement**:

* Each signup saves the **Device ID (IMEINo)** in the Customer table
* On any new signup attempt from another device:
  + Backend checks IMEINo
  + If different and ForceOverride is false, registration is blocked
  + If user confirms, ForceOverride = true, and previous device is logged out

This is crucial for **fraud prevention and session integrity**.

**7. Response Model: SignupResponse.cs**

Csharp

public class SignupResponse

{

public bool Success { get; set; }

public string Message { get; set; }

public string DeviceGuid { get; set; } // Sent for info or conflict resolution

}

**Example Responses:**

* **Success**

json

{

"Success": true,

"Message": "Signup successful. You can now login.",

"DeviceGuid": "your-device-guid"

}

* **Device Conflict**

json

{

"Success": false,

"Message": "Account already registered on another device.",

"DeviceGuid": "previous-device-guid"

}

* **Invalid Data**

json

{

"Success": false,

"Message": "PIN must be a 4-digit number."

}

**7. Summary**

The **Signup Module** is the gateway for secure onboarding of banking users. It ensures:

* Robust **input validation**
* **OTP verification** before registration
* Enforced **device lock mechanism**
* Seamless backend integration using REST APIs
* Reusability through centralized ViewModel and service classes

**Login Module**

**1. Purpose**

The **Login Module** enables an already registered user to securely access their account using their **4-digit PIN** and **registered device**. The login process is tightly coupled with the device ID (IMEINo) saved during signup, ensuring only the registered device can initiate login.

This module verifies:

* Whether the device is registered for login
* Whether the entered PIN matches the stored (hashed) PIN in the database

**2. Page Structure & Flow**

**🔹 File: LoginPage.xaml**

The UI allows users to:

* Enter their **4-digit PIN**
* Trigger the **Login** command
* Access navigation to **Forgot Password** or **Signup**

**🔹 Code-behind: LoginPage.xaml.cs**

csharp

BindingContext = new LoginViewModel();

**3. ViewModel: LoginViewModel.cs**

**Properties:**

* Pin – Bound to the secure Entry input
* LoginCommand – Validates PIN and calls API
* SignupCommand, ForgotPasswordCommand – For navigation to respective flows

**Logic Flow in ExecuteLoginAsync():**

1. Validates that:
   * PIN is provided and is exactly 4 digits
2. Retrieves the **DeviceId** from preferences
3. Constructs a LoginRequest with:
   * DeviceId
   * Pin
4. Sends a POST request to /api/auth/login
5. Handles the response:
   * On success: Stores the returned CustomerId, calls RecheckDeviceAsync()
   * On failure: Shows an alert with error message

**4. Request Model: LoginRequest.cs**

csharp

public class LoginRequest

{

public string DeviceId { get; set; }

public string Pin { get; set; }

}

The device ID is used to uniquely identify which customer is attempting to log in from which device.

**5. API Endpoint: POST /api/auth/login**

**Location: AuthController.cs → Login()**

**Logic Flow:**

* Validates input: DeviceId and Pin are required
* Queries the Customer table by IMEINo

sql

SELECT Id, UserPassword FROM Customer WHERE IMEINo = @DeviceId

* If no match is found:
  + Responds with "No customer found for this device. Please sign up."
* If a password exists:
  + The entered PIN is hashed using:

csharp

SecurityHelper.HashPassword(pin)

* + Compared with stored hash
  + If matches → success, else → "Invalid PIN"

**6. Device Lock Logic (Reinforced at Login)**

* **Only the device ID (IMEINo) registered at signup** can be used to login
* Any attempt from a different device is invalid unless the user goes through re-registration
* This guards against **multi-device access and unauthorized login**

**7. Response Model: LoginResponse.cs**

csharp

public class LoginResponse

{

public bool Success { get; set; }

public string Message { get; set; }

public int Id { get; set; } // CustomerId (if login succeeds)

}

**Example Responses:**

* **Successful Login**

json

{

"Success": true,

"Message": "Login successful.",

"Id": 101

}

* **Invalid PIN**

json

{

"Success": false,

"Message": "Invalid PIN.",

"Id": 0

}

* **Unregistered Device**

json

{

"Success": false,

"Message": "No customer found for this device. Please sign up.",

"Id": 0

}

**8. Summary**

The **Login Module** is a critical security checkpoint in the application. It enforces:

* Login only from the registered device
* 4-digit PIN authentication (hashed)
* Proper user feedback on login issues
* Navigation to Forgot Password or Signup if needed

It integrates tightly with the **device lock policy** enforced during signup, maintaining **session integrity and data privacy**.

**Forgot Password Module**

⚠️ **Note**:  
This module exists in the codebase but is currently **not used in production**. The current onboarding approach prioritizes direct device-locked signup using Customer ID and OTP.  
If this module is to be used, it **requires review and refactoring** to align with the updated authentication standards.

**1. Purpose (Outdated)**

The **Forgot Password Module** was originally designed to help users reset their login PIN using their **Aadhaar number** and **OTP verification**. It allowed:

* Verifying the user based on Aadhaar
* Sending an OTP to the registered mobile
* Allowing PIN reset only after successful OTP verification

However, this approach is no longer consistent with the current security model based on **Customer ID and device binding**.

**2. Page Structure & Flow**

**🔹File: ForgotPasswordPage.xaml**

This page includes:

* Entry for **Aadhaar number**
* **Send OTP** button to trigger mobile OTP
* Section (conditionally visible) for:
  + New password input
  + Confirm password
  + Reset button

The page layout and conditional rendering are controlled via the IsOtpVerified flag in the ViewModel.

**3. ViewModel: ForgotPasswordViewModel.cs**

**Properties:**

* Aadhaar – Aadhaar number entered by user
* NewPassword, ConfirmPassword – new credentials
* IsOtpVerified – controls visibility of the reset fields
* SendOtpCommand, ResetPasswordCommand

**Logic Flow:**

* **SendOtpAsync()**
  + Validates Aadhaar number format (12 digits)
  + Calls API /api/auth/getmobile to fetch registered mobile
  + Sends OTP using OtpService
  + If OTP is verified, sets IsOtpVerified = true
* **ResetPasswordAsync()**
  + Validates password fields
  + Sends new password to /api/auth/forgotpassword

**4. Request Model (Ad-hoc, Inline)**

The request model is defined inline during POST:

json

{

"Aadhaar": "xxxxxxxxxxxx",

"NewPassword": "yourpassword"

}

**5. API Endpoints**

**POST /api/auth/getmobile**

* Accepts Aadhaar number
* Returns registered mobile number (if any)

**POST /api/auth/forgotpassword**

* Accepts Aadhaar and new password
* If Aadhaar is found and verified, updates the user record with new hashed password

**6. Security Gaps (Requires Refactoring)**

This module has the following issues under the **current security model**:

* **Aadhaar** is no longer used as the primary identifier; **Customer ID and device ID** are
* OTP is handled using a service, but its delivery mechanism is unclear in this version
* PIN is treated as a "password", but security policies (length, complexity) may be outdated
* No device verification is included, allowing resets from any device

**7. Response Models**

**GetMobileResponse.cs**

csharp

public class GetMobileResponse

{

public string Mobile { get; set; }

}

**ForgotPasswordResponse.cs**

csharp

public class ForgotPasswordResponse

{

public bool Success { get; set; }

public string Message { get; set; }

}

**8. Summary**

|  |  |
| --- | --- |
| Status | ⚠️ Deprecated — Not in current use |
| Auth Flow | Aadhaar + OTP + New Password |
| Risk Areas | No device binding, old ID method, potential inconsistency |
| Recommendation | Update logic to match CustomerId + OTP + Device Lock |
| Migration Plan | Consider merging with Signup’s OTP system and reuse device enforcement |

**Home Module Documentation**

**1. Purpose**

The **Home Module** serves as the authenticated landing page for a customer after a successful login. It includes:

* A user greeting section displaying the customer's full name
* A **menu drawer** for quick navigation to various sections
* A grid or list of **banking features** such as fund transfer, statement, profile, etc.
* Access to **Settings**, **Contact Support**, and **Logout options**
* A visual and navigational anchor for the app post-login

This module also includes core session management features like:

* **Logout**
* **Logout from all devices**

**2. Page Structure**

**🔹 Files Involved:**

|  |  |
| --- | --- |
| File | Purpose |
| HomePage.xaml / HomePage.xaml.cs | Main home UI after login |
| MenuDrawerView.xaml / MenuDrawerView.xaml.cs | Reusable slide-out navigation drawer |
| SettingsPage.xaml / SettingsPage.xaml.cs | App settings and session management |
| AboutPage.xaml | Information about the app or developer contact |

**3. ViewModel: HomeViewModel.cs**

**Core Properties:**

|  |  |
| --- | --- |
| Property | Description |
| CustomerName | Full name of the user fetched from the API |
| CustomerId | Stored locally and used to query account data |
| MenuItems | List of features shown on the home interface |
| LogoutCommand | Command to perform standard logout |
| LogoutAllCommand | Command to logout from all devices |
| LoadUserCommand | Triggers the loading of user profile after login |

**Flow of Execution:**

1. LoadUserCommand is triggered when the HomePage appears.
2. Device ID is fetched using Preferences.Get("DeviceId").
3. CustomerId is fetched using the same.
4. An API call is made to /api/home/{customerId}.
5. If successful, user's full name is set and displayed.

**4. UI Composition**

**🔸HomePage.xaml**

Contains:

* A **welcome section** with user name
* A grid-based **feature list**
* **Logout icon** in header or settings
* Menu drawer access for additional options

**🔸 MenuDrawerView.xaml**

Included via layout binding or shell drawer. Features:

* Quick access links to:
  + Profile
  + QR Code
  + Transaction History
  + Statement
  + Scan to Pay
  + Contact Support
  + Logout options
* This allows smooth navigation from anywhere in the app.

**5. API Integration**

**🔹 Endpoint: GET /api/home/{customerId}**

**Purpose:** Fetches customer details using the customer ID stored on the device.

**Returns:**

json

{

"success": true,

"id": 123,

"fullName": "John A. Doe",

"message": "Device recognized."

}

Used in: HomeViewModel.LoadUserCommand

**🔹 Endpoint: POST /api/auth/logout-all**

**Purpose:** Logs the user out from all devices by removing IMEINo associated with the user.

**Input:**

json

{

"customerId": 123

}

**Output:**

json

{

"success": true,

"message": "Logged out from all devices successfully."

}

Used in: LogoutAllCommand

**6. Logout Functionality**

**🔸 Single Device Logout**

* Executed via LogoutCommand
* Clears stored device ID and customer ID
* Navigates back to LoginPage
* Maintains one-device policy

**🔸 Logout from All Devices**

* Invokes API /api/auth/logout-all
* Clears IMEINo in the database
* Clears local preferences
* Returns user to login flow
* Prevents reuse of same login from multiple devices

**7. UX Highlights**

|  |  |
| --- | --- |
| Feature | Description |
| Personalized Greeting | CustomerName fetched dynamically from API |
| Intuitive Navigation | Feature tiles and drawer layout ensure discoverability |
| Secure Sessions | Integrated logout and logout-all improve security |
| Responsive UI | Built with vertical and horizontal layouts for device flexibility |

**8. Summary**

|  |  |
| --- | --- |
| Component | Responsibility |
| HomePage.xaml | Presents main UI and banking features |
| MenuDrawerView.xaml | Provides app-wide navigation drawer |
| SettingsPage.xaml | Houses session management controls |
| HomeViewModel.cs | Orchestrates API calls and command handling |
| API /api/home/{customerId} | Provides user context and data post-login |
| API /api/auth/logout-all | Ensures user can log out of all active sessions |

This **Home Module** forms the foundation of the app’s authenticated experience. It ties together user identity, session control, and access to core banking features, following best practices for user engagement and security.

**Statement Page Module Documentation**

**1. Purpose**

The **Statement Module** allows users to view detailed transaction history and account ledgers based on selected account types, date ranges, and schemes. It is crucial for transparency, financial tracking, and auditing within the banking app. This module is also integrated with PDF export functionality for offline viewing or record-keeping.

**2. Page Structure**

**Files Involved:**

|  |  |
| --- | --- |
| File | Purpose |
| StatementPage.xaml / .cs | Initial page for selecting account type and loading accounts |
| ViewStatementPage.xaml / .cs | Displays the transaction ledger in a table format |
| CustomerLedgerPage.xaml / .cs | Alternate view for ledger data (if used) |

**3. Flow of Execution**

1. **StatementPage.xaml**
   * Displays dropdown to select **Account Type** (e.g., Savings, Deposit, Pigmy)
   * Upon selection, fetches list of **active accounts** for that type
   * Displays these accounts in a selectable list/table format
   * Allows user to proceed to the next step via "View Statement" button
2. **ViewStatementPage.xaml**
   * Accepts selected account data via navigation query parameters
   * Provides **date pickers** for "From" and "To" date selection
   * On command execution, fetches ledger transactions via API call
   * Displays ledger in **scrollable CollectionView**
   * Includes **PDF Export** functionality

**4. ViewModels**

**StatementViewModel.cs**

|  |  |
| --- | --- |
| Property | Description |
| AccountTypes | Populates dropdown using API call |
| SelectedAccountType | Bound to dropdown selection |
| Accounts | List of accounts under selected type |
| SelectedAccount | Account selected by the user |
| LoadAccountsCommand | API call for fetching accounts |
| ProceedCommand | Navigates to ViewStatementPage with data |

**ViewStatementViewModel.cs**

|  |  |
| --- | --- |
| Property | Description |
| Transactions | ObservableCollection of ledger records |
| FromDate / ToDate | Bound to DatePickers |
| IsStatementVisible | Controls ledger visibility |
| LoadStatementCommand | Fetches ledger from server |
| ExportPdfCommand | Triggers export to PDF |

**5. API Integration**

**1. Get Account Types**

**Endpoint:** GET /api/accounttypes

* Populates dropdown in StatementPage

**Returns:**

["Savings", "Deposit", "Pigmy"]

**2. Get Customer Accounts**

**Endpoint:** POST /api/accounts/customer

**Request:**

{

"CustomerId": 101,

"AccountType": "Savings"

}

**Response:**

[

{

"AccountNumber": "123456",

"Balance": 2500.00,

"SubSchemeId": 1,

"PigmyAgentId": null

},

...

]

**Used In:** LoadAccountsCommand

**3. Get Statement**

**Endpoint:** POST /api/statement/transaction

**Request:**

{

"CustomerId": 101,

"AccountNumber": "123456",

"FromDate": "2024-01-01",

"ToDate": "2024-01-31",

"SubSchemeId": 1,

"PigmyAgentId": null

}

**Response:**

[

{

"TransactionDate": "2024-01-10",

"Narration": "Deposit",

"Deposite": 500,

"Withdraw": 0,

"Balance": 3000

},

...

]

**Used In:** LoadStatementCommand

**6. PDF Export Logic**

**🔹 Class: StatementPdfExporter.cs**

* Receives customer name, account details, and transaction list
* Dynamically formats data into tabular PDF layout
* Saves file to platform-specific **Downloads** folder
* Filename Format: CustomerName\_AccountType\_FromDate\_ToDate.pdf
* Utilizes cross-platform APIs to ensure compatibility (Windows, Android, iOS)

**7. Navigation Flow**

StatementPage

└➜ Select Account Type

└➜ View Accounts

└➜ Proceed ➜ ViewStatementPage

└➜ Load Transactions

└➜ Export PDF

**8. Summary**

|  |  |
| --- | --- |
| Component | Description |
| StatementPage.xaml | UI to select type and account |
| ViewStatementPage.xaml | Transaction display with export option |
| StatementViewModel.cs | Controls account fetching |
| ViewStatementViewModel.cs | Loads and displays ledger |
| API /api/accounttypes | Lists available account types |
| API /api/accounts/customer | Returns accounts for selected type |
| API /api/statement/transaction | Returns filtered transaction data |
| StatementPdfExporter.cs | Responsible for PDF creation and saving |

This module provides a complete and robust implementation of **transaction statement viewing and exporting**, and follows modular best practices with proper separation of UI, logic, and data access.

**Beneficiary Module Documentation**

**1. Purpose**

The **Beneficiary Module** enables users to manage their list of beneficiaries for fund transfers. It allows users to add, view, and delete beneficiaries linked to their bank account. This module is critical to ensure secure and verified fund transfers. It also includes **navigation access points** both from the **main content area of the Home Page** and the **Menu Drawer**, providing multiple entry paths for better user experience.

**2. Page Structure**

**Files Involved:**

|  |  |
| --- | --- |
| File | Purpose |
| AddBeneficiaryPage.xaml / .cs | UI and logic for adding a new beneficiary |
| BeneficiaryDetailPage.xaml / .cs | Displays full details and actions for a specific beneficiary |
| BeneficiaryStatusPage.xaml / .cs | Lists all registered beneficiaries |

**3. Flow of Execution**

**Access Points:**

* From the **HomePage Main Content** (card UI element)
* From the **MenuDrawer** (via side navigation menu)

**Navigation Flow:**

plaintext

HomePage/MenuDrawer

└➜ BeneficiaryStatusPage

└➜ View/Add/Delete Beneficiaries

└➜ Tap ➜ BeneficiaryDetailPage (View/Edit/Delete)

**4. ViewModels**

**🔹 BeneficiaryStatusViewModel.cs**

|  |  |
| --- | --- |
| Property | Description |
| Beneficiaries | List of all fetched beneficiaries |
| LoadBeneficiariesCommand | Calls API to fetch beneficiaries linked to account |

**🔹 BeneficiaryDetailPageViewModel.cs**

|  |  |
| --- | --- |
| Property | Description |
| SelectedBeneficiary | Bound to the selected beneficiary data |
| DeleteCommand | Triggers API to delete selected beneficiary |
| GoToPaymentCommand | Navigates to fund transfer page for selected beneficiary |

**🔹 AddBeneficiaryViewModel.cs**

|  |  |
| --- | --- |
| Property | Description |
| AccountNumber, IFSC, Name, etc. | Bound to form inputs |
| AddCommand | API call to register a new beneficiary |

**5. API Integration**

**1. View Beneficiaries**

**Controller:** BeneficiariesController  
**Endpoint:** GET /api/beneficiaries?customerId={id}&accountNumber={account}

**Usage:** Fetch all registered beneficiaries for a customer  
**Used In:** LoadBeneficiariesCommand

**2. Add Beneficiary**

**Controller:** BeneficiariesController  
**Endpoint:** POST /api/beneficiaries

**Request Body:**

json

{

"CustomerId": 101,

"AccountNumber": "1234567890",

"BeneficiaryAccountNumber": "9876543210",

"IFSC": "WMDC0001234",

"Name": "Rahul Patil"

}

**Response:**

json

{

"Success": true,

"Message": "Beneficiary added successfully."

}

**Used In:** AddCommand

**3. Delete Beneficiary**

**Controller:** BeneficiariesController  
**Endpoint:** DELETE /api/beneficiaries?customerId={id}&beneficiaryAccountNumber={account}

**Used In:** DeleteCommand

**6. Module Logic Summary**

| **Feature** | **Details** |
| --- | --- |
| Add Beneficiary | User enters account details and submits; verified and stored via API |
| View Beneficiaries | Fetched via customerId and accountNumber |
| View Details | Detailed info + payment & delete action |
| Delete | Calls DELETE endpoint to remove entry |
| Entry Points | HomePage Main Section, MenuDrawer |

**7. Security Considerations**

* Beneficiary actions are restricted by CustomerId and AccountNumber
* All calls are authenticated per session logic in the app
* API input validation ensures no malformed data is submitted

**8. Summary**

|  |  |
| --- | --- |
| Component | Description |
| BeneficiaryStatusPage | Shows list of beneficiaries |
| AddBeneficiaryPage | Form to register new beneficiary |
| BeneficiaryDetailPage | Details + options (Pay, Delete) |
| API /api/beneficiaries | Central controller for view/add/delete |
| ViewModels | Handle state, commands, and navigation logic |
| Access | From Home and MenuDrawer |
| Security | Tied to user credentials and device ID |

The Beneficiary module ensures a user can manage all external accounts for fund transfer in a secure and organized manner. It is tightly integrated with the fund transfer system and supports efficient reusability of beneficiary data across modules.

**Payments Module Documentation**

**1. Purpose**

The **Payments Module** enables users to perform fund transfers to saved beneficiaries using different banking modes like NEFT and IMPS. The process includes:

* Beneficiary selection
* Amount entry
* OTP verification
* Transaction initiation
* Navigation to a confirmation screen

This module is **partially implemented**, and several areas require **bug fixes, enhancements, and error handling improvements**.

**2. Page Flow Overview**

plaintext

FundTransferPage

↓

TransactionListPage

↓

EnterAmountPage

↓

OTP → Send Transaction

↓

TransactionDetailPage

**3. Page & ViewModel Breakdown**

**🔹 1. FundTransferPage.xaml + FundTransferPageViewModel.cs**

**✔ Purpose:**

* Entry point for all payments.
* Loads list of active beneficiaries for the logged-in user.

**Logic:**

* Uses CustomerId from Preferences
* Calls: GET /api/beneficiaries/status?customerId={id}
* Binds list of BeneficiaryModel to CollectionView

**⚠️ Missing:**

* No explicit error UI if no beneficiaries are available.
* Tapping beneficiary navigates to TransactionListPage without full validation of data integrity.

**🔹 2. TransactionListPage.xaml + BeneficiaryDetailPageViewModel.cs**

**Role:**

* Displays full details of selected beneficiary.
* Temporarily repurposed to show one **beneficiary card**, not transaction history.

**Logic:**

* Again fetches all beneficiaries using:

bash

GET /api/beneficiaries/status?customerId={id}

* Filters using:

csharp

beneficiary.AccountNumber == \_accountNumber

* Displays:
  + Beneficiary full name, account number, IFSC
  + Mobile number, email, branch
* Includes **“Send Money”** button

**⚠️ Issues:**

* No fallback UI for invalid accountNumber
* Re-fetching the entire beneficiary list again is inefficient

**Navigation:**

csharp

await Shell.Current.GoToAsync(

$"EnterAmountPage?account\_number={beneficiary.AccountNumber}&customer\_id={beneficiary.CustomerId}"

);

**🔹 3. EnterAmountPage.xaml + EnterAmountViewModel.cs**

**Purpose:**

* Displays full beneficiary info again
* Allows:
  + Entering transfer amount
  + Entering optional remarks
  + Selecting payment mode (e.g., NEFT, IMPS)

**Logic:**

* Fetches Razorpay-supported payment modes from:

bash

GET https://api.razorpay.com/v1/methods

*(Only local usage – Razorpay is not integrated beyond this call)*

* Beneficiary is fetched again via:

bash

GET /api/beneficiaries/status?customerId={id}

* OTP is triggered via:

csharp

await \_otpService.SendAndVerifyOtpAsync("transaction");

* Final transaction call is:

http

POST /api/transaction/neft-transaction

**⚠️ Critical Issues:**

* ❌ SubSchemeId, AccountNumber, and BeneficiaryAccountNumber are not validated properly or passed safely.

**🔹 4. API Endpoint: /api/transaction/neft-transaction**

**✔ Controller:**

TransactionController → PostNeftTransaction

**✔ Calls:**

Stored Procedure: App\_NeftTransactionPOST

**Returns:**

json

{

"Success": true,

"TransactionId": 123,

"Message": "Transfer Successful"

}

**🔹 5. On Transaction Success**

**Navigates to:**

Csharp

await Shell.Current.GoToAsync(

$"TransactionDetailPage?transactionId={result.TransactionId}"

);

* Displays final details from TransactionDetailViewModel

**4. Bugs & Technical Issues**

|  |  |  |
| --- | --- | --- |
| Area | Problem | Fix Needed |
| TransactionListPage | Refetches all beneficiaries | Only fetch once or use shared service |
| Navigation | Case mismatch in query params (account\_number) | Use constants or helper method |
| EnterAmountViewModel | Razorpay API data is unused | Remove or integrate meaningfully |
| Data Passing | SubSchemeId, PigmyAgentId, AccountType not consistently passed | Add navigation bindings + ViewModel state |
| Validation | No checks for negative amount or empty remarks | Add before calling OTP |
| Razorpay | Modes like NEFT/RTGS shown but not enforced | Either enforce mode or hide |

**5. Summary**

|  |  |  |
| --- | --- | --- |
| Page | Role | Key Task |
| FundTransferPage | Entry point | Lists all beneficiaries |
| TransactionListPage | Shows selected beneficiary | Temporary repurpose (not showing history) |
| EnterAmountPage | Final input screen | Amount, remarks, OTP, transfer |
| TransactionDetailPage | Confirmation | Navigated to on success |
| API | **Endpoint** | **Purpose** |
| Razorpay API | GET https://api.razorpay.com/v1/methods | Lists supported transfer methods |
| Beneficiaries API | GET /api/beneficiaries/status | Fetches all beneficiary data |
| Transaction API | POST /api/transaction/neft-transaction | Final transaction execution |

**6. Suggested Improvements**

* Add strict validations in EnterAmountPage
* Move Razorpay integration to a dedicated service or remove if unused
* Refactor code to avoid duplicate beneficiary fetches
* Add success/failure UI on transaction result
* Consolidate state transfer between pages using a shared TransactionStateService (recommended)

**This module is functionally laid out but requires work on data flow, parameter handling, and validations to ensure safe money transfers.**

**Supporting Modules (Extended & Final)**

**Purpose**

This section encompasses **all remaining modules** that are not part of the primary feature flows (Signup, Login, Home, Statement, Beneficiary, Payments), but are either:

* Used for UI navigation
* Meant for feature support (e.g., profile, scanner, PDF, settings)
* Planned but not fully implemented

These modules are vital for enhancing **user interaction**, **maintainability**, and **system extensibility**.

**Final Coverage Includes:**

|  |  |  |
| --- | --- | --- |
| Module | Status | Purpose |
| Profile Page | 🔄 Planned | Placeholder for user info |
| QR Scanner | ❌ Incomplete | UI ready, no scanning logic |
| Transaction Detail | ✅ Working | Shows final transfer confirmation |
| Settings Page | ⚠️ Placeholder | No functionality |
| Logout & Logout-All | ✅ Working | Clears device info and session |
| Menu Drawer | ✅ Working | Static navigation setup |
| OTP Service | ⚠️ Simulated | Lacks SMS/email integration |
| Splash & Shell Routing | ✅ Working | Sets initial routes and themes |
| Shared Models/Services | ✅ Working | Common code for all modules |
| PDF Export Engine | ✅ Working | Only for statement module |
| Connectivity Checks | ❌ Missing | App lacks online/offline awareness |
| Error Handling Strategy | ⚠️ Partial | No global error handler |
| Device Access / Permissions | ❌ Not implemented | No use of camera, files, etc. |
| Notifications (Push) | ❌ Not available | Not integrated or scaffolded |

**Details**

**1. Profile Page**

**File:** ProfilePage.xaml *(not present yet, assumed to be planned)*  
**Status:**

**Purpose:** To show customer name, linked accounts, Aadhaar, mobile, etc.

**Remarks:**

* Not found in codebase
* May require new API endpoint like /api/customer/profile

**2. QR Scanner**

**File:** QRScannerPage.xaml  
**Status:** Incomplete

**What Works:**

* Basic UI layout

**What's Missing:**

* No logic for actual camera access
* No decoding/validation of QR
* No integration with any transfer process

**Suggestions:**

* Use ZXing.Net.MAUI or Camera.MAUI
* Route to EnterAmountPage after successful scan

**3. TransactionDetailPage**

**File:** TransactionDetailPage.xaml  
**Status:** Functional  
**Purpose:** Shows final confirmation with Transaction ID after fund transfer

**Remarks:**

* Pulls data from passed query parameter TransactionId
* Uses simple detail display

**Issues:**

* No error shown if TransactionId is invalid
* Cannot refresh/fetch manually

**4. Settings Page**

**File:** SettingsPage.xaml  
**Status:** UI only

**What Exists:**

* Menu item available
* Empty content page

**To Do:**

* Add toggle for dark/light theme
* Allow language, notification, update preferences

**5. Logout & Logout-All**

**Files:**

* AppShell.cs (LogoutCommand)
* API: POST /api/auth/logout-all

**Status:** Working

**Highlights:**

* LogoutCommand clears local storage
* Logout-All removes device ID from SQL

**Flow:**

csharp

CopyEdit

Preferences.Clear();

await Shell.Current.GoToAsync("//LoginPage");

**6. Menu Drawer**

**File:** AppShell.xaml  
**Status:** ✅ Static and Working

**Menu Items:**

* Home
* Statement
* Beneficiary
* Settings
* Logout

**Issues:**

* Static entries, not role-based
* No badge indicators or shortcuts

**7. OTP Service**

**File:** OtpService.cs  
**Status:** ⚠️ Simulated

**Used By:**

* Forgot Password
* Signup
* Fund Transfer

**What Works:**

* Generates OTP using C# Random()
* Prompts input via DisplayPromptAsync

**What’s Missing:**

* No SMS or email API
* No expiry timer, resend limit
* OTP not logged or verifiable server-side

**8. Shared Models & Helpers**

**Files:**

* CustomerModel.cs, BeneficiaryModel.cs, TransactionModel.cs
* SecurityHelper.cs, DbHelper.cs

**Status:** Core backbone of data handling

**Remarks:**

* Used across multiple modules
* Consistent property naming
* Easy to extend

**Suggestions:**

* Split DbHelper into service interfaces
* Add DTO validation logic

**9. PDF Export Engine**

**File:** StatementPdfExporter.cs  
**Status:** Only in Statement

**Highlights:**

* Exports scrollable table into neat PDF
* Saves file to platform-specific Downloads folder

**Suggestions:**

* Generalize for other use-cases (e.g., transaction receipts)
* Add watermark or branding

**10. Network / Connectivity Awareness**

**Status:** ❌ Missing

**What’s Needed:**

* Detect internet availability before API calls
* Show retry banner on failure
* Use .NET MAUI.Essentials: Connectivity API

**11. Global Error Handling**

**Status:** ⚠️ Incomplete

**Current:**

* Most error handling is inline (try/catch in ViewModels)

**Suggestions:**

* Add ErrorService to show user-friendly messages
* Centralize exceptions and API error logging

**12. Permissions (Camera, Storage)**

**Status:** Not Implemented

**Needed For:**

* QR scanner
* File Save (PDF)

**To Add:**

* Permissions.Camera, Permissions.StorageWrite
* Check and prompt before usage

**13. Push Notifications**

**Status:** Not Available

**Suggested Future Use:**

* Notify transaction success
* Marketing/Alert messages
* Use Firebase/OneSignal

**Final Suggestions Summary**

|  |  |
| --- | --- |
| Area | Recommendation |
| QR Scanner | Integrate real QR decode logic |
| OTP | Switch to SMS/email OTP provider |
| Settings Page | Add actual user controls |
| Profile Page | Create dedicated page + API |
| Error Handling | Centralize UI and logs |
| Network Awareness | Add retry on failed APIs |
| Role-based UI | Show/hide features based on customer role |
| PDF Export | Extend usage beyond statement |
| Permissions | Prompt and handle explicitly |