# Ticketing Screen Designer – User Manual

## Table of Contents

1. System Requirements
2. Installation Overview
3. Step-by-Step Setup Guide
   * Extracting the Published Application
   * Setting Up SQL Server and Database
   * Creating the Database Schema
   * Configuring the App Configuration File
   * Running the Application
4. Bank Access and Data Isolation Setup
5. Forms
6. Troubleshooting
7. Support

## System Requirements

* **Operating System:** Windows.
* **.NET Runtime:** .NET 8.0
* **SQL Server:** SQL Server 2019 (Express or full)
* **SSMS:** SQL Server Management Studio for database administration

## Installation Overview

This application is distributed as an executable file along with its dependencies, a Config folder that has an appsettings.json file and a Scripts folder that will have necessary scripts, for now it has the TicketingDesignerDB.sql script responsible for creating the database. You will need to create the SQL Server database using the provided SQL script.

## **Step-by-Step Setup Guide**

**A. Extracting the Published Application**

1. The “Published App” folder will serve as your application's installation directory.

**B. Setting Up SQL Server and Database**

1. Install SQL Server 2019 (Express or full edition) on your machine.
2. Install SQL Server Management Studio (SSMS) for database management.
3. Ensure the SQL Server service is running.

**C. Creating the Database Schema**

**Run SQL Script**

* Open SSMS, connect to your SQL Server instance, and execute the TicketingDesignerDB.sql file found inside the scripts folder in your application’s directory by opening the file in SSMS then clicking “F5” to execute query (If you want to change the database name before creating check the next step, if not skip it).

**Change Database Name (Optional)**



* When you first open the script in SSMS you should see the script beginning with the 10 lines shown in the image, to change the Database name before creation change the three instances you see in the image to the desired Database name (make sure you don’t accidently remove the quotation marks or the semicolon).

### D. Configuring the App Configuration File

1. Open the appsettings.json file (Inside the Config folder) using a text editor (like Notepad).
2. Locate the ConnectionStrings section. You will need to update the DbConnection attribute based on your SQL Server authentication method:
   * **If you are using Windows Authentication:** This method uses your Windows login credentials to authenticate with SQL Server.

Json:

"ConnectionStrings": {

"DbConnection": "Server=YOUR-SERVER-NAME;Database=YOUR-DATABASE-NAME;Integrated Security=True;TrustServerCertificate=true;"

}

**Replace** YOUR-SERVER-NAME **with your actual server name**.

**Replace** YOUR-DATABASE-NAME **with the actual DB name if you changed it. The default DB name in the script is** TicketingDesignerDB.

**Use** Integrated Security=True **if Windows Authentication is used.**

**We use TrustServerCertificate=true; only if no certificate is used, to bypass encryption.**

* + **If you are using SQL Server Authentication:** This method requires a specific SQL Server username and password.

Json:

"ConnectionStrings": {

"DbConnection": "Server=YOUR-SERVER-NAME;Database=YOUR-DATABASE-NAME;User ID=YOUR\_USERNAME;Password=YOUR\_PASSWORD;TrustServerCertificate=true;"

}

**Replace** YOUR-SERVER-NAME **with your actual server name**, YOUR\_USERNAME with your SQL Server login username, and YOUR\_PASSWORD with the corresponding password.

**We use TrustServerCertificate=true; only if no certificate is used, to bypass encryption.**

**E. Running the Application**

1. Double-click the main Ticketing-Screen-Designer.UI.exe file to launch the application.
2. Follow the on-screen prompt to access an existing bank by entering its name or create a new one.
3. You can now start managing screens and buttons within the application.

## Bank Access and Data Isolation Setup

**Important:**  
The application controls which banks a user can access. This is done inside the application (not the database itself).  
Each user must have their own SQL Server login, and each login must be mapped to the banks they are allowed to use.

### - What is BankUserMapping?

This table links SQL Server users (e.g., ABankUser) to their corresponding BankId. It is used to restrict each user's access to their assigned bank only.

### - Step-by-Step Instructions

#### ✅ 1. Create SQL Users

Before a user can run the application, you must create their SQL Server login and user:

1. Open **SQL Server Management Studio (SSMS)**.
2. Connect to your SQL Server instance with a user that has db\_owner role to ensure you have all permissions required to be able to continue with these steps.
3. Expand **Databases.**
4. Right click your database > New Query.
5. Run:

CREATE LOGIN ABankUser WITH PASSWORD = 'StrongPassword123!';

CREATE USER ABankUser FOR LOGIN ABankUser;

1. Repeat for each user.
2. Change the Login name and password as needed. The first line creates the login that you (or the user you’re creating this login for) use to connect to the SQL server instance, it also creates the user that will be mapped in the BankUserMapping.

#### ✅ 2. Grant Permissions

Each user must have basic permissions to work with the application, run in a new query:

GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::dbo TO ABankUser;

* Keep/remove the (SELECT, INSERT, UPDATE, DELETE) permissions as needed.
* Brief description of the difference between (SELECT, INSERT, UPDATE, DELETE):
* Select: allows for selecting/retrieving records
* Insert: allows for inserting records
* Update: allows for updating records
* Delete: allows for deleting records

#### ✅ 3. Determine the BankId

To get the list of existing banks and their IDs, run this in a new query:

SELECT \* FROM Bank;

Take note of the BankId you want to assign the user to.

#### ✅ 4. Assign Users to Banks

The application uses a table called **BankUserMapping**.  
This table tells the app which bank(s) each user can access.

1. To add a mapping, in a new query:

INSERT INTO BankUserMapping (UserName, BankId)

VALUES ('ABankUser', 1);

1. You can assign a single user to multiple banks:

INSERT INTO BankUserMapping (UserName, BankId) VALUES ('ABankUser', 1);

INSERT INTO BankUserMapping (UserName, BankId) VALUES ('ABankUser', 2);

1. Change the Values as needed

#### ✅ 5. What Happens in the Application

When a user launches the application:

1. The application automatically detects their SQL login name (for example, ABankUser).
2. It checks which banks this user is allowed to access by reading the **BankUserMapping** table.
3. When the user enters a bank name:
   * If the bank exists **and** the user has permission → access is granted.
   * If the bank exists but is **not assigned** to the user → access is denied.
   * If the bank **does not exist**:
     + The user will see a message asking if they want to create it.
     + If they say **Yes**, the application:
       - Creates the bank.
       - Automatically adds a row to **BankUserMapping** to link the user to that new bank.

#### ✅ 6. Verify Mappings

To see which users have access to which banks, in a new query:

SELECT \* FROM BankUserMapping;

You should see rows like:

| **UserName** | **BankId** |
| --- | --- |
| ABankUser | 1 |
| ABankUser | 2 |
| BBankUser | 3 |

#### ✅ 7. Example Scenario

1. You create:

CREATE LOGIN ABankUser WITH PASSWORD = 'StrongPassword123!';

CREATE USER ABankUser FOR LOGIN ABankUser;

1. Assign BankId = 1:

INSERT INTO BankUserMapping (UserName, BankId)

VALUES ('ABankUser', 1);

1. The user changes the credentials in appsettings.json to match the created login/user in the previous steps (refer to this [step](#_D._Configuring_the)).
2. When they run the application:
   * They type **"A Bank"**.
   * The application checks:
     + Does "A Bank" exist?
     + Is ABankUser mapped to its BankId?
   * If yes → they can continue.
   * If not → they cannot open it.
   * If the bank doesn’t exist → the application will create it **and automatically give ABankUser access**.

## Forms

### **1) Bank Form:**

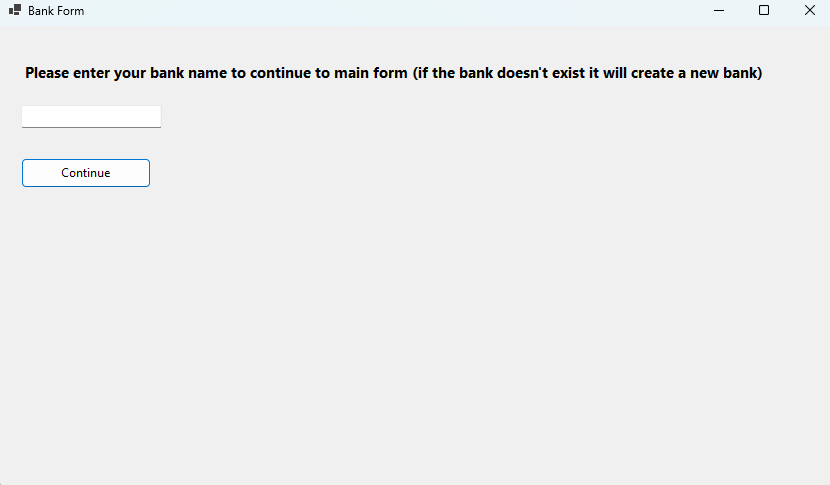


Figure 1 Bank Form

-Enter existing bank name or enter a new bank name to create one in the database

-Attempting to Continue while leaving bank name field empty shows this warning and returns you to the Bank Form to enter the bank name

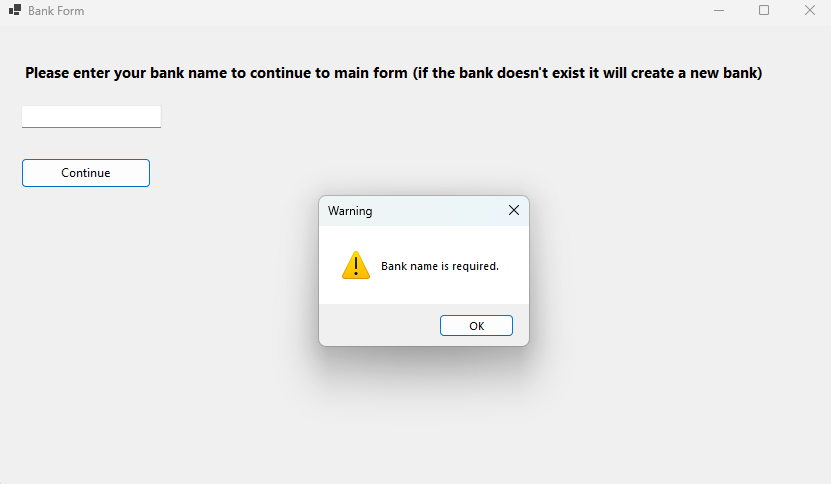


Figure 2 Bank Form Empty Field

-Entering a new bank name shows this Message Box, clicking yes will create a new bank in the database and continue to Main Form.

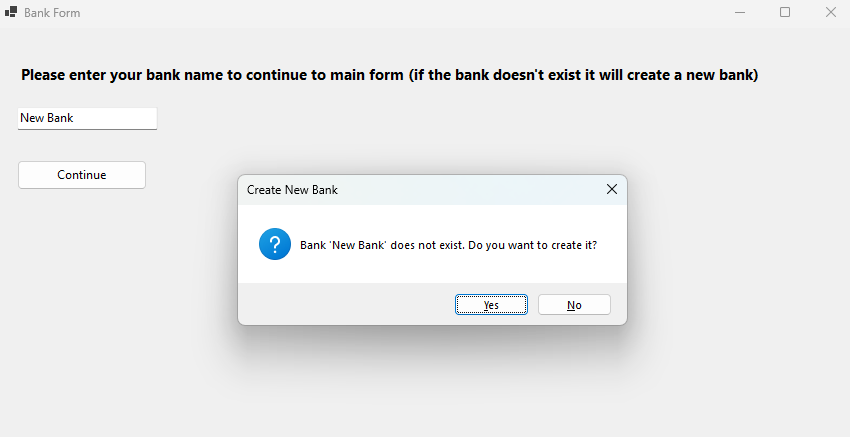


Figure 3 Bank Form | New Bank

### 2) Main Form:

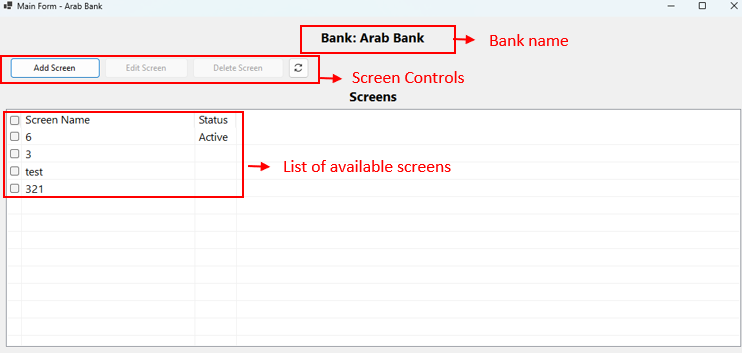


Figure 4 Main Form

-You can see the bank name you entered in the previous form, the controls for the screens as well a list of all available added screens and which screen is set as active.

-You can select one or more screens using the checkbox selection seen in the first column

-The right most control is for refreshing the screens in case another user has made changes to the screens.

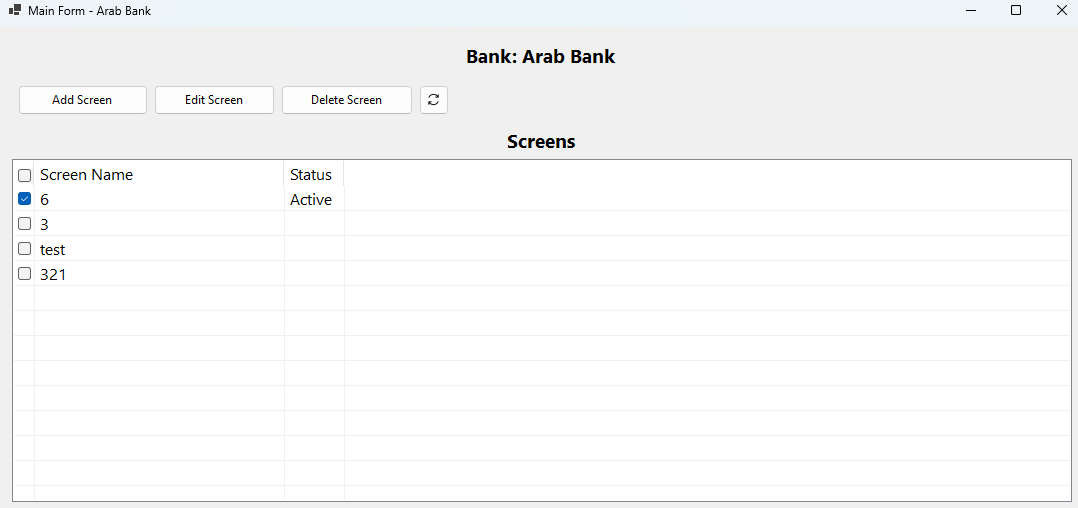
****

Figure 5 Main Form | Highlighted Screen

-Selecting one of the available screens activates the edit and delete buttons.

-Checking the checkbox in the first column header selects/deselects all screens.

### 3) Add Screen Form:

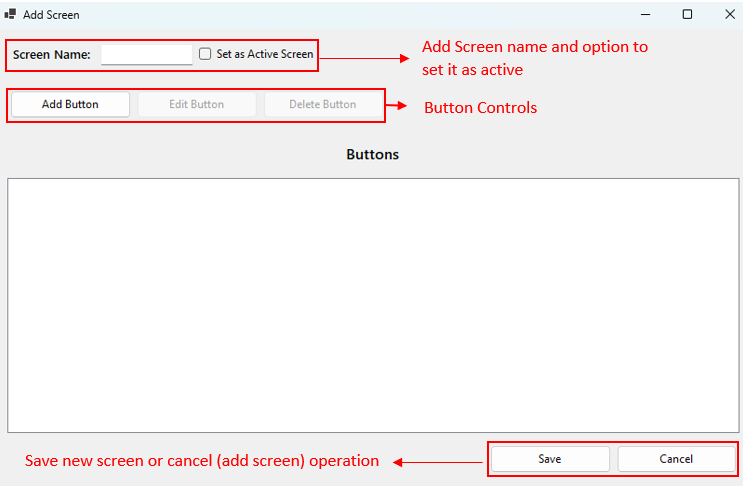
****

Figure 6 Add Screen Form

-You can only have one active screen at one time.

-Attempting to save the screen with no buttons is not allowed as per the requirement.

### 4) Edit Screen Form

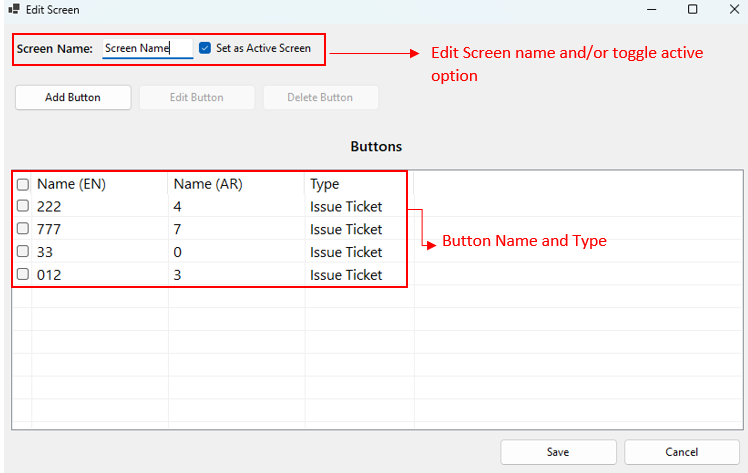
****

Figure 7 Edit Screen Form

-Edit screen name or keep it as is.

-Available buttons shown with the format:

English Button name | Arabic Button Name | Button Type

-Checkbox selection similar to screens.

### 5) Add Button Form:

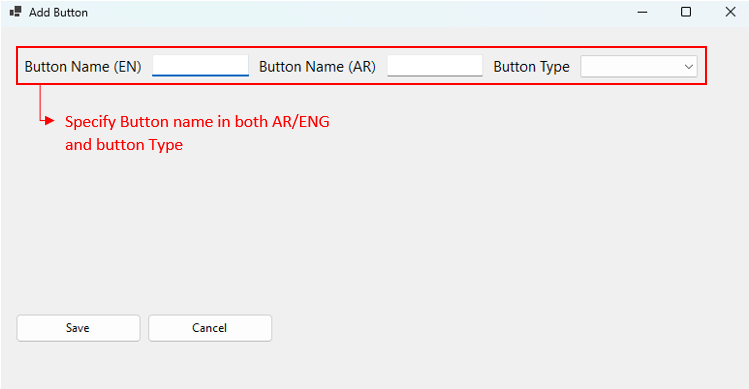


Figure 8 Add Button Form

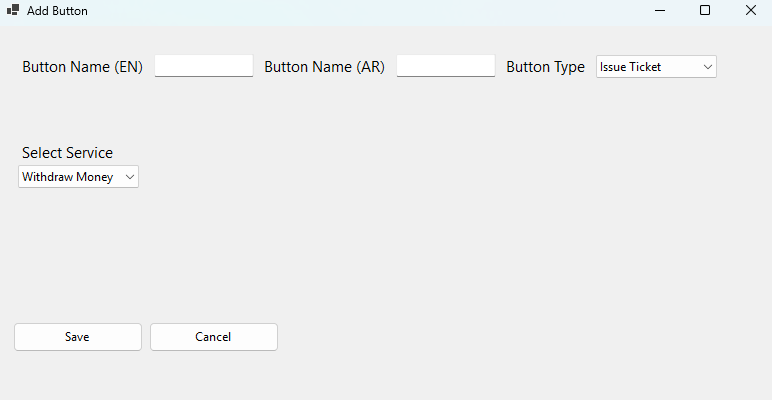
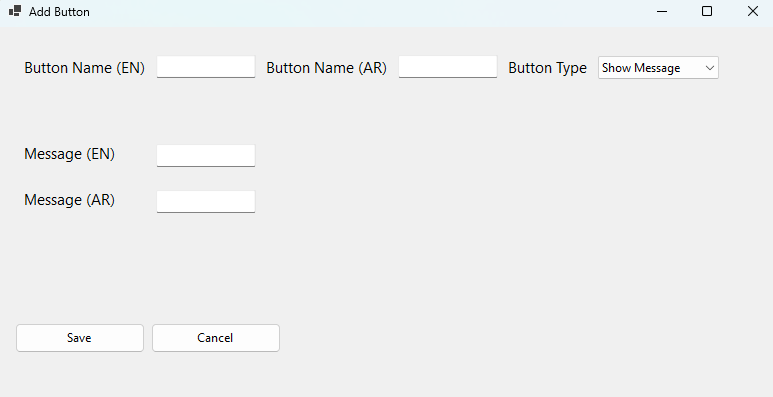


Figure Add Button | Show Message Type

Figure Add Button | Issue Ticket Type

-Only after selecting button type and filling the fields can you save the button.

### 6)Edit Button Form

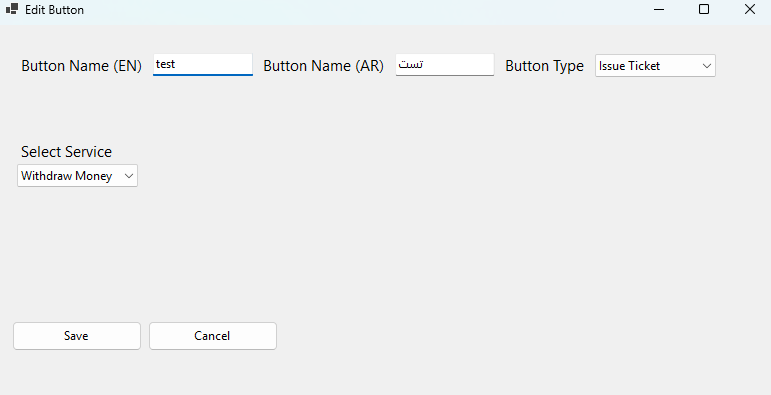
****

Figure 11 Edit Button Form

### -When running multiple instances of the application:

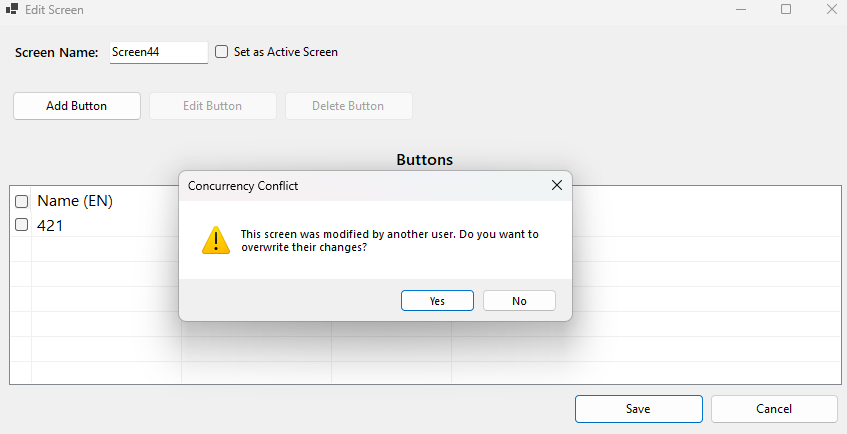


Figure 12 Edit Screen Form | Screen Concurrency Conflict

-Assume that two users are editing the same screen at the same time, when one user makes changes to the screen values (name, active toggle) and saves his changes, then the second user makes changes and attempts to save the warning seen in (**Figure 12)** is shown: clicking yes will overwrite the changes of the first user and take the latest changes of the second user, clicking no will cancel the changes of the second user and keep the changes of the first user.

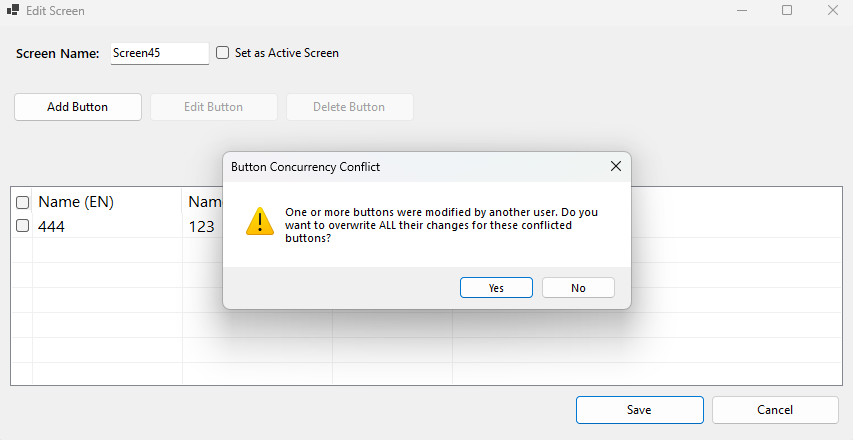


Figure 13 Edit Screen Form | Button Concurrency Conflict

-Same thing as with screen changes, when two users are editing the same button(s) and the first user saves his changes, and the second user attempts to save his changes the warning seen in (**Figure 13)** is shown clicking yes will overwrite all button changes of the first user and take the latest changes of the second user, clicking no will cancel the changes of the second user and keep the changes of the first user.

-Both warnings seen above can show if changes to both screen and button(s) are made, this ensures that the concurrency conflicts are solved gracefully.

## Troubleshooting

| **Message Shown in App** | **What It Means / What to Do** |
| --- | --- |
| **Failed to connect to the database. Please check your network or database server.** | The app could not reach the SQL Server. Ensure: SQL Server is running and accessible, the server name in the config is correct, Network connection to the database is available and SQL credentials (if used) are valid |
| **The operation timed out. Please try again later.** | The app waited too long for a response, usually from the database. Check for: Slow network or server, Heavy database load. Retry after a few seconds |
| **An internal operation failed. Please try again.** | An unexpected or invalid action occurred (e.g., clicking "Edit" without selecting a screen). Try the action again correctly. |
| **A file or disk operation failed. Please check file permissions or disk availability.** | The app could not access or write to a required file. Ensure: The application has read/write permission in its folder, the disk is not full or write-protected |
| **An unexpected error occurred. Please contact support.** | A general application failure happened. See the error log at AppFolder\Logs\error\_log.json, or send the file to support for diagnosis. |
| **Configuration file error: check appsettings.json.** | The app could not load the configuration due to a problem in appsettings.json. Verify: The file exists in the Config folder, JSON syntax is valid and the connection string is present and correctly formatted |
| **Config folder is missing. Please ensure all files are correctly deployed.** | The Config folder was not found next to the application. Ensure all published files are deployed properly. |
| **Config file is missing. Please ensure all files are correctly deployed.** | The file appsettings.json is missing from the Config folder. Reinstall or copy it back to the correct location. |
| **Exiting application due to error. Please try again later.** | A critical error prevented the app from loading required resources (e.g., screens or config). Check the logs and restart the app. |
| **The record you were trying to modify has been changed or deleted by another user.** | Another user has modified this screen or button. Reload the data and try again. This is part of concurrency control to avoid overwriting others’ changes. |
| **Nothing happens when clicking buttons** | Possible causes: No screen or button was selected before the action, an error occurred silently — check the logs, Restart the app and try again |

### ****Log Files****

* **Log File Location**: Published App\Logs\error\_log.json
* **The log file contains extensive information regarding any issue that might arise, which helps identify and solve any problems the application might run into.**
* Include this file when contacting support.

## Support

For further assistance, feedback, or to report issues, please contact the maintainer directly or open an issue in the project's designated repository.

# Bank Configuration Portal – User Manual

## Table of Contents

1. System Requirements
2. Installation Overview
3. Step-by-Step Setup Guide
   * Extracting the Main Application Folder
   * Setting Up SQL Server and Database
   * Creating the Database Schema
   * Configuring the App Configuration File
   * Configuring IIS
4. Bank Access & Adding Users
5. Troubleshooting
6. Support

## System Requirements

* **Operating System:** Windows.
* **.NET Runtime:** .NET 8.0
* **SQL Server:** SQL Server 2019 (Express or full)
* **SSMS:** SQL Server Management Studio for database administration
* **IIS Manager**

## Installation Overview

This application is distributed as a folder named “BankConfigurationPortal”, inside it you will find the necessary dependencies for the web application to run, a Config folder that has an appsettings.json file and a DatabaseScripts folder that will have necessary scripts, for now it has the TicketingDesignerDB-Extended.sql script responsible for creating or modifying the already existing database. You will need to create the SQL Server database using the provided SQL script if it doesn’t exist.

## **Step-by-Step Setup Guide**

### A. Extracting the Main Application Folder

1. The “BankConfigurationPortal” folder will serve as your application's main directory.

### B. Setting Up SQL Server and Database

1. Install SQL Server 2019 (Express or full edition) on your machine.
2. Install SQL Server Management Studio (SSMS) for database management.
3. Ensure the SQL Server service is running.

### C. Creating the Database Schema (Modifying the already existing one)

**Run SQL Script**

* Open SSMS, connect to your SQL Server instance, and execute the TicketingDesignerDB-Extended.sql file found inside the scripts folder in your application’s directory by opening the file in SSMS then clicking “F5” to execute query (If you want to change the database name before creating check the next step, if not skip it).

**Change Database Name (Optional)**



* When you first open the script in SSMS you should see the script beginning with the 10 lines shown in the image, to change the Database name before creation change the three instances you see in the image to the desired Database name (make sure you don’t accidently remove the quotation marks or the semicolon).
* If you are planning on modifying your already existing Database to work with the bank configuration portal make sure you put the same name as your already existing database when executing the script.

### D. Configuring the App Configuration File

1. Open the appsettings.json file (Inside the Config folder) using a text editor (like Notepad).
2. Locate the DbConnection section. You will need to update the DbConnection attribute based on your SQL Server authentication method:
   * **If you are using Windows Authentication:** This method uses your Windows login credentials to authenticate with SQL Server.

Json:

{

"DbConnection": {

"Server": "YOUR-SERVER-NAME",

"Database": "YOUR-DATABASE-NAME",

"TrustServerCertificate": true,

"IntegratedSecurity": true

}

}

Replace YOUR-SERVER-NAME with your actual server name.

Replace YOUR-DATABASE-NAME with the actual DB name if you changed it. The default DB name in the script is TicketingDesignerDB.

Use Integrated Security=True if Windows Authentication is used.

We use "TrustServerCertificate": true, only if no certificate is used, to bypass encryption.

* + **If you are using SQL Server Authentication:** This method requires a specific SQL Server username and password.

Json:

{

"DbConnection": {

"Server": "YOUR-SERVER-NAME",

"Database": "YOUR-DATABASE-NAME",

"UserId": "YOUR\_USERNAME ",

"Password": "YOUR\_PASSWORD",

"TrustServerCertificate": true,

"IntegratedSecurity": false

}

}

Replace YOUR-SERVER-NAME with your actual server name, YOUR-DATABASE-NAME with your actual database name, YOUR\_USERNAME with your SQL Server login username, and YOUR\_PASSWORD with the corresponding password.

We use "TrustServerCertificate": true, only if no certificate is used, to bypass encryption.

### E. Configuring IIS

#### Make sure IIS is on in windows (follow these steps):

* Open the Control Panel
* Go to Programs and then Programs and Features
* Select Turn Windows features on or off
* In the "Windows Features" window, check the box next to Internet Information Services (IIS) and click OK to install the web server
* Search for “Internet Information Services (IIS) Manager” in the taskbar and open it
* Check this [article](https://jotelulu.com/en-gb/support/tutorials/how-to-install-iis-on-windows-10/) if you’re lost (It’s for windows 10 but steps should be the same for windows 11).

#### Create a New Website:

* In the Connections pane, expand the server node and right-click on Sites.
* Select Add Website....
* Configure the website with the following settings:
* Site name: Enter a name for your website (e.g., BankConfigurationPortal) Note: This will create an ApplicationPool with the same name as your website.
* Physical path: Browse to and select the Main Application folder on your server.
* Binding: Configure the binding information (type, IP address, and port) for how users will access the site. For example, you can use the default settings (Type: http | IP address: All Unassigned | Port: 80) Note: Make sure the port isn’t occupied by another site.
* Click OK.
* Check the [documentation](https://learn.microsoft.com/en-us/iis/get-started/getting-started-with-iis/create-a-web-site) for more information regarding creating a website

#### **Granting File Permissions to the Application Folder:** Follow these steps to grant the necessary permissions to the IUSRS account, which is used by the web server to access your application's files:

* Locate your application's main directory (e.g., the "BankConfigurationPortal" folder) in File Explorer.
* Right-click the folder and select Properties.
* Go to the Security tab and click the Edit... button.
* In the permissions dialog, click the Add... button.
* In the "Select Users or Groups" window, enter IUSRS in the "Enter the object names to select" field and click Check Names.
* Click OK.
* In the permissions list, select the IUSRS group and check the box for Full control to grant all necessary permissions.
* Click Apply and then OK to save the changes.

#### Granting Admin Privileges to Application Pool:

* In the Connections pane, go to the Application Pools, right-click the application pool that has your website.
* Click Advanced Settings…
* Go under **Process Model 🡪 Identity** click the three dots.
* Click on Custom account and enter the credentials for an admin account on the server, click ok on both modals.
* If you don’t have an admin account or want to create a new one check the steps below:
  1. **Open Computer Management**
* Press the Windows key + X on your keyboard.
* From the pop-up menu, select Computer Management.
  1. **Navigate to Local Users and Groups**
* In the left-hand pane of the Computer Management window, expand System Tools.
* Click on Local Users and Groups.
* Click on the Users folder to see a list of all user accounts on your computer.
  1. **Create a New User**
* Right-click anywhere in the list of users in the right-hand pane.
* Select New User... from the context menu.
  1. **Fill in User Details**
* A new window will appear. Fill in the following fields:
* User name: This will be the account's unique login name (this is what you enter in IIS).
* Full name: The user's full name (optional).
* Description: A brief description of the user (optional).
* Password: A strong password for the new user.
* Confirm password: Retype the password.
* Uncheck the "User must change password at next logon" box if you don't want the user to be forced to change the password immediately.
* Check the "Password never expires" box if you want a permanent password.
* Click Create, then click Close.
* The new user account is now created, but it's just a standard user.
  1. **Add the User to the Administrators Group**
* Go back to the Users folder in Computer Management.
* Right-click on the new user account you just created.
* Select Properties.
* In the Properties window, click on the Member Of tab.
* Click the Add... button.
* In the "Select Groups" window, click Advanced....
* Click Find Now.
* In the search results, find and double-click Administrators.
* Click OK on the "Select Groups" window.
* Click OK on the user's Properties window.
* The new user is now a member of the Administrators group and has administrative privileges on the computer.
* Recycle the Application Pool.

**NOTE**: This step is necessary to ensure Windows Event Logger Source Is created successfully on first startup of the website.

#### **Creating a Database Login for the Application Pool:** The application pool identity is what your website uses to connect to the database. You must create a SQL Server login for this specific identity to allow it to authenticate and access the database:

* Open **SQL Server Management Studio (SSMS)** and connect to your SQL Server instance.
* In the Object Explorer, expand **Security**, right-click on **Logins**, and select **New Login...**.
* In the "Login name" field, search for the admin user you created in the previous step. For example, <Machine Name>\Admin.
* Navigate to the **User Mapping** page in the left-hand pane.
* Check the box next to your database (e.g., TicketingDesignerDB).
* In the "Database role membership for..." section, check the db\_datareader and db\_datawriter boxes to give the application permission to read and write to the database. You may also check db\_owner for full permissions.
* Click **OK** to create the login and map it to the database.

#### Verify the Deployment:

* To ensure everything is working, you can try to browse the website from IIS Manager or open a web browser and navigate to the URL you configured (e.g., http://localhost:80). This should launch the application, and you'll be able to proceed to the next steps of your user manual.

#### Resolving Anti-Forgery Token Errors: If you encounter the error message "The anti-forgery cookie token and form field token do not match" in the [log file](#_Log_Files) after deployment, you need to configure a static machine key in IIS. This is a common requirement to ensure that the application consistently uses the same key for generating and validating security tokens:

* Open IIS Manager.
* In the Connections pane, select your website (e.g., BankConfigurationPortal).
* In the main pane, double-click Machine Key.
* In the "Machine Key" dialog, uncheck the "Generate a unique key for each application" checkbox under both the Validation key and Decryption key sections.
* Click the Generate Keys button in the Actions pane on the right. This will generate unique, static keys for your application.
* Click Apply to save the changes.
* IIS will automatically update your application's web.config file with the generated keys.
* By performing this step, you ensure the application uses a consistent key for all sessions, which is crucial for the anti-forgery token validation to succeed.

## Bank Access & Adding Users (Create/Reset → Login → Change Password)

### Prerequisites (one-time setup for the maintenance tool)

1. Go to your Application Folder, Open **Web.config in a text editor → find the <appSettings>**, you will find these two fields:

<add key="MaintenanceMode" value="false"/>

<add key="MaintenanceSecret" value="YOUR\_LONG\_RANDOM\_SECRET"/>

* You need to change the value="false" to true, and the value="YOUR\_LONG\_RANDOM\_SECRET" to a secure random long string (e.g., YJ6P3n4fT\_u9zR\_V2yL0bXc1mG5e)
* You’ll open the page with a secret-bearing URL:  
  http(s)://<your-host>/Maintenance/CreateOrResetUser?secret=YOUR\_LONG\_RANDOM\_SECRET
* You will need to manually enter the URL as there is no button to lead to this page for security reasons.
* Requests not matching the secret are blocked with your custom forbidden access page.

Turn **MaintenanceMode** back to false after you finish creating/resetting users.

### Step A — Create or reset an application user (admin-only)

1. Browse to:  
   http(s)://<your-host>/Maintenance/CreateOrResetUser?secret=YOUR\_LONG\_RANDOM\_SECRET
2. Fill the form:
   * **User Name**: the application username (e.g., ABankUser)
   * **Bank Id**: the **existing** BankId the user should access
3. Submit. The page displays a **Generated Password**:
   * If the user didn’t exist → user is created + mapped to the bank.
   * If the user existed → the password is reset and bank mapping ensured.
4. Provide the **generated password** to the user (securely).  
   The account will be flagged **Must Change Password**.

**Notes**

* The bank **must exist**; otherwise you’ll see a friendly validation error.
* The user is also added (or ensured) in **BankUserMapping** so they can access that bank.
* All credentials are stored using salted PBKDF2 (hash + salt + iterations).
* The maintenance page is protected by secret.

### Step B — User’s first login (User)

1. Go to **Login**: http(s)://<your-host>/Login
2. Enter:
   * **Bank Name** (exact)
   * **User Name**
   * **Password** (the generated one from Step A)
3. On success, the app sets up the session and sees **MustChangePassword = true** for this account.

While **MustChangePassword** is true, the app restricts navigation and redirects the user to Change Password.

### Step C — Change password (first-time)

1. The navbar shows a **Change Password** button and/or redirects directly.
2. Open: http(s)://<your-host>/Login/ChangePassword
3. Enter:
   * **Old Password** (the generated one)
   * **New Password** (min length enforced)
   * **Confirm Password**
4. Submit. On success:
   * **MustChangePassword** becomes **false**
   * The user is redirected to **Branch** page (normal access)
   * The navbar no longer forces Change Password

Users **cannot** access Change Password again unless reset by the admin again; direct navigation is blocked once the flag is cleared.

## Troubleshooting

| **Message shown in app** | **What it means / What to do** |
| --- | --- |
| **Failed to connect to the database. Please check your network or database server.** | The app couldn’t open a SQL connection in DatabaseUtility.TestConnection. Typical causes: SQL Server not running/accessible, wrong server name/instance, credentials, or firewall. **Do:** Verify the connection string in Config\appsettings.json, confirm the server is reachable. Check **Logs\error\_log.json** and **Windows Event Log** (Source: *BankConfigurationPortal*) for details. |
| **The operation timed out. Please try again later.** | A DB command took too long. **Do:** Re-try; check SQL Server performance, indexing, or long-running queries. Consider adding DB timeouts and reviewing load. |
| **An internal operation failed. Please try again.** | A generic unexpected exception was caught (controller/BLL/DAL). Your logger writes to file **and** Windows Event Log. **Do:** Re-run the action; review Logs\error\_log.json and the Event Log entry for the exact exception and stack trace. |
| **A file or disk operation failed. Please check file permissions or disk availability.** | The app couldn’t read/write a file (e.g., log file, config file). **Do:** Ensure the app pool identity has read/write on the app folder (especially **Logs**), and disk isn’t full/locked. |
| **Configuration file error: check appsettings.json.** | Thrown by DatabaseUtility.TestConnection on ConfigurationErrorsException (or JSON issue during AppConfig.Initialize). **Do:** Validate Config\appsettings.json exists, is valid JSON, and includes a correct DbConnection. See logs for the message. |
| **Config folder is missing. Please ensure all files are correctly deployed.** | Thrown on DirectoryNotFoundException when Config\ is missing. **Do:** Deploy the **Config** folder alongside the app root. |
| **Config file is missing. Please ensure all files are correctly deployed.** | appsettings.json missing (caught as FileNotFoundException). **Do:** Copy Config\appsettings.json back; ensure build/publish includes it. |
| **Application Error** (Error/Generic) | The app redirected to your custom error page with a friendly message. **Do:** Check ErrorMessage on screen (if provided) and the logs. |
| **Startup error** (Error/Startup) | Application\_Start stored a failure in Application["StartupError"] (e.g., DB test failed). **Do:** Visit **/Error/Startup** to see the message. Fix config/SQL and recycle the app pool. |
| **Page Not Found** (Error/NotFound) | The route or resource doesn’t exist (HTTP 404). **Do:** Verify the URL or navigation item. |
| **Access Denied** (Error/Forbidden) | Maintenance endpoint blocked by **MaintenanceGuard** (HTTP 403) because MaintenanceMode=false, missing/invalid ?secret=… . **Do:** Enable MaintenanceMode, pass the correct MaintenanceSecret in the query string. |
| **Unauthorized** (Error/Unauthorized) | Access attempted without authentication (or cookie invalid/expired). **Do:** Log in again. If using multiple tabs/devices, ensure each has a valid sign-in. |
| **Invalid bank name.** | No bank with that name (IBankManager.GetByNameAsync). **Do:** Re-enter the correct bank name or create it via your first task’s app. |
| **You are not authorized to access this bank.** | Username isn’t mapped to the selected bank (BankUserMapping). **Do:** Map the user to the bank (via the maintenance page). |
| **Invalid password.** | Password didn’t validate via PBKDF2 (IUserManager.ValidateCredentialsAsync). **Do:** Re-enter password or ask admin to reset via **/Maintenance/CreateOrResetUser?secret=…**. |
| **You must change your password to continue.** | MustChangePassword=true for the user. The filter redirects until a successful change. **Do:** Go to **Login → Change Password**, then continue. |
| **The record you were trying to modify has been changed or deleted by another user.** | Concurrency conflict detected on update/delete (RowVersion mismatch). **Do:** Reload the item (or use your “Force Save” option if appropriate) and try again. |
| **Service time validation errors** (e.g., *Minimum must be less than maximum*) | Custom client/server validation failed: MinServiceTimeSeconds < MaxServiceTimeSeconds and Max > Min. **Do:** Enter values within the required ranges (30–999,999) and correct order. |
| **Nothing happens when clicking buttons** | Often due to validation errors preventing submit, or JS blocked. **Do:** Check for field messages. |

## Logging

The application logs errors to **two destinations**:

1. **File log (JSON)**
2. **Windows Event Log**

Use both when diagnosing issues.

### 1) File Log (JSON)

* **Location:** Published App\Logs\error\_log.json
* **What’s inside:** One JSON object per line containing the timestamp, message (with context), and stack trace.

### 2) Windows Event Log

* **Log name:** Application (configurable)(Default Value)
* **Source:** BankConfigurationPortal (configurable) (Default Value)
* **Enabled via config:** see **Configuration** below
* **What’s inside:** Structured text entries for errors/warnings/info written by the app (same context/message as the JSON log).
* **Where to view:**
  1. Open **Event Viewer** → **Windows Logs** → **Application**
  2. **Filter Current Log…** → **Event sources:** BankConfigurationPortal

#### Configuration

In Web.config → <appSettings> make sure these values are added:

<add key="WinEventLog.Enabled" value="true"/>

<add key="WinEventLog.LogName" value="Application"/>

<add key="WinEventLog.Source" value="BankConfigurationPortal"/>

<add key="WinEventLog.BaseEventId" value="9000"/>

* Enabled: turn Windows Event logging on/off.
* LogName: Application you can change this to other Log name on your device if needed.
* Source: the event source to write under. Must exist on the server (one-time setup), you can change the default value to any other value if needed.
* BaseEventId: starting ID; the logger offsets it for Error/Warning/Info (9001 = Error, 9002 = Warning and 9003 = Info).

Note: The app **can write** to the Event Log without admin rights **once the source exists**. Creating the source is a **one-time admin** action during deployment, check next section.

#### One-Time Server Setup: Create the Event Source

**Who:** Implementation/Deployment team (server admins)  
**When:** Once per server (or after you change Source/LogName)

Creation of Event source is done automatically in-code at first startup of the site, this can be verified in the error\_log.json file you should see clear messages depending on if the source exists or not, or if the operation was unsuccessful.

* If it doesn’t exist: “Created Windows Event Log (LogName:Source) source successfully!”, this message can also be seen in the event viewer
* If it does exist: “Windows Event Log (LogName:Source) source already exists.”, this message can also be seen in the event viewer
* If the operation was unsuccessful proper exception is logged and the event logger fails silently without causing the application to crash.

To view the events:

* Open Event Viewer → Windows Logs → Application (or the log name you chose).
* Filter by Source = BankConfigurationPortal (or the source name you chose) and confirm new entries appear.

#### Notes

* Creating the Event Source **requires admin privileges**, that’s why we had to give admin privileges to the application pool when [configuring the IIS](#_Granting_Admin_Privileges), writing events does not however.
* If the source doesn’t exist, the app will **silently skip Event Log writes** (it won’t crash).
* Event messages are capped by Windows (≈31k chars). The app truncates long payloads automatically.

## Support

For further assistance, feedback, or to report issues, please contact the maintainer directly or open an issue in the project's designated repository.

# Bank Web APIs – User Manual

## 1. Overview

The APIs expose the currently active Ticketing Screen Design for a given bank. Buttons of type “ShowMessage” are always returned. Buttons of type “IssueTicket” are returned (optionally) only if their associated service is active and allocated to at least one active counter in the target branch.

## 2. Installation

Setup is the same as [BankConfigurationPortal setup](#_Installation_Overview) , only changed script name to TicketingDesignerDB-Extended+API.sql, the script has an added stored procedure needed for APIs extension.

## 3. Base URL & Conventions

* Base URL: {your-host}/api
* Media type: application/json.
* Authentication: Opaque Bearer tokens via Authorization header.
* HTTP methods are used semantically (GET to retrieve, POST to create/operate).

## 4. Authentication

APIs are protected using opaque Bearer access tokens issued by the portal. Tokens are short‑lived; clients also get a longer‑lived refresh token to obtain new access tokens without resubmitting credentials.

### 4.1 Endpoints

|  |  |  |
| --- | --- | --- |
| Method | Path | Purpose |
| POST | /auth/token | Issue access/refresh tokens |
| POST | /auth/refresh | Issue new access/refresh tokens using old refresh token |
| POST | /auth/revoke | Revoke current tokens (access and/or refresh) |
| GET | /branches | Get all branches |
| GET | /screen-design | Get active Ticketing Screen Design |

### 4.2 Issue Tokens

**HTTP**  
POST /api/auth/token

***Body*** *(JSON)*

{  
 "userName": "string",  
 "password": "string",  
 "bankName": "string"  
}

***Response 200*** *(JSON)*

{  
 "access\_token": "string",  
 "expires\_in": 20,  
 "refresh\_token": "string",  
 "refresh\_expires\_in": 1440  
}

Errors: 400 (bad input), 401 (invalid credentials), 500 (server error).

### 4.3 Refresh Tokens

**HTTP**  
POST /api/auth/refresh

***Body*** *(JSON)*

{  
 "refreshToken": "string"  
}

Response: same as Issue Tokens.

Errors: 400 (bad input), 401 (Invalid refresh token), 500 (server error).

Notes:

1. The old refresh token is invalidated (rotated) upon successful refresh.
2. Using an old refresh token Invalidates all tokens of that user, and requires re-authentication (security feature).

### 4.4 Revoke Tokens

**HTTP**  
POST /api/auth/revoke

**Headers**  
Authorization: Bearer {access-token}

***Body*** *(JSON):*

{  
 "refreshToken": "string"  
}

Errors: 400 (Missing refresh token), 401 (Invalid access/refresh token), 500 (server error).

Response: 204 No Content on success.

Note: Any token provided (access via header, refresh via body) will be revoked.

## 5**. Branch Endpoint**

**HTTP**  
GET /api/branches?includeInactive={true/false}

**Headers**  
Authorization: Bearer {access\_token}

**Query Parameters**

* includeInactive (bool, default false): If false, return all active branches, if true, return all branches regardless if they’re active or not.

***Response 200*** *(JSON)* example:

{

"id": 10055,  
 "nameEnglish": "English Branch",  
 "nameArabic”: “الفرع بالعربي",  
 "isActive": true

},

Possible Errors:

* 400 Bad Request – Invalid query parameter.
* 401 Unauthorized – Missing/invalid token.
* 404 Not Found – No branches found for this bank.
* 500 Internal Server Error – Unexpected error (see server logs).

## 6**. Ticketing Screen Design Endpoint**

**HTTP**  
GET /api/screen-design?branchId={ID}&onlyAllocated={true|false}

**Headers**  
Authorization: Bearer {access\_token}

**Purpose**  
Returns the active screen design for the authenticated bank, including its buttons.

* **ShowMessage** buttons are bank-level (not tied to a branch).
* **IssueTicket** buttons are bank-level but **filtered by branch allocations** when requested.

**Query Parameters**

* **branchId** (int, optional; default: no value): use the desired branchId from the [Branch Endpoint](#_4._Branch_Endpoint).
  + **Omitted**: Return **only** ShowMessage buttons from the active screen (branch is irrelevant).
  + **Provided (positive integer)**: Return buttons from the active screen with behavior controlled by onlyAllocated (see below).
* **onlyAllocated** (bool, optional; default: false)  
  Controls whether **IssueTicket** buttons must be backed by an allocation to at least one **active counter** in the specified branch.
  + **true**
    - **Requires** a valid branchId.
    - Include **ShowMessage** buttons (always).
    - Include **IssueTicket** buttons **only if**: ServiceId is set **and** the service is **active** **and** allocated to ≥1 **active** counter in that **active** branch.
  + **false**
    - If branchId is **omitted**: return **only** ShowMessage buttons.
    - If branchId is **provided**: include **ShowMessage** and **IssueTicket** buttons **without** allocation checks (i.e., do not filter by counter/service allocations).

***Response 200*** *(JSON)* example:

* {  
   "screenId": 1,  
   "screenName": "Main Screen",  
   "buttons": [  
   {  
   "buttonId": 10,  
   "screenId": 1,  
   "nameEnglish": "Open Account",  
   "nameArabic": "فتح حساب",  
   "buttonType": 1,  
   "messageEnglish": null,  
   "messageArabic": null,  
   "serviceId": 5  
   },  
   {  
   "buttonId": 11,  
   "screenId": 1,  
   "nameEnglish": "Welcome",  
   "nameArabic": "أهلاً وسهلاً",  
   "buttonType": 0,  
   "messageEnglish": "English Message"  
   "messageArabic": "مسج بالعربي",  
   "serviceId": null  
   }  
   ]  
  }

Bank identity is inferred from the access token; do **not** pass BankId in the parameters.

**Validation & Errors**

* 400 Bad Request
  + branchId ≤ 0.
  + onlyAllocated=true while branchId is missing.
  + Invalid query parameters.
* 401 Unauthorized - Missing/invalid token.
* 404 Not Found - No active screen exists for the bank, or no buttons match the filters.
* 500 Internal Server Error – Unexpected error (see server logs).

## 7. Error Responses

On error, the API returns an HTTP status code and a JSON payload (where applicable) with a message. Server-side errors are logged to the Windows Event Log with timestamps.

## 8. Security Considerations

* Always send tokens via the Authorization header (Bearer).
* Store access tokens in memory within the client and avoid persistent storage where possible.
* Use refresh tokens to re-issue access tokens without resending credentials.
* Revoke tokens immediately if compromise is suspected.

## 9. Usage Sequence (Typical Client Flow)

1. CallPOST /api/auth/token with bankName, userName, and password to obtain tokens.
2. Attach the access token to subsequent requests (Authorization: Bearer {token}).
3. Call GET /api/branches to browse for the branchId needed for the next step
4. Call GET /api/screen-design as needed.
5. When access token expires, call POST /api/auth/refresh with the refresh token.
6. Optionally revoke tokens using POST /api/auth/revoke.

## 10. Glossary

**Access Token:** Short‑lived token (20 min) granting access to protected APIs.

**Refresh Token:** Longer‑lived token (24 hours) used to obtain new access tokens without resending credentials.

**Button Types:** 0 = ShowMessage, 1 = IssueTicket.