System Guide

Corner Club

November 20, 2021

Team Members

Wes McNall
Zach Halderwood
Ryan Williams
Colin McNish
Ian Chacey
Jayson Nguyen

Revision History

Version	Date	Name	Description
1	11/25/21	Wes McNall	Initial Document
2	12/01/21	Wes McNall	Filling out the Document based on the structure previously set in place
3	12/08	Ian	Filled out Unit Test information

Contents

1 Introduction

The Commerce Bank application allows users to login to their account through the web portal to be able to see their transactions, notifications, and more. This document will discuss how to get the code running locally on your machine.

2 Necessary Applications / Installs

1. Visual Studio
    a. NuGet Package Manager
2. SSMS
3. Python 3
    a. pip

3 Setting up the Database

1. Download main branch code from Git
2. Open SSMS and connect to local DB
3. Inside Visual Studio project
    a. Run update-database command
    b. Make sure pyodbc is installed locally
    c. Change folder to misc/
    d. Run insertData.py
4. Confirm DB is populated locally

4 MVC Code Structure

4.1 General MVC Code Structure

The **M**odel, **V**iew, **C**ontroller design pattern allowed us to generate a website that separates many of the elements of a website into different files that communicate together in different ways. We have data models, presentation information and control information.

The Model *only* contains pure application data, there is no logic within here on how to present the data to the user

The View takes the data from the Model and presents it to the viewer. The View accesses this data but does *not* know what it means or even how the user uses it.

The Controller communicates between the Model and the View.

4.2 Important Controllers

Within the MVC Design paradigm the **Controller** exists between the View and the Model. We have three main Controllers within the project:

4.2.1 Home Controller

This Controller came along with the pre-generated code but we still utilize it to return some common views, such as the Index, the Dashboard, and the Privacy View.

4.2.2 Notifications Controller

This Controller handles different functionalities related to Notifications within the final site:

- Generating Notifications
- Returning the View for Notification Settings
    - Changing These Settings
- Deleting Notifications
- Returning Details on a Notification
- Creating Notifications
- Editing Notifications

4.2.3 Transactions Controller

This Controller handles different functionalities related to Transactions within the final site:

- Index, which is just the main Transactions page, as in returning the View that has the list of all user transactions
- Filtering Transactions
- Details of Transactions
- Creating Transactions
- Deleting Transactions

4.3 Models and what they relate to

4.3.1 AccountRecord

An Account Record is nothing but an ID and an account type. As in a Bank Account ID and the type of account it is (Savings, Checking)

4.3.2 Balance

A balance for a bank account, as a floating point value

4.3.3 BalanceIndexViewModel

A list of Balances

4.3.4 CustomerRecord

A Customer Record consists of a customer's ID, their email, and whether that email has already be claimed on the system

4.3.5 DateRecord

Just a Date.

4.3.6 MonthlyResults

A year, a month, and an amount as a floating point value

4.3.7 Notification

A Notification is nothing but a customer ID, the type of notification, a description, a Date value, if that Notification was saved and if that Notification was read

- Customer ID
- Type of Notification
- Description
- Date Notification was created
- If the Notification was saved
- If the Notification was read

4.3.8 NotificationSettings

Notification -Settings is a collection of the different type of Notification Rules and whether or not those rules are active

4.3.9 TAggregatedIndexViewModel

A list of YearMonthAggregated_Transaction

4.3.10 TCategoryAggregatedIndexViewModel

A list of YearMonthAggregated_CategoryTransactions

4.3.11 TIndexViewModel

A list of Transactions

4.3.12 Transaction

A singular Transaction.

- Customer ID
- Account ID
- Account Type
- Date the Transaction was added
- The balance of the account at the time
- The type of transaction
- The amount of the transaction
- The description of the transaction
- If it was user entered
- The category of the Transaction

4.3.13 UserTransaction

A User Transaction is nothing but a Transaction and a list of user accounts

4.3.14 YearMonthAggregated_CategoryTransactions

Contains aggregated information, grouped by the customer ID, account ID, category

- Customer ID
- Account ID
- Account TYpe
- Category
- The Month, Year, and Date as a DateTime
- The SUM of the Transactions

4.3.15 YearMonthAggregated_Transaction

Contains aggregated information, grouped by the month, customer ID and account ID

- Month Name as a string
- Customer ID
- Account ID
- Account Type
- Number of Transactions
- SUM of Transaction amount

4.4 Important Views

4.4.1 Home

The Home view contains the Privacy page, the Index page and importantly the **Dashboard** page which is a user's first experience with the application

4.4.2 Transactions

The Transactions view contains all pages related to Transactions. The **index** is the main view and an organized view of a user's transactions. This is also where information related to Creating, Deleting, and seeings Details of a Transaction are stored

4.4.3 Notifications

The Notifications view contains all pages related to Notifications. The **index** is the main view and an organized view of all active notifications. This is also where information related to the Creation, Deletion, Editing, getting Details of notifications and notification Settings are

4.5 Shared Layout

The **Shared Layout** is a special view that is, well, *shared* amongst multiple different pages. The implementation here is critical to how we display information for two different scenarios:

- If a user has yet to login
- After a user has logged in

5. Graphs

The **Graphs** view is structured using a previously undiscussed part of the MVC design pattern, the **View Component**. View Component's are similar to the idea of a partial view, where a view can exist *within* a view. But, it has a few differences as well. You can have multiple partial views or multiple view components on a single View.

However, multiple View Components on a single View can each work with a *separate* model. This is *not* true of partial views. From a developer standpoint this means you can write multiple different SQL queries to ping the database and build a single View out of these View Components.

For every Graph featured on the **Graph** view is a separate View Component called on the page. This includes every table as well, so the balances and the top 5 recent transactions are both View Components.

6. Unit Tests

Our **Unit Tests** are built using the XUnit Framework, and run by creating an **In-Memory Database.** This is a local database that is created for the sole purpose of testing with unit tests, which allows for easy modification and flexibility when building the tests themselves. The In-Memory Database does not support raw SQL statements, and due to that, the methods tested from the MVC controllers are written using **Entity Framework** methods to run the queries to the database. This allows for the tests to properly cover the most important methods of the controllers, which manage the primary functions of the application, so testing them is rather useful.