CS451 - Commerce Bank Web Application
Architecture/Design Document

Table of Contents

1. Introduction **2**

2. High Level Hierarchy **3**

2.1 Hierarchy Diagram 3

2.2 Hierarchy Description 3

3. Components Classification **3**

3.1 Model Layer 3

3.2 View Layer 4

3.3 Controller Layer 4

3.4 Record Layer 5

4. Process Architecture **6**

5. Use Case View **7**

Document Control

Change History

Revision	Change Date	Description of changes
V1.0	4/4/2021	Initial release

Document Storage

This document is stored in the project's Git repository at: https://github.com/UMKC-CS451R-Spring-2021/semester-project-group-3-commerce.

Document Owner

All team members are responsible for developing and maintaining this document.

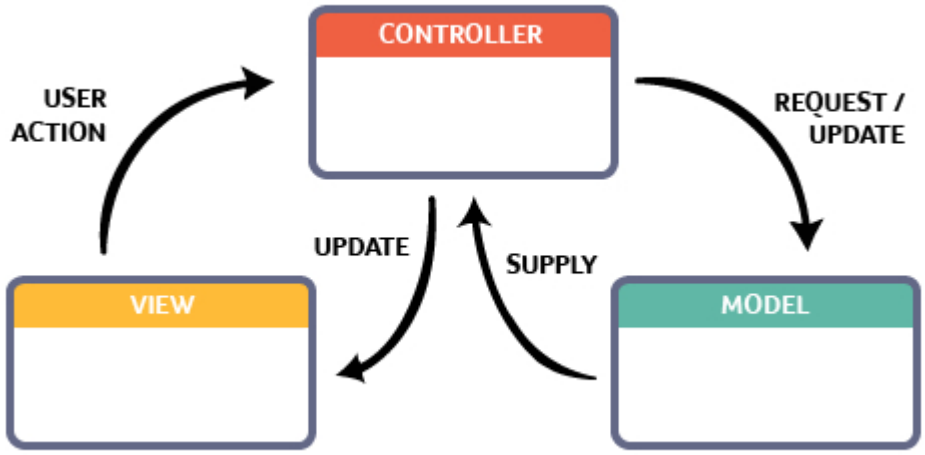
1 Introduction

The purpose of this document is to provide a detailed architecture design of the Commerce Bank Web Application. The high-level components involved in the application, as well as the relationships and interactions between these components, are featured in the following sections. There may be more architectures than detailed in this document, such as between the user and their web browser, but only the architecture that will affect the non-functional requirements will be examined and explained.

2 High Level Hierarchy

2.1 Hierarchy Diagram

Architecture Hierarchy



2.2 Hierarchy Description

The architecture hierarchy system for the Commerce Web Application is a Model - View - Controller architecture (MVC). With this architecture, we are able to maintain our separation of concerns in the application, splitting the data layer and view layer, separating them with a controller layer that drives the application flow. By doing this, we greatly ease the development process by maintaining separation of responsibility while focusing on creating a robust application.

3 Components Classification

3.1 Model Layer

Purpose: This layer will allow us to dictate the logical structure and management of our database.

Specific Nature: This layer is essential for any true functionality to be added to the website. The model will provide a logical, organized structure for our database and will

allow us to choose how data will be stored or managed. The data stored within the database will allow us to store, access, and modify any given user's account(s) and notification preferences. The structure should be robust and should be created with large amounts of user traffic in mind.

Subcomponents: ErrorViewModel, Financial_Transaction, Notification, Notification_Rule

ErrorViewModel- Used for pre-existing error control
Financial_Transaction- getters and setters to retrieve financial data

3.2 View Layer

Purpose: To display user accounts, transaction notifications, and notification management tools in such a way that promotes efficient task completion.

Specific Nature: The view layer will be responsible for displaying the proper menus and information relevant to a given user. This layer will contain the user interface that users will interact with in order to traverse the website and to view/manage their transaction notifications. The view layer will provide various forms of input that will act as listeners for events (i.e. mouse click) that will prompt the execution of any given form's corresponding code.

Subcomponents: Accounts, Home, Shared

Accounts – Uses the Financial_Transaction model to display information
Home – Displays the Home Page
Shared – Displays the entire layout for the application.

3.3 Controller Layer

Purpose: To ensure that events, which are usually triggered by a user's interaction with the view, are properly executed. These may result in changes to the model.

Specific Nature: The controller will be responsible for providing a bulk of the functionality to the project. It will dictate how events are handled and executed, as well as querying our database and modifying data found within it. It will be used to give HTML forms (i.e. login form, notification settings) functionality, allow us to fetch and view relevant user information, and create notifications.

Subcomponents: Account Controller, Home Controller
Account Controller – Returns the Financial_Transaction view.
Home Controller – Specifies how the server should respond to the request, such as writing data to the response or returning an error status code
Notification Controller – Creates, sends, and displays notifications based on rules set by the user.

3.4 Record Layer

Purpose: This layer is solely responsible for containing classes that strictly consist of data – no functionality is expected from these classes other than the ability to maintain consistency among relationships.

Specific Nature: This layer will store data pertinent to users, their financial transactions, and their notification settings.

Subcomponents: Customer Record, Transaction Record, Notification Record

Customer Record – The Customer Record class will consist of information pertaining to any given customer. Each customer will only require a single record, and each customer will have limited ability to modify their information.

- **Customer Record**
  - *User ID* – String: This is an immutable variable and refers to each user's unique, hashed user identification. This will allow us to access other user information (such as name, dob, address…)
  - *User Name* – String: This is an immutable variable and refers to the username given by the customer.
  - *User Email* – String: This is a mutable variable and refers to the email given by the user.
  - *User Password* – String: This is a mutable variable that refers to the password given by the user. Will be stored as a hashed string within the database.

Transaction Record – The Transaction Record class will consist of information pertaining to a given customer's bank account and transactions they have made. Each transaction corresponds with a single bank account, each bank account corresponds to at least a single customer, and a customer can have many bank accounts.

- **Transaction Record**
  - *Customer ID* – String: Refers to the hashed ID of a given customer.
  - *Account ID* – Int: This is an immutable variable that refers to each bank account's unique identification.
  - *Transaction ID* – Int: This is an immutable variable that refers to each transaction's unique identification.
  - *Transaction Type* – String: This is an immutable variable that refers to the type of each transaction.
  - *Transaction Amount* – Decimal: This is an immutable variable that refers to the amount of money processed in the transaction

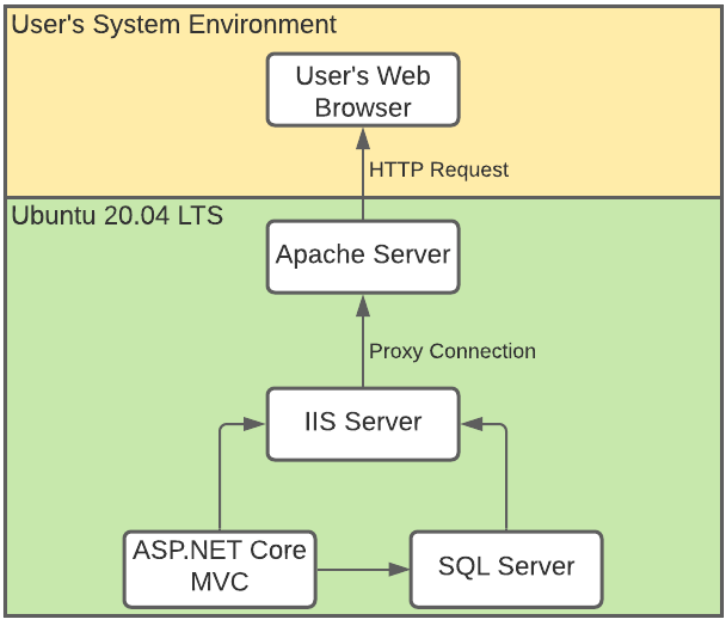
- *Transaction Description* – String: This is a mutable variable that refers to the description of the transaction.

Notification Record – The Notification Record will consist of information relevant to a customer's notification preferences, such as the message they would like to display and the conditions at which messages will display.

- **Notification Record**
  - *Notification ID* – Int: This is an immutable variable that refers to each notification's unique identification.
  - *Rule ID* – Int: This is an immutable variable that refers to each notification's unique identification.

4 *Process Architecture*

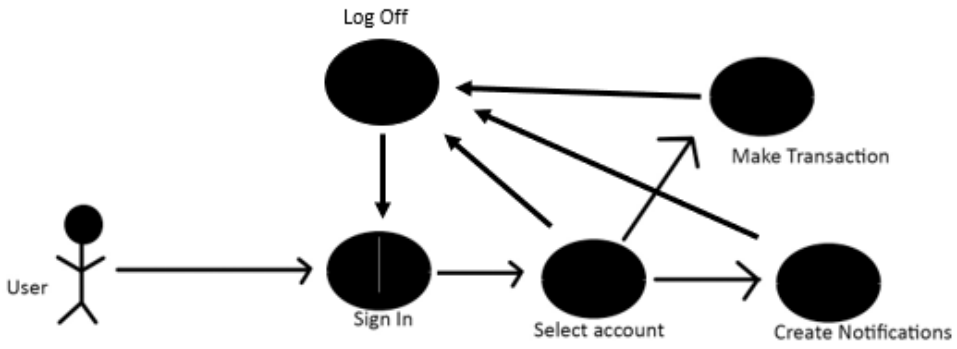
The process architecture or view consists of 5 major components:



- All requests originate from the **User's Web Browser**; the user will use a software application such as Google Chrome or Mozilla Firefox to send an HTTP request to the Apache server.
- Because the IIS Server isn't very secure for external connections, the **Apache Server** receives all external HTTP requests and acts as a reverse proxy. It sends the request to the IIS Server and handles the response.
- The **IIS Server** runs the compiled ASP.NET Core MVC object code, and it requests database information as needed from the SQL Server in order to fulfill all HTTP requests.

- The **SQL Server** is the Relational Database Management System used by the product; it provides persistent data storage, process, and retrieval for the data needed by the application.
- The **ASP.NET Core MVC** module is the main driver of the application. The MVC hierarchy is written here and operated on.

5 Use Case View



Sign In: All new users will be prompted to sign into their account upon entering the application.

Select Account: After a successful login attempt, the collection of Commerce bank accounts available to the user account will be made available for selection.

Create Notifications: The user will customize their account notifications to send alerts for various actions regarding that specific Commerce bank account, such as a notification of a large transaction or login attempt.

Make Transaction: The user will be able to add funds to or withdraw funds from their selected Commerce bank account.

Log Off: From all views, the user will have the option to log off of their user account, after which a new login prompt will be provided.