Commerce Bank Web Portal

Architecture/Design Document

**Table of Contents**

[**1**](#_heading=h.gjdgxs) **INTRODUCTION 3**

[**2**](#_heading=h.30j0zll) **DESIGN GOALS 4**

[**3**](#_heading=h.1fob9te) **SYSTEM BEHAVIOR 4**

[**4**](#_heading=h.3znysh7) **LOGICAL VIEW 5**

[**4.1**](#_heading=h.2et92p0) **High-Level Design (Architecture) 5**

[**4.2**](#_heading=h.3dy6vkm) **Mid-Level Design 6**

[**4.3**](#_heading=h.1t3h5sf) **Detailed Class Design 6**

[**5**](#_heading=h.4d34og8) **PROCESS VIEW 7**

[**6**](#_heading=h.3rdcrjn) **DEVELOPMENT VIEW** 7

[**7**](#_heading=h.2s8eyo1) **PHYSICAL VIEW** 7

[**8**](#_heading=h.17dp8vu) **USE CASE VIEW** 7

Change History

**Version:** 1.0

**Modifier:** Alec Shern

**Date:** 04/04/2020

**Description of Change:** Initial creation and draft of document.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Introduction

**Architecture and Design**

The purpose of the architecture/design document is to explain the organization of the code. A well-written architecture document will make it easier for new programmers to become familiar with the code.

The architecture/design document should identify major system components and describe their static attributes and dynamic patterns of interaction.

Software architecture and designs are typically expressed with a mix of UML models (class and sequence diagrams being the two most common) and prose. Dataflow diagrams are also helpful for understanding the interaction between components and overall flow of data through the system.

This document describes the architecture and design for the Commerce Bank Web Portal application being developed for Commerce Bank. This software allows easy access for Commerce Bank customers to view transaction data on their account. It gives the customer the means to set transaction triggers in order to see fraudulent or suspicious activity on their account and get notifications based on those triggers.

The purpose of this document is to describe the architecture and design of the Commerce Bank Web Portal application in a way that addresses the interests and concerns of all major stakeholders. For this application the major stakeholders are:

* Users and the customer – they want assurances that the architecture will provide for system functionality and exhibit desirable non-functional quality requirements such as usability, reliability, etc.
* Developers – they want an architecture that will minimize complexity and development effort.
* Project Manager – the project manager is responsible for assigning tasks and coordinating development work. He or she wants an architecture that divides the system into components of roughly equal size and complexity that can be developed simultaneously with minimal dependencies. For this to happen, the modules need well-defined interfaces. Also, because most individuals specialize in a particular skill or technology, modules should be designed around specific expertise. For example, all UI logic might be encapsulated in one module. Another might have all business logic.
* Maintenance Programmers – they want assurance that the system will be easy to evolve and maintain on into the future.

The architecture and design for a software system is complex and individual stakeholders often have specialized interests. There is no one diagram or model that can easily express a system’s architecture and design. For this reason, software architecture and design is often presented in terms of multiple views or perspectives [IEEE Std. 1471]. Here the architecture of the Commerce Bank Web Portal application is described from 4 different perspectives [1995 Krutchen]:

1. Logical View – major components, their attributes and operations. This view also includes relationships between components and their interactions. When doing OO design, class diagrams and sequence diagrams are often used to express the logical view.
2. Process View – the threads of control and processes used to execute the operations identified in the logical view.
3. Development View – how system modules map to development organization.
4. Use Case View – the use case view is used to both motivate and validate design activity. At the start of design the requirements define the functional objectives for the design. Use cases are also used to validate suggested designs. It should be possible to walk through a use case scenario and follow the interaction between high-level components. The components should have all the necessary behavior to conceptually execute a use case.

# Design Goals

There is no absolute measure for distinguishing between good and bad design. The value of a design depends on stakeholder priorities. For example, depending on the circumstances, an efficient design might be better than a maintainable one, or vice versa. Therefore, before presenting a design it is good practice to state the design priorities. The design that is offered will be judged according to how well it satisfies the stated priorities.

The design priorities for the Commerce Bank Web Portal application are:

* The design should minimize complexity and development effort.
* The design should allow for easy implementation of future feature enhancements.
* The design should be compatible between several different kinds of development environments in order to allow different developers and teams to contribute to different components of the application.
* The design should promote usability.

To achieve these goals we are using the .NET framework which works on object-oriented programming which abolishes unnecessary codes and involves less coding as it favors reusability of code which also reduces complexity. .NET allows us to specify method level security. The most important point is that it is cross-platform which allows the websites and portal to run on different platforms like Windows, Linux and macOS. Hence, the choice of the framework was crucial for this project and .NET seemed the best fit and was a requirement as well.

# System Behavior

The use case view is used to both drive the design phase and validate the output of the design phase. The architecture description presented here starts with a review of the expected system behavior in order to set the stage for the architecture description that follows. For a more detailed account of software requirements, see the requirements document.

* Login page
  + Enter existing credentials
    - Create new credentials
* Dashboard
  + View recent transactions
    - View all transactions
  + View recent notifications
    - View all notifications
* Notifications page
  + View all notifications
    - Create new transaction flags
* Transactions page
  + View all transactions
  + Export all transactions to a file

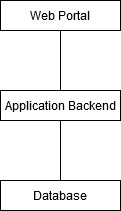
# Logical View

The logical view describes the main functional components of the system. This includes modules, the static relationships between modules, and their dynamic patterns of interaction.

In this section the modules of the system are first expressed in terms of high level components (architecture) and progressively refined into more detailed components and eventually classes with specific attributes and operations.

## *High-Level Design (Architecture)*

The high-level view or architecture consists of 3 major components:



* The **Web portal** allows customers to interact with and view Commerce Bank notifications and transactions.
* The **Application backend** controls queries to the database and interactions related to the front end and database portion of the application.
* The **Database** stores the customer data, transaction data, and notification trigger data.

## *Mid-Level Design*

The UI will be handled by view classes. These will incorporate C# and HTML code with CSS styling. The business logic will be handled by the model classes in C#. When a view receives a request from the user, it will call the correct method in the model class. This will perform the requested action and call a method on the view to update the webpage for the user.

## *Detailed Class Design*

TBD

# Process View

The application will be using a Model-View-Controller architecture in order to serve the application to users.

**Controller:**

The controller components will handle user requests and act as the main driver between the other threads. The controller will typically receive a request from the view, and then will render the requested view back to the user. The controller will also manipulate the models when a user requests to access or modify a model. The controller thread will be created when the user makes a request through the view to the web application.

**View:**

The view components will render the webpages to the user and allow interactivity so that the user may send requests to the web application. The view will display models, and push requests to the controller. The view will receive information from the controller on which view to display to the user next. This thread is the main thread the user may interact with, and will abstract the other threads.

**Model:**

The model components contain the business logic of the application and the underlying data. The model will be accessed through the controller and rendered to the user through the view in order to encapsulate and hide the data. This thread will be created by the controller when necessary, and not open to the user.

# Physical View

The database will be a single deployment. The web application may be deployed to multiple machines in order to increase responsiveness, but will all connect to the same database.

# Use Case View

**login**

1. Render home screen with login page
2. Accept input for username and password
3. if (valid?(username, password))  
    Render dashboard  
   else  
    display login error message

**dashboard**

1. Fetch transactions(username)
2. display top ten transactions for <username>
3. call flaggedTransactions(username)
4. display transactions flagged since last method call

**exportTransactions()**

1. Fetch transactions(username)
2. formatTransactions(fetchedData)
3. offer user download for formattedTransactions