Commerce Project – Group 4

Architecture/Design Document

Table of Contents

Change History

Version: 1.0

Modifier: Harrison R. Lara

Date: 03/15/2020

Description of Change: Initial draft with high level overview of development and architecture choices.

---

1 Introduction

This document describes the architecture and design for the Commerce Bank application being developed for the University of Missouri—Kansas City (UMKC). Commerce Bank project is a web application that enables the customers to manage and view their accounts. All users will have a secure login and will directed to a dashboard for an overview of each of their account's details. From there, users can view each account in detail, which allows the addition of transactions and exporting the account data. Each user will be able to update their profile and setup custom notifications rules.

The purpose of this document is to describe the architecture and design of the Commerce Bank web 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 be entirely backend focused.
- Maintenance Programmers – they want assurance that the system will be easy to evolve and maintain on into the future.
- Quality Assurance – they will system test the application as tasks come in and at the end of each iteration to make sure the application is fitting the users needs and discover any issues with implemented features so the quality of the product can stay at its peak for customers. Goal is to minimal customer difficulties and bug reports.

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 Pocket Campus Tour 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

functional design, stateless 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.

2 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 vise 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 application are:

- The design should minimize complexity and development effort.
- All page containers should have the same design patterns when using state, selectors, API calls and page layout.
    a. There should be a point were all pages have a common scaffolding that can be nearly copied for each new page then built upon following the design pattern listed above.
- The design should the same set of reusable components with interfaces that are adaptable for each piece of functionality. All components shall be functional and stateless or in other words dumb components.
- The design should be minimalist and aesthetically pleasing.
- The design should present only have three or less focus points per page to limit user confusion.

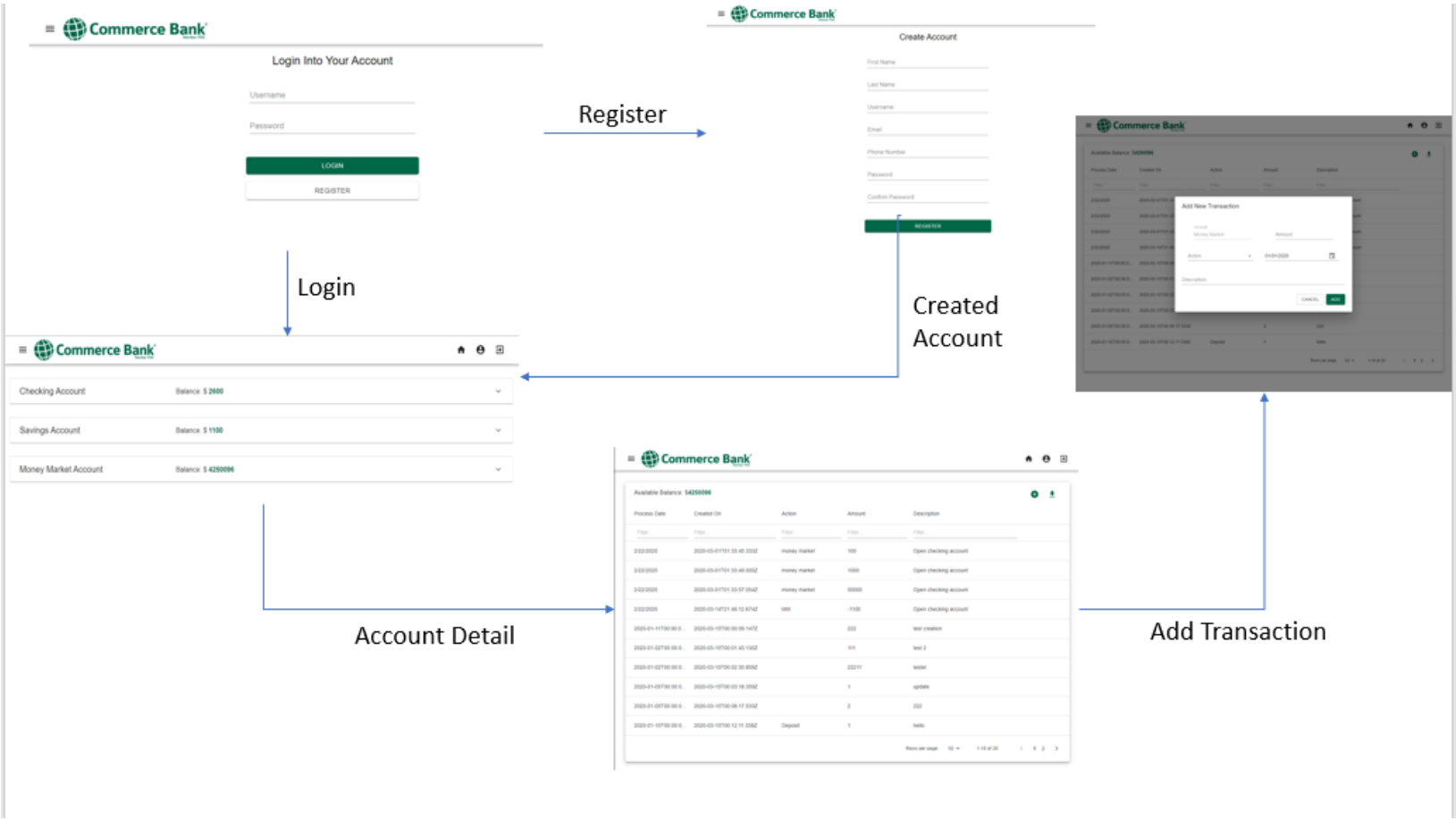
3 System Behavior

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 expect 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.

The user will land of the login page of Commerce Bank. From there, a user can either choose to login or register for a new account. Once the user has either logged in or registered, they will be directed to their dashboard which will provide a snapshot of each account and any notifications. The user can select an account and they will be taken that the selected accounts detail page where they can add new transactions, view current data

or export all account data to a spreadsheet. Each user will be able to manage their account profile and update any user information. The user can also take advantage of the side navigation menu which allows them to navigate their own account or go to another part of Commerce Bank website.

Note: These pages are for not designs, prototypes or dictating the actual site. These images are purely for workflow purposes.



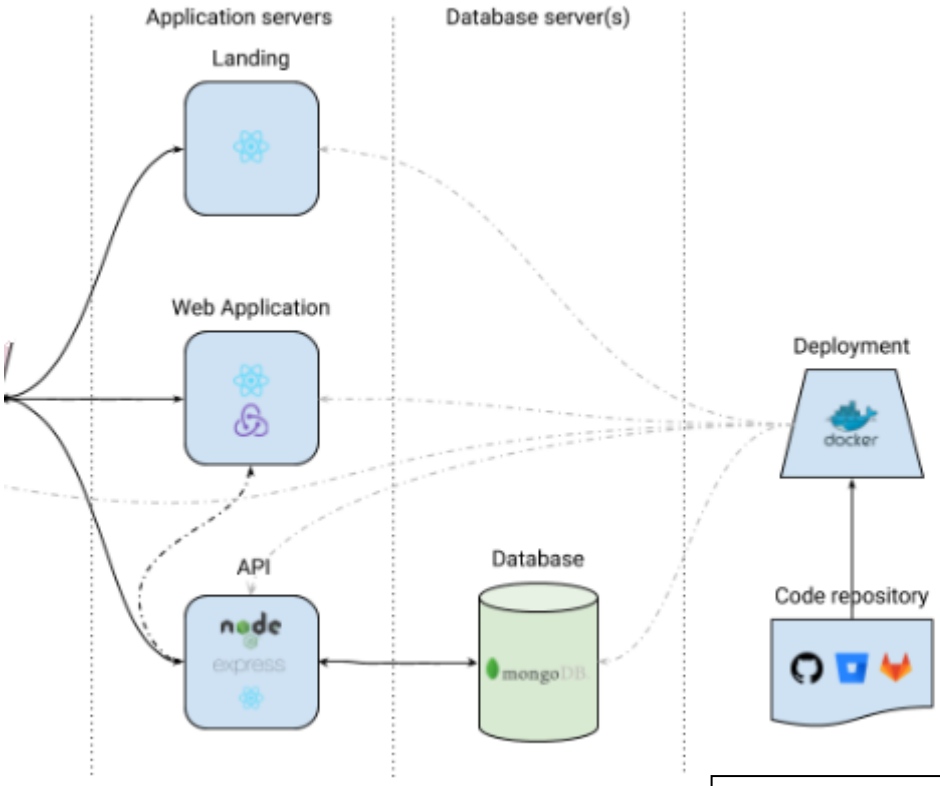
4 Logical View

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 functional components and containers with specific attributes and operations.

4.1 High-Level Design (Architecture)

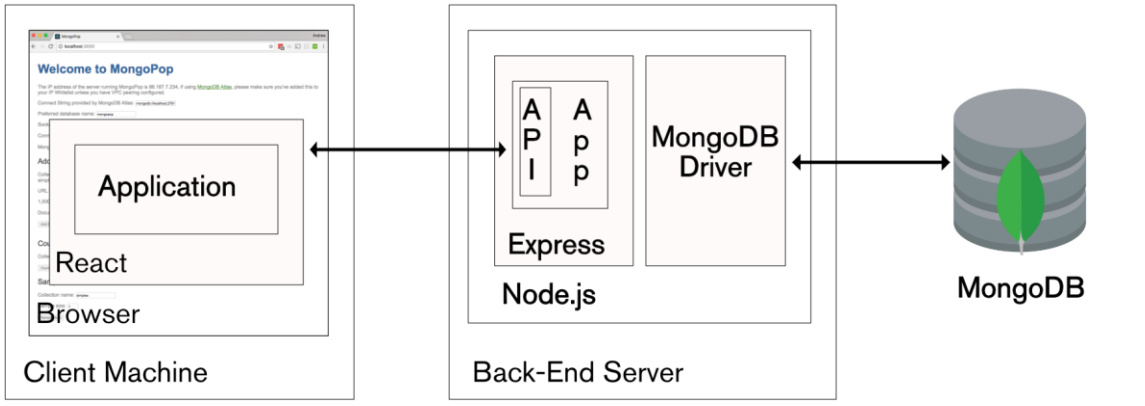
The high-level view or architecture consists of 6 major components:



1. The **web application** is the core of the application providing the essential state and functionality for the users.
2. The **Database** stores all information about the user and their accounts. This information is saved and retrieved by the API layer.
3. The **API** controls the data flow from the website to the database. This stage creates and authenticates users, gets users personal and account information.
4. The **Landing** is where all users will start when getting to Commerce Bank website. This is where the user has to either login or register.
5. The **Deployment** is how the web application will be served to the users and hosted via containers in Docker.
6. The **Code Respository** is where code will be updated, added and maintained and pushed out to the users as needed via the steps above.

4.2 Mid-Level Design

1. Components are functional and stateless therefore, they have no context of a parent.
2. The components will create each page for the user and provide the base interactions/ functionality.The router will use the components to navgiate between pages.
3. The components will use the service layer/ axios to fetch data to display to the user or save new data.
4. The controller will manage what data should be consumed and how to save and send data to the UI and database.



4.3 Detailed Functional Component Design (Presentation Layer)

1. Login
    a. Methods
        i. Validate
    b. Container
        i. Login Form
            1. Formik
        ii. React Hooks
            1. Routes
            2. Username
            3. useStyles
        iii. Axios
            1. getUser
2. Register
    a. Methods
        i. Validate
    b. Container
        i. Register Form
            1. Formik
        ii. React Hooks

1. useStyles
2. Routes
3. Username

iii. Axios

1. createUser

3. Dashboard
    a. Container
        i. React Hooks
            1. useStyles
            2. moneyMarketBalance
            3. savingsBalance
            4. username
            5. checkingBalance
        ii. Methods
            1. getBalances
        iii. Axios
            1. getMoneyMarketBalance
            2. getCheckingBalance
            3. getSavignsBalance
4. Account Detail Pages (Money Market, Checking, Savings)
    a. Container
        i. React Hooks
            1. useStyles
            2. moneyMarket
            3. checking
            4. savings
        ii. Components
            1. React Hooks
                a. Rows
                b. Filters
                c. Sorting
                d. PageSize
                e. columnWidth
                f. currentPage
        iii. Add New Transaction Form
            1. React Hooks
                a. Open
                b. selectDate
                c. amount
                d. description
                e. action
        iv. interface
            1. FormProps
        v. Methods
            1. getBalance
            2. getRows

3. handleOpen
4. handleClose
5. handleActionChange
6. handleDescriptionChange
7. handleDateChange
8. handleAmountChange
9. negativeValue
10. addNewTransaction

vi. Axios

1. createTransaction
2. getMoneyMarket
3. getSavings
4. getChecking

5 Models and Controllers (Controller Layer)

5.1 Models

1. User
    a. Username [string, unique]
    b. First name [string]
    c. Last nam e[string]
    d. Phone [string]
    e. Email [string, unique]
    f. Account number [number]
    g. Password [string]
    h. Confirm password [string]
    i. Checking account [array, account model]
    j. Money market account [array, account model]
    k. Savings account [array, account model]
2. Account
    a. Accout type [string]
    b. Processed date [date]
    c. Balance [number]
    d. Amount [number]
    e. Action type [string]
    f. Description [string]

5.2 Controllers

1. User
    a. CreateUser(payload)
    b. updateUser(payload, username)
    c. deleteUser(username)
    d. getUserById(username)
    e. getUser(username, password)
2. Accounts
    a. Checking account

i. createCheckingTransaction(payload, username)
ii. getCheckingAccount(username)
iii. getCheckingBalance(username)

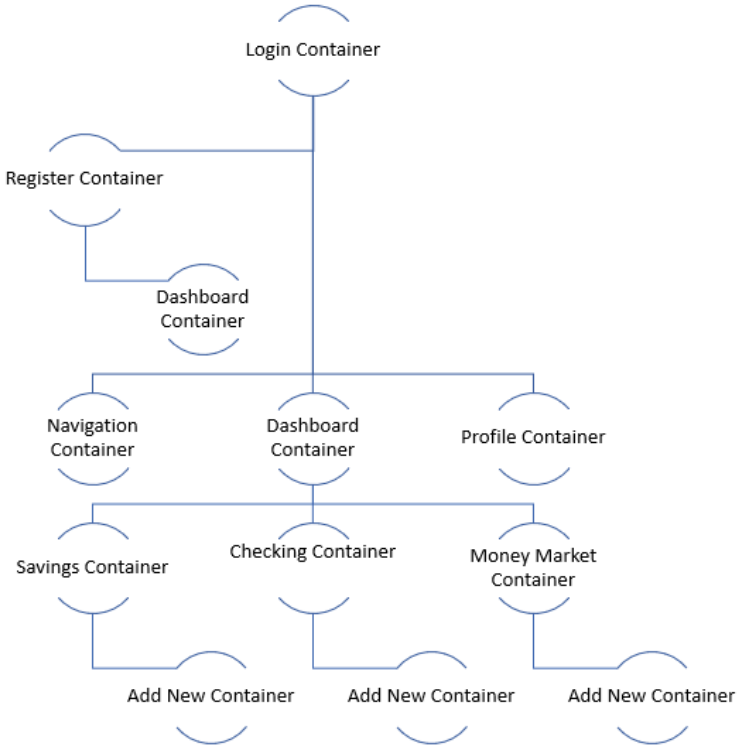
b. Savings account
i. createSavingsTransaction(payload, username)
ii. getSavingsAccount(username)
iii. getSavingsBalance(username)

c. Money Market account
i. createMoneyMarketTransaction(username, payload)
ii. getMoneyMarketAccount(username)
iii. getMoenyMarketBalance(username)

6 API Layer

1. User
   a. router.post('/user', userController.createUser);
   b. router.put('/user/:id', userController.updateUser);
   c. router.delete('/user/:id', userController.deleteUser);
   d. router.get('/user/:username', userController.getUserById);
   e. router.post('/users', userController.getUser);
2. Accounts
   a. Checking
      i. router.post('/addchecking/:username', accountController.createCheckingTransaction)
      ii. router.get('/getchecking/:username', accountController.getCheckingAccount)
      iii. router.get('/checkingbalance/:username', accountController.getCheckingBalance)
   b. Savings
      i. router.post('/addsavings/:username', accountController.createSavingsTransaction)
      ii. router.get('/getsavings/:username', accountController.getSavingsAccount)
      iii. router.get('/savingsbalance/:username', accountController.getSavingsBalance)
   c. Money Market
      i. router.post('/addmoneymarket/:username', accountController.createMoneyMarketTransaction)
      ii. router.get('/getmoneymarket/:username', accountController.getMoneyMarketAccount)
      iii. router.get('/moneymarketbalance/:username', accountController.getMoneyMarketBalance)

7 Development View



8 Use Case View

1. User Login
    a. Login form
        i. See if user exists
            1. yes
                a. Route to dashboard
            2. No
                a. Error message, user doesn't exist
        ii. Create account
            1. Route to register page
2. User register
    a. Validate form inputs
        i. No errors
            1. Create user
            2. Route to dashboard
        ii. Errors
            1. Error message, enter correct inputs
3. User dashboard
    a. Request account balances
        i. Display account balances
    b. Navigation bar adds in edit profile, logout, home button
    c. User can explore snippet of account details in dropdown

    d. User clicks checking account
        i. Route to checking account detail page
4. Account detail page
    a. Request all account data
    b. User clicks export all data
        i. Request all account data
        ii. Convert to spreadsheet
        iii. Download
    c. User clicks add new
        i. Dialog appears
        ii. Validate form
            1. No errors
                a. Create new transaction request
                b. Close modal
            2. Errors
                a. Error message, don’t create new transaction
5. User profile
    a. User updates phone number
    b. Validate field
        i. No errors
            1. Update user request
        ii. Errors
            1. Error message, enter proper input