Commerce Bank Notifications Application

Architecture/Design Document

**Table of Contents**

[**1**](#_gjdgxs) **INTRODUCTION 2-3**

[**2**](#_30j0zll) **DESIGN GOALS 3**

[**3**](#_1fob9te) **SYSTEM BEHAVIOR 3-4**

[**4**](#_3znysh7) **LOGICAL VIEW 4-7**

[**4.1 High-Level Design (Architecture)**](#_2et92p0) **4-5**

[**4.2 Mid-Level Design**](#_3dy6vkm) **6**

[**4.3 Detailed Class Design**](#_1t3h5sf) **7**

[**5**](#_4d34og8) **PROCESS VIEW 7-8**

**6 USE CASE VIEW 9-11**

Change History

**Version:** 1.0

**Modifier:** Team 2

**Date:** 4-5-2020

**Description of Change:** Initial Page

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

# **Introduction**

This document describes the architecture and design for the Commerce Notifications application being developed for Commerce Bank. The application allows users to set notification rules based on transactions that occur on their accounts. The web application has two main pages, Notifications and Transactions. The notifications page shows the users the notification rules that they have set up, with popup pages for viewing individual transactions that caused the notifications, and for adding/editing rules. The transactions page shows the user a list of their account transactions and (as a proof of concept) allows users to add transactions to their account with the potential of triggering new notifications.

The purpose of this document is to describe the architecture and design of the Commerce Notifications 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 Notifications 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**

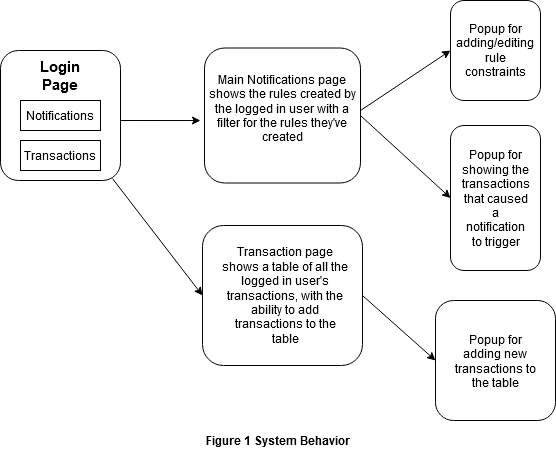
There is no absolute measure for distinguishing between good and bad design. The value of a design depends on stakeholder priorities. For example, depending on the circumstances, an efficient design might be better than a maintainable one, or vise versa. Therefore, before presenting a design it is good 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 Notifications application are:

* The design should minimize complexity and development effort.
* The design should consider the developers’ primary skills between front end and back end design.
* The design should be implemented in a way that enables future additions to be easily merged into the application..

# **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.



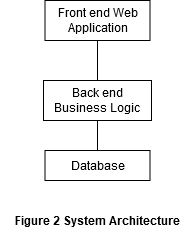
# **Logical View**

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.

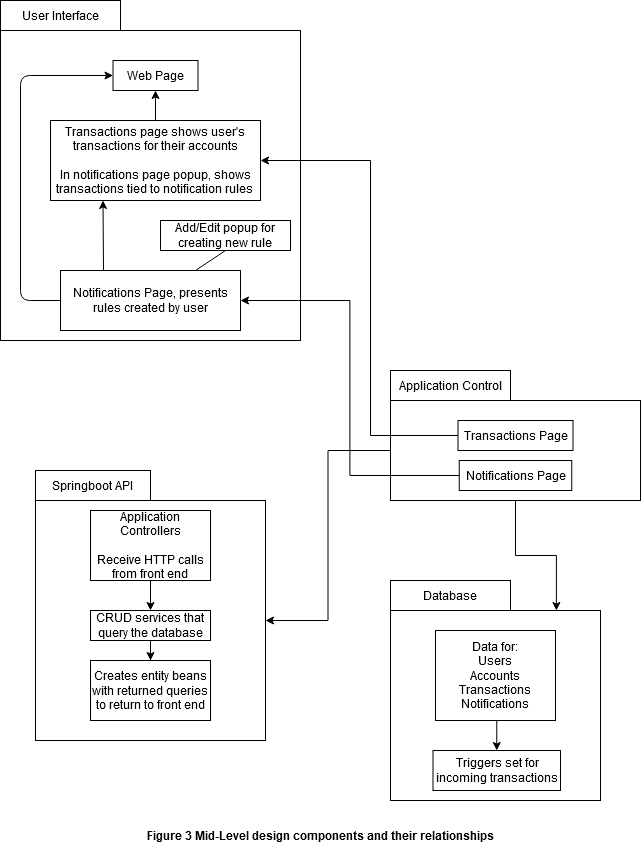
## ***High-Level Design (Architecture)***

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

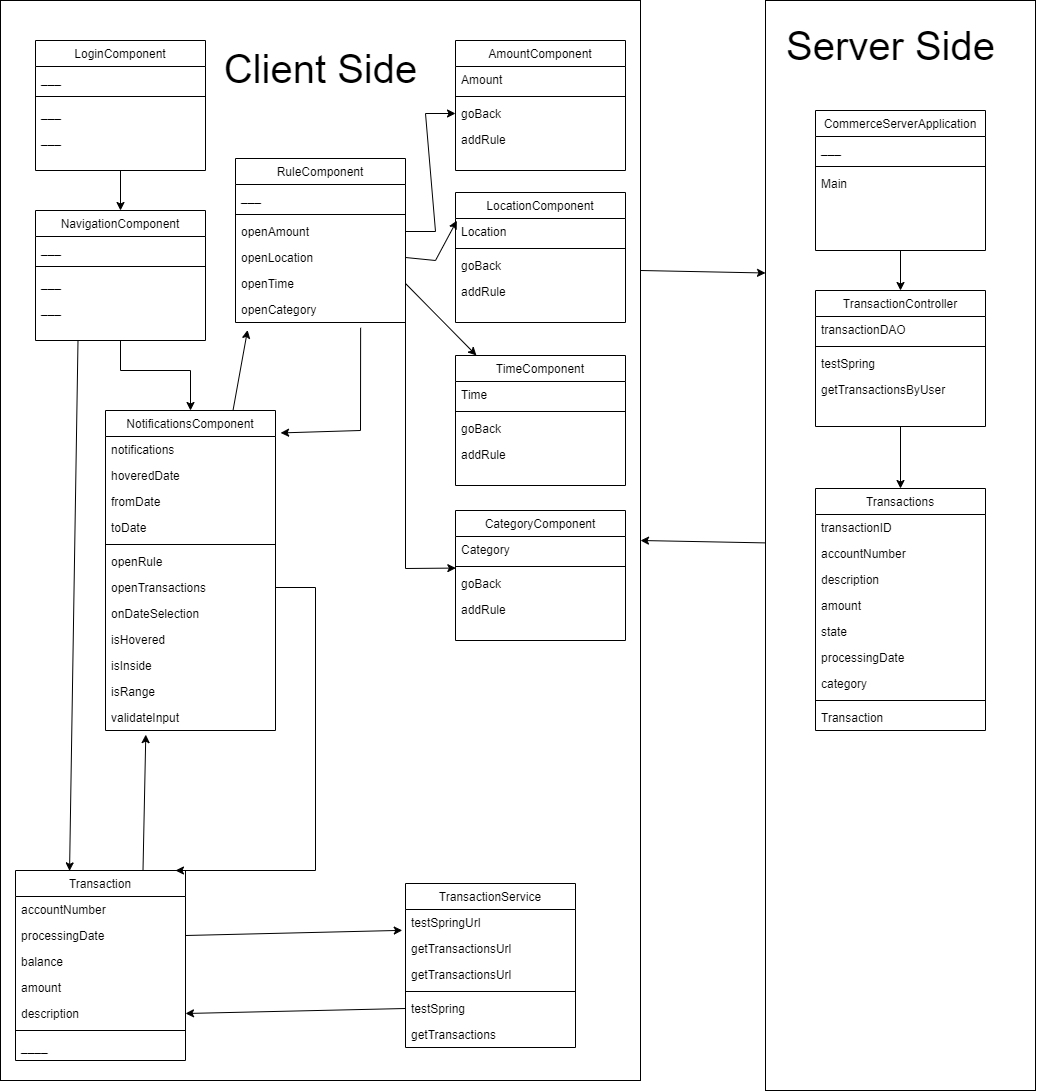


* The **Front end Web Application** is an angular and bootstrap created UI that requests user data from the back end to display on the page.
* The **Back end Business Logic** is a Springboot Java API that receives requests from the front endand makes database queries to format and return requested data
* The **Database** is a MySQL database that stores all of the applications data needed to handle user interactions with transactions and notifications

## ***Mid-Level Design***



## ***Detailed Class Design***



note: Working design, application is not finalized

# 5 **Process View**

* Front end thread

The front end thread is always awaiting user input to change pages and load/send data to the back end

* Back end thread

The back end is always listening for HTTP calls through its various endpoints. Depending on which endpoint is hit, it runs through a series of functions to either create, read, update, or delete data on the database, and returns necessary data in an expected format back to the front end.

* database thread

The database stores the data necessary for the application to function. New notification rules created on the front end make their way through the back end to be created as triggers on the transaction table in the database that listen for new transactions being created to determine whether or not they violate notification constraints that the user has set.

# Use Case View

### Use Case: 1

### Description: User Checks Notifications

Actors: Commerce Bank Customers

Basic Path

1. User logs in to application
   1. User information is verified in Spring
2. The Notifications dashboard is loaded
   1. SQL call is made by the frontend and the backend makes a SQL query to retrieve all of the notifications and then the backend serves the data to the frontend through endpoints created in Spring
3. User can view the number of times a notification can be triggered
4. User logs out
   1. A logout request is sent to Spring
5. System exits
   1. Spring ends user’s session

Alternate Path

1. User logs in to application
   1. User information attempts to be verified in Spring
2. System prompts user “Login information is incorrect” and asks user to re-enter information
   1. Spring cannot verify the login information so the backend serves a request for an error message to the frontend
3. User enters correct information and follows basic path

### Use Case: 2

**Description: User Login / Check Transactions**

Actors: Commerce Bank Customers

Basic Path

1. User logs into application
   1. User information attempts to be verified in Spring
2. User clicks “Transactions” tab in the navigation bar
   1. The frontend switches to the Transactions page
3. System loads the users transactions
   1. SQL call is made by the frontend and the backend makes a SQL query to retrieve all of the transactions for the user and then the backend serves the data to the frontend through endpoints created in Spring
4. User can view the transactions made on their account
5. User logs out
   1. A logout request is sent to Spring
6. System exits
   1. Spring ends user’s session

Alternate Path

1. User logs in to application
   1. User information attempts to be verified in Spring
2. System prompts user “Login information is incorrect” and asks user to re-enter information
   1. Spring cannot verify the login information so the backend serves a request for an error message to the frontend
3. User enters correct information and follows basic path

**Use Case: 3**

**Description: User Adds Transaction/Triggers Notification**

Actors: Commerce Bank Customers

Basic Path

1. User logs into application
   1. User information attempts to be verified in Spring
2. User clicks “Transactions” tab in the navigation bar
   1. The frontend switches to the Transactions page
3. System loads the users transactions
   1. SQL call is made by the frontend and the backend makes a SQL query to retrieve all of the transactions for the user and then the backend serves the data to the frontend through endpoints created in Spring
4. User clicks “Add Transaction” button
   1. The front end creates a form
5. Form generates for user:
   1. User checks box for deposit or withdrawal
   2. User enters amount
   3. User enters description
6. User clicks “Add” button within popup
   1. SQL call is made by the frontend and the backend makes a SQL query to insert a new transaction for the user combined with the user’s account number and the date of the transaction. The backend will verify the transaction and increment the user’s balance if the transaction is a deposit and decrement if the transaction is a withdrawal.
7. User will see transaction added to their transaction log
   1. SQL call is made by the frontend and the backend makes a SQL query to retrieve all of the transactions for the user and then the backend serves the data to the frontend through endpoints created in Spring
8. If added transaction triggers notification user will be able to see notification in notification tab
   1. The backend will scan the transactions for any notifications after the transaction is added.
9. User Clicks Log Out
   1. A logout request is sent to Spring
10. System Exits
    1. Spring ends user’s session

Alternative Path

1. User logs into application
   1. User information attempts to be verified in Spring
2. User clicks “Transactions” tab in the navigation bar
   1. The frontend switches to the Transactions page
3. System loads the users transactions
   1. SQL call is made by the frontend and the backend makes a SQL query to retrieve all of the transactions for the user and then the backend serves the data to the frontend through endpoints created in Spring
4. User clicks “Add Transaction” button
   1. The front end creates a form
5. Form generates for user:
   1. User checks box for deposit or withdrawal
   2. User enters amount
   3. User enters description
6. User clicks “Cancel” button within popup
   1. The frontend deletes the form
7. User Clicks Log Out
   1. A logout request is sent to Spring
8. System Exits
   1. Spring ends user’s session