High Level Design
MotoMoto

The New Panelists
10/3/2021

Blake Del Rey
Isabel Guzman
Jacob Sunia
James Austin Jr.
Naeun Yu

Team Leader: James Austin Jr.

Table of Contents

1. Introduction

1.1 What is the Purpose of this document?

This document explains the architecture and behavior of the web application MotoMoto and further clarifies details mentioned in the Business Requirements. Overall, the main goals of the HLD will be to ensure that the structure of the applications flow, application architecture, technology architecture, and database architecture are presented so that every stakeholder can have an understanding of how MotoMoto's system will behave.

1.2 Overview

The HLD addresses the following:

- Architecture diagrams of the application designs, and
- Describe Hardware and Software designs and clarify the components within.

2. General Description

MotoMoto is a single page application supported by outside sources such as affiliate programs and/or external databases to present data on the Presentation layer. Since our primary goal for construction will be to create a single page application (SPA), our user interface will dynamically be updated according to our Application layer. Requests are qualified via the Business layer which will communicate with the Service Manager in the Service layer to select the particular service to provide.

3. System Design and Architecture

3.1 Hardware Architecture Diagram

The diagram, Figure 1, is a generalization of which hardware MotoMoto is planned to interact with. The User will interact with the Front-End which will send requests and receive asynchronous responses (by definition of the SPA architecture) from the Back-End. The Back-End contains our Web Server and Relational Database Management System (RDBMS) which are in constant communication, but the Web Server depending on the request may query third party databases for the necessary data. The NHTSA RDBMS is read-only and is government sanctioned. The Affiliate Programs provide the necessary commercial data and analytics to support Part Price Analysis and Car Builder. The Email Client supports the Notification System feature and depending on the request, the Email Client may return asynchronous results in the form of an email directly to the user's email inbox.

3.1.1 Front-End

The user will be interacting with the application user interface located on the Google Chrome browser. Requests from the Front-End are sent to the

Back-End web server which processes requests and connects them to the proper resources.

3.1.2 Back-End

The Back-End will respond to requests and, depending on the request, the web server will verify data being transferred back to the Front-End or supply external services from APIs such as the Email Client currently.

3.2 Software Architecture Diagram

MotoMoto is a single page application and is organized into layers to separate the concerns of the application. The UI layer presents the MVC architecture style. The Application and Business layers are contained in the Server to ensure the application processes requests and responses appropriately. The Service layer employs a Microservices architecture.

3.2.1 Presentation Layer

The Presentation layer also known as the UI will represent all aspects of what the user will be able to interact with on their screen. This layer will be used to complete an action on our webpage whether it is typing or clicking. Overall, the goal of the presentation will be to provide a graphical interface of the application as well as handling any code to deal with any user interaction. Since our main architecture in this program is SPA, there will be logic represented in the presentation layer as it will dictate how information is displayed on a screen.

3.2.2 Application Layer

The Application layer will represent our middleman that will work with the Business layers logic while the application is running. In essence, this layer will represent a communicator with the Business Rules ensuring that all applicable application services can be used during the use of our program. In Figure 2 (Pg. 6) we share a brief breakdown of how the application layer will work with the UI to accomplish dynamic page transitions and work with the business layer to ensure that all business logic and validation is verified through third party API's, Internal and External RDBMS', and outsourced vendor links are communicated back to the users screen.

3.2.3 Business Layer

The Business layer will represent all the logical operations of the program that communicates between the application layer (communication to the user interface) and the database (validating information). This layer will be coded to solve all of the applications real world business logic where information will be stored, created, deleted, or modified. We have split up the business layer shown in Figure 2 (Pg. 6) into two sections: front-end and back end business rules where Front End Business Rules will represent the calculations that will be

transacted back to the user and Back End Business Rules will represent the manipulated data and server side logic that will take place.

3.2.4 Service Layer

The Back-End architecture is service-oriented, and applies particularly microservices. Microservices that are currently offered are Data, Vendor, and API resources. We compensate the Data layer with additional logic by allowing the Service Manager to communicate with the Data Access layer to ensure proper request and response. Vendor services will be provided wherever vendor products are presented and selected in the application; features with this functionality are Car Builder, Part Price Analysis, and Car Build Posts in the Community Board. The API will need access to

5. Security

We are aware that the security of the system can be compromised anywhere so we include various checks on data, requests, and services. Components such as the User Input Data Validation, Result Data Validation, and Service Validation will ensure the system can detect potential issues.

The Front-End Business Rules provide the security necessary for the Front-End, while Back-End Business Rules address the security of traffic originating from the Service layer. We prevent improper or sensitive data from being easily transferable with the Data Access layer. Service Validation guarantees the requested service was fulfilled so a different service does not interfere with the response. User Authentication is also enforced the Business layer to verify a user requesting access to information is a user logged in the system.

6. Error Handling

6.1 Client

The errors that appear on the client-side may propagate from several locations in the system. Most Front-end errors will be thrown from the Business layer that will check for improper user input or if user input satisfies business logic. The user should be notified of errors with a user-friendly message describing the cause of the error to distinguish whether the error is from user input or server issues. However, if the error is from the Business layer a solution should accompany the error message.

Most errors can be caught in the Business layer and should be handled immediately. Any errors from the Server or Service layer do not appear on the Client-side unless a particular component in the path of a request is not responding appropriately. Server errors will appear more frequently on the Client-side than service errors because an error from a service will not appear until that particular service is requested. Whereas the server communicates with the client constantly and, therefore, increases the chances of an error appearing.

6.2 Server

Server errors will be shown to the UI only when information within the database is corrupted or simply does not exist within both internal and external data sources. Although all prompted errors to the will extend from client side errors (i.e. such as user login input errors or invalid VIN number entries), not all errors will be displayed to the user as only ones that interrupt user functionality will only be displayed. All of the server errors will be within our external API's and our RDBMS as there can exist corruption between lines of code during edits.

We plan on managing most backend errors through code reviews because most relational database information will not be accessible to our users. Although most errors within the server side and service layer will be handled through code reviews during testing and errors that happen through the UI that are dependent on backend layers will ultimately be prompted to the user itself. As far as those minute server errors, most errors will ultimately have to be fixed during the coding process after application releases.

Appendix

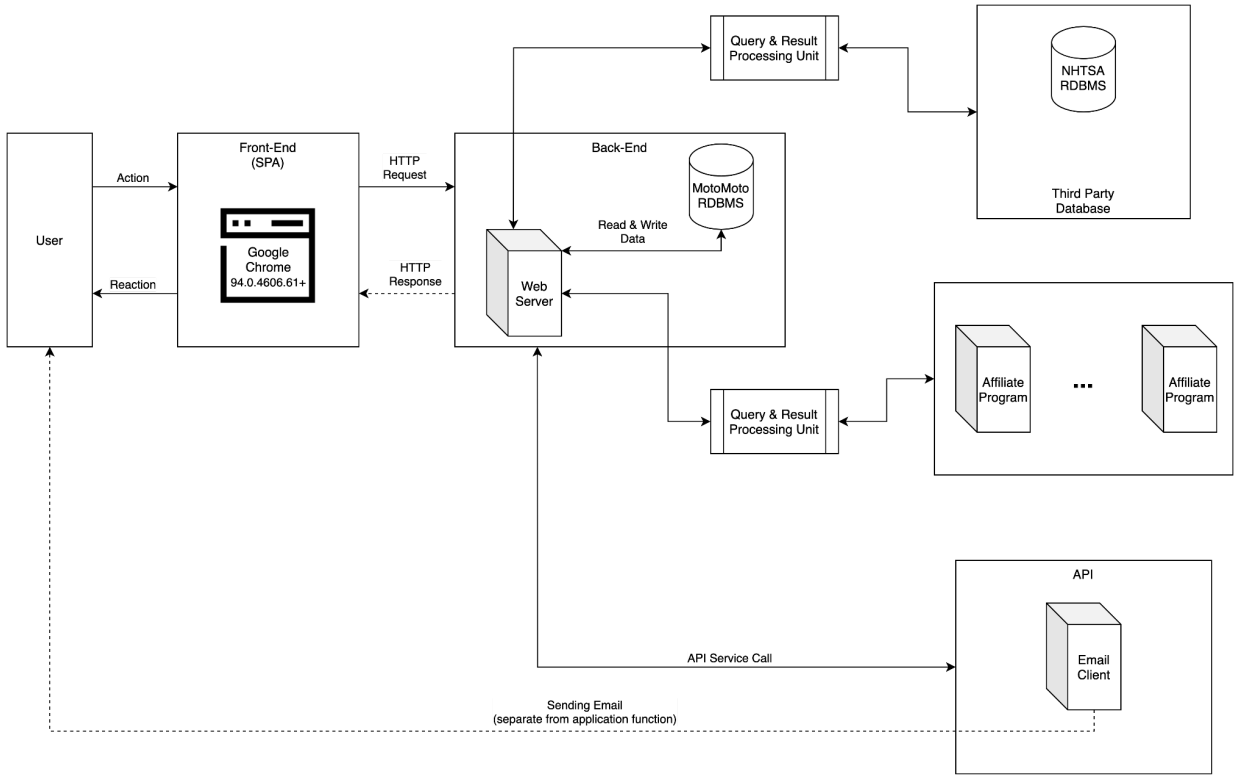

Figure 1: Hardware Architecture Diagram

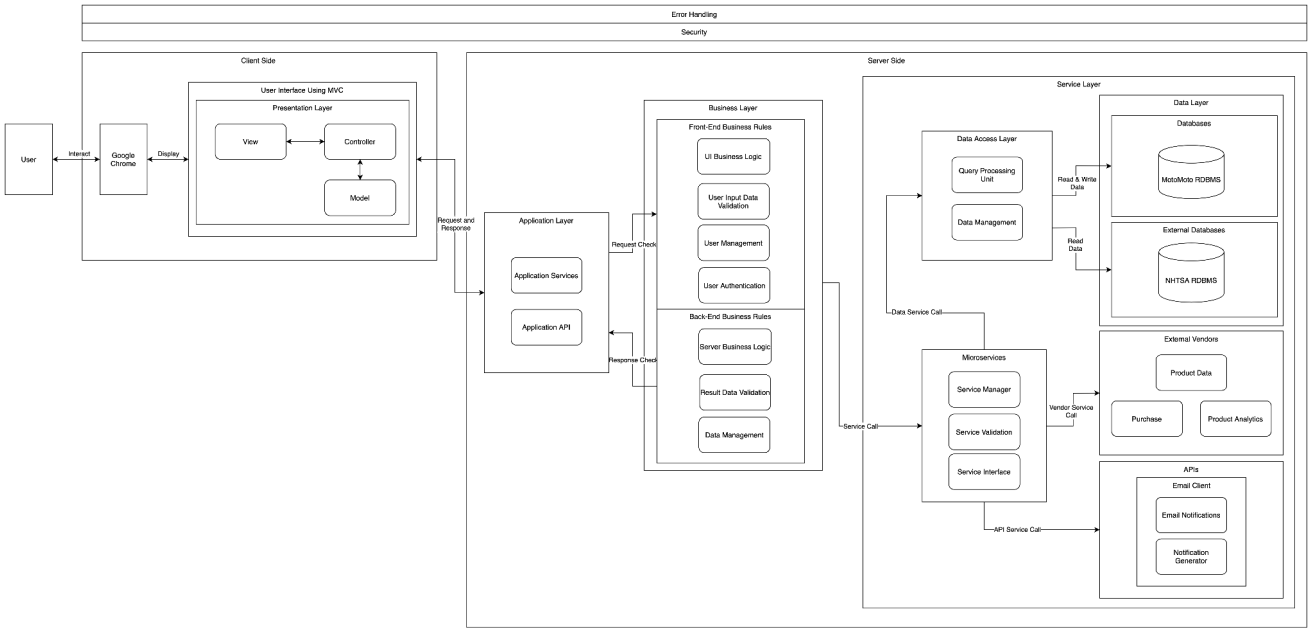

Figure 2: Software Architecture Diagram