High Level Design
MotoMoto

The New Panelists
10/20/2021

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,
- Describe Hardware and Software designs and clarify the components within, and
- Explain the behavior of the system.

1.3 Terms

- MotoMoto : Name of the application
- HLD : High Level Design
- RDBMS : Relational Database Management System
- NHTSA : National Highway Traffic Safety Administration (US Government)
- UI : User Interface
- MVC : Model, View, Controller Architecture
- SPA : Single Page Application
- API : Application Programming Interface
- REST : Representational State Transfer
- HTTP : Hypertext Transfer Protocol
- VIN : Vehicle Identification Number

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 (Pg. 8), 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 will be sent to the Back-End for all complex business logic where the web server processes requests and connects them to the proper resources.

3.1.2 Back-End

The Back-End will respond to all requests that have dependencies and, depending on the request, the web server will verify data being transferred back to the Front-End or supply external services from Affiliate Programs and 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 Presentation layer applies 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 which will become the foundation to maintaining product availability.

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 some basic logic in the

Controller in the Presentation layer as it will help with the speed of information displayed on a screen whereas all other business logic will be handled in the Back-End portion of the application's architecture.

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 communicate with the Business Rules ensuring that all applicable application services can be used during the use of our program. In Figure 2 (Pg. 8) 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 services are communicated back to the users screen. Essentially, the Application layer will act as the translator between the Presentation and Business layers.

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 Service layer (supplying services upon request). This layer will solve all of the application's real world business logic before information will be stored, created, deleted, modified, or provided. We have split up the Business layer shown in Figure 2 (Pg. 8) into two sections: Front-End and Back-End Business Rules where Front End Business Rules will contain the evaluations of user requests that are transferred to the Service layer to be satisfied, given the request is valid; and Back End Business Rules will contain the evaluations of responses (data or services) that are transferred to the user, given the response is valid. If a request, response, or any logical procedure is not valid then the error should be handled, see Error Handling (Pg. 5).

3.2.4 Microservices

The Back-End architecture is service-oriented and applies microservices particularly. Microservices that are currently offered are Data, Vendor, and API services and the Service layer will manage, validate, and direct requests to these services. We compensate the Data layer with additional logic by allowing the Service Manager to communicate with the Data Access layer to ensure proper requests and responses are being transferred to and from the Data layer. 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. External APIs will fulfill services we cannot do on our own, currently this service is an Email Client.

3.3 Network Architecture

The diagram, Figure 3 (Pg. 9), presents the general structure of our Network Architecture and is subject to change. The architecture is modeled to be RESTful and will be using HTTPS for network communication. Our goal is to gain a TLS certificate for the website.

The first upper portion of the image you can see the client using any means from their computer to connect to our website through the internet. Our website will be protected from any potential DDoS attacks thanks to AWS Shield Protective Service. Once the user's request passes the AWS Shield, then the request will go through the reverse proxy to then be processed through the elastic load balancer. Both are also provided by NGINX.

Once the client has entered through the load balancer, they are now inside our AWS VPC and directed towards any of our three public web servers. There is no extra firewall protecting these web servers, which makes them publicly accessible for users.

Otherwise, the HTTPS request will access our private network. First it must be cleared through our internal firewall then the request will be inside our private network.

In our private network the HTTPS request can access MotoMoto server's database.

4. System Behavior

4.1 Security

We are aware that the security of the system can be compromised anywhere so we include various checks on data, requests, responses, 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 Back-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 in the Business layer to verify a user requesting access to information is a user logged in the system.

4.2 Error Handling

Normally each error will display to the user's screen, but depending on the error, the information provided should be vague or not be provided. Since most Client-side errors will be caused by invalid input determined either in the Client-side or Business layer on the Server-side, there will be a more descriptive explanation of these errors than the errors caused by server errors to reduce system vulnerability exposure.

4.2.1 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. Contents may vary depending on the error source, mainly for Server-side errors they will be more vague than the Client-side error to reduce any security risk. However, if the error is from the Front-End Business Rules a solution should accompany the error message.

Most errors can be caught in the Business layer and should be handled immediately. Depending on the severity of error logged, most errors from the Server or Service layer may not appear on the Client-side unless a particular component in the path of a request is not responding appropriately which then the user should be informed that their request did not process correctly. Server errors may 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 if a server error were to occur .

4.2.2 Server

Server errors will be shown to the UI when a request is not correctly satisfied. Errors may include, but are not limited to, incorrect result data, failed service, and invalid information. Information that did not successfully pass Data Validation should be logged and depending on the severity and type of information can be presented to the user. For example, a service failure would only display for the user explaining that the request could not be satisfied and to try again instead of where in the system the failure occurred and what the specific cause of error was. Although all prompted errors will extend from Client-side errors (i.e. 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 with simple information.

Although all errors within the server side will most likely be database related, Server-side errors caused by Client-side input will be reported and sent back to the user if there exists invalid input. All other errors will be logged where instances of the service may be replaced with another deployment of that same service or if the response is corrupted the request can be sent again. Service validation will guarantee these errors are caught if a service was not adequately satisfied or not correct in general.

4.3 Logging

Data Logs will allow our developers to get insight on all key events regarding Server-side errors. Logged records may be about a failed service, query or result failures, or anything disrupted in the process of logical procedures. The Logging system

will allow for our system to have high availability due to it working without monitorization, minimal data loss which will be logged when errors occur, and horizontal scalability for additional components.

Components working in tandem to support the Logging system are mainly the Business, Service, and Data Access layers. Since most of our Server-side errors can be realized in these areas, the Application layer will transfer responses to the Client-side appropriately using log severity to distinguish which errors to send and how errors are presented to the user.

Appendix

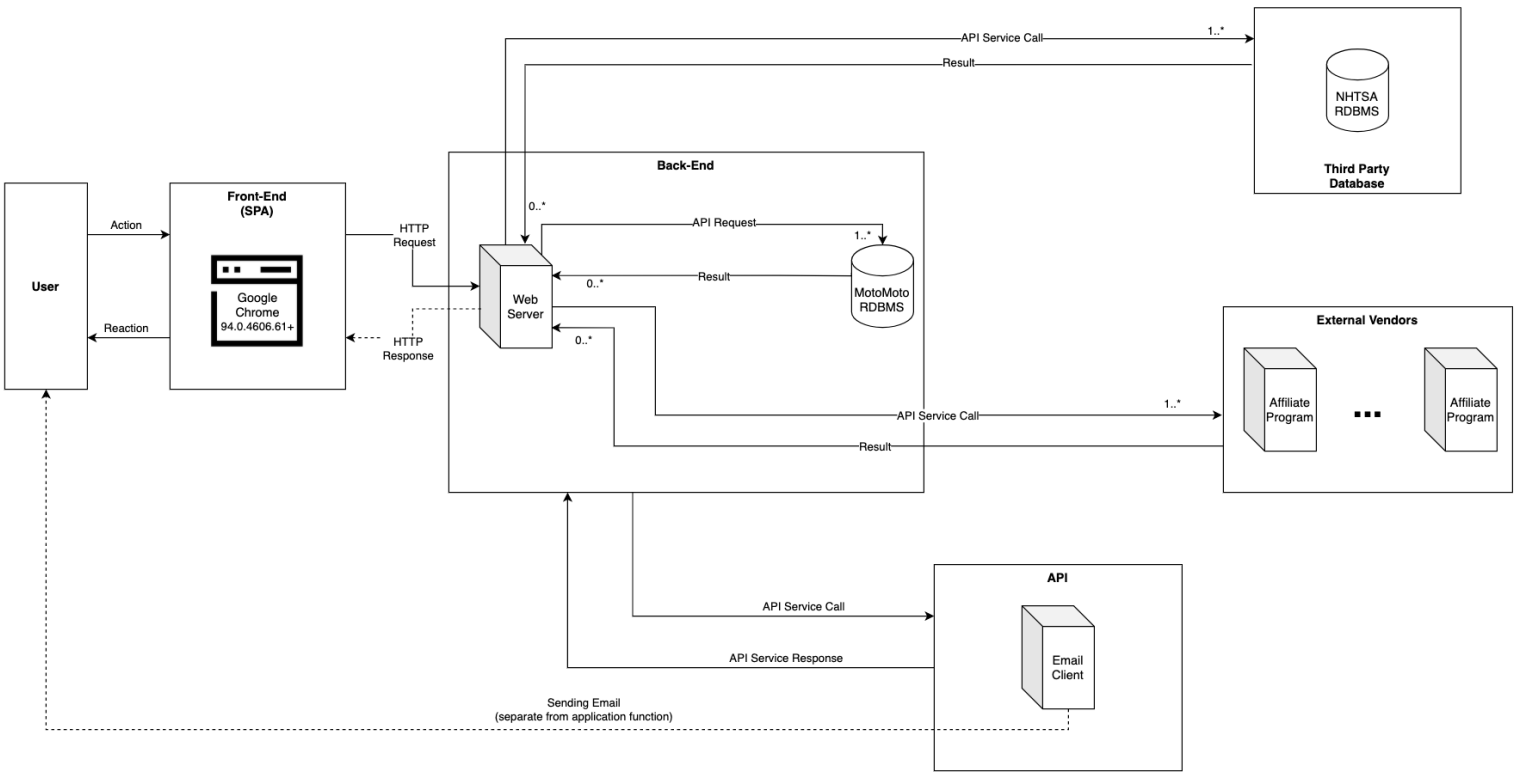

Figure 1: Hardware Architecture Diagram

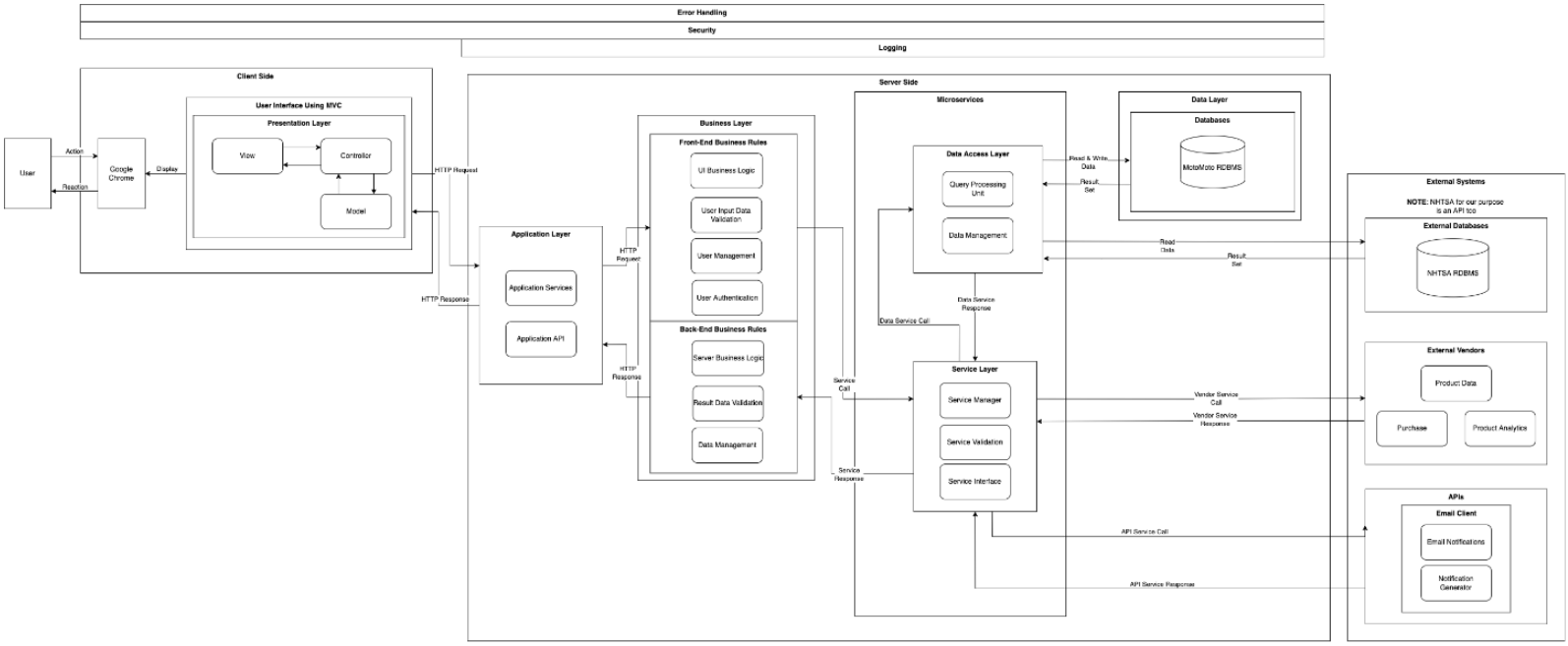

Figure 2: Software Architecture Diagram

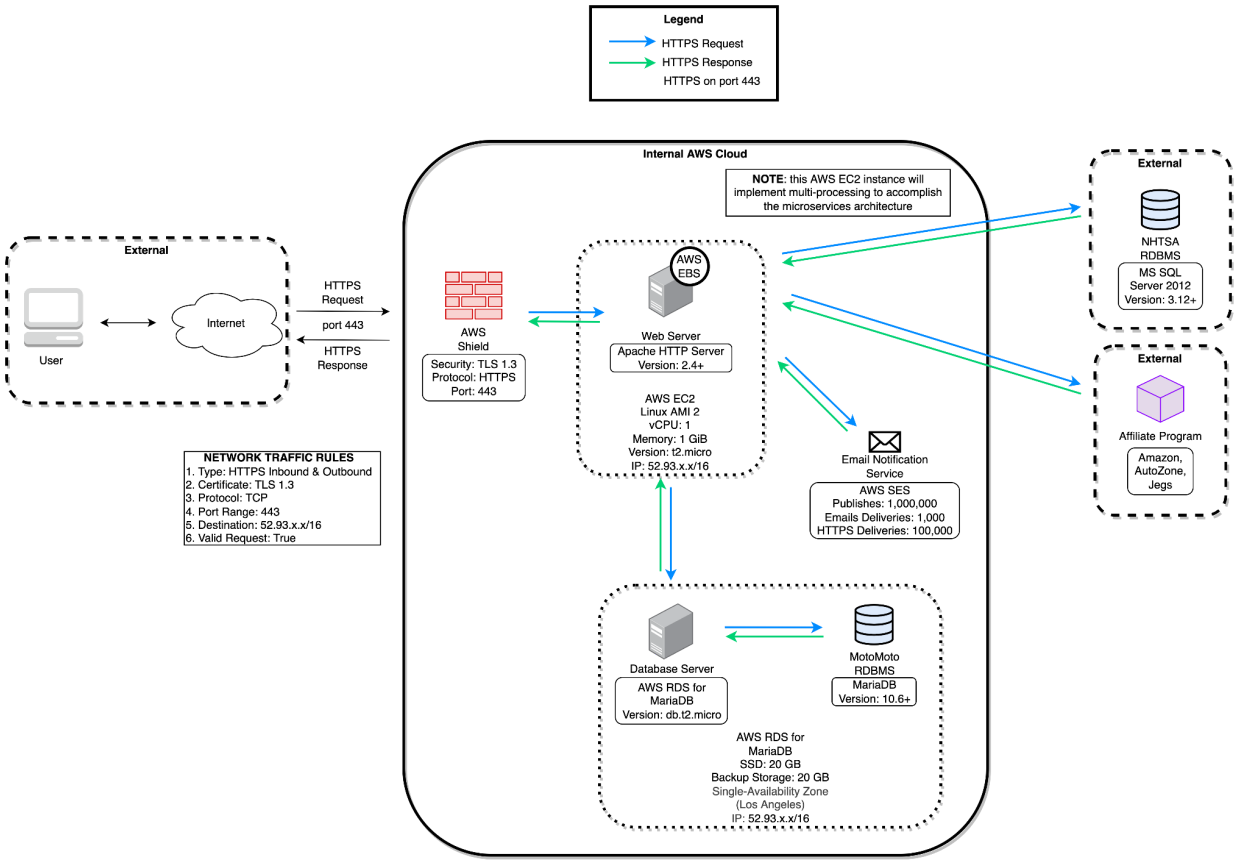

Figure 3: Network Architecture Diagram