



# API Design Fostering Payment Methods

Elmond Moyo<sup>#1</sup>; Trevor Makausi<sup>#2</sup>; Wellington S Manjoro<sup>#3</sup>; Wellington Makondo<sup>#4</sup>; Chiedza Hwata<sup>#5</sup>

<sup>#1</sup>Dept. of Software Engineering, Harare Institute of Technology, Harare, Zimbabwe

<sup>#2</sup>Dept. of Computer Science, Harare Institute of Technology, Harare, Zimbabwe

<sup>#3</sup>Dept. of Software Engineering, Harare Institute of Technology, Harare, Zimbabwe

<sup>#4</sup>Dept. of Software Engineering, Harare Institute of Technology, Harare, Zimbabwe

<sup>#5</sup>Dept. of Information Technology, Harare Institute of Technology, Harare, Zimbabwe

[wellersy@gmail.com](mailto:wellersy@gmail.com)

DOI: 10.47760/ijcsmc.2020.v09i12.002

**Abstract-** *The design of API patterns has grown exponentially over the past few years as compared to the old age meaning that the development of APIs has become of essence than back in the days. This is a research project that will look in the field of the Application Programming Interfaces (API) and be able to apply the techniques used, to the main functionality of the development of car hire management systems in Zimbabwe and abroad. The author is going to research mainly about REST APIs, the integration of payment methods so that at the end a theoretical application can be used to come up with a prototype of the API that can be used in above mentioned field. Integrated in the API are various payment methods such as (Ecocash, Telecash) and ZIMRA tax module. Mainly about REST APIs and integration of payment methods with a view of coming up with comprehensive and easy to use APIs [3].*

**KEYWORDS**—APIs, RESTFUL, SOAP, JSON, Pragmatic, Versioning

## I. INTRODUCTION

The research project is mainly focused on Application Programmable Interfaces that are being used with the major aim of coming up with a defined API pattern fostering payment methods and revenue collection for car hire companies. Application Programmable Interface (API) is a set of protocols and routines as well as the tools that are used to build software applications. It specifies the interaction between components in a software. APIs have become the most fascinating section of programming. The fact that they address programmer challenges rather than customer requirements makes it both fascinating and novel. As the software trade and the open-source effort continually grows, the number of public APIs is also gradually increasing. The good thing about APIs is that they can improve the development speed thus their popularity. They also contribute a standardized and higher quality software and at the same time increasing the reusability of software. APIs are designed to be used by programmers and most of the programmers are finding it easy to use APIs rather than developing their own program right from scratch. Hence this paper is aimed at reviewing the critical points of the current knowledge about APIs. Generally, APIs have a number of classifications which are based on what purpose they serve and/or how they work. There are internal APIs which are used strictly within the organization or firm, there are also external APIs which are specifically designed externally for customers.

The most important question that a developer should ask themselves whenever they plan on designing an API is what are they trying to achieve with that API, this will help to stay focus on the design and not end up getting carried away because the main purpose of designing APIs is to make the programmer as successful as possible in the least time given.

The orientation for APIs is to think about design choices coming from the application developer's opinion. The primary design principle when crafting your API should be to capitalize on developer's productivity and accomplishment, hence it is called the "Pragmatic Rest.

## II. RELATED WORK

CarQuery API [13] is a JSON based API meant for retrieving a detailed car and truck information which include the year, make, the model, trim, and also its specifications. The CarQuery API is so simple to use such that it can be equated to including a JavaScript file, and as well as inserting a few lines of code in your page. There is room for the developer to write their own JavaScript or a server-side API interaction.

Fuel API [14]. This is an API that was developed around the principles of REST. Allows one to search, download and save images and videos of vehicles, interior and exterior features.

The Locu Data API [15] provides real-time access to Locu's dataset of semantically-annotated venue and price list information. The API enables developers with the ability to make location based queries for detailed venue, price list, and price list item information. The Locu Data is divided into two APIs that is the API for matching a third party venues to Locu's, meaning that partners are permitted to send in their venues and in return will information that Locu knows about the venue. The search API supports the fetching of venues by multiple types of IDs, the date added or changed, looking for menus. The Locu API has what they call the venue object which is the core object for matching constructed venues by partners, {"fields": ["location", "menus"]}, this venues object searches for the location details and also menus which are found in that venue. It supports the CURL and JSON formats.

The Carvoyant API [16]. This is a RESTful web API which allows customers and the partners to access some Carvoyant data programmatically. All the API calls must be made over the SSL and the access to the system is controlled by an authentication process OAuth2 implementation. To access the API you use the URI: <https://api.carvoyant.com/v1/api>. All the requests are expected to include Authorization header that contains the bearer token that has been generated through the OAuth2 grant types. With APIs everything is a resource hence the API is structured in terms of resources. The various requests that can be made against a resource include PUT, DELETE, GET, and POST. It supports JSON format. It has the capabilities of giving a JSON success response format distinct from a JSON error format.

## III. METHODOLOGY

The author started with a very simple design for the solution which allowed more extensions and additions to be applied during stages of development. The design mainly dealt with views on how users will be interacting with the solution [8]. The author mainly used four controller patterns which allow him to split the solution into different parts so that it will be easier to change parts of the solution without affecting other parts of it. Model refers to methods for accessing and modifying states whereas Views are those that renders contents of model for user and Controllers translate user actions (interactions with the view) into operations on the model for example button clicks and menu selection. The author saw it important to use Model-View-Controller design pattern as it allowed him to separate major components that can be worked on in relative isolation. Therefore, the author mainly started with the design of different views on how he would like users to interact with the solution [1].

Later other additions were made to the design to add more functionality and ease of use with regards to the API and during designing of the solution views, the author decided to name the solution API as Morthpark API.

The system architecture is the conceptual model that defines the structure, behaviour and more views of a system. An architecture description is a formal description and representation of a system. The system was represented following the Model View controller design pattern. [5] The pattern's title is a collation of its three core parts: Model, View, and Controller.

A visual representation of a complete and correct MVC pattern looks like the following diagram:

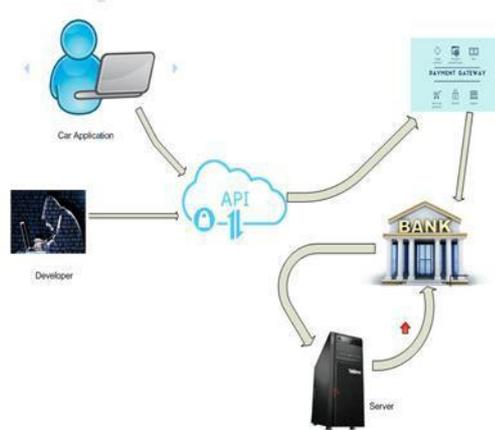


Fig .1. System Architectural Design

The diagram shows the flow layout of data, how it is passed within each component and how the relationship within each component works.

#### A. MODEL

The Model is the name given to the permanent storage of the data used in the overall design. It must allow access for the data to be viewed, or collected and written to, and is the bridge between the View component and the Controller component in the overall pattern.

One important aspect of the Model is that it's technically "blind" – by this we mean the model has no connection or knowledge of what happens to the data when it is passed to the View or Controller components. It neither calls nor seeks a response from the other parts; its sole purpose is to process data into its permanent storage or seek and prepare data to be passed along to the other parts. [2]

The Model, however, cannot simply be summed up as a database, or a gateway to another system which handles the data process. The Model must act as a gatekeeper to the data itself, asking no questions but accepting all requests which comes its way. Often the most complex part of the MVC system, the Model component is also the pinnacle of the whole system since without it there isn't a connection between the Controller and the View.

#### B. VIEW

The View is where data, requested from the Model, is viewed and its final output is determined. Traditionally in web apps built using MVC, the View is the part of the system where the HTML is generated and displayed. [3] The View also ignites reactions from the user, who then goes on to interact with the Controller. The basic example of this is a button generated by a View, which a user clicks and triggers an action in the Controller.

#### C. CONTROLLER

The final component of the triad is the Controller. Its job is to handle data that the user inputs or submits, and update the Model accordingly. The Controller's life blood is the user; without user interactions, the Controller has no purpose. It is the only part of the pattern the user should be interacting with.

The Controller can be summed up simply as a collector of information, which then passes it on to the Model to be organized for storage, and does not contain any logic other than that needed to collect the input. The Controller is also only connected to a single View and to a single Model, making it a one way data flow system, with handshakes and signoffs at each point of data exchange. [5]

It's important to remember the Controller is only given tasks to perform when the user interacts with the View first, and that each Controller function is a trigger, set off by the user's interaction with the View. The most common mistake made by developers is confusing the Controller for a gateway, and ultimately assigning it functions and responsibilities that the View should have (this is normally a result of the same developer confusing the View component simply as a template). Additionally, it's a common mistake to give the Controller functions that give it the sole responsibility of crunching, passing,

and processing data from the Model to the View, whereas in the MVC pattern this relationship should be kept between the Model and the View. [9].

#### D. USE CASE DIAGRAM

Below is a use case diagram depicting use of the developed system. The system has two users namely the administrator and the cost engineer. Figure 4 basically shows how these two users will be manipulating this systems various functionality.

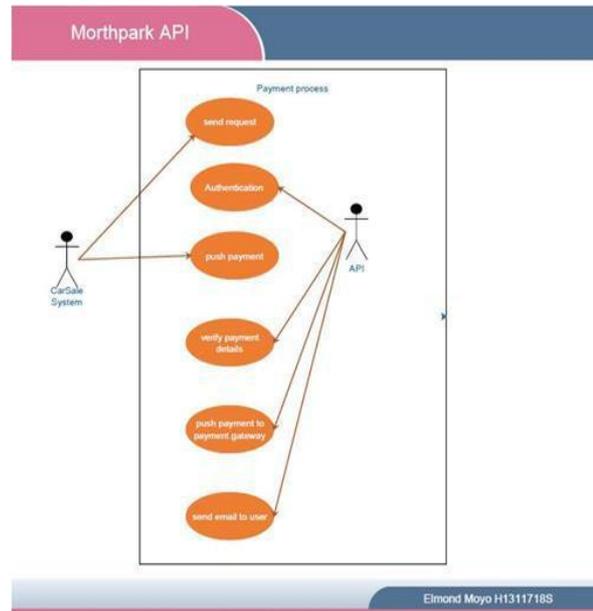


Fig.2. Use case diagram for the system

E. *IMPLEMENTATION AND TOOLS* The development tools that were used for the development of Morthpark API consisted of the following

##### 1) JAVASCRIPT

JavaScript is the client-side scripting language used during the development phase. In certain circumstances, JavaScript had to be used instead of PHP. One such instance includes displaying popup windows to alert users of errors in data entered during validation of input or possible loss of data when a delete operation is executed [8]. The fact that the language works in a run-time environment which is especially true in web browsers, makes JavaScript suitable for offering the required services that a server-side language cannot. However, JavaScript is not used to pass sensitive data such as passwords in view that the codes are executed at the client side where the system can be vulnerable to malicious attacks [1].

##### 2) HTML

HTML, acronym for Hypertext Markup Language, is a markup language used to describe the formatting of text in a document. It is useful in the sense that it allows text to be structured according to its purpose, namely as a heading, paragraph and so on. This is accomplished by writing the HTML in 'tags' that describes to the web browser how the text is to be displayed. A scripting language such as PHP and JavaScript can be easily embedded in HTML to enhance the functionality of HTML [11].

##### 3) SMWEBCREATE

SMWEBCREATE is short for Sadomba-Mahari Webcreate which is a platform for helping in the design and development of websites. The functionality of SMWEBCREATE is similar to that of ADOBE DREAMWEAVER where graphics and web components can be dragged and dropped to the appropriate location at the same time it automatically generate the code where additional code manipulation can be done by the programmer [10].

##### 4) DEVELOPMENT PLATFORM

The windows 8 operating system was selected as platform for developing MORTHYPARK API. The fact that the potential users are already using and are familiar with the Windows 8 environment played an important role in the selection [8].

## 5) DATABASE

MySQL is a relational database management system (also known as an SQL Database Server) which is widely used around the globe due to it being open-sourced. Most SQL servers provide reliability but not ease of use unlike MySQL

[6]. MySQL is also mostly platform independent which means it can run on most operating systems such as Windows and Linux [6].

## FUTURE WORK

The system can be further developed to cover a wide domain range not only focus on car sale systems. The system can be further developed to generate API codes to be integrated with the car sale system as another payment method.

## CONCLUSION

In a nutshell, the system design and implementation has been a success as all the objectives have been met despite the system developer encountering some challenges in the integration of various payments methods in the system.

## ACKNOWLEDGEMENTS

First and for most I would like to acknowledge my supervisors Mr. W Manjoro et.al for his guidance throughout this work and sharing some insight on my research which came useful to the project. You are the man sir! My heartfelt thanks. I would also like to extend my acknowledgements to the department of Software Engineering, Harare Institute of Technology for the support they rendered for the success of this API design. My heartfelt thanks. The success of this project could not have been accomplished without the support of my brother Mthokozisi Nyoni. You deserve the best in life! My heartfelt thanks. Finally to my caring and loving parents Edgar Moyo and Vonesai Moyo; my deepest gratitude. Your encouragement when times got rough are much appreciated and duly noted! My heartfelt thanks.

# REFERENCES

- [1]. Li, L., Chou, W., Zhou, W., & Luo, M. "Design patterns and extensibility of rest api for networking applications". IEEE Transactions on Network and Service Management, 2016.
- [2]. Ellis B., Stylos, J., & Myers, B. "The factory pattern in API design: A usability evaluation". Proceedings - International Conference on Software Engineering, 2007.
- [3]. Mulloy, B. "Web API Design – Crafting Interfaces that Developers Love", 36. 2012
- [4]. Stylos, J., Myers, B., Stylos, J., & Myers, B. "Mapping the Space of API Design Decisions". 2007
- [5]. Woods, D., Thurai, A., Dournaee, B., & Musser, J. "Enterprise-class API Patterns for Cloud & Mobile", (May), 15. 2012
- [6]. Zhou, W., Li, L., Luo, M., & Chou, W. "REST API design patterns for SDN northbound API". Proceedings - 2014 IEEE 28th International Conference on Advanced information Networking and Applications Workshops, IEEE WAINA 2014.
- [7]. A. Meiappane, J. Prabavadhi & V. Prasanna Venkatesan. "Strategy
- [8]. Pattern : Payment Pattern". (2012).
- [9]. Tak, P., & Tang, P. DFTI "A New Interface for Fast Fourier Transform Libraries", 31(4), 475–507. (2005).
- [10]. Santhiar, A., Pandita, O., & Kanade, "A. Mining Unit Tests for Discovery and Migration of Math APIs", 24(1). 2014
- [11]. Reed, J. A., & John, N. "Improving the Aircraft Design Process Using Web-Based Modeling and Simulation", 10(1), 58–83. (2000)
- [12]. Roy Thomas Fielding, "Architectural Styles and the Design of Network-based Software Architectures". 2000
- [13]. <http://carvoyant-api.readthedocs.io/en/latest/getting-started>
- [14]. [www.carqueryapi.com/](http://www.carqueryapi.com/)
- [15]. [fuelapi.com/](http://fuelapi.com/)
- [16]. <https://dev.locu.com/>
- [17]. <https://carvoyant-api.readthedocs.io/>