The Effect of MVC Architecture on Nocode and Lowcode Tools (using BackboneJs and GrapesJs as an example)

Ugochi Ukpai
Babcock University

The Effect of MVC Architecture on Nocode and Lowcode Tools.

(using BackboneJs and GrapesJs as an example).

Ukpai Ugochi Ibem
Babcock University Ilishan-Remo, Nigeria
ukpai0278@pg.babcock.edu.ng

Abstract

Several studies have explored the effect of the Model-View-Controller (MVC) architecture in the creation of web applications; however, not much thought has been given to the effect of the MVC architecture in no-code and low-code tools. Usually, the MVC architecture is used for code settings that require rapid logic scalability and customization. This is possible because software development, especially web development is constantly evolving. However, since no-code and low-code development is still in its infancy, most of its logic is written in “spaghetti format”.

In this paper, we present the MVC architecture and how it allows the creation of efficient no-code and low-code tools, using Backbonejs and GrapesJs as an example. This paper focuses on the MVC architecture Backbonejs provides for GrapesJs, a no-code/low-code tool. The MVC architecture provides not only the separation of the Model, View, and Controller logic but also security, validation, and other components.

Contrary to the initial templating setup used by no-code and low-code tools which results in “spaghetti code”; The load time and the ability to create correct and easy-to-maintain code is the main objective of this research.

Keywords**:** Low-code, No-code, software development, MVC architecture, GrapesJs, BackboneJs, JavaScript.

Introduction

No-code and low-code tools have come a long way since the creation of Excel in 1985 (IvyPanda, 2021). Now, not only do no-code/low-code tools simplify day-to-day tasks, but some tools allow you to build a functional web or mobile application (Liu et al., 2022). However with the constant evolution of the World Wide Web, the creation of multiple frameworks and languages to build web applications, no-code is playing catch up. This problem has led to the sprouting of different standards when building no-code/low-code tools; many of them are compact, unreadable, and difficult to maintain.

Today, the web presents data to users with HTML, CSS, and JS. For efficiency, many frameworks and libraries present "scaffolding" applications that separate logic using the MVC framework. GrapesJs a no-code tool, is using one of this framework Backbonejs.

The Problem

While the World Wide Web is constantly evolving, there is a need for web developers to evolve too. This has led to the creation of multiple frameworks and languages; developers can start their logic in a pre-defined template, instead of working from scratch. While this is a lot of change in the software industry, there is still a wait time between developers and engineers. The designer has to design the product while moving across multiple iterations with the product manager and developer. To bridge this wait time, developers have created a bunch of no-code/low-code tools. This way, there is no boundary between developers and business experts (Sanchis et al., 2019).

However, while still in its infancy state, these tools were built in a ‘spaghetti’ state, making them very difficult to maintain. Most of them are structured in their own languages or frameworks, making them unadaptable to the traditional environment. For progressive no-code/low-code applications like GrapesJs that are structured with MVC architecture, this is different.

Literature Review

No-code and low-code tools and applications have been at the center of the software industry since the creation of Excel in 1985. Lately, there has been some research on no-code/low-code tools in software development (Marcus, 2020). These studies also highlight how difficult it is to scale and maintain applications built by no-code/low-code tools, the possibility of vendor lock-in, and limited flexibility (Zhaohang, 2021).

However, with the constant evolution of web technologies, studies have shown the adoption and skill difference between citizen and professional developers. Further studies have shown how recent nocode applications allow customization and minimize vendor lock by creating plugins to leverage other systems (Giorgia, 2023).

While these studies present valid limitations of no-code/low-code tools, some studies also present advantages, especially as they relate to serverless applications (). However, these studies only highlight how no-code tools bridge the gap between IT and business experts, giving little or no thought to code quality and maintainability.

While there has been a lot of research on MVP and MVMM architecture (Akhmad, 2023), MVC is one of the most popular architecture patterns and is valid in organizing programming logic. The impact of the MVC architecture in the software development field has been well-researched. Although some research findings have explored frameworks that design desktop applications using the MVC framework (Sarker et al., 2014).

Some research findings have also explored frameworks that design and transform legacy web applications into the MVC framework (Kontogiannis et al., 2003). Such research shows that the MVC architecture can be used in all rounds of software development (Web, desktop, CLIs, etc.). However, for the purpose of this research, we will be focusing on the effect of MVC in nocode tools, especially web applications.

One limitation of the past studies is finding the relationship between the MVC architecture, frameworks that use the MVC frameworks, and no-code/low-code tools. This means that researchers do not have much insight into how the MVC architecture can affect no-code and low-code tools.
Future research could, for instance, investigate how this architecture can transform no-code/low-code tools, to create readable and easily managed code.

GrapesJs is an open-source user interface (UI) builder that allows developers to create no-code tools for websites. While GrapesJs is technically not a no-code tool, it is a library used to design and structure no-code/low-code tools (website builders). At the heart of GrapesJs is BackboneJS, a library that scaffolds applications using the MVC library. This study determines the effect of the MVC architecture in no-code/low-code tools, by carefully studying how GrapesJs uses BackboneJs for structuring, DOM manipulation, and creating readable code. It also evaluates the structure BackboneJS provides for GrapesJs and how GrapesJs creates maintainable, easily readable, and customizable code.

Results and Discussion

BackboneJs provides structure to applications. It does this by providing models; binding them with key-value and custom events, and collections; which provide a rich API of enumerable functions and views with declarative event handling. Previous studies on BackboneJs are mainly about single-page web Applications (SPAs) (Azat, 2018). However, the use of BackboneJs goes beyond web applications. For example, it can be used in other areas of software development like nocode or building command line interfaces (CLI).

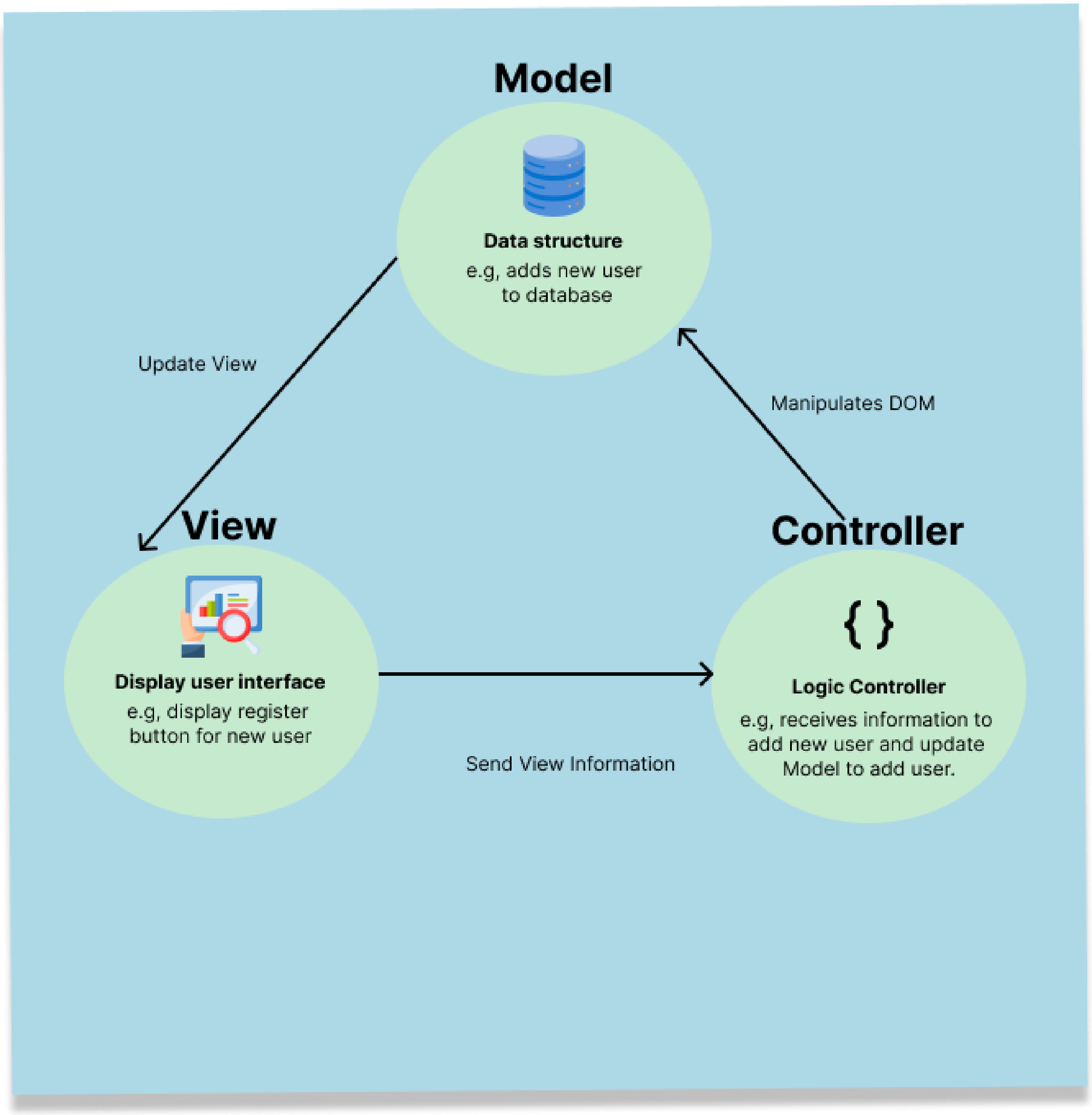
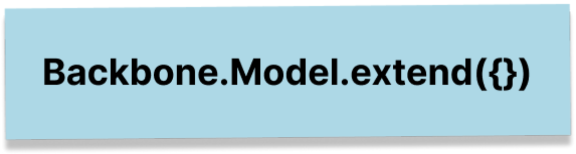

Model
Data structure
e.g, adds new user to database
Update View
Manipulates DOM
View
Controller
Display user interface
e.g, display register button for new user
Logic Controller
e.g, receives information to add new user and update Model to add user.
Send View Information

Figure 1. Overview of MVC Structure.

Model

The Model is the heart of the MVC framework. It is in charge of the business logic, database logic, validation logic, and all of the application logic that is not included in the `Views`. To create a Model in BackboneJs, you need to extend your domain-specific methods (Robert et al, 2005), Thereafter, the Model provides a basic set of functionality for managing changes. To create a model, you will extend the Model like this;



```
Backbone.Model.extend({})
```

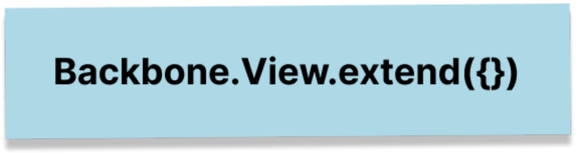
Figure 2. Creating a Model

An example of this usage is how GrapesJs exports the Model it extends here. This export is used in the Model module and also runs the editor and how it should be rendered. Thereby controlling and structuring the HTML, style, components, and other logic displayed to the user.

View

The Views don't control how your HTML and CSS are displayed. However, they help keep the structure of your application's Views into logical views. The purpose of the View component is to allow independent updating of the application's View after the Model update without the need to rewrite the whole view. For GrapesJs, we do not want to redraw the Panel and other default views whenever the user updates their design in the editor.

Just like the Model, to create a custom View you will need to create a custom view class, extending View.



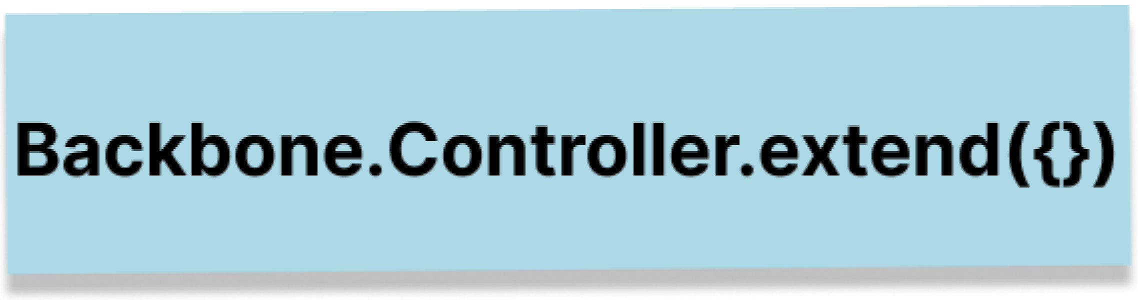
```
Backbone.View.extend({})
```

Figure 3. Creating a View

GrapesJs does this by exporting the View that'll be used in the editor.

Controller

The controller in BackboneJs can be likened to a group of Models that handles the filtering, aggregation, and sorting logic of your application's client side. When you want to effect a change across a list of closely related Models, you would want to create a Collection with those Models for ease of development.



```
Backbone.Controller.extend({})
```

Figure 4. Creating a Controller

In GrapesJs the Panel consists of a Collection of Models added to the Views, here's an example. This way instead of writing different logics for the Panel, we will be adding them to a Collection so that we can easily affect changes across them.

Security

Security is a difficult topic when it comes to web development especially when it comes to JavaScript. A lot of studies have explored the situation of security in JavaScript when it comes to web browsers. For example, this research (Nataliia, 2013) makes a survey on JavaScript security policies and their enforcement mechanisms in a web browser.

That said, BackboneJs solves this security issue by structuring your program in such a way that your server never sends information to a client's computer unless you want them to be able to see it. It is applied in GrapesJs in a way that the program doesn't read the data in the editor (DOM).

While security in web development is still in its infancy, MVC architecture frameworks like BackboneJs help in managing security. This way you can customize your application while reducing security breakdowns.

Customization and Logic Maintenance

BackboneJs allow for easy customization. This involves creating custom events, models, and even domain-specific methods. For GrapesJs, the customization that BackboneJs offers allows users to tailor the code generated from their editor to suit their needs.

Therefore, unlike other nocode tools that create web applications using new languages and limiting customization, GrapesJs' code can be built into already existing frameworks. This way, customization is easier.

While customization is an important factor in software development, it is useless if the code can't be maintained. An illustration of this is working with a framework and customizing it to suit your preference. However, writing "spaghetti" codes with the framework won't be maintainable, making the code "useless" and non-progressive.

To avoid this, BackboneJs separates the application logic using the MVC architecture. Now, the developer can work on the "logic" they want, without disrupting the other parts of the application. Hence, make the codebase readily maintainable. One way we can see this is how GrapesJs exports all architectures here allowing developers to maintain their codebase faster.

Conclusion, Limitations, and Future Research Suggestions

This study exposes the advantages that the MVC architecture provides for nocode applications. We used an MVC framework (BackboneJs) to expose how this architecture affects GrapesJs, a nocode tool. This study has potential limitations. It was limited to the MVC architecture that BackboneJs provides for a no-code tool (GrapesJs). Therefore it is subject to biases that may have influenced our studies. Our study may underestimate the gap between "citizen" - and "seasoned" developers and no-code tools. It may also be conservative in the area of customization as it relates to nocode applications.

Building on the limitations of this, future researchers can extend their research on the MVC architecture and its effect on customization and logic maintainability for no-code tools.

References

1. IvyPanda. (2021, July 24). *Microsoft Excel Overview: History, Usage, Features.* https://ivypanda.com/essays/microsoft-excel-overview-history-usage-features/
2. Liu, S., La, H., Willms, A., & Rhodes, R. E. (2022). A "No-Code" App Design Platform for Mobile Health Research: Development and Usability Study. *JMIR Formative Research*, *6*(8). https://doi.org/10.2196/38737
3. Sanchis, Raquel & García-Perales, & Fraile, Francisco & Poler,. (2019). Low-Code as Enabler of Digital Transformation in Manufacturing Industry. Applied Sciences. 10. 12. 10.3390/app10010012.
4. Woo, Marcus. (2020). The Rise of No/Low Code Software Development—No Experience Needed?. Engineering. 6. 10.1016/j.eng.2020.07.007.
5. Yan, Zhaohang. (2021). The Impacts of Low/No-Code Development on Digital Transformation and Software Development.
6. Masili, Giorgia. (2023). NO-CODE DEVELOPMENT PLATFORMS: BREAKING THE BOUNDARIES BETWEEN IT AND BUSINESS EXPERTS. 10.14276/2285-0430.3705.
7. Analysis of Low Code-No Code Development Platforms in comparison with Traditional Development Methodologies. (2021, January 1). www.academia.edu. https://www.academia.edu/67855214/Analysis_of_Low_Code_No_Code_Development_Platforms_in_comparison_with_Traditional_Development_Methodologies
8. Zakaria, Akhmad. (2023). Android Software MVVM and MVP Architecture Analysis with iTourism App Case Study.
9. Sarker, Iqbal & Zinnah Apu, Khalid. (2014). MVC Architecture Driven Design and Implementation of Java Framework for Developing Desktop Application. International Journal of Information Technology. 7. 317-322. 10.14257/ijhit.2014.7.5.29.
10. Ping, Yu & Kontogiannis, Kostas & Lau, T.C.. (2003). Transforming legacy Web applications to the MVC architecture. 133- 142. 10.1109/STEP.2003.35.
11. Mardan, Azat. (2018). Backbone.js and Parse: Learn Backbone.js, Node.js, and MongoDB. 10.1007/978-1-4842-3718-2_5.

12. France, Robert & Rumpe, Bernhard. (2005). Domain specific modeling. Software and System Modeling. 4. 1-3. 10.1007/s10270-005-0078-1.
13. Bielova, Nataliia. (2013). Survey on JavaScript Security Policies and their Enforcement Mechanisms in a Web Browser. The Journal of Logic and Algebraic Programming. 82. 243–262. 10.1016/j.jlap.2013.05.001.