Towards a web architecture structure based on intent

Matheus Marabesi

Abstract

Web applications, specifically javascript based on virtual DOM, has growing on its adoption over time and developers are choosing those libraries to use in new projects. Such applications might also be complex based on the business requirements. Folders and files play an important role in the source code architecture. The lack of organization leads to poor understanding of the software components and how they connect with each other. Furthermore, this problem can also affect experienced developers, as such, it might take longer for code understanding and cost rising as it might require more time to implement new features or fix bugs. Thinking about this scenario, this paper proposes a web architecture based on intent and communication focused on a real web application. The goal of such structure is to provide developers with a guide, each software component has a place on the architecture and improve communication across developers. Furthermore, the architecture aims being agnostic of any framework or library, thus providing a standard across specific frameworks or libraries implementation.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Keywords: web, architecture, intent, communication

ACM Reference Format:
Matheus Marabesi. 2023. Towards a web architecture structure based on intent. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Web javascript applications based on virtual DOM [4] are used by developers every day. For each specific library or framework, there are a set of guide lines for developers to create a folder hierarchy as well as files. This approach is not restricted for web applications only, for example, server side applications have their structure, and it also applies based on the framework used.

A server side application based on the framework Laravel [13], will have a opinionated way to create folders and naming files. Laravel uses the architectural pattern MVC (Model-View-Controller [8]) and is not limited to it, it also adds functionalities on top of it. For java, springboot [17] follows the same principle, it has the guide lines in which the developer follow to agree with the framework style. The guide lines rules leads to coupling into the framework philosophy, which in turn can lead to a harder approach for software evolution. Presenting a MVC style, induces developers to use MVC across the application.

Architectural styles are used to avoid coupling in a given framework style. For example, the clean architecture [12] is used in server side applications to decouple the framework style and focus on the application domain problem. The clean architecture, is a architectural style developed with the view being an implementation detail, therefore, its foundation is not restricted to server side applications.

Thinking about this context, the rest of this paper discuss the implementation of a architectural style focused on intent, inspired by the clean architecture and for web applications. The section 2 presents the topics on the clean architecutre, the section 3 enumerates related work and concrete implementations of the clean architeture in web applcations, the section 4 discuss the web architecture conception followed by the section 5 that provides a concrete implementation of the proposed architectural style, and finally the section 6 ends the paper eliciting future works.

2 Clean architecture

The clean architecture was presented by [12] as a combination of different architectural styles [11], named: Hexagonal Architecture [2], Onion Architecture [14], Data-Context-Interaction [3] and Boundary-Control-Entity [7]. As mentioned by [11] each of those styles vay in some details, but they have the same objective: separation of concerns.

The objective of having an architecture that is clean, is to focus on the application business logic, instead of implementation details, as such, it makes the application testable from the start, ease of maintanance and evolution in the long term. The clean architecture divides the application in four layers, named:

- Entities: those are objects that represents the business.
- Use cases: Those are objects where business rules happens
- Interface Adapters: Those are the bridge to access use cases

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

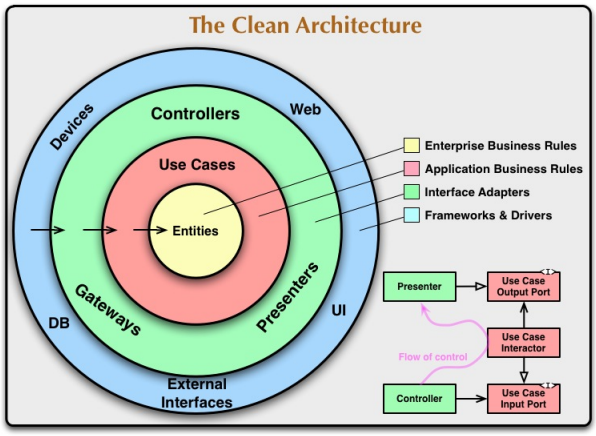

Figure 1. Clean architecture diagram [11]

- Frameworks and adapters: Those are implementation details, usually are external libraries

The conception of a clean architecture does not provide a concrete implementation in any programming language. It describes an architectural style, as such, [5] wrote his implementation and interpretation of the architecture. Still, it does not describe the details of implementing an user facing application. The further it gets is to use the MVC. The focus remains on the server side only.

This paper focus on the last layer described, the frameworks and adapters, as this is the representation of a web application. Complex Web applications require a modern architectural style, for that, using MVC might not benefit for long term and maintainability. Besides that, modern applications that use virtual DOM, brings thread offs that MVC does not address. For example, global state management.

3 Related work

As the web application space have different frameworks and libraries [6] [18] [15], the spread of different architectural styles among them is common. This section attempts to list projects that tried to follow the clean architecture in web applications regardless of the library of choice.

[1] made a transformation to bring everything defined in the clean architecture into the reacjs structure, it has an opinionated structure in the application state management and also assumes that the application will be used in the web and also in the mobile.

Other architectural examples catalogued by [16], focuses on a broader view of the overal components, and not just on one face of the architecture, for instance, the presentation layer. The layered architecture, named in his catalogue, has a layer named Presentational Layer, in which has components, still the way of how those components are structured are not concerned.

4 Conception

The clean architecture applies some principles that focus on the business rules, code maintainability and isolation of responsabilities. As such, it had to take decisions to what to take into account in the architecutre and what to leave for another time. The web application is the item in which the author names as a implementation detail, therefore, there is no definiton in how to structure it.

Given this scenario, developers tried to fit this architecture that is relavant for server side applications and fit into the web application. Such translation from server to web application does no take into account the specifities of web applications.

In the previous section for example, the extra boiler plate generated to follow the clean architecture in the web application made the overal structure more complex, with more folders and extra responsabilities. This section describes an approach to architect a web application, focused on the intent and communication.

The first step towards the definition to build a web application architecture is to acknowledge that the web application is different from server side applications, and it requires more attention. There are concerns that only web applications deal with, for example, the UI (user interface) and supporting different browser vendors. The architectural styles [12] and [16] doesn't focus on those aspects.

[12] treats the presentation aspect with three layers, View, Controller and Modal. This simplistic approach lacks different requirements that a complex application might require, such as, third party libraries, data flow and business rule.

As such, this paper aims to propose an architecture that deals with those aspects and provides an higher level of organization. There are four pilars, in which this architecture stands, name:

1. UI (user interface)
2. Third party
3. Data flow
4. Packages

Starting from the UI, this is the place to have visual components. Visual components are, pure components without complex logic. For example, a container component.

Third party is the place to wrapthrid party libraries that the application depends on, usually the third party is a dependency that is used across files and in different places. Moment is an example of third party library that falls on this category.

Data flow, is the place to have everything related to the data that the UI depends on. As such, this is the place for redux, mobx or any other library that handles data flow and has data itself.

The gotcha for this structure lies in the gray area, between components, and how they communicate. To address this

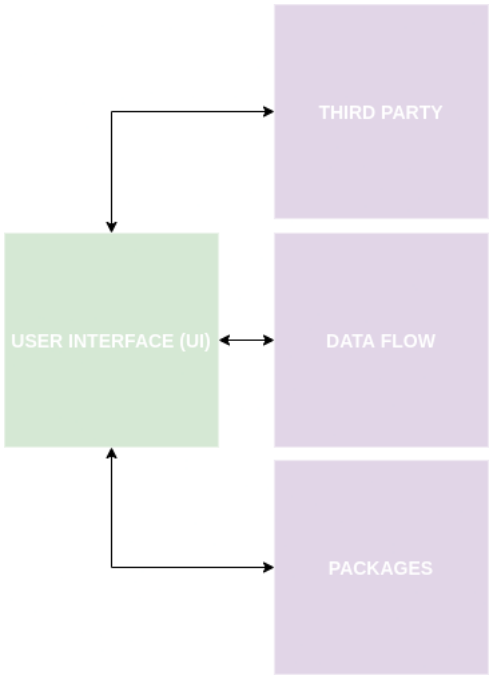

Figure 2. Layers interaction in the web application architecture.

issue I suggest to use a specific place, a specific folder, named packages.

The packages folder, in general would have a strict business rule, as such, it is the place to have, in this case, pure source code, withour interation with external libraris or data management.

There is a gray area, between components communication, as for each layer described in the architecture the inter communication will be required for a live application.

The arrows points which layers starts the communication and if the layers are connected as a two way flow or just one direction.

The gray area comes between the UI and the Data flow, as sometimes, the data flow will start from the user interface and later on the server will reply and automatically update the UI. For example, real time applications fall under the gray area. A chat application might initiate the answer through the UI, therefore, the server will automatically updates the UI to reflect the new message that arrived. The following is a folder description of organizing the folder structure following the proposed architecture:

Finally the proposed architecture brings benefits for testing purposes, as its structure forces the separation of concerns and isolates third party, improving its maintainability. Modern web applications also provide a documentation to interact with the UI components, as such, the architecture has a single place to keep its design system documetation.

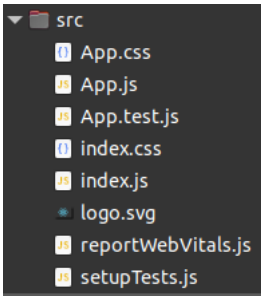

Figure 3. Default structure given by CRA

5 Implementation

This section provides a concrete example of the architectural style presented in the section 4. The project used is a gamified tool that aims to improve unit test teaching [10] [9]. The tool is built on top of web teachnologies, named: reactjs, redux, jsx, tailwind and nodejs. Besides that the tool is meant to run on a web browser.

Due the lack of adoption of a single architectural style, the tool started with the basic structure provided by the reactjs library, there is no consensus on the structure in the reactjs community, therefore, the skeleton provided by the tool CRA[1] is taken as a standard, Figure 3 depicts this default structure.

Threfore as the application became complex due the feature it required, a structured approach was needed to keep and evolve the application, as such, the implementation used was the structure discussed in the section 4.

6 Future work

This paper presented a web application architecture aimed on intent and communication, taking into consideration what the clean architecture and others architectural style lacks into details and definition for web applications, as well as existing broader architectural styles. This paper also discussed the different web application frameworks, and for each there is a architecture already in place, but none of them are unified.

This paper proposes an architectural style across libraries and frameworks. Improving the maintainability and providing an intent for developers to build on top. It is an attempt to close the gap between different libraries and frameworks focused on intent and communication. As web applications are becoming complex, it requires attention from developers to architect a long live solution through an architectural style.

Further research is required to evaluated the benefits or down sides of the proposed architecture. The focus is to evaluate if it brings any benefits in real world projects regardless of the frmework used or library.

[1]https://create-react-app.dev

References

[1] Rogério Brito. 2009. *The algorithms bundle.* https://github.com/falsy/react-with-clean-architecture.

[2] Alistair Cockburn. 2007. Hexagonal architecture. https://alistair.cockburn.us/hexagonal-architecture

[3] James O Coplien and Gertrud Bjørnvig. 2011. *Lean architecture: for agile software development.* John Wiley & Sons.

[4] Marianne Grov. 2015. *Building User Interfaces Using Virtual DOM.* Master's thesis.

[5] Tom Hombergs. 2019. *Get Your Hands Dirty on Clean Architecture: A hands-on guide to creating clean web applications with code examples in Java.* Packt Publishing Ltd.

[6] Facebook Inc. 2021. *React - A JavaScript library for building user interfaces.* https://reactjs.org.

[7] Ivar Jacobson. 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley.

[8] Glenn E Krasner, Stephen T Pope, et al. 1988. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming* 1, 3 (1988), 26–49.

[9] M. Marabesi and I. Frango Silveira. 2020. EVALUATION OF TESTABLE, A GAMIFIED TOOL TO IMPROVE UNIT TEST TEACHING. In *INTED2020 Proceedings* (Valencia, Spain) *(14th International Technology, Education and Development Conference)*. IATED, 330–338. https://doi.org/10.21125/inted.2020.0150

[10] Matheus Marabesi and Ismar Frango Silveira. 2019. Towards a gamified tool to improve unit test teaching. In *2019 XIV Latin American Conference on Learning Technologies (LACLO)*. IEEE, 12–19.

[11] Robert C Martin. 2012. *The Clean Architecture.* https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html.

[12] Robert C Martin. 2018. *Clean architecture: a craftsman's guide to software structure and design.* Prentice Hall, Boston, MA. https://cds.cern.ch/record/2288410

[13] Taylor Otwell. 2021. *Installation.* https://laravel.com/docs/8.x.

[14] Jeffrey Palermo. 2008. *The Onion Architecture : part 1.* https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1.

[15] Super powered by Google. 2021. *The modern web developer's platform.* https://angular.io.

[16] Mark Richards. 2015. *Software Architecture Patterns - Understanding Common Architecture Patterns and When to Use Them.* https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437.

[17] Inc. or its affiliates VMware. 2021. *Spring Boot.* https://spring.io/projects/spring-boot.

[18] Evan You. 2021. *The Progressive - JavaScript Framework.* https://vuejs.org.