# Data Structure

The app will use JSON objects to establish the communication between SPA and API. These object will be, mainly, a single cash flow and a list of cash flows.

## Single cash flow result

A single cash flow will be a cash movement, related to a user and will contain information such as the title, the classification, date and amount. The classification should be an id as it will be selected from a dropdown for the user so the list can be filtered by these classifiers without introducing mistakes.

An example for this kind of object could be the following:

{

“title”: “New keyboard”,

“classification”: 2,

“date”: “2012-04-23T18:25:43.511Z”

“amount”: 36

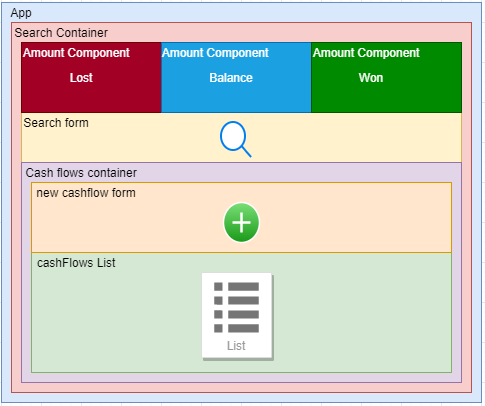
}

To bear in mind, the date variable should follow the JavaScript’s Date format, as the language specifies.

# Components

As components are the most important part in React, is necessary to determine which ones are going to interact with Redux and which ones are for presentation. In that way, the reuse of each component and therefore, the structure of the application will be highly improved.

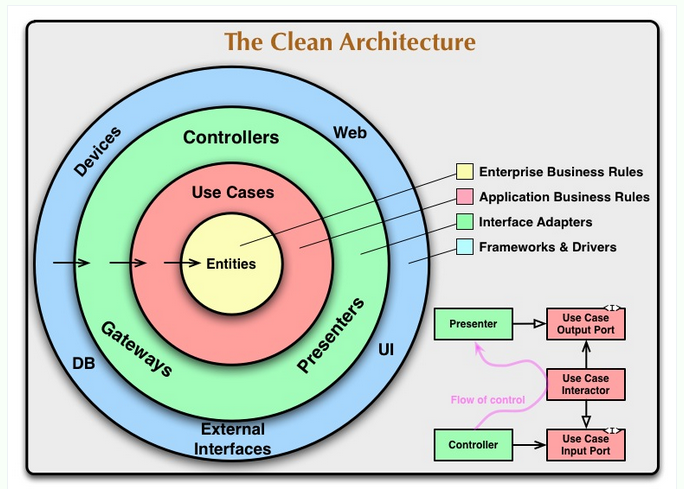
The component hierarchy should look something like this:



The Search Container will be rendered with the cash flows, amounts and the search form. All these components will need to communicate each other as the list of cash flows depends on the search made by the user, that will provide and action to the Redux store, the amounts also depend on the list. Also, a new item in the list should alter the list.

# Project Architecture

Basically, following [the article](https://janithl.github.io/2019/10/react-clean-architecture-part-1/) posted on [Janith´s Blog](https://janithl.github.io/) the this SPA React application it´s going to follow the Clean Architecture, trying to make a software much more readable, ready to change, in case the software had new requirements and makes it 100% testable.



https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Following that schema, the application will have three different parts:

* **Entities:** Should encapsulate the business rules, objects with methods or sets of data structures. Keeping it simple, should have models and mappers to these models. An operational change should not affect this layer.
* **Use Cases:** Contains application specific business rules. It contains the implementation of the use cases of the application, should have data flows to and from the entities. This layer should only change with operations in application or use cases specifications.
* **Presenter:** In short, it maps form use cases to the Interface layer.
* **Interface:** should contain the data ready to show (DTOs) and the UI. This layer it´s in charge of displaying the data and therefore should only change with UI design changes.

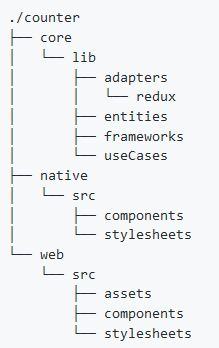
The principal idea behind this architecture is the domain is what dictates the codebase organization is the business rules and not frameworks used to achieve these business rules.

To bear in mind, the dependencies should only flow into the schema above, so the infrastructure can have dependencies with use cases and this with entities, but not otherwise.

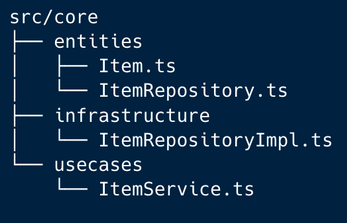
# Project structure

Taking as example [the project by eduardomoroni](https://github.com/eduardomoroni/react-clean-architecture) and [the project by janithl](https://github.com/janithl/react-clean-arch/tree/6224ee5f77b1398c41163ffedc78faf786962cc1) and the clean architecture briefly explained before, a basic structure should be stated for the project.

Both project structures are represented in these images:



2 project by eduardomoroni



3 project by janithl

With these structures in mind, the project should state a structure with these folders: **entities, use cases, presenters and interface.** In short, these folders should contain:

* **Entities:** all business rules entities and data structures.
* **Use cases:** specific business rules and data flows from/to entities layer.
* **Presenters:** datastructure mapping between use cases and interfaces.
* **Interface:** All components and DTOs from application, ready to show data.

The structure could have had the eduardomoroni approach but, as the application is simple, at least the first idea of it, it wasn´t considered necessary to add that kind of depth to work in this application. Also, it isn´t just like the janithl approach as the application, in my opinion, and looking for a more decoupled structure, should contain a presenters layer to map between layers.

So the structure approach is a mix between both example projects.

# References

How I architected a single-page React application, 13rd April 2019, by Gooi Ying Chyi <https://www.freecodecamp.org/news/how-i-architected-a-single-page-react-application-3ebd90f59087/> Part II: https://blog.usejournal.com/how-i-architected-a-single-page-react-application-part-ii-redux-d6eaf235f4d#956b

**react-clean-architecture** project, latest commit on 18th Jul 2019, by eduardomoroni <https://github.com/eduardomoroni/react-clean-architecture>

On doing ‘Clean Architecture’ in React applications (Parts 1 & 2), 6th October 2019, by Janith Leanage <https://medium.com/@janithl/on-doing-clean-architecture-in-react-applications-666d568362e> Part II: <https://janithl.github.io/2019/10/react-clean-architecture-part-2/>

The Clean Architecture, 13th August 2012, byRobert C. Martin <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>