Frontend development of the TV guide of Sky Italy

Master of Science Thesis

Written by:

JAIME ALBERTO LONDONO CIRO

Tutors:

ANTONIO SERVETTI (Academic)

SIMONE LAZZARONI (Enterprise)

POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING

TURIN, ITALY

2018

Table of Contents

1. Introduction

1.1. A Small introduction to the company

Sky plc is the leading entertainment company in Europe with more than 22.5 million customers, and 31 thousand employees distributed across five countries: UK, Ireland, Germany, Austria, and Italy. With revenues of more than £12.9 billion and a strong investment in innovation, development, and production of original content.

Sky Italy was founded in 2003 and is currently the leading media company in the country in terms of revenues. Its offer is available via satellite and fiber, through a range of products from Sky HD, to On Demand, to Sky Go, and so on. That way, the company has almost 5 million customers in Italy offering exclusive and unique content all the time.

Sky works constantly to maintain its leadership and has decided to support its evolution making a strong focus in the digital ecosystem. In that context born the Agile Software Center, a place where technical excellence is pursued in the fields of development and data processing. The present project was developed in that environment, surrounded by people that are eager to help, that collaborate and share knowledge, and that is motivated to create better solutions to the existing problems.

1.2. Sky tv guide review

Among the wide variety of applications offered by Sky, the tv guide is an important one, as it provides updated information about the events that are transmitted within the different available channels for its customers. It receives thousands of visits on a daily basis, and with the expansion of the company, the number is expected to grow fastly in the following years.

At the moment, the application could be found at http://guidatv.sky.it/guidatv/grid.html. A screenshot of the main grid of the web page could be seen in Image 1.

Image 1. Screenshot of the actual TV guide.

The above image shows the different channels offered by Sky in a grid format. At the top of the application, there are some controls to scroll horizontally and vertically, changing day, time of the day (morning, afternoon and evening) as long as a filter to select some channels based on category (News, Sport, Cinema, etc). Just below there are three main elements: a timeline with intervals of half an hour, a sidebar with all the channel numbers and logos; and a main table which contains some information about the programs (title and a small description).

After using the application for a while, and interacting with the different elements and controls of the page, some points that could be improved have been extracted from the perspective of an end user.

The first of these points is related to the loading time. Currently, that is one of the most important variables in the customer perception of the performance of a web page, therefore an improvement in the following actions/scenarios is essential:

- The first time the page is rendered.
- Each time the user change day.
- Each time the user applies a filter (at the time the whole page is reloaded after this action).
- Each time the user wants to go forward and backward in the timeline.

A second point that contributes a lot in the final user experience is the clearness of the information and intuitiveness of the controls that the page offers, which is not always an easy task taking into account how dynamic could be the schedule of events of the different channels. Consequently, it is important to improve:

- How the events are rendered in the grid (sometimes is difficult to see the name of the programs, the description, or some additional information).
- The signal about which events are on air.
- The responsiveness of the application
- Making all the controls as straightforward as possible, to make the navigation around the events fast and user-friendly.

Going from the actual user experience perspective to the understanding of the underlying project was the second step in the analysis of the application. It was developed some years ago when the offer of the company, its way of work, and the technology landscape were undoubtedly different and only supported/extended afterward to adapt to minor requirements. As a result, from the project perspective some of the problems that were found are:

- Lack of support: the app was developed by an external provider
- Technology change: the tooling landscape has evolved and transformed from the moment in which the application was developed.

- The documentation of the project wasn't updated regularly.
- Lack of automated tests or code quality checks.

1.3. Goals of the project

According to the above-described problems and points of improvement, some objectives of the project are defined in this section.

From the **product** perspective:

- Improve the performance of the application.
- Develop the product having in mind the usability:
    - Rendering time
    - Intuitive controls and faster responses
    - Graphical signal of on-air events.

From the **project** perspective:

To tackle the above-defined problems the project is intended to be self-documented through good practices, not extensive documentation, meaning:

- A Clear **architecture** definition
- Automated tasks to check code quality.
- Putting special attention and maybe also defining explicitly for the project what **clean-code** means
- Snapshot, unit, integration, and end-to-end **tests**.
- Meaningful logs.
- Continuous integration and delivery workflow.

1.4. Structure of the report

This report is intended to be a walk through the development of a new proof of concept for the tv guide of Sky Italy. Starting from the selection of the main technology and its surrounding ecosystem, as well as the description of the adopted methodology; following with some development details that go from the initial project setup to the configuration of a pipeline to integrate and deliver the code automatically; from where some of the problems faced during the development will be presented, going from the data model, to performance considerations, rendering, testing, and so on. Finally, the achieved results will be sum up, after which the report will end with the conclusions and proposed future work.

2. Background

2.1. Main technology analysis

2.1.1. React vs Angular vs Vue

Community and popularity:

In the frontend world of today, the most trading frameworks/libraries are React, Angular and Vue. It can be proved watching the statistics about downloads, stars on Github, questions in StackOverflow, etc, as shown in the images 2 and 3:

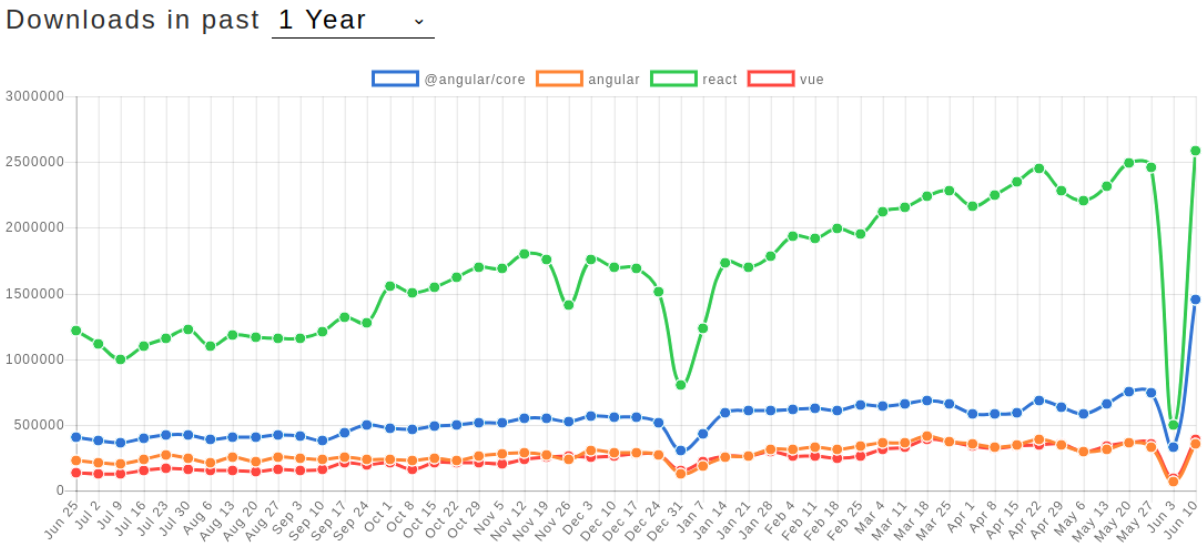

Image 2. Downloads of angular, angularjs, react and view in the past year.

Github Stats

	stars	forks	issues	updated	created
angular.js	58633	28940	523	Jun 16, 2018	Jan 6, 2010
angular	37335	9009	2413	Jun 16, 2018	Sep 18, 2014
vue	101961	14420	225	Jun 16, 2018	Jul 29, 2013
react	101961	18544	477	Jun 16, 2018	May 24, 2013

Image 3. Github stars of angular, angularjs, react and vue.

The wide support that these technologies have is clear from the images above. There are people following, using, and discussing about them which is the base to select a technology for a large project. But not only, the team behind the development is also important. Regarding it, is well know that Angular is a project from Google, while React comes from Facebook, and Vue, on the contrary, is an independent project supported mainly by the community.

It is astonishing finding a framework without a huge company behind it, with the vast support that Vue has. It makes it stable, reliable, and hopefully community-driven. On the other hand, being backed by a company gives React and Angular the advantage of having full-time development staff working in the support and improvement of the technology, but also makes it prone to be evolved toward the corporate needs.

In summary, besides the small trade-offs described above, whichever of these three technologies is a valid starting point for a modern frontend solution from the community viewpoint. Therefore, in the next pages a deep analysis of these tools will be presented remarking the differences, the advantages, and disadvantages of each one and the tradeoffs of selecting one in spite of the others.

Framework vs Library (Flexibility):

React is a library, Angular and Vue are frameworks. React helps only with the problem of building user interfaces, proposing a declarative, component-based approach. It does not have internal tools for routing, form validation, or testing. Instead a Framework like Angular come with everything inside, it has a defined structure with all the tools a modern frontend solution will require, as has been summarized in table 1:

	Angular	**React**	**Vue**
Forms	**X**	react-forms	**X**
Testing	**X**	Jest, Enzyme	**X**
HTTP communication	**X**	Fetch, Axios,...	vue-resource
Routing	**X**	react-router	vue-router
Components	**X**	**X**	**X**

I18n	**X**	react-intl	vue-i18n
Animation	**X**	react-motion	**X**
CLI	**X**	**X**	**X**

Table 1. Angular, React and Vue ecosystems.

Although these days all the frameworks are modularized so also Angular have a core library and helping modules to be included for routing, forms, etc. What table 1 summarizes is that while Angular has internally a clear solution for each aspect of development of a Single Page Application (is more opinionated), React does not offer internally the tools to solve those problems, but has a huge ecosystem around it that lets the developer to choose the best tool for each scenario. Which could be seen as an advantage in lot of cases, and as a problem in many others as will be explored in the following sections. Finally, Vue could be placed in the middle of the two solutions offering clear alternatives to all the traditional problems but in a more flexible way than Angular.

To sum up, the following highlights could be extracted from the discussion above:

Angular and Vue:

- More opinionated
- Less decision fatigue
- Automatic setup (using the CLI)
- Consistency across teams and projects (the technologies are almost the same)

React:

- Allows to choose the best tech for each user case / scenario
- Light-weight: it is possible to include in the project only what is necessary
- Automatic setup (could be customized) using “create-react-app”.

Separation of concerns:

A big difference between Angular/Vue and React is the approach to separate things internally in an application. One of the most interesting points React opens from its launch was the opportunity to think again what the pattern "Separation of concerns" mean, and how to achieve it. In the image 4, on the left, it could be seen the approach of Angular/Vue, that separate the technologies (html, js/ts, and css), but as often happens it leads to intertwined concerns. Instead, React (as it is represented on the right) mix lot of times logic and markup (and even style) in a single file, but encourages separation of concerns through the use of components.

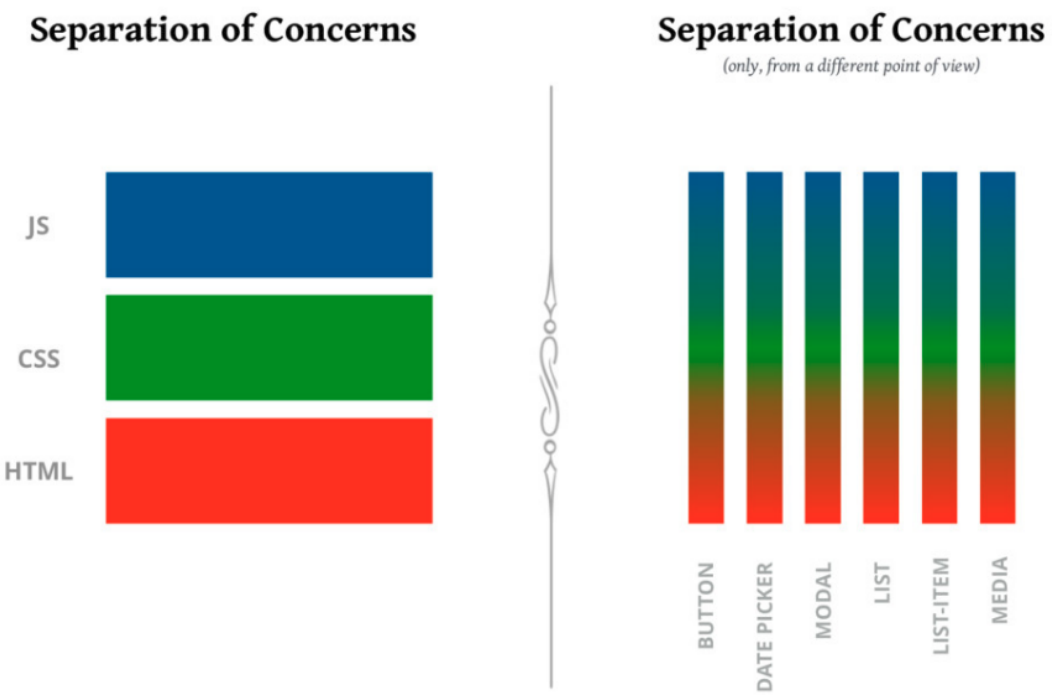

Image 4. Separation of concerns from different perspectives. (Rastelli, 2017)

The discussion about the way we understand this principle and some others was reopened in 2013, in a JSConf, when Pete Hunt presented and analysis about it in a talk called "React: rethinking best practices" (Hunt, 2013) from where I would like to highlight and discuss the following points.

First of all, his definition of React as a library for building user interfaces, that is, for rendering and respond to events. Secondly, the definition that he gives of separation of concerns as "reducing coupling and increasing cohesion", understanding coupling as "the degree to which each program module relies on each of the other modules" and cohesion as "the degree to which elements of a module belong together"

From that point of view, what he argues is that templates encourage a poor separation of concerns, as the view model tightly couples the template to the display logic. He says: "markup and display logic are inevitably tightly coupled and are highly cohesive", so inevitably, in his words "templates separate technologies, not concerns" and that is what is summarized in image 4. On the other hand, React components are proposed as "a **highly cohesive** building block for user interfaces **loosely coupled** with other components" which makes them truly reusable in different scenarios, and unit testable.

2.1.2. Why react?

After the initial analysis of the different technologies, React was selected as main technology for the development, some of the reasons that lead to the decision where:

- Flexibility: it could be used to develop web apps, mobile or desktop applications.
- Encourages the improvement of JavaScript skills
- Small API and framework specific syntax
- Corporate investment: Facebook
- Huge active community
- Performance (virtual DOM)
- Testability
- Less opinionated: could be complemented with other libraries depending on the requirements
- Initial learning curve

2.1.3. Problems with React:

Some of the problems that the people usually remark as drawbacks or difficulties of using React are:

- Decision fatigue: it only solves the problem of the UI, needs to be complemented with other libraries.
- JSX: not all the people like the idea to have JS and HTML in the same file, and the implications it brings, as for example the change of some important keywords like class for className, the limitations using css pseudo-classes, etc.
- Cross-team consistency, its flexibility could lead to projects that have completely different structures and technology stacks.

2.2. React ecosystem

2.2.1. Type checking

For modern apps that are intended to grow and evolve over time, a type-checking system is necessary to avoid bugs before going into production. In this field, the major competitors are Flow and Typescript. Additionally, React has its own built-in type-checking tool called prop-types. Let's review all these options in order to select one for the project purposes.

- React "PropTypes": https://reactjs.org/docs/typechecking-with-proptypes.html
- Flow: https://flow.org/
- Typescript: https://www.typescriptlang.org/

The simplest approach is to use the already included PropTypes as a validation method, but it is also the less powerful and flexible. It could be used only to validate and give default values to component properties. Flow and Typescript are more complex and complete tools

to accomplish the objective of type validation of a whole application. Flow is a Facebook project, while Typescript comes from Microsoft.

Typescript is the more popular solution, used officially in angular from version 2, it has strong support and is widely used. Flow allows to do the same things, but is a little more flexible and maybe convenient for projects that are traditionally made in vanilla JavaScript, but that want to introduce a type checker tool gradually, as it could be included partially only in the required files, using a special directive.

To sum up, Proptypes are not a complete solution for a big project, Typescript could be and excellent option, but maybe is too strict for newcomers not used to typed languages, while Flow offers similar capabilities to the other tools, but could be included gradually/partially in the projects. Furthermore, it is a project backed by Facebook too, so it plays very well with React, for that it has been selected for this project.

2.2.2. Linting

Linting refers to the process of analyzing statically (without executing) the source code based on some rules in order to control (test) the code and detect possible violations. Some of the most important benefits given by linting tools are: pre-code review, augment code readability, find syntax errors in the development phase (before production), and homogenization of the codebase. (T, 2017)

There are linters for almost all the major languages, in this project two of them will be used to validate javascript and css code:

- ESLint: https://eslint.org/ for JavaScript
- Stylelint: https://stylelint.io/ for CSS

2.2.3. Testing

Automated tests have become an essential piece in modern software development to ensure code quality and dynamic documentation. Having a good battery of tests give developers confidence when they are making changes to an existing functionality. According to the "test pyramid" model proposed by Martin Fowler (image 5), an application should have an excellent coverage of unit tests, integration tests of the critical paths and only end to end tests of the most important use cases.

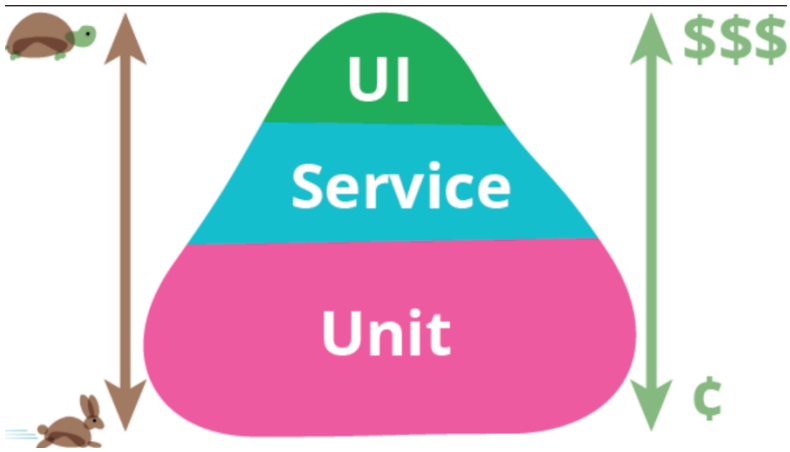

Image 5. Test pyramid (Fowler, 2012)

2.2.3.1. Unit, snapshot, and integration tests

There are lot of tools to make unit tests, the most popular frameworks to accomplish this objective are Jasmine and Mocha, being the former a most complete solution, while the latter a more customizable one. Angular uses Jasmine as framework and Karma as a test runner, instead React could be tested with any tool, but the recommended one is Jest.

- Jest: https://facebook.github.io/jest/

Jest is a testing tool for JavaScript in general, that has been developed by Facebook and is commonly used along with React. It has a built-in test runner, assertions, and code coverage functionality. It supports also the concept of snapshot testing, that helps to ensure that the user interfaces do not change unexpectedly.

Jest is commonly complemented with Enzyme (http://airbnb.io/enzyme/docs/api/), a specific library developed by Airbnb to facilitate React testing, including utilities to assert, manipulate, and render components in different ways.

2.2.3.2. End-to-end tests

Unlike unit testing, in this field there is no a predominant solution, so different tools were evaluated. Traditional tools rely on Selenium (https://www.seleniumhq.org) to control and automate the browser, being the most important ones:

- Nightwatch: http://nightwatchjs.org/
- Webdriver.io: http://webdriver.io/

There are other kinds of tools, that operate the tests in the headless versions of the browsers, Puppeteer, a project from Google, is an example of that:

- Puppeteer: https://developers.google.com/web/tools/puppeteer/

Finally, there are other kinds of frameworks that are gaining territory these days, for being all in one solutions that do not rely anymore on Selenium. TestCafe and Cypress are two good examples of that, as is illustrated in image 6.

Why Cypress:

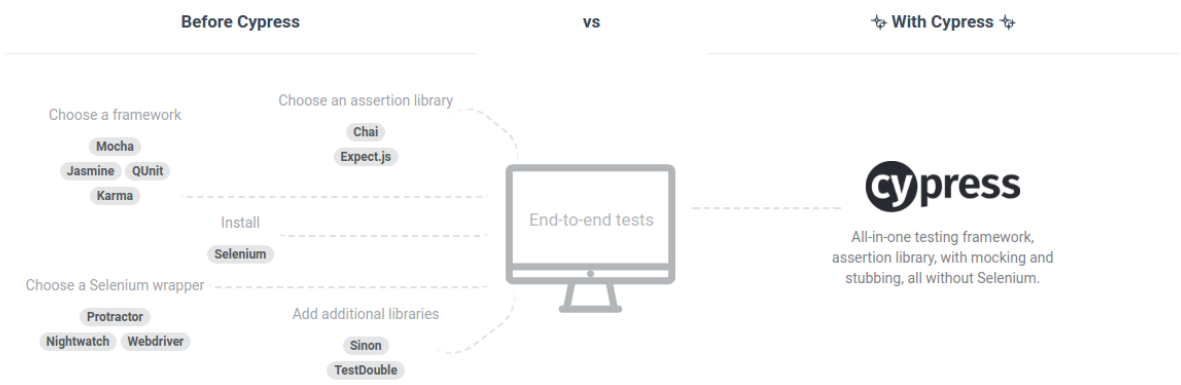

Image 6. Cypress vs traditional technologies. (Cypress, 2018)

An important part of this project was to experiment with different, non traditional technologies, that could offer better characteristics or new functionalities. Cypress has been selected for this reason, as an interesting tool that makes the initial configuration easy, that came already with all the necessary tools and that offer important features like debugging, video recording, between others

2.2.4. Styling

There are several options to style a react application, from the traditional vanilla CSS, to more recent techniques, like CSS modules or CSS in JS. A description of the possible methods is presented below:

- CSS in JS

"(...) is a technique of writing CSS styling in the JavaScript programming language. This encourages a common pattern of writing the styling with the JavaScript component it applies to, co-locating presentational and logical concerns." (Thoughtwords, 2018). Some of the advantages of writing CSS this way are avoiding dead code, scope the styles, and apply them conditionally, using the JavaScript capabilities.

- UI component libraries

Interesting approach to start fastly, have responsive utilities and standard behaviors out of the box, but for a big company like Sky, with a defined brand is limiting, as lot of custom styles and behaviors are required. (Saring, 2017)

- CSS Modules

"is a CSS file in which all class names and animation names are scoped locally by default" this is helpful to avoid problems associated to the global namespace nature of CSS, and also to detect dead code (which is sometimes a little bit tricky in plain CSS), among others.

- Plain CSS

Plain CSS is the classic approach, it could be the best alternative for teams in which there are user interface designers that take care of the styling. Anyway, when using plain CSS in a big project is important to introduce at least a methodology to scope the styles, such as BEM, or other similar notations.

3. Methodology

A software project is not only about technology, the way in which it is developed is almost as important as the tools used to build it. Traditional software projects are based mainly in methodologies called waterfall, where the work is divided into different sequential phases starting from the requirements, passing through the analysis and design of the solution from where the implementation is made and finishing with the verification, deployment, and maintenance, as is illustrated in the figure below.

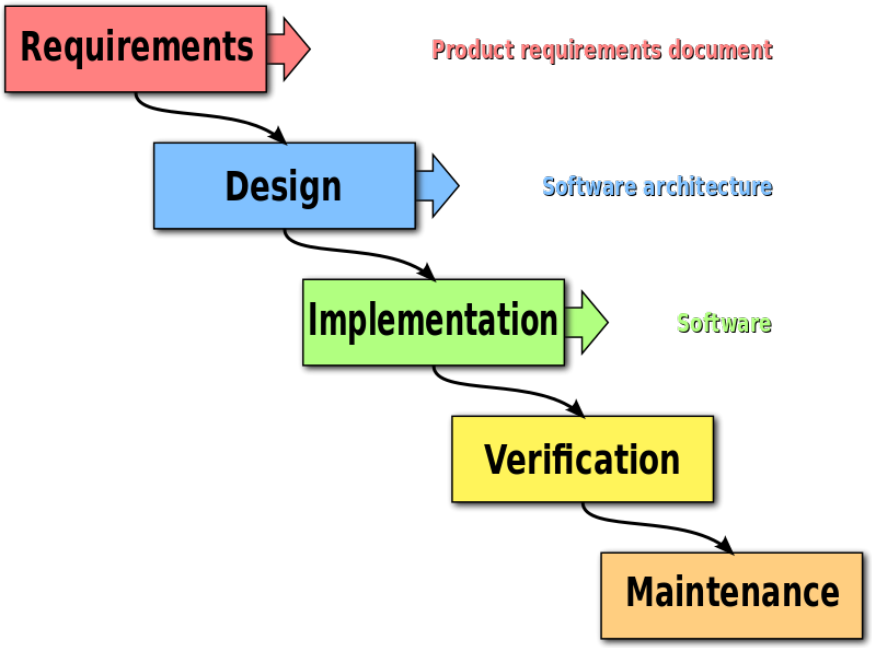

Image 7. Waterfall model (Kemp & Smith, 2010)

That was the predominant methodology until the first years of this century, when the so-called agile methodologies started to gain territory for their results in different software projects. It was in 2001 when the core values and principles of these methodologies were stated for the first time in the "manifesto for agile development" (Agilemanifiesto, 2001).

Sky, as part of a digital transformation process, decided to adopt these new methods through the use of SCRUM framework and XP programming techniques (see images 8 and 9 bellow), but extending or adapting them when necessary, having into account the core principles and practices of the agile movement.

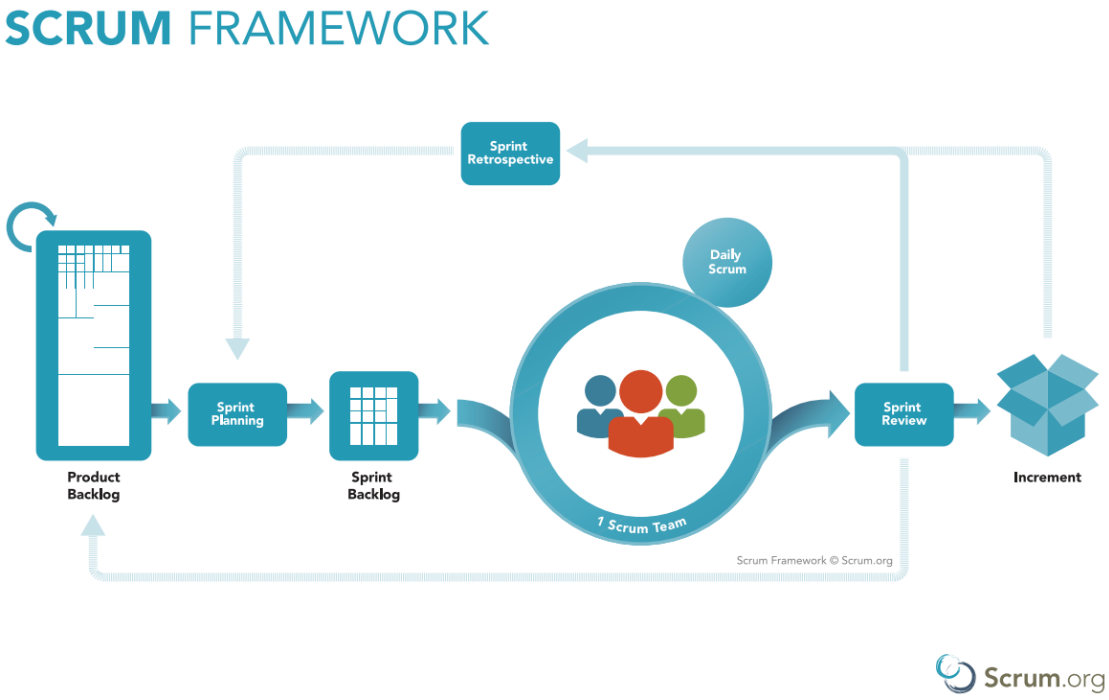

Image 8. Scrum framework (Scrum, 2018)

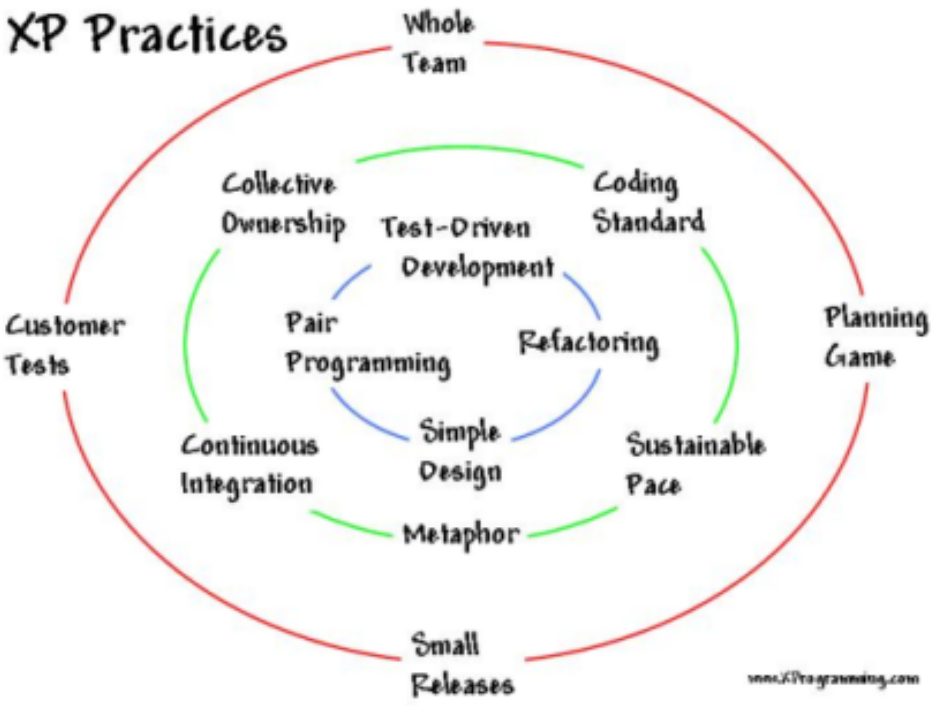

Image 9. Xtreme Programming practices (Jeffries, 2011)

All this to say that the present project has been developed in an environment that facilitates the collaboration among different people and teams, encourages the experimentation, validation, change, and continuous improvement through the review of the developed work periodically and frequently enough to allow, from one side, the addition of value/functionality to a given software, and from the other, detecting early enough if something is not what it was intended to be and "pivoting" or changing what becomes necessary.

4. Development

4.1. Starter-kit (Project structure + automated tasks)

In the first phase of the project, the objective was to develop an opinionated react starter kit that could be useful to start a project from scratch being productive from the first moment. It means, to avoid the decision fatigue described previously, have the main necessary tools already included, and a clear project structure with automated tasks to check and enforce code quality.

React itself has a command-line tool: "create-react-app" that helps to hide some of the complexity of the initial configuration of tools like Babel and Webpack, so it was used as a starting point and the following tools were included on top of it:

- For linting javascript: eslint.
- For linting css: stylelint.
- Automatic code format: prettier.
- Static type checker: flow.
- Unit/Snapshot tests: jest and enzyme.
- End-to-end tests: cypress.

Those dependencies were included, configured, and automated in the project, so the developer could run continuously all the code checks and detect immediately if something is going wrong as is shown in the images 10-12.

Image 10. Linting JavaScript code.

The image above shows two basic errors, one that is not too dangerous, but is better to solve it, to have the code clean; and the second, is really a problem that can cause unexpected results, so it should be solved.

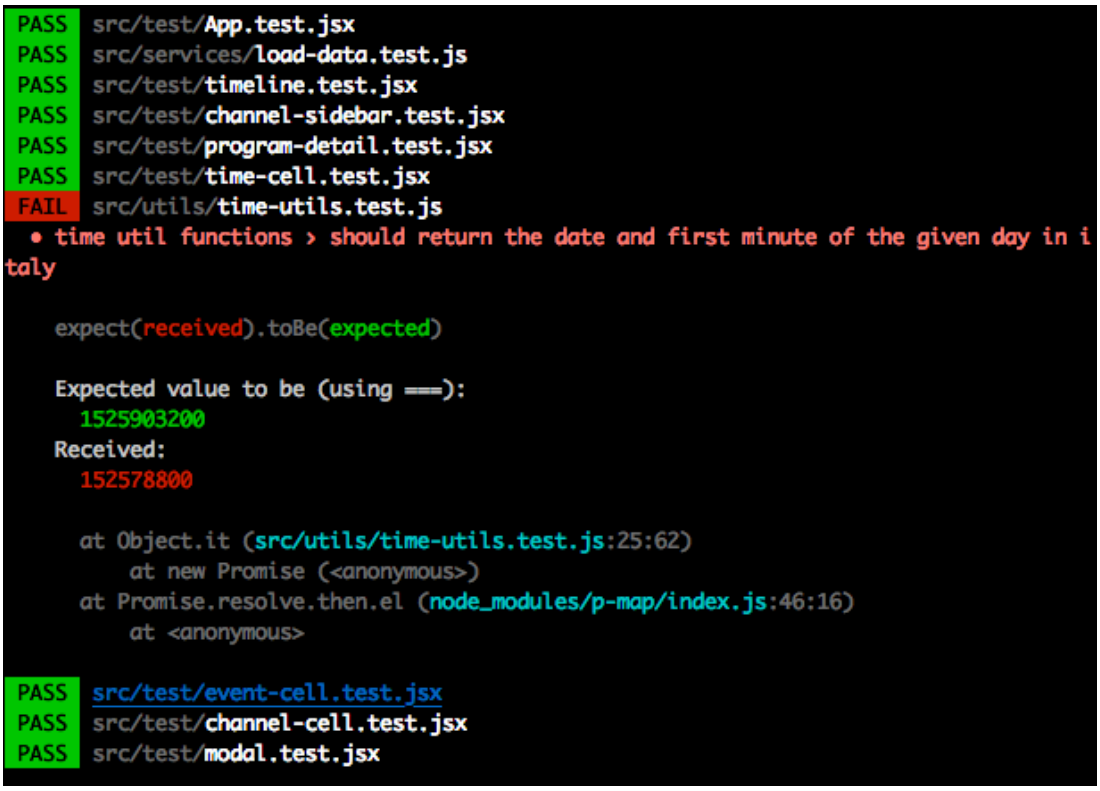

Image 11. Unit tests execution.

In the image above, the execution of some unit tests is shown. The output has one test failing, and is clear which is the problem in the assertion, the test in which it happened and the file. That way is very easy to find the breaking tests and fix the problem, that could be in the test itself or in the function that is not working in the expected way.

The last example, given in image 12, shows the results of the execution of the type checking validation using Flow. In this case is pretty straightforward to know which is the error, but still is enough to see the advantages of using tools like this, making easy to realize if a change breaks something and fix it before the execution of the code.

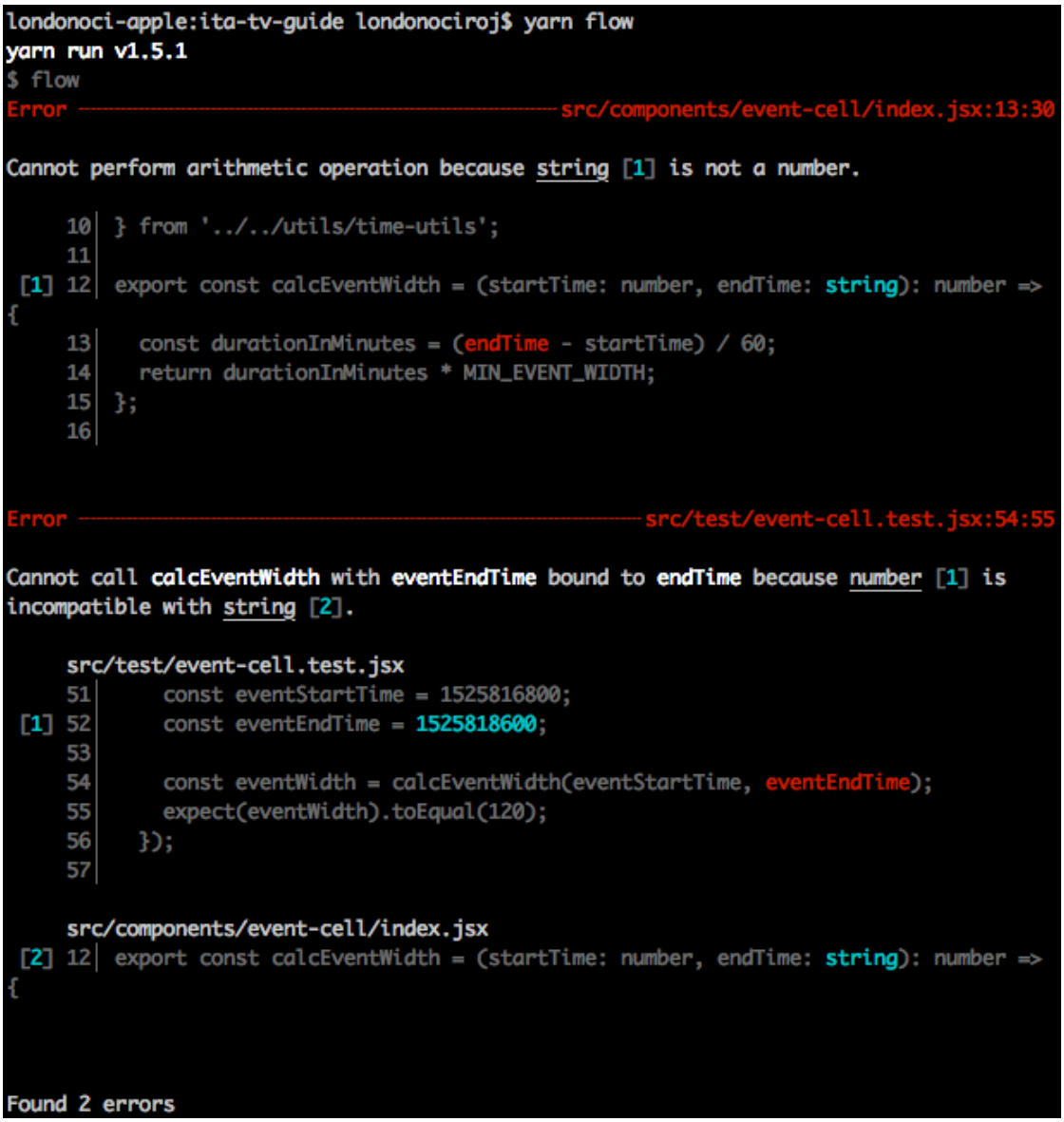

Image 12. Type-checking validation using Flow

As an IDE for the development of the project Visual Studio code was used, complemented with the following plugins:

- Debugger for chrome: https://github.com/Microsoft/vscode-chrome-debug
- Eslint: https://github.com/Microsoft/vscode-eslint
- Flow language support: https://marketplace.visualstudio.com/items?itemName=flowtype.flow-for-vscode
- Jest: https://github.com/jest-community/vscode-jest

- Prettier: https://github.com/prettier/prettier-vscode
- Stylelint: https://github.com/shinnn/vscode-stylelint

Some examples of the immediate feedback that these plugins give to the developer in conjunction with the automated tasks are:

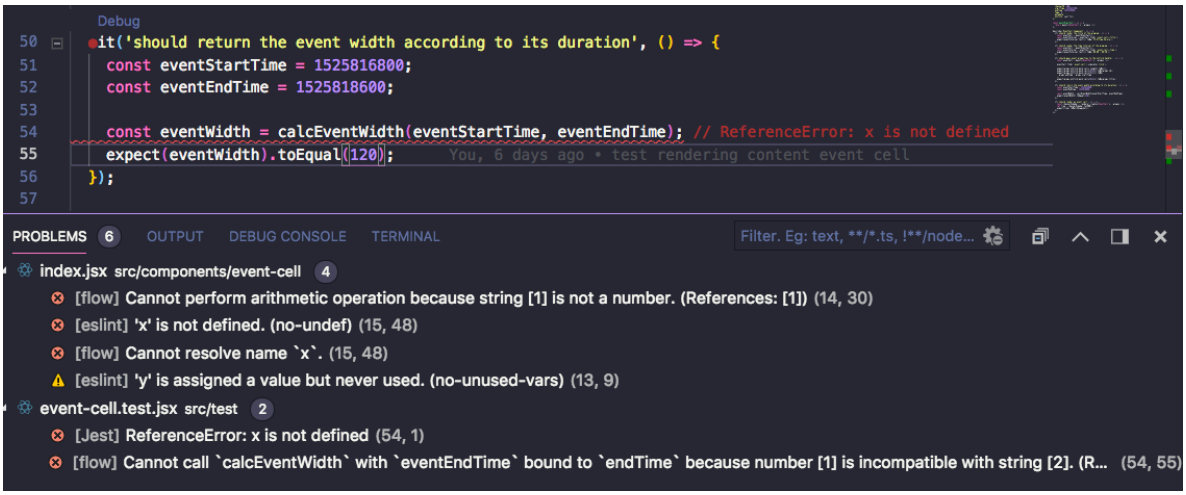

Image 13. Visual Studio Code

Which is the same output of the terminal, but integrated in the development environment, allowing the visualization of the errors in the place where they are, and in the case of tests, supporting the debugging inside the editor.

4.2. Continuous Integration (CI)

For this task Travis CI was used, thanks to the automation of the type checking, linting, testing and building tasks of the project the process consisted only in the configuration of a file with the description of the project, its dependencies, and the scripts to be ran after a commit to the project on Github.

The pipeline implemented for the project was:

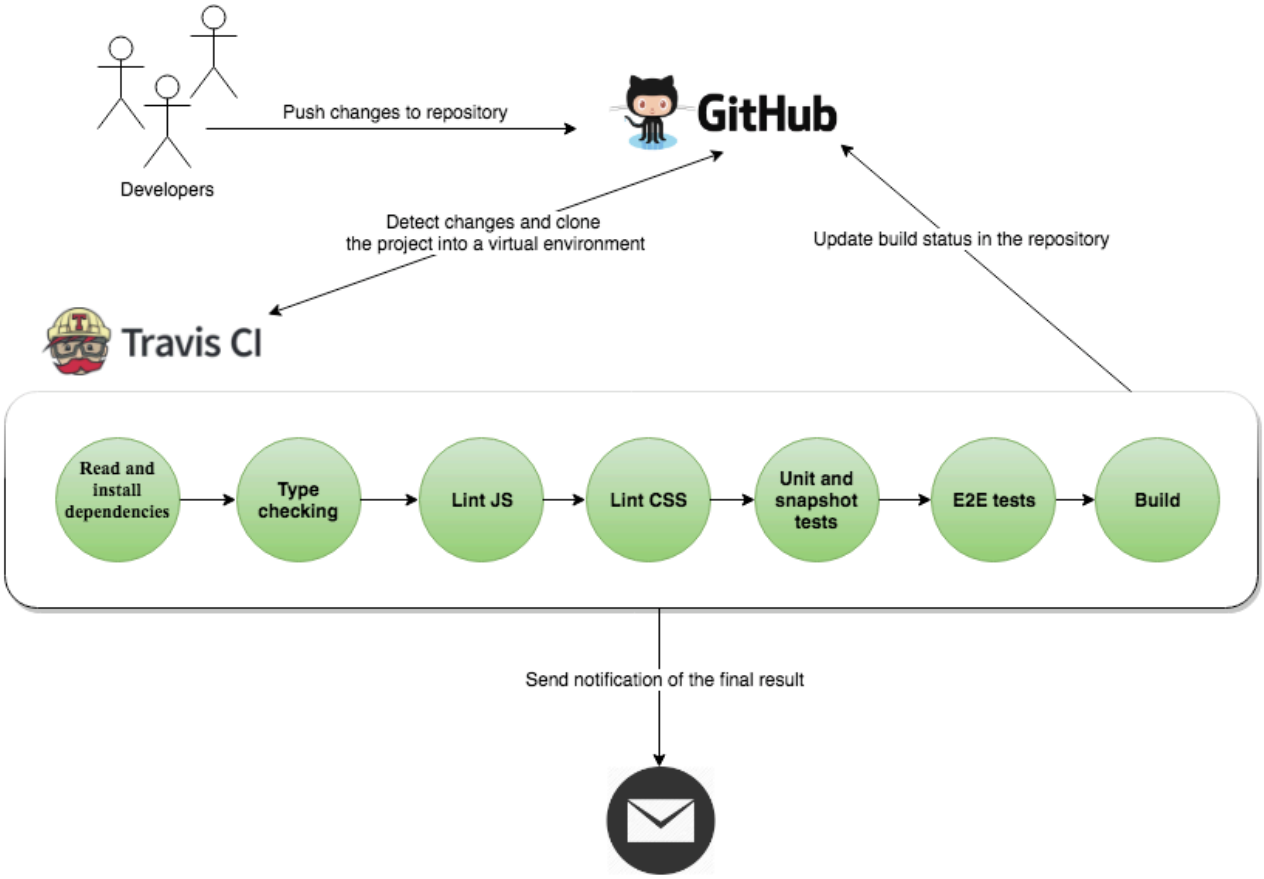

Image 14. Continuous Integration pipeline

Only if all the tasks end correctly the result is positive, other way it will fail giving a detailed description of the point of error and the problem that cause it.

Some captures of the process and end result of the process are:

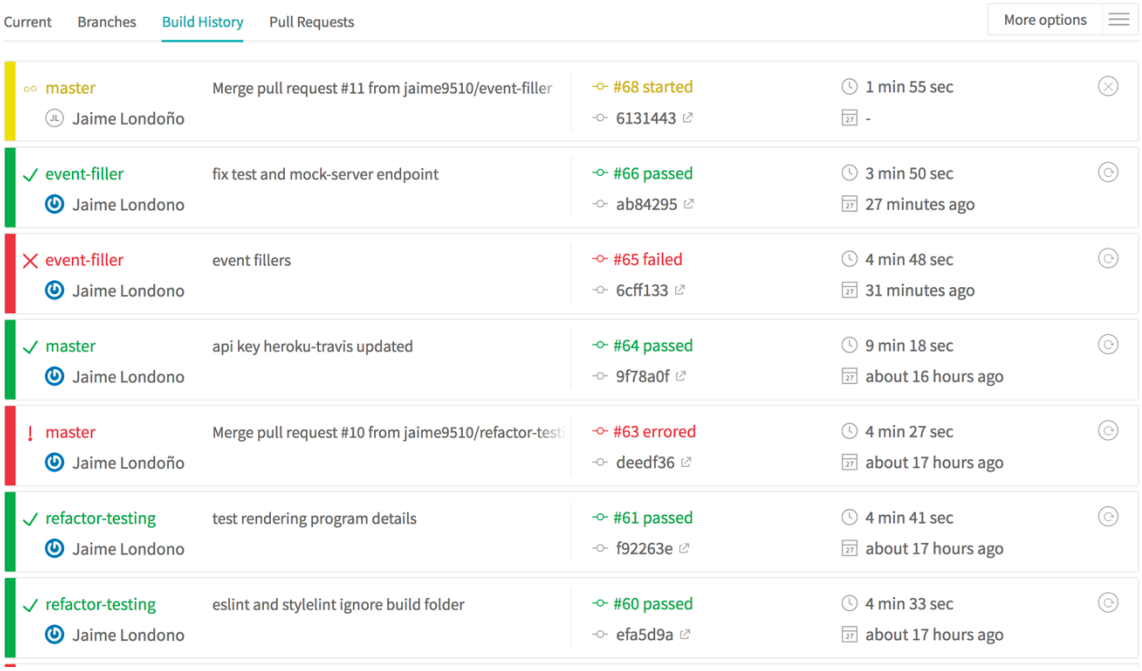

Image 15. Build history of the project.

In the image above, the build history section of Travis CI is shown. Red, green and yellow signals are used to indicate if a build fail, success or is in progress. Moreover, as it is integrated with Github, after someone makes a pull request another build is made to check if all the controls pass after the merge, helping the person who make the code review before accepting the changes and merge the code to the main line, as is shown in image 16.

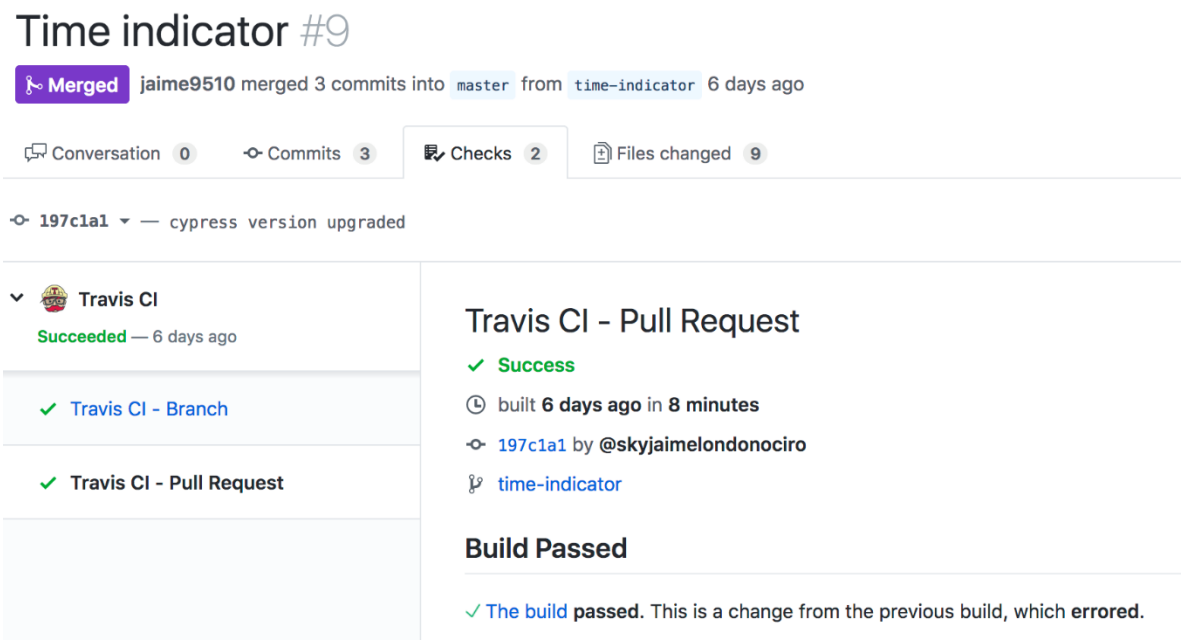

Image 16. Travis CI checks after a Pull Request

4.3. Continuous delivery (CD):

The development process proposed for the project is based on a mainline called trunk or master, from where short-lived feature branches are created. Each time a developer makes a push to the remote repository, Travis CI detect the changes and runs the pipeline described in the previous section.

Ideally the feature branches should be merged to the main line after a short period of time, passing through a pull request and consequently a code review. At that point Travis CI checks that in case of a merge to the master the pipeline continues to pass, giving feedback to the reviewer about the code as shown in image 16.

If the pull request is approved, the code is merged to the mainline and the pipeline departs again, this time promoting at the end the build artifact to the test environment, it means, deploying the application, in this case to an instance of Heroku in the cloud. All this process is illustrated in the image 17 bellow.

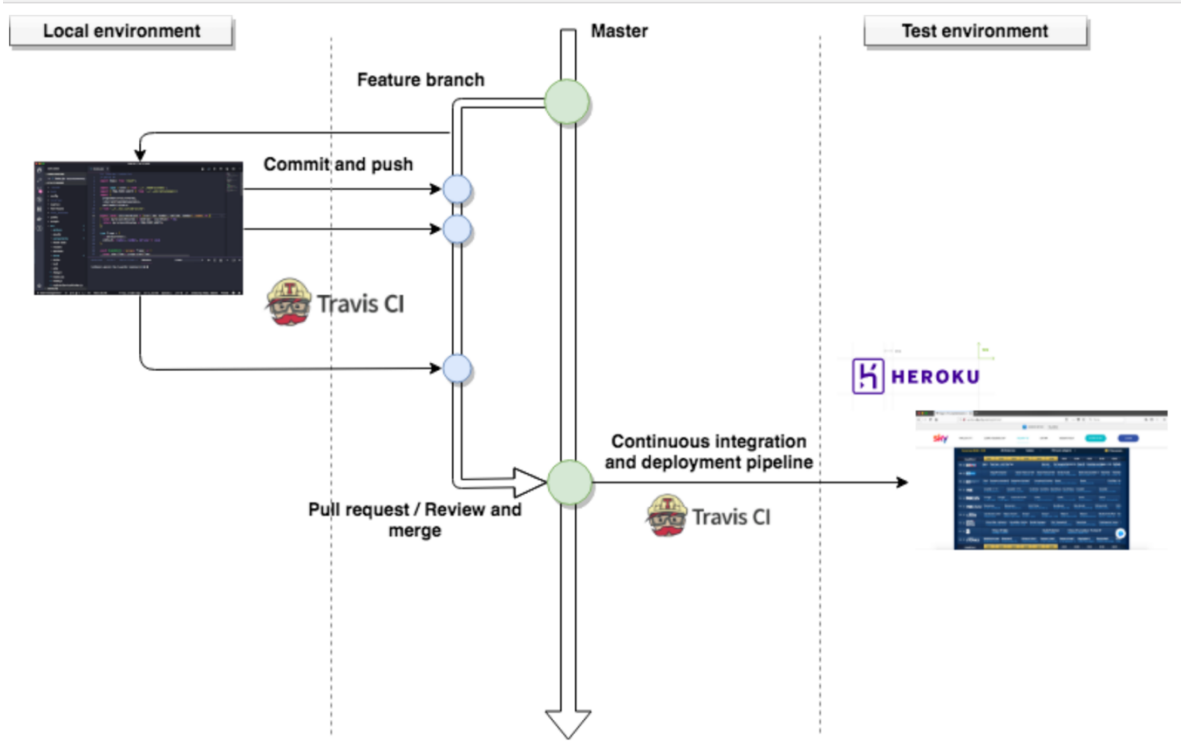

Image 17. Basic development workflow

4.4. From the actual data model to the new one

Currently, the service that generates the Electronic Program Guide (EPG) answers with a single JSON file that has all the data of the events and channels that are programmed for a single day. In the Sky world, nowadays, it means more than two hundred channels and around eight thousand events. Therefore, the model that maybe was efficient some years ago, starts to be a little bit heavy for certain devices or network conditions.

In that scenario and taking into account that not only the web application, but also the mobile one need access to the information, a new model to represent the information was conceived thinking in a more scalable solution for a growing tv offer.

The current model has a root element composed by an array of channels that have an id, a number, a name, two URL's to get the logo of the channel in different sizes and an array of events. Each event inside has a unique id, a program id (as will be explain later), the start and end time, a title, and some other additional information that is not useful for the grid representation of the tv guide.

The reason why an event has a program id is because an event is a single instance of a program, so a program could have different associate events that are shown in different channels or dates. As an example, a program could be a certain movie that lasts two hours, and there could be two events associated to that movie, one that last three hours and is shown in channel 100, and the other that lasts only two hours and a half (depending on the advertising or other factors in the transmission of the program) and is shown in channel 33.

Starting from that fact, one of the first differences in the new model is that the event has only the necessary information to be rendered in the grid, and that is unique for it (like the start and end time), while the program has the other data like genre, description, exact duration, etc. That way the amount of information in the event is reduced and also the duplication.

Secondly, the model of a channel was built independently of the events of a certain day, which take us again to the difference between a channel (that has for a long period the same associated information) and the events that could be associated to it and change every day, or even more frequently.

This change in the representation of the data could help the client applications that fetch the information, to get only what they need in the more efficient way depending on the conditions. A simple example in which this new representation helps the application to be more efficient is when the user loads the grid for the first time and starts to switch between the different days of the week. In that case, with the original model, each time the user changes the day, not only the information of the events has to be loaded, but also the channels,

because both things are tightly coupled. And load the channels again means fetch again their images which is not efficient nor necessary given the low probability of change in the information of the channel.

Instead, with the new model as the channels are loaded independently of the events, when the user changes day only the data of the new events is fetched, without reload the page, saving that way lot of network traffic and re-rendering efforts.

This new model was represented in the application using the type checking capabilities that were introduced with Flow. Also, the old model was represented this way, and some helping functions were made to transform the data from one to the other, which is not ideal, as this work should be made in the server, but it was useful to get started and allow the newly created application to reason in terms of this new domain instead of the old one.

Finally, the transformation of the model went one step further, responding to some user interface specific requirements to fetch the data from the server in a specific way. At this point there is an important difference to highlight between the general-purpose model, and a specific variation of it that could be highly coupled to a specific client application and user experience, to allow better performances. This is called backend for front end (Newman, 2015), and is illustrated in the following figure:

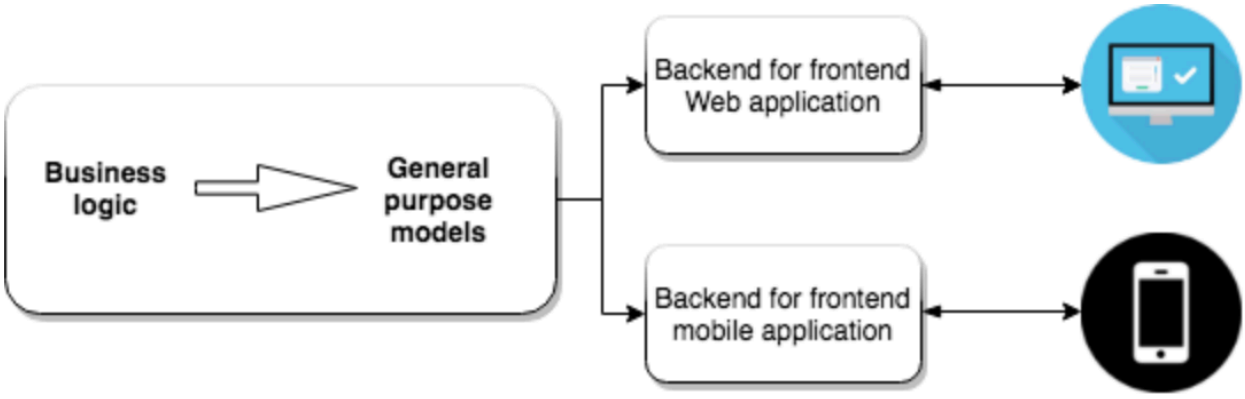

Image 18. Backend for frontend model

This way the general API is not affected for specific requirements, while the requirements that the clients could have are satisfied with a custom solution.

4.5. Rendering information

In this part of the project, the objective was to improve the rendering performance of the web application and with that the overall user experience. The first step here was the analysis of the current web page using some browser utilities. One of them measures the frame rate (expressed in frames per second) which represents the velocity at which the content of a page is updated. The image below shows the result of applying it to the current web page:

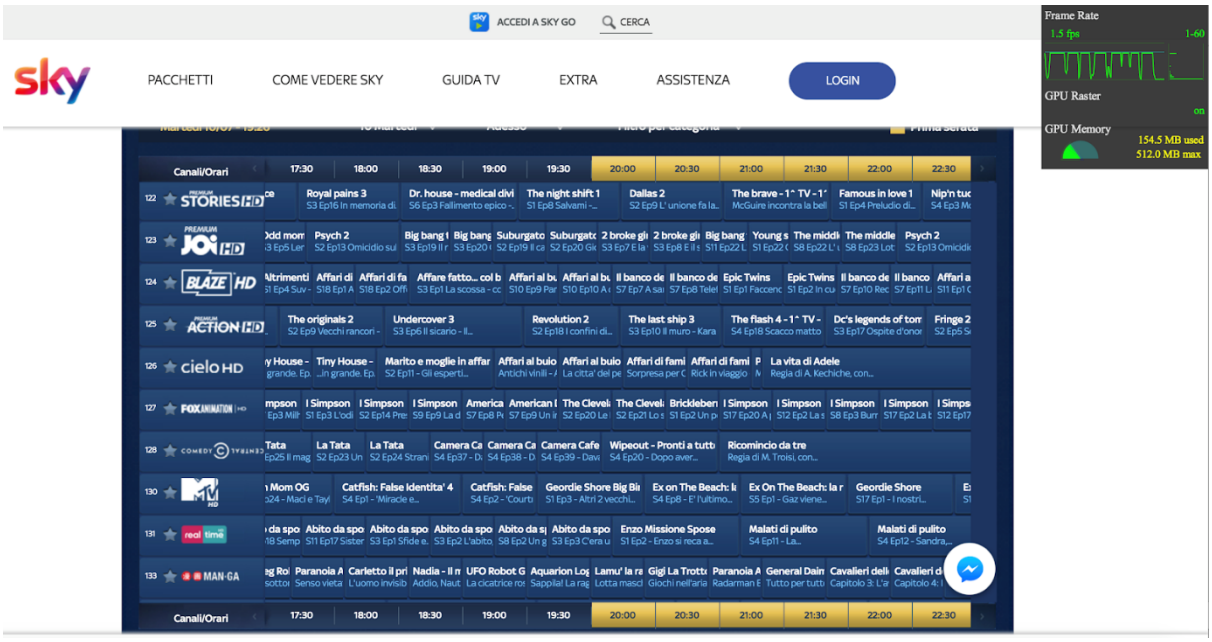

Image 19. Performance analysis current tv guide

In the left-upper corner a frame rate of 1.5 fps could be seen. It has been obtained navigating in the timeline. An ideal value here, according to google is 60 fps. This was the worst case, obtained navigating horizontally, the results were better navigating vertically, with values around 40-50 fps.

This situation, among others, is the consequence of rendering a huge amount of information (more than 200 channels and 8000 events), in the DOM at the same time. Because even if only a small number of them could be seen in the screen at a given time, the actual application has already created all the nodes to represent the information, explaining the consequently slow initial rendering of the page, and the lags on scrolling.

The proposed solution to these problems are based upon the combination of the componentization of the user interface plus the virtualization of the collection of elements to render. Therefore, following the "Think in React" (React Documentation, 2018) steps, and the "Atomic design principles" (Frost, 2013), the original UI could be decomposed as follow:

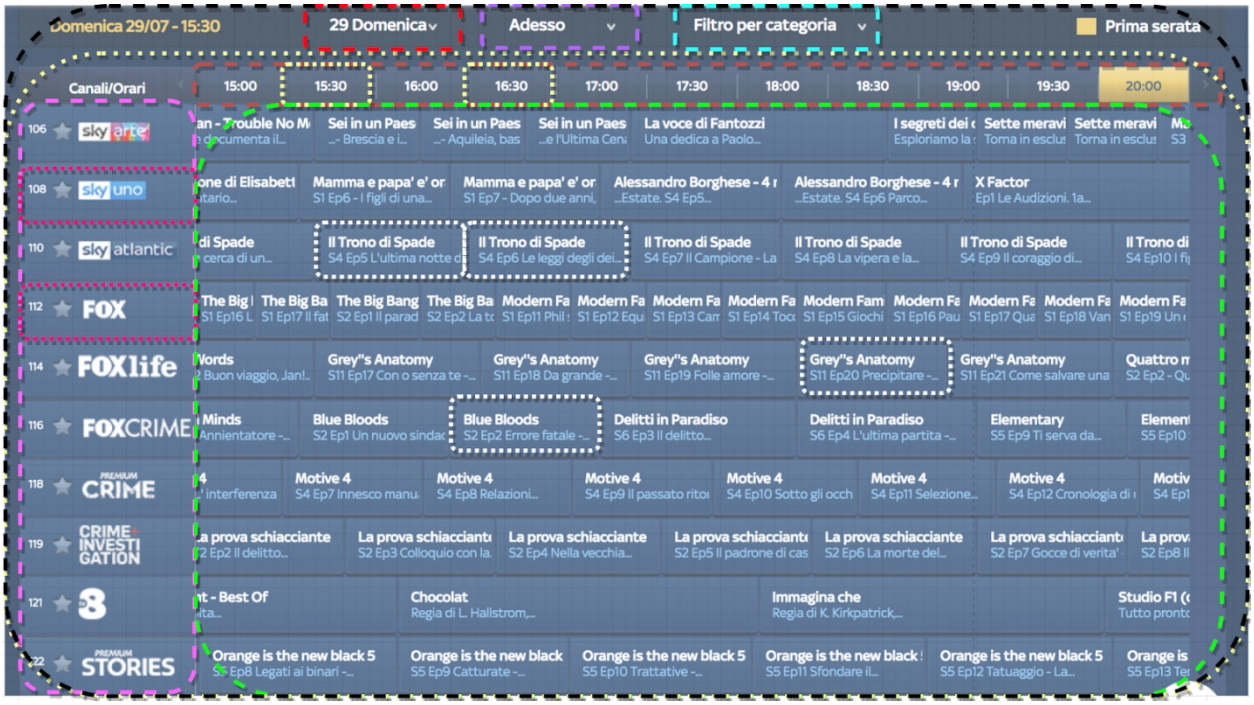

Image 20. TV guide, component hierarchy.

The image above is a visual representation of how the original tv guide could be decomposed in different functional units. As an example, a channel (rounded in red in the figure), could be seen as a molecule, composed of different atoms: a channel number, a selector, and an image. Putting together different channel molecules, we could form an organism: the whole sidebar of channels (rounded in pink on the image). In the same way the

timeline is composed of time boxes, and the whole grid is composed of single events. Another organism is the controls bar, composed by different kind of filters to fetch the data of the different days, move forward and backward on the timeline, and filtering the channels by category. As a result, we could see how all these organisms are arranged in the final page.

Starting from this analysis, a collection of components was developed in order to build an application based on small reusable parts, that could be rendered dynamically and independently, which take us to the second objective: avoid the representation of all the information in a single moment. Instead, through the DOM virtualization, and a technique known as “windowing” (React Documentation, 2018) it is rendered on the screen the subset of data that is visible for the user at a given time, replacing (instead of adding) the existing nodes in the DOM as the user scrolls to see the different content, as is illustrated on the image below (Vaughn, 2017).

Image 21. “Windowing” concept illustration

4.6. Routing

As have been explain in the background section, React does not provide an integrated solution to handle the routing requirements of a single page application. Instead, the community has developed different solutions to supply this requirement, between them, React Router is the most used library.

The routing requirements of this project are mainly related to the navigation without refreshing the whole page, i.e., changing views and URL's as the user interacts with the application without reloading each time all the content. More specifically, some of the routing related requirements of the projects are: changing the day of the grid (today, tomorrow, Saturday, etc), applying some filters to the guide based on the category of the channels, and allow the users to visualize the detail of the different events programmed in the week.

This is a classic user case in a Single Page Application, that is usually solved with the so-called static routing, where all the routes are declared from the beginning and included in the application from the initialization phase, before any rendering of the content takes place. Instead, the component-centric approach of React is extended also until this field, so in the implemented application also the routes are components, giving space to a more dynamic and declarative approach, as will be explained in the following pages.

Starting from the official react-router page, the authors define dynamic routing as: "the routing that takes place as your app is rendering, not in a configuration or convention outside of a running app. That means almost everything is a component in React Router". But not only, they encourage the people to start thinking the routing of an application as a part of the user interface, instead of the traditional static configuration.

In the actual app, adopting this development mode has meant:

1. Wrap the whole application in a higher order component BrowserRouter that allows to keep in sync the user interface with the browser url which is achieved under the hood by React Router reading and manipulating the HTML5 history.
2. Use mainly the Router component to match a given route to a view to render.
3. Use the Switch component to render the first route that matches a given URL, or a page not found component in case of an unknown URL.
4. Use the Link component to allow the user to navigate through the application without reload it.

4.7. Testing

One of the most important parts of this project was the development of automated tests. The objective was to follow a Test-Driven-Development (TDD) approach following the Red-Green-Refactor cycle, as shown in the figure below:

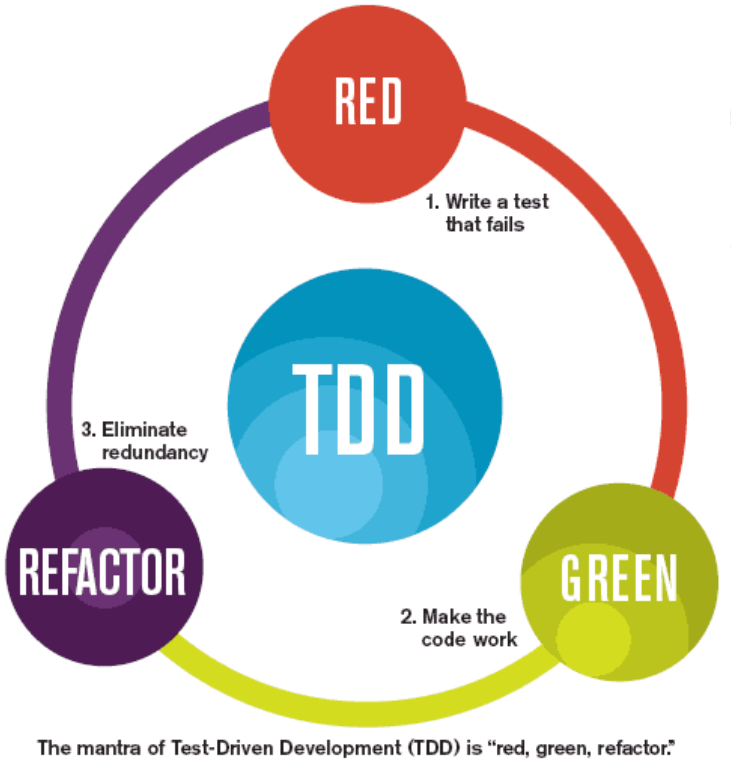

Image 22. Test Driven Development model lifecycle (Singh, 2017)

So, in the development of the different tasks and features of the project and iterative process was followed. As an example, in a given functionality of the project was required to detect if two dates in Unix format were from the same or different days. The process to develop it was:

1. Start writing the test following the format Arrange, Act, Assert (AAA), meaning that it had at the beginning the two dates to be tested, after which a function was called using them as parameters and finally and assertion of the result was made.
2. The first execution of the test was wrong (red) as the function was not implemented.
3. Hence, the next step consisted in the implementation of the functionality to pass the test (make it Green).
4. Finally, some more tests of edge cases were introduced and the necessary refactors were made to support all the scenarios.

This process was repeated fort the different functionalities, allowing the development of an application with an excellent test coverage, making the code more reliable and easier to maintain. If someday arrives another developer, changes some parts of the project and the functions start to return different values, the tests will catch immediately those differences and compel the developer to update them or correct the braking changes.

On the other hand, the end-to-end (e2e) tests have a different life cycle during the project. Meanwhile the unit tests are developed along with the functionalities, and according to the testing pyramid should be a lot, trying to cover the major quantity of functionalities, the e2e tests are developed at the end, and to test only the most important features because they are more difficult to develop and maintain.

5. Results

More than a specific solution for the proposed project, the results of this work are all the small experiments that have been done on the way. Through them, it has been possible to realize what works in this context, and what does not, helping in turn to arrive at the end not only to a first version of a new web application, but also to an overall development lifecycle, from a local environment to a testing one, based in quality principles through automation.

The first result of those experiments was the React starter-kit. It was the product achieved after the initial learning process in which we depart with a set of problems we wanted to address and come up with a possible (opinionated) solution to them as described along this work.

A second result, closely connected to the starter kit, and to the overall development of the project was the creation of a specific pipeline to check, build and deploy the code automatically and continuously. It has helped the whole development process, making easy to know objectively when something is working as it should be, and preventing the deployment of code that does not complain with the automated controls.

Finally, the end result was a new version of the tv guide that addresses the problems identified at the start of the project, in the way described along this work. Code quality, performance, automation, improved user experience, and responsiveness were all topics that came repeatedly through the project, until the desired results were achieved.

6. Conclusions

The end objective of any software project is to solve an existing problem. Beyond that, it is necessary to think about the solution as something that is not stationary, but that evolves over time. Therefore, it is important to understand what is going on, and to be flexible and adapt rapidly to the ever-changing necessities of the world.

The problem becomes even more complex when you consider that not only from the perspective of the product the requirements could change rapidly, but also the technology landscape moves with an unstoppable rhythm. In that context this project is born, trying to find the best solution at the time to the given problem, but taking into account that the most important thing is to be prepared for changing and evolving over time.

The first thing that has helped a lot in the process was the introduction of automated tasks to control the code quality. The use of linter tools makes the development slower at the beginning, as the developer should get familiarize to the different rules, and customize add or change what necessary. But once the linting constraints are interiorized, the development velocity gets back and the project can benefit from the early detection of programming errors, and from the homogenization of the style over the whole code base.

The same thing has happened with Flow, which at the beginning could slow down the development, but once accustomed, the benefits introduced with this tool are clear. Because from one side, a kind of documentation is built specifying what are the types of the variables, parameters, functions, components, etc.; and from the other, it helps to avoid lot of errors that other way are difficult to see before the application is running.

The only real problem that comes up using this tool was the inclusion of third-party libraries, as there is lot of code around that is not typed and even though it is possible to ignore the types of those libraries, it is not the best option. Then, the only alternatives are: write manually the “library definitions” (https://flow.org/en/docs/libdefs/) or use a tool called

"flow-typed" (https://github.com/flow-typed/flow-typed) that helps in finding those definitions. Unfortunately, it does not work in all the cases, making necessary to ignore or write manually the types of those libraries.

In overall, the static analysis helped to maintain the code clean and readable. Going further, the control of the code in a dynamic way was made through the writing of automated tests. Which was helpful in different ways: first of all, as source of documentation for functions, components, and even complete workflows in the case of integration or end-to-end test. Secondly, as quality assurance across the different environments, helping in the detection of breaking changes before the deployment of the application. And finally making easier the refactor of the code, as the correct working of the project is controlled all the time in an automated way. Nevertheless, it is important to remark that all these benefits are conditioned by the quality of the tests and its coverage of the code.

Finally, the real value of these tasks became even more clear once the continuous integration / delivery (CI/CD) pipeline was introduced. Despite the fact it is a simplified version of a real enterprise ready workflow, it has helped to understand the benefits of automation based on quality principles.

Up to this point the conclusions that have been presented are about how the introduced practices have benefited the development process, making it more dynamic, agile, and versatile; In the next paragraphs, the conclusions regarding the actual development will be presented.

Work with React is mostly about writing the traditional HTML, CSS, and JavaScript, but organizing them in a different way. Which makes starting with React easier than with other frameworks, in which it is necessary to learn the specific syntax to get started. Moreover, the component approach that is encouraged by using React helps the applications to be more declarative and readable.

But not only the code is better organized, also the performance gets benefit from the features of React. For example, the use of a virtual DOM, allows changing data and update only the related parts of the web page. Or going a step further, the use of the windowing technique makes possible to render instantly the visible part of the grid and avoid laggings on scroll, which definitively improve the user experience.

Finally, the actions taken regarding the user interface made the whole grid more readable and navigable, which also allowed to find some errors that were not evident in the current tv guide. Allowing, consequently, their correction as part of the development of the project. Moreover, the single page approach adopted for the development, helped improve the overall user experience, making possible for example, changing or filtering the events data without the refreshment of the whole page.

7. Future Work

The work presented in this report has gathered the effort mainly in the development practices, the performance and the user experience of the application. Nevertheless, there are still lot of things that could be done. Regarding the CI/CD pipeline, the definition of more environments with different purposes could improve the robustness of the process and make possible the addition of the continuous deployment step to the existing pipeline.

On the other side, from the development part, could be interesting to explore the possibilities offered implementing things like server side rendering. It could be really useful especially for the initial loading of the application, which is the most critical point at the time. Furthermore, it is important to implement server side the proposed data model, and consider the implementation of a backend for frontend, as described previously in this work.

8. References:

Agilemanifesto. (2001). Manifesto for Agile Software Development. Retrieved from http://agilemanifesto.org/

css-modules/css-modules. (2018). Retrieved from https://github.com/css-modules/css-modules

Cypress. (2018). Testing has been broken for too long. We figured it was time to fix it. Retrieved from https://www.cypress.io/how-it-works/

Fowler, M. (2012). TestPyramid. Retrieved from https://martinfowler.com/bliki/TestPyramid.html

Frost, B. (2013). Atomic Design. Retrieved from http://bradfrost.com/blog/post/atomic-web-design/

Hunt, P. (2013). React: Rethinking best practices. Retrieved from https://www.youtube.com/watch?v=x7cQ3mrcKaY

Jeffries, R. (2011). What is Extreme Programming?. Retrieved from https://ronjeffries.com/xprog/what-is-extreme-programming/

Kemp, P., & Smith, P. (2010). Waterfall model. Retrieved from https://en.wikipedia.org/wiki/Waterfall_model

Newman, S. (2015). Backends For Frontends. Retrieved from https://samnewman.io/patterns/architectural/bff/

Rastelli, C. (2017). Let There Be Peace On CSS. Retrieved from https://speakerdeck.com/didoo/let-there-be-peace-on-css?slide=62

React Documentation. (2018). Optimizing Performance – React. Retrieved from https://reactjs.org/docs/optimizing-performance.html

React Documentation. (2018). Thinking in React – React. Retrieved from https://reactjs.org/docs/thinking-in-react.html

Saring, J. (2017). 11 React UI Component Libraries You Should Know In 2018. Retrieved from https://blog.bitsrc.io/11-react-component-libraries-you-should-know-178eb1dd6aa4

Scrum. (2018). What is Scrum?. Retrieved from https://www.scrum.org/resources/what-is-scru

Singh, N. (2017). Test-driven development might seem like twice the work — but you should do it anyway. Retrieved from https://medium.freecodecamp.org/isnt-tdd-test-driven-development-twice-the-work-why-should-you-care-4ddcabeb3df9

T, F. (2017). How linting and ESLint improve code quality. Retrieved from https://hackernoon.com/how-linting-and-eslint-improve-code-quality-fa83d2469efe

Thoughtworks. (2018). Technology Radar | Emerging Tech Trends for 2018 | ThoughtWorks. Retrieved from https://www.thoughtworks.com/radar/languages-and-frameworks/css-in-js

Vaughn, B. (2017). Creating More Efficient React Views with Windowing. Retrieved from https://www.youtube.com/watch?v=t4tuhg7b50I&t=503s