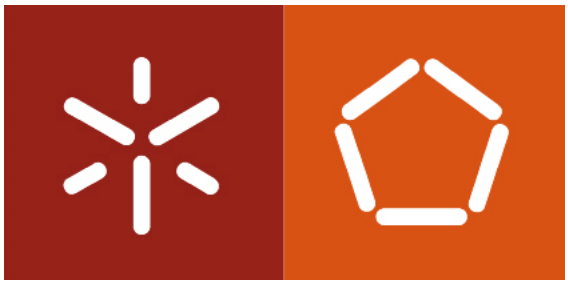
UNIVERSIDADE DO MINHO



MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

DISSERTAÇÃO

Modern Front-End Web Development

Author: António Manuel Pereira do Anjo A67660
Supervisors: Rui Couto, José Creissac Campos

October 28, 2018

Acknowledgements

I would like to firstly thank my supervising professors José Creissac Campos and Rui Couto for their availability and eagerness to help with their knowledge and experience during the work period.

Secondly, I would like to thank all the Jumpseller team members that I worked with, who have since the first day made me feel like part of the team and who I have been lucky to learn from and become friends with.

I would finally like to thank my family, my girlfriend Sara and my good friends José Francisco, João Miranda and Rui Pereira for the continued support over the years.

Abstract

The Internet is always evolving. The way content is generated, displayed and accessed is constantly changing with the advancements of technology.

As such, new tools, frameworks and technologies are constantly surfacing as a way to deal with the challenges of this evolution. Keeping up with an increasingly large number of options is, for Web Developers, as important as it is challenging.

With this challenge in mind, this dissertation aims to offer a deep look into the current state of Front-End Web Development by going through relevant concepts and doing an in-depth, comparative analysis of the different frameworks. In the process, data will be collected, a case study will be developed and a developed approach will be validated, thus obtaining results and taking conclusions that will help make the best possible decisions in the development process.

The author will be, during the work period, part of the Developer Team at Jumpseller, working hands-on with these technologies.

Resumo

A Internet está sempre a evoluir. A forma como o conteúdo de uma página é criado, visualizado, e acedido está constantemente a alterar graças aos avanços tecnológicos.

Como tal, existem novas ferramentas, frameworks, e tecnologias em constante surgimento, oferecendo métodos para lidar com os desafios desta evolução. O desafio de se manter a par de um número cada vez maior de opções é, para os Web Developers, algo tão importante como é exigente.

Com este desafio em mente, o objetivo desta dissertação é oferecer uma análise a fundo para o atual estado da arte do Desenvolvimento Web Front-End, percorrendo vários conceitos relevantes e realizando uma análise comparativa detalhada sobre as várias frameworks. Para atingir este fim, dados serão obtidos, será desenvolvido um caso de estudo e uma abordagem será desenvolvida e validada, obtendo assim resultados e tirando conclusões que irão auxiliar a tomada de decisões no processo de desenvolvimento.

O autor irá integrar, durante o período de trabalho, a equipa de desenvolvimento na Jumpseller, trabalhando diretamente com estas tecnologias.

Contents

List of Figures

List of Tables

List of Abbreviations

API	**A**pplication **P**rogramming **I**nterface
CLI	**C**ommand **L**ine **I**nterface
CSS	**C**ascading **S**tyle **S**heets
DOM	**D**ocument **O**bject **M**odel
HTML	**H**yper**T**ext **M**arkup **L**anguage
IDE	**I**ntegrated **D**evelopment **E**nvironment
SaaS	**S**oftware **a**s **a** **S**ervice
SDK	**S**oftware **D**evelopment **K**it
SEO	**S**earch **E**ngine **O**ptimization
UI	**U**ser **I**nterface
UX	**U**ser e**X**perience

Chapter 1

Introduction

For as long as the Internet has existed and visiting or creating a website has been something that is available to anyone, the look, design and usability of a web page has been a very relevant aspect of web development and a major aspect on whether or not people are going to want to use a website (Flavián, Guinalíu, and Gurrea, 2006).

While this fact has maintained itself throughout the years, its meaning in terms of what developers need to be able to achieve has always been changing as the Web itself changes and evolves.

Concepts such as the Web 2.0[1] emphasized that a page's content is, nowadays, rarely ever static. It can be dynamic and user-generated, gathering information from other places and displaying it as its own, with focus on usability and interoperability.

A computer is no longer the only way to view a website. Phones, Tablets or even Smart Watches exist now as a very common way for people to access their favorite websites and most needed services. This means that web pages have to be developed in such way that they can adapt to different user experiences.

For reasons such as these, many technologies have surfaced over the years. In different ways, shapes or forms, these technologies have been aiming to solve the challenges of the ever-evolving Web.

Facing issues like device compatibility and cross-platform development, these technologies strive to create simple ways for developers to build their projects in a way where they can focus on their own specific requirements, without having to worry about general issues. This is, in theory, a good thing for developers, as it makes the development process faster and easier. But as more and more of these technologies come to life, the more the issue of choosing which ones to use for each project becomes prevalent.

As such, this dissertation aims to investigate the currently most popular technologies related to Front-End Web Development in its current state, presenting conclusions about the strengths and weaknesses of each of them.

During the work period, the author of this thesis will be part of the Jumpseller[2] development team. As part of this internship, the author will work with front-end technologies and directly in a Web Development environment. This means that the work done as part of the Jumpseller team will be strongly tied to the scientific aspect of this dissertation, as the experience gathered will help to have a better understanding of the state of the art, and the extensive research will help make sure the best decisions are made, both during the development processes the author will be a part of and during the case study development and implementation.

[1] https://en.wikipedia.org/wiki/Web_2.0 - Web 2.0 (Accessed September 11, 2018)
[2] https://jumpseller.com/ - Cloud Ecommerce Solution (Accessed September 11, 2018)

1.1 Context

Throughout the work period, several projects will be developed within the scope of the Jumpseller product (the logo is presented in Figure 1.1), most of which consisting in the development of Web Interfaces, making use of technologies that aim to improve the user experience of the application.

The developed components and interfaces will be integrated in the production build of the Jumpseller application. This will be achieved in collaboration with the company's development team.

FIGURE 1.1: The Jumpseller logo.

Jumpseller is a SaaS (Software as a Service) solution, from Widetail[3], for small businesses to easily create an online store. It aims to make e-commerce easier for everyone, so that businesses can focus on what they do best: building and selling their products.

Today, merchants use the Jumpseller platform to manage every aspect of their online business — from products to orders to customers, selling online, on mobile, and on social networks.

Upon creating a store in the Jumpseller platform, store owners can then manage it on its administration panel, a web application consisting of different, often complex sections that enable the management and customization of various elements of their store and brand - from the products to orders to customers, changing the store appearance, selling on social networking platforms and integrating with a variety of payment and shipping methods.

1.1.1 Overview

Aimed at Small and Medium-Sized Enterprises, Jumpseller has clients spread all over the world, with a large percentage of those coming from Chile and other South American countries. Table 1.1 shows the country distribution of its clients as of December 2017, the time of writing of this section of the present document.

Chile	65%
Portugal	10%
Colombia	8%
Mexico	7%
USA	1.5%
Brazil	1%
Others	7.5%

TABLE 1.1: Jumpseller User Locations

[3] http://widetail.com/ - Widetail (Accessed Semptember 11, 2018)

As of this time, the percentage of clients coming from Chile has been decreasing, as Jumpseller keeps growing and gaining popularity in other countries. It is expected that in the next few years more than 50 percent of the clients will be from outside of Chile.

In November 2017, Jumpseller hit the milestone of 1000 active paying stores.

1.1.2 Motivation and Challenges

With a large amount of users - store owners and their clients - interacting with the Jumpseller application, it is of utmost importance that an appropriate, pleasing user experience is delivered to all of them.

This can be a challenging thing to achieve, as a large number of users implies a wide variety of device types - desktop computers, mobile phones, etc - , a varying degree of technical proficiency of the users - age and education levels of the users may influence this - as well as other factors such as the computational power that the average user from a certain country may have access to or the languages they speak. Aspects such as these come as a result of the variety of countries where Jumpseller is present.

As such, front-end work on the Jumpseller platform will revolve around the following principles.

- **Ease-of-Use** Some of the web components currently in use are old, outdated and generally don't provide an optimal user experience. This comes as a conclusion of the feedback obtained by the application's user base and the number of support tickets received over the years. As such, it's in the company's interest that they are improved or replaced in order to guarantee interfaces that are clean, free of unnecessary steps or annoyances and that are as simple as possible.

- **Device Compatibility** Similarly, some of the functionalities found throughout the platform work very poorly or not at all on mobile devices. Making sure all functionality and presentation quality is kept across different devices is a priority.

- **Lightweight Approach** The Jumpseller application should work well for everyone, regardless of their device's computational power or average Internet speed. This means that high performance should be a factor in choosing the right approach to development.

The current problem with the platform is that many of the sections don't respect these principles: Some are poorly designed and coded, with a difficult-to-understand flow and confusing layouts. Others don't function well - or at all - on mobile devices and some pages load unused code increasing the application's loading times unnecessarily. This may be a result of the current code having been written without these principles in mind and without there having been a proper technology selection methodology prior to their implementation.

During the work period, the author will be working on making the application a better user experience by firstly analyzing the current implementations and identifying their problems and, on a second phase, working closely with the researched technologies and using the most appropriate ones with these key principles in mind.

1.2 Objectives

The following objectives have been set for this dissertation.

- State of the Art Analysis
  - Analysis and comparison of the most popular available front-end technologies.
- Approach Development
  - Deep comparative analysis of the selected technologies, evaluating strengths, weaknesses and how they respond to common problems.
  - Present an initial idea of an approach which can be used to facilitate the process of choosing the right front-end technologies for a web development project.
- Case Study Development
  - Definition of a case study to be used in validating the proposed approach.
  - Propose a technological approach to the defined problem.
  - Development of web interfaces and components using the researched technologies.
- Approach Validation
  - Integration of the developed components with the enterprise back-end, with help from the Jumpseller team.

1.3 Document Structure

This dissertation has the following document structure.

- **Chapter 1 - Introduction** Briefly present the context and motivation behind this dissertation, as well as its main objectives and structure. Describe Jumpseller and its functionalities, user base and current issues.
- **Chapter 2 - Technology and Concepts** Further contextualize the work that will be done, by going over the most important concepts and technologies.
- **Chapter 3 - Framework Analysis and Approach Development** Present usage data and make a brief analysis of the most popular technologies in the front-end development field, taking conclusions about them. Present an initial idea of an approach in the form of a diagram.
- **Chapter 4 - Case Study** Define and implement a case study using the research technologies, obtaining a practical experience and allowing for a more detailed comparison between them and for validation of the proposed approach.
- **Chapter 5 - Validation - Jumpseller** Portray the author's experience as part of the Jumpseller development team, going over the work methodology, the projects taken part of and using the experience to further validate the proposed approach.

- **Chapter 6 - Conclusions** Provide conclusions about the work done on the dissertation, presenting a summary of what was accomplished and reflecting on what can be done in the future.

Chapter 2

Technology and Concepts

This chapter sets the context in which this dissertation is included, by providing some key ideas and concepts about the current state of Web Development and Front-End technologies.

Here, concepts relevant to this work such as Frameworks and Responsive Web Design will be presented so that they are further explored later on in the document.

2.1 Web Applications Development

The **front-end** and the **back-end** are familiar concepts to any software developer, as they are both essential components of any application, even if their distinction is not always clear.

As such, the first thing that should be made clear before delving into further concepts is the distinction between **Front-End** and **Back-End** in the context of the development of Web Applications.

2.1.1 Web Applications

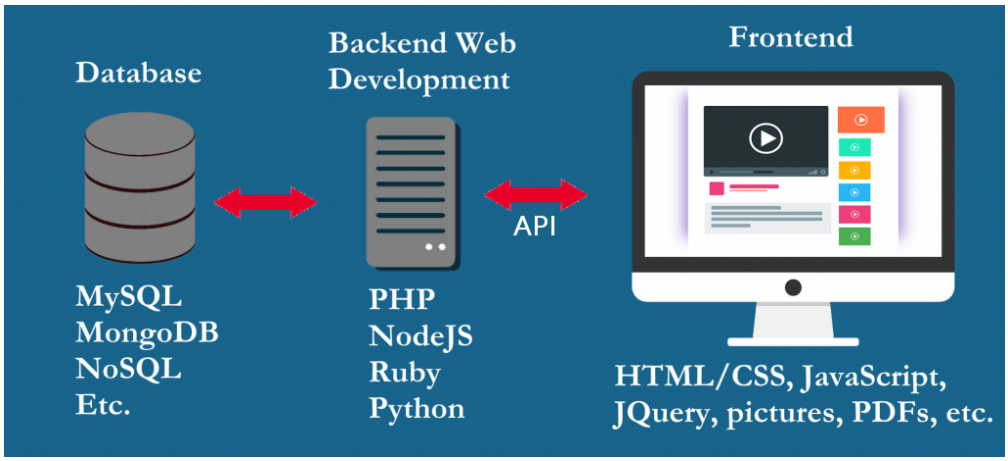


FIGURE 2.1: Front-End and Back-End.

Front-End

Front-End (see Figure 2.1, Frontend) refers to the parts of an application which exist and run on the client-side and are responsible for creating and supporting the elements visible to the end user, representing its visual and interactive components. In

web development, this takes the form of a website. Front-end covers how the content is presented and includes all the interface elements that are displayed, such as menus, transitions, dropdowns or modals (Nice, 2017). This part of web development is often referred to as 'client-side', as the code runs on the client's browser - as opposed to running on the server.

HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) (Meyer, 2006), DOM (Document Object Model), JavaScript (Crockford, 2008), HTTP (HyperText Transfer Protocol)[1], and browser skills such as debugging and the browser's console are assumed for any type of front-end developer (Linley, 2017). In addition to fluency in these languages and concepts, front-end developers generally need to be familiar with frameworks and libraries that ensure responsive, usable content on all devices that is a result of code that respects the principles of these practices.

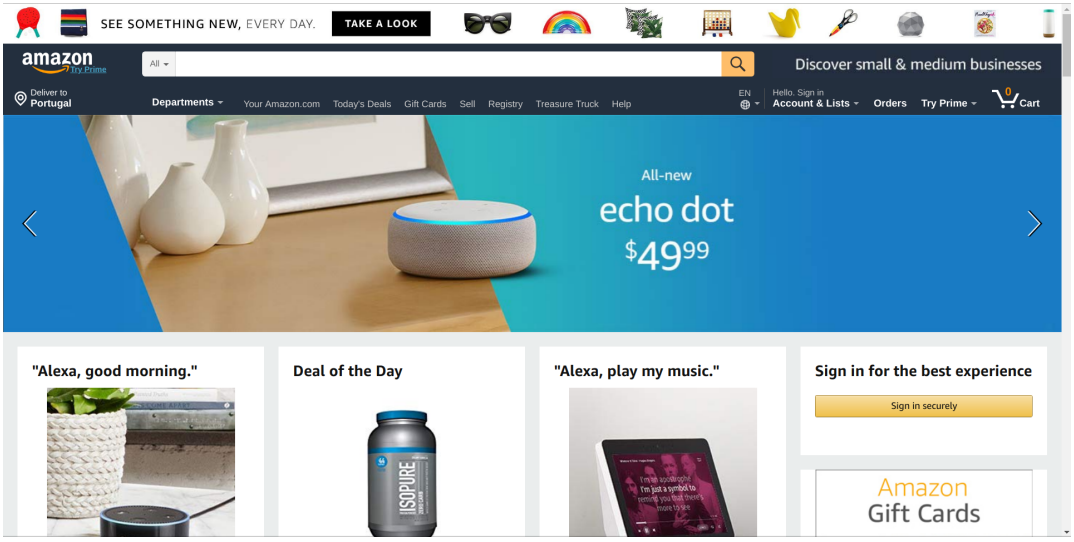


FIGURE 2.2: The Amazon front-page.

Figure 2.2 shows the front-page of the Amazon[2] website. Front-end elements such as images, links, buttons and input fields show how content is generated, displayed and how it is built to react to the user's actions.

Back-End

Back-End (see Figure 2.1, Backend) refers to the part of the application that is not visible to the end user and which lives and runs on the application server - where the application runs and services such as database operations are provided. As such, it is often called 'server-side'. The back-end of an application is responsible for the application logic, which includes features such as calculations, data management (c.f. databases) and providing APIs (Application Programming Interfaces, see Figure 2.1, API). The interaction between the front-end and the back-end makes it possible for the right data to be sent to the browser and for the client's actions to be interpreted and stored by the application and its database.

Back-End developers will generally have expertise in server-side languages such as Ruby(Thomas, Hunt, and Fowler, 2005), Python[3], PHP[4] or Java(Sierra and Bates,

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP - HTTP (Accessed September 11, 2018)
[2] https://www.amazon.com/ - Amazon (Accessed October 5, 2018)
[3] https://www.python.org/ - Python (Accessed September 2, 2018)
[4] http://www.php.net/ - PHP : Hypertext Processor (Accessed September 2, 2018)

2005) and the frameworks that run on them, as well as in database management systems such as MySQL(MySQL, 2001), Oracle[5] or MongoDB[6].

Both of these, as well as a way of communication between them, are essential for the functioning and represent the base components of a web application. This relationship is shown on Figure 2.1. The communication is asynchronous (meaning that the front-end application makes a request and waits for a response) and generally done using HTTP (R. Fielding, 1999) requests, such as GET, POST or PUT to, respectively, retrieve, add or alter data.

A common format used for passing data in this type of communication is JSON[7] - JavaScript Object Notation. This is a light-weight data interchange format that is easily readable to humans and easy for machines to parse and generate.

It should be noted that, along with these terms, it's common to find a third type of Web development, called **Full-Stack** (Ihrig and Bretz, 2014). A Full-Stack developer is one who is comfortable with both the client-side and the server-side aspects of development and can work with both. Full-Stack frameworks also exist and attempt to provide nearly everything needed to build and application.(Nice, 2017)

2.1.2 Web Pages

FIGURE 2.3: HTML, CSS and JavaScript.

HTML, CSS and JavaScript are the foundation of Front-End Development, and work together to create, style and set the behavior of a web page. They are commonly represented by the logos shown on Figure 2.3.

HTML

HTML is the most basic building block of the Web. It describes and defines the content of a web page. Its current latest major version is HTML5. (Mozilla, 2017b)

HTML uses markup - a system for annotating a document in a way that is syntactically distinguishable from the text [8] - to annotate text, images, and other content for display in a Web browser. HTML markup includes special "elements" such as

[5]https://www.oracle.com - Oracle (Accessed September 2, 2018)
[6]https://www.mongodb.com/ - MongoDB (Accessed September 2, 2018)
[7]https://www.json.org/ - JSON (Accessed September 2, 2018)
[8]https://www.merriam-webster.com/dictionary/markup+language - Markup Language Definition on Merriam-Webster (Accessed Sempteber 11, 2018)

`<head>`, `<title>`, `<body>`, `<header>`, `<footer>`, `<article>`, `<section>`, `<p>`, `<div>`, `<span>` or `<img>`.

HTML can, by itself, define a layout of a website and its sections, but other technologies are generally used to describe a web page's appearance/presentation (CSS) or functionality/behavior (JavaScript).

CSS

CSS is a stylesheet language used to describe the presentation of a document written in HTML or XML. (Mozilla, 2017a)

The current latest version, CSS3, is in the process of becoming the standard in Front-End Development, as it comes with optimizations and is split into smaller modules.

CSS works together with HTML by describing how elements should be rendered on screen. It uses properties such as *color*, *background-color*, *height*, *width* and many more.

```
header {
    background-color:red;
}
```

LISTING 1: CSS Example

For example, having an HTML page that contains a `<header>` tag and includes the CSS code snippet in Listing 1 will result in the header section of the page having a red background.

JavaScript

JavaScript is an interpreted programming language with object-oriented capabilities.

It is most commonly used in web browsers and, in that context, its general purpose core is extended with objects that allow scripts to interact with the use, control the web browser, and alter the document content that appears within the web browser window. This embedded version of JavaScript runs scripts embedded within HTML web pages. It is commonly called *client-side* JavaScript to emphasize that the scripts are run by the client computer rather than the web server.

Client-side JavaScript combines the scripting ability of a JavaScript interpreter with the DOM defined by a browser. HTML documents may contain JavaScript scripts - using the `<script>` tag - and those scripts can use the DOM to modify the document or control the web browser that displays it (Flanagan, 2006).

In sum, client-side JavaScript adds behavior to otherwise static web content.

2.2 Responsive Web Design

The concept of **Responsive Web Design** (Bryant and Jones, 2012) or **Responsive** relates to the practice of building websites which are capable to adapt to the screen size of the device accessing it, as shown on Figure 2.4.

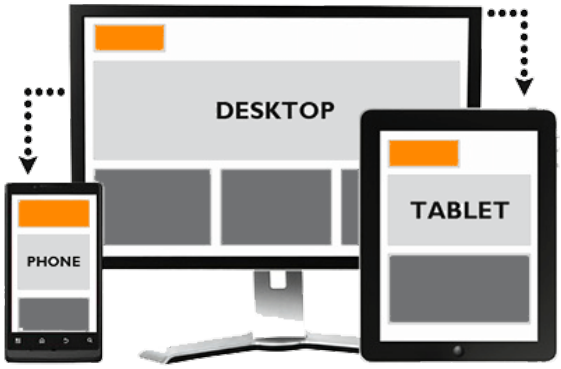


FIGURE 2.4: Responsive Web Design.

Its implementation revolves around some key ideas: Grid-based flexible layouts, flexible content with dynamic resizing and media queries.

2.2.1 Flexible Layout and Content

In *responsive*, layout and text sizes are not expressed in pixels. Instead, relative measurements are used such as **percentages**, **em** (size relative to the font-size of the element), **rem** (size relative to the font-size of the root element), **vw** (size relative to the width of the viewport[9]) or **vh** (size relative to the height of the viewport).

Through this approach it is possible that all the components of the web page can scale appropriately when the screen size changes, as seen on Figure 2.4. If absolute measurements were used instead, the elements would remain the same size and their readability and functionality would be compromised.

[9]Viewport - The browser's window size.

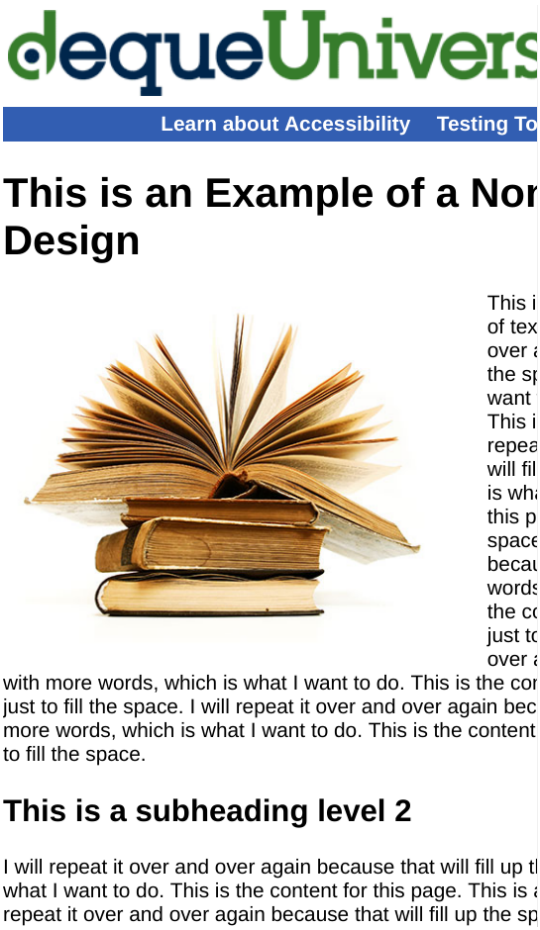


FIGURE 2.5: Example of a non-responsive Web Design.

Figure 2.5 shows an example of a website that is not responsive which, when viewed on a smaller screen, its content becomes illegible.

Flexbox

The Flexible Box Module, usually referred to as flexbox, aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic (Coyier, 2018).

Development with flexbox starts by creating a **flex container**: an HTML element with the `display:flex` CSS property. This establishes the formatting context for its contents and enables flexbox layout configuration on itself and its child elements.

Beyond establishing the context, several other CSS properties allow for deeper configuration of the created container and the layout of its contents. This includes, for example, `flex-direction` which defines in which direction the container wants to stack the flex items (row or column, shown in Figure 2.6) or `align-items` which is used to align the flex items vertically in a variety of different ways such as center, the top of the container or using a common baseline.

Additionally, there are properties which can be applied to the flex items, i.e. the child elements of the container, such as `flex-grow`, shown on Figure 2.7, or `flex-shrink` which specifies how much a flex item will shrink relative to the rest of the flex items.

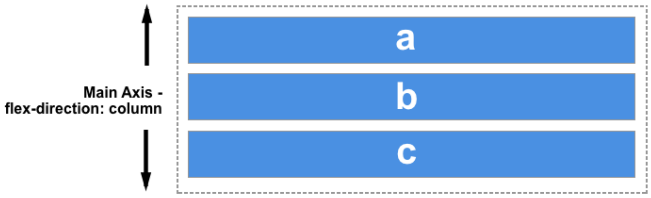


FIGURE 2.6: flex-direction

FIGURE 2.7: flex-grow

This module is used by the Bootstrap[10] framework to help create the popular grid-based layout.

2.2.2 Media Queries

Media Queries are a key component of responsive design (Mozilla, 2017c) and consist of expressions which verify certain conditions of the page, applying different CSS rules for each scenario.

This enables having a specific CSS rule set for certain screen sizes, thus supporting many different screen types and sizes without ever changing the actual content of the web page.

For example, the media query in Listing 2 checks the screen size and applies the `padding` property on the `medium` element only if the condition is met, i.e., if the screen's width is a value between 760 and 1080 pixels.

```
@media screen and (max-width: 1080px) and (min-width: 760px) {
    .medium {
        padding: 30px;
    }
}
```

LISTING 2: Media Query Example

Besides the screen size, media queries can adapt to different scenarios regarding screen resolution, orientation, light level and more[11].

The combination of these concepts creates a strong solution to build dynamic, responsive web applications (Sampaio, 2013).

[10] https://getbootstrap.com/ - Bootstrap (Accessed October 19, 2018)

[11] https://developer.mozilla.org/en-US/docs/Web/CSS/@media#Media_features - Media Features on Mozilla (Accessed Semptember 11, 2018)

Grid Systems

A **Grid** is a two-dimensional structure used to vertically and horizontally structure a page's content in rows and columns. For instance, Figure 2.8 shows the 12 column container, where the header occupies the full width (12 columns), the main content 8 columns and the sidebar the remaining 4 columns. Figure 2.9 shows a similar representation, where the different rows are composed of a different amount of columns, depending on the type of layout desired.

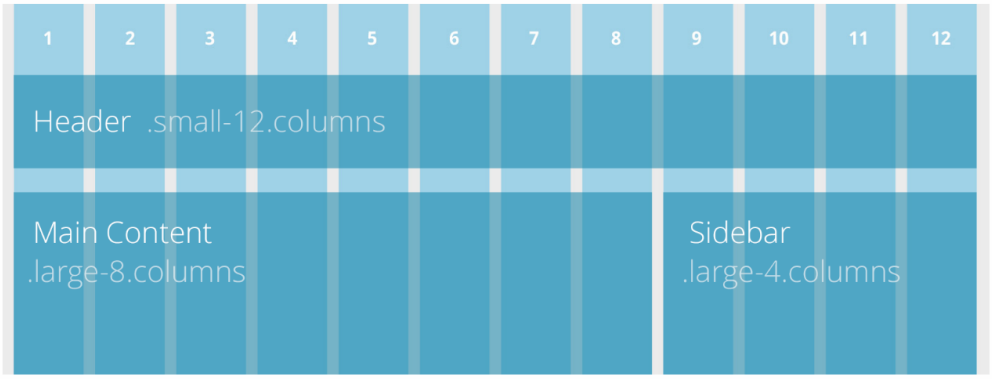


FIGURE 2.8: Foundation's grid system example

In frameworks such as Bootstrap or Foundation[12], the grid is composed by 12 columns that can scale to an arbitrary size based on their max-width, defined as a percentage of the screen size.

The use of responsive grids which scale with the window size is a key component of responsive web design, as it allows otherwise static component to be displayed in a fluid, scalable component.

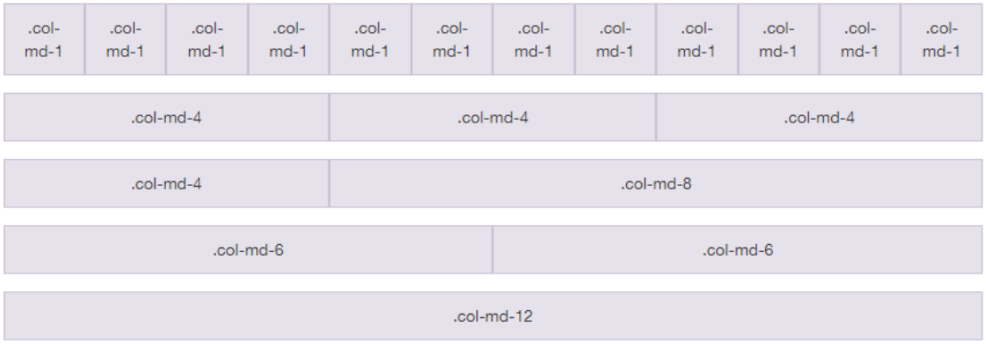


FIGURE 2.9: Example of a grid-based page layout using Bootstrap

[12]https://foundation.zurb.com/ - Foundation (Accessed October 19, 2018)

2.3 Frameworks

In general, the term **framework** describes a real or conceptual structure which serves as a base or guide towards building a product that expands it into something useful.

In Web Development, a large number of frameworks exist as a way to solve commonly faced problems and to provide a way to make development easier and more straight-forward, in such way that developers can focus on creativity and the specifics of their projects.

Next follows the description of two kinds of web frameworks - Styling and Behavior driven - where a brief introduction and comparison of each is made.

2.3.1 Styling and Behavior

The distinction between the styling and behavior aspects of front-end development means that the catalog of front-end frameworks can be divided into primarily visual-oriented frameworks and primarily behavioral or building frameworks - even if some frameworks include both style sheets and scripts.

Both kinds of frameworks can coexist within an application and will be considered in this analysis.

CSS Frameworks

A **CSS Framework** is a set of pre-prepared CSS styles that aim to easily provide the developer a set of well-designed, standard-compliant layouts or components, usually in line with the principles of Responsive Web Design. Some of the most popular CSS frameworks include *Bootstrap* and *Foundation*.

More functional frameworks - such as Bootstrap - also come with some JavaScript based functions and features, although these are generally design-oriented. This includes components like Dropdowns or Modals.

JavaScript Frameworks

JavaScript Frameworks, such as *React (Gackenheimer, 2015)*[13] or *Vue*[14], while all with their own specific features and quirks, aim to offer ways for developers to efficiently develop their front-end, with focus on metrics such as approachability, versatility and performance.

This is often achieved by providing unique code syntax, templates and re-usable components.

The two code examples shown on figures 2.10 and 2.11 demonstrate a “Hello World“ in React and in Vue, showing how these frameworks work differently and with different syntax to achieve a similar end - in this case, a blank page with the text *Hello World!*.

[13] https://reactjs.org/ - React (Accessed September 11, 2018)
[14] https://vuejs.org - Vue.js (Accessed September 11, 2018)

FIGURE 2.10: "Hello World" in React.

FIGURE 2.11: "Hello World" in Vue 2.0.

2.3.2 Node.js

Node.js (Tilkov and Vinoski, 2010) is a JavaScript runtime built on Chrome's V8 JavaScript engine[15]. This means that the Node run-time environment includes everything you need to execute a program written in JavaScript[16]. Its logo is shown on Figure 2.12.

[15]https://nodejs.org/en/ - Node.js (Accessed September 3, 2018)
[16]https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5 - What exactly is Node.js? (Accessed September 3, 2018)

FIGURE 2.12: The Node.js logo

Node Modules

Modules in Node.js can be considered the same as JavaScript libraries, and represent a set of functions one wishes to include in an application.

Node.js has a set of built-in modules which can be used without any further installation, and additional modules can be imported to an application using the `require()` function.

npm

npm, the Node Package Manager, makes it easy for JavaScript developers to share and reuse code[17] as well as managing dependencies within a project. It can be installed alongside Node.js and commands such as `npm install` or `npm update` can be used to integrate new packages into an existing application or update all the existing ones to their latest versions.

2.4 Summary

In this chapter some of the key concepts of modern web development such as Front-End, Responsive Web Design, Frameworks and the technologies that come with them have been presented, providing technological context for the next sections, where the most important front-end development frameworks will be analyzed in higher detail and put to test through a case study. These concepts and technologies will also be important on relating the work projects the author will work on as part of the Jumpseller team.

[17] https://www.npmjs.com/get-npm - npm (Accessed September 3, 2018)

Chapter 3

Framework Analysis and Approach Development

This chapter presents an overview of the most relevant Front-End frameworks, considering usage data as the major factor for selecting the set of frameworks for analysis.

This will define the basis for the analysis process to be developed on the later stages of this dissertation.

Then, a comparative analysis of the frameworks is done, in order to develop an approach to the issue of selecting the right technologies for a given project.

3.1 Overview

To get an idea of what are, as of December 2017 - the time of writing of the present document - the most used web development frameworks, GitHub[1] data was collected and is displayed on Table 3.1 and Table 3.2. The number of stars and forks is shown for each framework, respectively indicating GitHub users that liked and copied the source code of each project.

The first selection of frameworks to be evaluated was made using popularity and a GitHub collection of front-end JavaScript framework projects [2] as criteria.

	Stars	**Forks**
Bootstrap	119k	56k
Semantic-UI	38k	4k
Material-UI	31k	5k
Materialize	30k	4k
Foundation	26k	5k
Bulma	22k	1k

TABLE 3.1: CSS frameworks GitHub data as of December 2017

It can be concluded from this data that, looking at the CSS frameworks and based on the defined criteria, Bootstrap is by far the most popular, with a sizable lead over its contestants.

[1] https://github.com/ - GitHub (Accessed September 11, 2018)

[2] https://github.com/collections/front-end-javascript-frameworks - Collections : Front End JavaScript Frameworks (Accessed October 5, 2018)

	Stars	Forks
React	83k	15k
Vue.js	76k	11k
Angular.js	57k	28k
Angular	31k	7k
Backbone	26k	5k
Ember.js	18k	3k

TABLE 3.2: JavaScript frameworks GitHub data as of December 2017

In the JavaScript category on the other hand, numbers are distributed slightly more evenly, with React and Vue (the youngest of the three, having started in 2014) closely on the top spots, and the two versions of Angular - Angular.js (the oldest of the three, started in 2009) and Angular - Formerly known as Angular 2 on the next positions.

In the following sections a deeper look will be taken at the most popular frameworks in each category, evaluating its strengths and weaknesses and taking conclusions about what makes them the most popular. These sections will aim to compare the frameworks in terms of what their main features are, how certain common operations can be performed and their strengths and weaknesses.

A look will be taken at the main features that make Bootstrap the most popular CSS framework as well as how it handles common problems and, on JavaScript frameworks, a common point of comparison will be how **data-binding** works. This refers to how data stored in the JavaScript variables or objects can be used in the HTML template to be presented to the end user. Along with this, a small example of a component or page built using each framework will be presented on its respective section.

3.2 Bootstrap

FIGURE 3.1: The Bootstrap logo.

Originally created by a designer and a developer at Twitter, Bootstrap has become one of the most popular front-end frameworks and open source projects in the world. Since its first release in 2011, it has had multiple major updates and versions. The current latest version is a beta build of Bootstrap 4.1 and its logo is shown in Figure 3.1.

Bootstrap is an open source toolkit for developing with HTML, CSS, and JavaScript, which allows developers to quickly prototype their ideas or build entire apps with

Sass[3] variables (further explained on Section 3.2.1), responsive grid system, extensive pre-built components, and powerful plugins built on jQuery[4], a JavaScript library designed to simplify the client-side scripting of HTML.

3.2.1 Sass

Since the first beta release of Bootstrap 4, the framework has been built on **Sass**. Sass is a CSS extension language which strives to add power and elegance to the basic language.

It does so by allowing the use of variables, nested rules, mixins and inline imports. This helps keep large style sheets well-organized, and get small style sheets up and running quickly.

Sass code, written on a *.scss* file is then compiled to plain CSS to be interpreted by the browser.

```
#main {
  width: 97%;

  p, div {
    font-size: 2em;
    a { font-weight: bold; }
  }

  pre { font-size: 3em; }
}
```

LISTING 3: Nested CSS in Sass

The code example on Listing 3 shows the use of nested rules in Sass, and compiles to the plain CSS in Listing 4.

```
#main {
  width: 97%; }
  #main p, #main div {
    font-size: 2em; }
    #main p a, #main div a {
      font-weight: bold; }
  #main pre {
    font-size: 3em; }
```

LISTING 4: Plain CSS compiled from Sass

Note that the Sass snippet on Listing 3 was written on the SCSS syntax, which is similar to regular CSS. There is also the option of using the SASS syntax, which is more concise and uses indentation rather than brackets to indicate nesting of selectors, and newlines rather than semicolons to separate properties. The fact that it's more similar to regular CSS, however, makes SCSS a more popular choice.

[3] http://sass-lang.com/ - Sass: Syntactically Awesome Style Sheets (Accessed September 11, 2018)
[4] https://jquery.com/ - jQuery (Accessed September 11, 2018)

For Bootstrap, Sass brings advantages with its use of variables. This means that users can customize Bootstrap with a built-in custom variables file and easily toggle CSS preference with the `$enable-` Sass variables. Some of these variables are depicted in Figure 3.2.

Variable	Values	Description
`$spacer`	`1rem` (default), or any value > 0	Specifies the default spacer value to programmatically generate our spacer utilities.
`$enable-rounded`	`true` (default) or `false`	Enables predefined `border-radius` styles on various components.
`$enable-shadows`	`true` or `false` (default)	Enables predefined `box-shadow` styles on various components.
`$enable-gradients`	`true` or `false` (default)	Enables predefined gradients via `background-image` styles on various components.

FIGURE 3.2: Some of the Sass variables in Bootstrap 4.

3.2.2 Responsive Grid

Bootstrap uses a powerful mobile-first flexbox grid to build layouts of all shapes and sizes thanks to a twelve column system and five default responsive tiers for screen sizes. **Rows** and **Columns** are used as classes to set the grid layout of a page.

For example, the HTML code on Listing 5, using Bootstrap 4:

```
<div class="container">
  <div class="row">
    <div class="col-sm">
      One of three columns
    </div>
    <div class="col-sm">
      One of three columns
    </div>
    <div class="col-sm">
      One of three columns
    </div>
  </div>
</div>
```

LISTING 5: Bootstrap Grid Layout Example

Results in the layout shown in Figure 3.3.

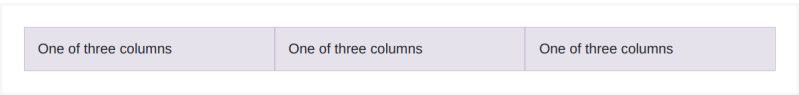


FIGURE 3.3: Bootstrap Grid Layout Example.

3.2.3 Strengths

- **Ease of Use** Bootstrap allows developers to add good-looking, functional and responsive components to their web pages with very little knowledge of HTML, CSS or JavaScript required. Most of the features of Bootstrap are enabled simply by using certain class attributes on HTML elements.
- **Large Community** Being the most popular CSS framework, Bootstrap has a great documentation, tons of community support and a lot of community made free and professional templates.

3.2.4 Weaknesses

- **Customization and Restrictions** If developers intend their website to deviate from the customary design used in Bootstrap, they will have to spend a lot of time overriding styles and re-writing files, slowing down development.

  This further increases one of the inherent issues of using a CSS framework - the fact that it limits creativity. This has been made less of an issue with the introduction of **Sass variables** in version 4.
- **Code Extension** While it is easy to understand, a Bootstrap page layout can be quite verbose. Between creating containers, rows and columns for each section, the resulting document will have a very large amount of elements and code even on visually simple applications.

3.2.5 Example

To get a better visualization and understanding of how Bootstrap works, let's take a look of one of the official code examples provided by Bootstrap, shown on Figure 3.4. The is from the Starter Template[5], the most basic of the ones available.

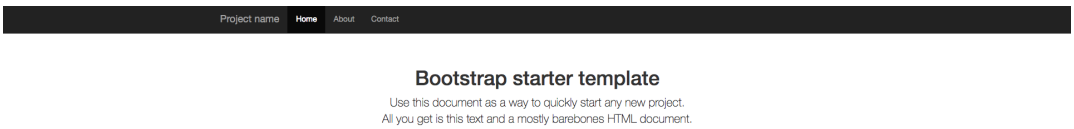


FIGURE 3.4: The Bootstrap 3.3 Starter Template.

[5]https://getbootstrap.com/docs/3.3/examples/starter-template/ - Starter Template for Bootstrap (Accessed September 11, 2018)

Examining the page source code, we can find the HTML markdown shown in Listing 6 used to define the layout and its structure.

```
<nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed"
            data-toggle="collapse"
            data-target="#navbar"
            aria-expanded="false"
            aria-controls="navbar">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">Project name</a>
        </div>
        <div id="navbar" class="collapse navbar-collapse">
            <ul class="nav navbar-nav">
                <li class="active">
                    <a href="#">Home</a>
                </li>
                <li>
                    <a href="#about">About</a>
                </li>
                <li>
                    <a href="#contact">Contact</a>
                </li>
            </ul>
        </div>
    </div>
</nav>
```

LISTING 6: Starter Template Source HTML

This section of the code deals with the navigation bar on top of the screen, as well as its responsive behaviour. Figure 3.5 shows that, on small enough screens, the navigation bar changes its presentation to provide a better user experience.

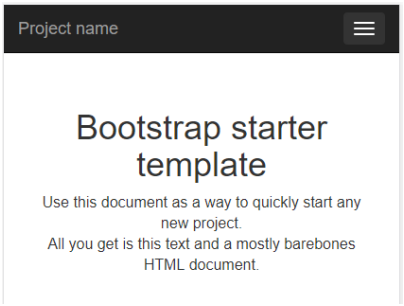


FIGURE 3.5: Bootstrap's Responsive Navbar.

This is achieved using the included `navbar` and `collapse` classes and their subclasses (`navbar-header`, etc.).

```
<div class="container">

    <div class="starter-template">
        <h1>Bootstrap starter template</h1>
        <p class="lead">
        Use this document as a way to quickly start any new project.
        <br>
        All you get is this text and a mostly barebones HTML document.
        </p>
    </div>

</div>
```

LISTING 7: Starter Template Source HTML - Continued

The part of the code shown in Listing 7 refers to the body of the page layout, where the content is displayed. It should be noted that this HTML uses some classes that are not defined in the document or part of Bootstrap, such as the `starter-template` class.

It can be seen on this example how, even for a simple page, the code is quite verbose and how this can pose an issue when scaling up to a large application.

3.3 React

FIGURE 3.6: The React logo.

Initially created by Jordan Walke and released in March 2013, React is now maintained by Facebook, Instagram and community developers and is currently the most popular front-end JavaScript framework. Its logo is shown on Figure 3.6.

React is a declarative, component-based and compatible JavaScript library for building user interfaces[6].

[6]https://reactjs.org/ - React - A JavaScript library for building user interfaces (Accessed September 11, 2018)

3.3.1 Features and Principles

- **Composition** The key feature of React is composition of components. Sections of a page are written as encapsulated components which manage their own state and are composed together to build complex UIs. By design principle, components written by different people should work well together.
- **Declarative API & the Virtual DOM** The virtual DOM is a programming concept where an ideal, or "virtual", representation of a UI is kept in memory and synced with the "real" DOM by a library such as ReactDOM. This enables the declarative API of React: You tell React what state you want the UI to be in, and it makes sure the DOM matches that state.

3.3.2 JSX

React components are typically written in JSX, a JavaScript extension syntax allowing quoting of HTML and using HTML tag syntax to render subcomponents.

The code shown in Listing 8 is an example of a simple React component written in JSX.

```
import React from 'react';

class Component extends React.Component {
  render() {
    return (
      <div>
        <p>My Content</p>
      </div>
    );
  }
}

export default Component;
```

LISTING 8: Sample React Component

JSX code is then compiled into plain JavaScript, meaning that fundamentally, JSX just provides s̈yntactic sugarf̈or the `React.createElement(component, props, ...children)` function. The JSX on Listing 9:

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>
```

LISTING 9: Small Component showing JSX syntax

Compiles into the code shown in Listing 10:

```
React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

LISTING 10: JavaScript compiled from JSX

JSX also uses self-closing tags for elements with no children - Note that the closing slash is syntactically required. Listing 11 shows an example:

```
<input type="text" className="sidebar" />
```

LISTING 11: Self-closing tags in JSX

While similar to regular HTML, JSX expressions present some differences. For example, the `class` HTML attribute does not work in JSX and should be replaced with `className`. Similarly, the value of the `style` attribute should be an object, not a string, as exemplified on Listing 12.

```
<p className="my-paragraph" style={{paddingTop:0}}> Paragraph </p>
```

LISTING 12: JSX specific attributes

Plain JavaScript expressions can be inserted into JSX code by using curly braces, as seen on the example in Listing 13, which shows a select form with dynamically rendered options.

```
<select>
{options.map(function(option){
        return (
            <option value={option.value}>{option.name}</option>
    )
})
}
</select>
```

LISTING 13: Inserting plain JavaScript in JSX

JSX is optional and not required to use React. However, it provides a more readable, easier and faster to write approach.

3.3.3 State and Props

Two essential concepts in understanding how component behavior and inter-component communication can be implemented in React are **State** and **Props**.

State and props (short for "properties") are both JavaScript objects that trigger a re-render when changed. While both hold information that influences the output of render, they are different in one important way: props get passed to the component (similar to function parameters) whereas state is managed within the component (similar to variables declared within a function).

The **state** of a component is an object that is initially declared in the `constructor()` method of a component, and its key-value pairs act as global variables which, when modified via the `setState()` method, trigger the `render()` function of the component to be called, thus re-rendering the component in the DOM.

This approach enables an easy way to do **data binding** in React.

```
constructor(props){
  super(props);
  this.state={
      myVariable="Hello";
  }
  this.onChange = this.onChange.bind(this);
}

onChange(e){
  this.setState({
      myVariable=e.target.value
  })
}

render(){
  return(
      <div>
      Value: {this.state.myVariable}
      <input defaultValue={this.state.myVariable}
      onChange={this.onChange} />
      </div>
  )
}
```

LISTING 14: Data binding using State

In the example in Listing 14, we want the page to display the value of an input field and have it automatically update with user input. So we create a function `onChange(e)` and set it as the handler for the `onChange` event of the input. By doing so, every time the value of the input field changes, the function gets called. When the function is called, it uses the API method `setState` to alter the value of a variable that we declared in the component constructor.

As changing the state of a component triggers a call to the `render()` function, the component will be render again and will display the new value for the variable.

The use of **props** enables components to communicate with each other, by providing a way for them to pass variables and functions to their child components.

```
<ChildComponent value={value} onChange={handleChange} />
```

LISTING 15: Passing Props to a Child Component

The code shown in Listing 15, when placed on the `render()` function of a component, calls for the rendering of a `ChildComponent`, and is passing a variable `value` and a function `handleChange` as props. The instance of ChildComponent can then access them, respectively, with `this.props.value` and `this.props.onChange`. This way, parent components can **communicate** with their child components and vice-versa.

3.3.4 Strengths

- **Performance** React introduced impressive performance benchmarks[7]. With the use of an in-memory virtual DOM, the framework knows just what to change resulting in inexpensive DOM updates.
- **Code Re-usability** React introduced the concept of components as the core of every large application. Components made code easier to reuse, maintain, and test.
- **Large, Bubbling Community** Stemming from a big name like Facebook and introducing innovative concepts, React enjoyed almost instantaneous adoption from the front-end community, leading to a very large number of auxiliary libraries and packages to be developed.

3.3.5 Weaknesses

- **Getting Started** Getting started with React is not as easy as it is with some other frameworks. Not only does it come with a steep learning curve, but its installation generally requires an external utility such as create-react-app and/or a package manager such as yarn or npm.

3.3.6 Seen On

Some popular web applications which heavily use React include Facebook[8], Instagram[9], Twitter[10] and Airbnb[11].

3.4 Vue.js

FIGURE 3.7: The Vue.js logo.

Vue (pronounced like view) is an open-source framework for building user interfaces. Originally created by Evan You, it was initially released in February 2014 and is currently maintained by an international team of developers. As of December of 2017, its latest version is 2.5.9.

[7] http://www.stefankrause.net/js-frameworks-benchmark7/table.html - JS Web Frameworks performance benchmark - 2017
[8] https://www.facebook.com/ - Facebook (Accessed September 11, 2018)
[9] https://www.instagram.com/ - Instagram (Accessed September 11, 2018)
[10] https://www.twitter.com/ - Twitter (Accessed September 11, 2018)
[11] https://www.airbnb.com/ - Airbnb (Accessed September 11, 2018)

Vue is a progressive framework for building user interfaces, designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects[12]. Its logo is shown in Figure 3.7.

3.4.1 Features and Principles

- **Compatibility** Designed to be incrementally adoptable, Vue can easily scale between a library and a framework depending on different use cases. This makes it so that the core library - which focuses on declarative rendering and composition - has high compatibility with other JavaScript libraries and additional features such as routing or state management are offered in officially maintained supporting libraries and packages.
- **Reactivity, Composition and Declarative Rendering** Similarly to React, Vue is based on the composition and nesting of components, and works with a Virtual DOM to achieve reactivity and declarative rendering. This means Vue keeps a virtual representation of the UI in memory and automatically (and only when necessary) updates the DOM.

3.4.2 The Vue Instance

Every application starts by creating a new Vue instance, as shown on Listing 16:

```
var vm = new Vue({
  // options
})
```

LISTING 16: The Vue Instance

When a Vue instance is created, an **options** object is passed to the function. This object can contain several types of options that define the component's behavior. These options can be split into the following categories: Data, DOM, Lifecycle Hooks, Assets, Composition and Misc. From the wide range of options available on the API[13], the following are some of the most used.

- **El** The `el` option provides the Vue instance an existing DOM element to mount on. It can be a CSS selector string or an actual HTML element.
- **Template** The `template` option indicates a string template to be used as the markup for the Vue instance.
- **Data** The `data` object in a Vue instance is what enables its unobtrusive reactivity system. When an object is passed to a Vue instance as its data option, Vue will walk through all of its properties and convert them to getter/setters. These are not visible to the user but under the hood they enable Vue to perform dependency-tracking and change-notification when properties are accessed or modified. When a property in the `data` object is modified - i.e., its setter method is called - the component will be re-rendered, thus achieving reactivity.

[12] https://vuejs.org/v2/guide/ - Introduction - Vue.js (Accessed September 11, 2018)
[13] https://vuejs.org/v2/api/#Options-Data - Vue.js API - Options https://vuejs.org/v2/guide/

- **Props** Properties - or `props` - refer to a list or hash of attributes that are exposed to accept data from the parent component. It has an Array-based simple syntax and an alternative Object-based syntax that allows advanced configurations. This a way for parent components to pass data to child components.
- **Lifecycle Hooks** Multiple lifecycle hooks exist as options. These represent functions that get called at certain points in the life cycle of a component. These include, for example, `beforeCreate` - called synchronously immediately after the instance has been initialized, before data observation and event/watcher setup. Other options[14] include `created`, `mounted`, `updated` or `beforeDestroy`.
- **Methods** The `methods` option includes user-defined methods to be mixed into the Vue instance. These methods are automatically context-bound to the Vue instance and can be accessed directly on it.

3.4.3 Templates and Syntax

Vue uses an HTML-based template syntax that allows for declarative binding of the rendered DOM to the underlying Vue instance's data.

Under the hood, Vue compiles the templates into Virtual DOM render functions. Combined with the reactivity system, Vue is able to figure out the minimal number of components to re-render and apply the minimal amount of DOM manipulations when the app state changes.

Vue also supports JSX and the direct writing of render functions instead of using templates - much like in React.

3.4.4 Interpolation, Directives and Data Binding

The most basic form of data binding is text **interpolation** using the "Mustache" syntax (double curly braces), as shown in Listing 17:

```
<span>Message: {{ msg }}</span>
```

LISTING 17: The mustache syntax

The mustache tag will be replaced with the value of the msg property on the corresponding data object. It will also be updated whenever the data object's `msg` property changes.

Directives, first made popular with AngularJS, are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell the HTML compiler to attach a specified behavior to that DOM element using a JavaScript expression.

In Vue, directives are special attributes with the **v-** prefix.

```
<p v-if="seen">Now you see me</p>
```

LISTING 18: The v-if directive

[14] https://vuejs.org/v2/api/#Options-Lifecycle-Hooks - Vue.js API - Options - Lifecycle Hooks https://vuejs.org/v2/guide/

In the example shown in Listing 18, the `v-if` directive would remove/insert the <p> element based on the value of the expression `seen`.

Directives also enable **data binding** in Vue. The directive `v-bind` can be used to handle a common data binding need - manipulating an element's HTML attributes based on a certain variable or condition.

```
<a v-bind:href="url">My Link</a>
```

LISTING 19: Data Binding using directives

In the example in Listing 19, `href` is the argument, which tells the v-bind directive to bind the element's `href` attribute to the value of the expression url.

The `v-on` directive listens for DOM events (such as when the element is clicked or an input field is changed) and runs a JavaScript expression when triggered.

```
<button v-on:click="counter += 1">Add 1</button>
```

LISTING 20: The v-on directive

The example code on Listing 20 shows a button with the `v-on:click` directive, meaning that when it is clicked, the expression `counter += 1`. The handler expression can be a function call, allowing for complex event handling.

These two directives shown can also be written in a shorthand version, as `:href` and `@click` respectively.

3.4.5 Example Component

The official vuejs-templates[15] repository offers some sample templates of Vue applications. The following code snippets are taken from the **simple** template, a very basic Vue setup in a single HTML file.

```
<head>
  <script src="https://unpkg.com/vue"></script>
</head>
```

LISTING 21: Installing Vue

Installing Vue on a new project is as simple as placing the line of code shown on Listing 21.

[15] https://github.com/vuejs-templates - vuejs-templates (Accessed Semptember 11, 2018)

```
<body>
  <div id="app">
    <h1>\{{ greeting }}</h1>
    <ul>
      <li>
        To learn more about Vue, visit
        <a :href="docsURL" target="_blank">
          \{{ humanizeURL(docsURL) }}
        </a>
      </li>
      <li>
        For live help with simple questions, check out
        <a :href="discordURL" target="_blank">
          the Discord chat
        </a>
      </li>
      <li>
        For more complex questions, post to
        <a :href="forumURL" target="_blank">
          the forum
        </a>
      </li>
    </ul>
  </div>
</body>
```

LISTING 22: Simple Template Source Code

The `body` section of the template, shown on Listing 22, is fairly straight-forward, containing a header and a list with three items.

Throughout this section, we can see data binding being done in the form of the "mustache tag" - displaying the greeting on the header section and by the use of `:href`, the short hand version of the `v-bind` directive, being used to bind the `href` attribute of the links to specific variables.

```
<script>
   var app = new Vue({
    el: '#app',
    data: {
      greeting: 'Welcome to your Vue.js app!',
      docsURL: 'http://vuejs.org/guide/',
      discordURL: 'https://chat.vuejs.org',
      forumURL: 'http://forum.vuejs.org/'
    },
    methods: {
      humanizeURL: function (url) {
        return url
          .replace(/^https?:\/\//, '')
          .replace(/\/$/, '')
      }
    }
  })
</script>
```

LISTING 23: Simple Template Source Code - Continued

A Vue instance called app is created on the script section, shown on Listing 23. Let's take a look at this example's options object.

- The `el` option has a value of `'#app'`. This means that the Vue instance is bound to the DOM element which has the id "app".
- On the `data` object, four variables are defined which are then displayed on the page via data binding. Having been defined on the data object, these variables are reactive - meaning that their display will update if their values are changed.
- Additionally, the `methods` option is used to define an auxiliary method.

3.4.6 Strengths

- **Performance** Vue is a highly performant framework. Having taken the idea of the Virtual DOM and intelligent, inexpensive DOM manipulation from React and having achieved greater performance in other aspects such as loading and compiling, Vue is an impressively high-performance framework.
- **Approachability** Vue is quick and easy to set up, is easy to understand and provides extensive documentation. Combined with the facts that it uses HTML-based templates and has relatively little code weight, this makes Vue a very approachable framework that should be simple to learn with a modest amount of JavaScript knowledge.
- **Familiar Concepts** Vue sought to combine the best parts of frameworks like React (Shadow DOM rendering, component encapsulation e.t.c) and AngularJS (Templates, Directives, Reactivity) into one stable system. This means that rather than introducing new concepts, Vue combines already familiar concepts into a single package, making it approachable to both beginners and developers experienced in other frameworks.

3.4.7 Weaknesses

- **Young Age** As a framework that combines all the best aspects from its strongest competitors, it is difficult to point out a weakness in Vue. However, being of relatively young age, it doesn't feature the largest amount of supporting libraries or answered questions throughout developer forums. This is barely a weakness, as Vue has impeccable documentation and support as well as a dedicated, fast-growing community of developers.

3.4.8 Seen On

Some popular websites build using Vue.js include Font Awesome[16], Laravel[17] and Gitlab[18].

3.5 Angular

FIGURE 3.8: The Angular logo.

Angular - Formerly known as Angular 2 - is a complete rewrite from the team behind Angular.js. Developed by Google, it was initially released in September 2016 and its current latest version is 5.0.1 and its logo is shown on Figure 3.8.

Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop[19].

3.5.1 Features and Principles

- **Cross-Platform** Angular provides the tools for developing progressive web apps[20], as well as native mobile apps and desktop-installed apps across Windows, Mac and Linux, thanks to tools like the Ionic[21] Framework and the ability to access native OS APIs.

[16]http://fontawesome.com/ - Font Awesome (Accessed Semptember 11, 2018)
[17]https://laravel.com/ - Laravel (Accessed Semptember 11, 2018)
[18]https://gitlab.com/ - GitLab (Accessed Semptember 11, 2018)
[19]https://angular.io/docs - Angular - What is Angular (Accessed Semptember 11, 2018)
[20]https://developers.google.com/web/progressive-web-apps/ - Progressive Web Apps (Accessed Semptember 11, 2018)
[21]https://ionicframework.com/ - Ionic Framework (Accessed Semptember 11, 2018)

- **Tooling and Productivity** Angular features extended IDE support (Thanks to Typescript), allowing for fast coding and debugging. It also comes with a command-line utility tool designed to speed up the development process. Additionally, it's a template-oriented framework, allowing for quick creation of UI views using powerful syntax.

3.5.2 Typescript

TypeScript - "JavaScript that scales" [22] - is an open source typed superset of JavaScript that compiles into plain JavaScript. Built for application scale, it adds optional types, classes, and modules to JavaScript. Typescript supports tools for large-scale JavaScript applications for any browser, for any host, on any OS.

For a large JavaScript project, adopting TypeScript might result in more robust software, while still being deployable where a regular JavaScript application would run.

Angular is component-oriented. This means that Typescript's **classes** will be a very helpful feature when working with a large scale application. Tooling and debugging is another important part of why using Typescript on Angular applications makes sense.

TypeScript offers classes, modules, and interfaces. It supports optional static type checking and is a perfect language for developers who are coming from Java and C#[23].

The Typescript code shown in Listing 24, found on the Typescript Playground[24]:

```
class Greeter {
    greeting: string;
    constructor (message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}
```

LISTING 24: Sample TypeScript code

Compiles into the JavaScript shown on Listing 25.

[22] https://www.typescriptlang.org/ - TypeScript - JavaScript that scales (Accessed Semptember 11, 2018)

[23] https://medium.com/this-dot-labs/building-modern-web-applications-in-2017-791d2ef2e341 - Choosing a front-end framework in 2017 - This Dot Labs - Medium (Last accessed September 29, 2018)

[24] http://www.typescriptlang.org/Playground/ - Typescript Playground (Accessed September 11, 2018)

```
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
})();
```

LISTING 25: JavaScript compiled from TypeScript

Notice how the TypeScript defines the type of member variables and class method parameters. This is removed when translating to JavaScript, but used by the IDE and compiler to spot errors, like passing a numeric type to the constructor.

3.5.3 Templating and Data-Binding

As per the MVC (Model-View-Controller)[25] or MVVM (Model-View-Viewmodel)[26] patterns, in Angular, the component plays the part of the controller/viewmodel, and the template represents the view.

The Angular template is written in HTML and almost all of its syntax is valid. A notable exception is the `<script>` tag, which is not allowed for security reasons, preventing script injections.

Apart from this, templating in Angular is fairly straight forward and similar to what we learned earlier in this document (3.4.3).

Data binding in Angular can be achieved in different ways, the most basic one being interpolation, via the use of double curly braces - such as on the example shown in Listing 26.

```
<p>My current hero is {{currentHero.name}}</p>
```

LISTING 26: Data binding using double curly braces

Structural Directives in Angular, such as `*ngIf` or `*ngFor` are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, or manipulating elements, based on the value of an expression.

```
<div *ngIf="hero" class="name">{{hero.name}}</div>
```

LISTING 27: The *ngIf directive

The code snippet on Listing 27 uses the `*ngIf` directive to specify that the element in which it is included is only displayed if the value of the expression `hero` is true.

[25] https://en.wikipedia.org/wiki/Mode-view-controller - Model-view-controller (accessed September 3, 2018)

[26] https://en.wikipedia.org/wiki/Model-view-viewmodel - Model-view-viewmodel (accessed September 3, 2018)

The Angular cheat-sheet[27] contains a list of all the directives that can be used.

Two Way Binding - The `*ngModel` directive is a special one that enables two-way data binding in forms.

```
<input [(ngModel)]="userName">
```

LISTING 28: Two-way binding

The code snippet shown in Listing 28 uses this directive to bind the value of the input field to the variable `username`. **Two-way data binding** means, in this case, that altering the value of the input field automatically updates the value of the variable, and, similarly, if the value of the variable changes, the input field will update to show the new value.

3.5.4 Strengths

- **Tooling** Angular has a big focus on Productivity. Having been written in Microsoft's Typescript, it has extended IDE support for Typescript Angular Apps. Angular also comes with Angular-CLI (Command Line Tools), which allow users to very quickly start building, add components or tests and instantly deploy.
- **Large Community** Stemming from Angular.js, which was initially released in 2009, Angular built up a large, dedicated community. This means more people are actively going to be improving it by reporting issues, forking the project, creating support packages and answering questions. It also means that there is a lot of good documentation and tutorials for newcomers to explore and gain knowledge in the framework.

3.5.5 Weaknesses

- **Code Load and Weight** Angular is a very heavy framework. A simple "Hello World" application could easily be 1MB plus in size. This becomes less of an issue as the scale of the application grows, but Angular generally requires a lot of code and disk space for small achievements. Additionally, the added compilation time and cost of parsing TypeScript into JavaScript can make development builds difficult to work with.

3.5.6 Seen On

Being a Google-developed framework, Angular can be seen being used in a lot of Google's internal / smaller projects such as Google Analytics[28] or AdSense[29]. Angular can also be seen being used on Splice[30], NBA [31] or Royal Caribbean[32].

[27] https://angular.io/guide/cheatsheet - Angular Cheat Sheet (Accessed Semptember 11, 2018)
[28] https://www.google.com/analytics/ - Google Analytics (Accessed September 11, 2018)
[29] https://www.google.com/adsense/start/ - AdSense (Accessed September 11, 2018)
[30] https://splice.com/ - Splice (Accessed September 11, 2018)
[31] http://www.nba.com/ - NBA (Accessed September 11, 2018)
[32] https://www.royalcaribbean.com/ - Royal Caribbean (Accessed September 11, 2018)

3.6 Brief Comparative Overview

Based on the analysis made of the three most popular JavaScript frameworks, the following conclusions can be taken.

Vue is a framework that, from the beginning, set itself to take the best from other frameworks and put them together into one powerful yet approachable package.

Having been successful at doing so, Vue seems like a great choice and the only significant argument against it is that it lacks the **user base and accumulated resources** of React or Angular.

Briefly, we can classify these three frameworks the following way:

- **Use React if** You want to take advantage of open-source auxiliary releases from big names and want to build a fast, powerful application.
- **Use Vue if** Time is a constraint and you're looking to quickly and easily build your project in an approachable yet powerful way.
- **Use Angular if** You're building large scale apps with a lot of planned maintenance and debugging and/or if you have plans to develop mobile apps.

These are initial conclusions that will serve as a leading point to a deeper comparative analysis that will be made in the next stages of the work period.

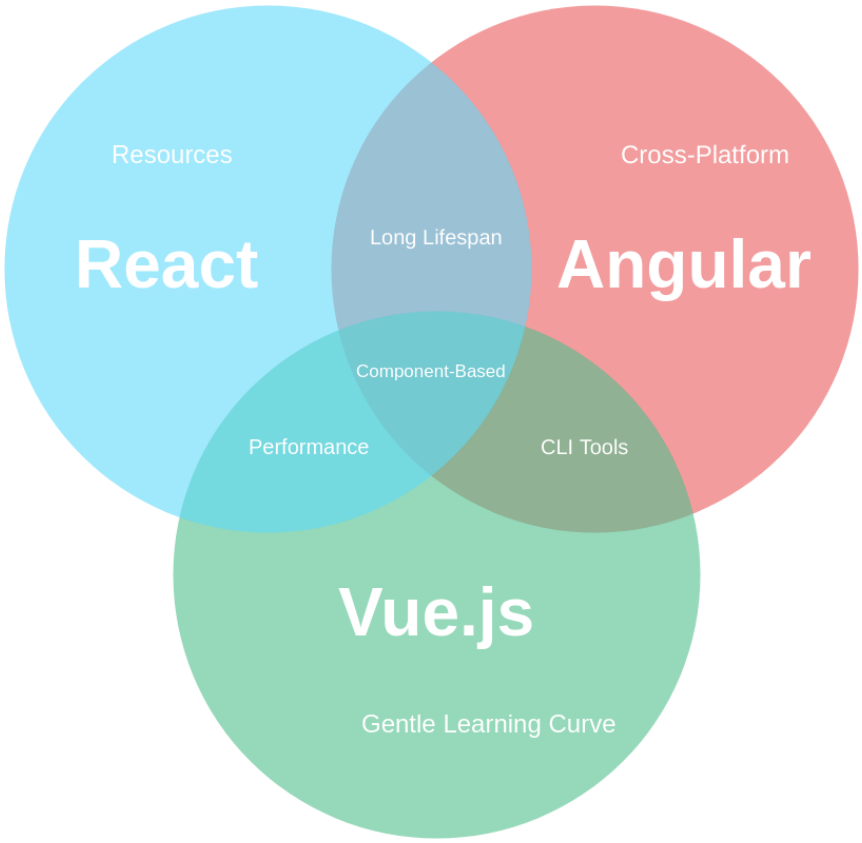


FIGURE 3.9: Venn diagram displaying main features of each of the selected frameworks.

To complement these conclusions, Figure 3.9 shows a different presentation of what could be learned about the most popular front-end frameworks at this stage of the dissertation.

3.7 Approach

With these learnings, the goal is to develop an approach, with the intent of helping developers make the decision on which JavaScript framework to use on their project, based on their needs and requirements.

An initial idea from which this approach can be fully developed is presented in Figure 3.10 in the form of a diagram.

In this diagram, only the three previously selected frameworks - React, Vue.js and Angular - appear as valid choices, as they are the ones that have been previously analyzed and of which a sufficient amount of knowledge has been gathered at this point in the work period.

On the diagram, each round element represents a need, requirement or preference and the arrow coming out of it will point to the framework or frameworks that have been selected as more appropriate to deal with it.

For example, the element positioned at the top right of the diagram and labeled "Performance" will point to Vue.js and React, as the analysis made led to the conclusion that similar applications built using these two frameworks generally perform better than ones built in Angular.

Additionally, the elements are color-coded, as the ones with the orange background refer to project requirements or developer-specific conditions such as being new to Front-End Web Development, having an Object-Oriented programming background, etc. The elements with the gray background color represent more technical aspects such as performance requirements or the complexity of the application's component hierarchy.

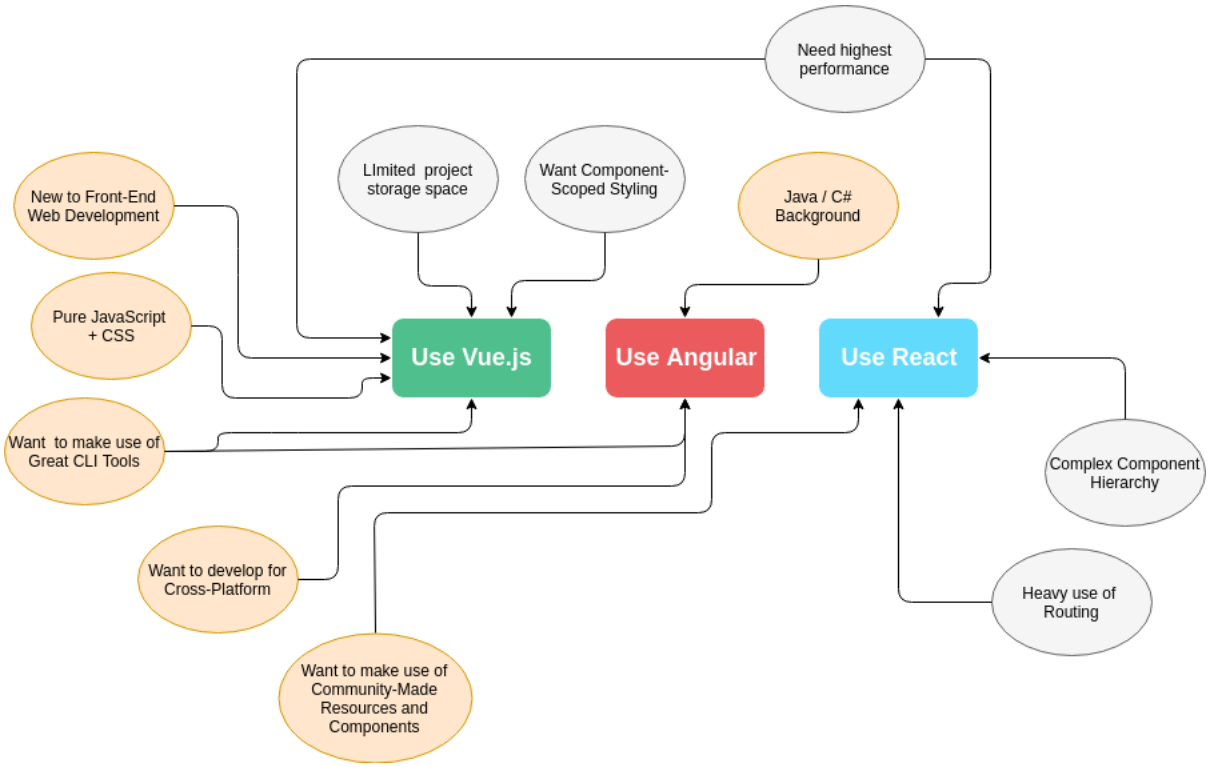


FIGURE 3.10: The developed approach diagram.

This represents an initial idea of what the final approach may look like, after it goes through further development and validation.

This initial approach will be validated in Chapters 4 and 5, where a case study development and the author's work at Jumpseller will offer a practical test to what was learned up to this point in this document.

3.8 Summary

In this chapter, an overview of the most popular Front-End frameworks was made, by firstly collecting usage data considering the number of stars and forks on GitHub to select the technologies to analyze.

Once selected, a look was taken at each of the frameworks by evaluating its strengths and weaknesses, and taking conclusions about what makes them the most popular. On each of these sections the main features of each framework were presented, small examples were provided and a look was taken at how certain common operations are performed.

This comparative analysis allowed some initial conclusions to be taken about how these technologies work and what situations they are most fit to handle. A Venn Diagram was created to show what was learned about each of the selected frameworks and how these learnings are shared between the three.

Finally, an initial approach was developed and presented in the form of a Diagram in the final section of the present chapter. This diagram presents several types of requirements, needs or preferences along with an indication of which of the analyzed JavaScript frameworks better handles the issue. With this, a base is set for developing an approach which developers can utilize to select which JavaScript framework to use on a given project, based on its needs and requirements of both the project and the developer.

Chapter 4

Case Study

There is only so much that can be learned and studied about a development framework by reading through its documentation and features list.

As such, this chapter intends to put the selected frameworks to test, by defining a case study upon which these technologies will be put to use, allowing for a better understanding of the strengths and weaknesses each one may carry.

To achieve this, the plan is to find a problematic section within the Jumpseller platform, analyze what could be done to improve it and, with the help and feedback from the team, envision a new, improved version by creating mock-up images of what it should look like. After that, the new application will be developed using the three different JavaScript frameworks previously chosen - React, Vue and Angular - and comparisons between them and conclusions will be drawn and taken from the experience.

4.1 Background

To define a case study upon which to work on, a few points were taken into consideration.

Given the context of this work and the opportunity for its validation that comes with it, it made sense to work on an existing section of the Jumpseller application, so that one of the new implementations could later be validated by the team and integrated with the enterprise back-end.

Beyond that, and being that this represents an opportunity to not only solidify the knowledge and experience about the selected frameworks but also about UI/UX Design, Design Patterns and Anti-patterns, it makes sense to find a specific application, section or page that presents some clear issues and upon which some work can be done to improve it - A section that could be agreed between the author and the Jumpseller team had a less-than-ideal user interface, one that is too slow, confusing or poorly implemented.

In these conditions, work is being done in a real environment in order to achieve a real solution that will be put to use, solidifying the importance of this study.

4.2 The *Languages* Section

After browsing the different sections of the admin panel on the Jumpseller application and consulting with the team, a consensus was reached that the Languages section was a good example of a page that needed some work.

On this section, store owners can manage their store's supported languages, by choosing which languages should be used - which of those is the main language - as well as customize the translations used on other languages. In its current version, it looks like Figure 4.1.

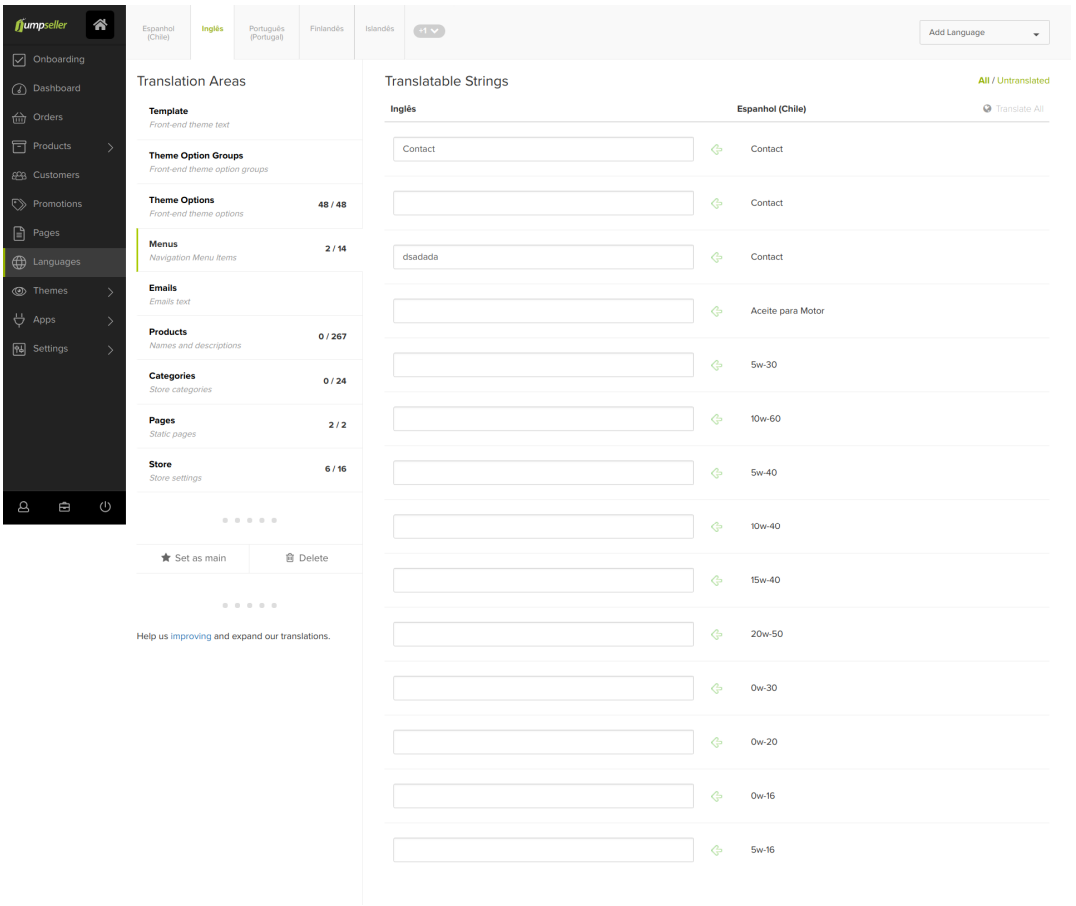


FIGURE 4.1: The Desktop version of the languages section.

From a brief analysis of this page it can be understood that this interface is not ideal in several aspects. Let's start by listing the features of the page and the actions which can be performed on it, shown on Figure 4.2

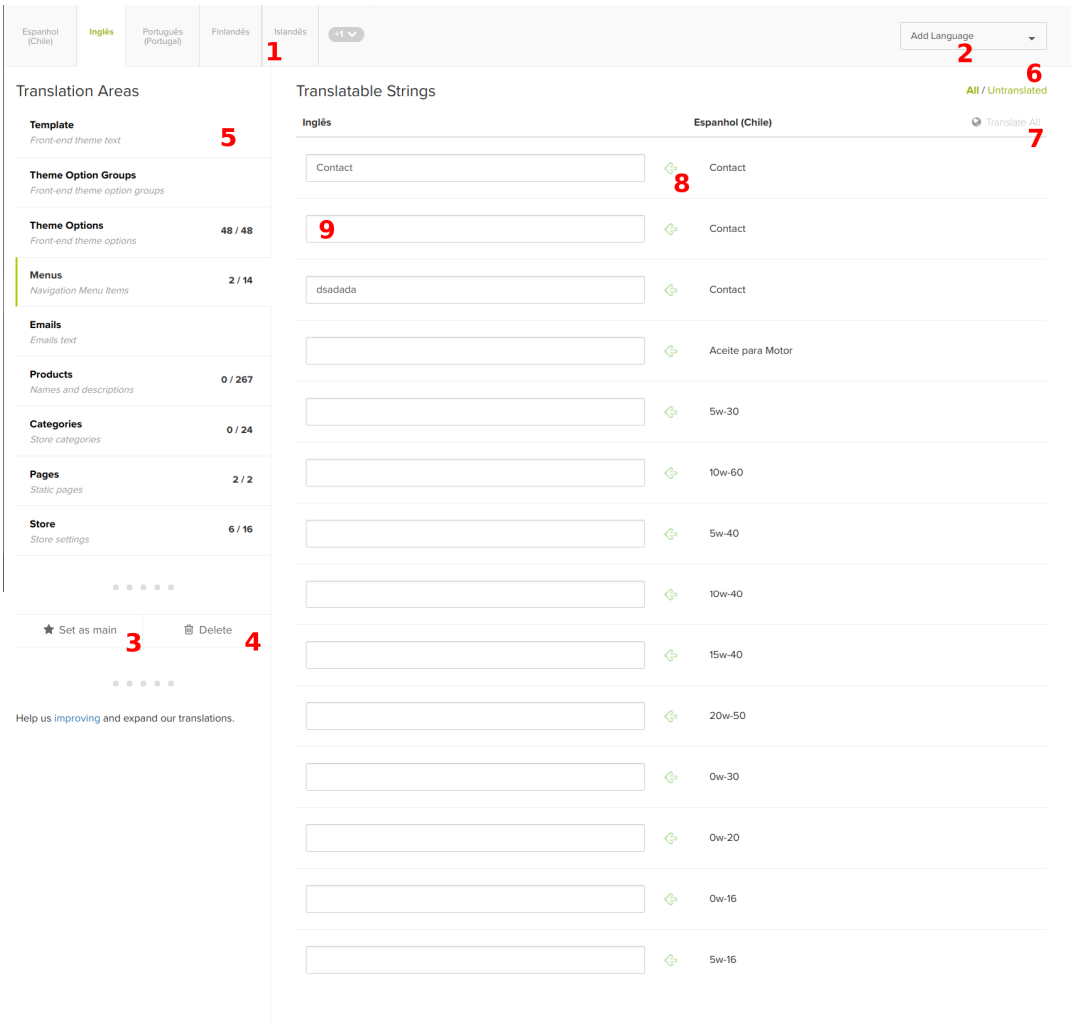


FIGURE 4.2: The features of the languages section.

1. Cycle through the different supported languages.
2. Add a new language.
3. Set a language as the main language of the store.
4. Delete a language
5. Cycle through the different sections in which the translated strings are divided.
6. Toggle between displaying all strings or only the untranslated ones.
7. Translate all strings automatically (Using the Google Translate service API).
8. Automatically translate a string (Using the Google Translate service API).
9. Manually translate a string.

Upon a first analysis by the author, a few problems were found:

1. The interface feels cluttered. Having 9 is performable actions at a given point is too much and represents the **Bloated Interface Anti-pattern** - A user interface that tries to incorporate as many operations as can possibly fit into it with the end result of confusing more than helping the user to perform his or her task. (Toxboe, 2009)

2. The combination of a tab-based navigation on the language selection and a list-based navigation for the strings makes the page feel incoherent.

3. The placement of the Delete and Set as Main actions is odd and unintuitive. Beyond that, it requires scrolling on certain screen sizes, making it difficult to find. Being that these are important actions, the user should be able to quickly identify them.

4. The limited-space, horizontal listing of the categories makes it difficult to, at a glance, identify which languages are currently supported. It is also currently impossible to compare their completion status from the list.

4.2.1 Mobile

The mobile version of this page shares most of these issues and has a few of its own, as seen on Figures 4.3 and 4.4.

1. The tab-based navigation works even worse in mobile, as the space is limited and clashes with the "Add language" button.

2. Finding the section where strings can be translated - which is, ultimately, the most important feature on this page - requires scrolling and is not immediately apparent when a section is chosen.

3. The design of the translation area is extremely cluttered - elements clash and are put out of their position and there are uneven margins and paddings. The input field is also very small, which represents a problem on longer strings.

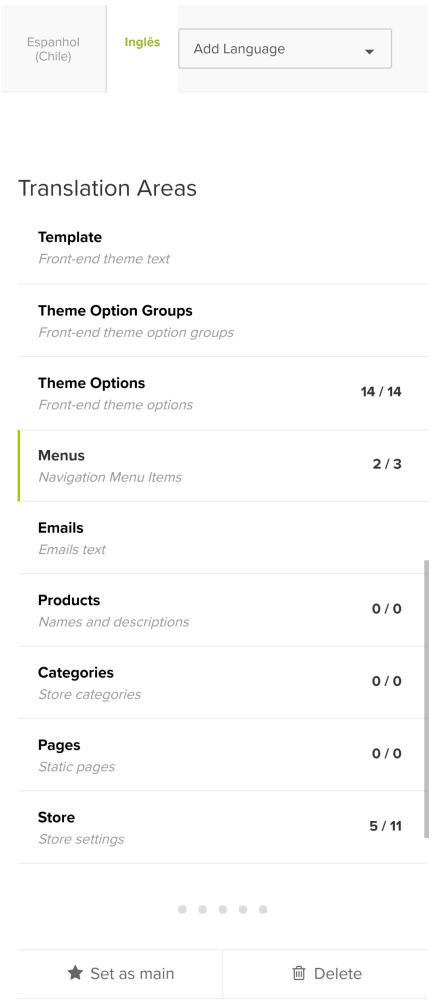


FIGURE 4.3: Languages on Mobile.

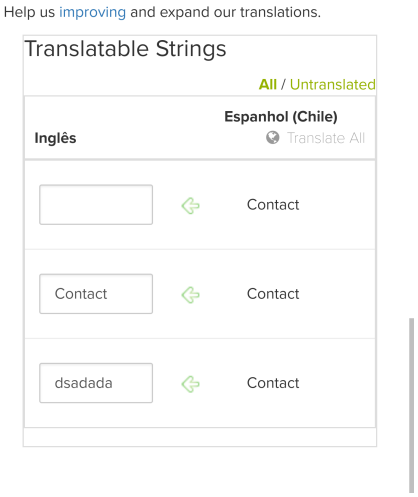


FIGURE 4.4: Languages on Mobile.

4.2.2 Approach

Considering the issues found, research was done on how to resolve these problems by looking at examples and studying common design patterns (Van Welie, Van Der Veer, and Eliëns, 2001).

A few things were decided:

- Separating the language listing from the strings translation area makes a big difference, as it allow a better distribution of features across interfaces and results in a cleaner look and a better user experience.
- The language list will be a table-like structure similar to the list of strings in the translation area, thus getting rid of the tab-based navigation and resulting in a more coherent interface.
- A status indicator will be added to the language listing, to indicate whether or not each language is currently in use on the store.
- The mobile version of the section will be completely re-done, using responsive design technologies and implementing design patterns such as Cards and Vertical Dropdown Menus.

With these ideas in mind, design work was done in order to better visualize these suggestions.

4.3 Mock-ups

The first step towards implementing a new, improved version of this application is to plan it by setting and end goal consisting of the features and experience that the final product will offer. With that in mind, an interface design tool - Figma[1] - was used to create mock-up images of what the final implementation will look like, considering the points and suggestions made on the previous section. These are shown on Figures 4.5 and 4.6.

4.3.1 Desktop

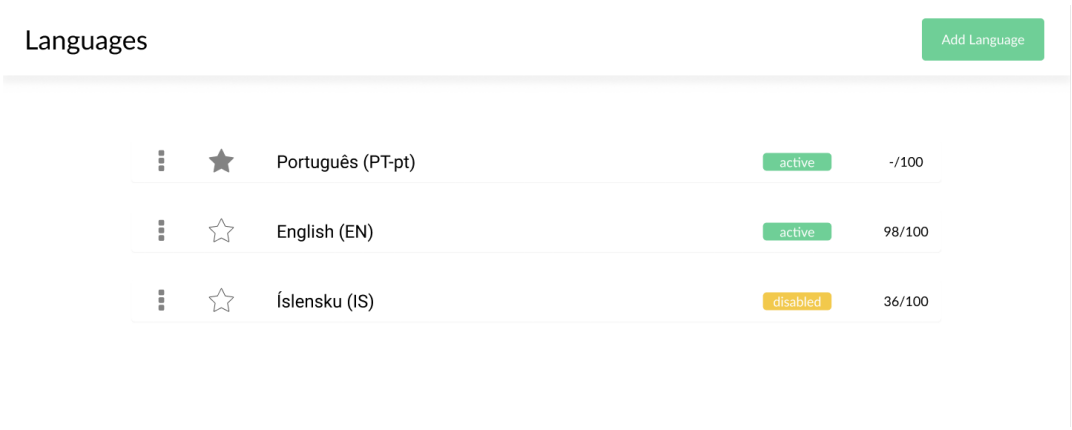


FIGURE 4.5: Language List Mock-up on Desktop

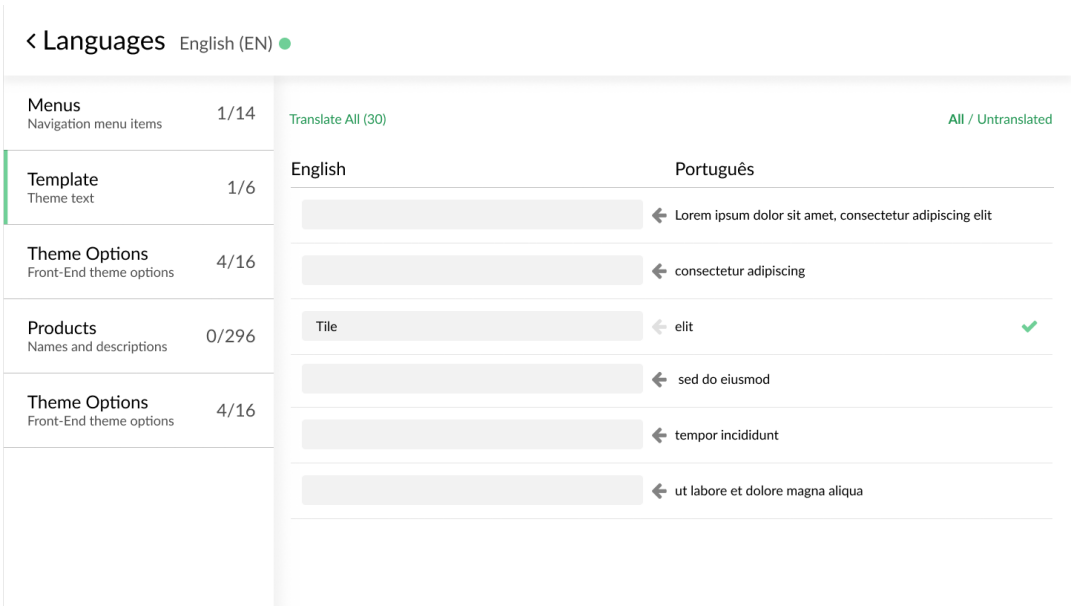


FIGURE 4.6: Language Mock-up on Desktop

[1]https://www.figma.com/ - Figma: The collaborative interface design tool (Accessed September 11,2018)

Looking again at the list of features, we can see that it has changed a bit and, more importantly, it's more distributed across the interfaces, allowing them to be more visible and easy to find and use on a cleaner design.

4.3.2 Mobile

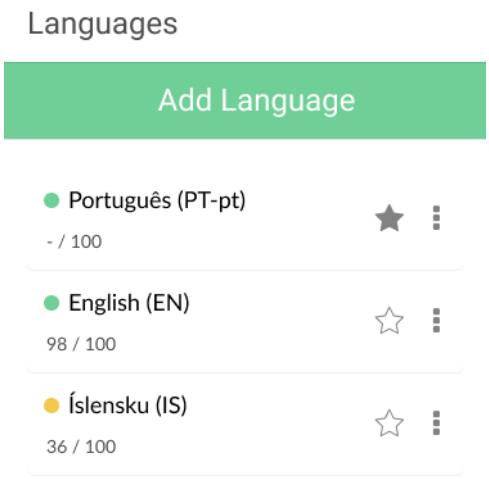


FIGURE 4.7: Mock-up on Mobile.

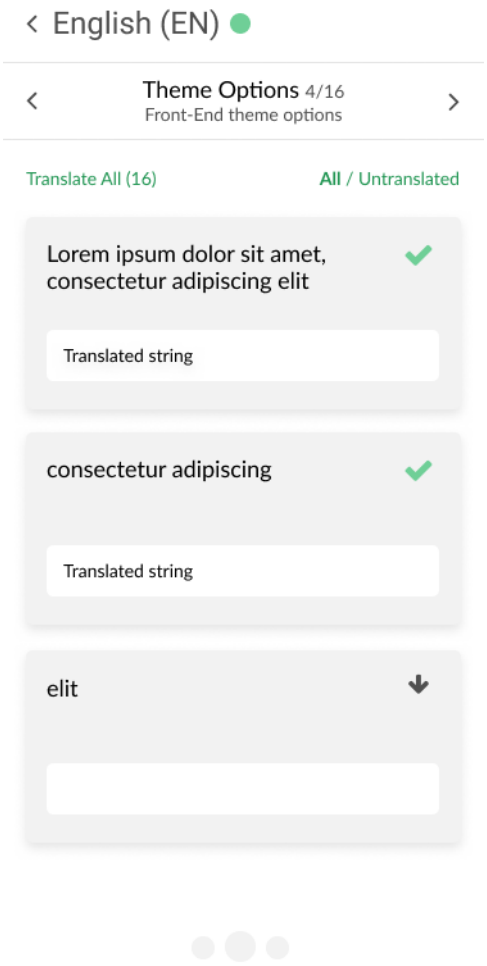


FIGURE 4.8: Mock-up on Mobile.

The proposed design for the mobile version of the app, shown on Figures 4.7 and 4.8, brings a lot of changes. Besides the better distribution of tasks among the two pages, the translation area looks much nicer and distributes the translatable strings using cards. These changes make it much more usable than the previous version.

4.3.3 Components

Additionally, and since this interface was designed keeping in mind its implementation in JavaScript and in component-based frameworks, Adobe Photoshop[2] was used to create the diagrams shown on Figures 4.9 and 4.10, representative of each component that will make up the final application. This will help the development process as it sets a plan of the structure that the application code will take.

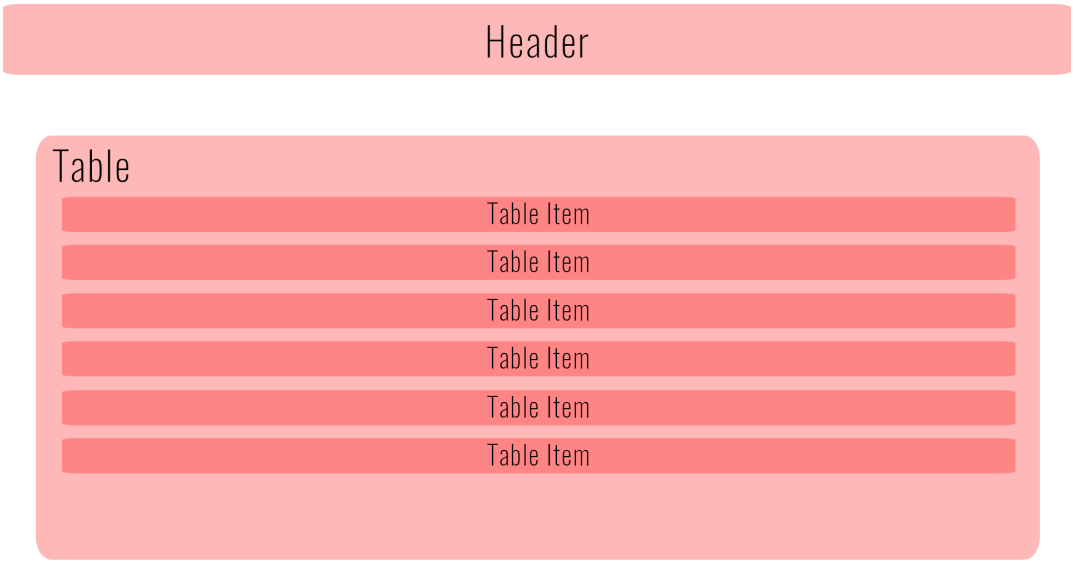


FIGURE 4.9: Language List Desktop Components

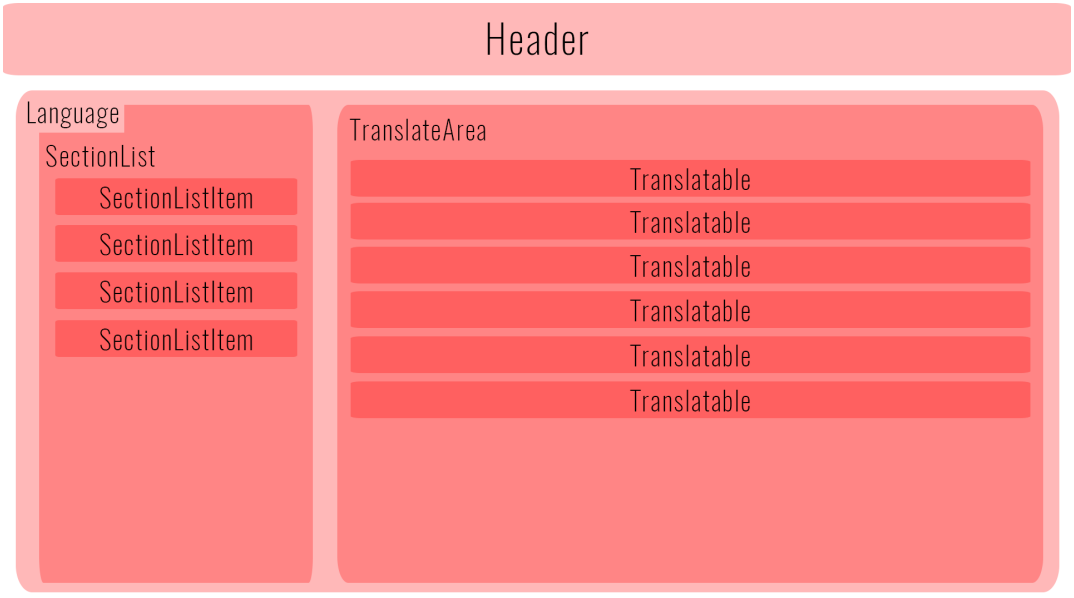


FIGURE 4.10: Language Desktop Components

[2]https://www.adobe.com/photoshop - Photoshop (Accessed September 11,2018)

4.4 Feedback

After developing the first mock-ups, a small meeting was arranged with the head designer and the author's supervisor at Jumpseller, with the objective of obtaining feedback from someone with experience with real world situations and the Jumpseller application, as well as UI design.

On this meeting, a few changes were agreed upon.

4.4.1 General

- The addition of an indicator of the **language's status (Active / Disabled) is unnecessary**, as there are no plans or necessity of implementing the feature on the application back-end.
- With the removal of the status feature, it was agreed that **the dropdown menu in the language list was unnecessary**, as the only feature it would unveil would be the language deletion. This should be replaced with a **Delete icon or button**.
- Adding a new language to the store is something that doesn't happen too often, so the team felt that **the "Add Language" button brought too much attention to itself**. It was suggested to find a new, more subtle placement for it.
- When facing the list of translatable strings, which can be quite long, finding a specific string that needs work may be complicated. As such, it makes sense to **implement a simple search feature**.
- The "Translate All" feature is one that is seen as a necessary evil to the Jumpseller team. While it has its use, it is costly in both processing and financial power (as the translation service used for this feature requires that the company pays for each translated string). As such, the team **requested that the "Translate All" button was presented in a more subtle, less obvious way**.
- The team pointed out that, along with simple one-or-two-word strings, longer and more complicated strings often find their way into the translation system. As such, it was requested that **new interface fully supported and adapted to long strings**.

4.4.2 Desktop

- As part of the principle of supporting long strings, it was requested that **the translation area be full-width**, meaning that **the sidebar should be removed and a new place for the section list should be found**.

4.4.3 Mobile

- While the horizontal scrolling to navigate the different translation sections in mobile was positively received, it was pointed that **there should still be a way to view the full list of sections**. It was suggested by the author and agreed upon that clicking the section selection area should open a modal with the full list of sections.

4.5 New Mock-ups

After gathering feedback, a new version of the mock-ups was created, with the intent of fixing the issues that were pointed out and further building on the presented ideas, represented on Figures 4.11 and 4.12.

4.5.1 Desktop

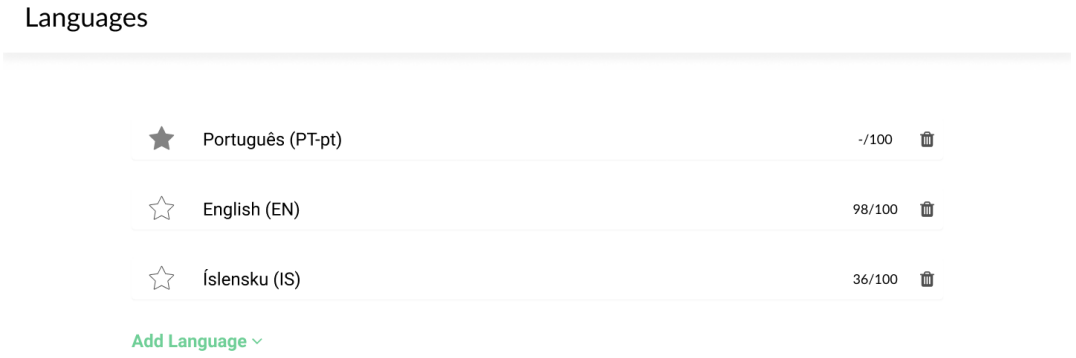


FIGURE 4.11: New Language List Mock-up on Desktop

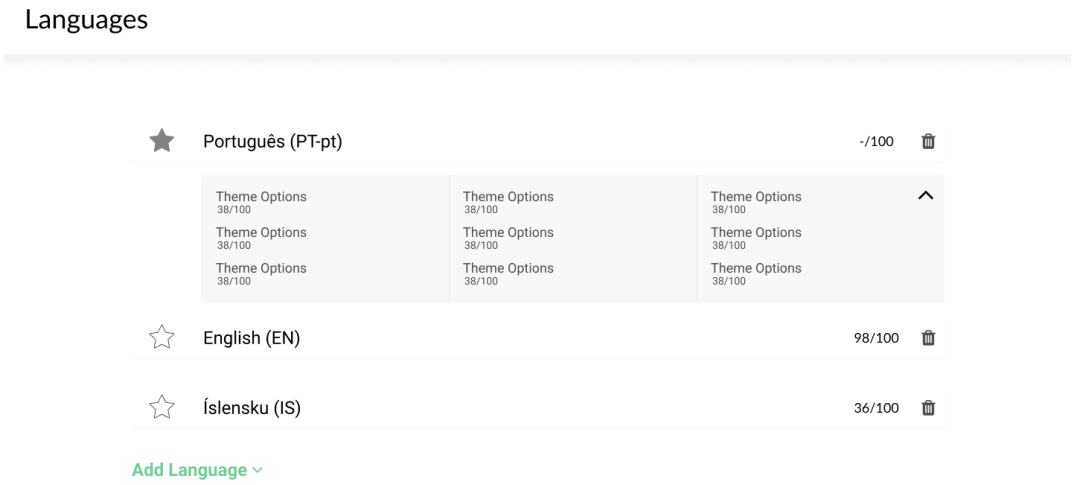


FIGURE 4.12: New Section List Mock-up on Desktop

Along with the suggested visual changes, a significant change to the mobile version was the introduction of an **accordion menu** where users can choose, from the language list, the section on which they want to translate strings. This can be seen on Figure 4.13.

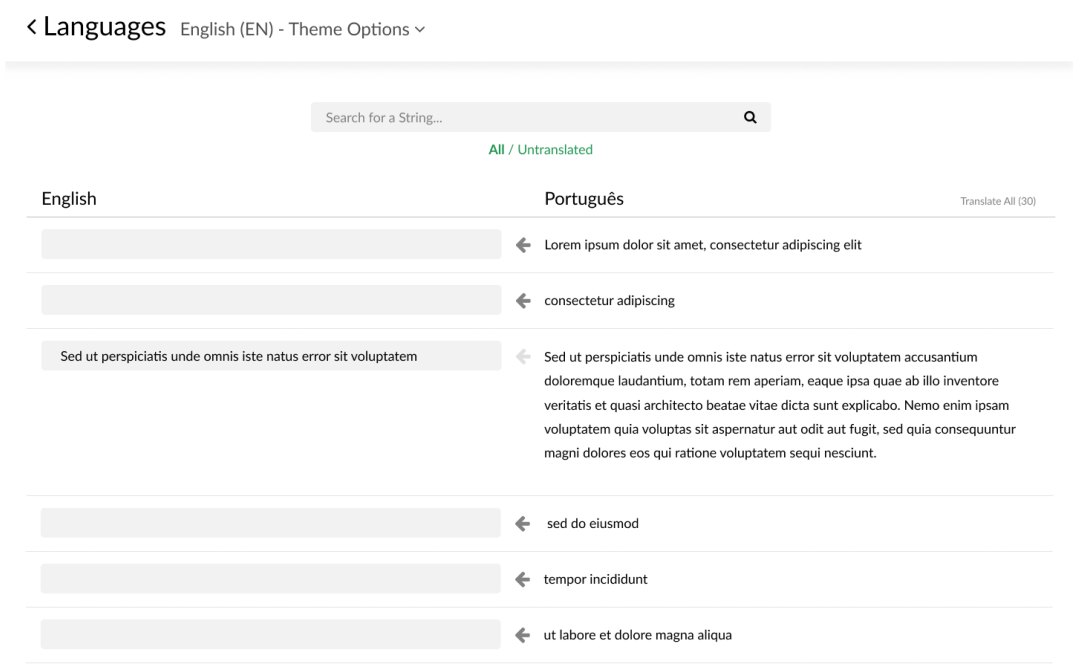


FIGURE 4.13: New Language Mock-up on Desktop

A simple search bar was added to the translation area, where users can input a search term used to filter the displayed translatable strings.

4.5.2 Mobile

Most of the changes done to the Desktop version transfered to their Mobile counterpart, including the removal of the dropdown menu, the inclusion of the Delete icon and the addition of a search feature, as shown in Figures 4.14 and 4.15.

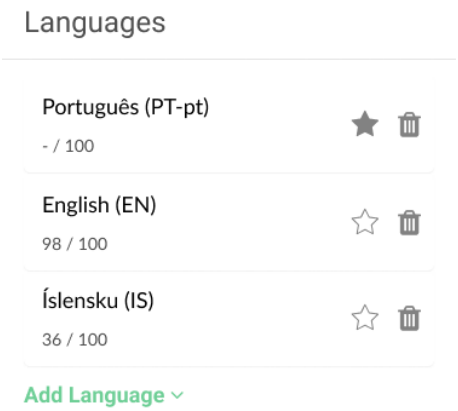


FIGURE 4.14: New Mock-up on Mobile.

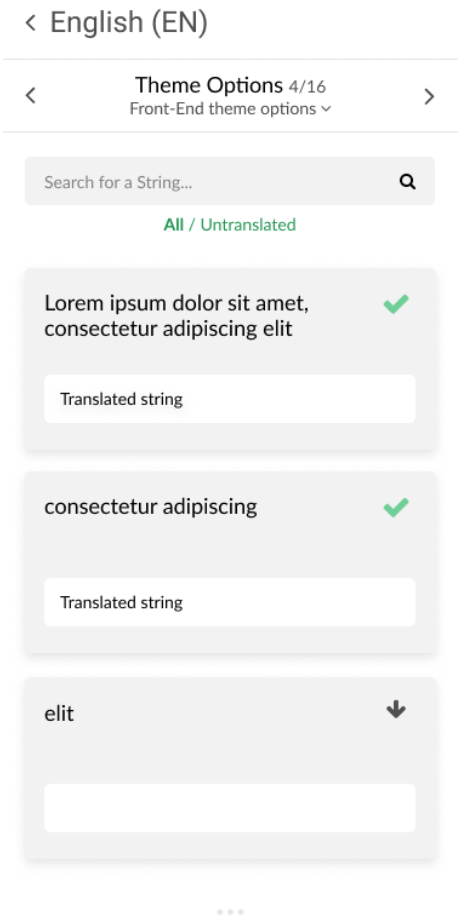


FIGURE 4.15: New Mock-up on Mobile.

4.6 Development

After all changes agreed upon were done to the mock-up images, the development phase could be started.

4.6.1 Data

Being strictly a front-end study, it was decided it was not necessary to develop a back-end application to serve the data. Instead, mock data was loaded into each app through a JSON[3] file containing the information to be presented. The data object is shown in Figure 4.16.

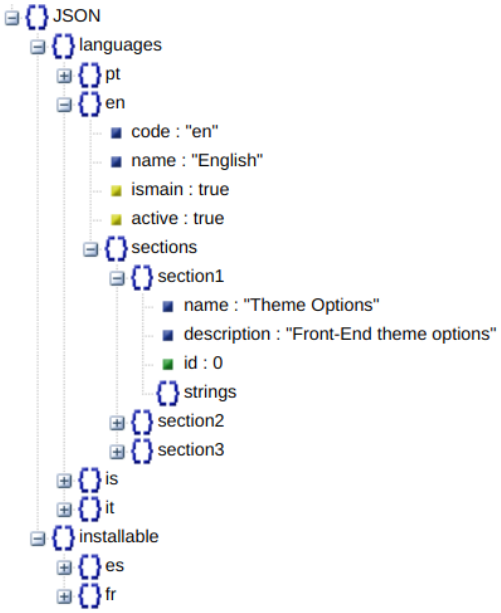


FIGURE 4.16: JSON file containing the Data to be displayed

4.6.2 System and Pre-Requisites

In order to use command line tools as well as managing packages installed within each application, Node.js and a package manager such as npm[4] (Node Package Manager) or Yarn[5] are required.

For this stage of development, the following version of each of the tools was installed:

- Node 10.0
- npm 5.6.0
- Yarn 1.9.2

The applications were developed in a Linux Mint[6] 18.2 system and the IDE of choice was Visual Studio Code[7].

[3]https://www.json.org/ - JSON : JavaScript Object Notation (Accessed September 11, 2018)
[4]https://www.npmjs.com/ - npm (Accessed September 11, 2018)
[5]https://yarnpkg.com/en/ - Yarn (Accessed September 11, 2018)
[6]https://www.linuxmint.com/ - Linux Mint (Accessed September 11, 2018)
[7]https://code.visualstudio.com/ - Visual Studio Code - Code Editing. Redefined. (Accessed September 11, 2018)

4.6.3 CSS

Bootstrap

Given that the main goal at this stage was to recreate the same application on different JavaScript frameworks, it was decided that Bootstrap would be used to create the layout of the interfaces.

SCSS

Since the intention was to maintain visual parity across different components and, further, different applications, it made sense to make use of Sass and, specifically, **variables** and **nesting**.

Variables were used to set the colors to be used for the whole project. The SCSS variables that were used to define colors such as green or light-gray are in the context of the project (i.e. what is the corresponding hex) are listed on Listing 29.

```
$green: #6FCF97;
$dark-green: #219653;
$red: #d44946;
$yellow: #F2C94C;
$gray: #e6e6e6;
$light-gray: #f4f4f4;
$gray-text: #4f4f4f;
```

LISTING 29: SCSS Variables used

Nesting of CSS properties is one of the most useful features of Sass, and was used to improve the readability and code weight of the style, specially considering the naming convention adopted. The excerpt shown on Listing 30 demonstrates both, as well as the use of the previously defined variables.

```
.languages {
  &__aux {
    &--links {
      color: $green;
    }
    &--chevron {
      margin-right: 1em;
    }
    &--badge {
      width: 5em;
      font-size: 0.7em;
      padding-top: 0.25em;
      padding-bottom: 0.25em;
    }
    &--badge-active {
      background-color: $green;
    }
    &--badge-disabled {
      background-color: $yellow;
    }
  }
}
```

LISTING 30: CSS Property Nesting

4.6.4 Developed Applications

Given that the developed applications were made to mirror the mock-up images created, they all share the same look. Figures 4.17 and 4.18 show screenshots of one of the applications.

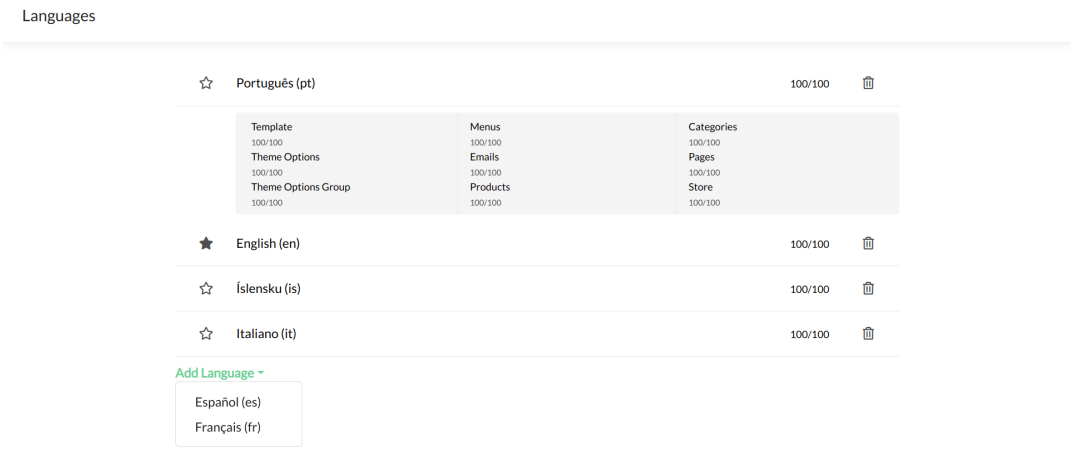


FIGURE 4.17: The developed applications, showing the list of sections within the language list

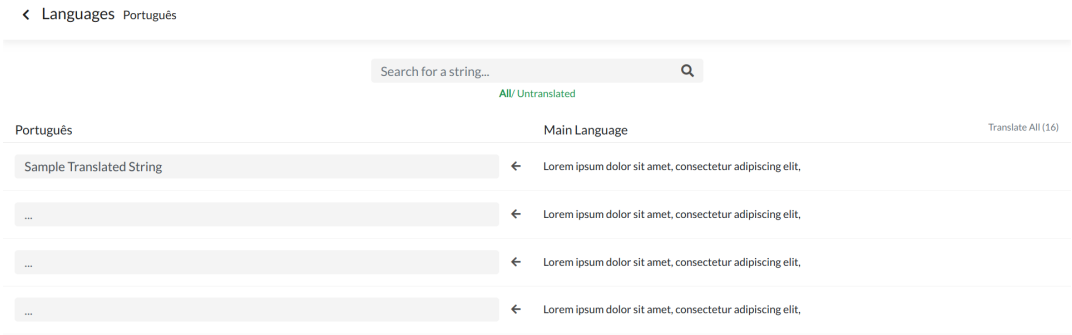


FIGURE 4.18: The developed applications, showing the strings translation area

All versions of the developed application have the following functionalities:

- List the currently available languages and their completion status
- Set any language as main language
- Remove a language from the set of available languages
- Add a new language to the set of available languages
- List the sections and their completion status for each language
- Choose a section within a language to edit the translatable strings

Upon choosing a language and section of strings to translate, the following features become available:

- Search for a specific string to translate
- Toggle a filter to show all translatable strings or only the currently untranslated ones
- Translate all the untranslated strings
- Manually translate a string
- Automatically translate a string

All these functionalities are present and working on mobile devices and smaller screens, as the application is responsive and device-compatible.

4.7 Comparison

During the development of the interface on each of the JavaScript frameworks, some differences (or similarities) among them came up and, as such, comparisons were made between the three of them.
On this section, a list of points will be presented upon which a brief comparison of how each of the framework handles specific stages or situations of development will be made.

4.7.1 Bootstrapping and CLI tools

While both React and Vue can be added to any page by "traditional" methods, such as adding a *script* tag to the index HTML file, other tools exist to get started with a project. Meanwhile, the Angular documentation officially recommends the use of Angular-CLI[8].

React

The developed React app was bootstrapped using **create-react-app**[9].

Create-react-app is a Facebook-created tool that allows for users to quickly create React apps with no build configuration.

The utility is easily installed using a package manager and doesn't require the installation or configuration of any additional tools such as Webpack or Babel, as they come pre-configured and hidden so one can focus on the code. It also sets up scripts for running the app locally or building it for production.

Vue

The Vue project was started using the **vue-cli**[10]. This is an official CLI tool provided by Vue that allows for the quick generation of projects using a variety of provided build setups.

This is achieved by running the **vue create** script, which prompts a series of configurations and generates a new, ready-to-work-on project, as shown on Figure 4.19.

[8]https://cli.angular.io/ - Angular CLI (Accessed September 11, 2018)
[9]https://github.com/facebook/create-react-app - facebook/create-react-app (Accessed September 11, 2018)
[10]https://cli.vuejs.org/guide/ - Vue CLI (Accessed September 11, 2018)

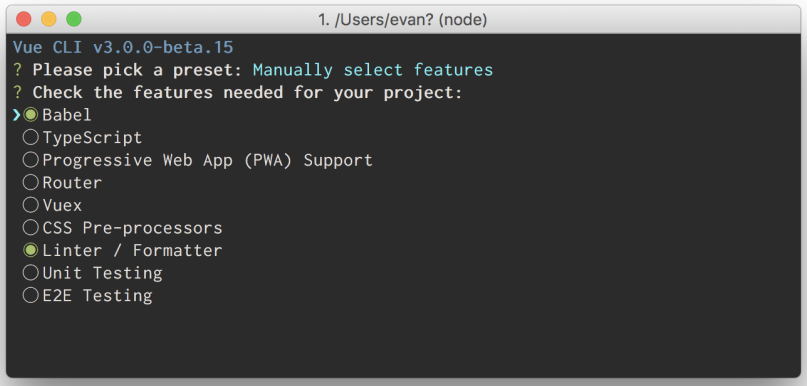


FIGURE 4.19: Configuring a new Vue app

Angular

Angular officially recommends the use of the Angular CLI.

Creating a new project with the Angular CLI is similar to how it is in React or Vue and is done by running the **ng new** script, which creates a working application out of the box, following the officially-recommended best practices.

The Angular CLI however steps up its game from the other CLI tools with the **ng generate** command. This command, along with two supplied parameters (type and name) allows the automatic generation and configuration of new **components**, **routes**, services or pipes along with simple test shells for all of them.

Verdict

React	**Vue**	**Angular**
No Considerable Advantage	Highly Configurable Setup	Useful, Time-Saving Development Scripts

TABLE 4.1: Bootstrapping and CLI

4.7.2 Styling

While the ability to easily include CSS files to style components is common to the three frameworks, Vue and Angular share the fact that styles can be component-scoped, while React generally imports styles globally.

React

Adding a stylesheet to React is quite conventional and consists of using an *import* statement specifying the path of the CSS file to import.

Vue

While *import* statements can be used, Vue components are different in that they have a **<style>** element. Within this element, regular CSS properties can be added to the project.

Additionally, this element can have the **scoped** property, which makes it so that properties specified in the element are applied only to the template of the component it's inserted in.

Angular

Angular components specify on their constructor which CSS file or files will be used for styling with the **styleUrls** parameter. This allows for component-scoped stylesheets at the expense of creating extra files.

Verdict

React	**Vue**	**Angular**
No Considerable Advantage	Scoped styling further enables component sharing	No Considerable Advantage

TABLE 4.2: Styling

4.7.3 Routing

While they each have their specific syntaxes and quirks, routing mostly works in a similar way in Angular and Vue. Routes are defined beforehand and router links are used to navigate through the different sections. In React, different options exist, and the one used in the case study features dynamic routing - Where routes are defined during the rendering.

React

Although several options exist for implementing routing in React[11], the most commonly used one is **react-router**, a declarative router built with the philosophy of having dynamic routing. This refers to routing that takes place as the app is rendering, not in a configuration or convention outside of a running app[12].

This means that the implementation of routes in React is quite simple for the developer. As seen on Figure 4.20, and assuming the router has been imported to the application, the application must be inserted within the `<Router></Router>` tags, and routes are defined with the `<Route/>` component, on which a path is passed along with which component or object should be rendered when the path is matched.

[11] https://reactjs.org/community/routing.html - Routing - React (Accessed September 4, 2018)

[12] https://reacttraining.com/react-router/web/guides/philosophy - React Router (Accessed September 4, 2018)

```
return (
    <Router>
        <div>
        <Header data={this.state.data} onAddLang={(lang) => {this.addLanguage(lang)}}/>
        <Route exact path="/" render={tableContents}/>
        <Route path="/language/:code" component={Language}/>
        </div>
    </Router>
);
```

FIGURE 4.20: Declaring routes in React

With the routes set, links can be placed to navigate through the application with the `<Link/>` component, which takes a parameter **to** that indicates the path it should redirect to.

Vue

Vue Router is the official router for Vue.js[13]. Implementing basic routing in a Vue project consists of three steps. Importing and adding Vue Router to the project, defining the routes and using them to initialize routing, and defining a `<router-view />`, i.e., where the matched route content will be rendered. The first two are done in the *main.js* file, whereas the router-view is placed on the template of the desired component. This is shown on Listing 31.

```
const routes = [
  { path: '/', component: Table },
  { path: '/language/:code', component: Language}
]
const router = new VueRouter({
  routes
})

Vue.use(VueRouter)
```

LISTING 31: Routing in Vue

Links can then be created using the `<router-link to="/"/>` syntax, where the **to** parameter refers to the desired target path.

Angular

The Angular Router enables navigation from one view to the next as users perform application tasks[14]. Routing in Angular is done very similarly to Vue, as it has the same three steps. On the *app.module.ts* file, the router is imported and the routes object is defined, and a `<router-outlet />` is created on a component's template.

Verdict

[13] https://router.vuejs.org/ - Vue Router (Accessed September 4, 2018)
[14] https://angular.io/guide/router - Angular - Routing & Navigation (Accessed September 4, 2018)

React	**Vue**	**Angular**
Less pre-configuration required and less code overall	No Considerable Advantage	No Considerable Advantage

TABLE 4.3: Routing

4.7.4 Component Communication

Being three component-based frameworks, the way that parent components can pass data down to their children and receive data back from them is similar and consists of defining data objects on the child component and specifying which data should be passed on the template of the parent component.

React

In React, when rendering a child component within a parent, data can be passed to it as **props**. For example, if the intention is to pass a language object to a component that renders list elements, one would declare the child component as `<TableItem language=english/>`. From there, the child component can access that data using `this.props.language`.

Methods can be passed as props, which enables the other way of communicating. If a child component wants to alert the parent of an event, it can call a method which was passed to it as props that will be executed in the parent component.

Vue

Vue, similarly, uses **props** for component communication, the difference being that when creating a new component, the props that component will accept should be declared. This is done by adding values to the props object of the Vue instance (As seen on section 3.4.2).

Child-to-parent communication is, however, slightly different. Instead of calling a method from the props, the child component can use the `$emit` function to emit an event which the parent will be listening for.

```
<button v-on:click="$emit('set-as-main', 'pt')">
  Set as Main
</button>

<table-item v-on:set-as-main="setAsMain($event)"/>
```

FIGURE 4.21: Component Communication in Vue

For example, if the intention is to alert the parent that a button has been clicked, the parent can add an event listener to the child component declaration and the button can, when clicked, emit a corresponding event to pass the information above. This is shown on figure 4.21.

Angular

Passing data to a child component is similar to Vue, as data received by the child should be declared. This is done by using the `@Input()` decorator on the child component, defining a set of parameters it expects to receive, along with their types. To declare that the component expects to receive a string called *name*, one would use `@Input() name: string;`.

On the parent, the data is passed down similarly to the other frameworks, by adding it to the declaration of the child component. For example, `<angular-child name="World"></angular-child>` would pass a name to the *angular-child* component.

Child components in Angular can communicate to their parents in a similar way to Vue, by emitting an event. To do so, the event must be declared by instantiating an `EventEmitter` object. Afterwards, the component can call its `emit()` method to pass data to the parent, who should have a corresponding event listener.

Verdict

React	Vue	Angular
Least verbose without the need to declare all data	No Considerable Advantage	No Considerable Advantage

TABLE 4.4: Component Communication

4.7.5 Conditional Rendering

While Vue and Angular use specific directives on that can be applied to HTML elements to define whether or not they should be rendered, React resorts to plain JavaScript to do so.

React

Conditional rendering in React works the same way conditions work in JavaScript. JavaScript operators like `if` or the conditional operator - as exemplified on Figure 4.22 - can be used to create elements representing the current state letting React update the UI to match them[15].

```
<div className="col-1 text-center">
    {this.props.lang.ismain ?
        <i className="fas fa-star languages__table-item--secondary" style={{cursor:'pointer'}}></i>
        :
        <i className="far fa-star languages__table-item--secondary" style={{cursor:'pointer'}}
        onClick={() => {this.props.setAsMain(this.props.lang)}}>
        </i>
    }
</div>
```

FIGURE 4.22: Conditional Rendering in React

[15]https://reactjs.org/docs/conditional-rendering.html - Conditional Rendering - React (Accessed September 4, 2018)

Vue

On Vue, conditionally rendering a certain element or component consists of adding the `v-if="condition"` directive to it. Doing this makes it so that the element is rendered only if the condition is verified as true. This can be a variable or an expression.

Angular

Similarly, Angular offers the `*ngIf="condition"` directive to determine whether or not an element should be rendered.

Verdict

React	**Vue**	**Angular**
JavaScript Operators	Useful, simple Directive	Useful, simple Directive

TABLE 4.5: Conditional Rendering

4.7.6 Iterative Rendering

Similarly to the previous point, looping through a data object to render it's elements is done in a much easier and less verbose way in Vue and Angular.

React

In React, iterating an object and rendering elements for each for its children makes use of vanilla JavaScript functions. Specifically, the `map()` function is used, allowing to, for each item in an object, run a function that takes it as an argument and returns an element.

For example, to loop through an object and render an empty `div` for each of its items, one would use `object.map(function(item)return (<div></div>)`.

Vue

In Vue, achieving this is simpler, and consists of creating an element to return, and adding to it the directive `v-for="item in object"`, as shown on Figure 4.23. Then, each item's properties can be accessed using `item.property`.

```
<template>
    <TableItem v-for="language in jsonData.languages"/>
</template>
```

FIGURE 4.23: Iterative Rendering in Vue

This results in rendering one of the specified element for each of the items in the object which the application is iterating.

Angular

In Angular, iterative rendering is identical to Vue, with the difference being purely syntactical. To achieve the same result, the directive used is `*ngFor="item in object"`.

Verdict

React	Vue	Angular
Verbose operator in comparison	Useful, simple Directive	Useful, simple Directive

TABLE 4.6: Iterative Rendering

4.7.7 Lifecycle

Another feature common to the frameworks is the lifecycle of a component. Each component has several "lifecycle methods" that can be overridden to run code at particular times in the process, such as when the component is created, immediately before, or when it updates.

React

React features lifecycle methods for mounting, updating and unmounting [16] such as `constructor()`, `componentDidMount()`, `componentDidUpdate()` or `componentWillUnmount()`.

Vue

Vue comes with lifecycle methods for creation / mounting, updating and unmounting [17] called `created()`, `mounted()`, `updated()` and `destroyed()`.

Angular

Angular features a similar set of lifecycle hooks as React and Angular for each of the three phases, namely `ngOnInit()`, `ngOnChanges()` and `ngOnDestroy()`.

Verdict

React	Vue	Angular
No Considerable Advantage	No Considerable Advantage	No Considerable Advantage

TABLE 4.7: Lifecycle

4.7.8 Event Handling

Event handling on elements is done similarly on the three frameworks. Angular and Vue keep up their habit of using specific directives, while React uses element properties on the HTML elements. This means the differences are very small and strictly about syntax.

[16] https://reactjs.org/docs/react-component.html - React.Component - React (Accessed September 4, 2018)

[17] https://vuejs.org/v2/guide/instance.html - The Vue Instance - Vue.js (Accessed September 4, 2018)

React

Handling events with React elements is very similar to handling events on DOM elements[18] with some small syntactic differences. Camel case is used rather than lowercase and the value passed should be a function, rather than a string. It can look something like Figure 4.24, where clicking the `a` element calls the `handleClick()` function.

```
function handleClick(e) {
  e.preventDefault();
  console.log('The link was clicked.');
}

return (
  <a href="#" onClick={handleClick}>
    Click me
  </a>
);
```

FIGURE 4.24: Event Handling in React

Vue

On Vue, the `v-on` directive can be used to listen to DOM events and run some JavaScript when they're triggered[19].

For example, adding `v-on:click="handleClick"` will make it so that clicking the target element will call the previously defined function `handleClick()`. The value passed to the directive is an expression, meaning that creating a separate handler method is not always necessary. In Vue particularly, just stating the name of the desired method is enough, with no need of the typical parenthesis that represent a function.

Angular

Similarly, in Angular, a directive can be added to an element to attach an event handler to it, with the difference being purely syntactical. For example, `<button (click)="onClickMe()">Click me!</button>` will make it so that clicking the target button will call the `onClickMe()` method. Similarly to in Vue, the passed value is an expression.

Verdict

React	**Vue**	**Angular**
No Considerable Advantage	No Considerable Advantage	No Considerable Advantage

TABLE 4.8: Event Handling

[18] https://reactjs.org/docs/handling-events.html - Handling Events - React (Accessed September 4, 2018)

[19] https://vuejs.org/v2/guide/events.html - Event Handling - Vue.js (Accessed September 4, 2018)

4.7.9 Project Weight

After development, the total project weight was measured for each of the frameworks used.

React

The React project totaled **157,5MB**.

Vue

The Vue project was the lightest of the three, at **118,7MB**.

Angular

Angular resulted in the largest project, with a total of **299MB**.

Verdict

React	**Vue**	**Angular**
No Considerable Advantage	Lightweight	Heaviest

TABLE 4.9: Project Weight

4.8 Summary

In this chapter, a case study was defined and developed, starting with the choice of a section of the Jumpseller application that showed some issues and would make for an interesting case study on all stages of Front-End web development.

Having selected the Languages section, an analysis was made of the current state of the page, listing its issues and discussing what could be done to improve on them. Once ideas were gathered, mock-up images were created in order to visualize what a new implementation of the page could look like. These images were brought to members of the design and development teams at Jumpseller, where they were discussed and feedback was gathered.

Once the agreed upon changes were made and the final product was envisioned, the interfaces were developed on the three selected JavaScript frameworks: React, Vue and Angular.

During and after the development stage, some conclusions were taken by comparing how certain aspects of development were handled on each of the frameworks. Aspects such as initializing a project, adding styles or configuring routes were compared and a verdict on which one featured considerable advantages on each area was presented.

Chapter 5

Validation - Jumpseller

This chapter aims to portray the author's experience as part of the Jumpseller development team, going over the work methodology and the projects worked on and technologies used during the work period. This also aims to further validate the approach developed in earlier chapters through practical experience in a real Web development setting.

5.1 Methods

5.1.1 Design and Development

The development of new user interfaces would begin on the design team, who would work with UI Design Tools to develop a mock-up of what new interface would look like.

Once complete, it would be presented to the development, encouraging discussion and the gathering of feedback.

This would be repeated until the final product was agreed upon, at which point the project would be passed to the developer, who would build the interface and integrate it with the Jumpseller application.

5.1.2 Shared Components

One of the big reasons to use a component-based framework such as React for the application Front-End is that it enables the sharing of code by different parts of the system.

With that in mind, it was a point of focus during the development process to keep the applications modular and separated in components that can be shared. An example of a shared component would be an image gallery / upload section, which will be part of the edition of a product, a category or a page, all these being different sections of the admin panel that share a common component.

This mostly brings benefits in terms of readability and code weight (as it avoids duplicate code and therefore wasted space), the latter being very valuable when dealing with a relatively large project on an enterprise context.

5.2 Work Projects

In this section, the projects developed during the work period at Jumpseller will be presented, along with their motivation, requirements, technologies used, a brief description of their implementation process and some screenshots showing the final product.

This section aims to help in further understanding the problems commonly faced in front-end development in a real world setting, as well as setting requirements and observing how the technologies used respond to those issues. All of this allows for a better overall understanding of front-end development and the technologies used, helping the author take conclusions about them.

5.2.1 Theme Options Remake

Rebuilt, from scratch, the Theme Options section of the Jumpseller application. On this section, users can customize the theme of their online store by changing a customizable set of options to their liking.

The Problem

Before, the Theme Options section had some design issues, as the navigation between its parts was often unintuitive, had too many buttons or was visually unappealing.

More importantly, some of its features were very difficult - or near impossible - to work with on devices with smaller screens, as the page content did not scale properly resulting in a poor user experience.

As such, it was decided that the entire section would be rebuilt from scratch.

Project Requirements

- Device Compatibility : All options and functionality should work on mobile devices, as the interface should adapt to different-sized screens.
- Ability to link to specific option set : Users should be able to get a link to a specific part of the options list, to make collaborative work easier.
- New design : As the previous design was very old, the new section will have a new and improved design. This is done in collaboration with a designer from the Jumpseller team.

Technologies Used

- React
- React Plugins (React-Color[1], React-FontAwesome[2], React-Router[3], React-Dropzone[4])
- Bootstrap 4

[1]https://casesandberg.github.io/react-color - React-Color (Accessed Semptember 11, 2018)

[2]https://github.com/danawoodman/react-fontawesome - React-FontAwesome on Github (Accessed Semptember 11, 2018)

[3]https://github.com/ReactTraining/react-router - React-Router on Github (Accessed Semptember 11, 2018)

[4]https://react-dropzone.js.org/ - React-Dropzone (Accessed Semptember 11, 2018)

- JSX, HTML5, CSS3
- Webpack

Implementation

The Theme Options section is independent from the rest of the store administration panel. As such, the new Theme Options was developed as a standalone React App.

To setup the basic app, the **create-react-app**[5] utility was used. This tool automatically creates the files needed for a basic React app to start running and be further developed.

Since responsiveness and device compatibility is the main requirement and, according to the methodology presented, it was in mind since the inception of this project. As such, the next step was to include the **Bootstrap** framework in the app.

With the foundation set, work started on the basic layout of the application, eventually bringing in functionality with sample data.

After most of the intended functionality was achieved, the developed app was integrated into the Jumpseller project, with the help of the development team. Once integration was done, the application would start working with the Jumpseller API to fetch and send real data. This brought some issues and posed an interesting challenge.

In parallel with the last stages of development, the design team at Jumpseller worked on CSS styles that would help perfect the new design of the application.

Once everything looked complete, the application was ready to be deployed and made public to the merchants.

After the application was deployed and made public, its utilization by hundreds of clients brought to the development team's eyes some issues in its functioning. As such, after the release some time was dedicated to fixing bugs and improving parts of the application.

Final Product

Screenshots of the final product of the Theme Options section can be found on Appendix A - Theme Options Screenshots.

5.2.2 Facebook Messenger Jumpseller App

Created, via the Jumpseller Apps platform[6], an application which can be installed on merchants' stores that allows them to log into their Facebook accounts and link their store's pages to Jumpseller. This gives Jumpseller permission to send messages through those pages, with the objective of giving customers the possibility to receive updates on their purchases, orders and deliveries via Facebook Messenger.

The Problem

When an order in an online store is placed, the store will sometimes follow up that order with e-mail messages. For example, alerting the customer when their order has shipped or if there is a problem with the order or the payment.

[5] https://github.com/facebookincubator/create-react-app - Create React App (Accessed Semptember 11, 2018)

[6] https://jumpseller.pt/support/apps/ - How to build a Jumpseller App (Accessed Semptember 11, 2018)

While this has been the standard for a while, it is a fact that the average customer often pays more attention to their Facebook account than to their e-mail inbox. This means that e-mail may not be the best way for stores to create a means of communication with their customers.

As such, the Facebook Messenger App was developed as a way to allow merchants to improve communication with their customers.

Project Requirements

- Customizable Opt-In message : Merchants should be able to customize the message that the client receives when they opt-in to the feature.
- Minimal Permissions : In order to function, the App needs to obtain some permissions from the user. It's essential that the app does not ask for more permissions than those it needs to function, as to protect each client's personal information.
- Usability : The App should be autonomous and functional in sending the messages. These messages should be simple, readable and with a good presentation.

Technologies Used

- Facebook for Developers
- Jumpseller API
- React
- Bootstrap 4
- JSX, HTML5, CSS3
- Webpack
- Ruby
- PostgreSQL

Implementation

The user application was developed as a standalone React App. As such, the project was created using the **create-react-app** utility.

Besides this React App, a Facebook App[7] was created. This is a necessary step to get access to the Facebook API and its functionalities.

The next step was integrating the Facebook SDK into the project. With this came a good amount of investigation about Facebook for Developers and how the API works.

With the help of the development team, a Ruby controller was developed to provide a back-end for the application. This was done using the Sinatra[8] framework, which allows for very simple creation of web applications and endpoints.

This controller also included interaction with the database. The database management system used in Jumpseller is PostgreSQL. For testing purposes, ngrok[9] was used for creating secure tunnels to localhost, ths allowing the development team to test the Facebook App while it runs on a local development machine.

Deploying the application and making it public to all stores and merchants meant that a large number of users would interact with it. As such, a support page[10] was written, with instructions on how to use configure it.

Following the deployment and the influx of users, some bugs came to surface which mean the following weeks involved support bugfixing tasks.

Being more than a front-end issue, this project posed a good, interesting challenge and allowed the author to better understand the behavior of the Jumpseller back-end. This will undoubtedly be useful on future projects where interaction between front-end and back-end may play a key role.

[7] https://developers.facebook.com/ - Facebook for Developers (Accessed September 11, 2018)
[8] http://sinatrarb.com/ - Sinatra (Accessed September 11, 2018)
[9] https://ngrok.com/ - ngrok (Accessed September 11, 2018)
[10] https://jumpseller.com/support/facebook-messengerapp/ - Facebook Messenger App (Accessed September 11, 2018)

Final Product

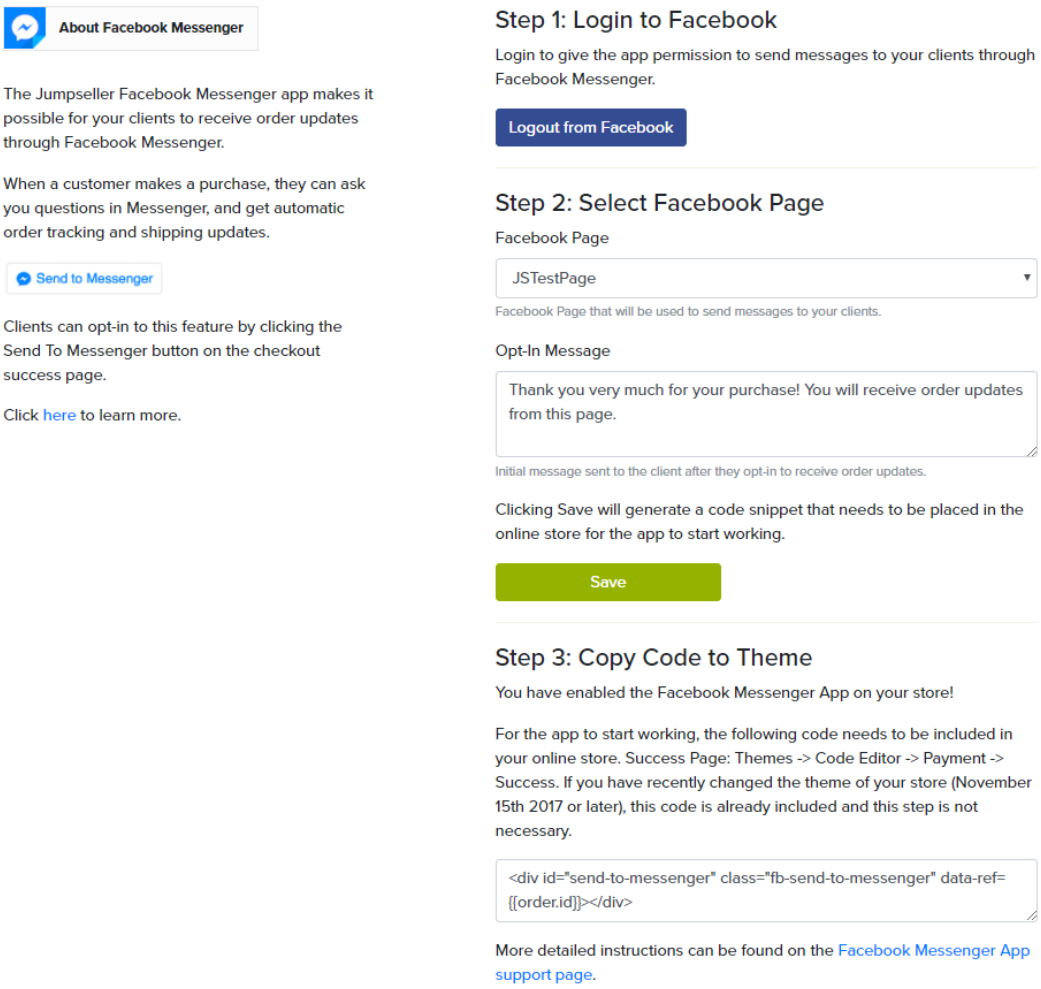


FIGURE 5.1: The Facebook Messenger Jumpseller App.

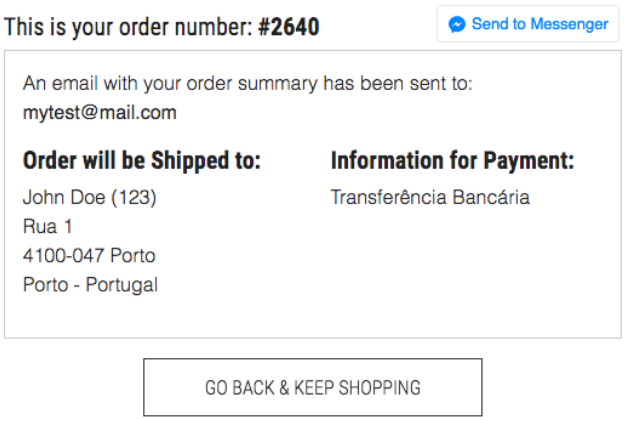


FIGURE 5.2: The Send-To-Messenger button included in a store.

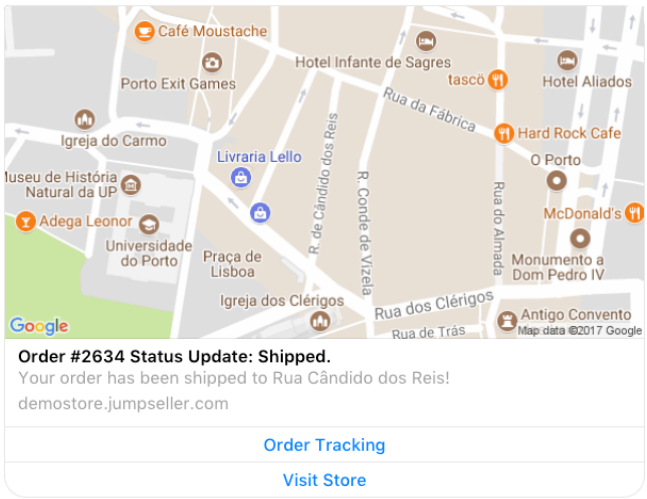


FIGURE 5.3: An example of a message sent automatically by the app.

5.2.3 New Admin Panel Menu Layout

Remade the sidebar menu layout of the Store Admin panel with a better design and, most importantly, more mobile-friendly interface.

The Problem

As each of the sections of the admin panel get re-made with better design and functionality, the sidebar menu that allows navigation between those sections would have to be re-done to keep up with the evolving designs and content.

As such, the menu was remade with a new, improved design that focuses on readability and usability on devices of all sizes.

Project Requirements

- Compatibility with both Bootstrap 3 and Bootstrap 4: As the new layout will be in use in a time period during which all the administration panel sections are being remade individually - thus migrating from Bootstrap 3 to Bootstrap 4 -, this sidebar should look and function correctly regardless of the version of Bootstrap the section is using.
- Device Compatibility: All options and functionality should work on mobile devices, as the interface should adapt to different-sized screens.

Technologies Used

- HTML5, CSS3
- Bootstrap 3
- Bootstrap 4

Implementation

Being a menu that will be displayed on the same page as many different standalone applications, it was decided that this would be a simple HTML/CSS menu with very minimal snippets of JavaScript, using only Bootstrap as a framework for design purposes.

Two versions of the same project were created, each one importing a version (3 and 4) of Bootstrap.

Starting with the basic layout and slowly adding the details, the projects were developed in parallel, with the intent of having them look exactly the same. The new design was based on mock-ups created by the Jumpseller design team.

Once the design and basic functionality was complete, the menu was integrated in the Jumpseller project by members of the development team.

Final Product

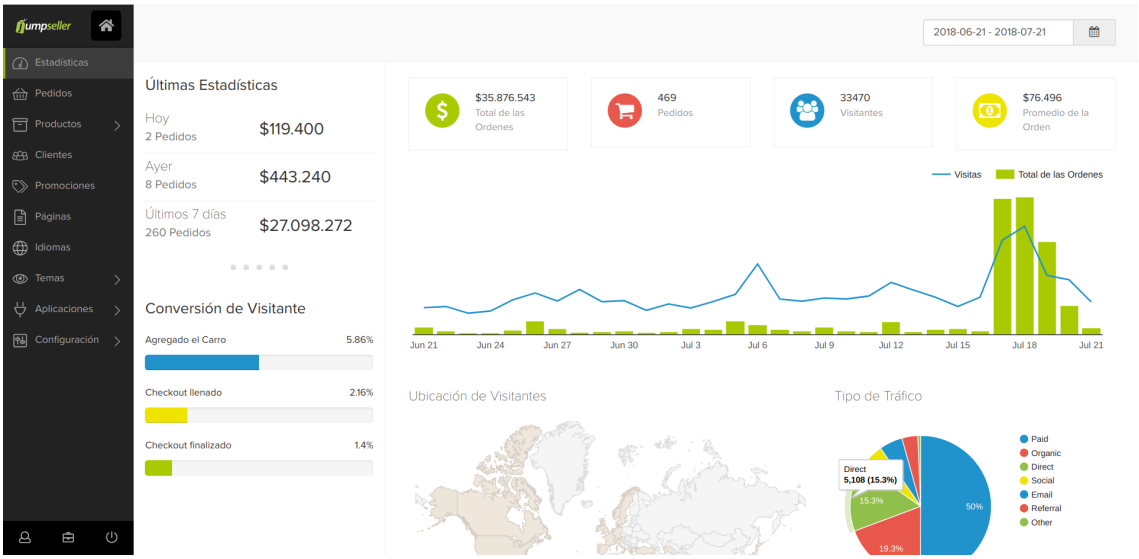


FIGURE 5.4: Desktop version of the new admin panel menu.

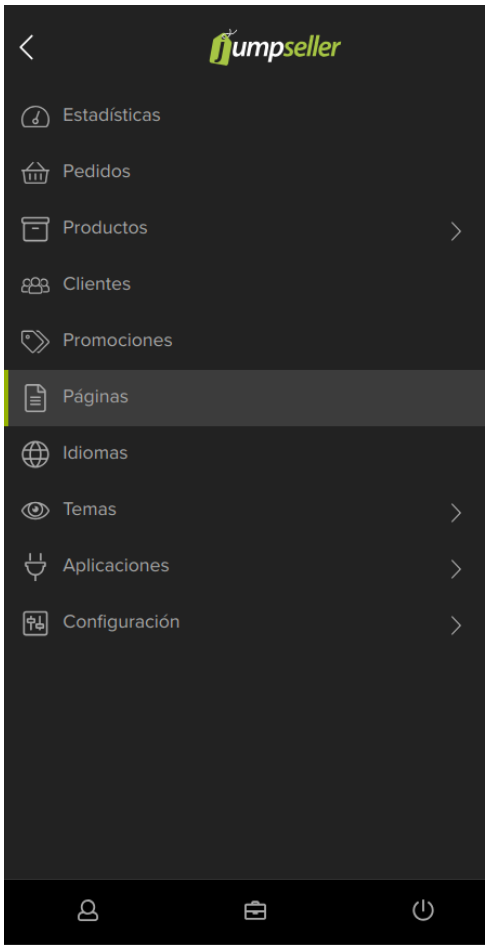


FIGURE 5.5: Mobile version of the new admin panel menu.

5.2.4 Product Categories Section

Created a new section in the admin panel for merchants to create and manage the categories under which their products are labeled. This functionality was previously on the same section as the product listing, making for an overly complicated section.

The Problem

The product category management section of the store administration panel was previously in the same page with the product listing and management section. This made for an overly cluttered, busy and complicated interface.

Besides that, the category management part itself was very poorly designed, in such way that too many clicks were required to do simple actions like editing a category or moving it around in the list.

This section was also very bad to work with on mobile, as it content did not scale properly requiring the user to navigate to the edges of the screen to find the desired actions.

It was decided that a new section would be built from scratch, with a simple, elegant design and very fast and easy functionality.

Project Requirements

- Drag-and-Drop: Users should be able to re-order the list of their store's categories by dragging and dropping items in a list. This order should be saved and remembered on the next visit.
- Hierarchy: Categories can have sub-categories. This should be kept in mind, allowing users to specify a parent category when creating a new one. It should also be compatible with the drag-and-drop feature.
- Searching: Users should be able to search for a specific category or set of categories by entering a search term.
- Device Compatibility : All options and functionality should work on mobile devices, as the interface should adapt to different-sized screens.
- New Design: The new section will have a new and modern design, done in collaboration with a designer from the Jumpseller team.

Technologies Used

- React
- React Plugins (React-Router, React-Dropzone[11])
- Nestable 2 [12]
- Bootstrap 4
- JSX, HTML5, CSS3, SASS
- Webpack
- Ruby

Implementation

As with previous React-based projects, the section was bootstrapped using **create-react-app**. After that, the **Bootstrap 4** framework was included in the project to enable responsive behaviour. Work was firstly done on the **categories list**, before moving on to the category edition page.

The first focus was on creating the page layout, keeping in mind the its separation in components such as the search area, the list and the list item. At this point mock data was being used and, as functionality started being brought into the application, the hierarchic drag-and-drop feature presented itself as the biggest challenge. Research was done and after experimenting with different plugins, it was chosen that the application would use **Nestable 2**, as this is a modern version of the plugin used on the old Jumpseller build and offers a way to implement all the desired features. The plugin works by annotating DOM elements with class names such as `dd-list` and `dd-item`, and initializing the plugin with a callback function, to be called whenever a change to the structure is made. On this callback method a request is made to the back-end via HTTP to update the affected categories' order and hierarchies.

[11] https://react-dropzone.js.org/ - React-Dropzone (Accessed September 11, 2018)
[12] https://github.com/RamonSmit/Nestable2 - Nestable 2 (Accessed September 11, 2018)

When functionality appeared done, the page was put to test with real data from some of the busiest and more complex stores. At this point, and because some stores have a large number of products and categories, performance presented itself as an issue. Searching, for example, took an unacceptably long time. This happened because a search request was being sent each time the user wrote a character in the search input field. This resulted in too many requests being sent, clogging up the back-end. As a measure to prevent this, it was made so that the search request would only be made when the user stopped typing for 2.5 seconds, ensuring only the necessary requests were made.

The mobile version of this section involved a small amount of changes, as its content is fairly small and doesn't cause many issues on smaller screens. Media queries were added to change the appearance of the delete category button and the page header.

The **category edition page** was implemented after the launch of the category list, and was created as a separated application. Its layout and content were of simple implementation, as they mostly consisted of input fields. The biggest challenge faced was the routing, as the page URL should reflect the category it corresponds to or refer to the creation of a new category.

As such, and using **react-router** as previously, a method called `urlControl()` was included and is called before rendering the page by the `componentWillMount()` lifecycle hook. On this method, the URL is checked and, depending on the category id found there, the page will present its information or open up the option to create a new category.

Final Product

Screenshots of the final product of the Product Categories section can be found on Appendix B - Categories Screenshots.

5.2.5 Products Listing

Re-made the products section of the admin panel, where products are listed and can be managed, sorted, filtered and searched.

The Problem

The product listing section of the admin panel of the Jumpseller app used an old layout which was cluttered, unresponsive and not visually appealing.

This is an extremely important section of a store's administration, as organizing the products they are selling is a big priority of a merchant. As such, it was very important that all functionality was simple to use and presented in a clean, simple way, and that all of it worked on a full range of devices.

Project Requirements

- Filtering: Products should be able to filtered by name (with a simple search input), categories they're inserted in, and status (available, out of stock, etc.).
- Sorting: The order in which products are displayed should be able to be customized by integrating a drag-and-drop feature to the list.

- Performance: Stores using Jumpseller can have up to thousands of products. This should be kept in mind during development, to make sure that the application doesn't have overly long loading times and isn't slow when performing actions on a large number of products.
- Usability: Actions such as changing status, adding to a category or deleting can be performed on products. It's an important requirement that these actions are easy to perform to the average merchant, whether they are applied to a single product or to many of them. This includes the ability to selecting specific products or selecting all and performing bulk actions.
- Device Compatibility: All the actions should be able to be performed just as easily on mobile devices, as product management represents one of the most important aspects of managing an online store and should therefore be available on any device.
- New Design: The new product list should follow a design mock-up created by the design team, with a clean and modern look that fits with the overall aesthetic of the Jumpseller product.

Technologies Used

- React
- React plugins (React-Router, react-select[13], React-Modal[14])
- Bootstrap 4
- JSX, HTML5, CSS3
- Webpack
- Ruby

Implementation

This was a project that was picked up halfway through its development and had been started by a former colleague at Jumpseller. As such, the process and the learning experience was slightly different for the author. This meant some time was spent learning how the currently implemented features worked and how the code was structured. Along with this process, the author was able to identify anti-patterns - such as the "lava flow"[15], code that was at some point used but is not currently, making it useless - that could easily be fixed, setting a better starting point to work on.

At the point of picking up this project most of the basic functionality was done, and the remaining work started with design changes and optimizations to the code.

As, again, the application will be dealing with a large amount of data - as stores can have up to thousands of products to list, performance was a big thing to be kept in mind. One of the decisions made in this regard was that the application would retrieve 30 products at a time from the database. This means that when an user opens up the product list, only 30 products are displayed immediately. If they wish

[13]https://github.com/JedWatson/react-select - GitHub - react-select (Accessed September 5, 2018)
[14]https://github.com/reactjs/react-modal - GitHub - React-Modal (Accessed September 5, 2018)
[15]https://sourcemaking.com/antipatterns/lava-flow - Lava Flow (Accessed September 6, 2018)

to see more, scrolling down to the bottom of the page will load 30 more, until all the products have been loaded. This avoids making a request to the database that will return all of the products as this is both heavy on the database and will lead to a long loading time on the front-end application. It should be noted that bulk actions can still be performed on all products, even if they have not all been loaded, by having a choice to select products individually, select all loaded products, or select all the products in the database. To the same effect, the search function is also set to only make a request if the user has finished typing in the input field, by using a request timer that waits 2.5 seconds after the last character is inserted.

Final Product

Screenshots of the final product of the Product List can be found on Appendix C - Product List Screenshots.

5.2.6 Product Edition

Re-made the product edition page, where a selected product can be edited in terms of its basic properties (name, description, etc), shipping properties, images, product options, custom fields, product attachments and SEO.

This is a page with a large number of features and high complexity, and represented the most ambitious and challenging of the projects developed by the author at Jumpseller.

The **product options** section in particular is of high importance and complexity, as it allows merchants to add different options (such as sizes, colors, patterns, etc) and automatically generates a table of product variants with all possible combinations of these options. This is exemplified on Figure 5.7 (with the old design).

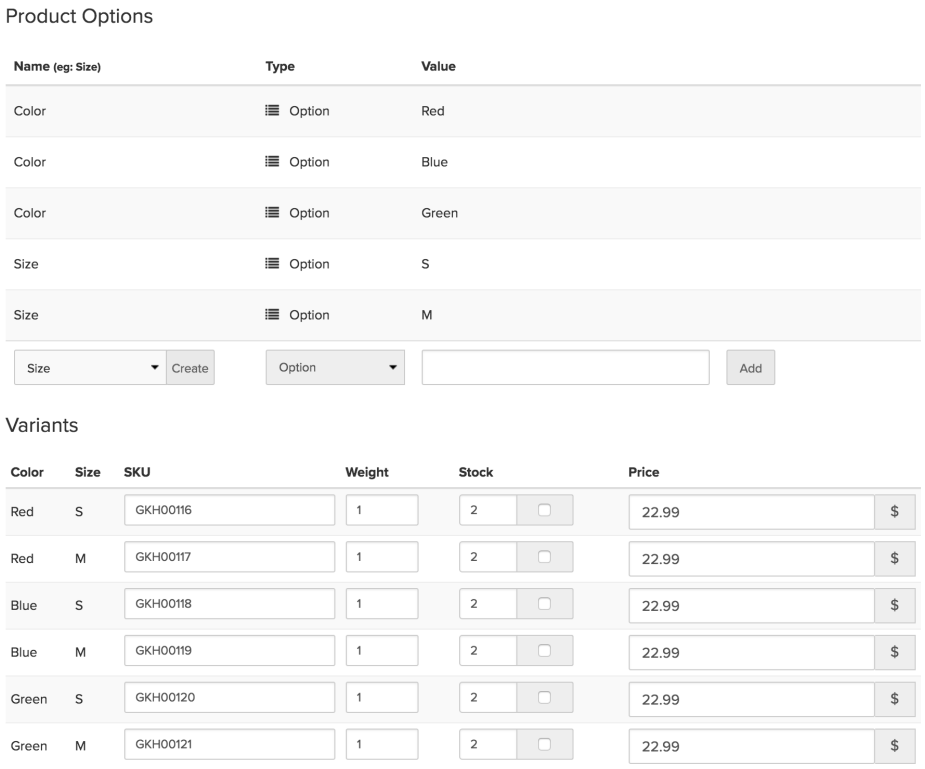


FIGURE 5.6: Product options on the old design.

The Problem

The main problem with the product edition page in the Jumpseller application is its usability in mobile devices. Some of the functionalities are very difficult to impossible to use on smaller screens as the interface is not responsive. Being one of the most feature-rich and important sections of the admin panel, it's important that this it is re-done with a responsive, clean and usable interface.

Project Requirements

- Category Management: When editing a product, it should be easy to specify which categories it's inserted in, or create a new one if necessary.
- Image Gallery: The product images should be able to be uploaded and presented in a grid layout, where they can be sorted, edited or deleted.
- Variants Table: The variants table, generated automatically with the product options set by the merchant, can become quite large as the number of options grows. It's important that performance is not an issue in this section, and that the table is readable and usable in all devices and screen sizes.
- Custom Fields: Adding custom fields to a product (such as specific product characteristics, comparisons or labels) on the old design is a laborious and unintuitive process, with multiple modal windows popping up. The process should be made more intuitive and simple, with a cleaner design.
- Tags input: For both the product options and custom fields, multi-value fields should have only one input field, where users can add comma-separated values.
- Device Compatibility: All actions and functionality should work on mobile devices, as the interface should adapt to different-sized screens.
- New Design: The new product page should follow a design mock-up created by the design team, with a clean and modern look that fits with the overall aesthetic of the Jumpseller product.

Technologies Used

- React
- React plugins (React-Dropzone, React-Tagsinput[16], React-Modal, React-Select)
- Redactor Editor[17]
- Bootstrap 4
- JSX, HTML5, CSS3, SASS
- Webpack
- Ruby

Implementation

It was decided that, rather than creating a new application from scratch using create-react-app as previously done, the product list application would be refactored to include a new page where products could be created or edited. As such, that was the first step on implementing the new product edition page, followed by the creation of the basic page layout according to the mock-up images developed by the design team.

Being a very functionality-rich page with several different components, all complex on its own, the implementation of each of its features was done in a more planned and pondered way, taking focus on each of its components at a time.

- **Product and Shipping Properties:** The first component of the page to be implemented to its full functionality after creating the basic layout was the section where users can edit product properties such as name and description, as well as shipping properties such as height, weight, or package format. These mostly consisted of simple input or select fields, with the exception being the description field, where the Redactor Editor was implemented by including a previously-created React component which uses the editor's source code.
- **Image Gallery:** Users can upload multiple images to represent each product. For the uploading, a previously-created component was used, where the React-Dropzone plugin was implemented. This allows users to upload one or multiple images at a time by dragging them onto a field or manually selecting the files. Uploaded images should were set to be displayed as squares in a grid layout - using CSS properties - with buttons that allow users to either delete or edit the image. Clicking the edit button launches an image editor which is part of the Redactor Editor. Finally, it was decided that if more than 6 images are uploaded, only the first 6 are shown at first, with the visibility of the remaining images being toggled by a "Show More/Less" button.
- **Custom Fields:** Merchants can add custom fields to their products to display additional information about them, such as comparisons to other products, extra labels or instructions. The most important aspect of the re-design was to simplify the interface. This was achieved by including the list of custom fields

[16] https://github.com/olahol/react-tagsinput - GitHub - React-Tagsinput (Accessed September 5, 2018)

[17] https://imperavi.com/redactor/ - Redactor WYSIWYG html editor (Accessed September 5, 2018)

and corresponding values on the main page and using a modal to display the field management options.

- **Product Attachments:** Users can attach digital files to their products. This usually refers to digital products such as music or films, but it was decided some changes would be made to accommodate a different scenario: Attaching public, downloadable files to physical products, such as instruction manuals for an electronic product. In terms of front-end this was quite simple, as it consisted in two image uploading sections and, for each, a list of the uploaded files, where they can be deleted or their names changed.
- **Product Options and Variants:** The product options section allows users to add custom options to their products, such as size or color, and automatically generates variants for every combination of those options, each of which having properties which can be edited. This is a very complex component, as with every change to the options list calculations need to be done to manage the variants that come from them and the table should automatically update accordingly. In order to simplify this process for the user, a big change was the implementation of a **tags input field** system that allows users to quickly add values to their options by simply adding the values in an input field, separated by a comma. Device compatibility was also a very important aspect of this section, as the old design didn't allow users to edit variant properties on mobile devices. As such, media queries were used to change how the variants table looks on smaller screens to, according to the design created by the design team, allow users to have access to all functionality of this page on smaller screens.

After the conclusion of each component, the application was reviewed and tested by the team, which led to the discovery of small bugs and issues and some discussion of improvements. As such, the full implementation of the product edition page has been a lengthy process which is approaching its conclusion at the time of writing this document.

Final Product

Screenshots of the final product of the Product Edition page can be found on Appendix D - Product Edition Screenshots.

5.3 Conclusions

In the following sections, conclusions will be taken about the work period and projects taken part of on the Jumpseller platform, in regards to the technologies used and how the experience with them contributed to the dissertation, as well as what was learned and how the work done contributed to the Jumpseller application and its development team.

5.3.1 Technology

The main technologies used for the interfaces built during the work period at Jumpseller were React and Bootstrap 4. This section aims to reflect upon the choice of these technologies and what made them a good option.

With the proposed approach in mind, React was chosen as the JavaScript framework to work with, as its component-based nature, combined with its high performance and how easily it handles routing and complex component hierarchies made it look ideal given the company's requirements of performance and code reusability. The fact that many different applications with varying features would be built made the fact that React is a widely popular framework with a very large amount of community-made resources extremely valuable.

Code Sharing

Creating shareable and re-usable code was, from the beginning, one of the aspects to keep in mind during development, as it was one of the reasons to adopt a component-based framework to work with.

During the work period, several components were created with the intent of being used in multiple parts of the store administration panel, such as the `ImageSection` component, which implements a section where users can upload and manage images, the `SEO` component that refers to the Search Engine Optimization properties of a product, page or category, the `RedactorEditor` component that allowed the easy implementation of the Redactor Editor on any page or the `ShareLinks` component, that easily adds links to share the current page on various social networks.

In conclusion, the choice of React as a JavaScript framework allowed and incentivized the principle of code sharing and re-usability to be respected and implemented.

Plug-ins

Being a very popular framework, React has a lot of people contributing with their own plug-ins. Throughout the work period, various plug-ins were used in response to specific needs or issues that came up during development. This includes React-Router, the plug-in used to creating routing within the developed applications, React-Dropzone, used for image uploading or React-Modal, used for creating modal menus.

This large amount of available plug-ins meant, during development, that there was a solution already made for almost any issue that could be run into. With a younger, less popular framework, this might not have been the case. As such, React was a positive choice in this regard.

Bootstrap Variables

Bootstrap felt like a good choice as a styling framework, not only for its core functionalities as a way to build responsive page layouts and its array of auxiliary classes, but also for how customizable it is.

Since work was set to be done on multiple sections of the Jumpseller application, the ability to customize certain aspects of Bootstrap such as the colors used on certain headings, border styles or the usage of gradients was very helpful. This was achieved by creating a SCSS file where these customizations were made, and then importing that file in all the developed interfaces.

As such, Bootstrap's ability to be customized combined with its usage of SASS variables made it a good choice in terms of customizing and convenience.

5.3.2 Contribution and Learning Experience

The hands-on experience with front-end technologies acquired at Jumpseller contributed to this dissertation by further validating and verifying the analysis made during the work period.

Working with a component-based JavaScript framework such as React in a real world application environment highlighted the benefits of developing an interface in separate, highly re-usable components. This also represents an advantage to the company's interests, as it allows future projects to implement this already made code, reducing the time and effort said project may take.

React was the main technology used in Jumpseller, and this practical experience validated the points made during the analysis part of this dissertation, such as its large community and amount of community-made extensions and plugins or how actions can be performed and goals can be achieved within the framework.

Learning the intricacies and best-practices of each framework and of front-end web development in general allowed for not only better code to be implemented, but for a knowledge base to be built upon which new developers can work on their assigned projects, which highlights the author's and this work's contribution to the company.

In a practical and measurable contribution, the number of customer support tickets related to the re-worked interfaces attributed to the author was very low in comparison to other parts of the administration panel and would be reduced to zero shortly after the release of each one, as any issues that existed would be promptly fixed. This is, from the company's stand point, a proof of an implementation of a well-designed and well-functioning interface.

Furthermore, the work experience at Jumpseller was a very significant learning experience for the author, as it allowed for technologies related to all the parts of a web application to be explored, for concepts such as version control (Loeliger and McCullough, 2012) and continuous integration (Fowler and Foemmel, 2006) in the context of a real-world application to be appreciated, and for gaining familiarity with work methodologies and frameworks.

5.4 Summary

In this chapter, an overview of each of the work projects taken part of by the author at Jumpseller was made, focusing on what the problems were and what was done to fix it. Finally, conclusions were taken in terms of what was learned and how the practical experience with these technologies helped fulfill the goals of this dissertation, as well as in what ways did this work contribute to the company and how the chosen technologies fit their needs.

Additionally, the work done in the context of this chapter provided practical use cases where the approach developed on Chapter 3 could be validated, by seeing some of the technologies in action and how well they responded to the issues and requirements of each project.

Chapter 6

Conclusions

In this dissertation, the author integrated the development team at Jumpseller, to work hands-on with modern front-end web development and some of the technologies it involves.

Motivated by the work to be done at Jumpseller, the concepts of modern, responsive web development and the constantly growing array of technologies that exist in the field to tend to the developers needs, three main objectives were set for this work.

Firstly, a state of the art analysis was made, by analyzing relevant front-end technologies and concepts and selecting the most popular frameworks. As a large and growing number of choices and possibilities in terms of technologies can both be intimidating and represent an obstacle in development, making an initial selection was an important step. Then, the selected technologies were compared and analyzed more deeply by evaluating their main features, strengths and weaknesses, in order to develop an approach which could be used to help developers to choose the most appropriate ones for a certain project, given a set of requirements. This approach was then validated in two steps, firstly through the definition and development of a case study and secondly through the implementation of developed components on the Jumpseller application as part of the development team.

The technological context in which this dissertation is included was first set by providing some key ideas and concepts about the current state of Web Development and Front-End technologies. In Chapter 2, important concepts such as Responsive Web Design, Frameworks and Automated Testing were introduced.

An overview of the most relevant Front-End frameworks was presented, taking usage data as the major factor for selecting the set of frameworks for analysis. The selected frameworks were analyzed by finding their main features, evaluating their strengths and weaknesses and taking conclusions about what makes them the most popular.

A case study was developed on Chapter 4, with the intent of putting the previously analyzed frameworks to a practical analysis that would allow for a better understanding of the strengths, weaknesses and quirks each one might carry. To achieve this, the plan was to find a problematic section within the Jumpseller platform and rebuild it from scratch, going through all the steps: Starting with an analysis of what the problems were, UI/UX Design and creating mock-up images, gathering feedback and implementing it on each of the selected frameworks. This allowed for conclusions to be taken about each of them, by comparing how certain features were implemented on each project and how the frameworks responded to common issues.

The technological analysis and comparisons made, as well as the developed approach were put through further validation on Chapter 5, where the author's experience as part of the development team at Jumpseller is portrayed by going over the

work methodology, the projects worked on and the technologies used during the work period.

6.1 Discussion

The detailed analysis and comparison made between the most popular front-end web development frameworks, along with the case study that was defined and implemented allowed for a better understanding of how the selected technologies work, and how each of them responds to common issues and requirements in front-end web development.

These findings allowed for an initial approach to be developed and to act as a guideline on when to use which of the analyzed frameworks. The diagram that was created points developers towards React, Angular or Vue.js, based on which requirements their project has or what the developer's background in programming looks like.

This initial approach represents an open-ended part of this work, as it is currently a foundation for future work to be done. The requirements shown in the diagram can be expanded and refined, and their associations with each framework can be made more accurate and justified.

This approach, while only an initial idea, has some limitations. The number of frameworks included could and should be larger in order to yield results that are accurate and contemplate a bigger scope of the available technologies. Additionally, the approach should cover a wider set of requirements, to ensure it can be used on a large number of projects with a large variety of given requirements.

The analysis made and the conclusions taken, along with the proposed initial approach were validated by the two practical components of this dissertation: The case study that was developed and the author's experience as part of the Jumpseller development team. While the latter method worked as a way of validation for this work, it also represents a very specific domain, meaning that in a different area of study the results obtained might have been different.

This work also provided a significant contribution to the company, as the combination of the analytical and the practical sides of this dissertation contributed to the development of well-implemented, functional interfaces with re-usable code and components as well as a knowledge base upon which future developers can learn from and work on their assigned projects.

The company-scoped objectives set at the beginning of the work period of making the application a better user experience by analyzing its issues, researching possible solutions and implementing new interfaces were completed, and validated by the growth of the company and its user base.

6.2 Future Work

In the future, and as a continuation of this work, more technologies such as Ember.js[1], Backbone[2] or Bulma[3] will be explored, analyzed and set to a comparison with the current leading frameworks.

The method of analysis will be expanded with the implementation of more and more significant tests, in order to obtain more meaningful results that will help in the decision making process in what comes to finding the right tools for the product to be developed. This includes, for example, the use of tools such as Selenium[4] or Mocha[5].

The developed initial approach needs to be continued to be worked on, as it currently represents a foundation upon which a better and more comprehensive approach can be built that will serve as a guide to choosing a front-end JavaScript framework.

In the context of Front-End development at Jumpseller, work will continue to be done on the administration panel, further exploring the experience acquired during the work period of this dissertation and using metrics like the number of support tickets as a way to verify the improvements made to the interfaces.

Finally, practical validation will continue to be done in different projects, both within and outside of the scope of e-commerce, as the concepts and good practices discussed and analyzed apply to all faces of front-end web development.

[1] https://github.com/emberjs/ember.js - Ember.js on GitHub (Accessed October 18, 2018)
[2] https://github.com/jashkenas/backbone - Backbone on GitHub (Accessed October 18, 2018)
[3] https://github.com/jgthms/bulma - Bulma on GitHub (Accessed October 18, 2018)
[4] https://www.seleniumhq.org/ - Selenium (Accessed October 18, 2018)
[5] https://mochajs.org/ - Mocha (Accessed October 18,2018)

Appendix A

Theme Options Screenshots

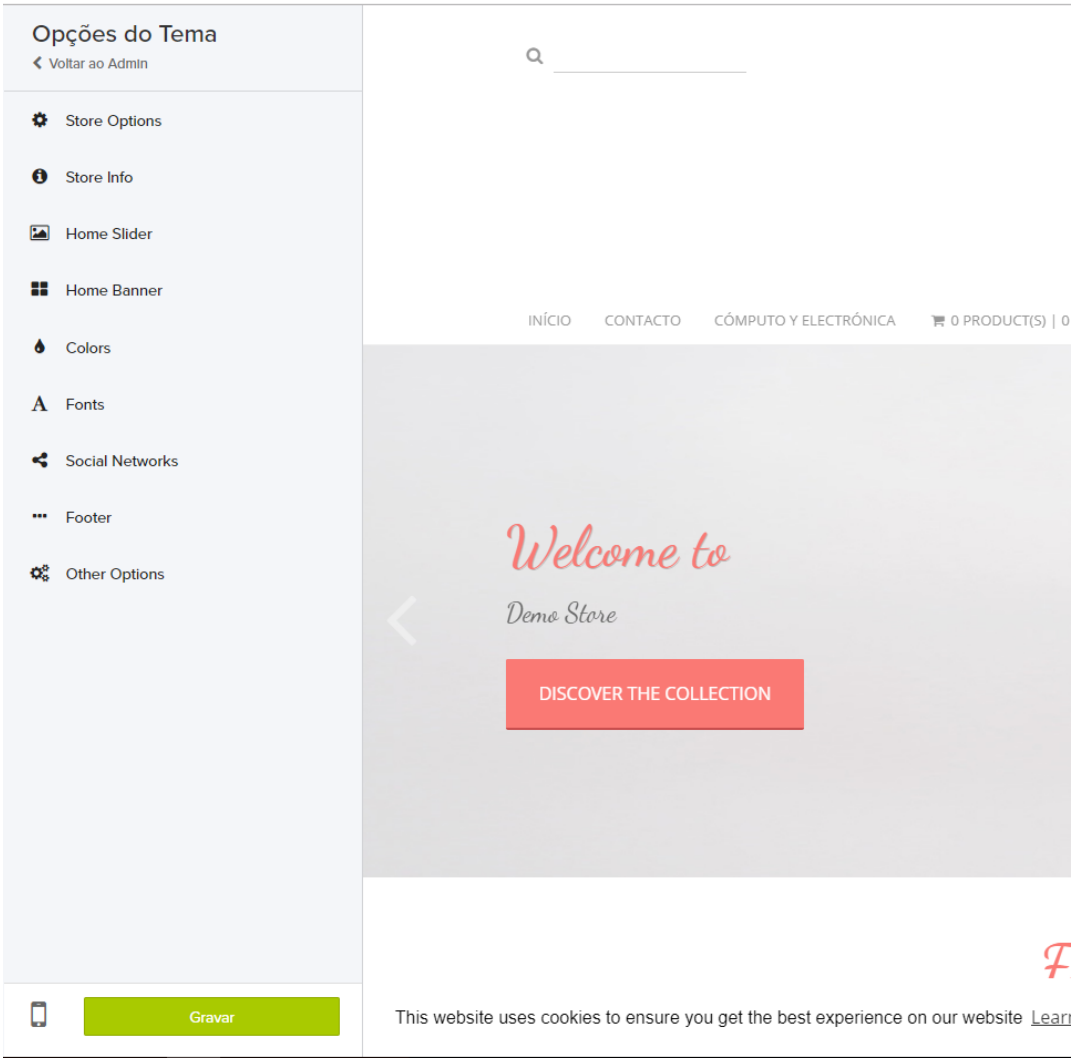


FIGURE A.1: The Theme Options App - Desktop version (Cropped).

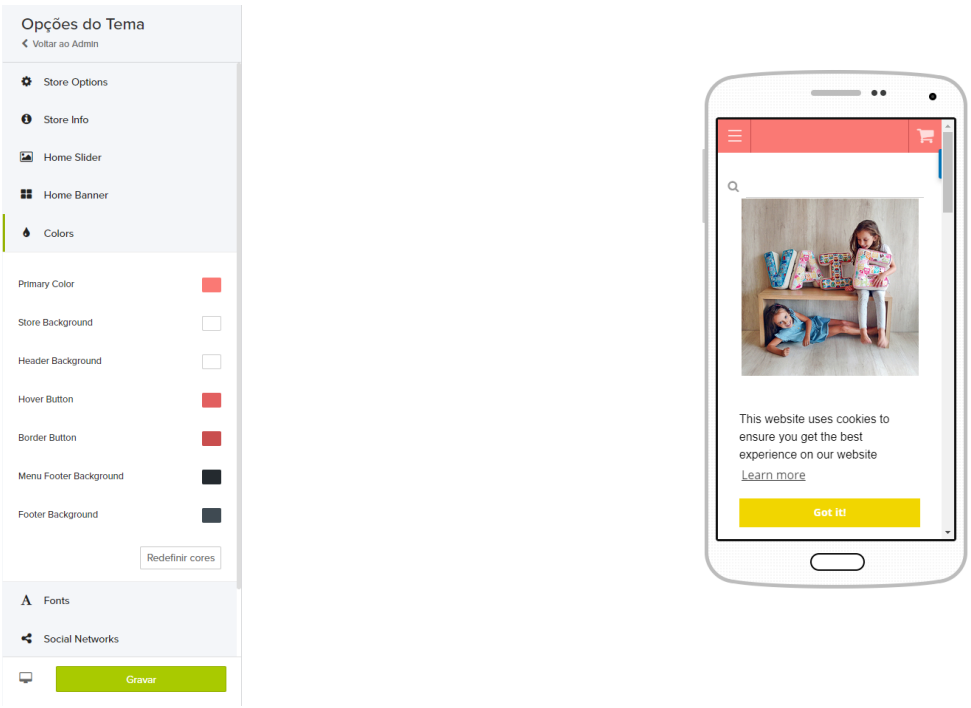


FIGURE A.2: The Desktop version allows you to preview how the store looks when accessed from a mobile device.

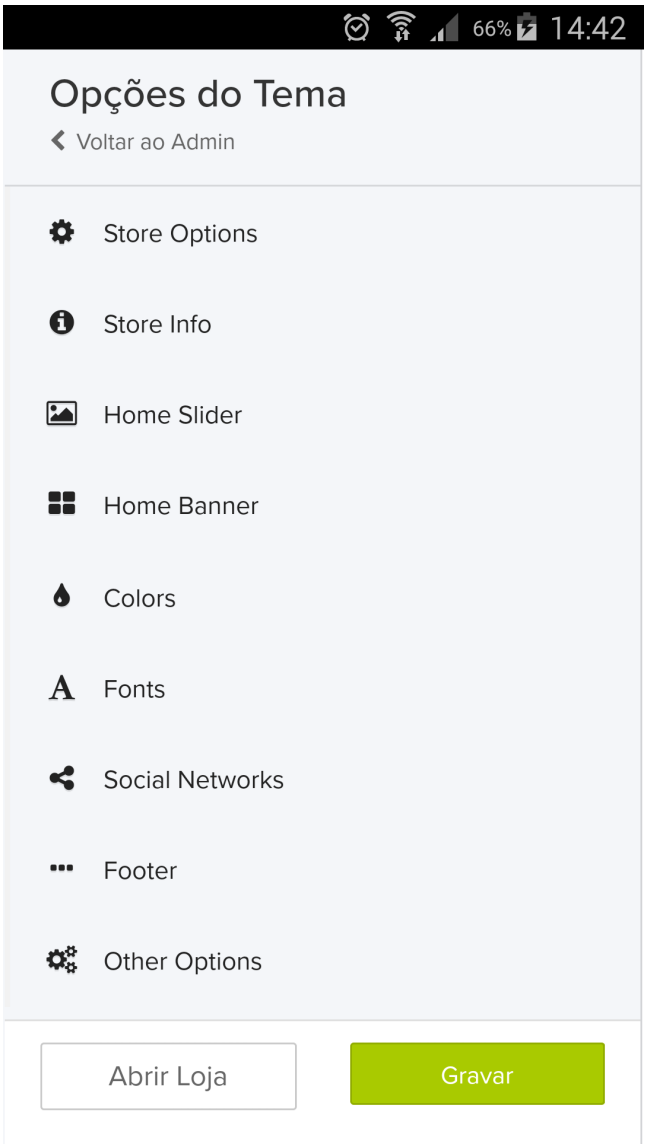


FIGURE A.3: The Theme Options App - Mobile version.

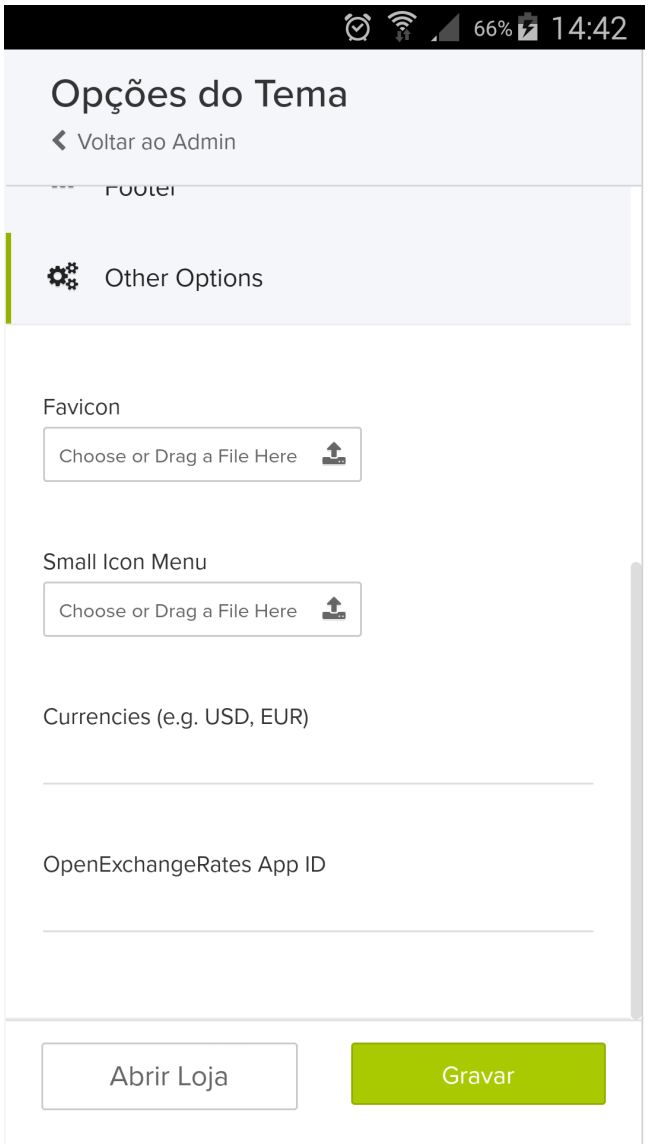


FIGURE A.4: All the functionalities of the App work on mobile, as was part of the requirements.

Appendix B

Categories Screenshots

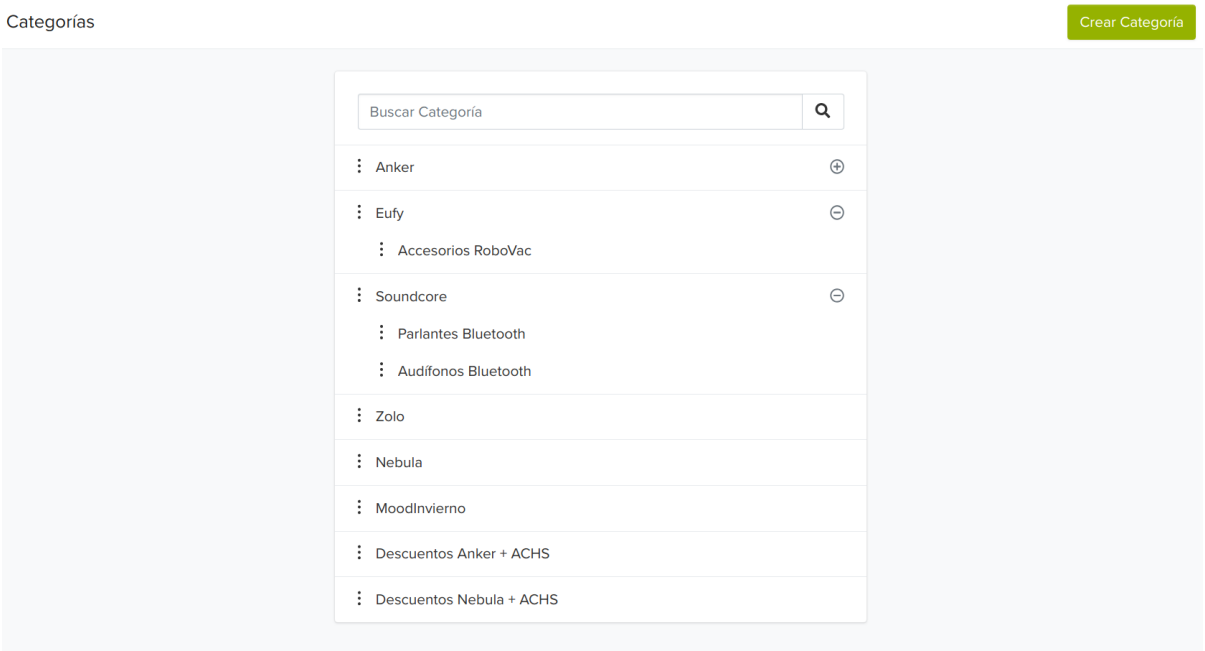


FIGURE B.1: The Desktop version of the product categories list.

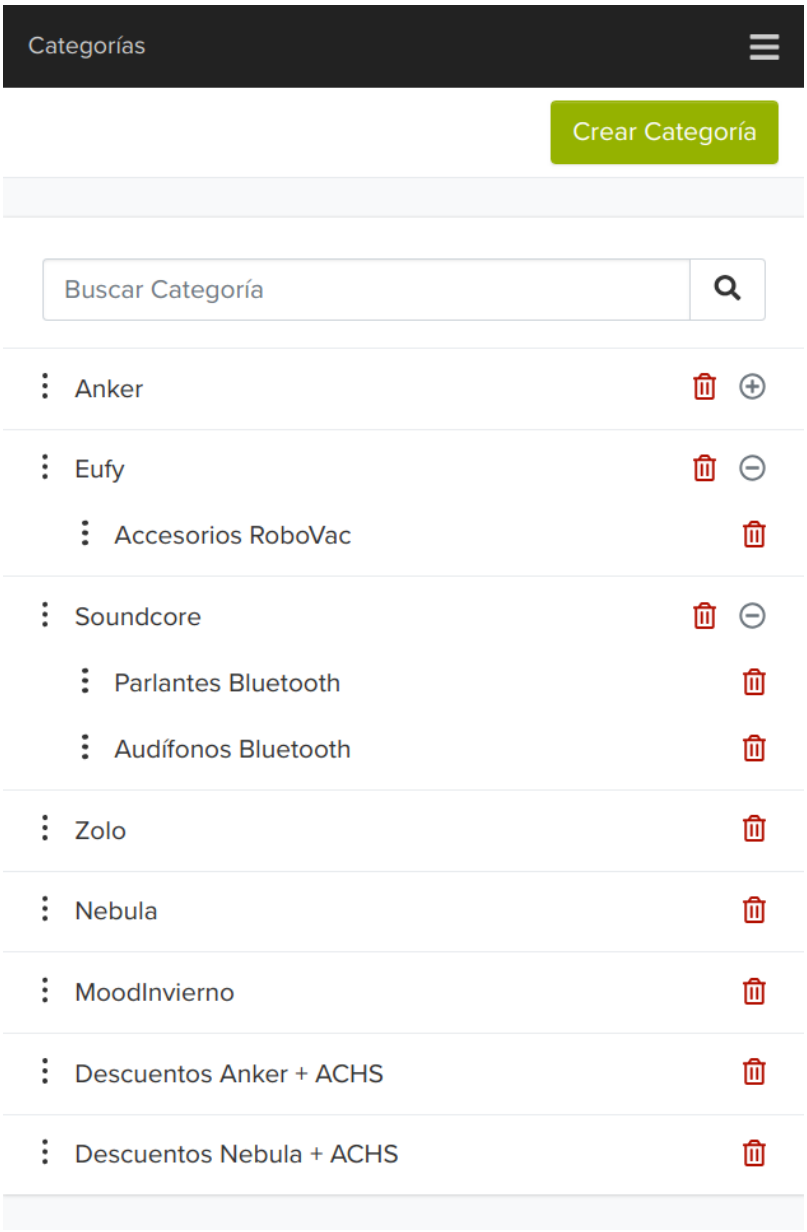


FIGURE B.2: The Mobile version of the product categories list.

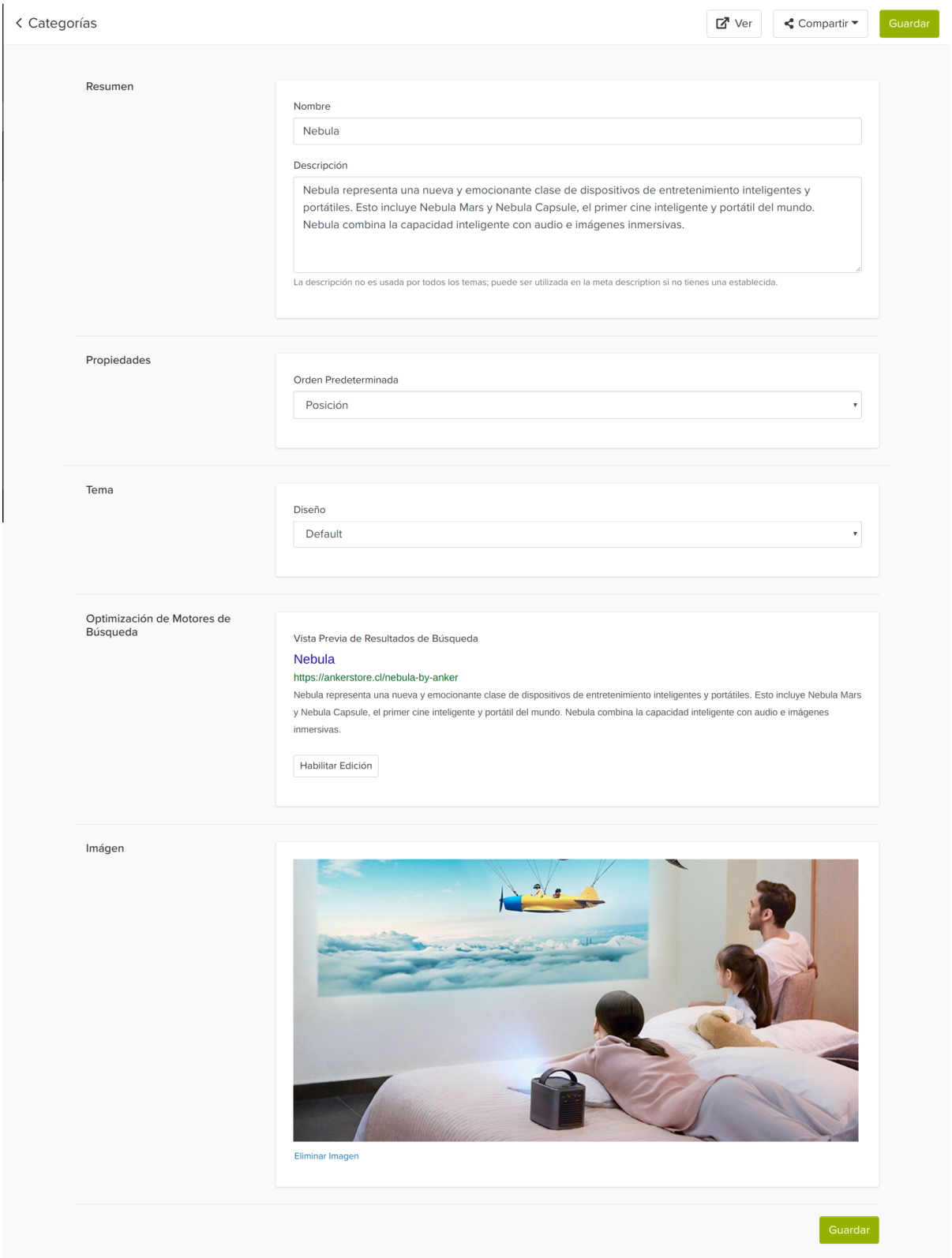


FIGURE B.3: The Desktop version of the category edition page.

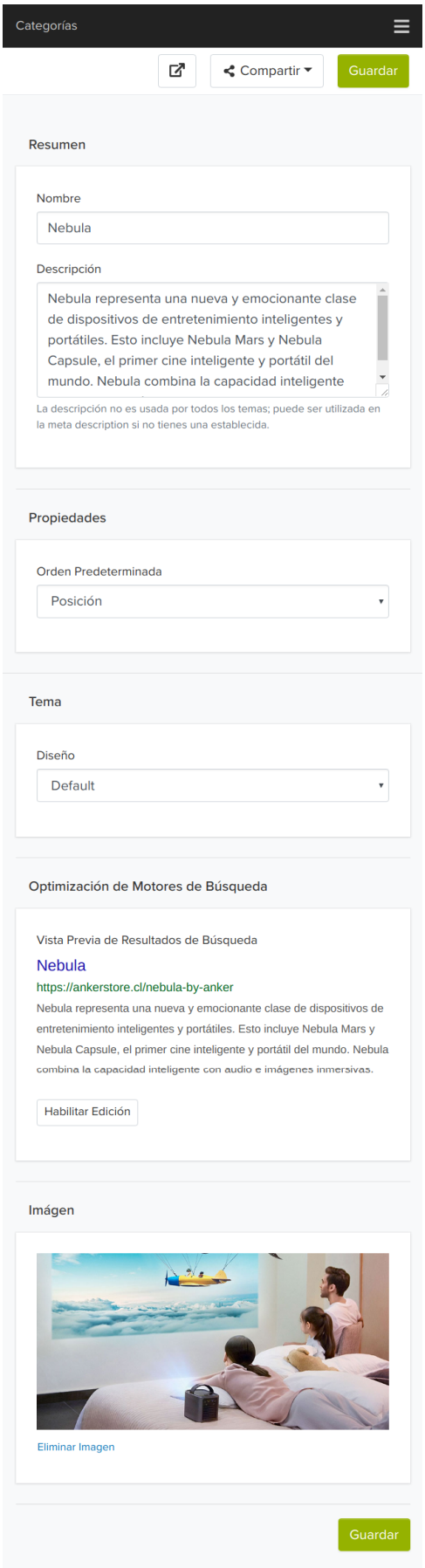


FIGURE B.4: The Mobile version of the category edition page.

Appendix C

Product List Screenshots

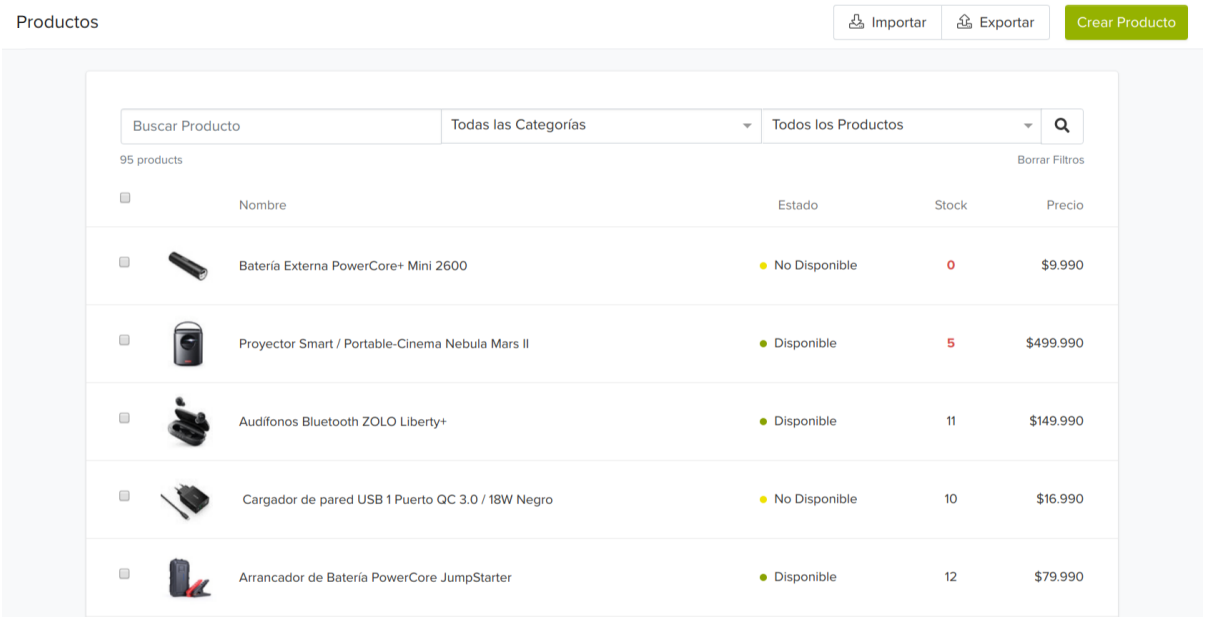


FIGURE C.1: The Desktop version of the product categories list.

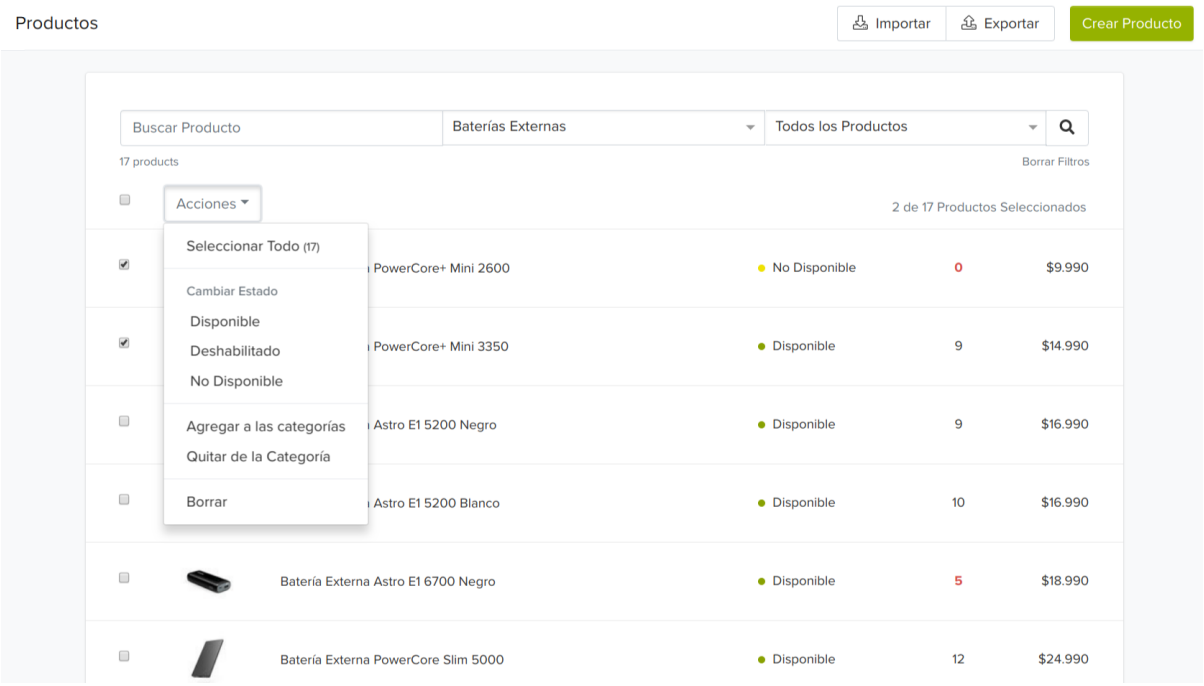


FIGURE C.2: Performing actions on the Desktop version of the product categories list.

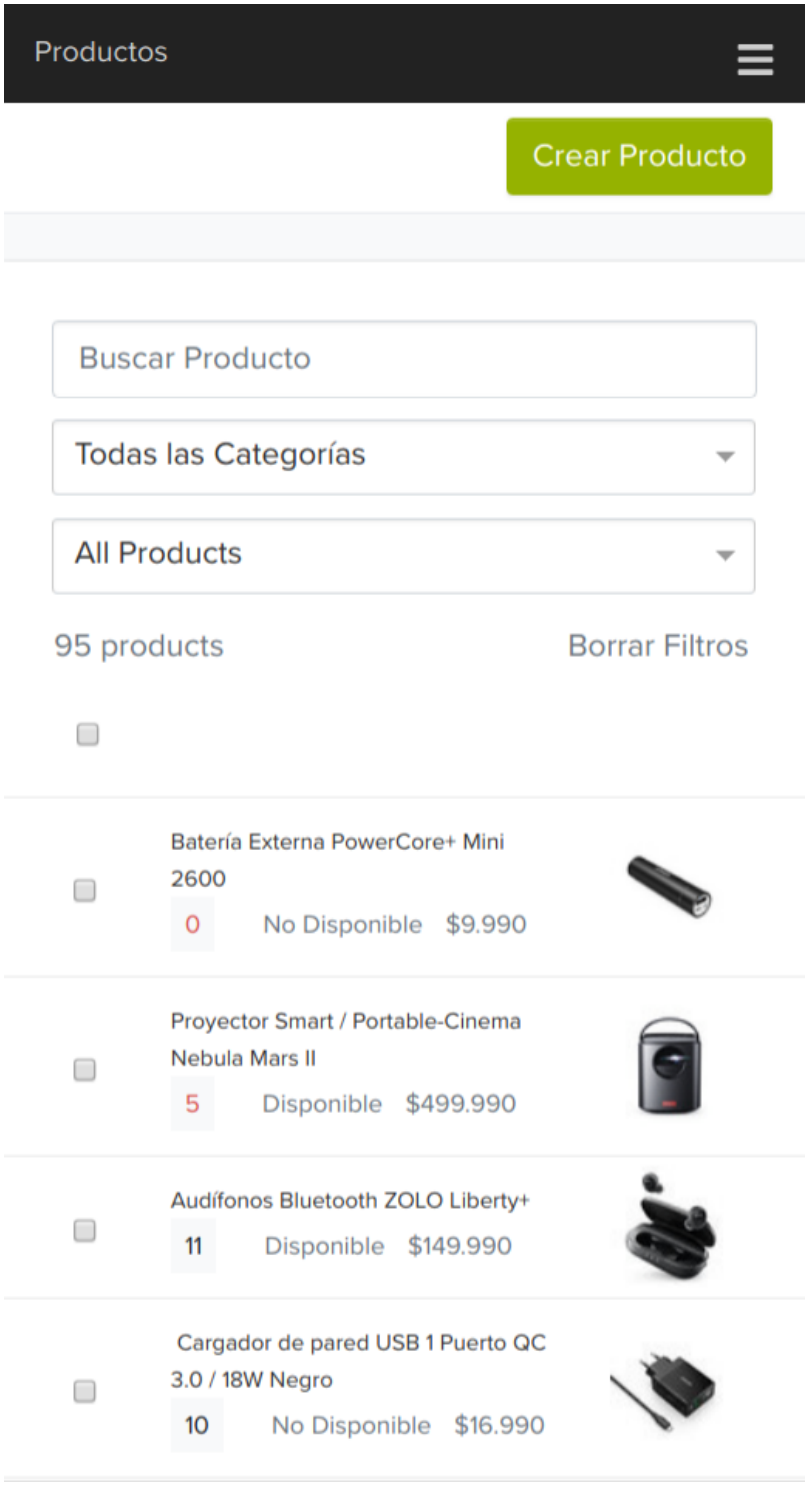


FIGURE C.3: The Mobile version of the product categories list.

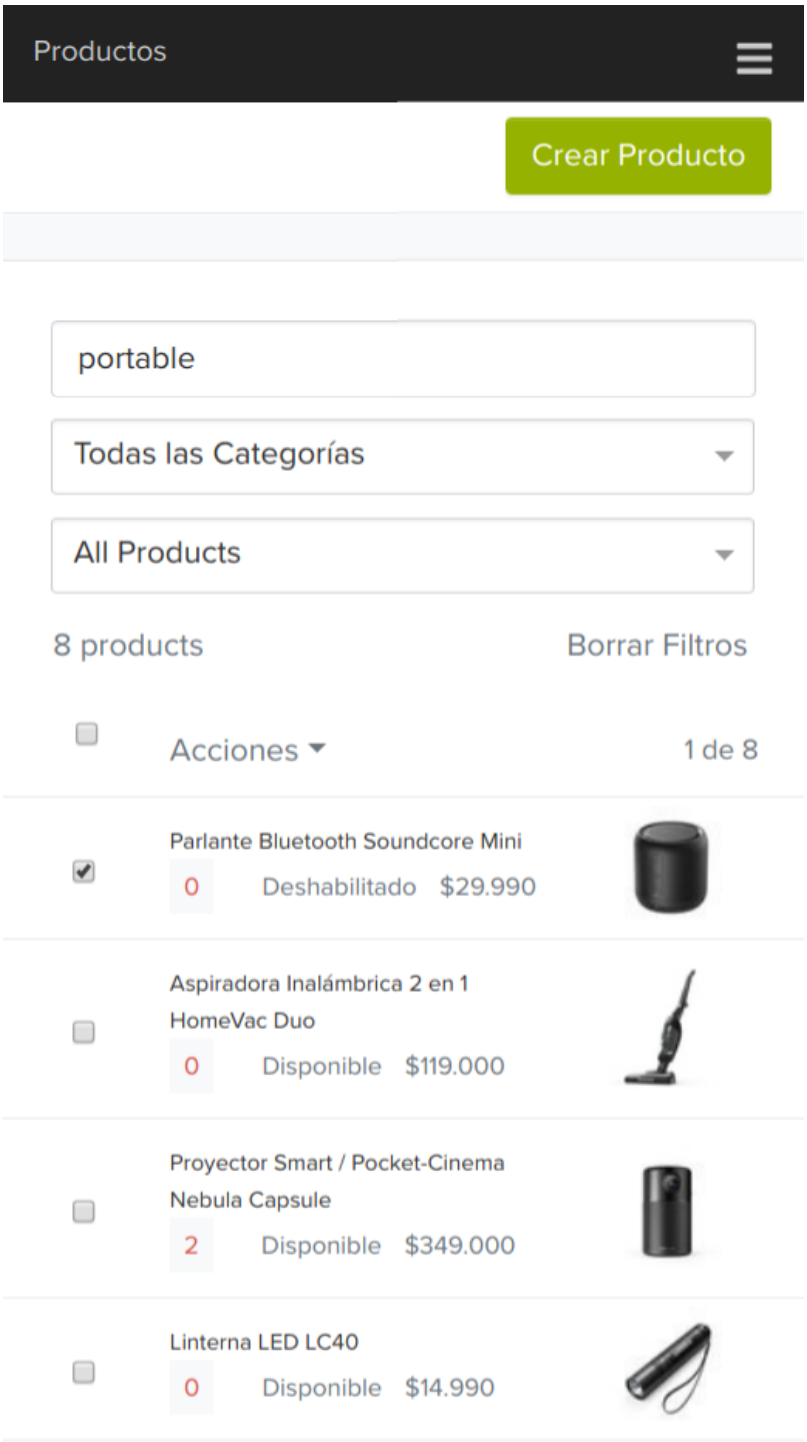


FIGURE C.4: Performing actions on the Mobile version of the product categories list.

Appendix D

Product Edition Screenshots

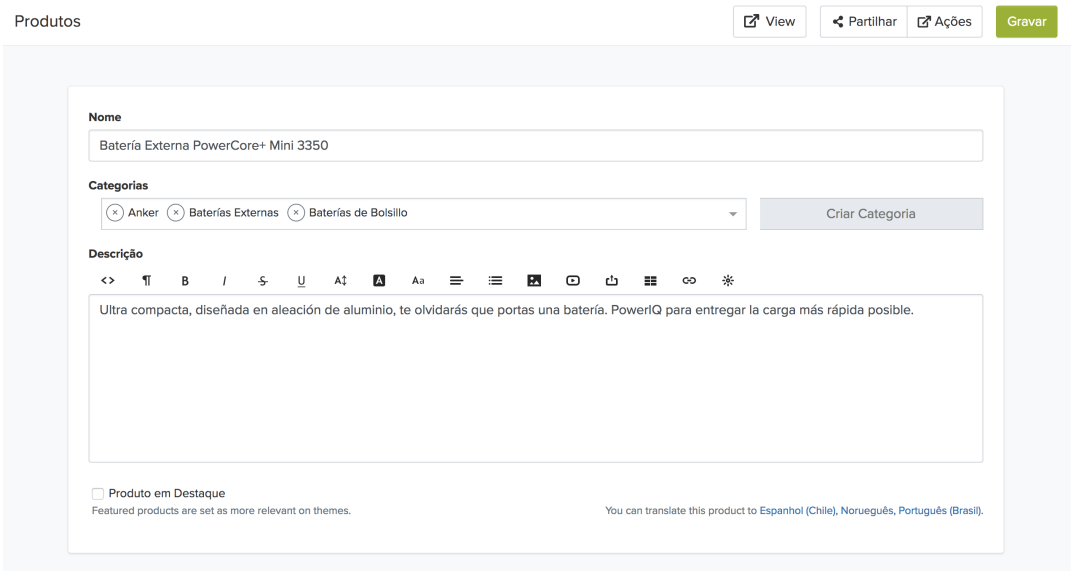


FIGURE D.1: The basic properties in the new Product Edition page, on Desktop.

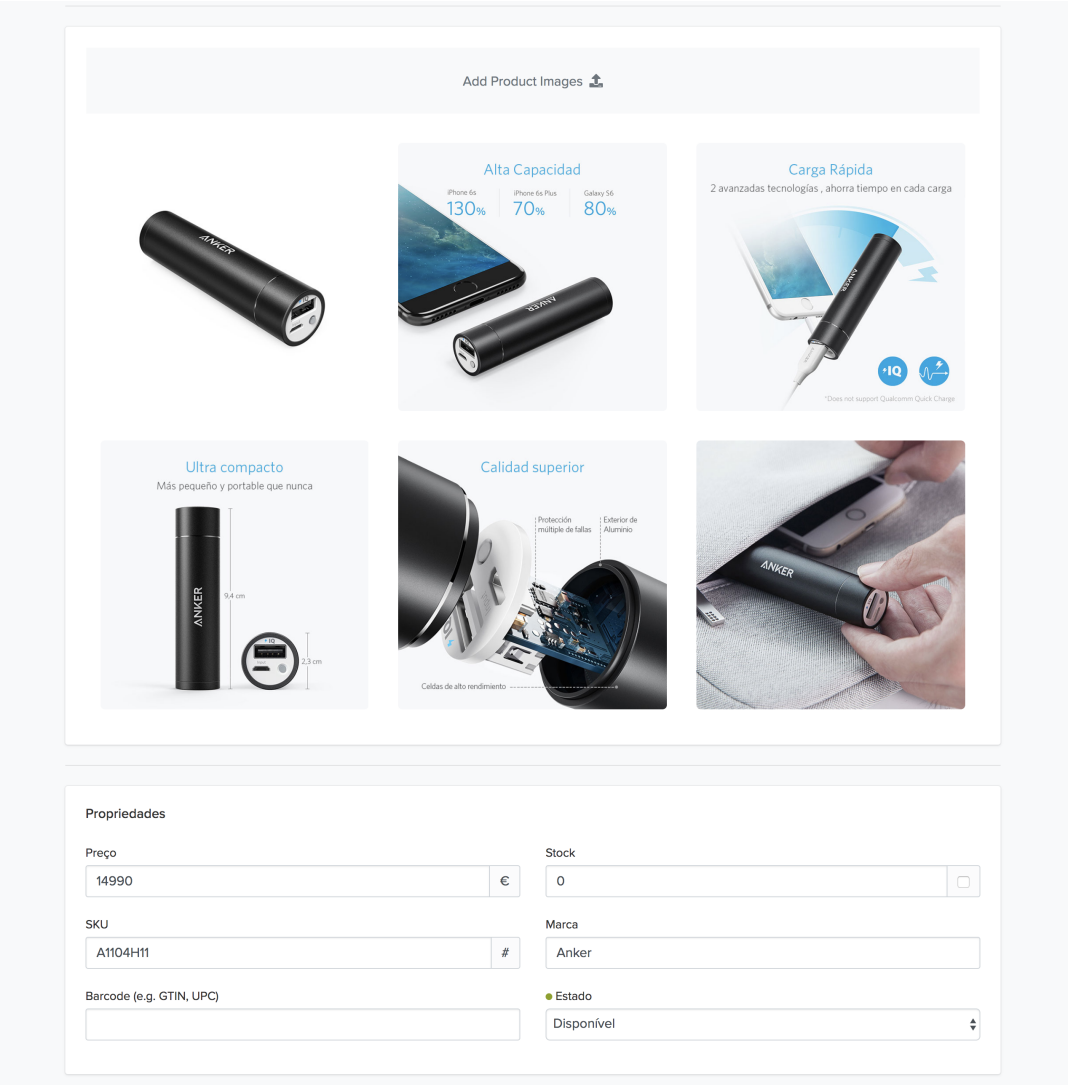


FIGURE D.2: The image gallery and product properties in the new Product Edition page, on Desktop.

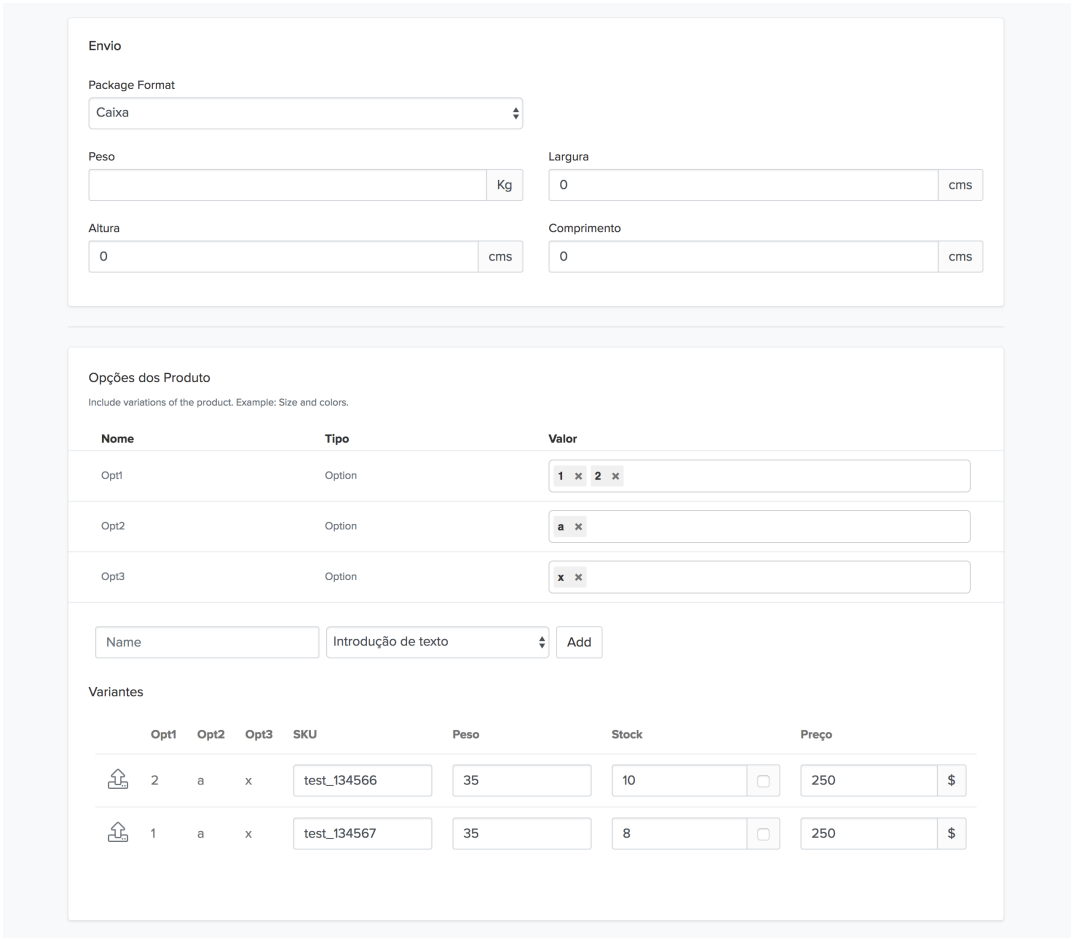


FIGURE D.3: The shipping properties and the product options and variables table in the new Product Edition page, on Desktop.

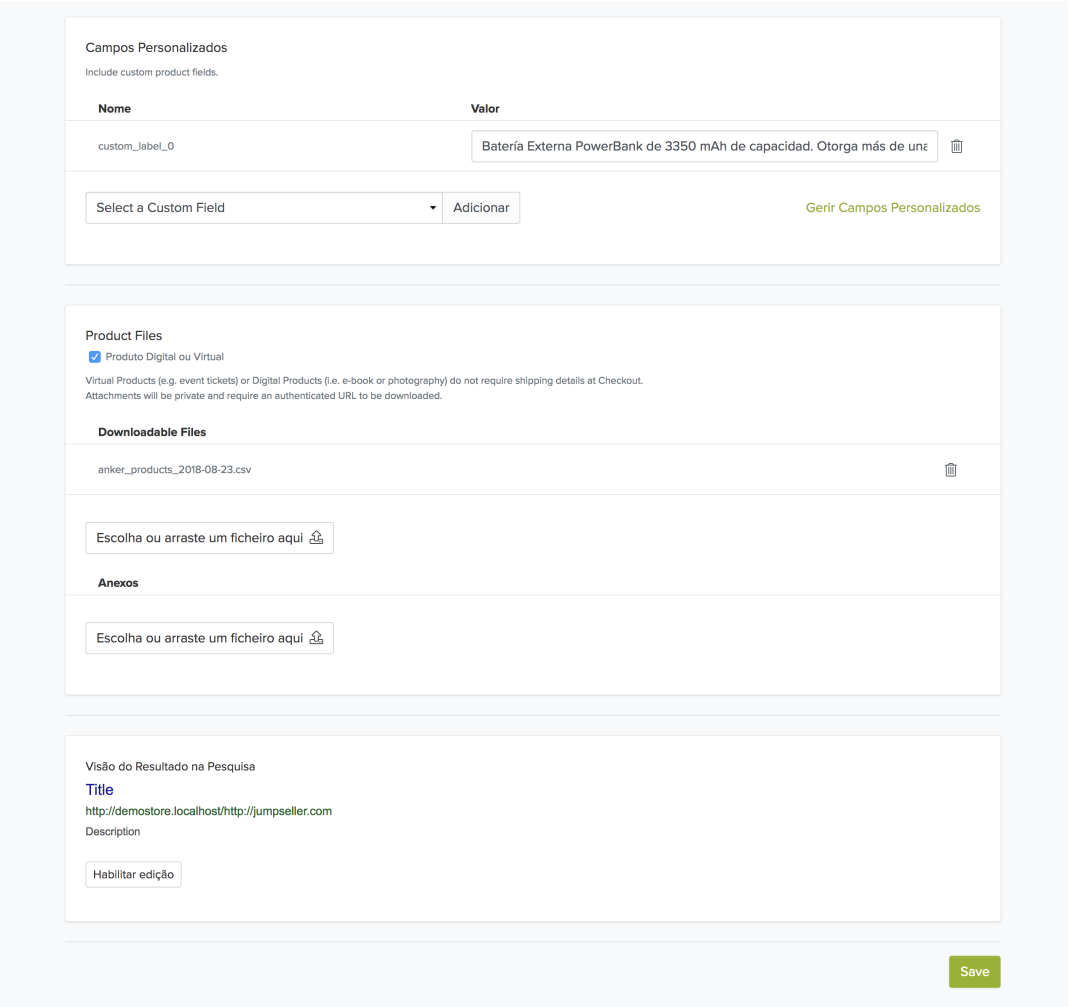


FIGURE D.4: The Custom Fields, Product Files and SEO in the new Product Edition page, on Desktop.

Figures D.5 through D.8 show the mock-up images created by the design team at Jumpseller, as the final product wasn't finished at the end of the dissertation's work period.

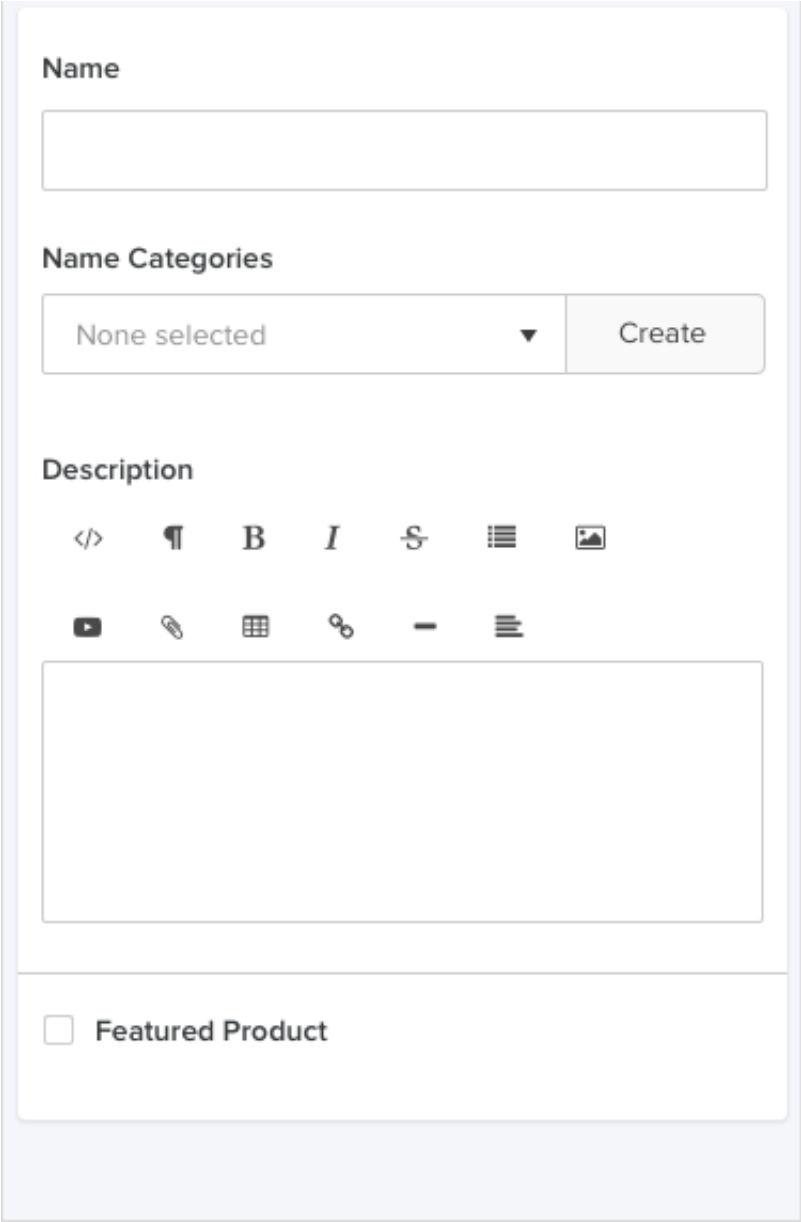


FIGURE D.5: Mock-up image of the Product Properties on mobile.

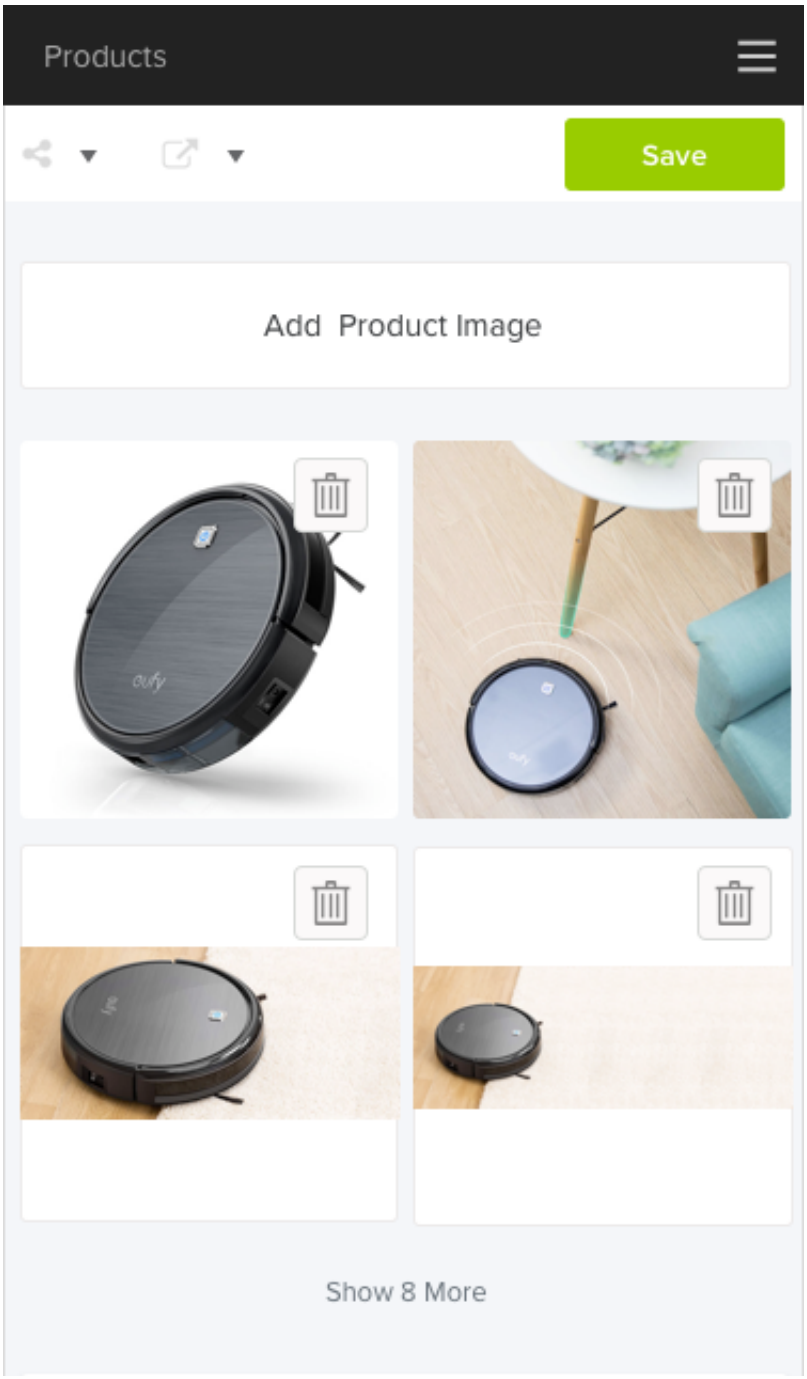


FIGURE D.6: Mock-up image of the Product Images Gallery on mobile.

FIGURE D.7: Mock-up image of the Product Options on mobile.

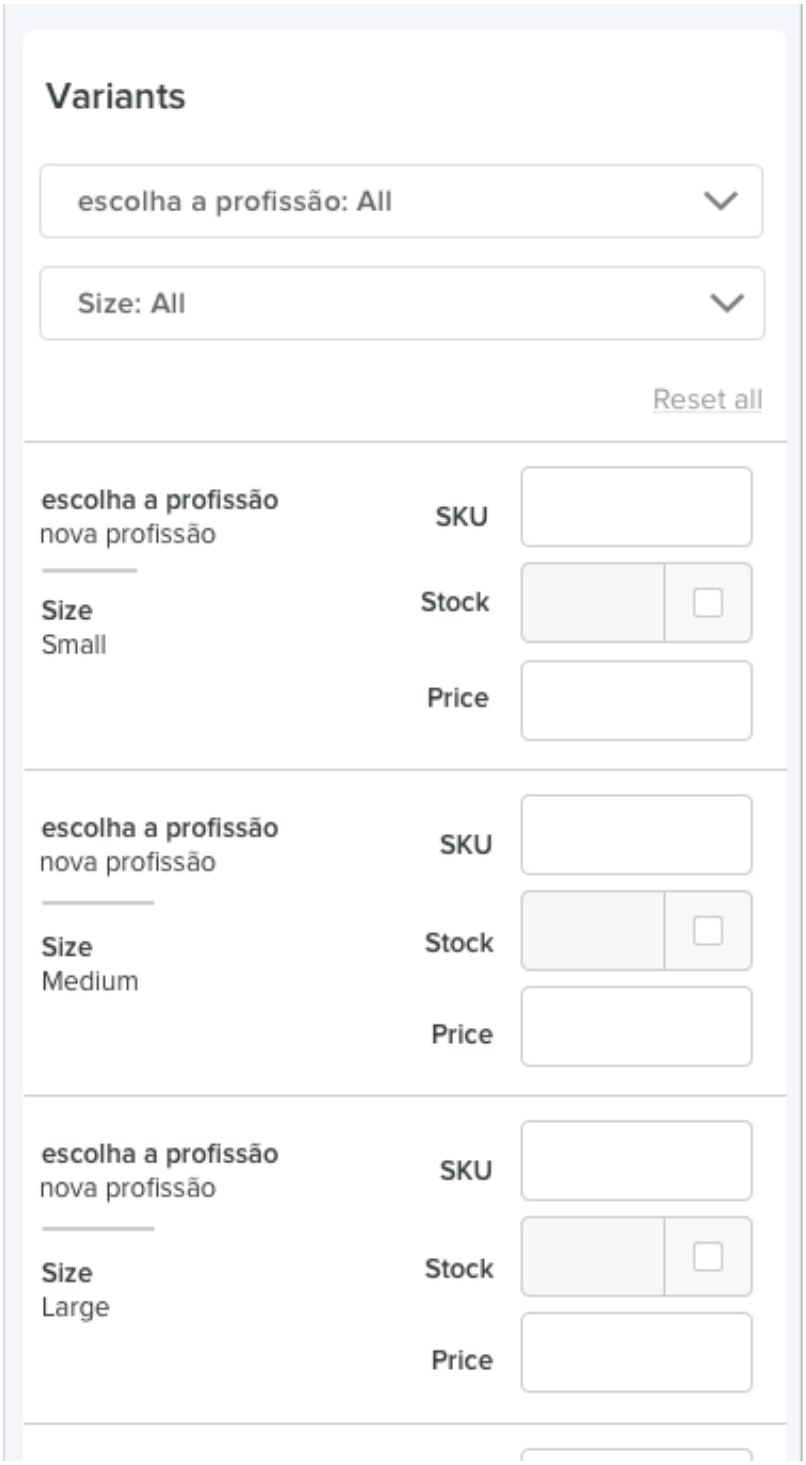


FIGURE D.8: Mock-up image of the Product Variants on mobile.

Bibliography

Bryant, Jay and Mike Jones (2012). "Responsive Web Design". In: *Pro HTML5 Performance*. Berkeley, CA: Apress, pp. 37–49. ISBN: 978-1-4302-4525-4. DOI: 10.1007/978-1-4302-4525-4_4. URL: https://doi.org/10.1007/978-1-4302-4525-4_4.

Coyier, Chris (2018). "A Complete Guide to Flexbox". In: URL: https://css-tricks.com/snippets/css/a-guide-to-flexbox/.

Crockford, Douglas (2008). *JavaScript: The Good Parts*. O'Reilly Media. URL: http://shop.oreilly.com/product/9780596517748.do.

Flanagan, D. (2006). *JavaScript: The Definitive Guide*. Definitive Guide Series. O'Reilly Media, Incorporated. ISBN: 9780596101992. URL: https://books.google.pt/books?id=k0CbAgAAQBAJ.

Flavián, Carlos, Miguel Guinalíu, and Raquel Gurrea (2006). "The role played by perceived usability, satisfaction and consumer trust on website loyalty". In: *Information & Management* 43.1, pp. 1 –14. ISSN: 0378-7206. DOI: https://doi.org/10.1016/j.im.2005.01.002. URL: http://www.sciencedirect.com/science/article/pii/S0378720605000169.

Fowler, Martin and Matthew Foemmel (2006). "Continuous integration". In: *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf* 122, p. 14.

Gackenheimer, Cory (2015). "What Is React?" In: *Introduction to React*. Berkeley, CA: Apress, pp. 1–20. ISBN: 978-1-4842-1245-5. DOI: 10.1007/978-1-4842-1245-5_1. URL: https://doi.org/10.1007/978-1-4842-1245-5_1.

Ihrig, Colin J and Adam Bretz (2014). *Full stack Javascript development with MEAN*. SitePoint.

Linley, C. (2017). *Front-End Developer Handbook 2017*. Frontend Masters. URL: https://www.gitbook.com/book/frontendmasters/front-end-handbook-2017/details.

Loeliger, J. and M. McCullough (2012). *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. O'Reilly Media. ISBN: 9781449345051. URL: https://books.google.pt/books?id=aM7-Oxo3qdQC.

Meyer, E.A. (2006). *CSS: The Definitive Guide: The Definitive Guide*. O'Reilly Media. ISBN: 9781449397258. URL: https://books.google.pt/books?id=rdtCRLXAL78C.

Mozilla (2017a). "CSS-MDN". In: URL: https://developer.mozilla.org/en-US/docs/Web/CSS.

— (2017b). "HTML-MDN". In: URL: https://developer.mozilla.org/en-US/docs/Web/HTML.

— (2017c). "Media Queries -MDN". In: URL: https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries.

MySQL, AB (2001). *MySQL*.

Nice, Bradley (2017). "Front-End vs Back-End vs Full Stack Development". In: URL: https://medium.com/level-up-web/front-end-vs-back-end-vs-full-stack-development-78267f545121.

R. Fielding J. Gettys, J. Mogul H. Frystyk L. Masinter P. Leach T. Berners-Lee (1999). *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616. RFC Editor. URL: http://www.rfc-editor.org/info/rfc2616.

Sampaio, Ana Isabel (2013). “Responsive Web Design”. In: URL: http://mei.di.uminho.pt/sites/default/files/dissertacoes//eeum_di_dissertacao_pg20190.pdf.

Sierra, Kathy and Bert Bates (2005). *Head First Java: A Brain-Friendly Guide*. " OŔeilly Media, Inc."

Thomas, David, Andrew Hunt, Chad Fowler, et al. (2005). *Programming Ruby: the pragmatic programmersǵuide*. Raleigh, NC: Pragmatic Bookshelf,

Tilkov, S. and S. Vinoski (2010). “Node.js: Using JavaScript to Build High-Performance Network Programs”. In: *IEEE Internet Computing* 14.6, pp. 80–83. ISSN: 1089-7801. DOI: 10.1109/MIC.2010.145.

Toxboe, Anders (2009). “User Interface Anti-Patterns”. In: URL: http://ui-patterns.com/blog/User-Interface-AntiPatterns.

Van Welie, Martijn, Gerrit C Van Der Veer, and Anton Eliëns (2001). “Patterns as tools for user interface design”. In: *Tools for Working with Guidelines*. Springer, pp. 313–324.