# Towards the integration of user interface prototyping and model-based development

Catarina Machado
*HASLab/INESC TEC & University of Minho*
Braga, Portugal
a81047@alunos.uminho.pt

José Creissac Campos
*HASLab/INESC TEC & University of Minho*
Braga, Portugal
jose.campos@di.uminho.pt

***Abstract*—The main objective of this paper is to make a contribution in the automation of web applications' development, starting from prototypes of their graphical user interfaces.**

**Due to the exponential increase in the use of internet-based services and applications, there is an also increasing demand for Web designers and developers. At the same time, the proliferation of languages, frameworks and libraries illustrates the current state of immaturity of web development technologies. This state of affairs creates difficulties in the development and maintenance of Web applications.**

**In this paper, we argue that integrating concepts of model-based user interface development with the more traditional user-centred design approach to development can provide an answer to this situation. An approach is presented that allows designers to use prototyping tools, in this case Adobe XD, to design graphical interfaces, and then automatically converts them to (Vue.js + Bootstrap) code, thus creating a first version of the implementation for further development. This is done through the interpretation of the SVG file that Adobe XD exports.**

***Index Terms*—Web Development, Prototyping tools, Web frameworks, Code generation.**

## I. Introduction

Web Development has become one of the main areas of Software Development, covering the development of websites, web services and web applications [1]. In the early 90's, most web pages were static HTML documents [2]. Websites have since become complex web applications that perform transactions, present real-time data, and provide interactive experiences to users [3]. Due to this complexity, development support has evolved through the availability of frameworks that automate recurring implementation patterns, thus reducing the need for *boilerplate* code [4]. More recently, low code (or even, no code) development platforms aim to further automate the programming of user interfaces [5]. Automation, however, is usually achieved at the cost of reduced flexibility in the development process.

The first web development frameworks were developed in the late 1990's, and since then over five thousand have been released [1]. Although all of them target (multi-tiered) web development, they vary in terms of the architectural layers that are considered (from server-side to client-side, or full stack vs. support for specific layers of the architecture) and the programming languages used (from more or less general purpose programming languages such as Java, JavaScript or Ruby, to domain-specific languages such as HTML or CSS). This proliferation of languages, frameworks and libraries demonstrates the current state of immaturity of web development technologies, which creates difficulties in the development and maintenance of applications [6].

Model-based development aims to support software development by increasing the level of abstraction of the development process [7]. Models can be executed for simulation or testing purposes at any stage of the development process, which means that the behaviour of the system can be evaluated from the requirements phase to the production phase, with no need to change the system's description [8]. Additionally, the models can be used to (at least partially) automate the generation of the source code. However, the adoption of model-based approaches in user interface development has been slow [9]. Prototyping-based approaches are favoured due to the flexibility they afford during design and development. Raising the level of abstraction of the development process, however, seems a good solution to the current technological immaturity of web development.

This paper thus arises with the general objective of studying how user interface development can take advantage of model-based approaches, with a particular focus on web applications user interfaces, and on the role that user interface prototypes might play. In summary, the main contribution is in the area of web development by proposing an approach to automate code generation from user interface prototypes.

The remainder of the paper is structured as follows. Section II, presents background information, discussing model-based user interface development and user-centred design, prototyping tools and Web development frameworks. Section III then presents the approach proposed to go from user interface prototypes to source code. To this end, the structure of a prototype and a meta-model of a web framework are presented and the mapping between them discussed. In Section IV a short example is used to illustrate how this approach is applied. In the last two Sections, the discussion and conclusions of the work are put forward.

This work is financed by National Funds through the Portuguese funding agency, FCT - *Fundação para a Ciência e a Tecnologia*, within project UIDB/50014/2020.

978-1-6654-8343-8/21/$31.00 ©2021 IEEE

## II. Background

### A. Model-Based User Interface Development vs. User-Centred Design

Model-Based User Interface Development (MBUID) [10] aims to support the development of quality interactive computing systems, while lowering costs through automation of the development process. MBUID is particularly suited for efficient development of multi-target applications.

As captured by the Cameleon Reference Framework [11], MBUID approaches are based on the development of models of both requirements and envisaged solutions, and their gradual refinement until a running system is reached. Abstract models of the user interface are progressively transformed, by incorporating details about interaction modalities (first) and the technological platforms (second), to generate concrete user interface models (first) and final user interface models (second), so that a running system can be achieved. Automating this refinement process means that development can progress efficiently once details of interaction modalities and target platforms are known. Note, however, that full automation of the process is typically considered unfeasible, with partial automation being favoured [9].

MBUID can be seen as the User Interface (UI) counterpart to Model Driven Engineering (MDE) [7]. However, while MDE has seen considerable success, take up of MBUID in industry has been slow. According to Meixener et al. [9], this can be addressed by MBUID approaches that better integrate with User-Centred Design (UCD) approaches [12], [13].

This paper addresses this need to better integrate MBUID and UCD. Unlike approaches that strive to integrate models into UCD [14], [15], thus changing the nature of the UCD process, we take the view that the notion of what a model is must be revisited. Indeed, as far back as [16], models were seen as part of the problem, in the sense that they are costly to develop. They are also not how an UCD practitioner is used to think about the development process.

Where MBUID might be seen as a somewhat deterministic process from abstract model to final code, UCD is about iterativel refining and evaluating design solutions and ideas. This iterative nature of the process is supported by prototypes. These prototypes, consisting of mock-ups of the user interface, plus some representation of dialogue control, capture the intended design. Hence, instead of introducing MBUID modelling notations into the UCD process, we want to use UCD prototypes as the basis for the MBUID process.

Considering that prototypes already express a notion of the user interface modalities to be used, we posit that prototypes can be considered an expression of a concrete user interface model in MBUID terms. Thus, if we are able to (partially) automate the process of generating a final user interface from a user interface mock-up, we will be able to get the best of both approaches. We take advantage of UCD flexibility to explore and evaluate design ideas using user interface prototypes, until a good design is achieved. We then take advantage of MBUID automation to generate the corresponding UI at lower cost.

Note that the goal is not to produce the final version of the UI. Instead, we aim to produce a first version of the code, which can then be refined by the developer. This enables us to place less requirements on the prototype, regarding the amount of information that it must contain.

### B. Prototyping tools

Although paper prototyping is proposed as the lower cost, most flexible, approach to UI prototyping, UI mock-up tools have become popular, particularly for higher-fidelity prototyping. They are specially relevant in our case since they can generate machine readable representations of the mock-ups. Figma, Sketch and Adobe XD stand out as some of the most popular tools [17]–[19].

*1) **Figma**:* Figma is characterized by being browser-based and features a complete interface development suite, with integrated solutions ranging from wireframing to graphic design itself, interaction prototyping, and UX presentation through hyperlinks. One of the biggest advantages of this tool is the automated version history support. It is very powerful when it comes to collaborative work between teams, whether they are designers or developers [20]. It is possible to integrate user testing into the design workflow by creating tests right on the artboard.

Figma supports exporting prototypes to JPG, PNG, SVG, and PDF. Figma can also export to Hyper Text Markup Language (HTML) and Cascading Style Sheets (CSS) code through plugins. It supports both low fidelity and high fidelity prototyping [21]. Because it is browser-based, it can be used in all major operating systems.

*2) **Sketch**:* Sketch is another tool that features a complete design suite, supporting the creation and prototyping of every kind of graphical interface. The biggest advantage is that it is a software that has been used for many years by many professionals, thus ensuring that the tool has been improved year after year. It is a simple tool to use with powerful functionality for testing and collecting customer feedback. It also has good features for teams to work collaboratively [20].

Sketch features a document's full version history and version management, and supports low fidelity and high fidelity prototyping. This tool can export prototypes to both bitmap (PNG, JPG, TIFF and WebP) and vector (SVG, PDF, EPS) images, and it is possible to export to HTML and CSS through plugins [22].

One limitation of Sketch is the fact that the tool is only available for macOS.

*3) **Adobe XD**:* Adobe XD is another tool to create mock-ups, whether for websites, wire-frames, prototypes or art-boards. Adobe XD offers support for building high-fidelity prototypes. However, it also allows designers to create low-fidelity versions for the early stages of the design process. It has the option of collaborative work, both for teams and also for presentation to clients [20].

Adobe XD has document history for cloud documents, which enables the review of previous saved versions of documents, and labelling and preserving those versions. It supports

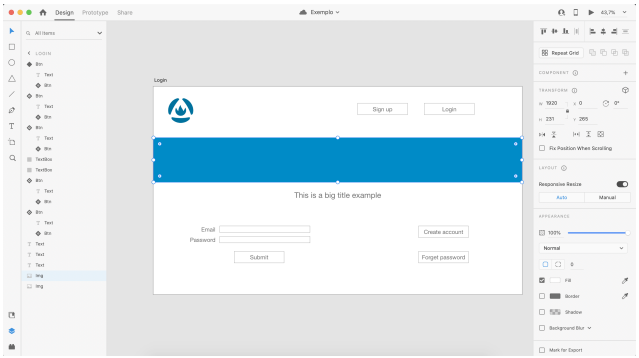

Fig. 1. Adobe XD working environment.

TABLE I
TOOLS COMPARISON

| | **Figma** | **Sketch** | **Adobe XD** |
|---|---|---|---|
| User testing | yes | yes | yes |
| Collaborative development | yes | yes | yes |
| Version management | yes | yes | yes |
| Platforms | web | macOS | Windows/macOS |
| Level of fidelity | low/high | low/high | low/high |
| Export formats | R V W | R V W | R V W |

Export formats: **R**aster graphics; **V**ector graphics: **W**eb (HTML+CSS)

exporting the prototypes to JPG, PNG, SVG and PDF [23]. The tool also allows users to export prototypes to HTML and CSS code through plugins and it's possible to share prototypes with customers.

This tool belongs to a well-known and well-established company, Adobe, and is available on both Windows and macOS systems [24].

*4) Discussion:* Table I presents a summary of the analysis above. As can be seen, the tools have similar characteristics. All aspects considered, Adobe XD was chosen as the prototyping tool to adopt. Being natively supported by both the Windows and macOS operating systems means that the approach will have a broader reach.

One aspect, in particular, deserves some discussion. All tools have the functionality to convert UI mock-ups to HTML and CSS code. However, this feature has several shortcomings, since the generated code is geared towards creating a visualization for users, not towards the implementation of the UI. This means the code is not appropriately structured from an implementation perspective, making it difficult to use and even more to maintain. In summary, from an UI development perspective these tools have poor UI code generation support.

Alternatively, we note that exporting to SVG is also common to all of these prototyping tools. SVG (Scalable Vector Graphics) is an open, XML-based, format to define vector-based graphics for the Web. Converting a mock-up to SVG ensures great reliability and robustness, and because it is created in XML, it gives us access to the XML code of the design. With this XML file, the goal is to interpret the components and their respective positions in order to automatically generate the code for the web page.

However, exporting to SVG is also not a perfect solution as it was not designed for prototypes representation. While it gives us an open format to work with, the trade-off is that information is lost in the export. SVG is a graphics format and (some) information about the structure of the prototype is lost during the export process. A simple example is that a button is no longer a button, but a drawing of a rectangle.

### C. Web development frameworks

Since the dawn of the digital age, more than five thousand web development frameworks have been developed [1]. All these frameworks have different characteristics and are recognized by different particularities.

Frameworks can be classified into two different categories: client-side and server-side. Client-side frameworks are responsible for implementing and enhancing user interfaces with animated features and responsive layouts that adapt to different display sizes. By doing so, they enable a style of user interaction similar to that of native applications. Additionally, because web applications can be accessed through so many different devices, UI responsiveness is an important factor to take into consideration when designing the user experience of a web application. This is a feature that allows the web application UI to automatically adapt to any type of device, from a cell phone with a small screen to a computer with a large monitor [25]. These frameworks are an asset in this sense, since they incorporate libraries and guidelines that facilitate this work.

Server-side frameworks also have well-defined rules and architectures and support creating different types of pages. These frameworks can also provide security factors for web pages. In this way, client-side frameworks allow a user experience that is closer to that of a native application, while server-side frameworks are more tied to the web navigation model.

Several studies try to rank web frameworks, based on different factors such as the size of the frameworks' communities, their popularity, or even the amount of successful usage (cf. [26]). Herein we focus on online rankings, as they are better able to keep up with the constant evolution of web development frameworks.

The HotFrameworks ranking[1] is based on two different popularity metrics: the GitHub score and the Stack Overflow score. The former is based on the number of stars the repository has on GitHub. These stars represent the number of users who have favorited that particular repository. The latter is based on the number of questions asked on Stack Overflow that contain the framework's tag. At the time of writing, the most popular framework, according to this ranking is React, followed closely by ASP.NET MVC and a group of five frameworks in third place (Angular, Ruby on Rails, Angular.js, Vue.js and Django). If we focus on client-side frameworks for programming the interfaces in the browser we are left with: React, Angular, Angular.js and Vue.js.

Since 2011, Stack Overflow has been doing a yearly study where they ask programmers about their preferred

[1]https://hotframeworks.com. Retrieved January 25, 2021.

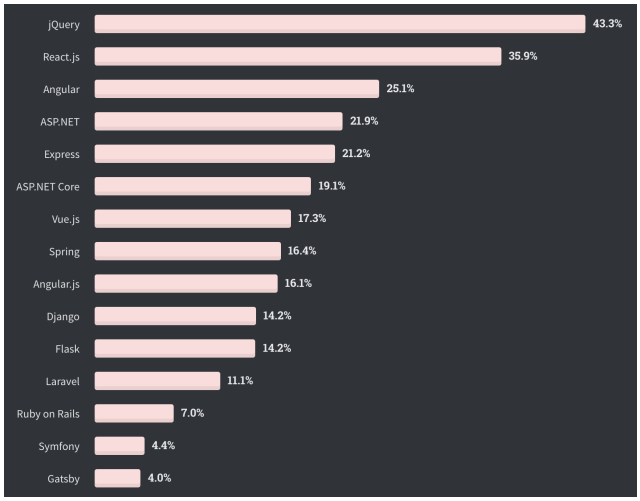


Fig. 2. Ranking of frameworks according to Stack Overflow. Retrieved from [28].

technologies, their programming habits and their work preferences [27]. In the latest study, from 2020, about 65,000 programmers were surveyed and one of the questions was to select the web frameworks they use in their day-to-day work. The fifteen most used web frameworks in 2020 can be seen at Figure 2. If we only consider client-side we are left with React, Angular, Vue.js and Angular.js.

Web frameworks are widely used today, however they are better or worse depending on the project to be developed. A web application is typically divided into three layers: user interface, business logic and data layers. In this specific case, our goal is the generation of source code for native-like user interfaces. We are thus interested on client-side frameworks for user interface development.

Thus, we can single out four web development frameworks based on the before mentioned rankings. They are: React, Angular, Vue.js and Angular.js. Angular.js can be discarded since it has been superseded by Angular. This leaves us with three web frameworks to consider.

*1)* ***Angular****:* Angular is a web framework based on TypeScript, which follows the Model-View-Viewmodel (MVVM) pattern [29], and is based on a hierarchy of components. Components are the main building blocks and can display information, render models, and perform actions on data. Angular has several types of data bindings within a component, such as property binding, event binding and two-way binding [30].

*2)* ***Vue.js****:* Vue.js is a JavaScript web framework, focused on the development of graphical interfaces. As with Angular, the architecture of Vue.js is usually described as MVVM. Vue.js stands out for its high degree of scalability and is characterized by being component-based and using directives in templates, such as for data binding and event handling [30].

*3)* ***React****:* React is a JavaScript web framework that aims to develop graphical interfaces. React is characterized by being component-oriented and composition-oriented. The overall goal is to transform the current state of the application into a view that can be presented to the user (via the DOM). Components can be written using two different approaches: components as functions (most recommended) and components as ES6 classes [30].

*4) Discussion:* From these brief descriptions we can see that the three frameworks have similar characteristics. All are very suitable for UI layer development and either one is a good choice. Vue.js, however, is known for its easy learning curve. Thus, it was decided to choose Vue.js as the framework to use. To complement the use of Vue.js and to facilitate the layout process and responsiveness we can use CSS libraries. In this case, we chose to use Bootstrap[2].

## III. From Prototypes to Implementations

We can look at frameworks as having a meta-model. This allows us to break down the framework into different pieces and schematize how a web page is developed through the framework. To design user interfaces, we resort to prototypes. Analyzing these prototypes, we can realize that they can be mapped to the meta-model of the web framework. This mapping is possible since, both in how the web framework is used and in how the graphical interfaces are designed, we have components. Components are small pieces that when added together give rise to the web pages we are used to browsing. These can be buttons, images, text forms, tables, and so on.

### A. Structure of a prototype

Figure 3 presents an example of a high fidelity mock-up developed with Adobe XD, where we can already identify some web page characteristics. As we can see, on a web page we can identify different components and navigation. It is common for a page to have a navbar, which usually has a logo (image) and buttons. The body of the page can have other images, forms, text and buttons or hyperlinks, which can redirect the user to other pages, such as the create account or password recovery pages. Other elements can be considered such as a footer.

Figure 4 shows the general structure of a prototype. A prototype consists of several artboards, i.e., several prototype pages. Each of these pages contains components, which can be simple components such as an image, a text box, a button or text, or they can be a set of components which we call being inside a container, such as a navbar, a footer, a form or a table. Each of these components has properties such as size and color, and an identifier (ID).

Since when exporting to SVG we loose information, we will use the component's ID to establish a correspondence between the SVG elements (e.g. a rectangle) and the prototype's components (e.g. a button or simply text). Containers are used to be able to realize whether a set of components, for example a set of text and text boxes, are a form and not simply these randomly elements. In this way, you can directly infer the Bootstrap component itself.

[2]https://getbootstrap.com/. Retrieved July 15, 2021.

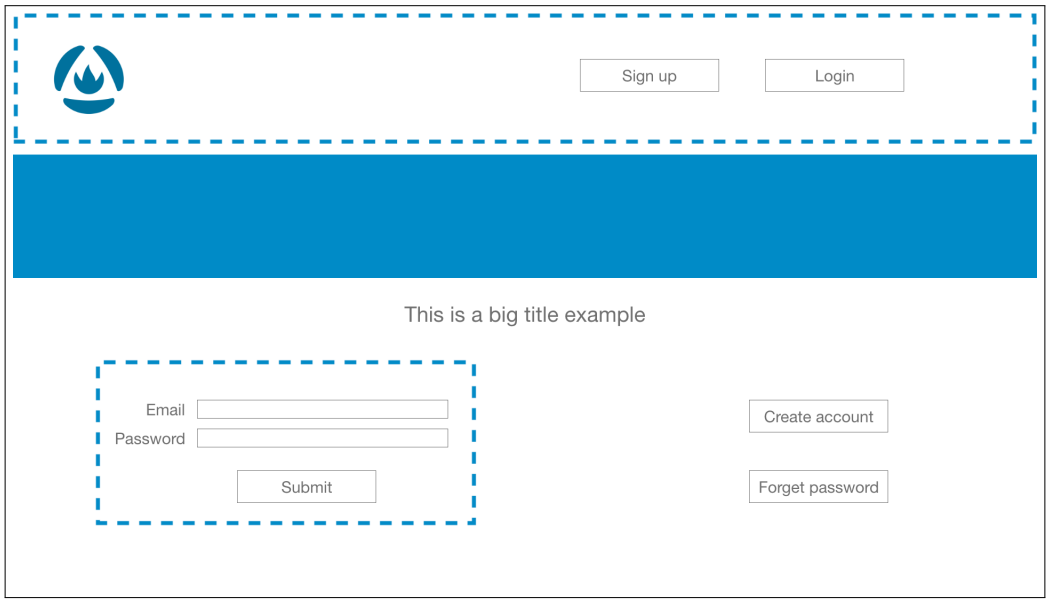


Fig. 3. Prototype example in Adobe XD. The blue interrupted rectangle in Adobe XD is not part of the design. It is used to outline the container.

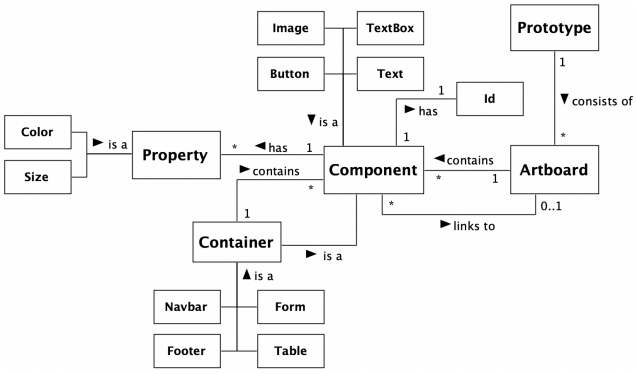


Fig. 4. Structure of a prototype.

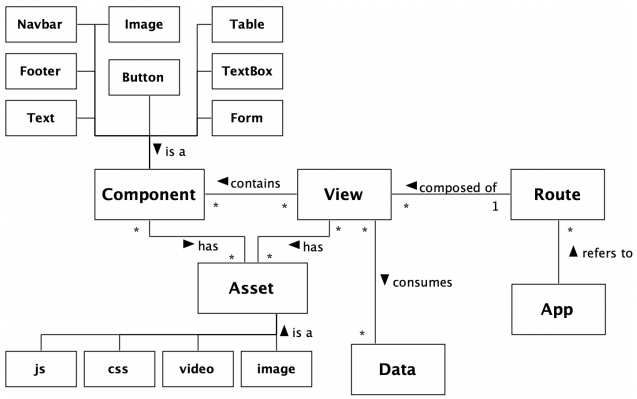


Fig. 5. Vue.js component architecture meta-model.

A prototype can have many more components, such as all the components provided by Bootstrap, but at this stage we decided to focus on just a few to simplify the diagrams and the explanation.

### B. The framework meta-model

One first goal is to define an architecture to implement the UI. In the UI layer of a web application there are architectures at several levels, namely a components architecture, which corresponds to the program structure that implements the interface that the user sees (components), and the application's source architecture, which corresponds to the folder structure/-source code. Thus, when developing a new project in Vue.js it is important to use a proper folder structure, component architecture and naming convention (coding style) [31].

So the next step is to schematize, through a meta-model, how an application in Vue.js is structured. Based on the Vue.js documentation [32], and on the analysis of well-regarded Vue.js projects[3], such as the VUEMMERCE[4] and COPILOT[5] projects, the meta-model presented in Figure 5 was defined.

As can be seen in the figure, when creating a project in Vue.js, one can start by creating **components** (which can be headers, footers, menus, modals, search bars, lists, charts, cards, forms, among other items) that are later aggregated into **views**, the graphical interface pages that will be made available to users. Thus, a **view** is composed of one or more **components** and may also have other presentation code. In addition, it also consumes data, for example, through APIs, in order to present the final result to the user.

[3]https://www.bacancytechnology.com/blog/top-21-amazing-vuejs-projects. Retrieved March 25, 2021.

[4]https://github.com/ivanlori/Vuemmerce. Retrieved April 4, 2021.

[5]https://github.com/misterGF/CoPilot. Retrieved May 7, 2021.

As a way to enrich the application, there are assets, which can be images, videos, CSS or JavaScript code, among other elements, used by the **components** and the **views**.

The application's **routing** corresponds to the workflow of the pages and respective URLs, i.e., it is through the **route** that the URL that will allow viewing a respective **view** is specified.

### C. Mapping between prototypes and the meta-model

In order to map a prototype into a Vue.js implementation, we have to map the structure of the prototype (see Figure 4) into the Vue.js meta-model (Figure 5). Once a designer has designed a prototype of a GUI, such prototype will have to be exported via Adobe XD to SVG. The SVG format can be read as an XML file. The information in the SVG is then used to create the skeleton of the project (Vue.js + Bootstrap).

To solve the loss of information problem, created when mock-ups are exported to SVG, some assumptions must be made about the input. The components used in the prototype (which will have to be mapped to the implementation) must be identifiable by their ID. Hence, the designer must use a set of predefined values in the components' ID field, each identifying a specific type of component.

The list of the components present in the Figure 4 and Figure 5 and the annotation (ID) to use for each of them in the prototype is the following:

- Text - "Text";
- Image - "Img";
- Button - "Btn";
- Navbar - "Navbar";
- Footer - "Footer";
- Form - "Form";
- Text Box - "TextBox";
- Table - "Table".

Using these ID, it is then possible to determine which component is to be used in Vue.js, to represent each particular element of the prototype. Otherwise, it would be more complicated to understand if a rectangle is after all a button or simply a text box. For simple components such as buttons, the mapping is direct; for more complex components (which aggregate several components into one), we can deduce which one to use through the ID of the container that aggregates them in the prototype. It is important to note that to consider more components, we simply give the designers the component and its annotation (ID) and add this mapping between the ID and the code to be generated to our algorithm.

Another big challenge is not to use absolute positions in the location of the UI components. To do this, the Bootstrap grid system can be used. Through the absolute positions described in the XML file it is possible to deduce in how many columns (12 maximum) and sections the page is divided into and, after that, in which section/column the component is located.

In summary, the algorithm that the approach follows is explained in two parts: positioning and components. Starting with positioning, the steps are as follows:

1) Break the prototype into sections (lines);
2) Break the prototype into columns;
3) Make the skeleton of sections and columns in Vue.js.

After having the skeleton complete, it remains to place the different components in their respective positions. So, with regard to the components, the algorithm is as follows:

1) Match the component ID and the component code (already previously established);
2) Make the necessary changes to the size and color of the component.

Thus far, we looked at the Components part. Usually, a web application has more than one page and navigation between them. Using Adobe XD, multiple pages and their navigation can be prototyped. However, when exported to SVG this information is lost and we only get to know which pages exist, but not the information on navigation between them. To solve this problem, it will be necessary to express the navigation in a separate model. SCXML [33] seems a viable solution, considering that Statecharts have already used to express user interface navigation [34].

## IV. An example

The goal of this section is to illustrate how the steps presented in the previous section can be applied to the example in Figure 3. With the proposed approach we were able to conclude that this prototype page is divided into 4 sections (rows), due to the components' positions. The first, which is the navbar (container with "Navbar" ID), is composed of three columns, the first of which represents half the width of the page (i.e., it will correspond to half of the Bootstrap grid). We use the coordinates of the components within the artboard to calculate their position on a 12-column grid, and consequently position them in Bootstrap.

The second section consists only of one image that occupies the entire row, so it has only 1 column. Along with this, the third section also has only 1 column with a title (text). Finally, the last row consists of 2 columns. The first column has a form (container with ID "Form", which has two "Text", two "Text Box" and a "Button"), and the second column has two buttons.

To design the components, we have to read the IDs (matching them with the corresponding Vue.js + Bootstrap code) and analyze the respective tags to see if their color or size have been set.

As an example, let's take the login form present on the page. On the SVG file, we have a container with "ID=Form". We use the coordinates of the components to determine which components are inside the "Form" container. Each of these components also has an ID, identifying the component. Note that to avoid having repeated IDs (when we have multiple components of the same type) indices are added to the repeated IDs, and the original ID is preserved in the "data-name" tag. The relevant SVG elements are:

```
<g id="Form"
  transform="translate(178 665)">
  <rect width="704" height="297"/>
</g>

<text id="Text"
```

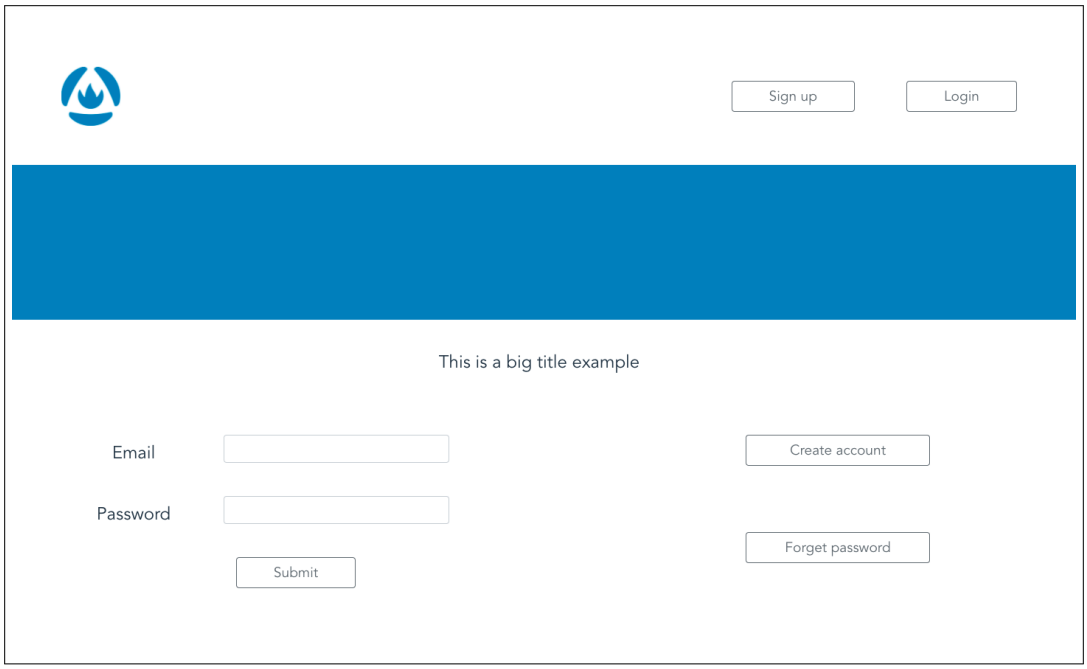


Fig. 6. Prototype example in Vue.js.

```
  transform="translate(257 807)">
  Password</text>
<text id="Text-2" data-name="Text"
   transform="translate(286 753)">
   Email</text>

<g id="TextBox"
  transform="translate(345 778)">
</g>
<g id="TextBox-2" data-name="TextBox"
  transform="translate(345 724)">
</g>

<g id="Btn" transform="translate(420 856)">
  <g id="Rectangle"></g>
  <text id="Text"
      transform="translate(83 42)">
      Submit</text>
</g>
```

To map these components to Vue.js, we already know that we will have to use the "Form" component. This way, the "Text" will correspond to a "Label" in Vue.js; the "Textbox" will correspond to an "Input" and, finally, the "Btn" to a "Button". Then, on Vue.js + Bootstrap we have the following code.

```
<form>
  <label>Email</label>
  <input type="email">

  <label>Password</label>
  <input type="password">

  <button>Submit</button>
</form>
```

In this way, from the user interface mock-up presented in Figure 3 we obtain the user interface programmed in Vue.js + Bootstrap presented in Figure 6.

## V. Discussion

As previously mentioned, several UI mockup and prototyping tools include the functionality to convert UI mock-ups to HTML and CSS code. In many cases this is accomplished by embedding a raster or vector graphics image of the design in the HTML code. While we will typically get a result that is almost identical to the one designed. this approach has several shortcomings, and it is particularly poor from a UI code generation perspective.

With the proposed approach, it is not possible to obtain such an identical result, but the result obtained is still quite satisfactory, particularly taking into account other trade-offs, especially the evolvability of the code itself. The biggest advantage is the flexibility of the generated code. The degree of responsiveness is clearly superior, while embedded images are rigid.

The Fireblade[6] plugin for Adobe XD has similar goals to our proposal. However we do not require a plugin to be installed, replying instead on conventions on how the IDs are used. This means that our approach is more easily portable to other tools.

The proposed approach, allows us to save time while giving designers flexibility, as obtain code for a widely used web framework, that is easier to maintain and can be use as the starting point for the final implementation.

[6]https://docs.fireblade.io/. Retrieved October 11, 2021.

## VI. Conclusion

Throughout the article, it was possible to address the topics of model-based user interface development, user-centred design, prototyping tools and web development frameworks. A contribution has been made to the area of web development by proposing an approach to automate code generation from user interface mock-ups.

Going into detail on the Vue.js web framework, it was possible to develop a component architecture meta-model of the framework, which allowed us to schematize the pieces to develop a web page in this language. Was also made a prototype structure in order to schematize the components and navigation within a prototype. Using a prototyping tool and taking advantage of the option to export the mock-up to SVG (XML), it was possible to define an approach that maps the prototype to Vue.js code.

In this mapping, the main challenges are the positioning of the components and the design of the components themselves. These challenges were met by dividing the mock-up into rows, and then into columns. For the components design, as they are previously identified by an ID, it is possible to match the component code with the designed component, with only the possibility of customizing the color and size.

In this way it is possible to get more out of the designers' work, which saves a lot of time. The generated code is simple to maintain and evolve, flexible and responsive.

As future work, it remains to implement the algorithm in the Python language.

## References


[1] Z. Mehrab, R. B. Yousuf, I. A. Tahmid, and R. Shahriyar, "Mining developer questions about major web frameworks," in *Proc. of the 14th International Conference on Web Information Systems and Technologies (WEBIST 2018)*, INSTICC. SciTePress, 2018, pp. 191–198.

[2] I. P. Vuksanovic and B. Sudarevic, "Use of web application frameworks in the development of small applications," in *Proc. 34th International Convention MIPRO*, 2011, pp. 458–462.

[3] J. Plekhanova, "Evaluating web development frameworks: Django, ruby on rails and cakephp," Fox School of Business, Temple University, The IBIT Report, 2009.

[4] S. Liawatimena, H. L. H. Spits Warnars, A. Trisetyarso, E. Abdurahman, B. Soewito, A. Wibowo, F. Gaol, and B. Abbas, "Django web framework software metrics measurement using Radon and Pylint," in *Indonesian Association for Pattern Recognition International Conference (INAPR)*, September 2018, pp. 218–222.

[5] R. Marvin, "The best low-code development platforms," Aug 2018, (Retrieved January 25, 2021). [Online]. Available: https://www.pcmag.com/roundup/353252/the-best-low-code-development-platforms

[6] J. I. Fernández-Villamor, L. Díaz-Casillas, and C. A. Iglesias, "A comparison model for agile web frameworks," in *Proc. Euro American Conf. on Telematics and Information Systems (EATIS '08)*. ACM, 2008.

[7] S. Kent, "Model Driven Engineering," in *Integrated Formal Methods. IFM 2002*, ser. Lecture Notes in Computer Science, vol. 2335. Springer, 2002, pp. 286–298.

[8] A. Bergmann, "Benefits and drawbacks of model-based design," *KMUTNB International Journal of Applied Science and Technology*, vol. 7, pp. 15–19, September 2014.

[9] G. Meixner, F. Paternò, and J. Vanderdonckt, "Past, present, and future of model-based user interface development," *i-com*, vol. 10, no. 3, pp. 2–11, 2011.

[10] P. Pinheiro da Silva, "User interface declarative models and development environments: A survey," in *Interactive Systems Design, Specification, and Verification*. Springer, 2001, pp. 207–226.

[11] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, "A unifying reference framework for multi-target user interfaces," *Interacting with Computers*, vol. 15, no. 3, pp. 289–308, 2003.

[12] A. Monk, "User-centred design," in *Home Informatics and Telematics: Information, Technology and Society*. Springer, 2000, pp. 181–190.

[13] ISO, "ISO 9241-210:2019 Ergonomics of human-system interaction – part 210: Human-centred design for interactive systems," International Organization for Standardization, 2019.

[14] F. Paternò, C. Santoro, and L. D. Spano, "Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments," *ACM Transactions on Computer-Human Interaction*, vol. 16, no. 4, pp. 19:1–19:30, November 2009.

[15] Q. Limbourg and J. Vanderdonckt, "Multipath transformational development of user interfaces with graph transformations," in *Human-Centered Software Engineering: Software Engineering Models, Patterns and Architectures for HCI*. Springer, 2009, pp. 107–138.

[16] A. R. Puerta and P. Szkeley, "Model-based interface development," in *Conference Companion on Human Factors in Computing Systems (CHI '94)*. ACM, 1994, pp. 389–390.

[17] K. Baker, "15 best wire-frame tools for your website design [2021 guide]," 2021, (Retrieved May 23, 2021). [Online]. Available: https://blog.hubspot.com/website/wireframe-tools/

[18] M. Myre, "The 8 best wireframe tools in 2021," 2021, (Retrieved May 23, 2021). [Online]. Available: https://zapier.com/blog/best-wireframe-tools/

[19] T. May and C. Cahill, "53 web design tools to help you work smarter in 2021," 2021, (Retrieved May 23, 2021). [Online]. Available: https://www.creativebloq.com/features/best-web-design-tools/

[20] G. Pimenta, "Tire suas dúvidas: Adobe XD, Figma ou Sketch, qual ferramenta de design escolher e por quê?" 2020, (Retrieved May 23, 2021). [Online]. Available: https://comunidade.rockcontent.com/adobe-xd-figma-ou-sketch/

[21] Figma, "Figma documentation," (Retrieved July 20, 2021). [Online]. Available: https://help.figma.com/

[22] Sketch, "Sketch documentation," (Retrieved July 20, 2021). [Online]. Available: https://www.sketch.com/docs/

[23] Adobe, "Adobe XD documentation," (Retrieved July 20, 2021). [Online]. Available: https://helpx.adobe.com/xd/user-guide.html

[24] A. Ivanovs, "Figma vs Sketch vs Adobe XD: Whichis the better design tool?" 2020, (Retrieved May 23, 2021). [Online]. Available: https://www.codeinwp.com/blog/figma-vs-sketch-vs-adobe-xd/

[25] J.-P. Voutilainen, J. Salonen, and T. Mikkonen, "On the design of a responsive user interface for a multi-device web service," in *2nd ACM International Conference on Mobile Software Engineering and Systems*, 2015, pp. 60–63.

[26] M. Salas Zarate, G. Alor-Hernández, R. Valencia-García, L. Rodríguez, A. González, and J. Cuadrado, "Analyzing best practices on web development frameworks: The lift approach," *Science of Computer Programming*, vol. 102, May 2015.

[27] Stack Overflow, "Developer survey results 2017," 2017, (Retrieved January 12, 2021). [Online]. Available: https://insights.stackoverflow.com/survey/2017

[28] ——, "2020 developer survey," 2020, (Retrieved January 12, 2021). [Online]. Available: https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages

[29] J. Smith, "Patterns - WPF apps with the Model-View-ViewModel design pattern," *MSDN Magazine*, vol. 24, no. 2, February 2009.

[30] E. Wohlgethan, "Supporting web development decisions by comparing three major JavaScript frameworks: Angular, React and Vue.js," Bachelor Thesis, Hochschule für angewandte Wissenschaften Hamburg, 2018.

[31] S. Adittane, "How to structure a Vue.js project," 2018, (Retrieved April 30, 2021). [Online]. Available: https://itnext.io/how-to-structure-a-vue-js-project-29e4ddc1aeeb

[32] Vue.js, "Vue.js documentation," (Retrieved April 5, 2021). [Online]. Available: https://vuejs.org/v2/guide/index.html/

[33] W3C, "State Chart XML (SCXML): State Machine Notation for Control Abstraction," W3C, W3C Recommendation 1, September 2015, (Retrieved October 11, 2021). [Online]. Available: http://www.w3.org/TR/scxml/

[34] I. Horrocks, *Constructing the User Interface with Statecharts*. Addison-Wesley, 1999.