IxSketch: A Design Tool for Sketching Interactive Prototypes

Bachelor Thesis

Josep López
<josel@student.ethz.ch>

Prof. Dr. Moira C. Norrie
Dr. Michael Nebeling
Christoph Zimmerli

Global Information Systems Group
Institute of Information Systems
Department of Computer Science
ETH Zurich

13th February 2015

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

globis

Abstract

Currently, there are many ways of designing and developing a user interface prototype. We can take a conventional approach, using physical tools such as pencil and paper, or digital tools that allow us to create and use prototypes that almost resemble real applications. Whichever we choose, there is a wide range of options available for users to prototype their interfaces.

Both approaches, however, come with advantages and disadvantages. In this bachelor thesis, we will explore ways in which the entire creation process of interactive prototypes can be supported going beyond what is currently possible with tools available. We will experiment with different approaches to interaction with the prototype as well as tools to empower several aspects missing on the current tools.

As part of this thesis, IxSketch will be developed, a proof of concept tool for prototyping interfaces that aims to bring the strengths of paper prototyping into the digital realm. We aim to do so by providing ways of sketching interactions as well as the traditional interface layout and components. Under the umbrella of our global key concept: 'Everything is contained in the sketch' several key features appear. Different modes of interaction like annotations, visually linking interfaces, the possibility of visualizing and interacting with our prototype in different definition levels, as well as customizing the interfaces to get closer to the final application we are trying to develop, all of these contribute to that goal.

Contents

1 Introduction

1.1 Motivation

IxSketch is a proof of concept prototype born to provide an answer for three main issues regarding the current sketching interactivity tools available:

1. Even though there is a wide variety of tools for sketching graphical user interfaces at different levels of fidelity, there is relatively little support for sketching interactivity in these tools.

2. There is not a relevant support for the transition from lower fidelity to higher fidelity in the form of static mockups to interactive prototypes.

3. The interaction, in the form of communication and revisioning, between roles involved in the development of a graphical user interface is not empowered nor eased by the current existing tools.

Paper prototyping [8] is a very useful and powerful tool to prototype applications. It is widely used mainly due to the ease of the process, and the speed with which a user can develop a prototype using this technique.

To illustrate these topics, if we choose paper prototyping, we have a really fast solution to have a prototype with certain amount of detail. We can even have explanation of interactions, either described as text or in the form of connections between interfaces with the use of visual elements like arrows, so it definitely appears like a good solution for prototyping.

But when the time to get feedback from the users come, the lack of interactivity support leaves us without the ability to detect user experience issues before developing the actual application. A change later on takes much more effort than rather early to be fixed.

On the other hand, if we take the example of digital prototyping with an application, we have other interesting advantages, like the possibility of seeing our prototype react to our

interactions.

But digital prototyping has other drawbacks as well. We might lose the closeness with the prototype for instance, as the creation or manipulation of the same may not be that immediate and direct as it would be with paper prototyping in which we directly sketch everything manually.

Think also of the scenario where you have several people discussing about an interface developed in a prototyping application. The UX expert might want to point to the designer the need for some components to be positioned, and the designer has another layout in mind, and even the programmer has something to say about the feasibility of that solution.

They have to send mails back and forth with revisions of the prototype, along with explanations of why they were doing it in that manner. There is no fast way of creating alternative versions and exploring the combination of them or just visually pointing some issues on the same prototype for other colleagues to review it.

As we can see, both options possess their strengths and weaknesses. IxSketch is created with the aim of merging the strengths of both fields, to offer all of the potential these options have, without having to suffer from the limitations that they carry with them as well.

1.2 Contribution

We had three goals to achieve with this thesis which were:

1. Support for sketching interactivity

2. Support for the transition from lower fidelity to higher fidelity in the form of static mockups to interactive prototypes.

3. Easing the interaction, in the form of communication and revisioning, between roles involved in the development of a graphical user interface.

The first goal, the support for sketching interactivity, was supplied by the key concept of our application: everything is contained in the sketch. We explored ways of creating and interacting with interfaces as we will see in detail in the Approach section. We wanted to narrow the gap between the user and the prototype, to make the experience of creating the interfaces more direct, more natural. That was the moment when this global key concept was born, in order to bring this closeness to the user.

With this in mind, the user was able to see everything with a glance of the sketch of his application. He could understand the several interfaces of the application he was developing as well as the connections between them. This was possible because the interactions were also part of the sketch allowing the user to have an understanding of the whole application by just looking at its sketch as it would happen in a paper prototyping scenario.

With regard to our second goal, the support for the transition from lower fidelity to higher fidelity, we have supplied a mechanism to offer the user with a simple way to toggle the definition of the interface that is being edited. That allows a device-agnostic layout designing. In addition, we offer a high fidelity version of the same palette of widgets which enables the user to visualize how his interface would look in a real environment or device.

This goal would be empowered with the usage of a server-side backend that would allow the creation of custom theming, allowing thus the styling of the device of choice and also, the potential visualization of an interface in several devices.

On behalf of the last goal, easing the interaction in the form of communication and revisioning between roles involved in the development, our prototype offers the possibility of different roles co-creating the final interface through the use of two of its features: the layer concept and the annotation mode.

On the one hand, with the use of the layers, we allow the users to have several versions of the interface, the visibility of which can be easily toggled to display combinations or focus in one revision of the same.

On the other hand, through the use of the annotation mode, we let different users collaborate live and discuss about the particularities of the interfaces, or share comments or opinions between different users later on.

1.3 Thesis overview

In the remaining document, we will explain the following topics of this thesis:

In chapter 2, we review some of the current existing tools at several levels; sketching, interactivity, etc. We analyze them and conclude by extracting the features desired for our proof of concept prototype.

In chapter 3, we explain the development and refinement of the sketch versions of the application, how the features and interface evolved towards the final version, and some decisions taken in the process regarding usefulness and feasibility of features, and also distribution of elements on the prototype interface.

In chapter 4, the actual implementation of the prototype and architecture of the same will be explained in detail. We will explain how the connection between interfaces work behind the scenes, how we achieved several features that we had in mind because of our research and exploration with the sketches, and also the technologies we used to develop them and the reason why we chose them on the first place.

Finally, we conclude this thesis in chapter 5 with a discussion, where we recapitulate the goals and assess how we achieved them. We also take a look at possible future work to extend or make use of our proof of concept prototype.

2
Background

There are already a lot of prototyping tools in the market, some of them remarkably good and accepted by most of the people in the field. There are also groups of researchers who are doing really interesting explorations of ways to empower sketching interactivity in tools they are developing. In this chapter we will have a look at several related scientific literature and a set of currently existing tools in order to see the approaches that have been taken on both sides, and try to extract what we need for our proof of concept prototype.

2.1 Scientific literature

There is previous research on the field of interface prototyping which has investigated tools that, as well as allowing to build relatively simple interactive mockups, support collaborative sketching, like GAMBIT [7]. In addition, some other tools have started exploring interface metaphors and gestures specifically for mobile user interface design [5].

Some of the tools we will describe in the following subsections are focused in exploring ways of transforming the drawing input the user gives them to generate motion and different kinds of animations which is very interesting regarding the goal of this thesis, even if they belong to another paradigm of applications, that do not fit in our prototyping tools field.

Others try to focus in a very specific field, prototypes focused in mobile interactions, targeting the very specific needs and lacks of the current existing applications in that field and supplying a very effective solution that addresses them, using clever approaches that gives them all the flexibility they needed and that we will take with us in the development of our proof of concept prototype as they also serve to supply some of our key concepts.

2.1.1 Draco

Figure 2.1: Draco motion path example creating fluxes of objects

Draco [3] is a sketch-based interface that permits users to add a rich set of animation effects to their drawings, seemingly bringing illustrations to life. While it does not fit into the category of prototyping tools, Draco contains in its core amazing interaction mechanisms, which were interesting to explore in order to come up with ideas for interacting with our proof of concept prototype that are really far away from the ones that current tools nowadays use.

The contribution of this application is a unified framework of motion controls that allows users to add motions to object collections. The user can define objects by selecting them in the sketch and then apply animations and effects through the use of kinetic textures.

The framework is mainly built around these kinetic textures, which provide continuous animation effects for the objects enabling many dynamic effects difficult or not possible with previous sketch-based tools.

They also pursue to create a tool that is intuitive to use thus removing the gap between the technology and the direct manipulation that would be natural for the user in a paper-like environment.

In particular, the definition of motion paths, or the use of tracing to define properties of animations, could be adopted in our proof of concept to define properties of transitions for example. As it will be discussed in the next chapter, this idea was embraced in an alternative version of the sketches, which allowed the definition of the transition by tracing a curve on a tactile area of our interface.

Figure 2.2: Draco motion profile example determining properties of animations

2.1.2 Kitty

Kitty [2] is another tool developed by the authors of Draco. As with the previous tool, we can sketch our drawings, with a basic set of sketching tools to later add powerful effects through the use of its interaction mechanisms.

Figure 2.3: Kitty example of associating elements to create objects

However, the difference with Draco relies on the fact that you can create many kinds of interactions between objects. Objects can be defined by associations between strokes previously defined on the sketch.

When the interaction mode is enabled now, we can interact with the objects for which we have defined translation paths or other kinds of motions and move them around the sketch by dragging them manually on the same interface.

Later, when we have this animations properly set, we can finally activate the graph mode where we can create certain triggers that activate some behaviour between objects. We will do this by graphically drawing a path between the objects. Then, contextual menus appear on both objects, to allow us to change all of the animation properties and the interactions that will trigger these changes in the current properties of the animation for both objects.

Figure 2.4: Kitty example of interaction between objects

For instance, we can set that the rotation of the first object will fasten the animation speed on the second object, that is connected to this, and so on. The vast variety of animations and properties of an object that can be customized allows the user to create really complex interactive scenes, but really easy to manipulate for a non-technical user as well, making it perfect for every user, from kids, to designers.

It is interesting to note the way in which they manage the display of properties and options when the graph mode is enabled. This would be a good approach for our proof of concept prototype's Linking Screens mode as we could efficiently display all of the possible customizable options of the transitions, gesture recognition, speed of the animation, and so on, without having to use traditional menus.

In addition, this would help us to achieve the goal of easing the use of our application, and keeping everything inside the context of the sketch.

2.1.3 i2ME

i2ME [4] is a framework for building interactive mockups with concrete mobile interaction elements. It allows to add a layer of interactivity to currently existing prototypes in the form of gesture recognition, and also the creation of new digital prototypes with the help of the first tool that it provides for that matter, the iMocBuilder.

iMocBuilder is a tool that allows the users to create interactive mockups easily. It uses images as input to create a HTML5 + JavaScript digital prototype that is uploaded on creation to the main server. Later, this prototype can be accessed either from the mobile device's browser or with the second tool of the framework, the mobile app called iMocTester.

The ways in which we can generate these inputs for the mockup builder are two: the user can upload his own paper mockups and transform them with the tool into a digital prototype ready for the app, or he can generate the mockup himself with the mockup builder, using a series of templates.

When the inputs are already inserted, we can select areas in the mockup interface, much like we will see in the next section when talking about Pop (2.2.1), and then assign gestures to it. This gesture will trigger the transition, which type can also be selected in this process, to the desired destination interface.

Once all of the interactions have been assigned and the interactive mockup is done, with the help of the iMocTester the user is able to interact with his interactive mockup in his own device, getting rid of the browser toolbars that usually are hard to hide.

The generated mockup makes use of a special library built on top of hammer.js[1] to enable the recognition of a variety of gestures, and runs in every device due to being written in JavaScript.

We borrowed this concept of the generation of a HTML5 + JavaScript playable prototype, as it is a way of allowing our proof of concept prototype to run in as many devices as there are able to run a modern browser with HTML5 capabilities. Luckily, most of the mobile browsers, due to its young age, implement all of the necessary features of the last specification, fact that would make our prototype suitable for many different devices.

[1] `http://hammerjs.github.io/`

2.2 Reviewed applications

In order to have a better understanding of how different interaction approaches can be used when it comes to creating interfaces in a prototyping application, we had a look at several tools that are currently available.

Those tools vary in their goals and nature, some being restricted by the devices they can be used in.

For instance, the case of a mobile app which is limited by the screen size of the device we are using, thus having to deal with a simple and well-thought interface due to the lack of space for the user to interact with it.

Finally, we have reviewed relevant applications in our area of research as well, which are used by many teams and possess the well-structured interfaces that allow a lot of possibilities to create a rich interaction experience. But they do it while keeping it relatively simple for a non technical user to be able to develop and iterate over a digital prototype without having to worry a lot about how it works behind the scenes.

With these various approaches we aim to achieve a better understanding at what has been done and also what is trying to be achieved, giving us an overview of how things have been done, but also evaluate whether they work appropriately or not.

2.2.1 Pop

PopApp[2] is a mobile application available for Android, iOS and Windows Phone. It has several pricing plans but all of the features are available also in the free plan with the exception that you cannot share your prototype because there is only one user allowed in this one.

The main point to remark about this app is the lack of difficulty when it comes to translating a paper prototype into a digital one. The main feature of this app is that it lets you take pictures at wireframes or storyboards and use them as interfaces of your digital prototype.

If you already have a storyboard of the application you wish to develop, you can obtain a digital version of the same in no time, and in really easy steps. This makes it perfect for designers who might not have the knowledge required to re-develop a digital version of their prototype nor have the time to invest (usually they will have a small amount) to learn how to develop a digital one.

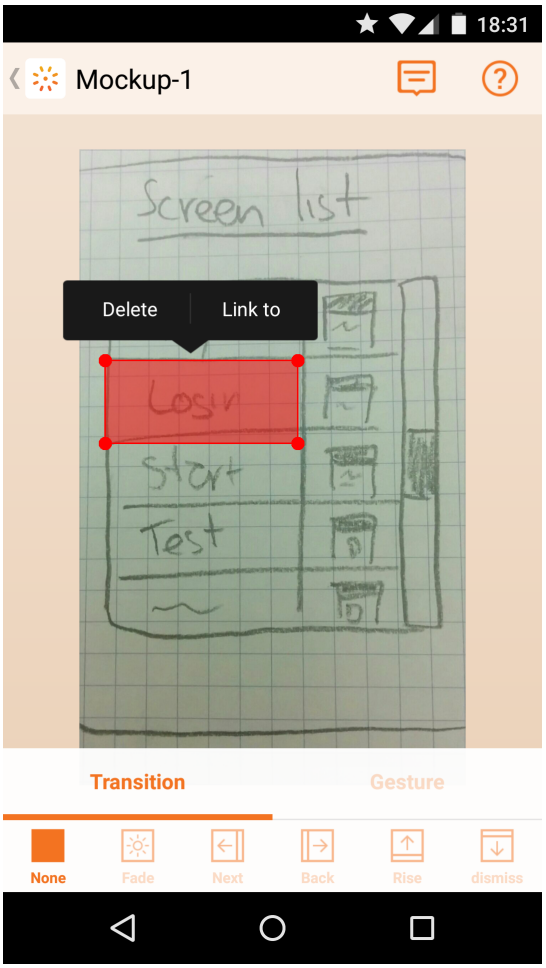

Figure 2.5: Pop App editing interface

On behalf of the interactivity level of support that this application provides, they focused their

[2] https://popapp.in/

application in the already mentioned behaviour. You scan your wireframe and obtain a series of screens that serve the purpose of being the interfaces of your digital prototype.

The interesting thing is that once you have this digital version, you can connect the screens between them also in a really fast manner just by selecting regions of the snapshots and assigning them a gesture and transition mode to lead to the selected destination screen and that ends the process. In these two steps you obtain a functional digital prototype that you can play and interact with.

One interesting thing extracted from the analysis of this application, is that everything can be a connection to another interface, something that is more restricted in other prototyping tools. We took this into the development of our own, by allowing every element in our interface to be a connection to some other interface, much like the freedom you would have in a paper scenario.

This, combined with the possibility of extending the widget palette, would give more flexibility to our tool, making every kind of interface possible by creating the appropriate custom elements.

However, as this ease of use may be the strength of the application, it is also its main weakness due to the fact that you cannot do much further than sharing this prototype with other users in your group in case you chose the pricing plan to enable these capabilities.

2.2.2 Balsamiq

Balsamic Mockups[3] is a trialware application available for Windows, Linux and Mac OS. There is also a web-version of the application which provides the same features and allows you to share your prototypes with other users as well.

As a wireframing tool, Balsamiq is one of the top tools in the field, widely used among designers. Its sketchy appearance removes the pressure of designing something that looks well aesthetically, and instead, lets the designer concentrate in the layout, how to make important elements stand out, etc.

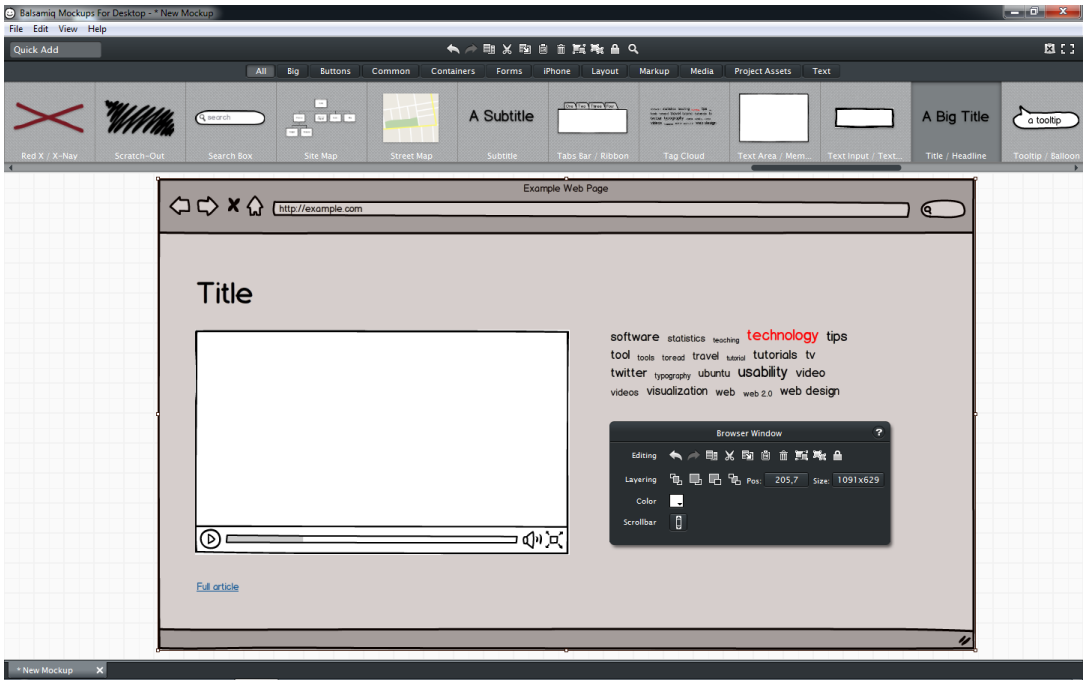

Figure 2.6: Balsamiq main interface example

It has a wide range of web-based interface components or widgets to use as well and, within little time, one can see this tool really serves it purpose of allowing you to achieve visible results fast and without much complication. The usage of the widgets is in the form of a widget library or palette from which you can drag and drop, and later resize and reposition the widgets in the interface.

This is one of the main features that we would adopt in our prototype as well. We did so because it enables a truly quick creation and edition of the interfaces as well as being quite intuitive. Any person, as potential user of our tool, that has used a computer, has had to drag and drop some file or resized a window in his desktop at some point. This knowledge relieves them from the task of learning how to manipulate the interfaces in our application, which at the same time, closes the gap between the prototype and the user creating it.

Other useful controls of this application that should be mentioned are that it possesses the

[3] `http://balsamiq.com/products/mockups/`

concept of grouping elements together built in. That makes readjusting parts of interfaces fairly effortless by just selecting the elements you want to stick together and then just dragging the block to the desired position.

On the other hand if what we want is to lock the position of a widget of group of elements, we also have the possibility of making them stay in the current position with a property of the mentioned group. These two options make refinement an easier and less manual labour.

There are also other libraries of widgets apart from the web styled ones, created by users of this application that are available to use for free in the same product's website, thus expanding the range of devices you can use this application to develop for.

Regarding its interactivity support, we could say it is actually quite limited, only allowing the linking of mockups via web links, or other so labeled 'control objects'. If you still want to get out of this scope you can do it using several workarounds.

Once you have the interfaces of your prototype properly connected, you can use the presentation mode to interact with the mockups in full screen mode. There, you will see those same interfaces except for the connected widgets over which appears a colored box to highlight them. When you click on them the application will navigate to the selected destination screen. However, there is no support for other gestures or transition types beyond this option.

2.2.3 Axure

Axure RP[4] is a very powerful enterprise application to do wireframing and create digital prototypes that has a lot of features apart from the typical toolset that we would find in a tool like Balsamiq, such as documenting, better interface editing options, more choices when it comes to events that can trigger actions in the prototype, conditional programming of events, variables, and much more.

It has several pricing plans according to some feature availabilities, especially involving the sharing options that are available in the professional version which is addressed at teams, while the standard one is intended to be used only for the development of the prototype itself by a single user.

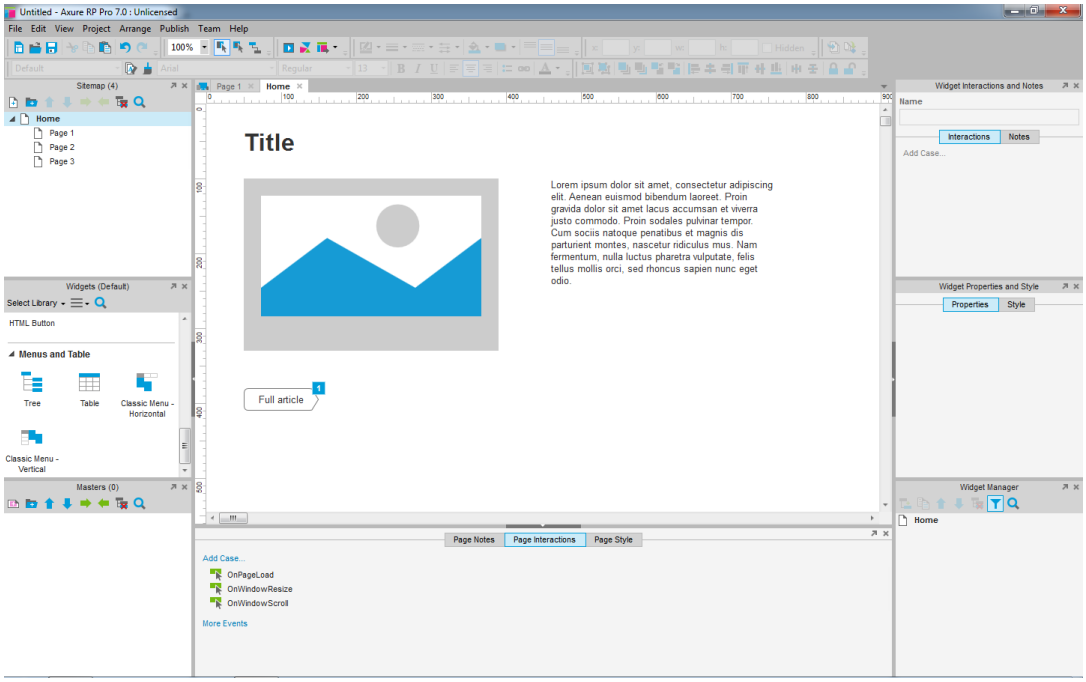

Figure 2.7: Axure RP main interface example

The usage of the widgets in this application is also in the form of a widget library or palette from which you can drag and drop, and later resize and reposition the widgets. However, this application also possesses shapes, and other objects that you can use apart from the web-styled typical elements to expand the variety of interfaces you can develop with it.

Additionally, this tool allows the user to update the widgets' instances with background images, colour fills, gradients, placeholders, shapes, and more in order to provide theming capabilities. With this, the prototype can also exist in a High Fidelity environment, making it useful to review how the design-less version would work in a real environment and also allowing the user to create personal styling for prototypes. This way, they extend the use to as many devices as one can design.

[4] http://www.axure.com/features

By default, the application has a lot of elements, web-styled, but there is also the possibility of downloading more widget libraries from some linked sites in the product site, thus allowing the user to generate wireframes with previously build packages of styles, relieving the user from having to create customized styles for the prototype himself.

As the previously referenced Balsamic, Axure RP incorporates the concept of grouping elements and locking them in a fixed position as well. In addition to these features, it has others also in the realm of widget editing like aligning elements between themselves in a Photoshop-like way, and the one that might be more useful: templating.

Axure RP provides this in the form of 'masters' which are templates for any kind of widget or complete interface allowing the reusing of components, and speeding the development of interfaces up, by removing the time of creating applications once the user has a good list of templates which he has created and reuses most of the time.

On behalf of the interactivity level supported by this application, it allows to create transitions or links between pages by selecting the widgets you want to act like triggers and later selecting the kind of behaviour you want them to react to (e.g.: events like hover, onMouseOut, onClick, etc).

But not only that, it provides ways of linking events to other kind of actions such as toggling the visibility of widgets, enabling/disabling them, setting images, updating texts, selecting options, setting variables, and so on. That brings a huge flexibility when it comes to programming more complex behaviours and interactions with our prototype but it also makes the use of it less accessible and rough for unexperienced users or simply users that might not be used to programming when creating a prototype for their application, such as designers.

It is also important to note that this application not only lets you interact with the prototype you have created with it, but it lets you export the prototype to html format as well, since it is the same format it uses when you use the presentation mode to interact with the same making it easy to share and interact with the prototypes in different computers also due to the fact that web pages can be displayed in almost every device.

It is arguably the most complete tool available, and also the more flexible, due to all of the possibilities that it offers, from several ways of deeply customizing the aesthetics of the interfaces, to complex logic behind the actions that the user can trigger in the presentation environment. Some of the capabilities that it possesses would be desirable in a potential evolution of our proof of concept prototype into a final product.

However, we have not considered them necessary as they overcomplicate the process of creation and manipulation of the interfaces and also because they do not go further into enabling more natural ways of interacting with the application itself, rather than taking a very programatic-like approach to the development of the application prototype.

3 Approach

3.1 Design Process

Having the issues explained in the Motivation section(1.1) in mind and also all of the information about how tools we had analyzed solved those explained in the previous section, as well as their strengths and weaknesses, we began the second stage of this thesis, the sketching.

With that latter analysis we also strengthened our ideas about what concepts we needed in our proof of concept prototype, but also discovered interesting features that would be considered as future extensions of the tool we were about to design.

In this following phase, we sketched several interfaces of how our prototype should look, trying to find the way to accommodate all of the features we considered necessary while not overloading the main interface. As we will see at latter stages of the process we had interfaces with lots of elements that made difficult the understanding of how the tool behaved and tried to simplify to the maximum the controls in order to make the barrier less high for the users.

3.1.1 Previous literature

Before starting sketching interfaces, we needed to get a deeper knowledge of how sketching and storyboards work in real life. That would give us a better understanding of ways of solving certain situations, such as representing connections between parts of the application, and it would give us ideas that we could translate into features or characteristics of our future proof of concept prototype as well.

In order to do this, we took a look at several paper prototypes, and also literature about how to create them, such as Sketching User Experiences: The Workbook [1], to get a better grasp at the concepts beneath this topic.

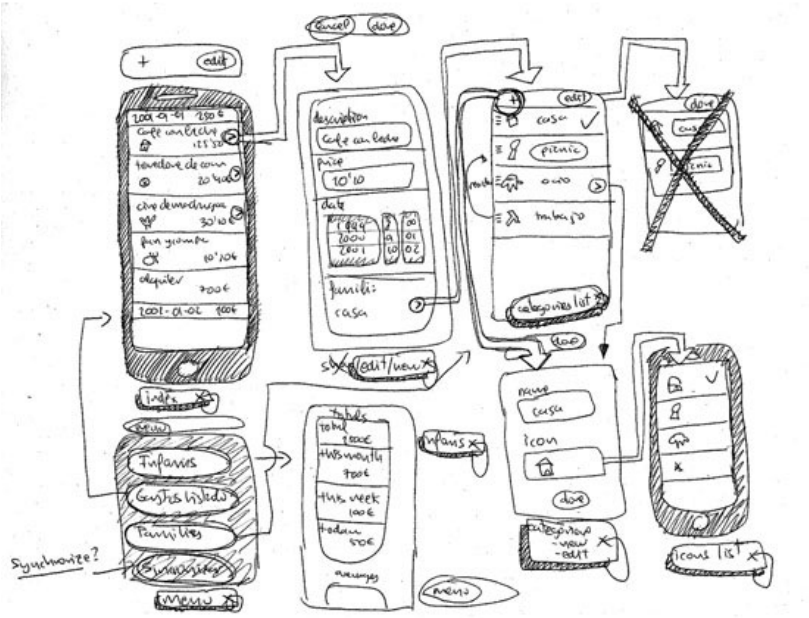

Figure 3.1: Rough sketch of user interface flow on a mobile app. Image by Fernando Guillen[2]

We focused on the idea of the storyboard itself, and the way it transmits both the content and the intent of the actions a user can do with only looking at the sketches of the interfaces and the links between them. That idea would be refined into the key global concept that is the umbrella of our proof of concept prototype: Everything is contained in the sketch.

From this research, we extracted the main ideas that would be part of our next sketches on how to represent interactions, and would be later incorporated into the future proof of concept prototype such as the general view of the application when we activate the Linking Screens mode.

That would help us see the whole application at once and understand the relationship between the elements or screens of the same through the usage of arrow elements to connect the different parts of the application in a visual way like we would have in a paper environment.

At the same time, we would provide another set of features, feeding from the knowledge of digital prototyping applications to finally offer a tool that had the ease of paper prototyping combined with the powerful features of digital tools.

In the next subsections, we will see different sketches of our proof of concept prototype. We will explain how they evolved in time, and the decisions behind that. The aim of this phase was to explore new ways of creating and manipulating interfaces rather than obtaining a fully detailed paper prototype to later translate into our digital prototype.

For this reason, we will also see some alternative ways of interacting with our application that could be interesting for future extensions or deeper exploration on the field.

[2] http://www.flickr.com/photos/d2clon/4402993445/

3.1.2 Initial sketches

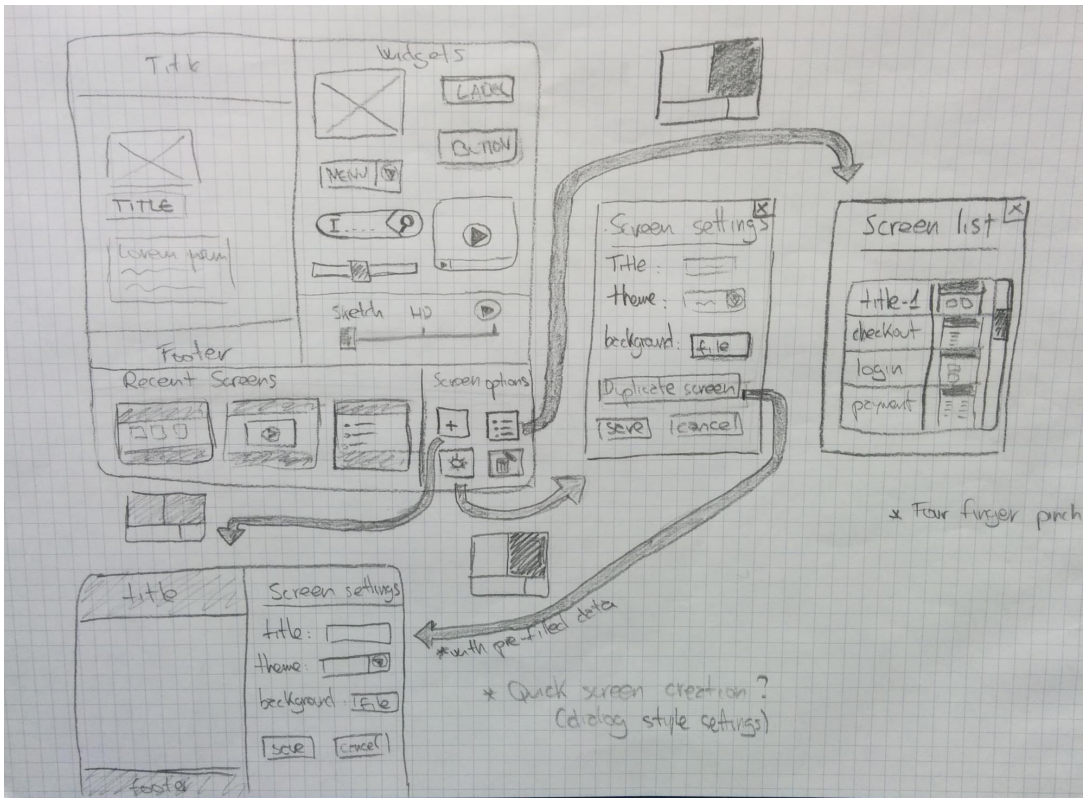

Figure 3.2: First 'complete' sketch

On this sketch, we had all of the features we had in mind at first, except for the collaboration that would come in the form of another editing mode later on. We can see the main interface which contains several areas, and at first we see the screen which is currently being edited, the palette is to its right, containing several widgets that can be dragged to the screen editing area to compose the interface. Below the palette we find the mode selector panel, with a slider to select between the modes that were available at that point: Sketch mode, High Definition mode, and Play mode.

In the lower part of the interface, we find two areas more, the recent screens panel and the screen options panel. In the recent screens panel, you could have an overview of the last screens you had edited in order to have a quick access to interfaces you had recently worked in, assuming that those would be the ones you wanted to access with a higher probability. Later in the refinement of the prototype we would find that feature to be less useful in comparison with the layer concept that will be later introduced, and would be removed from the prototype in favor of letting space for the layer panel to be big enough to be useful.

At its right we had the screen options panel, which contained 4 buttons.

The first one was the 'add screen' button which triggered a new pane appearing on top of the right side of the main interface, covering the widget palette, that allowed us to customize the screen we were creating in that moment. It contained several options, such as name, theming,

a background image, that would not be part of the final prototype due to the lack of a backend to support the mentioned features.

The second one, the one to its right, was the 'screen list' button which replaced the formerly mentioned right area with a list of screen selectors in order to change the screen that was currently being edited. It also contained a little preview of the screen that would be dropped from the prototype later on due to the lack of space for the preview to be useful. The interface was already quite loaded with elements to interact with, so we did not want to add more complexity for the user to do the simple tasks.

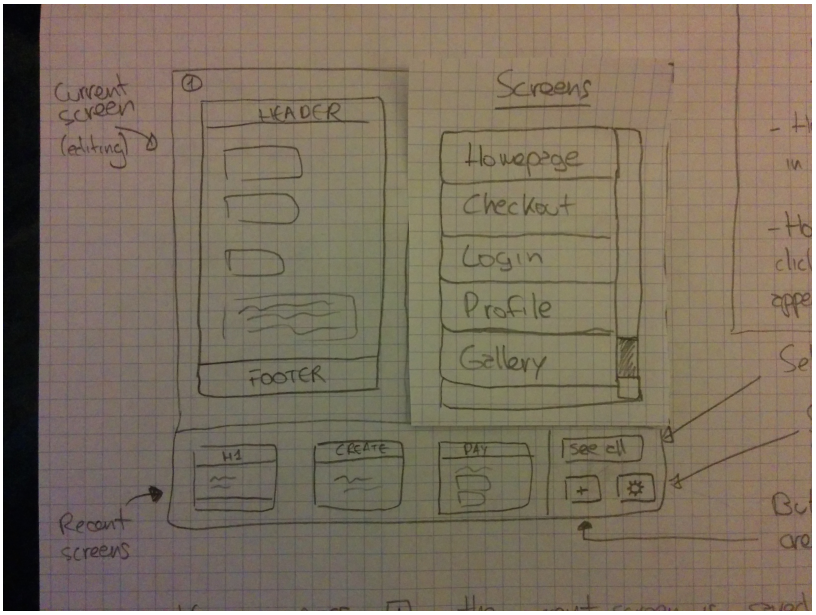

Figure 3.3: Selecting a screen from the list

The third button, was the 'screen settings' button which triggered the already mentioned screen settings pane on the right area, with the exception that the panel that appeared also contained an extra button in order to duplicate the current screen to a new one. That was meant to ease the revision of interfaces process, something that later on would be provided by the layer concept.

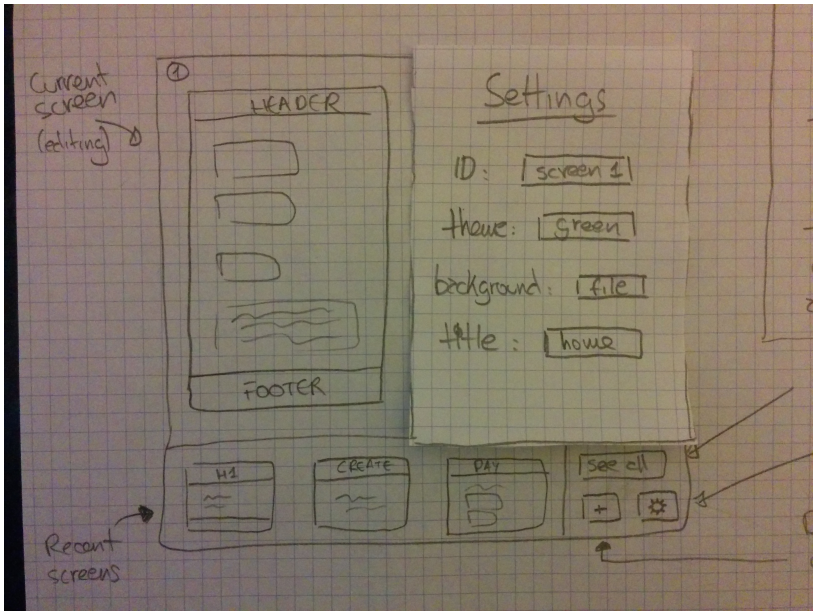

Figure 3.4: Changing the screen's settings

Finally, we found the 'delete screen' button that deleted the current screen being edited.

It can also be seen in the annotations, that a quick screen generation was considered, in the form of a dialog style window that would let us create an interface with pre-determined parameters in a fast way.

3.1.3 Final sketches

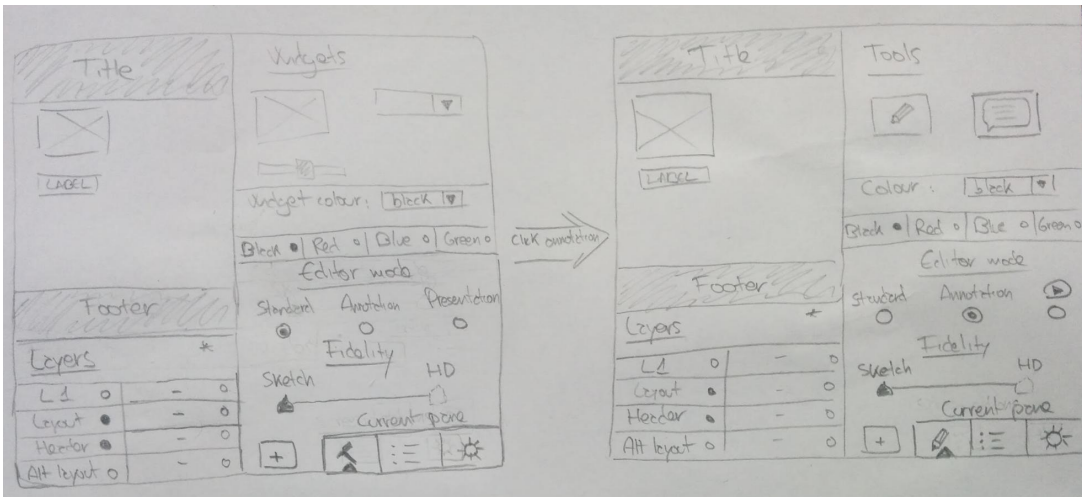

Figure 3.5: Final sketch version of the main interface

On this sketch, we can see several variations the main interface has undergone. It would not be the final interface the prototype would have, as there are other aspects of the interaction with the application we discovered while developing the prototype itself, but it resembles it the most. Below the screen editing area we see the formerly mentioned replacement of the recent screens panel by a first version of the layer concept, that did not include yet the previews of the layers in it.

This first approach already gave us though, the possibility of different revisions of interfaces, and also took into account the visibility toggling with the use of a little round selector within the button itself. The button, on the other hand, would be used to select the layer.

Below the widget palette, where we could find the mode selector before, now lies a widget colour selector which would aid in creating different versions of the interface. That concept was later transformed in an specific colour per layer on the final prototype to simplify the separation of revisions or roles. Under this selector we could find the colour selector for the annotation mode that was appearing here for the first time. This selector would be finally displayed only if the annotation mode was selected, making the interface more clear and simple to use for the user in the real prototype.

On the lower part of the right area of the interface we find the previous mode selector, but with slight changes. We realized that, conceptually, the previous organizing of mode selectors did not make sense with the new mode added, so it was reorganized into different modes of edition, and a new selector was added to chose the current level of definition of the interface displayed which contained the two previously mentioned modes: Sketch and High Fidelity.

Finally, on the bottom of the right area we found the add screen button, and a tab selector which changed the tool displayed on the widget palette area between the palette, the screen list view, and the settings view. This tab selector was moved on top of the area to which it changed the contents later on, to make it more understandable for the user.

On the sketch next to the one described above, we can see an example of the interface of the prototype when the annotation mode was activated, replacing the palette of widgets with some tools to annotate the sketch which where the pencil and the text box. This tools would be later replaced by a stroke thickness selector and the colour selector that we could also see in the regular main interface, as the text area was considered not valuable for this prototype.

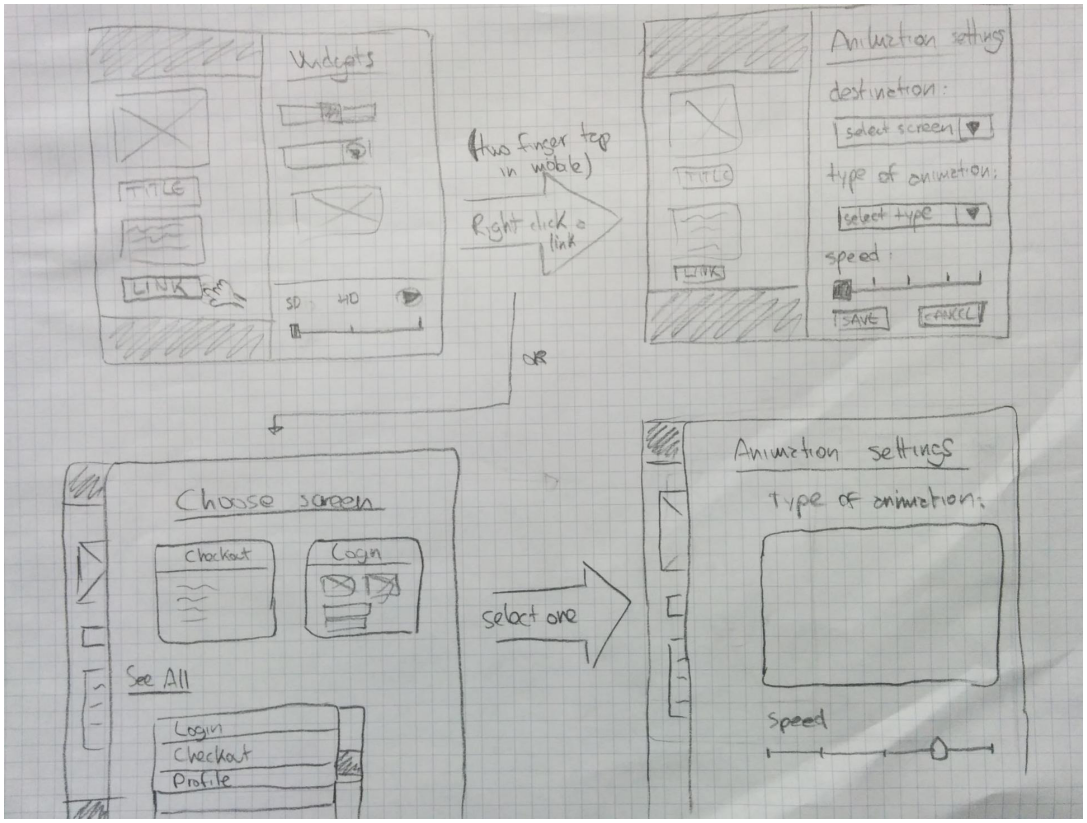

Figure 3.6: Ways to link screens

On this sketch we can see a possible process of linking screens. Everything begins by right-clicking a widget while editing an interface, which would trigger a menu on the right pane asking us to select some properties: the destination screen, the kind of animation, and the speed in the version that is on the right of the image.

Below, we find a 2-step version of the same which asks us to select a screen from the recent used ones, and also allows us to select another one if needed from a complete list of the screens.

Once selected, it would show us a tactile area expecting us to draw a curve or function which would determine the kind of animation the transition would have, and also the speed of the same.

However, the latter option was dismissed due to the technical difficulty of properly recognizing different functions added to the lack of connection between a drawn shape and the animation it would trigger making the whole process not very intuitive in the end.

The option implemented in the prototype quite resembles the first one, only with the change that it is executed in a special mode, the screens mode, that allows us to have an overview of the whole list of screens and connect them visually rather than in a select menu.

That added to the fact that when a connection is done, we obtain a visual connection between the widget and the destination screen in the form of an arrow, contributes to the goal of containing everything in a sketch-like medium, and lets us understand the relationships between

interfaces in a fast glance at the overview of screens.

3.1.4 Alternative features/layouts

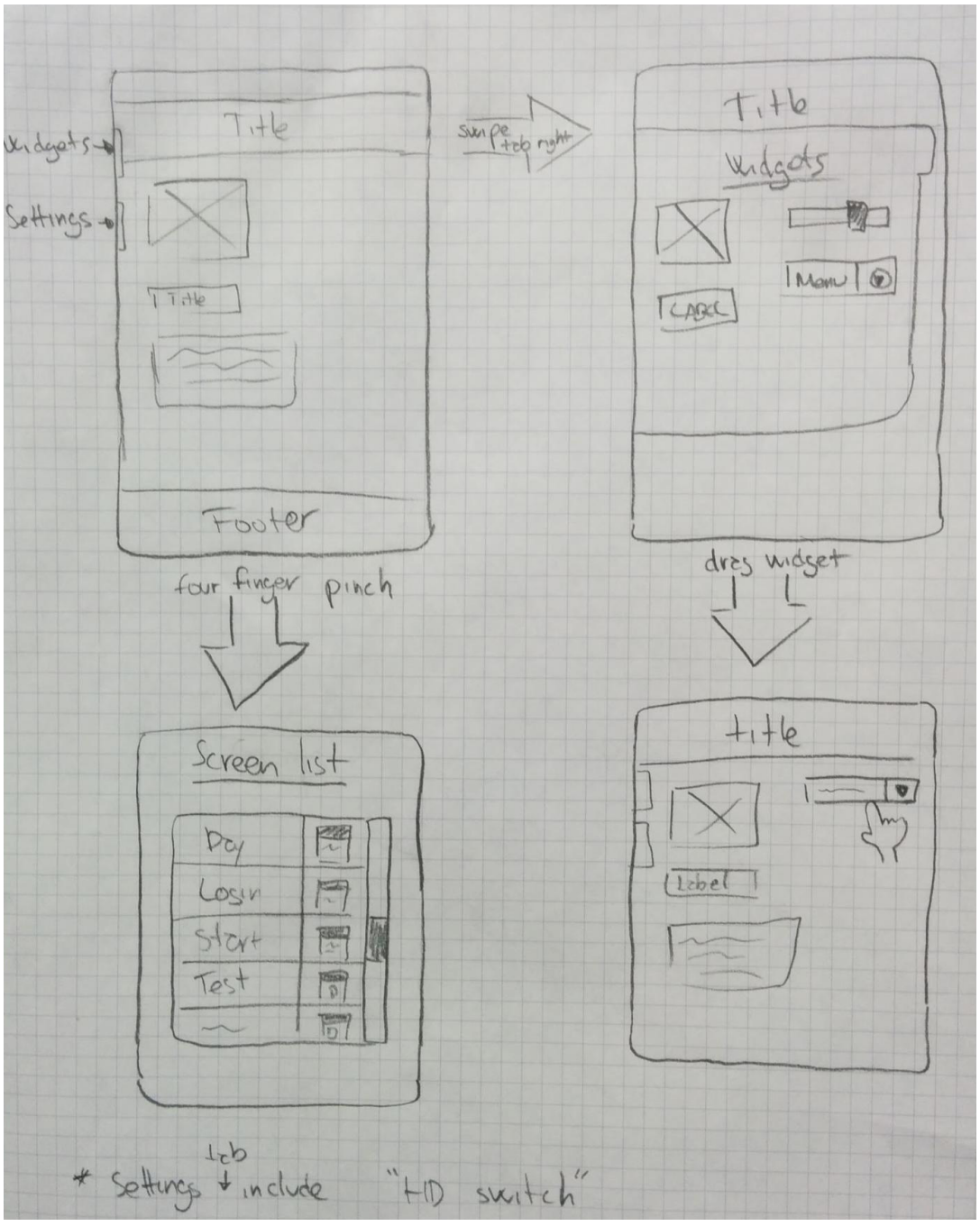

Figure 3.7: Potential mobile (small display) layout

On this sketch there is an alternative layout for mobile devices with reduced screen resolution, in which we can see how several features could be displayed so that we do not lose the ability to still be able to create interfaces in those devices. In this very simple exam we can see how the right area panels are replaced by little tabs that can be shown by clicking on them, displaying then the known panels such as the widget panel, and the screen list.

However, later on the support for this kind of resolutions was dropped, mainly due to two reasons. First, it was not a goal of our prototype to be able to develop interfaces in this kind of scenario, which also presents severe restrictions when it comes to usability and layout spacing.

Secondly, we decided to drop it in favor of focusing in other core features that were more essential to the thesis, and thus there was not a thorough redesign of the interface and all of the options to adapt it to this reduced environment.

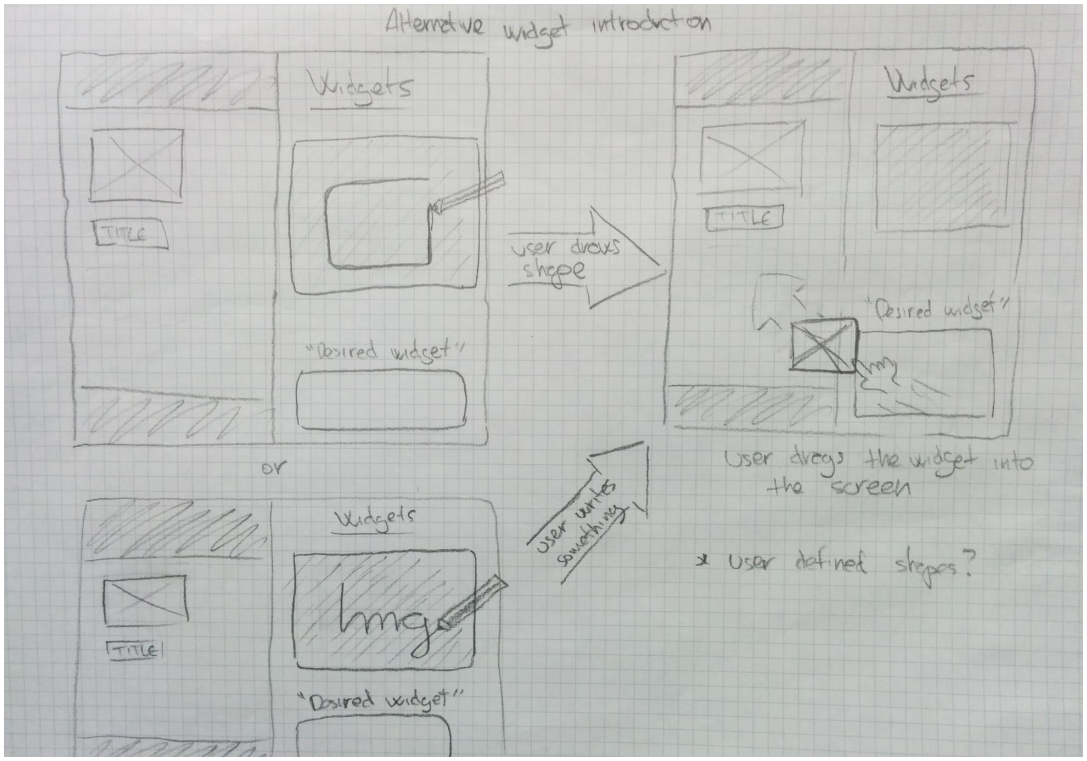

Figure 3.8: Alternative way of introducing widgets

On the last sketch we can see two alternative ways of introducing widgets that were conceived in an attempt to seek different ways of interacting with the prototype, closer to a full sketching interactivity experience. The first one, shows us a tactile area, where we could draw a shape, namely the shape of the widget we desire to use, and the system would recognize it and dispose it in the container below it for us to use it.

In the second version, instead of drawing the shape of the widget, we chose to draw the name of the widget, or possibly a shorter alias, and by doing that we would obtain the same result as in the previous version, our desired widget would appear in the container under the tactile

area, and then we would drag it to the screen.

Even though this solution can seem to be really good at first, it results in a slower creation of the interface itself, by having to repeat this gestures in order to obtain a single widget every time, also it can not be that obvious how to draw quickly a shape for a menu widget, not to say to recognize it by the prototype. The same would happen in the scenario where we chose to implement the second alternative where we write the name of the widget we want to use.

3.2 Key Concepts

As formerly mentioned, this proof of concept relies under one global key concept: Everything is contained in the sketch.

This articulated the research that we did as well as the features we would chose to implement. From being able to grasp all the aspects of the whole application in a single view, to being able to create and manipulate every particularity of our application with a visual approach, everything fell back to this big principle in the end.

This in turn, would be in favor of our goal of empowering sketching interactivity, the first of the targets we had in mind at first.

Once we finished our refinement of the sketches, we had a small set of sub-key concepts that would be the pillars that would sustain the global one on our proof of concept prototype:

3.2.1 Widget sketch-like palette easily draggable and adjustable

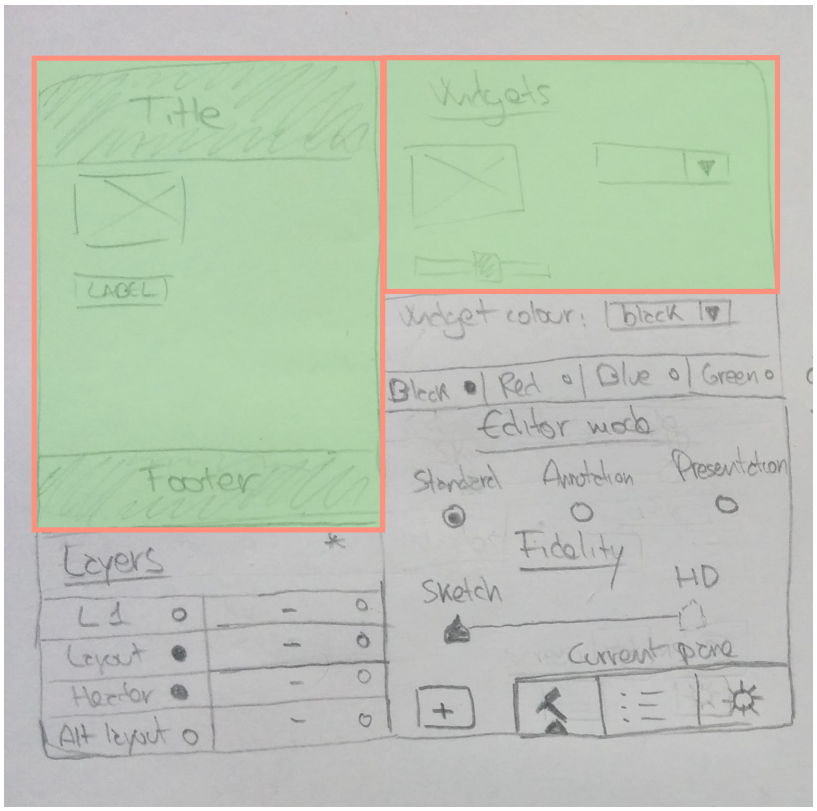

Figure 3.9: Closer look at the widgets palette

A palette of widgets like the one we have seen provides us with an easy way of creating interfaces, just by dragging the elements that we want to use to the screen area, and then it also lets us to reposition them at ease by redrawing them to where they have to be as well as resizing them like we would resize a window in a computer. Editing is easy as it is to remove

them, which only consists, again, in dragging them to the trash bin contained in the widget palette area, which will remove the widget as well as the connection that it could have to other screens in the form of transitions.

This approach makes the creation and refining of the layout of an interface really easy and fast for the user, letting him create an interface in a few minutes as no further parameters are required in order to obtain a complete interface.

3.2.2 Layers for versioning of interfaces

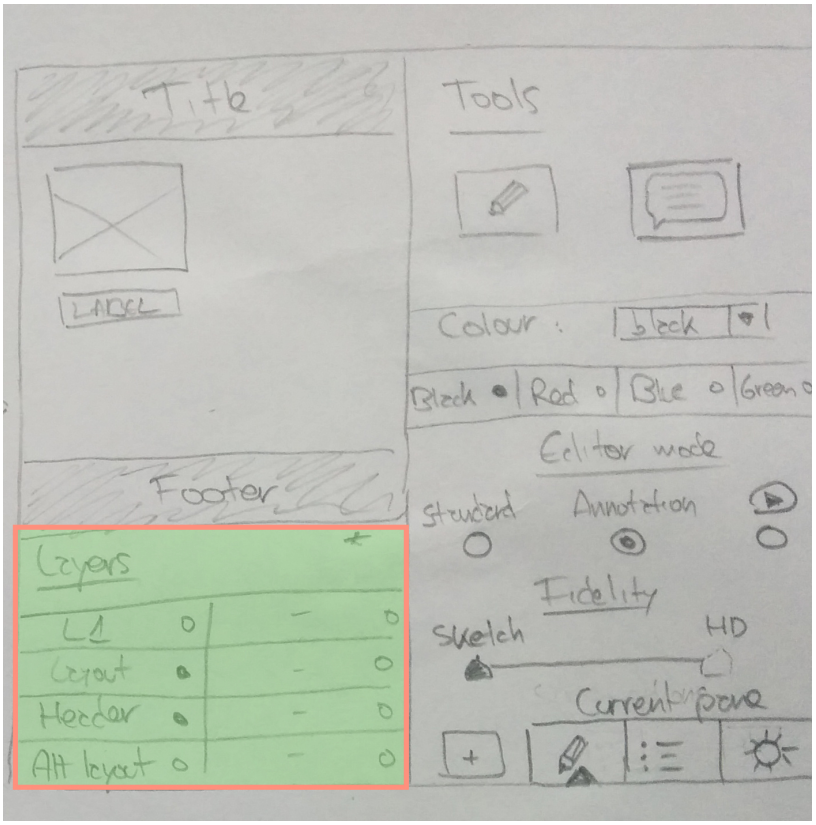

Figure 3.10: Early version of the layer concept

This eases the creation of alternative versions of interfaces, as well as the fast visualization and combination of different layouts of components. It also allows the collaboration of different roles by separating their changes, revisions or annotations in different layers provided with different colours, to quickly distinguish between them.

This way, we can just clone a layer with a certain layout structure, manipulate it, combine it with other layers, and explore possible different interface compositions without having to worry about storing them in complex ways in order to have them physically and visually separated.

3.2.3 Different fidelity modes for agnostic, sketch-like styling but also specific High Fidelity version

Providing a switch that changes the level of fidelity of the components of the interface, enables the prototype to be a tool for general purpose prototyping, without being too specific of a device or environment.

This in turn allows the user to focus on the layout of the widgets and forget about concrete theming issues. In addition, it provides a way of checking how the application would look in a real scenario, with the use of High Fidelity widgets, that belong to a particular device, framework, or operating system.

This, combined with the possibility of creating custom themes, allows the user to be able to represent any kind of device and interface by just using his own custom widgets, which are represented by images, so that the user only has to create a set of images that resemble the target he wants to emulate.

3.2.4 Play mode to enable real interaction

A most necessary mode within the proof of concept prototype to permit the user the real interaction with his prototype, allowing as well a detection of usability issues when actually using it like a real user before going into the development stage. We avoid in this way major changes later on that would be much more expensive to address.

We can use our prototype either in the sketch-like fidelity level to focus on the distribution of the elements for example, and later in the High Fidelity level, getting us closer of the final application we would like to develop next. In any case, we can trigger a transition from any element on our interface which will bring us to the destination screen using the transition properties selected when connecting them.

3.2.5 Annotation mode empowering revisions and comments

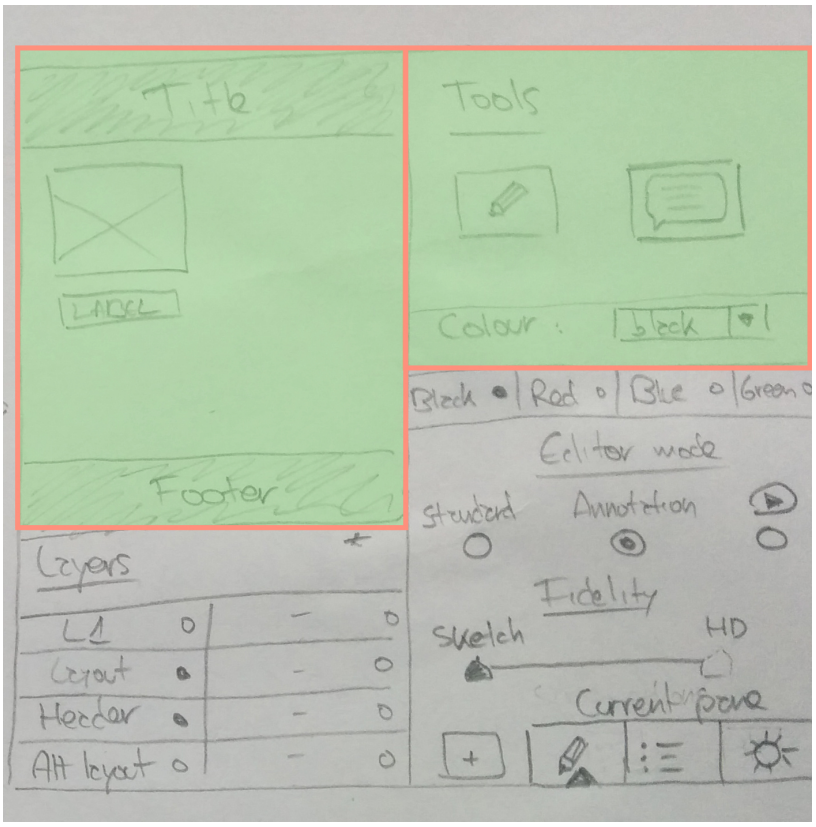

Figure 3.11: Closer look at the annotation mode

A mode that makes it possible for different users, or the same one who is currently using the application, to annotate comments and details about the interface, on the same interface that he is editing, empowering collaboration in real time between different users, or easing the collaboration in conjunction with the layer concept.

This allows the users to share their opinions and revisions of the interfaces they have reviewed in a fast manner, instead of having to send notes and comments back and forth on other channels of communication like e-mail.

This concept, as well as the layer concept, could be vastly empowered with the addition of a backend component. With it, a remote collaboration live session could be established, enabling the simultaneous interaction of several users, and bringing the real-time co-development of interfaces to a fairly interesting possibility.

3.2.6 Linking Screens mode: Visually connecting interfaces

A mode that allows us to connect interfaces by establishing linkings between source elements and target screens. This is done by visually tracing a path between those two elements, a trace that also expresses the intent of the connection.

This path becomes an arrow on completion of the trace, showing the direction of the connection. In addition, for instance, depending on the kind of gesture that would be used to recognize that interaction, the resulting visual arrow would have different properties, like colour, tip shape, etc.

With this approach we achieve a really visual and intuitive way of connecting the interfaces of our prototype, as well as allowing the user to have a complete understanding of the prototype and its behaviour by just looking at the overall sketch, and its internal relationships.

In turn, this could be combined with an already mentioned, tracing shape recognition, in order to specify properties of the transition between the two interfaces, from the same manual input from the user.

4 Implementation

In this section, we will give a little explanation about the technologies involved in the development of this proof of concept prototype, as well as the reasons why they were chosen in relation to the needs we had and the features wanted to build that we distilled from the previous analysis of the current existing tools.

We will also discuss the architecture of the application. We will take a look at what services are involved, what is their purpose, and how they interact in order to provide the features we needed.

Finally, we will see how some of the features were actually implemented, taking advantage of these technologies.

4.1 Architecture

IxSketch is developed using the latest web-technologies in order to be able to bring interesting interacting features to life. It is written in JavaScript with the help of jQuery, and uses HTML5 and CSS3's recent specifications to achieve this purpose as well at the interface level.

4.1.1 General overview

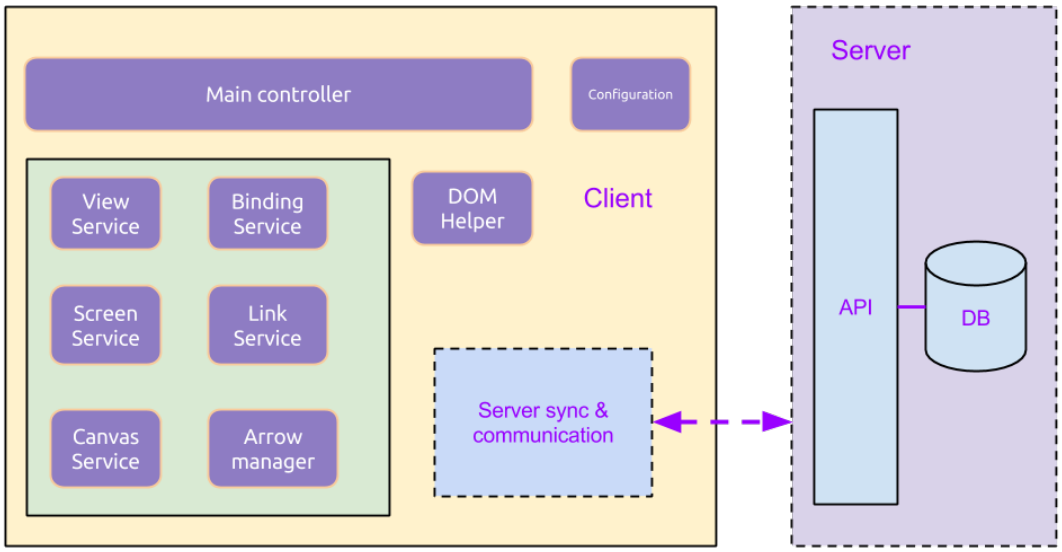

Figure 4.1: Architecture diagram

As we can see in the diagram, the application uses a series of services, controllers and classes that will be explained in the following section. Note that the parts with dashed lines belong to the backend part of this prototype. This part is not implemented, even though it is referenced several times in this thesis, as it is considered to be one of the potential paths for the future works.

The main controller's task is to initialize all of these services and then start the application itself. Beyond that, the rest of the logic of this application relies on the several services that compose it.

To abstract the main application from all of the configurations the libraries we use need to be provided with, we have encapsulated those in a ***Configuration service***, which exports the constants and configuration declarations, for the rest of the application, to keep everything clean and organized.

The main orchestrator would be the ***View Service*** who is in charge of manipulating the interface, and also saving the state of the same. When a mode change is triggered, it is the View Service that updates the application's interface for the same. To abstract the View Service of directly manipulating the DOM, and thus being coupled to the specific 'implementation' or structure of it, we work with a ***DOM Helper*** that mainly deals with the creation of raw DOM elements, abstracting other components of knowing how to build the elements of the interface themselves.

In order for the interface to receive the inputs of the user, there is a ***Binding Service*** which encapsulates and initializes the bindings of the UI controls of the application to the behavior they have to execute. And every time some action happens, it is likely we have to save the new state to the appropriate screen. This will be the task of the ***Screen Service***, which has

the set of available screens in its properties as well as it exposes useful Screen management operations to the rest of services.

If we happen to be using the Annotation Mode, we will be interacting mostly with the ***Canvas Service***, which encapsulates all of the behaviour regarding the this mode. It manages the creation, erasing, copying, and other operations that affect the canvas elements in the application due to their special behaviour.

Finally, if we activate the Linking Screens mode, the ***Link Service*** will manage the arrow connections as it will be explained in the next subsection.

Apart from the services, we also find these classes that represent the main concepts of the application:

- ***Screen*** class, which encapsulates all of the data related to a screen, such as, layers, connections, canvases, etc.

- ***Connection*** class, which owns all of the information related to a connection, such as, source and destination elements, and properties of the connection like the gesture to trigger it or the speed of the transition between screens.

- ***Arrow Manager*** class, which encapsulates the logic regarding the linking screens feature of the application, that we will see in further detail in the next section.

With this general view of the architecture of the application, we will now explain some of the inner workings involved in the features we had to develop.

4.1.2 Standard mode

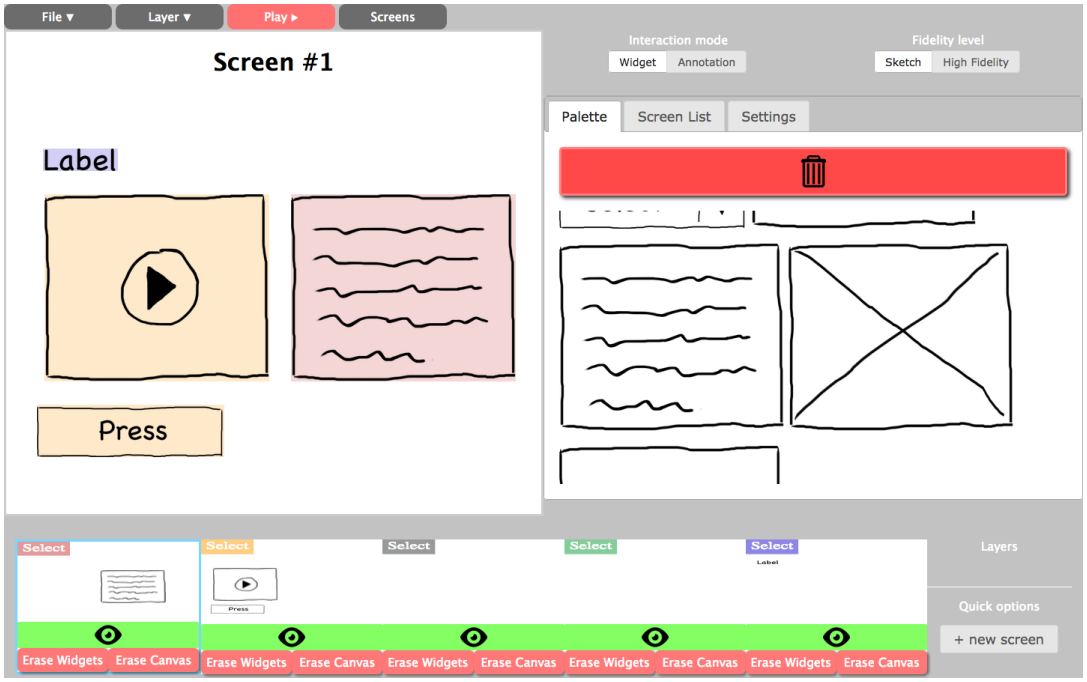

Figure 4.2: Screenshot of the Standard editing mode

The first mode that we will see of this prototype is the Standard mode. This mode is the default mode active when the application starts and uses three main areas. In the first one on the left, we see the interface we are currently developing. At the right, we have the palette of widgets that we are going to use in order to build our screen.

We also possess other controls like the screen selector, to change the screen that is currently being edited and the fidelity toggler, to change the level of fidelity of the widgets. As the last area, we find the layer previews below these two areas on top with their display and clearing controls.

To implement these last preview controls we considered interesting to take advantage of two features of the broader set of features from CSS3 [11]: the scaling transformations and the flexbox layout model [6].

The first one would allow us to be able to replicate our screens or interfaces all over our application without having to take all of the size calculations into account, enabling an easy management of the screen concept which is actually composed by several layers with different behaviours.

The flexbox layout model was useful in many ways. It allows us to develop our prototype faster and in a less static manner, as flexbox makes the distribution of space between elements easy and number-free, meaning that we don't assign explicit values to the sizes of the element. Instead, we let the flexbox grid distribute them intelligently according to the desired properties we use.

It helped in the elasticness of our design as well, letting it grow accordingly to the size of the display in use. It is used all over the interface of our proof of concept prototype, as well as in particular, in the layer previews.

Figure 4.3: Screenshot of the High Fidelity toggling

The way to build an interface is straight. We just drag the elements of the palette that we want on our interface area on the left, adjust their position or size if needed, and we already have an interface ready to be connected.

We also have other possibilities that facilitate the editing of this interface as it is not obvious at first how they should be built. In order to ease this process, we provide the concept of layers.

Layers can be rapidly distinguish as the widgets contained in them will get a characteristic colour depending in the layer they are in. These colors can be either used to distinguish between different revisions of the interface, or to be used by different roles collaborating in the development of the interface. In addition, layers can be easily copied from one to another with the layer menu on top.

This allows us to create variations of a particular interface without having to modify the original, or re-create it in another layer manually.

Layers can also be hidden to allow the user to compare specific iterations of the interface in particular. The contents of the hidden layer are still displayed in the preview area, making it possible to quickly toggle them.

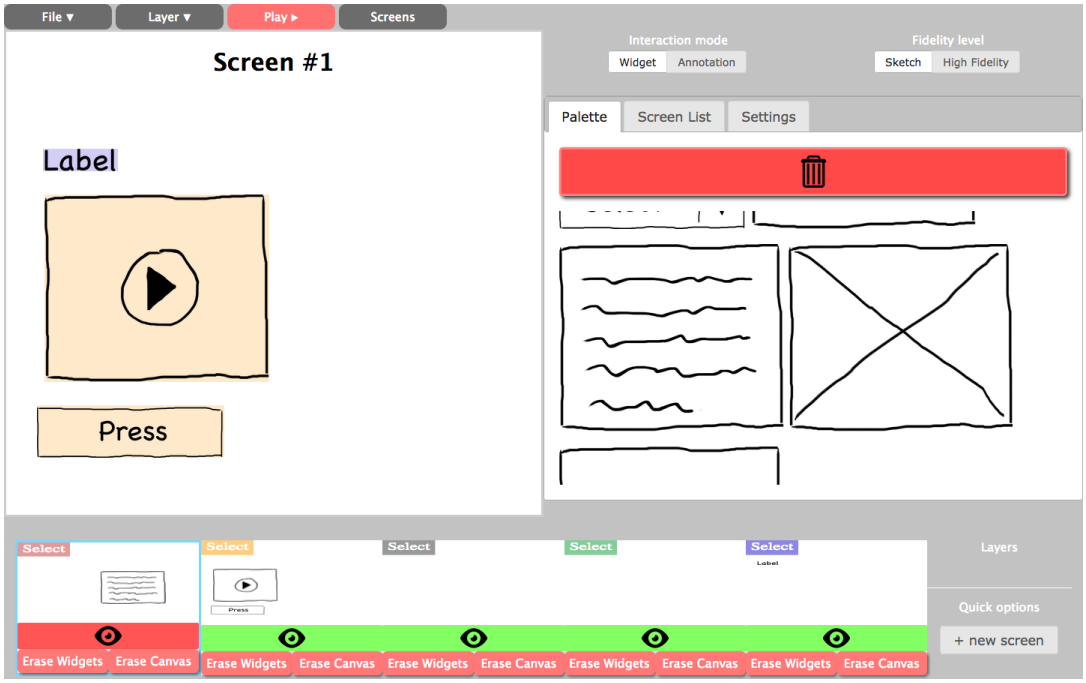

Figure 4.4: Example of toggling layers visibility

All of this behaviour relies in a set of layer elements stacked on top of each other as it will be explained in further detail in the Annotation Mode section. Every time we drag an element from the palette to the editing area, it will be dropped in the most upper layer.

This layer will be conveniently positioned through the use of the z-index positioning, and then the widget will be cloned into that layer with all of the resizing capabilities and the appropriate layer colour.

If we do not want to use it anymore, we just need to drag it to the trash bin area in the palette, and it will be removed from the layer as well as the connections there could be starting from that element.

In order to achieve the preview concept in the lower part, we made use of the CSS3 Scaling transformations previously mentioned. We cloned the separate layers that are stacked in the editing area into the preview boxes and then applied an appropriate ratio of scaling calculated dynamically with the help of these new properties.

The result is a segmented view of the whole interface in little previews that can be helpful when it comes to combining interfaces parts, separating revisions, etc. These previews are refreshed every time there is a drop event in the top area, or a new trace is added to the canvas layers in the Annotation Mode that will later be explained.

This keeps them consistent at all times with the contents that are appearing on the main editing area.

4.1.3 Presentation mode

The Presentation mode, identified with the play button on the top menu, enables our prototype to be interactive, by adding all of the connections we have defined in the Linking screens mode into the actual interface.

With this we can explore the prototype in use, detecting usability issues, or other problems that can be seen by other persons involved in the process of the development.

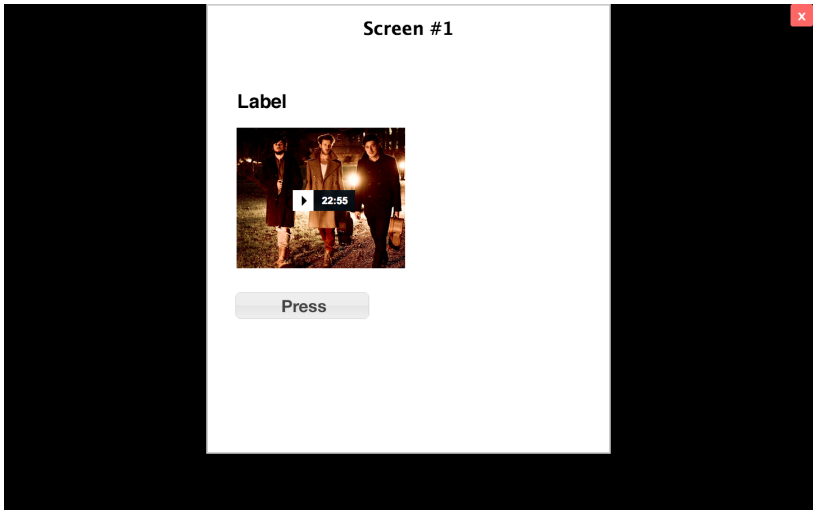

Figure 4.5: Screenshot of the Presentation mode

To enable this feature, on runtime, when this mode is activated, we copy the contents of the current screen being edited to the Presentation Mode area, and then iterate through all of the connections that screen possesses.

On each connection, we obtain all of the information regarding the kind of link we are dealing with, the gesture involved to trigger it, the speed of the animation, the destination, and we dynamically use those values to generate the linking in the screen.

Even though the architecture is using the jQuery library to recognize gestures, we could easily introduce a wrapper in the middle of the process that not only took the gesture, but that could also determine which library it needs in order to recognize or bind the event given a certain gesture.

This would extend the possibility of using several tools to broaden the range of gestures the application could recognize.

4.1.4 Annotation mode

The Annotation Mode, as previously stated, brings us the possibility of sketching on top of our interface, revisioning interfaces, collaborating, etc.

In order to implement this feature, we have taken advantage of the HTML5's Canvas element which provides us with an area in which we can trace lines easily by using the HTML5 Canvas manipulation API [9], that is crucial in the development of the this mode.

The Annotation Mode basically relies on several layers of canvas elements coexisting in the frame of the interface area.

In contrast with the rest of the markup that a particular screen contains, these elements have to be persisted in another way, but thanks to this JavaScript Canvas manipulation API the process is fairly simple.

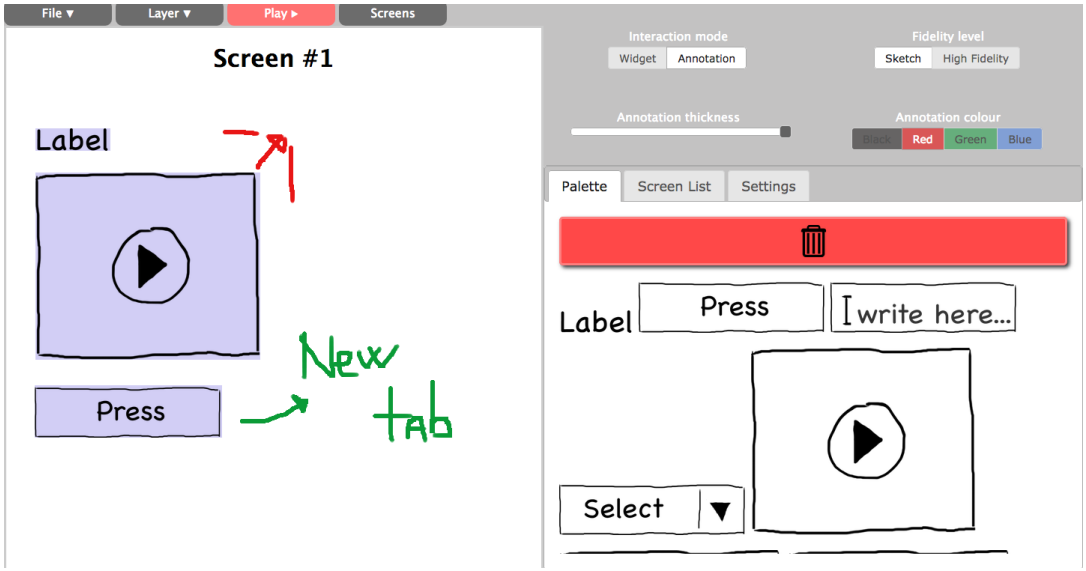

Figure 4.6: Example of Annotation mode

When using this feature in the editor, we find ourselves handling several properties of the elements that compose the screen editing interface.

First, we have the several layers that contain the different versions of the interface, and also in its other mode, the canvas associated with those, that allow tracing in each of them. They are all stacked on top of each other, and reordered when the layer is selected through the use of dynamic z-index positioning.

That manages the layer concept ordering, but when we are in the standard edition mode we still have a canvas element on top of the layer that prevents us from dragging elements to it, or manipulating the ones that are already in it.

We could use again the z-index property in order to solve that, but instead we used another tool to make it less demanding for the browser and clearer in use.

We disable the pointer-events property of the element. This allows the events that would be

detected on this element to go through the canvas layer to the objects layer, thus only having to toggle this setting every time we change from Standard Mode to Annotation Mode, and not recalculating z-index's and re-positioning elements for this feature as well.

Another particularity of the canvas element that we already mentioned when talking about JavaScript and the Canvas API, is that in order to save its contents, it does not suffice with storing the html contents of the same.

Instead, we have to export an image of the canvas itself, that will be stored on the Screen class, and then re-paint it when we reattach the screen and its contents to the main interface of the application.

4.1.5 Screen linking mode

In order to be able to link our screens, we dispose them in a grid of N rows and 2 columns. As we are using SVG graphics for the arrows, and they need special position as HTML elements in order to exactly match the center of the widget that acts as a 'source' and the center of the destination screen of the transition, we find ourselves a situation regarding the calculation of the positioning.

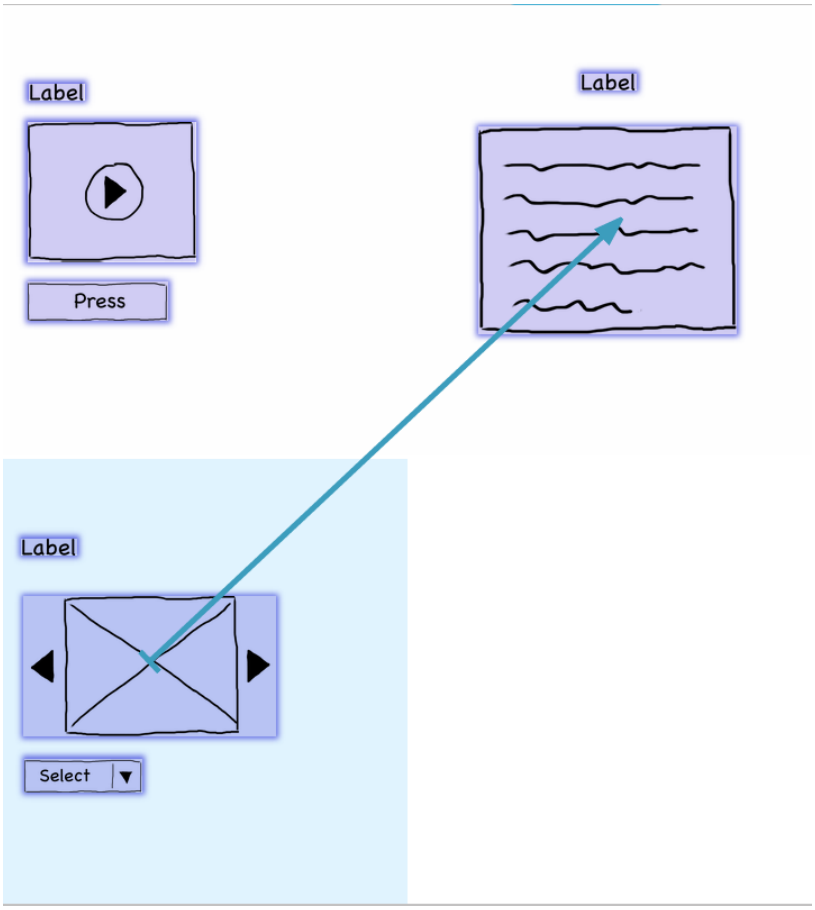

Figure 4.7: Example of screen linking

We already have a grid of screens that possess relative positioning. All of the elements inside the grid are appropriately tagged using HTML5's Data Attributes [10] containing its number of element inside the grid (data-screennumber) and the row which they belong to (data-rownumber). These values are embedded in the html elements that contain the screens generated at runtime when the Screen Linking Mode is activated.

We are also using the CSS absolute positioning for the SVG arrow elements inside them to be able to place them exactly in place but due to the nature of this positioning, we have to position the box containing the arrow element by providing a top and left offset in reference to the parent container.

However, first we need to know the distance between the two points we want to connect in

order to define the vector that will serve as a line for the arrow, to later be able to calculate the full size of the container that will host it.

To do this, we establish the absolute value of the difference in rows of the elements that are connected, and we add one in order to obtain the full height of the container. The width of the same will be twice the screen width so that we will always work with a two screens width container with variable height. We will always place this SVG container on the first element of the appropriate row.

This will be determined by which element, source or target, is above the other one, fact that will determine the row in which we will position this container due to the fact that the arrow element cannot overflow its SVG container on the top part, or it will not be displayed.

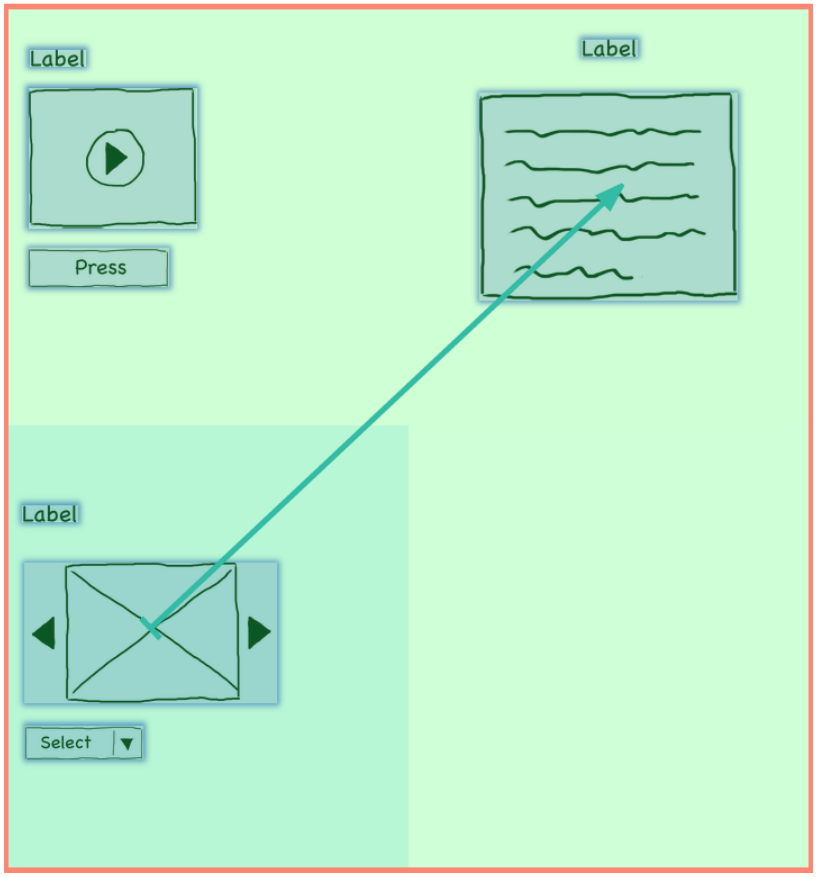

Figure 4.8: Linking example with the SVG final size

Once we have decided which one is above, we apply the appropriate top offset to the destination element, adding the row difference formerly mentioned multiplied by the screen height and also taking into account the offsets for centering the tip of the arrow in the center of the destination screen.

In the case of the source element, the procedure is the same, and instead of adding the offset of screens we have to take into account the original widget's offset as well as the offset to

center the source of the arrow element in its center.

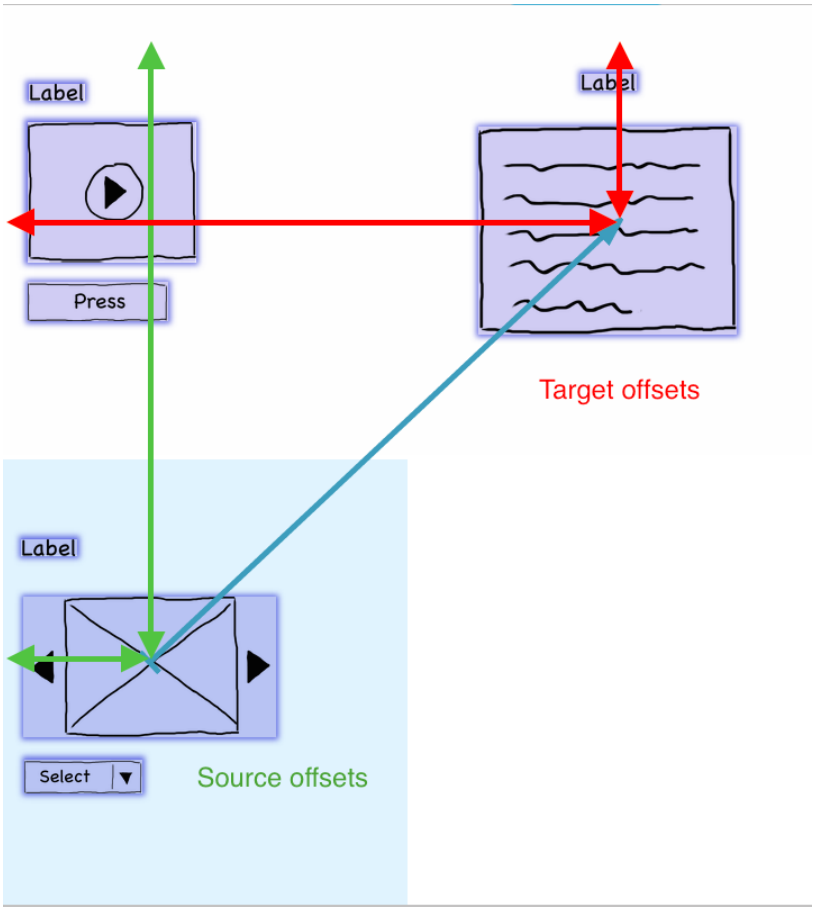

Figure 4.9: Linking example with the calculated offsets for the arrow element

On behalf of the horizontal positioning, it is a really simple matter of checking whether the element, source or target, is in the right or left position in our grid. Only in the case of being in the right column, we would add one screen's width offset to position the tip or source of the arrow in the appropriate column, and then refine the positioning with the element's current offset if it is the case of the source widget or half a screen's width in the case of being the destination screen.

5 Conclusion

5.1 Discussion

In this thesis, we have explored several possibilities of ways of interacting with a prototyping tool. We have taken a look at what is already possible, and what could be improved. We also presented IxSketch, a proof of concept prototype tool that facilitates the development of application prototypes that feeds from this exploration and analysis.

The application attempts to be a bridge between the paper prototyping and the digital prototyping, taking the best of both scenarios.

We have brought sketching interactivity through the global key concept of 'Everything being contained in the sketch' as well as the sub-concepts that support it. Future improvements in the linking of the interfaces, and also new ways of creating and manipulating the interfaces that have been described in previous sections, would allow this proof of concept tool to grow into an interesting and functional tool.

In addition, we have provided a way for users to select between a sketch-like definition, and a high fidelity version of their interface components, allowing the transition from those two stages in a really simple way. In a potential future work, this mode could be extended by the use of a backend part. That would enable us to create custom themes for our widgets, or expanding the palette of widgets available to build an interface. This in turn, would expand the variety of devices and environments that could be represented.

Finally, we have also explored the possibilities of collaboration between users implicated in the development of a prototype through our layer system, and the annotation mode. This permits users to have a way of discussing about decisions taken on the same prototype, sharing their opinions with other teammates or experimenting with different layouts, and elements in an easy way. Again, this goal would be really empowered by the use of a backend component, which would allow the remote collaboration, turning this feature into a strong plus for

our tool.

5.2 Future Work

A user study would be the first step that should be taken in order to extend this proof of concept tool that has been developed along with the thesis. It would bring important and necessary feedback for this tool to grow into a functional product that could solve the issues that common users of this kind of applications encounter on a daily basis.

A potential user study could evaluate the ease of use of the prototype, proposing our users to build a simple prototype in a short span of time, forcing them to face all aspects of the application without previous knowledge of how to use it, testing this way the ease of use, as well as the intuitiveness of the same.

After that, we would ask them to grade the level of difficulty as well as the degree of comfortability they felt when using the tools the application provides. In addition, there would be open text areas for them to explain how they would have expected the tool to behave, or react to certain kinds of interaction. Also, they could suggest ways in which we could approach certain situations, like for instance, connecting two screens together. With these two aspects covered, we would be able to improve significantly the way we approach the interaction between the user and the prototype.

Another part of the user study that would certainly help this tool to become a final product, would be one that looked deeper into collaboration. We could propose our users to cooperate in pairs to develop this same prototype they did in the first part. We would ask them later, how they would rate their collaborating experience in terms of usefulness and if they considered that they detected more or less issues while working together rather than in the previous case when constructing the prototype alone. Finally, we would ask them again for suggestions to refine and empower this collaboration mode.

Another clear way in which this proof of concept would grow in terms of providing more tools and flexibility would be by adding a server side component. A backend could enable a lot of interesting features, such as the already discussed possibility of custom theming of our prototypes, in order to extend the number of devices a prototype can cover, but also, allowing the user to add his own widgets, thus expanding the kind of experiences a prototype can recreate.

This backend could also provide a remote live session concept, in which our users could interact remotely with the same prototype, removing the limitation of having to be in the same physical space, which could turn out to be a great feature for discussing and revising prototypes while being in a videoconference for example, if we place ourselves in a job-like environment.

On behalf of the research and exploration future work, there are several paths that can be followed. On the one hand, further exploration could be done when it comes to interaction techniques, broadening the knowledge about how to improve the approach we have taken towards how the user can use our prototype. This in turn would make it more usable, while bringing new ways of allowing the user to express through our application. New ways of directly sketching the layout of the application, which would be recognized and converted

into a digital functioning prototype would also expand the possibilities of interfaces that could be represented as well as giving the user a more direct way of creating a prototype.

On the other hand, and also with the sketching interactivity in mind, we could explore ways of translating sketching inputs, such as the ones discussed in the design process, or just simple traces, into more elaborate properties of interfaces, like animation types, transition easing, and so on. That would allow the user to fully develop a prototype solely relying in his manual input, removing completely the technological barrier, and also making the development quick and easy for any kind of user.

In summary, IxSketch is a proof of concept that opens a variety of possibilities to explore regarding sketching interactivity, as well as providing a toolset to develop a digital prototype. Further investigation would help it become a more solid tool, and would bring an interesting knowledge for others to build on top as well opening the possibility of achieving a fully functional tool that would help research in terms of interactivity and collaboration as well.

List of Figures

Acknowledgements

I would like to thank:

Prof. Dr. Moira Norrie, head of the GlobIS research group, for allowing me to write this thesis.

Dr. Michael Nebeling for his support, his ideas and his reviews.

Christoph Zimmerli, for his assistance.

Bibliography

[1] Saul Greenberg, Sheelagh Carbondale, Nicolai Marquardt, and Bill Buxton. *Sketching User Experiences: The Workbook*. Morgan Kaufmann, 2012.

[2] Rubaiat Habib, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. Kitty: Sketching dynamic and interactive illustrations. 2014.

[3] Rubaiat Habib, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice. Draco: Bringing life to illustrations with kinetic textures. 2014.

[4] Shah Rukh Humayoun, Steffen Hess, Felix Kiefer, and Achim Ebert. i2me: A framework for building interactive mockups. 2013.

[5] Dietrich Kammer, Deborah Schmidt, Mandy Keck, and Rainer Groh. Developing mobile interface metaphors and gestures. 2013.

[6] Mozilla Developer Network. Using css flexible boxes — mdn.

[7] Ugo Braga Sangiorgi. Electronic sketching on a multi-platform context: A pilot study with developers. 2014.

[8] Carolyn Snyder. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces (Interactive Technologies)*. Morgan Kaufmann, 2003.

[9] W3C. Html canvas 2d context, 2014.

[10] W3C. A vocabulary and associated apis for html and xhtml, 2014.

[11] Wikipedia. Cascading style sheets — wikipedia, the free encyclopedia.