Touchable Code: Building a multi-touch IDE

Carina Neves Fonseca

Thesis to obtain the Master of Science Degree in
Telecommunications and Informatics Engineering

Supervisors: Prof. Sandra Pereira Gama
Tiago Simões

Examination Committee

Chairperson: Prof. Helena Isabel De Jesus Galhardas
Supervisor: Prof. Sandra Pereira Gama
Member of the Committee: Prof. Hugo Miguel Aleixo Albuquerque Nicolau

September 2017
Acknowledgments

My first acknowledgment goes to my project advisor Tiago Simões. I am really grateful for having the pleasure to work with such a wise person. Thank you for the incessant motivation in the most difficult moments and for all the help along this sinuous path. I also thank so much to my advisor Professor Sandra Gama and Eng. Sara Cruz for all support and their seamless help in keeping me motivated.

I would like to express my profound gratitude to my family for their unconditional love and support.

Last but not least, to all people that helped me grow as a person and were always there for me during the good and bad times in my life.

To each and every one of you - thank you.
Abstract

OutSystems is a rapid desktop application delivery platform that enables the development enterprise-grade web and mobile apps. One of the main tools in the platform is an IDE to develop applications, called Service Studio, currently only available as a desktop version. To follow the recent adoption of mobile devices, this dissertation presents an iPad version of the Service Studio, called OutSystems Touch whose goal is to allow OutSystems users to develop and test their applications' user interface continuously on a mobile device, in a user-friendly way. To accomplish this goal, a study of similar software and applications for business purposes is presented as well all steps taken for the development of OutSystems Touch. At the end, is presented the final evaluation of the developed product, OutSystems Touch, and is also presented a comparison between the two versions (desktop version and iPad version).

Keywords

OutSystems; IDE; Service Studio; touchable environment; user-friendly interface; technology.
Resumo

A OutSystems oferece uma plataforma que permite a rápida entrega de aplicações móveis e web para um nível empresarial. Uma das principais ferramentas da plataforma é um IDE para desenvolver estas aplicações, chamado Service Studio, atualmente disponível apenas numa versão para desktop. De forma a seguir a recente e crescente adoção dos dispositivos móveis, esta dissertação apresenta uma versão iPad do Service Studio, denominada OutSystems Touch, cujo objetivo é permitir aos utilizadores da OutSystems desenvolverem e testarem a sua interface de utilizador das aplicações diretamente no dispositivo final em ambiente touch, de maneira fácil. Para atingir este objetivo, é apresentado um estudo de software e aplicações semelhantes para fins empresariais, bem como todas as etapas realizadas para o desenvolvimento da OutSystems Touch. Por fim, é apresentada a avaliação final do produto desenvolvido, OutSystems Touch, e é também apresentada uma comparação entre as duas versões (versão desktop e versão iPad).

Palavras Chave

OutSystems; IDE; Service Studio; ambiente touch; interface do utilizador; tecnologia.
Contents

1 Introduction 1
  1.1 Goals ................................................................. 1
  1.2 Document Organization ........................................... 2

2 Background 3
  2.1 The OutSystems Platform ........................................ 5

3 Related Work 7
  3.1 The rise of the mobile and low-code development on mobile ... 7
    3.1.1 TouchDevelop ................................................... 7
    3.1.2 ScriptKit ......................................................... 8
    3.1.3 Pythonista ....................................................... 8
    3.1.4 Codea ............................................................... 9
    3.1.5 Swift Playgrounds ............................................... 10
    3.1.6 Comparative summary of low-code development apps ........ 11
  3.2 Touchable Applications for work .................................. 12
    3.2.1 Microsoft Office ................................................ 12
    3.2.2 Google Docs ..................................................... 13
    3.2.3 Photoshop ....................................................... 13
    3.2.4 AppCooekr ....................................................... 14
  3.3 Final Remarks / Conclusion ....................................... 15

4 OutSystems Touch 16
  4.1 Architecture ...................................................... 16
  4.2 Technology and Proof of Concept ................................ 19

5 Preliminary Work 21
  5.1 User Research ..................................................... 21
  5.2 Learning OutSystems ............................................... 24
  5.3 Target Application ................................................ 25
  5.4 Evaluate the current state of OutSystems ....................... 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5</td>
<td>Touch Challenges</td>
<td>31</td>
</tr>
<tr>
<td>5.6</td>
<td>Prototype</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>Implementation</td>
<td>35</td>
</tr>
<tr>
<td>6.1</td>
<td>Defining the Data Model and Creating Widgets</td>
<td>35</td>
</tr>
<tr>
<td>6.2</td>
<td>Develop an Interpreter to Save and Load Screens</td>
<td>39</td>
</tr>
<tr>
<td>6.3</td>
<td>Allow Screen Edition</td>
<td>40</td>
</tr>
<tr>
<td>6.4</td>
<td>Refactoring and Storage</td>
<td>44</td>
</tr>
<tr>
<td>6.5</td>
<td>Formative Evaluation</td>
<td>46</td>
</tr>
<tr>
<td>6.6</td>
<td>Prototype Iteration</td>
<td>47</td>
</tr>
<tr>
<td>7</td>
<td>Evaluation</td>
<td>59</td>
</tr>
<tr>
<td>7.1</td>
<td>Protocol</td>
<td>59</td>
</tr>
<tr>
<td>7.2</td>
<td>Results</td>
<td>60</td>
</tr>
<tr>
<td>7.3</td>
<td>Discussion of the evaluation results</td>
<td>61</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion and Future Work</td>
<td>67</td>
</tr>
<tr>
<td>A</td>
<td>Questionnaire to calculate SUS</td>
<td>70</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Result of forrester’s evaluation of top vendors in low-code development platform [1] . . . 4
3.1 Screen with the expression editors. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
3.6 Some changes of Microsoft Office [2] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
4.1 Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
4.2 Service Studio allows developing Processes, Interface, Logic and Data. OutSystems
Touch will only focus on the Interface. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
4.3 Model of OutSystems Touch . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
4.4 Proof of Concept . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
5.3 Screenshot of the usability test script . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
5.4 Ranking of SUS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
5.5 Screenshot of the tables of each task . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
6.1 WidgetKindOnStartDrag client action . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
6.2 Example of the generated screen JSON . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
6.3 JSON screen load on the left and correspondent result of the load on the right . . . . . 39
6.4 Client action executed when users do load of JSON . . . . . . . . . . . . . . . . . . . . . . 40
6.5 OnWidgetSelect client action . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
6.6 Example of the widget selected with highlight . . . . . . . . . . . . . . . . . . . . . . . . . 41
6.7 Widget Structure on Service Studio . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
6.8 Popup icons and icon properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
6.9 ChangeProperties client action . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
6.10 Hiding properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
6.11 Adding a new color picker . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
6.12 Changing the upload photos mode . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
6.13 OutSystems Touch data model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
List of Tables

3.1 Synthesis of existing software to make code on mobile devices .................. 11
5.1 Records of the usability tests performed to Service Studio ....................... 29
5.2 Records of the usability tests performed to Service Studio ....................... 30
5.3 Records of the usability tests to the low-fidelity prototype ....................... 33
7.1 Records of the usability tests performed to final version of OutSystems Touch 60
Technology is evolving at an incredible speed and that is changing the way we interact with it. Mobile devices are becoming the prevailing computing platform for most people and consequently, the number of applications available on the market is increasing.

Traditionally, software applications for mobile devices such as tablets or smartphones are developed on the desktop and this poses a challenge for developers. Unlike web apps, mobile apps need to be developed on a device (computer) and tested in a completely different one (tablet or smartphone). This makes the development process longer, harder and more error-prone. Even though there are already some tools that allow the development of applications, games or prototypes using a touch environment without a traditional keyboard, all of them were developed for individual use or for specific use cases and there is no development tool available for mobile devices in the enterprise market.

1.1 Goals

OutSystems is a rapid application delivery platform that allows the development of enterprise-grade web and mobile apps faster than traditional technologies. However, as other development platforms, it only works for desktop. In order to keep up with this evolution, the main goal of this project is to allow OutSystems users to develop and test their interfaces of the apps continuously on a mobile device, building an iPad version of the OutSystems Service Studio. This will have several benefits such as:

1. By designing the platform for a smaller screen with larger touch targets, the resulting experience will be simpler and easier to learn.

2. Touch and desktop technologies are merging into a single one as time goes by, with devices such
as iPad Pro and Microsoft Surface, it is important for companies such as OutSystems to start designing experiences that go beyond the keyboard and mouse.

3. By developing directly in a mobile device, the resulting interfaces of the apps will be more suited to the target form factor.

4. Developers will get immediate end user feedback, in the field, and react fast by tweaking the interfaces of the apps in real time.

Even though mobile devices are typically equipped with powerful batteries, graphic processors and high-resolution screens, high productivity applications are still a challenge to develop. Moreover, when developing a mobile version of an existing platform, it is important to keep the same patterns given that people are already familiar with them. OutSystems has many features, and making them available in a touchable environment was a big challenge.

1.2 Document Organization

This document is composed of 8 sections. The current one states the motivation for the development of this project, as well as the expected goals upon its completion. Chapter 2, Background, presents a contextualization of low-code platforms and OutSystems. The third, Related Work, presents some already existing tools of low-code platforms for touch. Chapter 3 also presents other touchable applications environment for work with high productivity or desktop tools with mobile versions. The fourth chapter, OutSystems Touch, presents the architecture of the solution, as well as the technology and proof of concept made. The Chapter 5 describe the preliminary work did before implementation that is also explained in Chapter 6. Chapter 7, Evaluation, presents the evaluation protocol used of the OutSystems Touch as well as the discussion of the results and their conclusions. Finally, Chapter 8, presents the conclusion and the future work of this dissertation.
Background

Traditional hand-coding methods for software development are too slow to keep up with the rising demands of consumers. As a result, many companies are innovating and providing users with an easy way to develop applications with low-code platforms. Low-code platforms are defined by Forrester\(^1\) as platforms that enable rapid delivery of business applications with a minimum of hand-coding and minimal upfront investment in setup, training and deployment\(^1\). End users can change apps quickly by modifying logic flows and using new User Interface (UI) elements, and developers can also take advantage of new features without the need to invest in infrastructure and software.

Forrester evaluated the strengths and weaknesses of top vendors in low-code development platforms with a set of evaluation criteria and grouped them into three high-level buckets: current offering, strategy and market presence. The result can be seen in figure 2.1.

Some of the platforms that enable the development of applications without extensive programming are OutSystems\(^2\), Mendix\(^3\), Salesforce\(^4\), Appian\(^5\), AgilePoint\(^6\), K2\(^7\), Bizagi\(^8\), Caspio\(^9\), ServiceNow\(^10\), MatsSoft\(^11\), Nintex\(^12\), MicroPact\(^13\), QuickBase\(^14\) and MIOSoft\(^15\).

OutSystems is the leading low-code platform and it distinguishes from competition for the user experience and advanced features it provides\(^1\). However, Google recently launched Google App Maker

\(^1\)https://forrester.com/marketing/about/about-us.html
\(^2\)https://outsystems.com/
\(^3\)https://mendix.com/
\(^4\)https://salesforce.com/
\(^5\)http://apian.com/
\(^6\)http://agilepoint.com/
\(^7\)https://k2.com/
\(^8\)http://bizagi.com/
\(^9\)https://caspio.com/
\(^10\)https://servicenow.com/
\(^11\)http://matsoft.com/
\(^12\)https://nintex.com/
\(^13\)https://micropact.com/
\(^14\)http://quickbase.com/
\(^15\)https://miosoft.com/
which can change the competitive landscape, because it is an application accessible to everyone (developers or not), so it will be easy to use.

Google App Maker\textsuperscript{16} is a low-code application development tool that people inside organizations can use to develop custom applications based on their business needs. It is possible to create a new data or import the data users already have and store in user’s drive or on Google Cloud Platform. It has a drag-and-drop UI editor that complies with Google’s Material design principles with templates but it is possible to customize with scripts, as well as HTML, CSS, JavaScript, and JQuery content and a click to publish an application without servers, administrators or provisioning. App Maker integrates with user’s Google accounts, allowing users to integrate their applications with Google Ecosystem (Google Calendar, Google Contacts, Google Maps, among others). Lastly, when the apps are live, it is possible to monitor usage through Google Analytics.

With this set of features, the close connection to Android and material design, and the ease-of-use that is standard in all Google products, Google App Maker will certainly come as a big disruption in the low code market. As a response, OutSystems and the other platforms will need to innovate in order to maintain their status as leaders in the low-code spectrum.

\textsuperscript{16}https://developers.google.com/appmaker/
2.1 The OutSystems Platform

OutSystems\textsuperscript{17} provides a complete solution for developing enterprise-grade mobile and web applications. It has a set of components to visually develop apps and automatically generate their code, integrate them with external custom code (JavaScript, Java, C\#, SQL, CSS, and HTML), manage the application's configuration and lifecycle and monitor the apps' performance. One of the main tools in the platform is Service Studio, an IDE where users can develop applications. It will be the main focus of this thesis.

Service Studio allows users to develop web and mobile applications with high productivity, integrates with existing systems and delivers visually appealing and usable native mobile experiences and responsive web apps.

It allows users to create interfaces by providing a set of over 140 designer-approved patterns and controls for an interface with great look and User Experience (UX) by default. More proficient developers will also be able to use custom code in HTML, CSS and Javascript whenever needed (see figure 2.3).
It is also possible to model custom business logic, as well as to integrate with 36 different services and applications like Google Drive, Facebook, Paypal, Dropbox, etc., that can incorporate in the visual data workflow (see figure 2.4).

Figure 2.4: Screenshot of the business logic of the Service Studio

Finally, it allows users to create a visual data model and later display the data in the UI and change it in the business logic (see figure 2.5).

Figure 2.5: Screenshot of the model of data of the Service Studio

The generated apps are secure by default, they run fast regardless of the number of users and data volume and can be deployed and monitored in cloud or on-premises environments.

https://outsystems.com/platform/
Technology is changing the world and access to smartphones by all levels of society has multiplied the number of available mobile applications. As a result, mobile devices are becoming the prevalent computing platform for most people and the development of mobile applications has multiplied. New platforms have emerged to give developers greater speed in development and testing, and allow them to take advantage of the mobile development environment. This way, developers are able to get immediate feedback from customers on products and services and react fast. Overtime, the need to follow this progress has also increased and nowadays, there are several low-code platforms that enable the development of applications or prototypes directly on a mobile device (section 3.1) and a great number of existing professional or high-productivity desktop applications for work have already been adapted to use the touchscreen as the only input device (section 3.2).

### 3.1 The rise of the mobile and low-code development on mobile

In this section, we present some existing mobile tools that allow the development of applications or prototypes using low-code in an entirely touch environment, such as TouchDevelop developed by Microsoft, Scriptkit, Pythonista, Codea and Swift Playground.

#### 3.1.1 TouchDevelop

TouchDevelop by Microsoft\(^1\) is a mobile development environment and a visual programming language that allows users to write applications, called “scripts”, directly on the smartphone, without a PC or a traditional keyboard. For editing scripts, this environment offers a semi-structured code editor that operates on statements making it impossible to make a syntactically incorrect program (see figure 3.1) and the scripts can be shared with other people. It provides building primitives that make it easy to

---

\(^1\)https://microsoft.com
access the data available on a mobile device like camera, touch, accelerometer, compass, gyroscope, contacts, calendar, email, pictures, and songs [3]. The state of the program is automatically distributed between mobile clients and the cloud, with automatic synchronization of data and execution between clients and cloud, freeing the programmer from having to be concerned with (or even having to know) the underlying technical details [4].

Figure 3.1: Screen with the expression editors.

3.1.2 Scriptkit

Scriptkit\(^2\) is a touchable programming environment for building simple mobile prototypes on iPad using native iOS UI components and social media APIs, available via a drag and drop interface (see figure 3.2). ScriptKit is designed for touch and is optimized to help users go from idea to prototype in the shortest amount of time possible. It comes with several examples to help users get started. Scriptkit is built using C4\(^3\) which is an open-source creative coding framework.

3.1.3 Pythonista

Pythonista is a complete development environment for writing Python scripts on iPad or iPhone. Examples are included — from games and animations to plotting, image manipulation, custom user interfaces, and automation scripts. In addition to the standard library, Pythonista provides support for interacting with native iOS features, like contacts, reminders, photos, location data, and more. Pythonista’s code editor provides syntax highlighting, smart code completion, and an extended on-screen keyboard (see figure 3.3). It contains an outline view that allows users to navigate their scripts, as well as multiple tabs to simplify working on larger projects. Pythonista comes with easy-to-use libraries for 2D graphics,

\(^2\)http://scriptk.it/
\(^3\)http://c4ios.com
\(^4\)http://scriptk.it/
multi-touch and sound effects. When to quickly try a snippet of code or perform an ad-hoc calculation, the interactive prompt is easily accessed with just a swipe. Users also use it to inspect the results of their scripts interactively and they can create user interfaces for scripts without any code, using the integrated UI editor\(^5\).

\[\text{(a)} \text{ Smart code completion in Pythonista.} \quad \text{(b)} \text{ Some features to programme in Pythonista.} \]

\[\text{Figure 3.3: Pythonista} \]

### 3.1.4 Codea

Codea\(^5\) allows the creation of games and simulations or visual ideas users may have for iPad. It simplifies the development of interactive applications that make use of iPad features like multi-touch and

\(^5\)https://ivankahl.wordpress.com/2015/05/23/app-review-pythonista/
\(^6\)https://codea.io/
the accelerometer. Codea is designed to let users touch the code and change it through contextual visual editors. It is built using the Lua\textsuperscript{7} programming language.

![Figure 3.4: Programming in Codea\textsuperscript{8}]

### 3.1.5 Swift Playgrounds

Swift Playgrounds\textsuperscript{9} is an iPad app to teach children how to code in Swift with the goal of inspiring the next generation to learn how to code. It assumes users have no prior programming knowledge.

Swift Playgrounds has a set of lessons, challenges and interactive tutorials where users control a character in a 3D world using code (see figure 3.5).

It was designed for touchable environments, allowing the creation of programs using only taps on the iPad screen, helping children to write correct code, identifying mistakes as they type and offering suggestions to help correct errors. This includes QuickType (an Apple’s predictive keyboard that learns with users use) for code with a shortcut bar that presents the most likely next commands or values based on context; for example, tapping a number shows an in-place a popover keypad and touching a color displays a color picker; common code snippets such as loops, structures, and switch statements can be dragged out of a library to create new code with less typing. When the user taps the button to run the code, if it fails, the hero looks disappointed so that children understand that they made a mistake. At that point, they can repeat or see a more detailed explanation\textsuperscript{10}.

\textsuperscript{7}https://lua.org/
\textsuperscript{8}https://codea.io/
\textsuperscript{9}http://apple.com/swift/playgrounds/
\textsuperscript{10}https://developer.apple.com/swift/playgrounds/
3.1.6 Comparative summary of low-code development apps

The table 3.1 allows to compare the software presented above based on their main characteristics and identified patterns.

<table>
<thead>
<tr>
<th>TouchDevelop</th>
<th>Scriptkit</th>
<th>Pythonista</th>
<th>Codea</th>
<th>Swift Playground</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>mobile</td>
<td>mobile</td>
<td>mobile</td>
<td>mobile</td>
</tr>
<tr>
<td>Operating System</td>
<td>iOS (iPad)</td>
<td>iOS (iPad and iPhone)</td>
<td>iOS (iPad)</td>
<td>iOS (iPad)</td>
</tr>
<tr>
<td>Delivered Apps</td>
<td>Application prototypes</td>
<td>Anything (IDE)</td>
<td>- Games Interactive Apps</td>
<td>- Games Learning</td>
</tr>
<tr>
<td>Collaboration</td>
<td>Share scripts and libraries</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Programming Language</td>
<td>New language based on Java and C#</td>
<td>C4 (open source framework)</td>
<td>Python</td>
<td>Lua</td>
</tr>
<tr>
<td>Integrations</td>
<td>-</td>
<td>iOS UI components</td>
<td>iOS UI components</td>
<td>-</td>
</tr>
<tr>
<td>Extra features</td>
<td>- Syntax assistant - Develop and share libraries</td>
<td>Pre-loaded templates Native libraries</td>
<td>-</td>
<td>- Syntax assistant Libraries</td>
</tr>
<tr>
<td>Market</td>
<td>Consumer</td>
<td>Consumer</td>
<td>Consumer</td>
<td>Consumer</td>
</tr>
</tbody>
</table>

Table 3.1: Synthesis of existing software to make code on mobile devices

Although these tools allow users to build applications and prototypes in a touchable environment,
they are typically suited for small applications or tweaking larger apps that are mainly developed on a
computer. They require knowledge of specific programming languages and limited visual editing capa-
bilities. Moreover, they are all targeted for the consumer market.

3.2 Touchable Applications for work

All the software presented above is far from the final goal of the project: developing a visual low-code
platform for business purpose. For that reason and for a complete analysis, it is important to analyze
mobile versions of other complex, high productivity, desktop tools with mobile versions, used in a work
environment, such as Microsoft Office, Google Applications, Adobe touch apps and App Cooker. These
are not programming apps, but their interfaces will be used as benchmark for powerful mobile interfaces.

3.2.1 Microsoft Office

Microsoft Office allows users to read Word documents, to use Excel data and to create and deliver
PowerPoint presentations. With Office 365, users can edit and create new documents on the touch
environment with the same quality as they do on computer. The interface is familiar if you already use
office on PC or Mac and this is important because, in this way, users don’t have new ways of working. No
keyboard or mouse is required to create, edit and format documents and the app is optimized for touch.
An example of this is that selecting text or objects shows relevant commands, the large touch areas on
the ribbon and overlay menus are clean and simple (see figure 3.6), resizing and rotating objects can
be done using touch-friendly handles and users can even use tablet’s features like voice dictation.

(a) Commands are shown when relevant
objects are selected

(b) Modification of the large touch areas.

Figure 3.6: Some changes of Microsoft Office [2]

In addition, it allows users to work simultaneously on the same document or presentation with other
people because it’s seamlessly integrated with Microsoft’s cloud services so that users can access them anywhere, on any device. There’s no Save button, but the AutoSave ensures that changes are saved automatically and frequently and also allows restoring the original document with a feature that tracks changes. Users can decide which ones to accept or decline, as well as add their own comments [2]. Currently, it doesn’t allow sending files to a printer and users can’t access the camera.

3.2.2 Google Docs

Google has responded to the launch of Microsoft Office for iPad. It allows the user to see other document types. The interface in a web browser for tablet is familiar at first glance but there are some differences11. It is easy to create a new document and it’s possible to share it, in an easy way, so that several users can collaborate simultaneously in real time. But the problem is with the extremely limited functionality Google docs offers in terms of document editing. For example, there’s no option to upload photos from your camera, customize fonts, alignment, indentation, bullets and comments. Although users can leave comments, they can’t track changes (see figure 3.7), there’s no autocorrect or word count. It is actually faster to use Google Docs in a mobile browser than using the mobile app, specially for Android12. Like Docs, Sheets is also a bare-bones version of Google’s Web-based app13.

3.2.3 Photoshop

Adobe Photoshop touch app doesn’t replace the desktop version but work with files with up to 1600x1600 pixels and it comes with popular photoshop tools, filters, effects and a finger-friendly touch

---

12http://imore.com/google-docs-iphone-and-ipad-review
13http://laptopmag.com/reviews/apps/google-docs-for-ipad
14https://badpeoplegooddramatv.wordpress.com/2015/03/01/another-experience-with-google-docs/

interface. It is possible to move the work between tablet and desktop through the connection to Adobe’s Creative Cloud but it only works with a plug-in\(^{15}\). It is also possible to share images via email and Facebook but it is not possible to use other popular APIs, like Twitter. The interface is familiar and, one more time, it is important for to minimize the learning curve. Even so, the app contains interactive tutorials to help users getting used to the interface. The selection tools are limited by the accuracy of the finger tip, and the icons and menus are large and finger-friendly\(^{16}\). There are not many options for text edition yet but there are new features like Camera Layers which allow users to take a picture on the iPad, view it in the existing layers and access more image sources such as Facebook and Google image search (see figure 3.8). For simple tasks and emergency issues, it works efficiently. For more complex images, it does not have the effectiveness required for certain tasks, so if users plan to use the app for extensive periods of drawing or painting, a stylus is recommended.

![Figure 3.8: Image search import from Google\(^{17}\)](image)

### 3.2.4 AppCooker

AppCooker\(^{18}\) is another mobile application that provides users to plan every aspect of the app. Thus, the application is divided into six areas: idea, icon, model, detailed information and business plan. There is a space for ideas for helping clarify them and thinking about interface style and definition statement. The incorporated app icon helps users to solidify and to draft an idea for an app icon and to use freehand drawing tool, to add vector shapes, colors, or images to create the icon (see figure 15). The mockup editor allows to use many features, like to create workflows, to take a look in full screen, to use widgets from device like to import images from photo library, to create own drawings using the hand drawing tool, among others\(^{19}\).

\(^{15}\)https://blogdoiphone.com/2012/02/testamos-o-photoshop-touch-para-ipad-a-nova-ferramenta-grafica-da-adobe/

\(^{16}\)http://pcadvisor.co.uk/review/android-tablet-apps/adobe-photoshop-touch-app-review-3320900/


\(^{18}\)http://appcooker.com/

\(^{19}\)http://ipad.appstorm.net/reviews/productivity/app-cooker-an-entrepreneurs-best-friend/
3.3 Final Remarks / Conclusion

While web-based versions of these tools can lead to a higher user satisfaction, the mobile versions lack many features, mostly because the majority of mobile users work with their devices using their fingers. This has several implications for mobile app designers, since they can’t rely on hover events, right-clicks and they have to design for smaller screens with less precision. These limitations end up interfering with the familiar interface. Another problem about mobile devices is that their hardware is not yet ready to run many applications at the same time, and the more complex and rich an application is, the higher the level of resources it will require from the mobile phone, and the slower it will become. As a result of this, a desktop app with a good user experience may become hard to use when developed for mobile. All these frustrations tend to change with the evolutions of the user experience area and the evolution of quality of mobile devices as well.

---

20 http://appcooker.com
This section presents the architecture of the proposed solution as well as the technology used for its development and its proof of concept. OutSystems Touch is the name adopted for the proposed solution as it extends the OutSystems functionality to touch environments.

4.1 Architecture

To follow the main goal of this project, allow OutSystems users to develop and test their applications continuously on a mobile device, the following architecture has been proposed.

![Architecture Diagram]

**Figure 4.1: Architecture**

As depicted in Figure 4.1, the OutSystems platform IDE, called Service Studio, uses an Model-View-Controller (MVC) architecture that separates the implementation into three distinct layers and is a standard for interactive applications:
• Model - responsible for reading, writing and validating the application model, that contains a representation of its interface, business logic and data;

• View - responsible for displaying the application model to the user;

• Controller - responsible for processing the user’s requests and change the application model accordingly.

OutSystems Touch Scope

Service Studio allows users to develop complete web and mobile applications, as it allows the creation of processes, interfaces, business logic and the data model. The interface of Service Studio has tabs to develop each one of this parts (figure 4.2).

Due to time restrictions, the scope of this project, OutSystems Touch, is the user experience associated with the development of interfaces directly on the final device. This means that OutSystems Touch only allows users to develop the application interface. Processes, logic and the data components are part of the future work as will be explained later on.

The architecture of the proposed solution also follows the MVP model, as OutSystems Touch was developed using Service Studio and the applications it generates follow this same model.

The existing desktop version is too rich and complete, containing a large number of features. As such, the application model used in OutSystems Touch will also be a simplified version of the one used by Service Studio.

For example, OutSystems allows users to use the data previously created on the user interface and change it in the business logic of the application. The binding of UI to data and expressions is not going to be covered in this prototype of the OutSystems Touch, because as mentioned before, the scope of this project is only to develop the user interface, so the business logic and data do not exist and, consequently, can not be used in the construction of the interface. Another example is that in
OutSystems, an application is stored in an OML (OutSystems Model Language) file which are composed mainly by large and complex XML structures. The model in the proposed solution will be simpler, more suited to the specific use cases and represented in a JSON file due to JSON being much more human-readable compared to something like XML. For web APIs it also makes sense because the overhead is much smaller, so we decided that JSON is the most appropriate format for storing and consuming information in this case.

Finally, the OutSystems Touch could be integrated with Service Studio, via a plugin, so that the new model can be imported into the platform and make use of all its capabilities. The plugin will consume the JSON file and instrument Service Studio to generate and publish the app in the cloud, where it will be accessible to end users, as well as produce an OML file that can be stored and manipulated in a standard way. This integration was discussed with the OutSystems Engineering team and it was concluded that, although being feasible given the structure of the JSON that will be implemented, the cost of its execution made no sense in a usability project like this one, so it was outside of the scope of this thesis.

Figure 4.3 shows in more detail the architecture of the model of the OutSystems Touch, represented by a class diagram:

- An application contains screens;
- The screens contain widgets that are fragments of UI components;
- Widgets are buttons, texts, icons, etc. and define how the data is shown to end user (lists, graphs, images) with their properties;
- Properties can be colors, sizes, names, among others.

Figure 4.3: Model of OutSystems Touch
4.2 Technology and Proof of Concept

To implement the solution presented above, it was necessary to decide which technology to use. OutSystems Touch is a mobile application developed for a specific platform (Apple’s iOS and Google’s Android) and for these type of applications, the following development technologies are traditionally used:

- React Native: This technology is a framework for building mobile native apps using Javascript code library developed by Facebook and Instagram in 2013. React Native Compiles to native code for both iOS and Android. But working across separate iOS and Android codebases is challenging.

- iOS Native: To develop a native application for iOS, it's necessary to have a Mac running the latest version of Xcode. Xcode includes all the features users need to design, develop, and debug an app. It also contains the iOS SDK (Software Development Kit), which extends Xcode to include the tools, compilers, and frameworks you need specifically for iOS development.

- Android Native: Android applications are scripted in Java language with the help of a rich set of libraries. Google has released an IDE, the official development tool for Android applications called Android Studio and supported on Windows, Mac and Linux. It also contains the Android SDK and documentation is structured and easy to find.

However, there are other non-traditional solutions, and given the context of this thesis, it was decided that it would make sense to evaluate the possibility of using the OutSystems programming language to develop OutSystems Touch, which meets the main objective, to develop and test applications directly on mobile devices.

Can OutSystems Service Studio be used to create OutSystems Touch?

OutSystems Service Studio is optimized towards the creation of applications that deal with structured data (approval workflows, digital operation applications, among others). It is not as well suited to create creativity and productivity applications that use drag and drop. So there were some interactions that needed be tested before starting the development. The interactions that have been tested are the following:

1. Define drag areas and drop areas;

2. Order elements in drop areas;

3. Clone an element upon drop.
OutSystems Service Studio does not contain draggable elements/widgets, which constituted a big obstacle in the development of an application that allows the drag and drop of the elements. To deal with this limitation a new component was developed to allow the implementation of drag and drop in applications. Also, at this stage the decision to simplify the development of interface layouts was made.

OutSystems Service Studio used the standard HTML document layout, where elements are stacked to the right or bottom of previous elements, and where they can be nested. In OutSystems Touch, due to time constraints and also because it would be easier for users, we adopted the more direct approach of just setting the positioning of elements as absolute. The tradeoff of this decision is that when users are creating an interface they are doing it just for the current screen size (in this case a tablet in landscape mode). To improve on this, multi-device targeting is something that should be explored further ahead.

This way, it was possible to test the aforementioned interactions successfully using Service Studio, as can be seen in figure 4.4, despite all the difficulties found and also mentioned above.

This component can now be used to develop other applications, making OutSystems a possibility to develop this kind of applications.

It is important to mention that Service Studio is a low-code platform, but because it was not optimized for the development of these type of applications, it was necessary to develop a new component for the platform, producing in this way much code that would not be expected in the development of applications using OutSystems technology.

Figure 4.4: Proof of Concept
This chapter outlines the work done prior to the implementation of OutSystems Touch.

It started with user research, which counted on interviews to understand the users, defining personas and scenarios. Then it continued with learning the technology that would be used for the development of OutSystems Touch, OutSystems, as well as its evaluation with usability tests. This resulted in understanding the real needs of the users, leading the development to a familiar result from the existing version, with the possibility of improving usability and eliminating some of the more common current user errors. In this chapter, it is also defined the target application to be used later in all the usability tests and formative evaluations, in order to maintain coherence. Finally, a proposal of the developed prototype is presented, as well as a formative evaluation and consequent iteration.

5.1 User Research

In order to develop a solution that meets the goals: allow OutSystems users to develop and test their apps continuously on a mobile device, it was fundamental to know users, to understand how they think and express opinions, and synthesise all this information into personas. Personas are fictitious people that represent a group of users. They impersonate a given user profile (stereotype) and have a background story, role, goals, attitudes, activities, context, etc., supported by data [5].

For this, it was decided to do a set of interviews. First, some categories were defined, to help structure the results. The categories are the following:

1. OutSystems’ developers who focus on user experience.

2. People that use touchable applications for work, such as those mentioned in Chapter 3.

All the interviews were recorded with the intention of not losing the momentum with note taking. The goal was to have a fluent and informal conversation of around 20 minutes, giving the interviewee some
context, and getting as much information as possible, including the answer to the following points:

**Both Categories:**

- Demographic data, academic background and occupation;
- Usage of any touchable work apps, like Google Docs;
- Their experience of using these applications (if it was difficult, if they have the same experience as desktop).

**Only for OutSystems’ developers:**

- Five features most used at Service Studio;
- Five features least used at Service Studio;
- Something that does not exist in Service Studio that users find necessary;
- Their thoughts about if a tablet version would minimize the time between the application development and the end-user test.

Eleven people were interviewed, ages between 23-38 years. The interviewees are attending or have already attended Computer, Telecommunication or Electrical Engineering degrees.

Regarding the first categories, OutSystems’ developers or testers, all respondents presented the same less-used features in Service Studio, and everyone thought it would be good to **reduce the need to produce CSS** because it would minimize the complexity and time in the application interface construction.

Regarding the use of work applications, almost all of them avoid using them because they are familiar with desktop applications and when they try to use the same application on tablets, they are faced with a great lack of features and they can’t do everything they could on the desktop, which leads to frustration from the users. Overall, they feel the applications aren’t ready to be used in the touch environment and are not adequate to produce more complex content. However, although writing in touchable devices is considered annoying, applications like google docs were well rated by interviewees as applications that work great for reviewing documents or making small changes.

**Personas**

The data obtained allowed to define three different personas:

1. Joseph is 30 years old and he is a computer engineer. He’s an OutSystems’ developer and he uses Service Studio every day on his work to build applications. He would like to write less CSS,
because he loses a lot of time defining properties and customizing elements such as buttons. He would love to work in a tablet and if he could, he would do everything in this environment because he has had positive experiences with some applications designed for touchable devices, such as AppCooker.

2. Gaspar is 36 years old and he is a computer engineer. He’s an OutSystems’ developer and user researcher. When he is building applications with Service Studio he feels the need to test the result on a device in real time. For example when gathering feedback from usability tests, it would be useful to correct some errors found. Gaspar avoids working in a touch environment due to the negative experiences with the common work applications he used on tablet or smartphone.

3. Emilly is 22 years old and she is a student of Telecommunication and Computer Engineering. She has to take the computer to school every day and she doesn’t like it because it’s heavy and uncomfortable and she doesn’t understand why there aren’t more work tools available on tablet yet. It would be much simpler because “I could follow the practice classes using the tablet and do the essentials, and this would be much simpler because the tablet is easier to learn, since the interfaces are cleaner and simpler”.

Scenarios

The second step is to define scenarios defining when, where and how the story of the personas takes place. The scenario is the narrative that describes how the persona behaves as a sequence of events [5].

1. John Mayer is developing an application for a customer in OutSystems Service Studio, but he is going to travel and still has to adjust some details. The trip is long, so he plans to finish the application on the plane. With the mobile Service Studio version, he wastes no time in testing the small changes in the final device, as he would on desktop. This way, he could take advantage of the mobile interface to make changes in real time and finish everything on time, even without a network connection available.

2. Marta is an Intern at OutSystems and she is using Service Studio for the first time. She is learning and she’s finding the platform confusing, because it has too many features that are not used. With the mobile version of the OutSystems platform, with only the features needed to produce the application interface, Marta’s job is easier. Adding to this, Marta no longer needs to take her computer every day and carry a lot of weight, since she can already work on her tablet that she uses to do pretty much everything else.
5.2 Learning OutSystems

It is important to understand the user needs, as it was done in the previous section. But when you want to develop a new version of an existing product, it is imperative to assume the role of the user of the existing platform, to complement the needs of these users. This allows the production of a consistent and familiar solution to users.

This step was important also because, since OutSystems Touch will be developed using the OutSystems technology, it was important to learn to work with the Service Studio.

OutSystems offers a wide range of courses and documentation to enable users to develop high-quality web and mobile apps. Since I had never used this technology, I took the beginner course entitled “Developing OutSystems Mobile Apps”, with 32 hours. This course addresses the development of mobile applications while learning the basics of OutSystems, supported by a series of discussions, guided instructions and hands-on exercises.

It was difficult to absorb this amount of knowledge in a short time and to test learning. Because of this, we decided to develop a small mobile application using the major components needed to build an application (user interface, logic and data) without tutorials and guides. The application developed only consisted of keeping a list of friends / family, to register the gifts that could give each of them. This way, when someone on my friends list commented to me that he wanted a certain thing, I could register a new gift for this friend and consequently always offer a useful and desired gift at their birthday. The figure 5.1 shows the screenshots of my sample application, called GIFTS.

![Screenshots of my first application developed with Service Studio](image)

*Figure 5.1: Screenshots of my first application developed with Service Studio*

We can conclude that for those unfamiliar with the platform and the OutSystems technology it is a bit
complicated to build applications quickly without ever seeing the tutorials over and over again. However, when you are familiar with the platform, it is a good thing to use this technology because you really can build a complete application quickly.

5.3 Target Application

It was important to define a target application to test and compare the two versions, OutSystems Service Studio and the new OutSystems Touch. This way, all evaluations can follow the same pattern.

When starting we did a brief search about the most used apps in the last year (2016)\(^1\). Among others, Spotify is in the ranking of the most downloaded apps. For this reason, and also because it is one of the applications most used by me, we decided to define as the target application a simple Music Player application that follows the same pattern as Spotify (figure 5.6).

![Mockups of the target application](image)

Figure 5.2: Mockups of the target application

5.4 Evaluate the current state of OutSystems

After defining the target screens, we proceeded to prepare the usability tests to evaluate the current state of the existing OutSystems platform. Since the goal was to develop the version of Service Studio for touch environment, this is important because the goal of a usability test is to identify usability problems, determine user satisfaction and collect data of user’s performance. This way, it is possible to understand user needs, leading the development of a more familiar result and eliminating frustrations and some existing errors of the existing desktop platform, contributing to the product improvement [5].

Although we already had done the interviews to define personas and already had realized the most used features in Service Studio (like button widget, label widget, css to edit widgets properties, among

---

\(^1\)http://www.businessinsider.com/top-apps-of-2016-so-far-2016-8
others), there was a need to make usability tests to the existing platform to evaluate the current state of Service Studio. The information taken from the interviews previously will serve to help define tasks using mostly selected features as the most “important”.

The first task of the Service Studio usability test only aims to familiarize the testers with the platform, since this is complete and can become complex for those who have no previous experience with it. The remaining two will be the same tasks that will be used for the usability testing of the OutSystems Touch. This way it will be easier to evaluate the new system and compare it with the current Service Studio. The tasks use the screens defined as the target screen and encompass a large number of elements of the Service Studio for UI development. Therefore, they are considered complete tasks. Then, we did the script for usability tests to present to the testers- The script consists on the development of the targets screens and it was written in portuguese, as we can see:

**TESTE DE USABILIDADE**

Imagine que és um freelancer e um cliente te pede para fazerem uma aplicação para ouvir as suas listas de músicas guardadas.

Esta app deve permitir aos utilizadores adicionar uma música a uma biblioteca, ver as bibliotecas de músicas guardadas, pesquisar músicas ou ouvir rádio.

Tu acabaste de ouvir falar desta nova tecnologia, chamada OutSystems e queres descobrir se esta é a melhor maneira de desenvolveres a aplicação.

Atenção: Apenas vais desenvolver a UI (User Interface).

**TAREFA 1**

Cria uma nova aplicação para mobile phone, com um novo módulo, chamada “Test C” com o logotipo fornecido e cor #FF3C6C.

No ecrã splash apenas deverá estar o logotipo, sem títulos. A mesma coisa para o ecrã de login, como nos seguintes ecrãs.

**TAREFA 2**

Deverá criar os ecrãs “Your Library”, “Radio” e “Search”.

Depois, cria um menu como o da figura abaixo. Nota: Este menu deverá ter links para todos os respectivos ecrãs e deverá ter os ícones list all, music e search.
16 usability tests were performed on Service Studio, with potential OutSystems users: students of the Telecommunications and Informatics Engineering. The tests had a duration of 45 minutes and were composed by a small introduction of contextualization, task accomplishment, final questionnaire and debriefing.

To record the task accomplishment, there are some metrics we used in usability evaluation:

- **SUS (System Usability Scale)** is a method of ascertaining the satisfaction of a system. To measure it, the user answers 10 questions (final questionnaire of the usability tests that can be seen in annex A), with a scale from 1 (Strongly Disagree) to 5 (Strongly Agree). Then, it is necessary to do some calculations:
  
  - For each of the odd numbered questions, subtract 1 from the score;
  - For each of the even numbered questions, subtract their value from 5;
  - Take these new values, and add up the total score. Then multiply this by 2.5.

If the result is 80.3 or higher, the system is considered an A: users are satisfied with the system and will recommend it to their friends, 68 or thereabouts is a C, the system is acceptable but could improve and 51 or under is an F, which means it is important to make usability the top priority now and fix this fast. The ranking can be seen in figure 5.4, too.

![Figure 5.3: Screenshot of the usability test script](image)

![Figure 5.4: Ranking of SUS](image)
• Task Time is a measure of efficiency and productivity. It records how long the user takes to complete a task in seconds or minutes, starting when the user finishes to read the task scenarios and ending when the user has finished the whole task, including reviewing.

• Task Completion consists of registering if the user finished the task successfully (1) or not (0).

• Number of Errors consists of registering any unintended action, slip, mistake or omission a user makes while attempting a task with a description and then we can add severity ratings to errors or classify them into categories. Expectation evaluates how difficult users expect a task to be comparing it to actual task difficult rating (from the same or different users).

Due to the fact that the results of the usability tests were poor, in the sense that few tasks were being performed without any suggestion, we decided to create a table for each task, in order to define the aspects to be evaluated to convert task completion in percentage, otherwise, they would be almost all 0 and it was impossible to make the desired evaluation. The screenshot of the tables for each task is in figure 5.5.

**Figure 5.5:** Screenshot of the tables of each task
Lastly, we analyzed the results of the usability tests performed to consolidate all information and draw conclusions. We did a summative evaluation, which aims at evaluating a particular product, in this case Service Studio, and we decided to do two kind of evaluation:

- **Quantitative evaluation** (as can be seen in table 5.1)

  This evaluation consist in evaluating the **effectiveness**, through the metrics task completion and number of errors; **efficiency**, through the time elapsed during the accomplishment of a specific task and the **satisfaction**, measured through the SUS.

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Comp.</td>
<td>N. errors</td>
<td>Time</td>
</tr>
<tr>
<td>User 1</td>
<td>60%</td>
<td>10</td>
</tr>
<tr>
<td>User 2</td>
<td>70%</td>
<td>11</td>
</tr>
<tr>
<td>User 3</td>
<td>50%</td>
<td>15</td>
</tr>
<tr>
<td>User 4</td>
<td>60%</td>
<td>14</td>
</tr>
<tr>
<td>User 5</td>
<td>100%</td>
<td>3</td>
</tr>
<tr>
<td>User 6</td>
<td>30%</td>
<td>17</td>
</tr>
<tr>
<td>User 7</td>
<td>40%</td>
<td>25</td>
</tr>
<tr>
<td>User 8</td>
<td>80%</td>
<td>17</td>
</tr>
<tr>
<td>User 9</td>
<td>80%</td>
<td>9</td>
</tr>
<tr>
<td>User 10</td>
<td>80%</td>
<td>15</td>
</tr>
<tr>
<td>User 11</td>
<td>90%</td>
<td>6</td>
</tr>
<tr>
<td>User 12</td>
<td>40%</td>
<td>15</td>
</tr>
<tr>
<td>User 13</td>
<td>70%</td>
<td>32</td>
</tr>
<tr>
<td>User 14</td>
<td>80%</td>
<td>10</td>
</tr>
<tr>
<td>User 15</td>
<td>60%</td>
<td>17</td>
</tr>
<tr>
<td>User 16</td>
<td>80%</td>
<td>13</td>
</tr>
</tbody>
</table>

| Average | 68.1% | 14 | 9min | 68.1% | 20 | 9min | 68% | 44 | 19min | 57.5 - C |
| Standard Deviation | 20% | 7 | 3min | 16% | 6 | 2min | 10% | 17 | 2min | 4 |

**Table 5.1:** Records of the usability tests performed to Service Studio

- **Qualitative evaluation** (as can be seen in table 5.2)

  This evaluation classifies the most common mistakes made by order of severity. This means that the more users made that error, the more serious it is.
<table>
<thead>
<tr>
<th>Frequent 11+ users</th>
<th>Drag and drop the elements inside the screen to the desired location</th>
<th>16 users</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In the window to import image to screen, by default there is an image called “logo” that people select because they think it was the logo they introduced at the beginning, but it’s an example that OutSystems provides and it is OutSystems logo</td>
<td>16 users</td>
</tr>
<tr>
<td></td>
<td>People do not add bottom sidebar items to bottom sidebar, add logo icons.</td>
<td>16 users</td>
</tr>
<tr>
<td></td>
<td>People do not know that there are default screens or where they are</td>
<td>15 users</td>
</tr>
<tr>
<td></td>
<td>Copy + paste of elements</td>
<td>14 users</td>
</tr>
<tr>
<td></td>
<td>They find the menu but do not open because the symbol is not a screen as in other screens like login screen</td>
<td>13 users</td>
</tr>
<tr>
<td></td>
<td>They do not know to put the name of the CSS class they have created</td>
<td>13 users</td>
</tr>
<tr>
<td></td>
<td>They do not know to create new screens</td>
<td>12 users</td>
</tr>
<tr>
<td>Probable 6 – 10 users</td>
<td>They click once to open the screen, instead of two clicks</td>
<td>10 users</td>
</tr>
<tr>
<td></td>
<td>People change the container of the menu screen to look like the menu, because the preview does not match the reality</td>
<td>9 users</td>
</tr>
<tr>
<td></td>
<td>They do not put several elements on the same line (they do not use containers or columns)</td>
<td>9 users</td>
</tr>
<tr>
<td></td>
<td>They do not know where to put the color code</td>
<td>8 users</td>
</tr>
<tr>
<td></td>
<td>People do not know where they can create a new class of CSS because it appears a window with classes but that is blocked and do not see that they can change of tabs in the top of the window</td>
<td>8 users</td>
</tr>
<tr>
<td></td>
<td>People do not know how to center elements</td>
<td>7 users</td>
</tr>
<tr>
<td></td>
<td>People do not find editor of the elements (CSS)</td>
<td>7 users</td>
</tr>
<tr>
<td></td>
<td>They select the color, upload the logo and do not realize that the color changes to the primary color of the logo and that they have lost the color they introduced</td>
<td>6 users</td>
</tr>
<tr>
<td></td>
<td>Cannot resize the logo</td>
<td>6 users</td>
</tr>
<tr>
<td>Occasional 1-5 users</td>
<td>They do not select a logo because it says “Upload icon” and they think that icon may not be the logo</td>
<td>5 users</td>
</tr>
<tr>
<td></td>
<td>They do not find the widget - range</td>
<td>5 users</td>
</tr>
<tr>
<td></td>
<td>Difficulty finding the icons</td>
<td>5 users</td>
</tr>
</tbody>
</table>

*Table 5.2: Records of the usability tests performed to Service Studio*
After this evaluation to the OutSystems platform, Service Studio, we can conclude that, mostly, the system was evaluated with a C in System Usability Scale, which means that the system is acceptable but could be improved. The most common mistakes in the accomplishment of the tasks were registered, to take into account in the development of the OutSystems Touch, thus trying to improve the satisfaction, effectiveness and efficiency of the users in relation to the platform.

5.5 Touch Challenges

To design an application for an touchable environment that already exists on the desktop, it is necessary to maintain continuity, consistency and maintain brand (the look of each version should be similar) to convey familiarity with the existing system. However, there are several factors that make this process challenging

- **Mobile screens are smaller.** Mobile screens are physically smaller and the user can see a lot less information at once. Smaller screens means fewer pixels than desktop displays, too. As such, it is important to present important information “above the fold”, use an easy-to-read font, and not overwhelm the user with too much content on the page because simple is best and less is more. So the use of tabs and trays is great to stretch screen space.

- **Slower processors.** Mobile devices generally have much less processing power than desktop computers, so mobile apps can run slowly. This problem is already being solved with the arrival of the iPad Pro and Microsoft Surface.

- **Touch input.** Mobile users work using their fingers and this has several implications, like no hover events or mouse pointers. On the existing OutSystems platform, these are used a lot. It is also an issue because the finger is thicker than the mouse pointer, so it is important use large controls and indicators that resemble their physical equivalents and use gestures to enhance the experience for your mobile users like swiping, pinching, and so on.

- **Inadequate keyboard.** Normally, mobile devices have a little keyboard or tapping a minuscule on-screen keyboard, so typing on them is still far from being a pleasant experience. It is easier if there are autocomplete function to text fields and search fields, reducing the inputs required from users.

Mobile devices are becoming the prevailing computing platform. But the needs of mobile users are different from those of desktop users. There are still few developments or high productivity applications

---

3https://developer.apple.com/design/tips/
4https://www.paradoxlabs.com/blog/mobile-vs-desktop-10-key-differences/
for mobile devices, or which are properly optimized for use in these devices. It is important to change this, following the evolution of the technology, providing a great work and development experience in a touchable environment. When the mouse appeared in 1968, existing software and users took time to adapt to this new, more natural, input device. The same seems to be happening with touch today.

5.6 Prototype

Following this analysis, the low-fidelity prototype below was proposed. This low-fidelity prototype has two sidebars: one with the widgets that can be added to the screen and the other with links to the various screens and the properties of the selected widget. Each of these sidebars also contains a hide button, which allows users to hide them and view the final application. The prototype also contains a structured menu of the end application that can be seen in the view mode of the application.

![Low-fidelity prototype](image)

**Figure 5.6:** Low-fidelity prototype

Formative evaluation with case studies and iteration

A formative evaluation helps the designer of a product, during the development stages, to increase the likelihood that the final product will achieve its desired goals. It involves evaluating a product, with
the goal of detecting and eliminating usability problems. Observations and recommendations are used to immediately improve the design of the product and refine the development specifications [7].

For a formative evaluation of the low-fidelity prototype presented above, we performed a usability test with three Service Studio expert users. Users would have to do task 3 (corresponding to the development of the defined target screen) of the script for the usability test used to evaluate the current status of Service Studio, which can be queried again in Annex A.

It was evaluated the effectiveness, through the metrics task completion and number of errors, and efficiency, through the time elapsed during the accomplishment of a specific task, as you can see in table 5.3.

![Table 5.3](image)

Table 5.3: Records of the usability tests to the low-fidelity prototype

The task was performed quite easily and almost without errors. However, it was possible to collect some feedback from testers:

- The side buttons that allow editing of the screen are too small;

- There is some confusion between the on-screen menu button and the button that allows editing the screen (the only error which occurred during the tests).

In order to iterate the low-fidelity prototype, it was only necessary to solve the problems mentioned by the users, because the feedback obtained was positive. The iterated prototype (as you can be seen in figure 5.7) was developed in Justinmind with the following change: increase the side buttons, which help the existing swipe to open the same sidebar that allows the creation or editing of the screen.

This change also makes it possible to resolve the error committed during the execution of the task, since the buttons are larger and thus do not mislead the user with the on-screen menu button.
Figure 5.7: Prototype iterated with improvements of the low-fidelity prototype
With the prototype described in the previous section, the implementation of the OutSystems Touch resulted in the definition of the necessary components and layers that allow the final user to develop a mobile application directly from the touch environment. In order to have a more clear picture, we started by defining in section 6.1 the data model that would serve to store the different screen components (widgets). The next challenge, as explained in section 6.2, was to build an interpreter to convert the visual language into the data model and later, the functionality to edit screens was added (section 6.3).

With the full application model in place, we refactored the code and we changed the structure of the application to be saved to a database due to reasons explained in section 6.4. After this, we prepared a first functional prototype and then, we performed a formative evaluation to clarify doubts and to decide what was the best way to build a user-friendly application (section 6.5). We finally developed some additional functionalities to improve the prototype, as explained in section 6.6.

### 6.1 Defining the Data Model and Creating Widgets

At this point we had target screens with a visual language that needed to be translated into computer language. Since the future work of the application (although that is not the scope of the project) is to make OutSystems Touch compatible with the desktop version, the data model used has to be a standard that is easy to convert between both.

JSON (Javascript Object Notation) is a way to write objects in JavaScript, often referred to as a “universal” format that is convenient for exchanging information between applications. JSON can be easily stored inside a variable as an Object, making it easy to use.

For that reason we defined JSON as our data model to store the screen data. This was also important to define the widgets supported in the application and the attributes that were required for each kind of widget.
Then, we started by adding the widgets (Text, Button, Image, Icon and User Avatar) to the user
interface that we wanted to implement.

Each widget was mapped to an OutSystems object and the status changes of each widget are Client Actions on the OutSystems Service Studio platform.

To add widgets, a client action named WidgetKindOnStarDrag (figure 6.1) was created, to be executed when a widget starts dragging. This client action allowed to create a widget with the default/initials properties and execute another client action, DrawWidget, to draw the widget depending on the kind and the corresponding default properties. It was decided to do two distincts client actions for this because as the properties of the elements are changeable, it will be necessary to use the DrawWidget more often.

![Diagram](image)

**Figure 6.1:** WidgetKindOnStartDrag client action

After that, the JSON of each widget added to the screen was generated. At this stage, it still did not have all the attributes of each widget, it only had the initial attributes (like you can be seen in figure 6.2), like the kind of widget, identification (id) of widget and x and y position.
6.2 Develop an Interpreter to Save and Load Screens

At this point, there were widgets with some properties, but it was not possible to create new screens yet. To be able to accomplish this we need to store the data model corresponding to a screen (the JSON model defined in the previous section) and be able to load it, restoring the corresponding interface. To perform this translation we developed a JSON screen interpreter. Since the screen JSON is being saved in a variable, one possibility to persist screens is to save the JSON of each screen separately. Therefore, a screen is a piece of JSON that can be loaded at any time. We can see an example of the JSON load (the JSON used was the JSON example presented in previous section and correspond to the target screen) in figure 6.3.

For this, it was created a client action that deserialize JSON to draw an corresponding widget, like we can see in figure 6.4.
6.3 Allow Screen Edition

After being able to save and load screens, the next step is to allow screen edition. This is basically allowing the edition of the different widgets that are present in the screen which can be translated to the corresponding OutSystems actions.

To allow screen edition, it was necessary to define several steps: define the selected widget, show the widgets’ properties and edition of the widgets properties.

Define the selected widget

To define and show the selected widget by highlight was raised a new client action, called `OnWidgetSelect` (figure 6.5). The result can be seen in figure 6.6.
Show the widgets’ properties

The properties of each widget are the same that were defined in JSON. The widget structure created can be seen in the figure 6.7. In the case of the icon widget, it was necessary to make a popup with the icons that could be added, as in Service Studio (figure 6.8).
Edition of the widgets properties

To edit the widgets properties’ the client action *DrawWidget* was used. created to draw the widgets on the first drag with the initials properties. This was possible because properties are fields that allowing the user to input data. So, when the user changes the property field, the client action *ChangeProperties* is executed in order to draw the widget with new properties (figure 6.9).
At this stage, the interface was not optimized. So, there was many improvements to be fixed:

- Hide properties that are not in use from each widget (example of image properties in figure 6.10)

![Figure 6.9: ChangeProperties client action](image)

![Figure 6.10: Hiding properties](image)

(a) Image properties before hide properties unused

(b) Image properties after hide properties unused
• Add new color picker (figure 6.11)

(a) Old color picker to change widgets colors
(b) New color picker to change widgets colors

Figure 6.11: Adding a new color picker

• Upload images from iPad (figure 6.12)

(a) Text field to change the image (with a link)
(b) Button to upload images from iPad with current image preview

Figure 6.12: Changing the upload photos mode

6.4 Refactoring and Storage

At this point, we just serialized data (screens with widgets and their properties) to JSON, saving and loading when necessary, using a data structure. But there are many advantages to store data in databases:

• Data security;

• On-Demand Scalability;
• High Performance;
• Minimized data inconsistency;
• Improved data access.

Therefore, a data model to improve the storage of the OutSystems Touch was created, as we can see in figure 6.13. So, the code was refactored in order to follow this change. The main change was the way to create new screens. In the data model, there is a database entity attribute called `ScreenId`. So, if we wanted load the screen, it would display all the widgets with that `ScreenId`. The client action called `GetScreenOnAfterFetch` is the first action to be executed. If there are not screens, it was created a new screen. Otherwise, the app initializes in current screen.

![OutSystems Touch data model](image)

**Figure 6.13:** OutSystems Touch data model

After this, we decided to add a menu sidebar like it appears in Service Studio when an application is created, and we also added an icon to the widgets sidebar to allow users to see the final application, hiding the two sidebars, opening in this way, the View mode. When this mode is active, it only appears on the screen an icon that allow to back to the Edit mode, showing again the both sidebars. At this stage, the first functional prototype was done. There were still many improvements to do. But first, the formative evaluation was done on the prototype that can be seen in figure 6.14 to help clarify doubts and make decisions, as explained in the next section.
6.5 Formative Evaluation

The formative evaluation of the prototype development (figure 6.14) was made to increase the potential and functionality of the final product.

The main doubt to improve the user interface was: we had two sidebars, the widgets sidebar and their properties sidebar, taking up too much space on the screen - what should happen to the sidebars when the user wants to put a widget in one of the places where each of the sidebars is?

I performed an informal evaluation to five students of Computer Science with the task 3 (corresponds to the development of the defined target screen), the same task that was used to evaluate the low-fidelity prototype.

At this stage I did not use the metrics. I just registered the errors that people did:

- Resize image with fingers did not work. The only way to resize images was to change the width and height on properties sidebar;

- Try change the content of the text widget with double click on that field. This can only be accomplished by changing the text field on properties sidebar.

At the end, I asked users to put a button on the top right corner, because on the task (target screen) no widgets overlaps the sidebars.

The widget stayed ahead of the properties and screen sidebar. At this moment, I asked which solution was more suitable for them. I got the following opinions:

- Horizontal Scroll;

- Click on the sidebar and drag it to where I want it;

- To have a sidebar to add widgets on top and the second sidebar to move around according to the position of the widgets;
To have an option to change the position of the sidebars like developer tools of Chrome Web Browser;

The sidebars move automatically when the widget stay ahead of it.

6.6 Prototype Iteration

As a result of the formative evaluation, it was decided that two sidebars to build an application directly on mobile device (sidebar to add widgets and sidebar to view screens or edit properties) took up more screen space than desired and that it was better to gather all the content just in a unique sidebar (with tabs, the first one to add widgets and the second with the properties). This way, the problem is somewhat minimized. Then, it was possible to use the users’ suggestions mentioned above, adding to this unique sidebar an option to change the position and also move automatically when the widget stays over it. The result can be seen in figure 6.15.

![Figure 6.15: Edit sidebar](image)

(a) Edit sidebar on the left with Widgets+ tab opened  
(b) Edit sidebar on the right with Properties tab opened

After analyzing the implementation result of the move option of the edit sidebar as suggested by potential users on the formative evaluation, it was concluded that the result produced was not the expected one. This feature is quite intuitive in the developer tools of the Chrome browser, because it containsmouseover and tooltips to help the user understand the options offered, as we can see an example in figure 6.16. Since it is not possible to use these techniques to help user in touch environment, as already mentioned in Chapter 5, Section 5.4, called Touch Challenges, this was considered an inappropriate solution, because it is not clear which of the options is selected (grey or blue color). To solve this problem, it was decided to place only one arrow in the edit sidebar that allows changing the sidebar to the opposite side of where it is (figure 6.17).
To improve the prototype, and also as result of the formative evaluation, a work list was created to improve the user experience, eliminating the keyboard needs, taking into account the natural gestures of the touch environment and also to improve the functionality and the flow of the OutSystems Touch.

Work list to improve user experience and flow:

1. Automatically move the tabs of the edit sidebar to “Properties” when a widget is selected

When users selected a widget, the edit sidebar still has the same default content with the widgets tab open. It’s more user friendly if the tabs move automatically to properties, because if the user select an widget, the user’s purpose is to view or change its properties, instead of adding new widgets.

2. Facilitate editing of text properties (bold, italic, normal, size)

To edit the text properties bold, italic or underline, it was necessary to edit the corresponding
field (style field) of the edition with the keyboard. How it was explained on chapter 5, section 5.5 - Touch Challenges, the use of keyboard should be minimized on iPad, so it was easier if the users could choose the style of text through the buttons. This had already been proposed on prototype designed initially, because it is a familiar feature to all users, used by Microsoft Office, Service Studio and all others applications to create or edit contents. To edit the text size, it was also necessary to use the keyboard to edit the font size field. This was not practical, so a field was created, aided by two buttons at the ends, the first one to decrease and the second one to increase the size. Despite this, it is still possible edit the field with the keyboard if the users need increase or decrease to a distant number. To implement this changes, a button was created for each property mentioned, with a client action on click to assign the correspondent property and update the widget.

![Figure 6.18: Text properties before (on the left) and after (on the right) easy style](image)

3. Automatically builds an application menu with links to all screens

In the formative evaluation, some testers opened the menu of the application because they thought that it is the menu for editing the application. Because of this, it was decided to structure the menu of the resultant application and, as in Service Studio, the menu of an application already exists. This menu contains all links to the corresponding screens.

4. Do not open the keyboard when using color picker or when change the icon selected

When users use the color picker or when they change the selected icon, the keyboard opened. This caused confusion because some users thought they had to enter the color code they wanted, for example. So, in this cases, it was changed so the keyboard does not open in those cases.

5. Allow renaming screens
Figure 6.19: Structured menu with links to existing screens

It is possible to add a screen with the wrong name or to need to rename it later, so it is important to allow users rename a screen. For this, was created a popup to rename the screen that appears after a double click in a screen name.

6. Allow deleting screens and widgets

It is possible to add a screen or widget unintentionally, so it is important to allow users delete one screen of the screens list or one widget. For this, the approach implemented for the screens was one of the more used on touch environment to delete elements: swipe to left. For the widgets, it was added a button on properties that allow to delete it (figure 6.20).

Figure 6.20: Button to delete a screen (on the left) and button to delete a widget (on the right)

7. Resize images with two fingers (pinch gesture)

Pinch is a natural and intuitive gesture on touch environment. We are all quite familiar with this
gesture, for example to zoom a photo on our smartphone. It uses two fingers at the same time to resize an image. This allows scaling the image proportionally. To implement this functionality, it was used a javascript open-source library for multi-touch gestures, called Hammer.js\(^1\).

8. Lines to guide alignments of the elements

It was difficult to build applications with elements aligned with consistent look, so we decided to implement lines to allow users to guide alignments of the elements. The alignments were based on Microsoft Office alignments because it is a widely used tool and it’s quite intuitive. To more easily align elements, users can snap elements to a grid line that runs through the vertical edges, horizontal edges, and centers of other elements (figure 6.21). The line becomes visible only when users drag an element with the same alignment of another element.

![Figure 6.21: Example of the left alignment on OutSystems Touch](image)

9. Add link property to allow linking screens

To be able to view a complete application, it must be possible to navigate between screens in view mode. It is important that widgets have destination links if needed. For this, a Link property has been added to the widgets. This way, each widget can have a destination link and the user can navigate to it when it is in view mode.

10. Suggest colors already used in the application, in addition to the color picker option

To make sure users create applications with consistent look, the suggested colors are the four colors already used in the current application, and they are an alternative to the color picker. To select them, the columns corresponding to font color and background color of the widgets were

\(^1\)http://hammerjs.github.io/
imported from the database and a column, called Count, was created to count the number of times a particular color has already been used. This way it was possible filter the four most commonly used colors, by sorting the colors in descending order of the count column.

11. Distinguish between the view mode or edit mode

At this stage, when users clicked on icon to open view mode, the edit sidebar disappears, but it was possible continuing to move widgets. In the view mode, it is important to see the application totally like in the final application. For this, it was blocking the movement of the widgets in this mode, adding JavaScript code to remove all existing highlights and not execute the Event Listener `touchmove` when the application is in view mode.

12. Allow scroll in edit sidebar when there are many screens and the sidebar is larger than the height
of the device screen

The edit sidebar was too large when multiple screens is added and some widgets properties were cut off. To prevent this, it was allowed to make scroll on the edit sidebar. This way, it is possible, to always see all the content of the sidebar.

Developing and Testing

During the implementation of these improvements, we used the Chrome Simulator to test code and used its debugger when necessary. Because of this, we came across a few unexpected things when, in the end, we tested OutSystems Touch on the final device, the iPad. For example, to rename the screens, the solution developed consisted of, after double click, to open a popup that allowed to rename the screen. This feature does not work on iPad, so we took another approach - the use of long press. Another example, it is not possible to simulate the pinch touch gesture in Chrome, so the development time of this functionality was much higher than expected, since it was necessary to always test on iPad and also because the iPad did not have debugger.

Ironically, these kinds of problems that arise from developing on a different device than the one used by end users, are exactly the ones that will no longer be a problem when applications are created with OutSystems Touch.

Applications Page

Service Studio allows users to build more than one application, so it was decided that the OutSystems Touch should allow the creation of more than one application simultaneously, too. So, the data model for the OutSystems Touch was changed, because a new entity was added, called Application, with three attributes: id, name and color. Additionally an applicationId attribute was added to the Screen Entity so it would be possible display the correspondent screens when an application is opened.

To design the application page, we based ourselves on the application page of the Service Studio (figure 6.25(a)). The result can be seen on figure 6.25(b). It is a simple page, where it is possible to create a new application, choosing its name and its main color, like in Service Studio.

The color of current application appears on the menu sidebar in the application and it will be the first of the suggested colors, to promote consistency. The name of current application also appears on the
top of the edit sidebar as well as an icon to users go back to the application page. In this place, it was an icon to add new screens, so it was decided to add another screen item to the list which allow to create a new screen. The result can be seen in figure 6.26.

Animations to improve the user experience

Well-designed animations make the difference in modern applications. Animations reduce cognitive load and prevents change blindness. Well thought-out and tested functional animation has the potential to fulfill multiple functions like visual feedback on server actions, navigational transactions, among others. This is a natural part of the design process and can turn a digital product with memorable good
So, we decided to add animations to edit sidebar. The edit sidebar moves to the right and to the left, as we know. Sometimes, jumping users to a new position without transition is confusing. So, we used the animation to smoothly transport users between the positions of the edit sidebar. This way, we can guide the user’s attention in ways that both inform and delight. These animations are also used on the tabs that allow users move to add widgets or see or edit widgets properties and are also used to hide edit sidebar when users clicked on view mode or to show it when users clicked to open edit mode.

Feedback from OutSystems R&D Product Design Team

At this stage, a demonstration of the current state of the OutSystems Touch was performed for the OutSystems R&D Product Design Team, whose vision is “to deliver a product users will fall in love at first sight, and keep on loving forever”. The team has two designers and three engineers. The demonstration consisted of simulating a real case of the OutSystems Touch use. The story was as follows: I am a babysitter and one day, I needed to call INEM for an emergency. Faced with this tense situation, I had look for the location of the child's address in Google Maps. So the next day, I sat on the sofa after putting the child to sleep and started developing my application where I gathered all the information of all the children for whom I do babysitting. It was simple and fast because the application is simple to use and allowed to develop the complete interface of this application. They have a lot of knowledge on user experience and user interface, so this was really important to my work, since it allowed me to get some feedback and suggestions, which were later implemented:

- When the application is in view mode, there is a static button that allows users to re-enable edit
mode. There may be issues if a widget is dragged to that same point, because it overlaps. Suggestion: allow dragging the edit button to multiple corners or automatically move the button with the edit sidebar. The second one was the approach implemented. When the sidebar is on the right and the users click to open view mode, the icon that allows to open the edit mode is on the right too. It is the same thing to the left side (sidebar on the left, icon for edit application on the left too).

- It is weird when users click on a text element and the editing place is in a field of the edit sidebar and not where users have clicked, because they were focused on where the text element was. Suggestion: when users click on the text element, it redirect their attention to the editing place; Like it was expected for the users, when selected text to edit, was expected to view the edition field in the same place where they are. So, we tried this, when clicked on the text element, all the text became selected and it was possible to replace it. Although it worked perfectly on Chrome Simulator, it never worked on a mobile device, after several tentatives. So, it was decided aborted this approach and to implement the approach suggested above, as we can see in figure 6.27.

![Figure 6.27: What happens when users click to edit the text widget](image)

- Add input widget, because it is an important widget to build more complete applications and it is one of the most important widget in Service Studio. The input widget is a way to make the text editable. This way, it is possible to build interfaces with forms, among others.

Adjustments of interface details

In the end, we decided to make some adjustments to OutSystems Touch to make this application more familiar to current Service Studio users, like using Service Studio icons and adding the color of the Service Studio to the edit sidebar. Some icons that we used in OutSystems Touch do not exist in Service Studio, like the icon to add new screen or the box widget. For these, we tried to make the icons as similar
to the others as possible. In figures 6.28 and 6.29, we can see Service Studio and OutSystems Touch to compare both.

In addition, we added a shadow to the edit sidebar and optimized some buttons in the properties (figure 6.30) and icons to obtain an application more visually appealing.

Figure 6.28: Service Studio Interface to build interface of applications

Figure 6.29: OutSystems Touch Interface to build interface of applications
Figure 6.30: Text properties in the edit sidebar of the OutSystems Touch
7.1 Protocol

The evaluation allows us to assess the effectiveness, efficiency and usability of a system and therefore it is essential in this study.

Its main goal is to ensure that users can successfully complete the assigned tasks. It is important to perform a significant number of tests in order to get enough feedback and ideas for improvement.

The evaluation methodology of OutSystems Touch was based on the interactive design method - performing usability tests based on low-fidelity prototypes that would be used to quickly test, iterate and refine the prototype until it reached good levels of usability.

The tasks used for the usability tests are two tasks (task 2 and task 3) of the usability test script already used for the tests previously performed, as mentioned. The script for usability tests to present the testers is the same presented in Chapter 5 - section 5.4 - Evaluate the current state of OutSystems and consists in building a music player screen and a menu application with link for the created screens.

The metrics analyzed were the same as those previously mentioned also in Chapter 5 - section 5.4 - Evaluate the current state of OutSystems, in the same way that we performed the usability tests to Service Studio, in order to make a better comparison between the two systems.

The usability tests were performed on OutSystems Touch with potential users. They had the estimated duration of 20 minutes and were composed of a small introduction for contextualization, task accomplishment, final questionnaire (annex A) to later calculate the System Usability Scale and a debriefing.
7.2 Results

The table below (table 7.1) shows the results obtained in each of the tasks, as well as the average obtained and the standard deviation.

<table>
<thead>
<tr>
<th>Task 3</th>
<th>Task 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Comp.</td>
<td>N. errors</td>
</tr>
<tr>
<td>User 1</td>
<td>100%</td>
</tr>
<tr>
<td>User 2</td>
<td>100%</td>
</tr>
<tr>
<td>User 3</td>
<td>100%</td>
</tr>
<tr>
<td>User 4</td>
<td>100%</td>
</tr>
<tr>
<td>User 5</td>
<td>100%</td>
</tr>
<tr>
<td>User 6</td>
<td>100%</td>
</tr>
</tbody>
</table>

| Average   | 100% | 1,3   | 5min | 83,3% | 3,6   | 4,1min | 96,6 - A |
| Standard Deviation | 0 | 2 | 0,9 | 26 | 5 | 3 | 7 |

Table 7.1: Records of the usability tests performed to final version of OutSystems Touch

The errors occurred during task accomplishment were:

- The button that allowed to move the edit sidebar to left or right was mistaken for a button to undo features (back button). This problem was fixed and the icon was changed to another (figure 7.1)

![Figure 7.1: On the left, the old icon to move edit sidebar. On the right, the new icon to move edit sidebar.](image)

- The structured application menu was not found and a new screen was created to build it.
To discuss the results, we did a box plots to show distribution of a set of data, comparing the results of the analyzed metrics (task completion, number of errors, time and SUS) to the tasks of the usability tests performed to both systems (task 2 and task 3 that can be seen in Chapter 5 - section 5-4 - Evaluate the current state of OutSystems): OutSystems Service Studio and OutSystems Touch. Due the different number of samples in each system, it was decided to make a different graph for each of the metrics of each of them, but with the same scale for a better interpretation of the results.

- Task 2 (create a menu with links to the screens):

![Figure 7.2: Box plots of the % task completion of the OutSystem Service Studio (on the left) and the OutSystems Touch (on the right)](image)

When OutSystems Touch is used, a higher average is recorded to task completion (83%) compared to OutSystems Service Studio (68%).

![Figure 7.3: Box plots of the number of errors to complete the task of the OutSystem Service Studio (on the left) and the OutSystems Touch (on the right)](image)

When OutSystems Touch is used, a lower average is recorded of the number of errors to complete.
the task (4%) with a maximum error value of 12 compared to OutSystems Service Studio (20%) with a maximum error value of 32.

When OutSystems Touch is used, a lower average is recorded of the time to complete the task (4%) with a maximum error value of 8 compared to OutSystems Service Studio (20%) with a maximum error value of 15.

- Task 3 (Create a Music Player screen):

When OutSystems Touch is used, all testers completed the task successfully. When OutSystems Service Studio is used, was recorded to task completion of only 68%.

62
When OutSystems Touch is used, a lower average is recorded of the number of errors to complete the task (1%) with a maximum error value of 5 compared to OutSystems Service Studio (44%) with a maximum error value of 90.

When OutSystems Touch is used, a lower average is recorded of the time to complete the task (5%) with a maximum error value of 6 compared to OutSystems Service Studio (19%) with a maximum error value of 25.

- System Usability Scale:
When OutSystems Touch is used, a higher average in usability scale is recorded (97%) than OutSystems Service Studio (58%).

We can conclude that with OutSystems Touch it was possible to perform the same tasks more efficiently and efficiently, with more satisfaction than in OutSystems Service Studio.

In addition, a Shapiro-Wilk normality test was also performed on the System Usability Scale data from each of the systems, OutSystems Service Studio and OutSystems Touch. This test detects all departures from normality, through the p-value/significance:

- If P is higher than 0.05, it may be assumed that data are normally distributed;
- If P is less than 0.05, then the hypothesis that the distribution of the observations in the sample is normal, should be rejected, and data are not normally distributed.

Below, we can observe the histograms that show the frequency of SUS relative to the two systems as well as the corresponding normal curve and the tables with the data referring to the normality test with values for each of them to be able to make a conclusion (figure 7.9 and 7.10).

We can conclude that the data of SUS of the OutSystems Service Studio are normally distributed (with significance value higher than 0.05) and data of SUS of the OutSystems Touch rejects the hypothesis that data are normally distributed (with significance value lower than 0.05). This way, we advanced to nonparametric tests, because there is at least one sample with a non-normal data distribution. So, we used a nonparametric test called Wilcoxon (signed-ranks) and we can see that the difference of the results is big (figure 7.11).
OutSystems Touch has a positive evaluation, because the final prototype has higher scores in the metrics defined than the current version of the platform, objectively bringing OutSystems closer to the goals defined in the introduction:

1. New users find it easier to learn how to use OutSystems for front-end development.
2. The development experience is more suited for devices that allow touch as well as typing, such as Microsoft Surface.

3. The prototype makes it easier to perform quick development and test cycles, directly in the device.

4. Developers are able to quickly tweak their applications, in the field and in close contact with the end users.
Conclusion and Future Work

This dissertation presents a prototype of an iPad version for the Service Studio IDE, called OutSystems Touch. We began by contextualizing the OutSystems platform and presenting existing software that is used in touch environments, concluding that there is still nothing for the enterprise market, as what we intend to develop. We also analyzed the challenges of creating mobile versions of existing desktop apps, so that we are aware of the problems that are usually faced when using complex apps in a mobile environment.

A proof of concept was also conducted to evaluate the possibility of using OutSystems Service Studio to develop OutSystems Touch, as well the research and learning of this technology to better understand the real needs of the users and to familiarize ourselves with the platform. To complete this challenge, we prepared usability tests to Service Studio to evaluate the current state of this platform and to collect sufficient data to compare with OutSystems Touch in the end, with the possibility of improving usability.

After this, we designed the prototype of OutSystems Touch taking into account the touch challenges previously analyzed and we started the implementation. We did three iterations of the prototype after other three evaluations.

Finally, we present the final evaluation of the OutSystems Touch where we concluded that it has a positive evaluation within the stipulated timeframe and the proposed goals.

To the future work, the OutSystems Touch will to be integrated in the Outsystems Service Studio by a plugin that allow users to open an application developed in OutSystems Touch on the Service Studio and to continue the development there, using all functionalities that it offer to users (business logic and data model). As we know, due time restrictions, OutSystems Touch only allows the development of the user interface of an application. Ideally, in the future, OutSystems Touch will be a full version of Service Studio on the iPad, with the possibility of developing complete and high-productivity applications.

Regarding the design decision of making layouts using absolute positioning, as said, they imply the limitation that users created interfaces for a single display size. To make this more powerful, some other
strategies must be explored, either using Responsive Web Design, as the current version of OutSystems Platform does, creating special widgets just for this (e.g. using the new CSS Grid), or adopting a technique similar to Apple’s Auto Layout that allows users to define constraints between different elements sizes and margins.

It’s believed that the development of the OutSystems Touch will contribute to the breakthrough of touch technology, by delivering a prototype of the first fully visual prototyping platform for mobile devices, directed to the enterprise market. Most importantly, OutSystems Touch allows OutSystems users to develop and test their apps continuously on a mobile device through a Service Studio that is easier to learn, can be used anytime, anywhere, and brings developers, end-users and mobile devices closer together.
References


Questionnaire to calculate SUS

Having in mind the test we just had, please answer the following questions. Don’t be afraid to hurt our feelings, we are trying to improve and need your help doing it!

<table>
<thead>
<tr>
<th></th>
<th>Don’t Agree</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I think that I would like to use this system frequently.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I found the system unnecessarily complex. (The system should be simpler)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I thought the system was easy to use.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I think that I would need the support of a technical person to be able to use this system.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I found the various functions in this system were well integrated.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I thought there was too much inconsistency in this system.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I would imagine that most people would learn to use this system very quickly.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I found the system very cumbersome to use. (cumbersome - slow or complicated and therefore inefficient)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I felt very confident using the system.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I needed to learn a lot of things before I could get going with this system.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>