



Licenciatura em Sistemas e Tecnologias da Informação

***Green Store Initiative***

Projeto Final de Licenciatura

by Tiago Enrique Afonso

Student nr 20172160

Supervisor: Professor Doutor Alexandre Humberto dos Santos Barão

Barcarena

Março de 2020

Atlântica

Licenciatura em Sistemas e Tecnologias da Informação

***Green Store Initiative***

Projeto Final de Licenciatura

by Tiago Enrique Afonso

Student nr 20172160

Supervisor: Professor Doutor Alexandre Humberto dos Santos Barão

Barcarena

Março de 2020

O autor é o único responsável pelas ideias expressas neste relatório

## **Acknowledgements**

First and foremost I would like to thank my parents for the help and financial support throughout these last three years as well as my grandparents that have always supported me and my education.

I would like to thank the staff as well, professor Alexandre Barão, for his invaluable help, support and guidance during this project, a special thank you as well to professor António Aguiar that helped me maintain realistic views of this project. Also, I care to mention the rest of the faculty that aided me and helped me learn throughout these three years, some of them are still at Atlântica, some have moved on, but all stay in my heart alongside the cherished memories I hold of the experiences and joy I felt.

A special note of appreciation is also due to all my classmates, more than once they helped me out in a pinch and their support has not been overlooked and certainly not forgotten.

Thank you so much for everything.

(Blank page for formatting)

## **Abstract**

### **Green Store Initiative**

The goal of this project is to solve a real world issue, that of reducing waste.

One of the major waste management companies in Portugal offers a service, the green store initiative. It is a simple, yet thoughtful service targeted towards small commerce businesses such as stores and restaurants. This service helps the owners of the establishment by offering door-to-door waste collection services. The way it is presently being done is not very efficient, and this project will attempt to offer a solution to that issue.

This report begins with an analysis of the current situation and an outlining of the plan and strategy to follow in order to achieve the desired results, followed by a literature review where several related and important subjects regarding the development of the application are detailed and analysed. After, a detailed presentation of the functional and non-functional requirements is made followed by an in-depth overview of the main architectural processes of the application. Furthermore, chapter 5 showcases many of the architectural processes discussed in the previous chapter, but with a focus on UI and UX followed by a conclusion and analysis of the accomplished goals and plans for the future.

*Keywords: Waste, Web, Cross-Platform, Application, OutSystems, Green.*

(Blank page for formatting)

## **Resumo**

### **Green Store Initiative**

O objetivo deste projeto é resolver um problema real, aquele de reduzir desperdício e resíduos.

Uma das maiores empresas de recolha de resíduos em Portugal oferece um serviço direcionado para o pequeno comércio, nomeadamente lojas e restaurantes. Este serviço ajuda os responsáveis por estabelecimentos registados no serviço oferecendo recolha de resíduos porta a porta. A situação atual deste serviço não é muito eficiente, este projeto tem o alvo de melhorar a situação presente.

O relatório começa com uma análise à situação presente e os planos e estratégias para alcançar o resultado desejado, seguido de uma revisão de literatura onde vários tópicos e temas relacionados com o desenvolvimento da aplicação são mencionados e estudados. O terceiro capítulo mostra os requisitos funcionais e não funcionais da aplicação, por fim, o quarto e o quinto capítulo têm como objetivo providenciar uma análise profundo às componentes e processos arquitetónicos da aplicação tal como mostrar como esses processos se traduzem em ecrãs navegáveis pelos utilizadores. O relatório conclui com uma deliberação sobre os objetivos alcançados, tal como uma reflexão sobre a continuação do projeto.



(Blank Page for Formatting)

## Table of Contents

Acknowledgements.....	iii
Abstract.....	v
Resumo .....	vii
Table of Contents.....	ix
Table of Figures .....	xii
Table Index .....	xiv
Abbreviations and Acronyms .....	xv
1. Introduction.....	1
1.1.Context and Motivation .....	1
1.2.Problem Definition .....	1
1.3.Research Goals .....	2
1.4.Research Methodology .....	2
1.5. Achieved Results .....	4
1.6. Document Structure .....	4
2.Literature Review .....	5
2.1. Cloud Computing.....	5
2.1.1. Characteristics of Cloud Computing.....	5
2.1.2.Cloud Computing Services .....	6
2.1.3. Deployment Models.....	8
2.2. Cross-Platform Development .....	8
2.2.1. Advantages of Cross-Platform Development .....	9
2.2.2. Disadvantages of Cross-Platform Development.....	10
2.3.Responsive Web Design .....	10

2.3.1. Media Queries .....	11
2.3.2. Flexible Grids .....	11
2.3.3. Flexible Images and Media.....	11
2.4. Low-Code Development Platforms .....	12
2.5. OutSystems Development Platform.....	13
2.5.1. OutSystems Tools and Components .....	15
2.5.2. OutSystems as a Business.....	17
2.5.2.1. OutSystems’ Strengths and Advantages .....	17
2.5.2.2. OutSystems’ Weaknesses and Cautions .....	19
3. GSI Application Requirements .....	21
3.1. Functional Requirements .....	21
3.2. Non-Functional Requirements .....	28
4. Architecture Framework .....	31
4.1. Application Model .....	31
4.1.1. Domain Model .....	31
4.1.2. Use Cases .....	31
4.1.3. Database Structure .....	33
4.2. Architecture .....	40
4.3. Logical Processes.....	44
4.3.1. Login Screen .....	44
4.3.2. Dashboard Screen .....	46
4.3.3. New Case .....	50
4.3.4. Case Edit .....	54
4.3.4. Independent Processes .....	56

4.3.4.1. CaseUpdate Process .....	56
4.3.4.2. Upload a File Process .....	58
4.3.4.3. AddComment Process.....	58
5. System Prototype .....	61
5.1. General description .....	61
5.2. User Interface.....	61
5.2.1. Responsive Interface.....	61
5.2.2. Screen Flow .....	65
5.2.2.1. Login Screen .....	65
6. Conclusion .....	71
Bibliography .....	73
Appendix.....	75
A. OutSystems Best Practices .....	75
1) Standard Language .....	75
2) JavaScript, CSS and HTML .....	75
3) Performance.....	78

## Table of Figures

Figure 1 - DSR Cycles extracted from (Hevner, 2007) .....	3
Figure 2 - Key differences between cloud computing service models extracted from (Watts & Raza, 2019) .....	7
Figure 3 - Magic Quadrant for Enterprise Low-Code Application Platforms extracted from (Gartner 2019).....	18
Figure 4 - Domain Model .....	31
Figure 5 - Use case diagram .....	32
Figure 6 - Entity Diagram (From physical Database Diagram).....	33
Figure 7 - 4 Layer Canvas extracted from (OutSystems, 2019) .....	40
Figure 8 - Extended 4 Layer Canvas extracted from (OutSystems, 2019).....	41
Figure 9 - Architecture Iteration extracted from (Hevner, 2007) .....	42
Figure 10- Green Store Initiative Modules .....	43
Figure 11 - Green Store Initiative's 4 Layer Canvas.....	43
Figure 12 - Login Screen .....	44
Figure 13 - Login Process.....	45
Figure 14 - Login Logic.....	45
Figure 15 - Dashboard Screen .....	46
Figure 16 - Role Verification Process.....	47
Figure 17 - Role Attributes .....	47
Figure 18 - Get Workload Process.....	48
Figure 19 - Get Daily Activities Process .....	48
Figure 20 - Dashboard Breakdown.....	49
Figure 21 - New Case Screen .....	50

Figure 22 - Autocomplete Process.....	51
Figure 23 - New Case Process .....	52
Figure 24 - Case Edit Screen .....	54
Figure 25 - Save and Delete Logical Process .....	55
Figure 26 - CaseUpdate Process Segment .....	56
Figure 27 - Upload a File Logical Process .....	58
Figure 28 - AddComment Process.....	59
Figure 29 - Responsive View - Browser.....	62
Figure 30 - Responsive View – Tablet .....	63
Figure 31 - Responsive View - Mobile.....	64
Figure 32 - Screen Flow – Login .....	65
Figure 33 - Screen Flow – Dashboard .....	66
Figure 34 - Screen Flow - Pickup Staff Dashboard .....	66
Figure 35 - Screen Flow - New Cases .....	67
Figure 36 - Screen Flow - Working Cases.....	67
Figure 37 - Screen Flow - Waiting Cases .....	68
Figure 38 - Screen Flow - Case screen .....	68
Figure 39 - Screen Flow – History.....	69
Figure 40 - Screen Flow - Edit Case.....	70
Figure 41 - Future App Concept .....	72

## Table Index

Table 1 - Functional Requirements - Customers .....	22
Table 2 - Functional Requirements - Waste collection staff .....	23
Table 3 - Functional Requirements - Office staff .....	24
Table 3 - Functional Requirements - Office staff (Continuation) .....	25
Table 4 - Functional Requirements - System.....	25
Table 4 - Functional Requirements – System (Continuation) .....	26
Table 4 - Functional Requirements – System (Continuation) .....	27
Table 5 - Non Functional Requirements .....	28
Table 5 - Non Functional Requirements (Continuation) .....	29
Table 6 - User Entity.....	34
Table 7 - Contact Entity .....	35
Table 8 - Company Entity.....	36
Table 9 - Case Entity .....	37
Table 10 - Status Entity .....	38
Table 11 - Priority Entity .....	38
Table 12 - Type Entity .....	39
Table 13 - FAQ Entity .....	39

## **Abbreviations and Acronyms**

DSR – *Design Science Research*

RG – *Research Goals*

GSI – *Green Store Initiative*

DB – *Database*

FAQ – *Frequently Asked Questions*

4LC – *4 Layer Canvas*

FR – *Functional Requirement*

NFR – *Non Functional Requirement*

UI – *User Interface*

UX – *User Experience*

SaaS – *Software as a Service*

PaaS – *Platform as a Service*

IaaS – *Infrastructure as a Service*

AWS – *Amazon Web Services*

RWD – *Responsive Web Design*

CSS – *Cascading Style Sheets*

PX – *Pixel*

AD&D – *Application Development and Delivery*

IDE – *Integrated Development Environment*

SOAP – *Simple Object Access Protocol*

REST – *Representational State Transfer*

MABS – *Mobile Apps Build Service*

API – *Application Programming Interface*



*QA – Quality Assurance*

*3GL – Third-Generation Language*

*B2C – Business-to-Customer*

*AI – Artificial Intelligence*

*IoT – Internet of Things*

*LCAP – Local Control Accountability Plan*

*SQL – Structured Query Language*

*HTML – Hypertext Markup Language*

*URL – Uniform Resource Locator*



## 1. Introduction

### 1.1.Context and Motivation

Nowadays waste management and recycling has come more and more into the limelight. People are increasingly focusing more on the importance of the environment and, most importantly, it's conservation. Because of this, many companies and businesses are focusing more efforts into implementing green alternatives to existing solutions as well as attempting to reduce waste and environmental damage caused by existing solutions.

As mentioned earlier and additionally developed in the next section, one of the biggest waste management companies in Portugal approached me with an idea to implement a web based service that would allow them to do just that: reduce their waste and environmental impact. Moreover, this service also serves to motivate businesses and individuals to recycle and separate their waste, as it removes the burden of having to carry the waste into the respective disposal containers.

### 1.2.Problem Definition

The company offers a door-to-door waste collection service for small businesses such as restaurants and small stores. This is an initiative that already exists. The way it is designed now works as follows:

The owner, manager or any other designated staff member, requests to join this service. Upon approval, they will then be added to the list and incorporated into the Green Store collection route. This route, though, is static, fixed. One, two or more collection days are chosen eg. Monday and Thursday. Then the collection trucks will come by the establishment every Monday and every Thursday, at the same time, in order to collect the waste. The issue with this is that often, the trucks will go by the establishment and none or minimal waste is collected. This results in a loss and a waste of time and resources and could have been avoided, had the company been provided with a way in which to communicate directly with the management.

And thus, this idea was born. The gist of the project is this: **Create an application, that will allow registered users to send waste collection requests to the waste management company when they need. This avoids wasteful situations, such as that of sending trucks to collect waste when there is none to be collected, and also helps the establishment with a quality of life improvement on their service, as it also avoids the overflow of waste due to unforeseen circumstances as well as saving time for both parties.**

Keywords: *Green, Garbage, Efficiency, Recycling, Collection, Route*

### 1.3. Research Goals

Based on the aforementioned problem and context, Research Goals (RG) were put in place in order to attempt to create a solution for the issue at hand, these were:

- **RG 1** – Perform a literature review in order to further investigate the availability of technologies that may be able to solve the issue, as well as deepen the understanding of topics that may be related to the project (such as cross-platform, responsive web design, OutSystems, etc.);
- **RG 2** - Determine the functional and non-functional specifications regarding the Green store initiative application;
- **RG 3** – Draft a Use Case model for the GSI (Green Store Initiative) application;
- **RG 4** – Create a domain model, identify and apply the necessary architectural processes and best practices for the app;
- **RG 5** – Develop and test a working prototype.

### 1.4. Research Methodology

The chosen Research Methodology for this project is Design Science Research (DSR).

DSR seems like an ideal choice for this project, as at its core, this relatively new approach to research has the goal of solving problems, instead of necessarily explaining an existing reality. According to Hevner A., Chatterjee S. (2010), there are seven Design Science Research Guidelines:

- **Guideline 1: Design as an Artifact** – Design science research must provide a viable artefact in the form of a construct, a model or an instantiation (Hevner A., Chatterjee S. 2010);
- **Guideline 2: Problem Relevance** – The objective of a design science research is to develop technology-based solutions to important and relevant business problems (Hevner A., Chatterjee S. 2010);
- **Guideline 3: Design evaluation** – The utility, quality, and efficacy of a design artefact must be rigorously demonstrated via well-executed evaluation methods (Hevner A., Chatterjee S. 2010);
- **Guideline 4: Research Contributions** – Effective design science research must provide clear and verifiable contributions in the areas of the design artefact, design foundations, and/or design methodologies (Hevner A., Chatterjee S. 2010);

- **Guideline 5: Research Rigor** – Design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artefact (Hevner A., Chatterjee S. 2010);
- **Guideline 6: Design as a search process** – The search for an effective artefact requires utilizing available means to reach desired ends while satisfying laws in the problem environment (Hevner A., Chatterjee S. 2010);
- **Guideline 7: Communication of research** – Design science research must be presented effectively to both technology oriented and management-oriented audiences (Hevner A., Chatterjee S. 2010).

According to Hevner A. (2007) there are three key Design Science Research cycles that are present in all DSR projects as can be seen in fig 1:

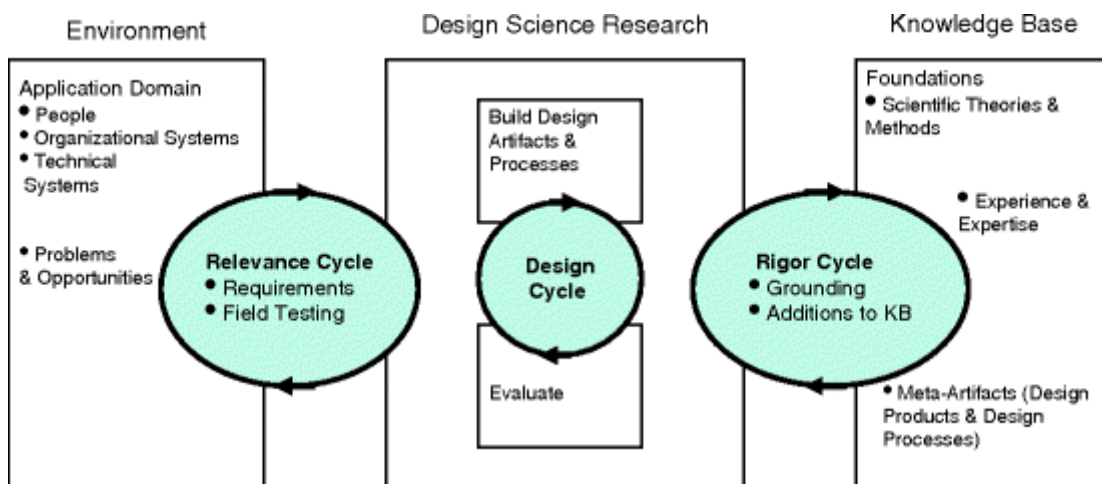


Figure 1 - DSR Cycles extracted from (Hevner, 2007)

With this figure in mind, we can identify the existence of three phases, or cycles, to follow in a DSR project.

- **The Relevance Cycle:** Any DSR project is born out of the desire to improve any given environment by introducing to them new and innovative artefacts or processes. According to fig 1, the Relevance cycle has its application domain on People, Organizational and Technical Systems. A good question to ask when designing an artefact using this method is: “Does this Design artefact improve the environment in which it is inserted into? And how can we measure this improvement?” (Hevner A., Chatterjee S. 2010);
- **The Rigor Cycle:** The rigor cycle provided past knowledge to the research project to ensure its innovation (Hevner A., Chatterjee S. 2010). It is the duty of the researcher to make sure that the designs produced are in fact innovative and not just routine designs based on well-known processes. This is clearly seen when a researcher

skilfully selects and applies appropriate methods and theories for constructing and evaluating an artefact;

- **The Design Cycle:** The design cycle is where the hard work of design science research is done. At the centre of fig 1 the design cycle borrows from both the Relevance and Rigor Cycles. A balance is needed between designing and evaluating an artefact, and for this to be successfully done, both Rigor and Relevance need to constantly be at the forefront of one's mind (Hevner A., Chatterjee S. 2010).

## **1.5. Achieved Results**

With the project we were able to identify an environmental issue that could be improved, namely, reducing waste and incentivising recycling and a more efficient waste management system for stores and restaurants. In addition to this, with regards to the prototype, we were able to:

- Implement an architectural framework that not only follows the best practices of this sort of development as recommended by the OutSystems' documentation, but also future-proof the prototype, thus making it scalable, easily modifiable and easily integrated in other environments;
- Implement a working prototype that fulfils all of the initial goals and readily have it available for all users across all devices.

## **1.6. Document Structure**

This paper is organized in six main chapters. The first one discusses the introductory concepts such as context and motivation, problem definition and highlights the goals that are to be achieved throughout the rest of the document. In chapter two we can find a literature review, where relevant concepts regarding the completion of the project are discussed in depth. Chapter three discusses the functional and non-functional features and specifications for the GSI app. Chapter four contains a study in the architectural framework concept for the GSI app, it's also where we can find the Use Case Model, Domain Model, an overview of the Database structure and some of the main logical processes behind the actions that can be performed in the GSI app. Chapter five showcases the final working prototype, here we can see all the major screens, what can we expect to find in them and how to connect and interact with each other. Finally, the sixth chapter presents the preliminary conclusions and plans for future development of the concept.

## 2.Literature Review

This chapter discusses and describes concepts and technologies that were used and are relevant to the project that has been developed, they are important for an enhanced understanding of the work and the theoretical and practical matters around it.

### 2.1. Cloud Computing

Cloud computing is based upon the delivery of computing services, such as servers, storage or software over the internet (“the cloud”). It is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can rapidly be provisioned and released with minimal management effort or service provider interaction ( Mell & Grance, 2011). A fundamental concept behind cloud computing is that the location of the service, in addition to details such as hardware or operating system on which it is running, are largely irrelevant to the user.

According to Peter Mell and Timothy Glance (2011), the cloud model is composed of five essential characteristics, three services and four deployment models.

#### 2.1.1. Characteristics of Cloud Computing

According to Mell & Glance (2011), the five essential characteristics of the cloud model are:

- **On-demand self-service.** A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider;
- **Broad network access.** Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms such as mobile phones, tablets and laptops;
- **Resource Pooling.** The provider’s computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Because of this, there is a sense of location independence, as mentioned earlier, where the user has no control or even knowledge over the exact location of the provided resources;
- **Rapid elasticity.** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward depending on the demand of said service. This results in the available capabilities of the service to often seem unlimited to the end user as they can thus be appropriated at any quantity at any time;
- **Measured service.** Cloud systems automatically control and optimize resource use by leveraging a metering capability as some level of abstraction appropriate to the type of service. Therefore, resource usage can be monitored, controlled and reported, providing transparency for both the provider and consumer of the utilized service.

In addition to these five characteristics, various authors mention other points that can be associated with Cloud computing, such as:

- **Pay-per-use Pricing** – Cloud computing providers often charge their users by their expenditure in a per-second model. IT usage of a company is often not static; therefore, this model saves companies money as they only pay for what they use;
- **Resiliency** – Cloud providers often use several techniques to guard against downtime, such as minimizing regional dependencies to avoid single points of failure. Users also have the option of extending their workloads across availability zones which have redundant networks connecting multiple data centers in relative proximity (Jones et al 2019).

### **2.1.2. Cloud Computing Services**

According to Mell & Glance (2011), there are three main service models present in the cloud model, Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Figure 2 illustrates the key differences between them compared to an On-Premises approach.



# Summary of Key Differences

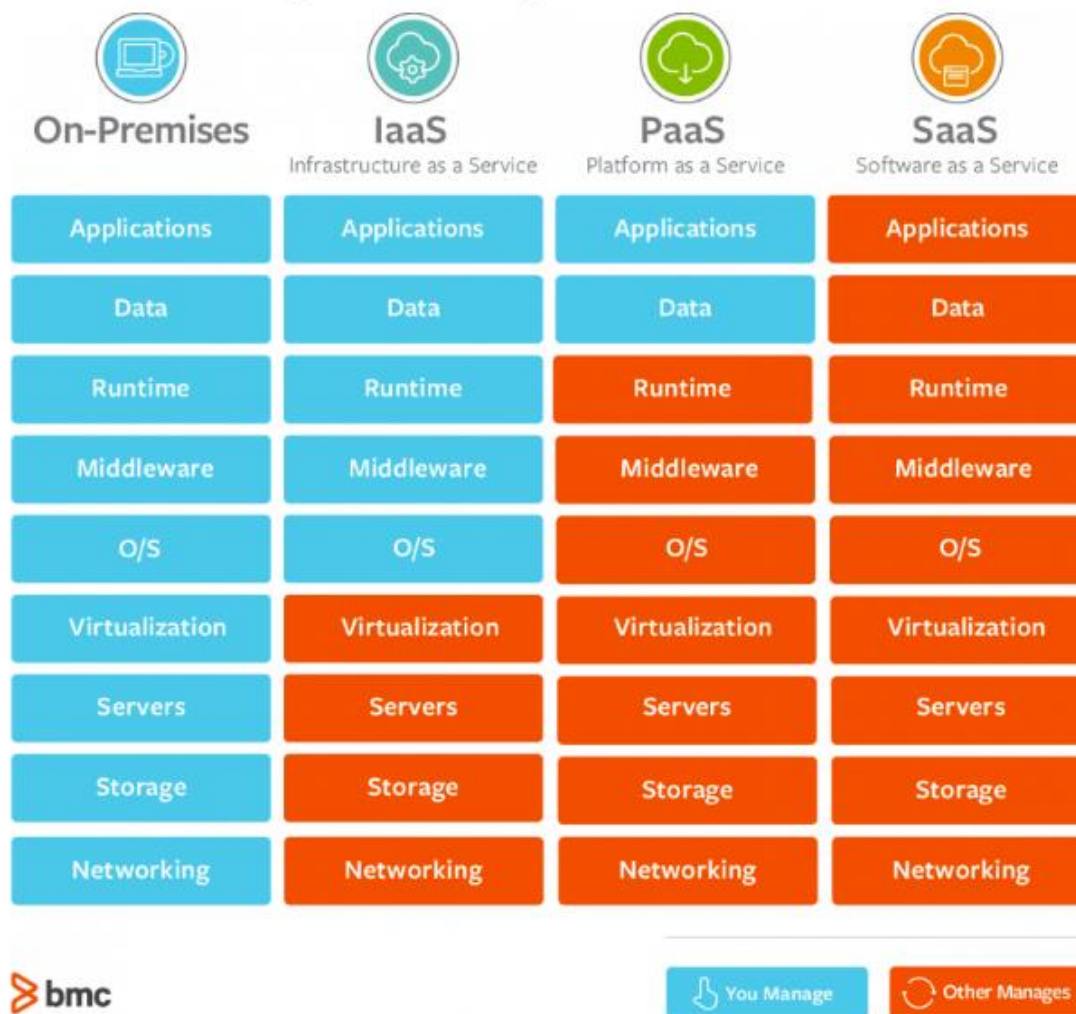


Figure 2 - Key differences between cloud computing service models extracted from (Watts & Raza, 2019)

- **Software as a Service (SaaS)** – The capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices (mobile phones, tablets, laptops, etc.) through a thin client interface, such as a web browser (for example, web-based email), or program interface. The consumer does not manage or control the underlying cloud infrastructure, this includes servers, operating systems or storage, with the possible exception of application configurable settings (Mell & Glance, 2011); Common examples: Google Apps, Salesforce, Dropbox, Cisco WebEx.
- **Platform as a Service (PaaS)** – The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using

programming languages, libraries, services and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment (Mell & Glance, 2011); Common examples: AWS Elastic Beanstalk, Windows Azure, Heroku, Apache Stratos.

- **Infrastructure as a Service (IaaS)** – It provides virtualized computing resources over the internet. In an IaaS model, a third party provider hosts hardware, servers, storage and other infrastructure components on behalf of its users in addition to also host user's applications and handle tasks including system maintenance backup and resiliency planning. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, data and deployed applications and possibly has limited control of select networking components such as firewalls (Mell & Glance, 2011). Common examples: Amazon Web Services (AWS), Cisco Metapod, Microsoft Azure.

### 2.1.3. Deployment Models

According to Mell & Glance (2011), there are four main deployment models for a cloud service:

- **Private Cloud** – The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers. It may be owned, managed and operated by the organization, a third party, or some combination of them, and it may exist on or off premises (Mell & Glance, 2011);
- **Community Cloud** – The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns. It may be owned, managed and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises (Mell & Glance, 2011);
- **Public Cloud** -The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider (Mell & Glance, 2011);
- **Hybrid Cloud** – The cloud infrastructure is a composition of two or more distinct cloud infrastructures that remain unique entities but are bound together by a standardized or proprietary technology that enables data and application portability
- (Mell & Glance, 2011).

## 2.2. Cross-Platform Development

The aim of any mobile development company to target as many users as possible by providing the same App for different platforms. Each platform vendor provides the developers with different Integrated Development Environments (IDEs), programming

languages, Application Program Interface (APIs), and Apps distribution market (store) (El-Kassas, Abdullah, Yousef & Wahba, 2017). Because of this scenario, a mobile development company must determine how it wishes to operate. The first alternative is to have developers work together in order to produce an App for each different platform, this option allows for the support of all the desired platforms simultaneously, but on the other hand increases development time exponentially as well as the overall cost of the project. Another option would be to divide all developers into teams that work specifically on each project. This allows greater focus and parallel development for all Apps, but may require more developers than the first option, resulting in even higher costs. There is always a focus and need in today's world to deliver an App or project as fast and cheap as possible. This problem led to the creation of cross-platform solutions.

The idea of cross-platform development is that a software application or product should work well in more than one specific digital habitat. As mentioned earlier, this capability is typically pursued in order to sell a product for more than one proprietary operating system, such as to accommodate use on both Android and Apple platforms (technopedia). Some of the fundamental strategies for cross-platform development include compiling different versions of the same program for different operating systems. In addition to this, another common technique is to create an abstraction at certain levels of the code in order to accommodate different software environments, software that follows this technique could be called “platform agnostic” in that it doesn't value support of one platform over another.

### **2.2.1. Advantages of Cross-Platform Development**

This section analyses some of the advantages we can expect, as developers, when choosing to use a cross-platform solution, these include:

- **Faster development** – With cross-platform solutions, developers mostly work with one codebase rather than many single platform projects. These codebases handle iOS and Android (possibly others as well), therefore, there is no need for separate projects. It achieves this, largely, by reusing a big part of an app's code between platforms, thus accelerating development.
- **Lower costs** – It is estimated that a cross-platform solution can be about 30% cheaper than building a native iOS and Android apps separately (Nitecki S. 2019).
- **Consistency and Uniformity** – Because the code is developed only once, the UI is consistent throughout all devices and operating systems. In native development, even in teams with great communication, there can be differences when attempting to implement the same function or UI pattern in different platforms.

### 2.2.2. Disadvantages of Cross-Platform Development

This section analyses some of the disadvantages we can expect, as developers, when choosing to use a cross-platform solution, these include:

- **Lower performance** – Performance is one of the most important characteristics of an app. Although it depends on many aspects, in general, cross-platform solutions tend to display less performance ratings than native options. This can happen because of the large number of redundant files that are often stored in the codebase in order to make sure the app really works in different environments. This can also occur due to the inconsistent communication between a device’s native and non-native components. All of this affects the overall performance of the app, regardless, these performance hits are often minimal and unnoticeable, especially in simple applications.
- **Long wait for new features** – Every time Google or Apple introduce a new feature for their Android or iOS platforms respectively, it is readily available to be integrated by native apps. This, on the other hand, does not happen with cross-platform solutions as it takes some time to update applications to support this new feature.
- **Poor User Experience (UX)** – Not all cross-platform solutions allow a developer to use native functionalities, meaning, for instance, in the case of an iOS device, swiping the screen from the left side to the right will perform a “back” action. This is a native functionality, some cross-platform solutions cannot implement these types of actions, this may cause some frustration to a user as he cannot perform certain native actions that he expects and is used to.
- **Integration Problems** – Much like the previous point, it is not easy to integrate cross-platform apps with local settings, preferences and notifications.

### 2.3. Responsive Web Design

Responsive web design (RWD), according to Ethan Marcotte (2011), lies in the concept of addressing the ever-changing landscape of devices, browsers, screen sizes and orientations by creating flexible, fluid and adaptive websites, opposed to the traditional (at the time) way of adapting ones website toward the most common screen resolution alongside a particular mobile version, the idea is to approach this issue using flexible and fluid layouts that adapt to any screen. For this, three core concepts are highlighted:

- Media queries and media query listeners;
- Flexible grid-based layouts;
- Flexible images and media, through dynamic resizing or CSS.

A truly responsive website requires that all three of these core principles be implemented.

Web responsiveness, though, is not based on resizing assets or grids in order to fit them into a screen. The key point is adapting to the user’s needs and device capabilities. It

means thinking about what a mobile user will require to see first when visiting a website and subsequently laying out the content accordingly, this may involve presenting information in a different order for different device sizes, changing fonts or interaction areas to respond better to a touch environment. All these factors influence responsive web design.

### **2.3.1. Media Queries**

A media query allows users to target not only certain device classes but to also inspect the physical characteristics of the device that will be rendering the application (Marcotte, 2011). Not only that, but media queries also allow designing pages for mobile devices with limited colour palettes, low resolution, and so on.

To further explain this, the following declaration is a typical media query using the @media rule:

```
@media screen and (max-width:400px) {div {border:none;}}
```

In this case, “screen” indicated the target media type, and max-width is the target media property. This declaration thus states that the specified rules (no border on div elements) are only to be applied when the page is displayed on a screen with a width of most 400 pixels (HTML5 – Responsive design, 2011).

### **2.3.2. Flexible Grids**

A flexible grid-based layout is one of the cornerstones of responsive design. The term “grid” doesn’t imply, however, a requirement to implement any of the available grid frameworks, rather, it refers to the use of CSS (Cascading Style Sheets) for positioning and for laying out margins and spacing, and for implementing various web layout types in a new way. These layouts are typically expressed in pixels (px). The issue with pixels is that one device, a single pixel can be one dot, but eight dots on another. For this reason, in order to achieve optimal web responsiveness, it is ideal to replace the use of pixels with percentages for resizing components. By basing text sizes, widths and margins on percentages, we can turn a fixed size into a relative size (HTML5 – Responsive design, 2011).

### **2.3.3. Flexible Images and Media**

The final aspect of responsive web design is flexible images and media. This feature allows a developer to adapt his images or other media to load differently depending on the device, either by scaling or by using the CSS overflow property.

Scaling CSS can be done by setting the media element's max-width to 100% (as opposed to using pixels for the same reason mentioned earlier), this way, the browser will make the image shrink and expand depending on it's container.

An alternative to scaling images is cropping them with CSS, for example, applying `overflow:hidden` allows us to crop images dynamically so that they fit into their containers as the containers resize to fit a new screen environment (HTML5 – Responsive design, 2011).

## 2.4.Low-Code Development Platforms

Forrester defines low-code development platforms as *products and/or cloud services for application development that employ visual, declarative techniques instead of programming and are available to customers at low- or no-cost in money and training time to begin, with costs rising in proportion of the business value of the platforms*. In addition to this, the OutSystems' documentation states that, Low-code is a software development approach that enables the delivery of applications faster and with minimal hand coding. Low-code development platforms (LCDPs) allow the creation of software through graphical user interfaces and configuration, as opposed to the traditional procedural computer programming. Often, these platforms focus on design and development of databases.

According to Gartner (2019), by 2024, three-quarters of large enterprises will be using at least four low-code development tools for both IT application development and citizen development initiatives. In addition to this, also by 2024, he states that low-code application development will be responsible for more than 65% of application development activity.

In his paper, Rymer (2017) states that, indeed, an immense growth can be seen when it comes to Low-code platform use, creation and implementation by big AD&D's (application development and delivery), he gives three main reasons for this:

- **They speed up application and innovation delivery** – A 2017 survey performed by Rymer indicated that out of the 41 AD&D leaders surveyed, most of them were satisfied with using LCDPs as it dramatically raises their team's ability to respond in time to demands for business software (Rymer 2017);
- **They prove useful for large-scale applications** – AD&D leaders who adopt low-code development platforms do so with big expectations. Detractors in this segment assert that low-code platforms are only good for "departmental" (code for small, noncritical) applications, but this usually isn't the case. Many adopters apply the platforms to applications used across the enterprise or by multiple departments (Rymer 2017);

- **They contribute to AD&D’s move to public clouds** – Adopting public cloud services, ranging from full applications to infrastructure services, remains a major goal of enterprises worldwide. As low-code vendors have built public cloud services accessible via self-service interfaces, these platforms have emerged as one of the three major platform-as-a-service (PaaS) options. And so, AD&D leaders will sometimes adopt low-code platforms as an avenue to broader public cloud services (Rymer 2017).

Further, when analysing low-code vendors in the market, Rymer (2017), four main aspects that make these platforms most valuable when used by AD&D pros, those platforms:

- **Cater to pro developers with rich sets of controls and deep features** – Developers adopting low-code platforms want dramatically higher productivity without sacrificing features that allow them to get under the hood if the application they’re building calls for it. The vendors in Rymer’s 2017 assessment generally serve pro developers by providing a rich array of UI controls; deep features in data management and reporting, integration, process automation, app management and scaling, and other aspects of AD&D; and extensive configuration options (Rymer 2017);
- **Risk overwhelming to citizen developer personas** – Feature richness for pro developers creates complexity challenges for people without development skills and backgrounds. Among citizen developers, business developers and power users will be overwhelmed by low-code platforms designed for AD&D. Line of Business (LOB) developers have a better chance of succeeding, but they’re AD&D pros in training. Recognizing this, several of the vendors evaluated by Rymer (2017) have created specific tools for citizen developers without development experience, whose output pro developers can incorporate into their projects (Rymer 2017);
- **Support developer-business expert collaboration in app projects** – Many enterprises adopt low-code platforms for use by their AD&D pros, but expect AD&D teams to invite collaboration by citizen developers and other business pros. All the platforms in Rymer’s (2017) report support this app-delivery approach well, making it easy for business experts and AD&D pros to test ideas together on the low-code platform (Rymer 2017);
- **Are capturing the lion’s share of revenue in the low-code platform market** – Forrester’s analysis of the low-code platforms market shows that vendors creating platforms for AD&D pros are in a dominant position. One or two of the vendors creating low-code platform for citizen developers have substantial revenues, but most are still small (Rymer 2017).

## 2.5.OutSystems Development Platform

OutSystems defines itself as *a low-code rapid application delivery platform that accelerates the delivery of mobile and web applications*. OutSystems’ platform allows developers to use a single, integrated development environment that covers the entire

development lifecycle: development, quality assurance (QA), deployment, monitoring and management.

Since the whole application is built on a single integrated development environment using a low-code approach, this platform includes development of front-end, back-end, database and integration of existing systems or services. It does this, for each of the aforementioned concepts, as described below:

- **Front-end** – It allows a developer to build the UI of his applications using templates and pre-built blocks that work well across all devices using its responsive qualities. In addition to this, a developer is not locked into these templates as he can use standard HTML, JavaScript, and CSS to extend the UI as well as accessing any of the devices capabilities (camera, calendar, GPS, etc.) by creating or importing simple plugins;
- **Back-end** – All aspects regarding the back-end are developed using a visual language, including APIs, web services, workflows, and business rules. Using custom code, a developer can have even further control and expand these capabilities even further than the standard platform capabilities, if he deems necessary;
- **Database** – OutSystems allows the user to model his application's database visually and/or connect to any existing data-source;
- **Integration** – The platform allows developers to use open-source connectors to connect to software packages, current apps, or existing databases. It also allows a developer to then visually consume or expose web-services or build his own connectors with custom code.

As mentioned earlier, OutSystems supports developers throughout the entire lifecycle of development, it does this by helping with:

- **Rapid development and integration** – It helps devs to quickly develop applications that are integrated with existing systems and databases. It ensures the quality of applications created by using impact analysis and a self-healing engine;
- **Deployment** – It allows us to move our applications all the way from development to production safely with a couple of clicks. Version tracking, dependency checking, and impact analysis help ensure total visibility on the impact of the staging process;
- **Monitoring** – It allows a developer to keep track of the health of his application by tracking client-side, server-side and network performance with applications that are automatically instrumented with monitoring capabilities;
- **Manage** – It helps developers to easily perform day-to-day operations on his applications. This includes identity management and configuration management of application services.



### 2.5.1.OutSystems Tools and Components

The OutSystems platform has environments, tools and components that cover the full applicational lifecycle management process. The OutSystems development and deployment environments are:

- **Visual development environment** – Also called “service studio”, this is the environment for creating all parts of the application stack (data model, application logic, UI, business process flows, integrations and security policies). In this environment, designers drag and drop visual elements to create components for their application. Here, applications also consume and expose SOAP and REST web services. All application layers have hook points so that developers can extend layers with their own code if they need to work outside the scope of the development environment. As mentioned in the previous section, this environment has a full reference-checking and self-healing engine that works behind the scenes to ensure applications are error-free and that any changes made won’t critically impact the applications. It is designed to be lightweight and forwards most of the post-design work to the platform server component;
- **Platform Server** – This server component is the core of the OutSystems platform. It generates, optimizes, compiles and deploys applications to a standard web application server. It takes care of all the steps required to generate, build, package and deploy applications. It has three main specialized services:
  - **Code Generator** – This takes the application modelled in the IDE (Integrated development environment) and generates native .NET code;
  - **Deployment services** – These services deploy the generated .NET application to a standard web application server;
  - **Application services** – These services manage the execution on scheduled batch jobs and provide asynchronous logging services to store events like errors, audits and performance metrics.

When deploying an application, the environment validates the application model that was created and sends the application model to the platform server, once it reaches, the server applications assigns a version to the application using its built-in version control system, compiles the code into native .NET code and optimizes it for performance and security, and finally, deploys it to all front-end servers.

- **Integration environment** – This is the environment for creating components to extend the OutSystems platform and to integrate with third-party systems and microservices. Developers can also use it to extend OutSystems with their own code. These components are deployed once and can be reused by all applications built with OutSystems. Developers use Visual Studio to code integration components and can take advantage of existing .NET libraries to do so.
- **OutSystems Now** – This is a native iOS and Android app that allows developers to test their mobile apps without having to create developer accounts and certificates, provisioning profiles, and generating a mobile app package to install on the devices. It provides instant access to apps, debug logs, app sharing and access to device capabilities. When changes are made in the app, they are instantly available in OutSystems Now. It is

ideal for debugging and doing quality assurance due to its debug logs that access all the information written in the console;

- **Mobile Apps Build Service (MABS)** – This OutSystems cloud service generates the native app package. It bundles all the static resources, plugins and generated app code into a native package and signs it with the provided certificates and provisioning profiles for the target platform. It removes the need to worry about certain aspects of mobile deployment such as SDK and API versions, build tool versions, etc. MABS abstracts all this complexity and is fully integrated into the OutSystems deployment and 1-click publish process.

The OutSystems management consoles are:

- **Operational management console** – This console manages the operational aspects of an environment, such as connection strings, web service end-points and application properties.
- **Deployment and environment management console** – This console enables centralized management of all development, QA and production environments, from staging apps between environments and monitoring their performance to IT team permissions. This web console automates DevOps processes so individuals and teams can stage applications from development to production. It can manage all environments, no matter if they're on the cloud, on-premises or in a hybrid configuration.

Other notable OutSystems tools and services include:

- **Forge** – This repository houses open-source modules, connectors and components that can be used in OutSystems projects. It contains UI components, libraries and connectors. It aims to help developers speed delivery by providing existing modules that can be reused in applications. Furthermore, due to its integration with the development environment, these components are highly adaptable and modifiable. Most of the components in the Forge are actually built and shared by the OutSystems community, moreover, trusted components go through a curation process by OutSystems community experts to ensure they deliver the promised functionality and follow best practices for security, documentation and code quality. In addition to these community components, there are also supported components, these are the ones OutSystems itself provides maintenance and support to.
- **Architecture Dashboard** – A technical debt monitoring tool that enables quick identification and repair of technical debt. It automatically performs code and runtime performance analysis before recommending solutions to help improve the performance, security, architecture and UX of applications.

## **2.5.2.OutSystems as a Business**

Rymer (2017) describes OutSystems as a *strong platform, big ambitions*. It is one of the founders of the low-code platforms market and continues to expand its features into new areas of business applications, including sensor and real-time data and AI. OutSystems has a very mature and rich public cloud service, including an extensive sharing marketplace and developer community (Rymer 2017).

### **2.5.2.1.OutSystems’ Strengths and Advantages**

When discussing OutSystems’ strengths, Forrester (2017) says: “OutSystems has strengths across the board, including in strategy. The platform’s tooling to support professional AD&D teams in excellent – broad in application and portfolio management, platform and security administration, and development-process management and deep in troubleshooting and performance management. OutSystems’ strengths make it an oft-mentioned choice by clients and contribute to its high growth rate and expanding roster of customers.”.

Gartner (2019) provides a figure that compares several low-code platforms with each other by dividing them in a so called “Magic Quadrant”, he rates each platform in two metrics, completeness of vision and ability to execute. Keeping those two metrics in mind, he categorizes each platform in one of four categories: Niche Players, Visionaries, Challengers and Leaders. Figure 3 show this quadrant.



Figure 3 - Magic Quadrant for Enterprise Low-Code Application Platforms extracted from (Gartner 2019)

From figure 3, we can see that Gartner places OutSystems as not only a market leader, but one of the top leaders. He states a few strengths that helped him to give OutSystems this favourable rating, those are:

- **Overall viability** – OutSystems continues to grow strongly by providing an alternative to many older third-generation language (3GL) platforms, while enabling low-code productivity improvements with access to capabilities such as an advanced web a mobile user experience, B2C support and even batch processing. Reference customers stated that they selected OutSystems for its strong vision and innovation (Gartner 2019);
- **Product Strategy** – OutSystems supports both AWS and Azure deployments, but it particularly interesting to Microsoft shops due to its underlying use of .NET. Both AWS and Azure versions support access to the underlying cloud services for the IoT (internet of things), AI and analytics (Gartner 2019);

- **Innovation** – OutSystems’ multipersona developer experience encourages professional and line-of-business developers to participate in governed AI-assisted application development using web-based and fat-client integrated development environments; deployment can be container-bases. Over 2,500 components are available on OutSystems’ Forge app store, and reference customers’ usage of shared components in this ecosystem was among the highest (Gartner 2019);
- **Customer experience** – Reference customers scored OutSystems higher than the average for developer productivity and most critical LCAP (local control accountability plan) capabilities. They also scored it above the average for modernizing the user experience of existing applications.

### 2.5.2.2.OutSystems’ Weaknesses and Cautions

Based on Rymer (2017) and Gartner (2019), OutSystems seems to be a strong contender for any developer or business wishing to dive into low-code platforms. But both these authors mention a few weaknesses and cautions to keep in mind regard OutSystems.

According to Rymer (2017), OutSystems’ weaknesses are in specific tools for citizen developers, cloud security certifications and to some degree its partner ecosystem.

Gartner (2019), mentioned four specific areas he calls “cautions” when it comes to OutSystems, those are:

- **Product** – OutSystems’ LCAP is often used for automation of business processes, but is not yet competitive in all aspects of process-centric application development, especially with regard to aspects such as complex business logic modelling (Gartner 2019);
- **Market understanding** – Outsystems tend to be favoured by professional developers, rather than citizen developers. Reference customers were less satisfied with its LCAP’s ease of use for citizen development (though citizen development was not a goal of any of them) (Gartner 2019);
- **Sales strategy** – OutSystems’ customers need to ensure its pricing remains acceptable as their usage increases. The vendor’s pricing model uses a blended metric (“application objects,” which measures user experience pages, data tables, and API or service operations) to gauge consumption and therefore price, which clients can find it difficult to predict. In addition, increasing B2C usage is forcing increasing adoption of an unlimited user-count pricing model (Gartner 2019);
- **Cloud strategy** – OutSystems’ reference customers scored it lower than the survey average for cloud functionality. Its application PaaS credentials for larger deployments have also received some criticism from Gartner clients. Additionally, surveyed reference customers requested continued improvement in the area of quality, reliability and availability (Gartner 2019).

(Blank page for formatting)

### **3.GSI Application Requirements**

As highlighted in the section 1.2, the aim of this project is to resolve a real world problem. For this, a few details regarding the project were a must, fundamental concepts that would prove to be the backbone of the entire prototype.

Due to the nature of the service and the deadlines that were established, this application must be available to all users, on all devices and all operating systems. Because of this, instead of making several apps, in different programming languages, a unified solution was preferred, therefore, the app must be cross-platform. Furthermore, the app would have to allow the administration staff to heavily control and manage most functionalities of the app and simultaneously restrict access to information for those who don't need it.

With all of this in mind, the solution developed lies on the creation of a responsive website using Outsystems' software. This website works in a similar way to a web application where the login credentials and roles of the user determine what information and screens are available to him.

With this being mentioned, the rest of this section will discuss the functional and non-functional features of the GSI app.

#### **3.1. Functional Requirements**

This section shows four different tables, each one representing the three user types (admin, customer and collection staff) and an additional one for system related functional specifications. These tables will list the functional requirements desired for each of these types. Every functional requirement (FR) will be given a code, a priority tag and a brief description. The priority for each FR can be: High (Crucial for the project), Medium (Important but not a top priority) and Low (Minor and optional).

Table 1 refers to the functional requirements of the application regarding the “Customer” user. These represent some of the interactions desired for this user to have with the application in addition to the permissions and features that he can enjoy.

**Table 1 - Functional Requirements - Customers**

<b>Code</b>	<b>Priority</b>	<b>Description</b>
<b>FR01</b>	High	Allow the User to perform a login operation.
<b>FR02</b>	High	Allow the User to make collection requests.
<b>FR03</b>	High	Allow the user to specify what type of waste he needs collective and how many bags.
<b>FR04</b>	High	Allow Users to receive emails from the staff.



Table 2 refers to the functional requirements of the application regarding the “Waste Collection Staff” user. These represent some of the interactions desired for this user to have with the application and all the features that are crucial for a successful fulfilment of his responsibilities.

**Table 2 - Functional Requirements - Waste collection staff**

<b>Code</b>	<b>Priority</b>	<b>Description</b>
<b>FR05</b>	High	Allow staff member to perform a login operation.
<b>FR06</b>	High	Allow staff member to view New Cases.
<b>FR07</b>	High	Allow staff member to view working cases.
<b>FR08</b>	High	Allow staff member to view waiting cases.
<b>FR09</b>	High	Allow staff member to assign cases to themselves.
<b>FR10</b>	High	Allow staff member to view information regarding each case.
<b>FR11</b>	High	Allow staff member to contact by phone or email the customer.
<b>FR12</b>	High	Allow staff member to close a case upon completion.
<b>FR13</b>	Medium	Allow staff member to add comments to a case.
<b>FR14</b>	Medium	Allow staff member to upload files to the case.
<b>FR15</b>	Low	Allow staff member to hide cases not assigned to him.

Table 3 refers to the functional requirements of the application regarding the “Office Staff” user. These represent some of the interactions desired for this user to have with the application and all the features that are crucial for a successful fulfilment of his responsibilities and administration duties.

**Table 3 - Functional Requirements - Office staff**

<b>Code</b>	<b>Priority</b>	<b>Description</b>
<b>FR16</b>	High	Allow office staff member to perform a login operation.
<b>FR17</b>	High	Allow office staff member to create new cases.
<b>FR18</b>	High	Allow office staff member to view New Cases.
<b>FR19</b>	High	Allow office staff member to view Working cases.
<b>FR20</b>	High	Allow office staff member to view Waiting cases.
<b>FR21</b>	High	Allow office staff members to assign cases to other users.
<b>FR22</b>	High	Allow office staff member to view information regarding each case.
<b>FR23</b>	High	Allow office staff member to contact by phone or email the customer.
<b>FR24</b>	High	Allow office staff member to close a case upon completion.
<b>FR25</b>	High	Allow office staff member to have access to a dashboard of important information.
<b>FR26</b>	High	Allow office staff member to edit cases and assign priorities to each case in particular.

**Table 3 - Functional Requirements - Office staff (Continuation)**

<b>Code</b>	<b>Priority</b>	<b>Description</b>
<b>FR27</b>	High	Allow office staff member to create new users for new customers.
<b>FR28</b>	High	Allow office staff member to add new companies to the database.
<b>FR29</b>	Medium	Allow office staff member to add comments to a case.
<b>FR30</b>	Medium	Allow office staff member to upload files to the case.

Table 4 refers to the functional requirements that the system should have in order to provide all users with an enjoyable and efficient experience.

**Table 4 - Functional Requirements - System**

<b>Code</b>	<b>Priority</b>	<b>Description</b>
<b>FR31</b>	High	The web application must have a header that allows each user to easily navigate from tab to tab.
<b>FR32</b>	High	The web application must determine the role of each user upon login and show them only the pages they are allowed to see.
<b>FR33</b>	High	The web application must store information regarding each case in the correct section according to their status.
<b>FR34</b>	High	When creating a new case, the application must not allow submission if there is important information missing from the input boxes (Name, Email, Type, etc.).

**Table 4 - Functional Requirements – System (Continuation)**

<b>Code</b>	<b>Priority</b>	<b>Description</b>
<b>FR35</b>	High	When creating a new case, the web application must show a error message if any data is incorrect or missing.
<b>FR36</b>	High	When viewing a case, the web application must display the information regarding the customer to which the case is associated with.
<b>FR37</b>	High	When viewing a case, the web application must display the information regarding the company to which the case is associated with.
<b>FR38</b>	High	When viewing a case, the web application must show the information regarding its type, priority and status.
<b>FR39</b>	High	When viewing a case, the web application must display the name of the staff member, if any, assigned to the case.
<b>FR40</b>	High	When viewing a case, the web application must provide a way to edit said case.
<b>FR41</b>	High	When viewing a case, the web application must provide a way to close said case.
<b>FR42</b>	Medium	When viewing a case, the web application must provide input fields in order to write comments.
<b>FR43</b>	Medium	When viewing a case, the web application must provide a button that allows a user to upload a file to be attached to that particular case.

**Table 4 - Functional Requirements – System (Continuation)**

<b>Code</b>	<b>Priority</b>	<b>Description</b>
<b>FR44</b>	Medium	When viewing a case, the web application must provide a way to contact the customer by email.
<b>FR45</b>	Low	When viewing a case, the web application must show a section showing the history of all actions performed on a case.
<b>FR46</b>	High	When editing a case, the web application must provide a way to assign said case to someone.
<b>FR47</b>	High	When editing a case, the web application must allow an authorized user to edit all fields of information for that case (Title and description).
<b>FR48</b>	High	When editing a case, the web application must allow the case to be assigned a priority and a status.
<b>FR49</b>	High	When editing a case, the web application must a user to cancel said case.
<b>FR50</b>	Low	When editing a case, the web application must show a section showing the history of all actions performed on said case.
<b>FR51</b>	Low	When viewing each of the tabs (New Cases, Working Cases and Waiting Cases), the web application must provide a search functionality to find specific cases by a keyword.
<b>FR52</b>	Low	The web application should have a Frequently Asked Questions (FAQ) tab.

### 3.2. Non-Functional Requirements

Table 5 displays information regarding the web application’s non functional requirements. Every non functional requirement (NFR) will be given a code, a priority tag, a category and a brief description. The priority for each NFR can be: High (Crucial for the project), Medium (Important but not a top priority) and Low (Minor and optional).

**Table 5 - Non-Functional Requirements**

Code	Priority	Category	Description
NFR01	High	Implementation	The web application must be available across as many devices and operating systems as possible.
NFR02	Medium	Usability	The UI must be simple enough that many different users with different levels of technology proficiency will be able to use it without difficulty.
NFR03	High	Usability	The UI must be <i>responsive</i> in order to fit all screen sizes.
NFR04	Medium	Reliability	The web application must be available everyday between 7am – 00am.
NFR05	High	Compatibility	The web application must be able to run on any browser.
NFR06	Low	Performance	After performing the login operation, the user must be redirected to the according page in less than 5 seconds.
NFR07	Medium	Performance	The web application must be as fast as possible, avoiding waiting times.

**Table 5 - Non Functional Requirements (Continuation)**

<b>Code</b>	<b>Priority</b>	<b>Category</b>	<b>Description</b>
<b>NFR08</b>	Medium	Performance	All new case requests should be instantly added to the database and displayed.
<b>NFR09</b>	Medium	Performance	The web application must be as light as possible in order to avoid strain on users that may have older devices.

(Blank Page for Formatting)



## 4. Architecture Framework

### 4.1. Application Model

#### 4.1.1. Domain Model

Figure 4 represents the domain model for the application. We can see the interaction between all the actors, the commercial entities (stores and restaurants) represented by an establishment owner of representative interacts with the collection officers that make up the collection service. In turn, these employees interact directly with the office staff that assigns them cases. All these make up the group “Users” that interact with each other through the application.

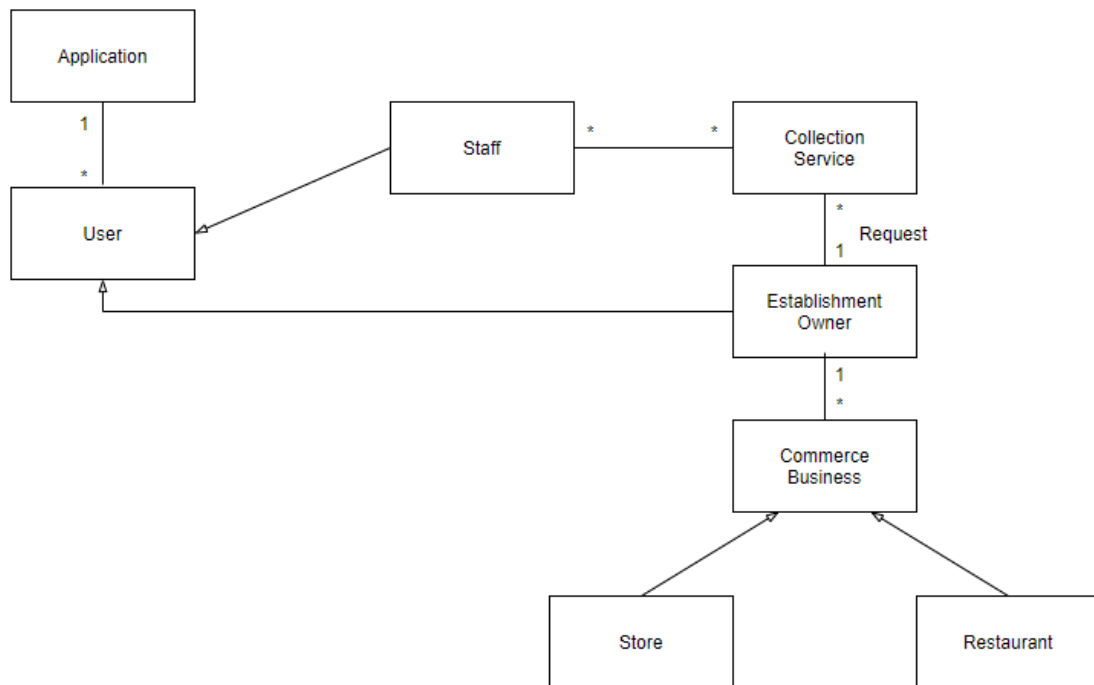


Figure 4 - Domain Model

#### 4.1.2. Use Cases

A use case diagram is a structured way of presenting information in which are specified concepts such as the expected behaviour of users, known as actors, are shown. But not the exact method of making it happen. They can be denoted in both textual and visual representation methods. In this case, this Use case model will take the visual form through the use of a diagram, as we can see below:

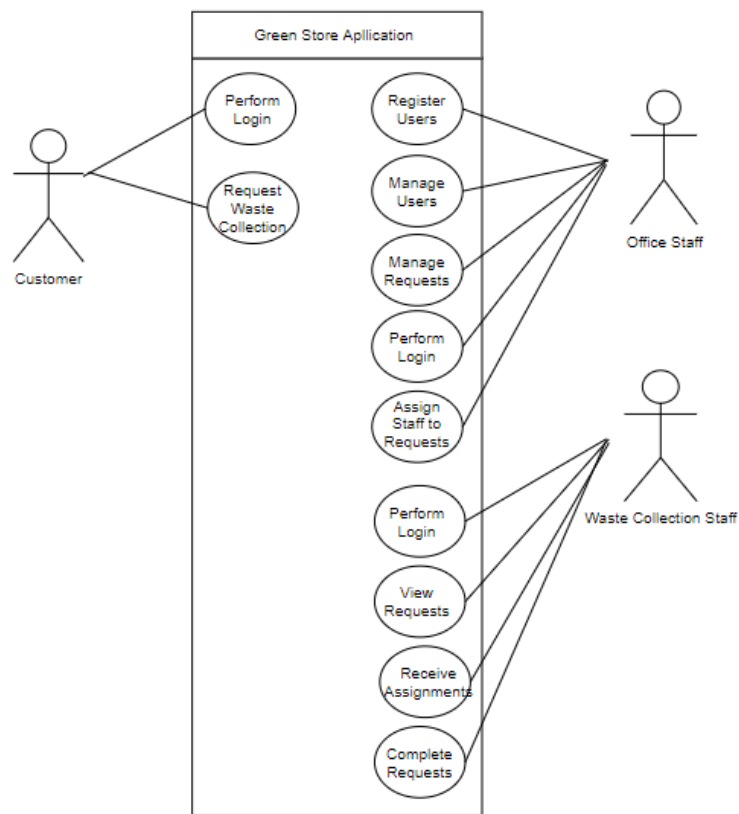


Figure 5 - Use case diagram

As visible in figure 5, we can identify three distinct actors:

**Office Staff** – This actor represents all the employees in the offices. These may have different roles from each other, each role holding a set of privileges, rights and duties within the flow of the app. These actors, as seen in the diagram, have the power to create users. There is no way for anyone to self-register on the database, this is done for security reasons, therefore, each user is created and registered by staff. They also have the ability to view and subsequently manage collection requests made by a user, these actions include assigning a priority to each ticket in addition to assigning them to a specific member of the collection staff;

**Waste Collection Staff** – These are the members of staff that go door-to-door collecting the waste from the establishments that requested it. They only have the ability to view requests assigned to them and then mark them as complete;

**Customer** – This represents the generic person that is part of the Green Store initiative. This person has limited access to the application and can only request new collections to their store.

### 4.1.3. Database Structure

This application was created in OutSystems, because of this, there is a great benefit when it comes to DB management as Service Studio (Outsystems’ development environment) automatically generates a relationship model between all the entities and their interactions with each other. Figure 6 shows the relationship model generated by OutSystems for the Green Store Initiative application. There we can see most of the entities, their respective attributes and how they behave and intertwine with each other.

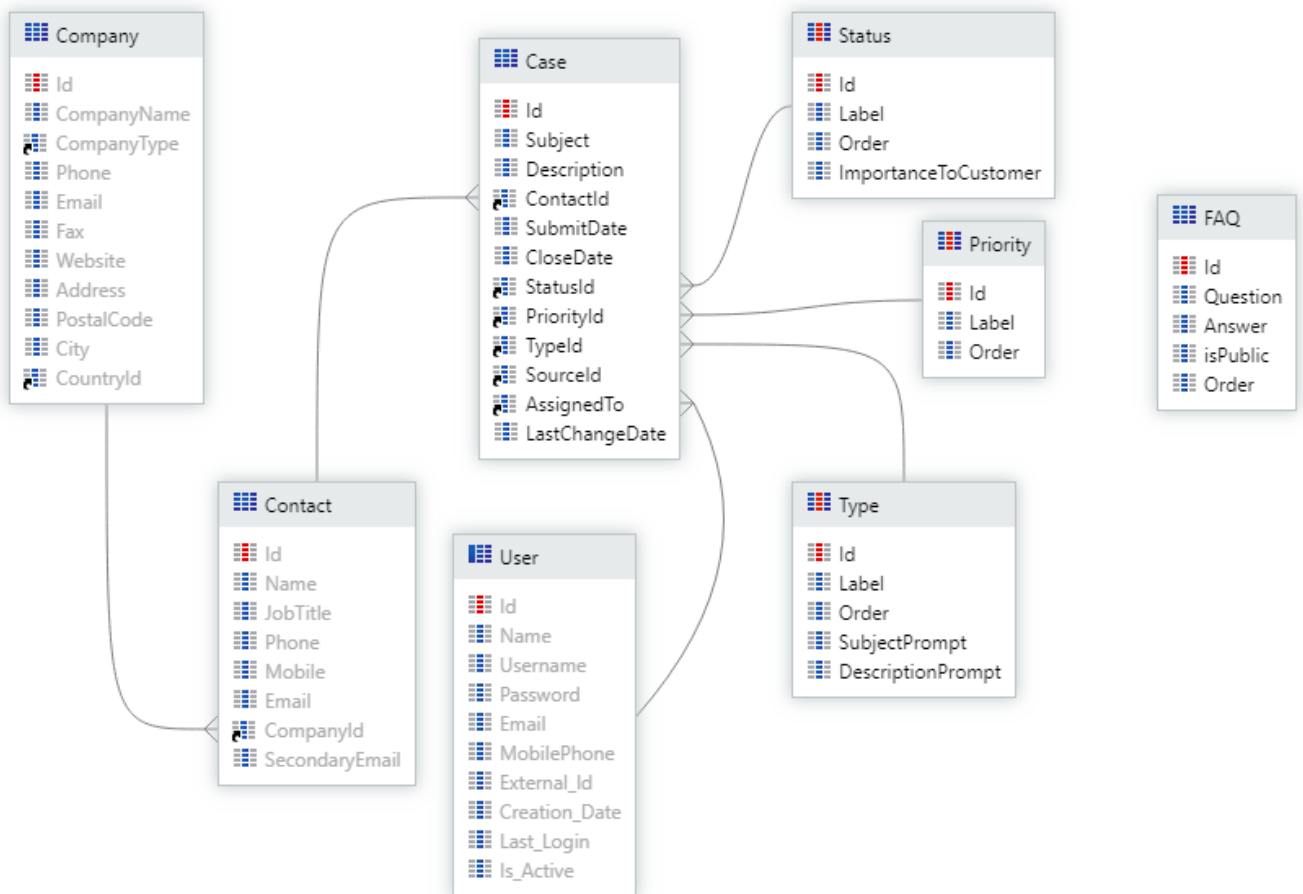


Figure 6 - Entity Diagram (From physical Database Diagram)

To visualize these entities, 8 tables were created:

Table 6 (Users) - The User entity represents all those that have access to the web application, where they be admins, customers or waste collection staff, this entity envelops all users. They interact exclusively with the “Case” entity, because of their roles, however, each user interacts with it differently.

**Table 6 - User Entity**  
**User (Every User registered on the platform)**

Attribute	Type	Description
Id	Integer	Unique identifier of the user.
Name	Text	Full name of the user.
Username	Text	Login name of the user.
Password	Text	Login password of the user.
Email	Email	Email contact of the user.
Mobile Phone	Phone Number	Mobile phone number of the user.
External_Id	Text	The user identifier in an external system to the Platform.
Creation_Date	Date Time	The date the user was created.
Last_Login	Date Time	Last time the user logged in.
Is_Active	Boolean	Indicates if the user is still active.

Table 7 (Contact) – This represents clients, those that have their establishments as a part of the GSI. They are directly associated with a company, which is another entity, borrowing their primary key to use as one of their own attributes. Each contact is therefore associated with a company and has the ability to interact with the “Case entity”.

**Table 7 - Contact Entity**

**Contact (An individual person from an external company)**

Attribute	Type	Description
Id	Integer	Unique identifier of the contact.
Name	Text	Full name of the contact.
Job Title	Text	Job title of the contact.
Phone	Phone Number	Phone number for the contact
Mobile	Phone Number	Mobile number for the contact.
Email	Email	Email for the contact.
CompanyId	Company Identifier	The Identifier for the company the contact is associated with.
SecondaryEmail	Email	The secondary email for the contact.

Table 8 (Company) – The company entity stores all information related to an establishment enrolled in the GSI, they interact exclusively with the “Contact” entity, giving them their Id as a foreign key, does allowing these two to complement one another.

**Table 8 - Company Entity**

**Company (The company part of the initiative)**

Attribute	Type	Description
Id	Integer	Unique identifier of the company.
CompanyName	Text	Full name of the company.
CompanyType	CompanyType identifier	Job title of the contact.
Phone	Phone Number	Phone number for the company
Email	Email	Email for the company.
Fax	Phone Number	Fax number for the company.
Website	Text	The website associated with the company
Address	Text	The address for the company.
PostalCode	Text	The postal code for the company.
City	Text	The companies' city.
CountryId	Country Identifier	The companies' country.

Table 9 (Case) – This is the main entity in the DB through which all others interact and operate. This entity interacts with the rest of the entities by borrowing keys from other entities, such as priority and status, and then presenting the information gathered, as a whole, to the respective user.

**Table 9 - Case Entity**

**Case (An individual waste collection request)**

Attribute	Type	Description
Id	Integer	Unique identifier of the case.
Subject	Text	The subject of the request.
Description	Text	The description of the request.
ContactId	Contact Identifier	The Id of the person which the request is associated with
SubmitDate	Date Time	Date the request was submitted.
CloseDate	Date Time	Date the request was closed.
StatusId	Status Identifier	The status of the request.
PriorityId	Priority Identifier	The priority of the request.
TypeId	Type Identifier	The type of request.
SourceId	Source Identifier	The source from which the request was made.
AssignedTo	User Identifier	The user the request is assigned to.
LastChangeDate	Date Time	Last time request was modified.

Table 10 (Status) – The “status” entity represents the status of the current case (Working, New or Waiting).

**Table 10 - Status Entity**

**Status (An individual waste collection request)**

Attribute	Type	Description
Id	Integer	Unique identifier of the status type.
Label	Text	The status' label.
Order	Integer	The order of the status.
ImportanceToCustomer	Integer	Unimplemented future functionality that allows a user to set his own priority.

Table 11 (Priority) – This entity stores the information regarding the priority of a given case (High, Medium, Low, or none, in case no priority has been assigned).

**Table 11 - Priority Entity**

**Priority (The priority of a given request)**

Attribute	Type	Description
Id	Integer	Unique identifier of the priority type.
Label	Text	The priorities' label.
Order	Integer	The order of the priority.



Table 12 (Type) – This entity stores the information regarding the type of collection being made, whether it be blue, yellow or green.

**Table 12 - Type Entity**

**Type (The type of collection being requested)**

Attribute	Type	Description
Id	Integer	Unique identifier of the type.
Label	Text	The type's label.
Order	Integer	The order of the type.
SubjectPrompt	Text	What is shown by default to the user for the Subject.
DescriptionPrompt	Text	What is shown to the user by default as a template for the Description of the request.

Table 13 (FAQ) – This is a standalone entity, it does not interact with any other entity, it serves the sole purpose of storing information regarding its own page, the FAQ page.

**Table 13 - FAQ Entity**

**FAQ (Information for the FAQ)**

Attribute	Type	Description
Id	Integer	Unique identifier of the type.
Question	Text	The question in the FAQ
Answer	Text	The answer to the FAQ question.
isPublic	Boolean	The value that shows if the question is public or not.
Order	Integer	The order of the FAQ.

## 4.2. Architecture

Based on the previous explanation found in section 2, OutSystems has a unique way to create, develop and structure applications, therefore, in the following sections we will be covering the specific components and specific information regarding the Green Store Initiative Application.

OutSystems themselves provide extensive documentation and guidelines regarding their recommendations for a clean, organized and efficient architecture, they call this the 4 Layer Canvas.

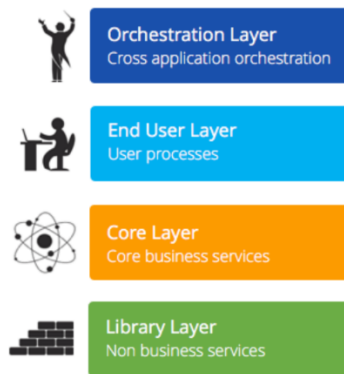


Figure 7 - 4 Layer Canvas extracted from (OutSystems, 2019)

The 4 Layer canvas (4LC) has the goal of simplifying the design of Service-Oriented Architectures (SOA). It promotes the correct abstraction of reusable services and microservices in addition to the correct isolation of distinct functional models.

When using this type of design, the applications and modules have independent lifecycles, therefore, have a minimum number of dependencies. Because of this, modules are more resistant to overall change, not in a bad way, but in a cost-effective way, as it requires little micromanagement when designing and implementing new features. This allows projects to be easier to maintain and also facilitates its vertical evolution.

As we can see in figure 7, the four layers suggested by OutSystems are:

**Orchestration Layer** – This is where dashboards and home pages are typically found. The goal of this layer is to provide any given user with the unified and centralized experience. Much like a maestro pulls together an orchestra, this layer pulls together the entire project in order to supply the user with the desired information;

**End User Layer** – This is where would find our UI. It reuses components from the Core and Library services to implement the user stories;

**Core Layer** – This is the “Brain” of the project, this is where we find services regarding business concepts, rules and widgets. It is also responsible for exporting reusable entities to the other layers;

**Foundation Layer** – The goal of this layer is to extend the framework with highly reusable assets, UI Patters, connector to external systems and integration of native code.

To better understand what these 4 layers do, OutSystems provides a figure that identifies what we could expect in each of these 4 layers.



**Figure 8 - Extended 4 Layer Canvas extracted from (OutSystems, 2019)**

It is extremely helpful to create the architecture beforehand, to think about it and design it carefully before developing any code-based features. But this doesn't mean that architecture design is shackled and exclusive to the first stages of a project. Designing the application's architecture is an ongoing process, something that requires constant attention and refining. It is a constant cycle of identifying new concepts and defining new modules. This can be visually represented as seen in figure 9.

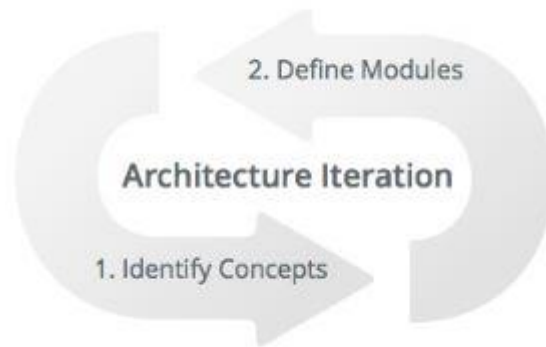


Figure 9 - Architecture Iteration extracted from (Hevner, 2007)

Keeping these aspects in mind, the Green Store initiative uses the following modules:

**Recolhas** -This is the main module of the application, where everything comes together (hence the little arrow on the left of the name, this shows it's the main module. Fig.10). In this module we can control and keep track of all customer interactions. This module is responsible for not only managing and creating new pickup requests, but also where the admins, staff and pickup staff operate. Instead of creating different modules for each user type, they share the same one, but their roles change what they are allowed to see, making it seem like multiple screens, but in reality, it is a shared module;

**Recolhas\_API** – This is where we can implement any API features that we may deem necessary. It is not currently in use, but might be needed later on;

**SampleData** – This module manages data; Here we can see all the registered users and their specific roles. OutSystems makes it quite easy to add and create new users, this module helps have that in a centralized location within the application. A screen only admins have access to;

**Emails** – This is a module with the sole purpose of email configuration for the contacts.

Figure 10 represents in a visual all the modules that are part of the Green Store Initiative Application

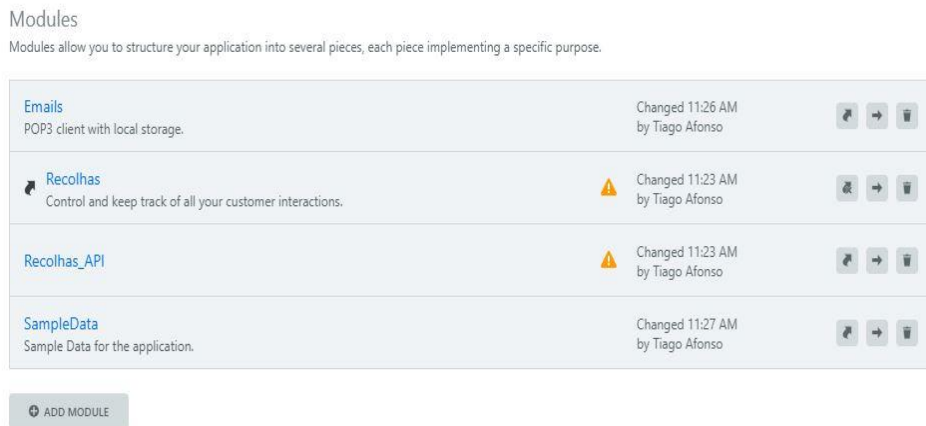


Figure 10- Green Store Initiative Modules

OutSystems offers a tool called Discovery, this tool allows a developer to select a certain application and then easily visualize all the architectural layers that it has, colour coded with the same scheme as we saw in the previous pictures. It also contains relevant information about the number of screens, web blocks, logical functions, entities etc. Figure 11 shows that very view regarding the Green Store Initiative Application.

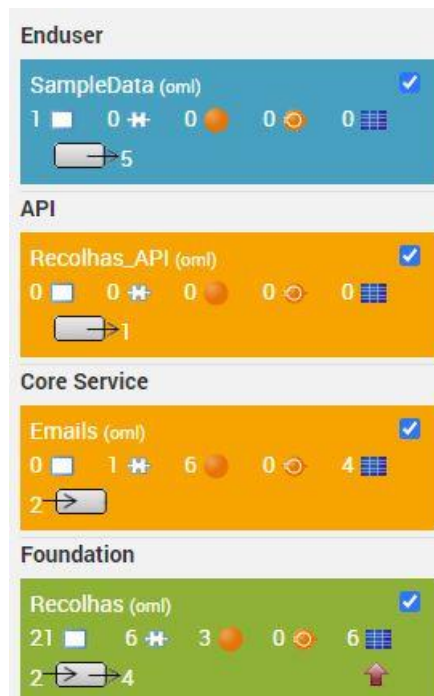


Figure 11 - Green Store Initiative's 4 Layer Canvas

From this figure we can thus discern that only 3 of the 4 layers were used. Because of the nature of the app and restricted access to its contents, the orchestration and end user layers are combined into one, for now. In the future, there might be a need to create these 2 layers separately, but this is further explained in the next chapter.

As we can see, the SampleData is the only one considered to be an End user layer. This is because it is the only module, besides the *Recolhas* module, that can be seen and interacted with independently from the others. It is an end user module, but only used by admins.

Next we have the two core service layer modules, the API module (not used, but because of the nature of APIs, being classified as a reusable component makes the system think it's a core service) and the Emails module. With all its reusable components.

And finally, we see the main module *Recolhas*. As we can see in the image, the 21 screens, 6 web blocks make this module the undisputed hub for all the application.

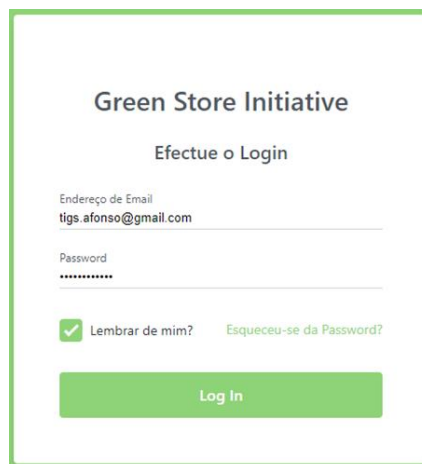
### 4.3.Logical Processes

This section discusses some of the main features that we may expect to encounter when using the GSI application and analyse the logical processes behind them, in a graphical way, as provided by the OutSystems platform.

#### 4.3.1.Login Screen

When first accessing the application, the first thing we find is the login screen. This is where a user can login (but not register) using the credentials provided to him.

Figure 12 - Login Screen



Green Store Initiative

Efectue o Login

Endereço de Email  
tigs.afonso@gmail.com

Password  
.....

Lembrar de mim? [Esqueceu-se da Password?](#)

Log In

Figure 13 - Login Process

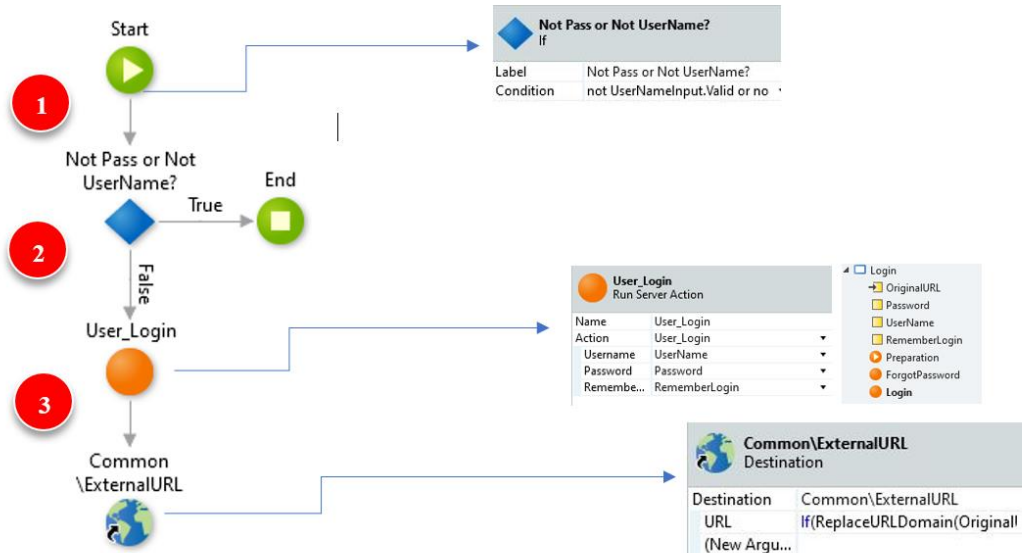


Figure 14 - Login Logic

- 1) The first step of the process is an *if* loop, it runs the command:

```
not UserNameInput.Valid or not UserNameInput.Valid
```

This checks if the Username and password are correct. If true, it proceeds to step number 2, if not, it end the process, prompting an exception (error message).

- 2) The second step happens if the user's credentials are correct, then it will perform the validation and authentication of the credentials, using the three local variables *Username*, *Password* and *RememberLogin*.
- 3) The third and final step is redirecting the user to the website, namely, it's home page.

### 4.3.2. Dashboard Screen

The main screen for this application is the dashboard, (fig 15) this is where the User is redirected to, if he has permission to view it. If he does not (for example customers), he will be redirected to *Novos Pedidos* page.

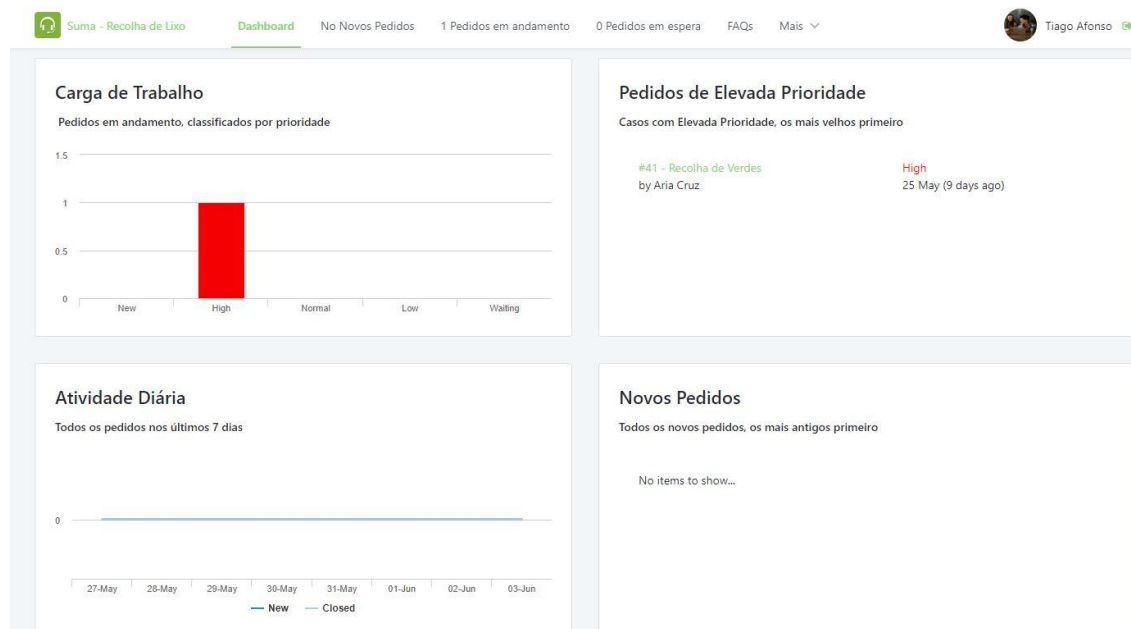


Figure 15 - Dashboard Screen

Figure 15 shows us the dashboard screen from the perspective of an admin user. Figure 16 identifies in a graphical way all the process behind the verification of the user's role following a login attempt and the subsequent redirection to the correct page containing the authorized widgets for each user role.



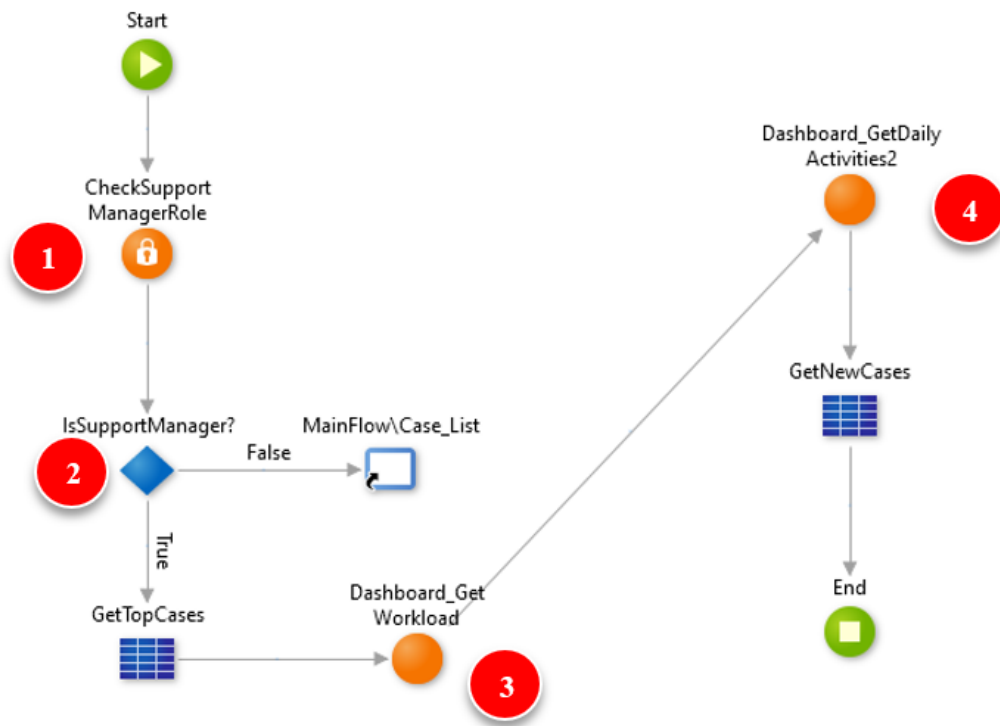


Figure 16 - Role Verification Process

As the user is redirected the system follows four steps:

- 1) It checks the user's Id and role;

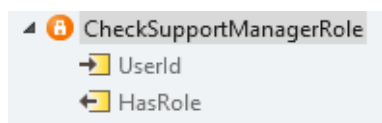


Figure 17 - Role Attributes

- 2) After step 1, an *If* loop is initiated, it runs the `CheckSupportManagerRole` script. If it is true, then it proceeds to loading the information allowed to that specific role, if not, it sends the user to the case list screen, where he only has access to the information from that screen;

- 3) Next, the systems loads all of the current workload for the company, this includes loading and displaying all New Cases, Cases with the Low, Normal and High priority assigned to them and the waiting cases;

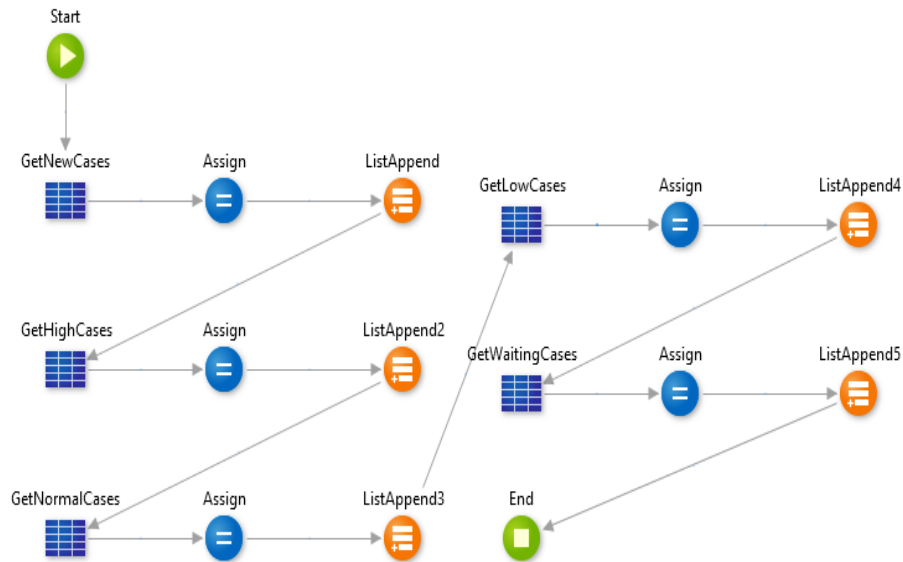


Figure 18 - Get Workload Process

- 4) Finally, the systems loads the cases for that specific day.

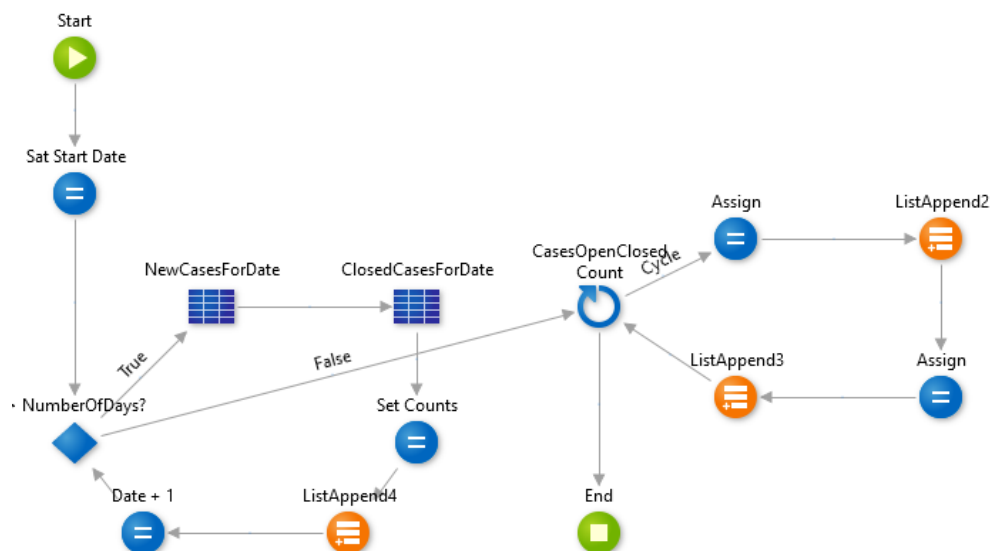


Figure 19 - Get Daily Activities Process

After these four steps are performed successfully, the dashboard screen is fully initiated with all its components, a complete view of this is seen in figure 20.

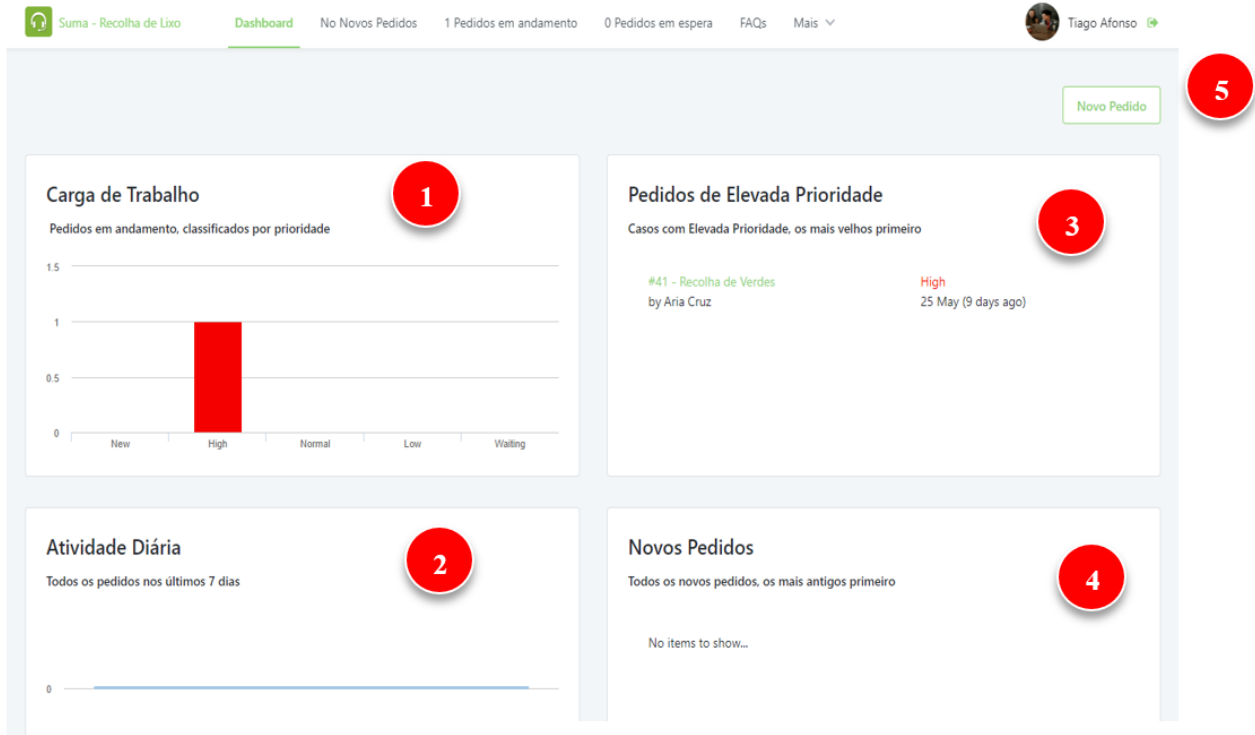


Figure 20 - Dashboard Breakdown

The dashboard contains a lot of useful information for an admin responsible for assigning pickup staff to each specific case, he has access to:

- 1) A widget that contains the overall workload, it shows in a simple graphical way the amount of New cases pending completion, those that are idle (waiting) and three extra bars to see all cases by their priority type;
- 2) An overview of the last 7 days, showing a daily total of cases per day;
- 3) A widget exclusively devoted to the high priority cases, with more details about them specifically, such as the title, and the date when it was made;
- 4) A widget that contains information regarding new cases only. Here the admin can check and assign new cases to pickup staff member;
- 5) A button that allows new cases to be logged;

### 4.3.3. New Case

This section explores the process and flow behind and for the New Case (*Novo Pedido*) button. Upon pressing said button (seen figure 20 under bullet point number 5), the following screen will be prompted.



The screenshot shows a web form titled "Submeter novo Pedido". The form is contained within a light blue header area. Below the header, there are several input fields: "Nome" with a dropdown menu showing "Type or double-click for list"; "Email"; "Telemóvel"; "Tópico"; "Descrição" which is a large text area; and "Tipo de Recolha" which is a dropdown menu with a "-" symbol and a downward arrow. At the bottom of the form are two buttons: "Submeter" and "Cancelar".

Figure 21 - New Case Screen

This screen allows a user to insert the name of the client in need of a pickup. The advantage of the current format using a single DB is that by inserting the name of a registered client, all the fields regarding his or her personal information are autocompleted.

Since only registered and verified users are part of the program, this system tends to work flawlessly and efficiently. The logic behind this functionality is seen in figure 22.

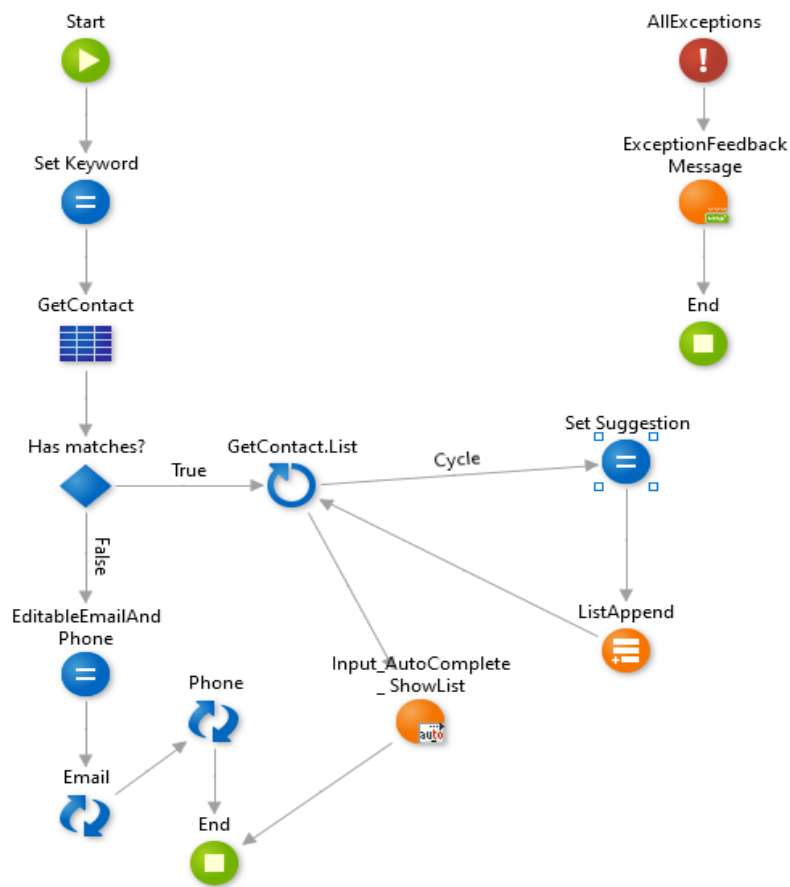


Figure 22 - Autocomplete Process

This process uses an *assign* to call the `GetContact_SearchKeyword` function and then uses the following “Has matches” if loop to see whether there is a suggestion or not, if true, it gets the contact list and updates it once the user has verified that the suggestion is the correct one for him.

Next we see some input fields regarding the type of collection to be made, here a user can write a title and a description if he so wishes, in addition, he is able to specify the type of waste he needs collecting, whether it be green, yellow or blue. This helps the waste management company to send the appropriate vehicle to collect that specific waste, since it is for recycling, one truck does not suit all requests.

Once all the details are all filled out as needed, the user then submits the Case, which prompts it to be created, the logic behind this is as follows:

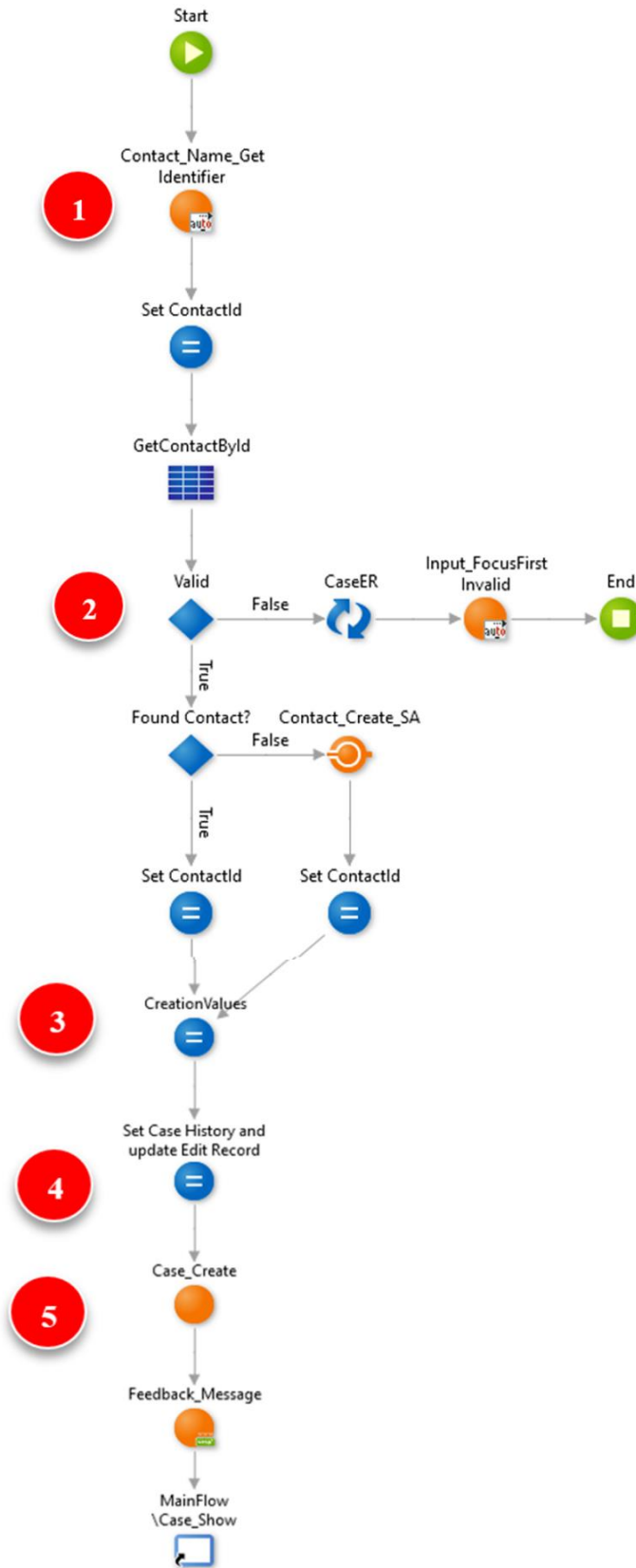


Figure 23 - New Case Process

- 1) The first step in this process is getting the autocomplete input, once this is done and confirmed, it will set a contact Id and get the contact details from the DB.
- 2) The next step is verifying that the information that was inserted is all correct, here, the system checks the data and ascertains if the information is correct or incorrect. If it is correct, the process continues to the “Found Contact?” if, where the system confirms that the user in fact exists. If the information is not correct, two scenarios are typically encountered. The first possibility is that the information is incorrect, as in doesn’t match the database (for instance, the name exists but the phone number doesn’t) this will prompt the user back to the field where the error was detected. The second scenario is that the entire user does not exist, and this is a tricky one to tackle. The way the current setup works makes it that when this occurs, the user is created anyway, and the case is created regardless. This makes sense for a few reasons, such as if the waste collection request was made by phone, from a registered business, but one of the staff members of said business, that isn’t part of the DB. On the other hand, it also provides a way to circumvent the need to register in order to make a request. This is something to be modified and changed at a later date in harmony with the desire of the waste management company when this system is implemented, for now, with the goal of having the most functional app possible, it remains how it is currently.
- 3) Here the system records and set the information regarding the case itself, such as the case submit date, it’s status and priority.
- 4) On this step, the final part of the setup is complete, it prepared the history of the case (seen somewhere else) and the Edit Record screen.
- 5) Once all is done, the case is created.

### 4.3.4. Case Edit

After the case is created, we have access to the most important screen of the entire application, the edit screen.

#41 - Recolha de Verdes

Name: Aria Cruz  
Email: aria.cruz@lexcorp.example  
Company: LexCorp

1 Designado a: -

Tópico: Recolha de Verdes

Descrição: 3 sacos verdes

2 Prioridade: Elevado

3 Status: Working

Tipo: Vidro

4 Source: Telefone

Guardar Apagar Cancelar

Figure 24 - Case Edit Screen

Here, the admin has access to four new components, not shown in the previous screen:

- 1) The assigned to button is the first one and one of the most important ones, here the admin can use a dropdown menu to assign this specific case to any of the pickup staff members.
- 2) The admin is also able to assign a priority to the case. He can choose from “low”, “medium, or “high”.
- 3) A status can be given to the request, “Working”, “Waiting”, “Closed”, “Discarded” and “New” are the available options, by default all cases are given the “New” status Id.
- 4) Here the admin can select how this request was made, he can choose from “Email”, “Phone” or “application”, this is useful for statistics and other analytical metrics.



Once the user presses “*Guardar*”, or “*Apagar*” it saves the case with the new information, or deletes it, respectively. The logic behind this can be seen in fig. 25.

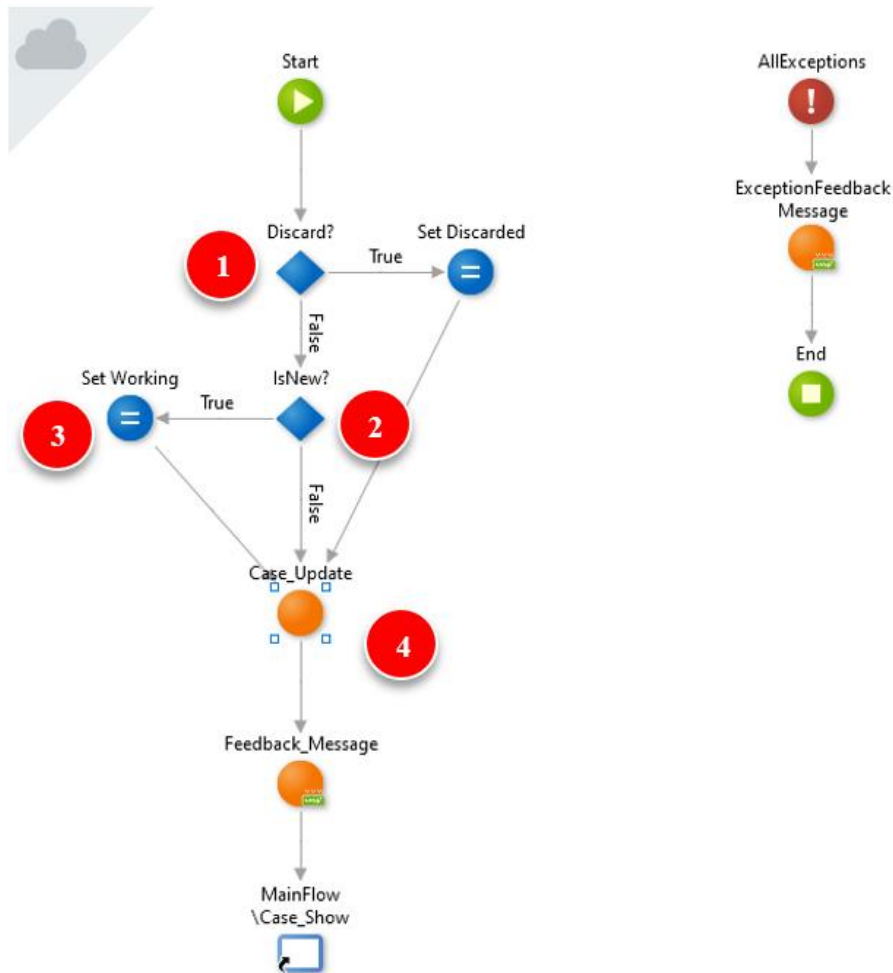


Figure 25 - Save and Delete Logical Process

The process behind the saving and deletion of a case in the Case Edit screen is largely a four-step process:

1. The system first checks what button was pressed, if it was the discard button, it sets the case as the “Discarded” type and goes to step 3, if not, it goes to step 2;
2. Here the system verifies if the case is “New” by checking it’s type , if in fact is it New, it will set its type as working (as the case has already been edited, therefore, it’s no longer considered “New” and untouched) and moves on to step 4. If the case was not set as “New”, it moves straight to step 4;

3. As mentioned previously, here the system checks the status type for the case and corrects it to display the “working” type;
4. The final step of the process, here the system updates the case with the new information.

### 4.3.4.Independent Processes

This section will showcase some of the processes that either are not clearly seen in the sections mentioned previously, or those that aren't part of the main processes, yet improve the UX.

#### 4.3.4.1.CaseUpdate Process

Certainly, one of the biggest processes of the application, the CaseUpdate is responsible for updating a case whenever any changes are done to it. It is used in all screens that require, or allow, a case to be modified. A segment of the process can be seen below in fig, 26.

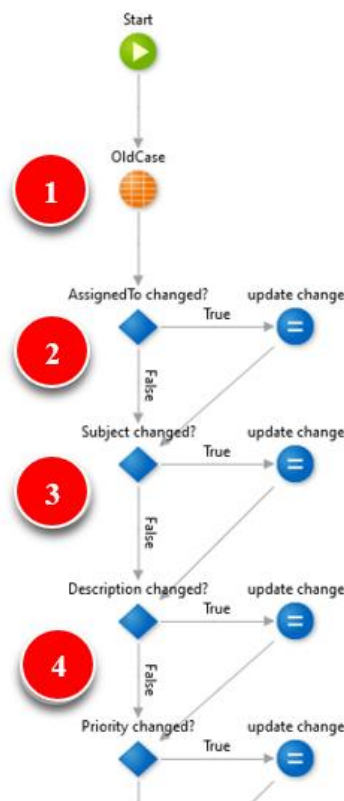


Figure 26 - CaseUpdate Process Segment

1. The first step this process is getting the case Id, the function is named “OldCase”.
2. Now begins a long line of change verification, the system goes through all the possible variables that can be modified and checks if the suffered change or not, the thing is checking if the case itself was assigned as “changed”, it does this by running the `OldCase.Record.Case.Assigned <> NewCase.Case.AssignedTo` command, if true, it updates, if false, continues to step 3.
3. Now the system verifies if the Subject was changed, it does this by running the `OldCase.Record.Case.Subject <> NewCase.Case.Subject` command, then, it either updates or moves on to step 4.
4. Step 4 checks if the priority was changed by running the `OldCase.Record.Case.PriorityId <> NewCase.Case.PriorityId` command.

The system then follows the same process for all the modifiable components (Status, Type, Source, Changes) until it goes through all of them. At the end it updated the case history and ends the process.

#### 4.3.4.2. Upload a File Process

The process to update the file can be seen in fig 27. Technically, the file is a comment, therefore, it follows the same process as the AddComment logical flow.

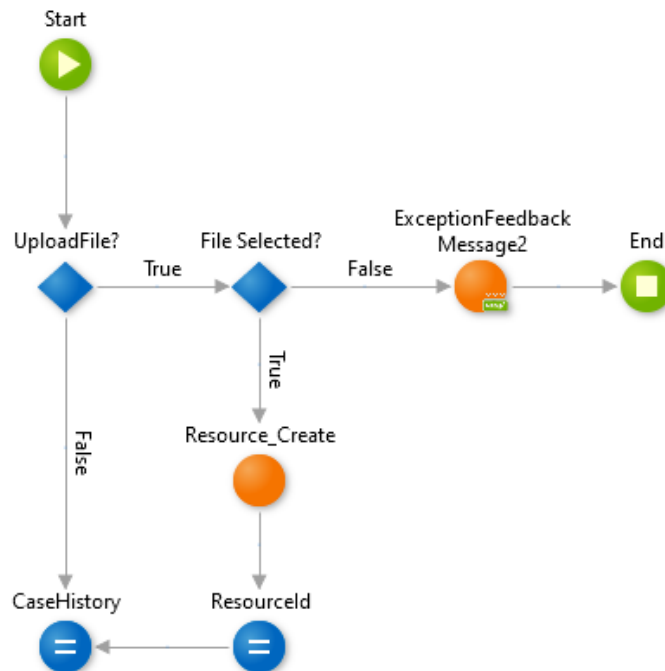
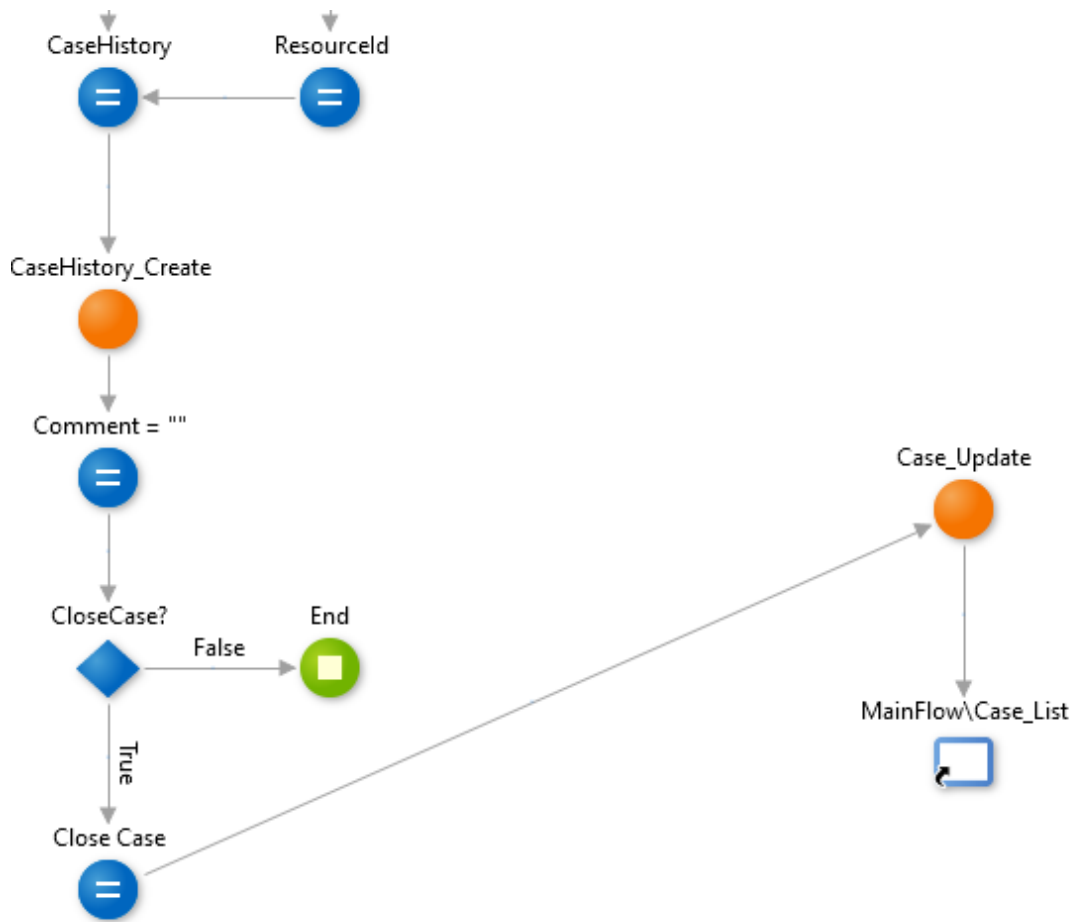


Figure 27 - Upload a File Logical Process

Upon pressing the button that prompts the upload file process, it follows the flow seen in fig. 27, first, the system verifies whether the user in fact has a file to upload, if false, it notifies the CaseHistory that nothing was changed. If true it checks the file that was selected and if everything is okay, it creates the resource by filling it in with the necessary information, namely, it's type, name and, if in the case of media files, it's binary type.

#### 4.3.4.3. AddComment Process

The AddComment process begins with the Upload a file process, as can be seen in the previous sections in fig. 27. After completing that flow, it moves on to comment verification, fig. 28 depicts this process.



**Figure 28 - AddComment Process**

The system fills the information for the “Comment” attribute with whatever information was inserted in the text area, then it proceeds to identify what button was pressed. If the “*Fechar Pedido*” button selected, it proceeds to close the case, if not, it simply updates the case.

(Blank Page for Formatting)

## 5. System Prototype

In this chapter, we will take a look at the application from the perspective of an admin user, we will analyse all the components from a UX and UI focused point of view and take a deeper dive into flow of the app and how it can be used efficiently.

### 5.1. General description

As briefly discussed earlier, the purpose of this application is to facilitate and improve existing systems.

### 5.2. User Interface

#### 5.2.1. Responsive Interface

OutSystems uses a template called “*Silk UI*”. It provides a set of adaptive and responsive primitives that allow developers to easily create their applications with a vast array of multiple forms in mind, particularly, for mobile. The adaptive patterns allow developers to adapt layouts and content for phone, tablet and different orientations.

Depending on the size and resolution, *Silk UI* captures device information (phone or tablet) to adapt content to the screen. Layouts for phone and tablet have different ways of handling screen size when used on other devices, for instance:

- The Phone Theme will create guard rails on the side when used on tablets. This prevents the original content from stretching and breaking the UI.
- The Tablet Theme will have a UI similar to the Phone Theme when used in phones and by making use of the adaptive patterns, like columns, it's very easy to adjust content.

Every app is different, therefore, the way each app reacts to a specific device is unique. To address this *Silk UI* Mobile offers a set of adaptive patterns that give the developer full control of how the application looks of different screen sizes.

These adaptive patterns contain a set of *adaptive actions*. These actions give the developer further control by allowing him/her to lock screens and hide them from users on specific devices. This allows a dev to, for example, create a home screen for a tablet, another for PC and a third for mobile, then, uses the adaptive action that detects what screen size the user has and then showing the correct home screen.

In this particular application, the use of this responsive design is clearly seen.

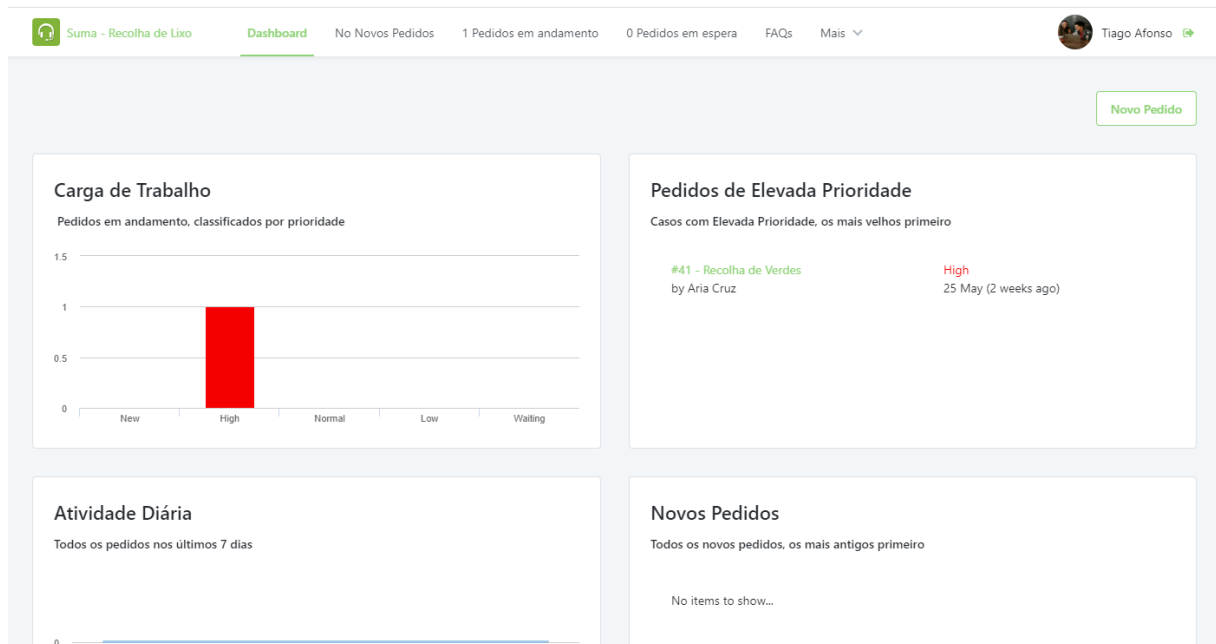


Figure 29 - Responsive View - Browser



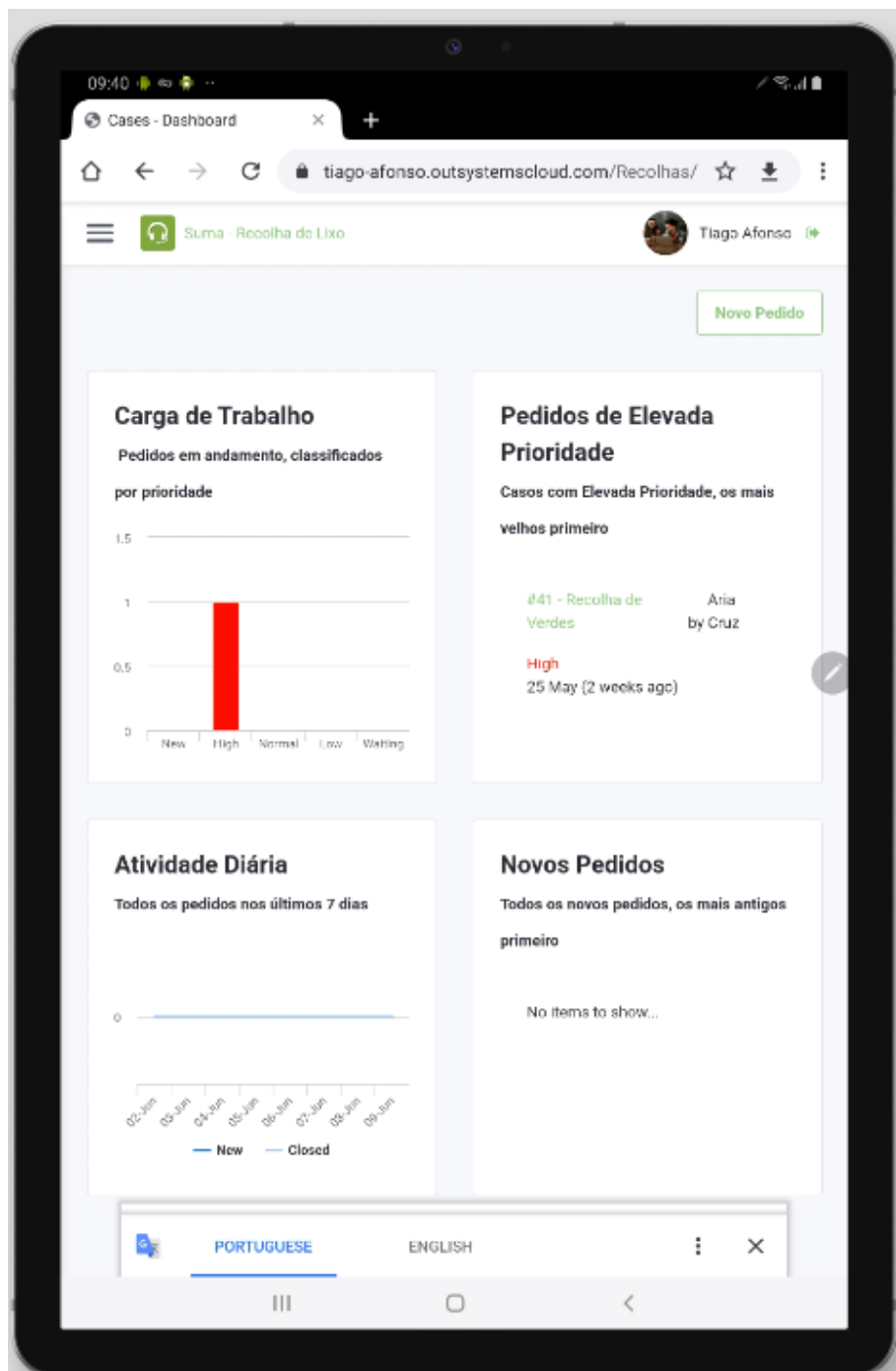


Figure 30 - Responsive View – Tablet

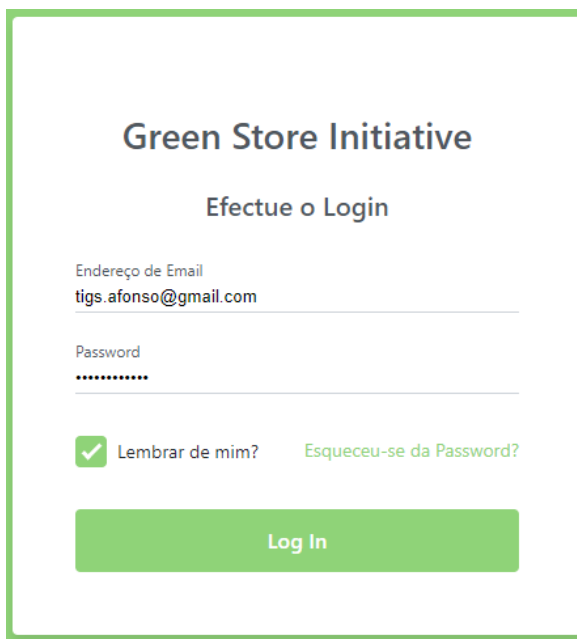


Figure 31 - Responsive View - Mobile

## 5.2.2.Screen Flow

In this subheading we will be taking a look at all the screens and explaining the main actions available on each one with a focus on UX (User experience).

### 5.2.2.1.Login Screen



Green Store Initiative

Efectue o Login

Endereço de Email  
tigs.afonso@gmail.com

Password  
\*\*\*\*\*

Lembrar de mim? [Esqueceu-se da Password?](#)

Log In

Figure 32 - Screen Flow – Login

The first thing we see when loading the website is the Login screen. As mentioned earlier, there is no way for a user to register, as this is an exclusive service, one only gets his/her credentials after receiving them from a staff member.

Any given user can login using his email and password, the process behind this has already been explained, at once he is logged in, the user is redirected to the main page.

### 5.2.2.2. Dashboard Screen

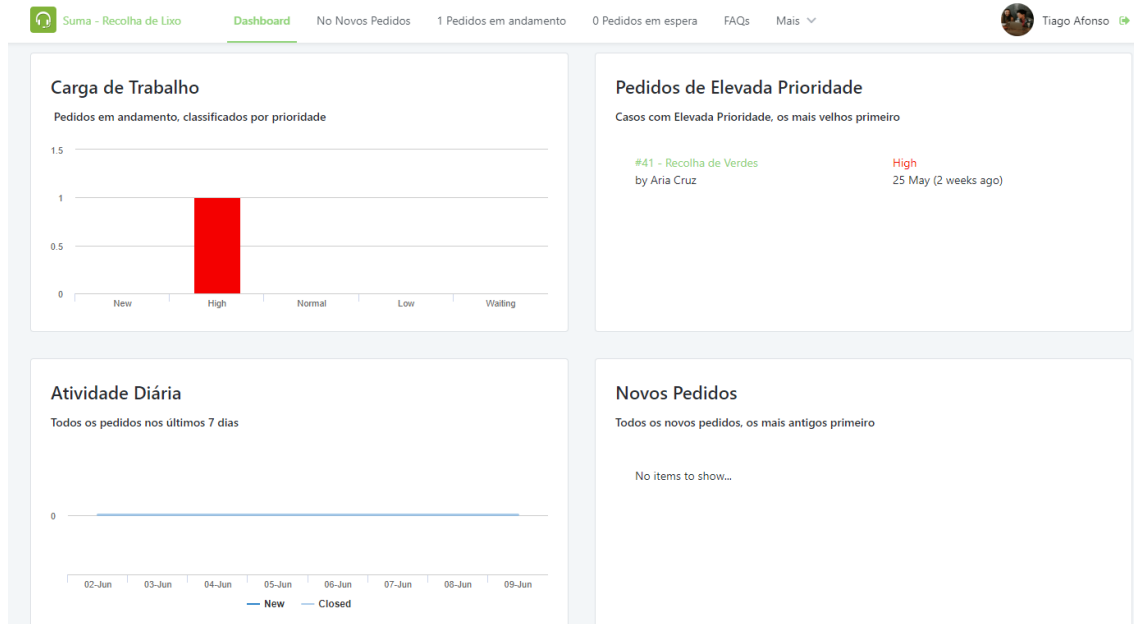


Figure 33 - Screen Flow – Dashboard

This is the main page where all users are redirected to, depending on each user’s role, he has access to information useful to him, for instance, if the user is a pickup staff, the following screen is what he will see:

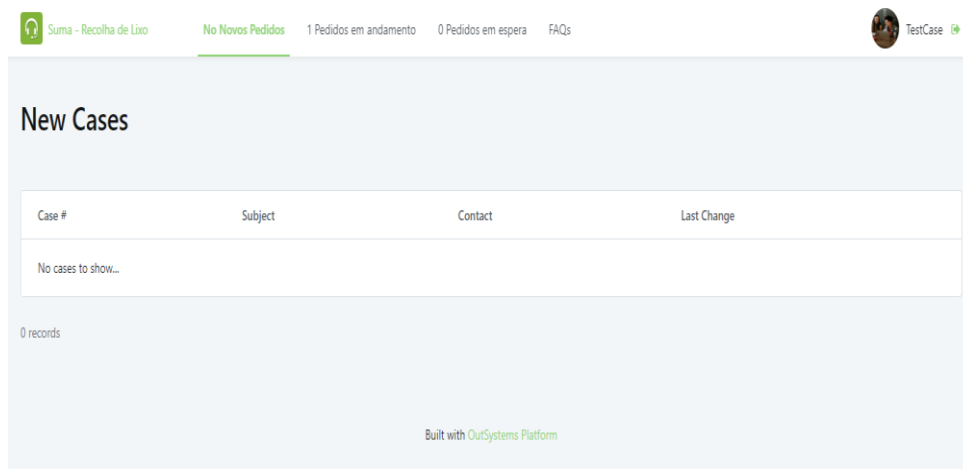


Figure 34 - Screen Flow - Pickup Staff Dashboard

As we can see, the staff member shown above does not have a role that gives him access to most of the information, only allowing him to see the widgets that are relevant to him.

### 5.2.2.3. New Cases Screen

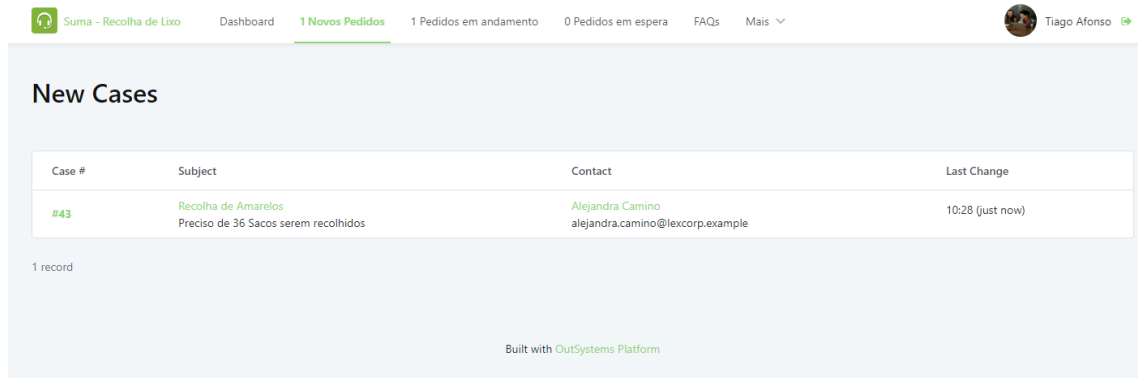


Figure 35 - Screen Flow - New Cases

As we can see in the image above, the next screen a user can access is the “*Novos Pedidos*” or New Cases screen. Here, the staff member is able to view, access and edit all of the cases that are marked as “New” in their status Id dropdown.

### 5.2.2.4. Working Cases Screen

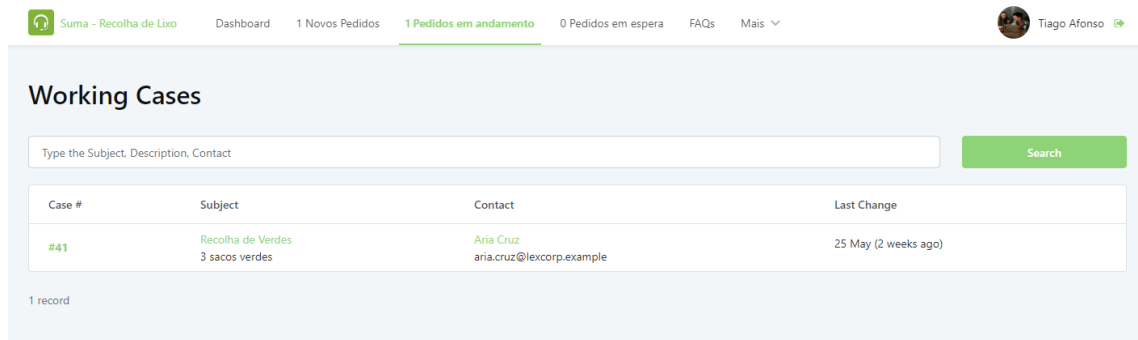


Figure 36 - Screen Flow - Working Cases

Like the “New Cases” screen, the Working Cases one shows a comprehensive list of the cases marked as “Working” as their status Id. On this page, the user can do the same as in the previous one, as in, he can access and edit the case. But there is noticeably a new addition on this one, the search bar. This feature allows a user to search throughout the whole list for a specific subject, description or contact, allowing a staff member to find the specific information they are looking for, amongst potentially hundreds of working cases.

### 5.2.2.5. Waiting Cases Screen

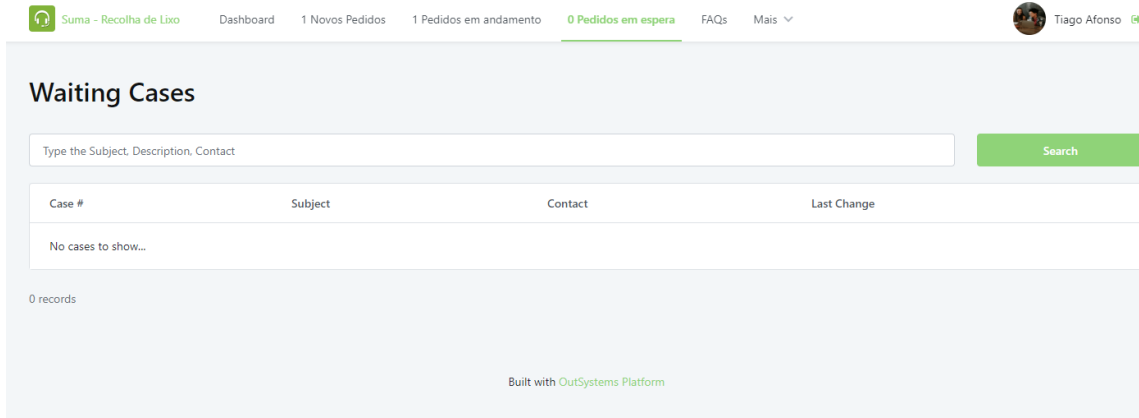


Figure 37 - Screen Flow - Waiting Cases

Similar to the previous two screens, this one shows the list of cases marked as “waiting” as their status Id. Like the “working cases” screen, this one also features a search bar with the same functionality as the aforementioned screen.

### 5.2.2.6. Case Screen

The case screen is where a user has access to all the information regarding a specific case and where he also can perform useful actions that will be discussed next.

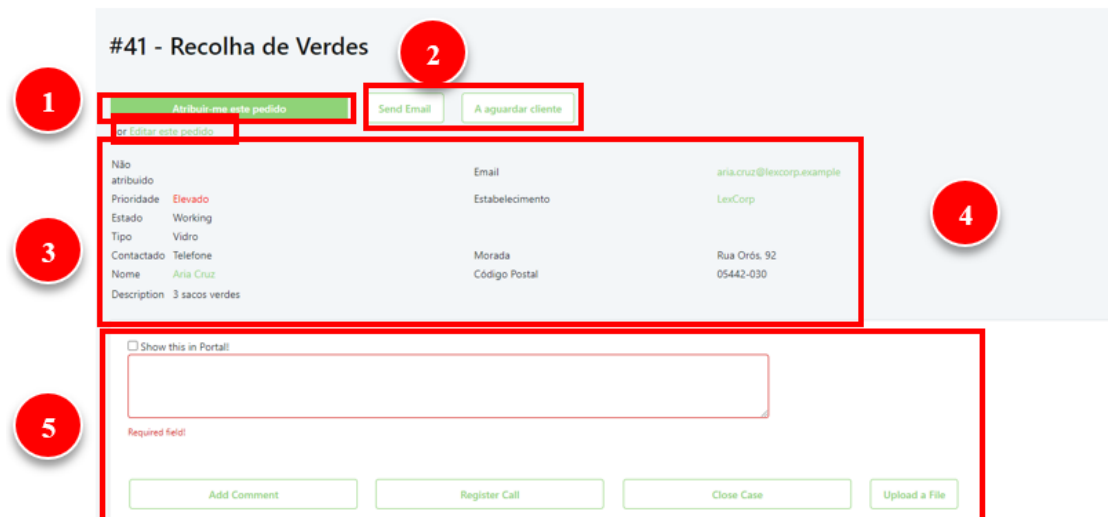


Figure 38 - Screen Flow - Case screen

- 1) Here we find a button “*Atribuir-me este pedido*” this button will seldom be used, yet it is important it remains there. Simply put, this allows a pickup staff member to assign that specific case to himself, although typically, cases would be assigned to each pickup staff member by an admin, in some circumstances, having this feature can prove useful.
- 2) Next we see two different buttons, the first one send email opens the device’s email app and automatically inserts the contact’s email into the “to” field, allowing a user to quickly email the client. The next button, the “*A aguardar cliente*” sets the status of the case as waiting.
- 3) The third section has a tiny button that reads “*Editar este pedido*”, this opens a new screen that allows a user to change certain information regarding the case.
- 4) In this section we find all the relevant information we possibly need. We can see who it is assigned to, what is its priority, status and type. We also find information regarding the contact, such as their name, number, email, establishment, address and postal code, amongst other relevant information.
- 5) This section allows a user to insert comments for that specific case, upload images or other files and mark it as complete. Thus, closing the case.

History	Updated By	Show in Portal
Priority changed to Elevado Status changed to Working	Tiago Afonso 25 May (2 weeks ago)	<input type="checkbox"/>
Recolha de Verdes 3 sacos verdes Vidro submitted by Telefone	Tiago Afonso 25 May (2 weeks ago)	<input checked="" type="checkbox"/>

**Figure 39 - Screen Flow – History**

This screen also has a section called History, the system tracks and records all the changes done to the case and inserts them here for easier control and viewing.

### 5.2.2.7. Case Edit Screen

In the last section, under point number 3 of the description regarding the case screen it was noted the existence of a button, the “*Editar este pedido*”, button. As mentioned, upon it’s pressing, the user is redirected to another page, it can be seen below.

**#41 - Recolha de Verdes**

1 Name Aria Cruz  
Email aria.cruz@lexcorp.example  
Company LexCorp

2 Designado a Tiago Afonso  
Tópico Recolha de Verdes  
Descrição 3 sacos verdes

3 Prioridade Elevado  
Status Working  
Tipo Vidro  
Source Telefone

Guardar Apagar Cancelar

**Figure 40 - Screen Flow - Edit Case**

This screen is comprised of three distinct sections:

- 1) An area with a few details regarding the contact and his or her business.
- 2) A section where an admin can assign the case to a pickup staff, edit its topic and description.
- 3) Finally, a section that provided the admin with the ability to assign the case a priority and status, in addition to editing its source and type fields if need be.



## **6. Conclusion**

Based upon the initial problem definition of creating an application that would improve an existing service related to waste management, following the DSR methodology, five research goals were established:

- **RG 1** – Perform a literature review in order to further investigate the availability of technologies that may be available to solve the issue, as well as deepen the understanding of topics that may be related to the project (such as cross-platform, responsive web design, OutSystems, etc.);
- **RG 2** - Determine the functional and non-functional specifications regarding the Green store initiative application;
- **RG 3** – Draft a Use Case model for the GSI (Green store Initiative) application;
- **RG 4** – Create a domain model, identify and apply the necessary architectural processes and best practices for the app;
- **RG 5** – Develop and test a working prototype.

Regarding each of these research goals:

- **RG 1** – A literature review focused on highlighting some of the main relevant aspects of concepts regarding this project was written in chapter 2;
- **RG 2** – A list of functional and non-functional features was made in chapter 3;
- **RG 3** – A diagram showcase the main uses of the application was created and presented in chapter 4;
- **RG 4** – A domain model was created and list of architectural processes regarding the GSI app were created and showcased in chapter 4;
- **RG 5** – A working prototype was successfully implemented and tested as demonstrated in chapter 5.

Based upon these reasons, it can be verified that the identified issue has been solved and the research goals successfully accomplished.

As future work, with the goal of improving and expanding upon the project, the need for a mobile application to be available for customer use was identified. This application would replace the current web application in the sense that although staff members would use the current prototype for management and assigning tasks, the customers would have access to a mobile app that allows them to place a pickup request in a simplistic way as seen in figure 41. This can be done by creating a project in OutSystems then connecting the app with the existing DB allowing therefore, one to update the other.

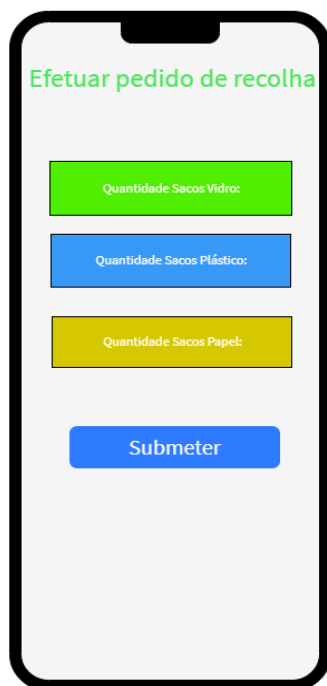


Figure 41 - Future App Concept

## Bibliography

- Gartner. (2019, August 8). *Magic Quadrant for Enterprise Low-Code Application Platforms*. Retrieved from gartner.com:  
<https://www.gartner.com/en/research/methodologies/magic-quadrants-research>
- Graeve, K. D. (2011, November). *Responsive Web Design*. Retrieved from MSDN Magazine: <https://docs.microsoft.com/en-us/archive/msdn-magazine/2011/november/html5-responsive-web-design>
- Hevner, A. (2007). A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems 19*.
- Hevner, A., & Chatterjee, S. (2010). *Design Science Research in Information Systems*. Springer.
- Jones, T. (2019, July). 7 Key Characteristings of Cloud Computing.
- K. (2015, August 10). *Responsive Web Design*. Retrieved from MSDN Magazine: <https://docs.microsoft.com/en-us/archive/msdn-magazine/2011/november/html5-responsive-web-design>
- Majidimehr, R. (2016). Media Queries and Listeners.
- Marcotte, E. (201, May 25). *Responsive Web Design*. Retrieved from A List Apart : <https://alistapart.com/article/responsive-web-design/>
- Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing.
- Nitceki, S. (2019). Cross-Platform Mobile Apps Development.
- OutSystems. (2019, June 6). *The 4 Layer Canvas*. Retrieved from OutSystems: [https://success.outsystems.com/Support/Enterprise\\_Customers/Maintenance\\_and\\_Operations/Designing\\_the\\_architecture\\_of\\_your\\_OutSystems\\_applications/01\\_The\\_4\\_Layer\\_Canvas](https://success.outsystems.com/Support/Enterprise_Customers/Maintenance_and_Operations/Designing_the_architecture_of_your_OutSystems_applications/01_The_4_Layer_Canvas)
- OutSystems. (2020, April 8). *OutSystem Platform Best Practices*. Retrieved from OutSystems\_BP: [https://success.outsystems.com/Documentation/Best\\_Practices/Development/OutSystems\\_Platform\\_Best\\_Practices](https://success.outsystems.com/Documentation/Best_Practices/Development/OutSystems_Platform_Best_Practices)
- OutSystems. (2020, June 16). *Tools and Components*. Retrieved from Evaluation Guide: <https://www.outsystems.com/evaluation-guide/outsystems-tools-and-components/>

- OutSystems. (n.d.). *Can OutSystems Be Used To Build Complex/Big Applications?* Retrieved from Evaluation Guide: <https://www.outsystems.com/evaluation-guide/can-outsystems-be-used-to-build-complex-big-applications/>
- OutSystems. (n.d.). *Why OutSystems?* Retrieved from OutSystems Evaluation Guide: <https://www.outsystems.com/evaluation-guide/why-outsystems/>
- Ranger, S. (2018). What is Cloud Computing? Everything You Need to Know About the Cloud Explained.
- Rymer, J. R. (2017). Low-Code Development. The 13 Providers That Matter Most And How They Stack Up. *The Forrester Wave*.
- Wafaa , S. E.-K., Bassem, A. A., Ahmed, H. Y., & Ayman, M. W. (2017). Taxonomy of Cross-Platform Mobile Applications Development Approaches.
- Watts, S., & Raza, M. (2019, June 15). *SaaS vs PaaS vs IaaS: What's The Difference and How to Choose*. Retrieved from BMC: <https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/>
- What is Cross-Platform Development?* (n.d.). Retrieved from technopedia.com: <https://www.techopedia.com/definition/30026/cross-platform-development>

## Appendix

### A. OutSystems Best Practices

#### 1) Standard Language

- Use English for code and comments.

#### Naming Conventions

- Meaningful names;
- Use PascalCase;
- Suffix foreign keys with “Id”;
- Include the Entity’s Name in the Record’s name;
- Group Screens with a prefix;
- Prefix actions invoked by Timers with “Timer\_”;
- Set the Name property of ShowRecords, EditRecords and Tableecords.

#### Coding Conventions

- Avoid empty labels and descriptions;
- Comment unclear or complex logic;
- Set the example string of Expressions;
- Keep Action Flows vertical and tidy;
- Use Static Entities instead of hard-coded values.

#### Reusability

- Reuse logic with User Actions;
- Reuse screen parts with Web Blocks;
- Encapsulate data formatting with User Functions;
- Use RefreshQuery to rerun a Query.

#### 2) JavaScript, CSS and HTML

- Use Web Blocks for JavaScript encapsulation and reusability;
- Make sure your JavaScript blocks are perfectly identifiable;
- Comment your JavaScript;

- Use cross browser JavaScript;
- Avoid writing custom HTML using unescaped expressions;
- Avoid duplicate CSS styles;
- Minify JavaScript and CSS whenever possible.

## Database

- Avoid overcrowding entities with attributes;
- Avoid overcrowding attributes;
- Check the Delete Rule of Foreign Keys to an Entity;
- Remember to set the Is Mandatory property;
- Add descriptions at least to the entities.

## Aggregates and Advanced Queries

### Queries

- Use Aggregates;
- Avoid using indexes when iterating a Query output;
- Minimize the number of Queries you need to execute;
- Avoid type casts in Aggregates.

### Advances Queries

- Use Advanced Queries only when absolutely necessary;
- Use Parameters instead of hard-coded values in the SQL;
- Indent SQL code consistently;
- Use comments and inline comments in your SQL;
- Use Advanced Queries for bulk operations;
- Do not put business logic in the SQL;
- Use the EncodeSQL built-in function for inline Parameters;
- Keep in mind that Advanced Queries are database specific.

## Build for Change

- Favour Aggregates over Advanced Queries;
- Keep in mind that Advanced Queries are harder to maintain;
- Avoid creating custom HTML/JavaScript;
- Split functionality in different Web Screens;
- Don't use Extensions to implement business logic.

## User Interface

- Move JavaScript blocks to the end of the screen;
- Find out what needs improvement by observing your users;
- A Preparation with more than 10 nodes normally means trouble;

- Reduce the size of the eSpace's images.

## Architecture

Use a 3 layers approach:

- Business Processes: a service available for use by users or business processes;
- Core Entities: logical grouping of operations per responsibility;
- Connectors: extensions or integration with other systems.

Also:

- Wrap Extensions in eSpaces;
- Avoid circular references between eSpaces;
- Clearly define the responsibilities of each eSpace;
- Remove unused references Encapsulate business logic in User Actions;
- Create user story driven interfaces Avoid isolating the data model in one eSpace. eSpaces should be functional units instead of architectural units;
- Use asynchronous logic when possible.

## Teamwork

- Know the recommended project team for your project;
- Use a single user for each team member;
- Everyone in the team should use the same Service Studio version;
- Use LifeTime to perform application versioning;
- Avoid publishing single eSpaces: use LifeTime;
- Avoid working on the same elements at the same time;
- Create team processes for active log monitoring;
- Create team processes for manual deployment procedures such as data update scripts;
- Define what "done" means and include testing.

## Logic and Development

- Remove unused or duplicate code;
- An eSpace with a size > 4 MiB normally means bad architecture;
- Set at least one default button in your Web Screens;
- Avoid changing Site Properties values in your logic;
- Be careful about infinite recursion;
- Hide warnings only for a good reason;
- More than 20 site properties in one eSpace normally means bad design;

- Avoid using Extensions solely for Web Service consumption;
- Disable auditing of Extensions in Production.

## **Security**

- Set the Web Screen's Roles;
- Be aware of sensitive data exchange;
- DON'T send sensitive information in screen parameters;
- Remember: Web Screen's variables or Preparation outputs might be exposed in the URL or Viewstate;
- When applicable, use SSL for sensitive information;
- Do not rely on the Web Screen widgets interface to control permissions;
- Validate user's Roles before executing Screen Actions;
- Use encrypted passwords in the database;
- Use Internal Access Only for Web Flows and Web Services.

### **3) Performance**

#### **Data Model**

- Index your Entities;
- Isolate large Text and Binary Data in another entity;
- Beware of Large Excel Files Performance;
- Make good use of the Delete Rules in Entities;
- Archive old data in separate Entities.

#### **Infrastructure**

- Setup database maintenance plans;
- Configure Web server memory settings;
- Backup database transaction logs often;
- Tune database file growth;
- Apply virtual memory and system settings;
- Use a dedicated server for Timers;
- Prefer 64-bit architectures;
- Plan and monitor timers schedule so there are no conflicting schedules.



## Logic

- Look into Service Center reports;
- Avoid long-running Timers and batch jobs;
- Simplify screen Preparations;
- Place as little information as possible in Session Variables;
- Avoid using isolated Aggregates in User Actions;
- Avoid using queries inside If branches in Preparation;
- Avoid chained Web Service calls. Return as much as possible in a single call.

## Queries

- Don't perform joins over linked servers;
- Minimize the number of fields fetched from the database;
- Keep Max Records consistent with your needs;
- Iterate over queries once and avoid using indexes ([i]) ;
- Minimize the number of executed queries.

## References

- Never create circular references between eSpaces;
- Choose wisely between Local Web Services or Public Actions;
- Use Connection Pooling.

## User Interface

- Avoid using Preparation data in screen actions;
- Use AJAX with care - it may impact overall performance;
- Use multiple Web Screens instead of creating complex ones;
- Use caches frequently;
- Minimize Screen Parameters;
- Import JavaScript files as resources;
- Move JavaScript Web Blocks that are not needed at loading time to the bottom;
- Use CSS Sprite Images.