# DeTouch
## Extending service channels in an airport through touch technologies

MASTER DISSERTATION

**Bruno Pires Lavigne Quintanilha**
MASTER IN INFORMATICS ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

November | 2015

# DeTouch
## Extending service channels in an airport through touch technologies

MASTER DISSERTATION

**Bruno Pires Lavigne Quintanilha**
MASTER IN INFORMATICS ENGINEERING

SUPERVISOR
Leonel Domingos Telo Nóbrega


# DeTouch – Extending service channels in an airport through touch technologies

**Bruno Pires Lavigne Quintanilha**

Constituição do júri:

José Luís Cardoso da Silva

Karolina Baras

Leonel Domingos Telo Nóbrega

Fevereiro 2016

Funchal – Portugal

# ACKNOWLEDGMENTS

I want to thank everyone involved in this project influencing its result by encouraging, providing feedback or helping on the development directly.

Professor Leonel Nóbrega who guided me through the whole course of development and enriched the project contributing with all his knowledge from the process to the design and development demonstrating high availability.

Eng. Duarte Ferreira, the Madeira Airport general manager, for allowing this project to be developed at the Madeira airport and for the feedback provided.

Eng. Elsa Franco, the Technical department manager, such a busy person but always found some time to provide her valuable feedback and suggestions.

Eng. Rafael Silva, the IT department manager at the Madeira Airport, who always believed in this project and provided valuable feedback during the development process.

Hugo Leão who accepted to join us on this project in such a short notice and gave all he had to produce a high quality result.

Inside the Madeira Airport a couple of departments helped on bringing this project to reality. First, all the IT department staff members who helped with the configuration and some strategies to achieve our goals (thank you Luciano Franco, Vitor Gouveia and Eduardo Teles). Second, the electric department for supporting the installation of the equipment where it was supposed to (Antonio). Third, Crispim from the civil department for installing the barcode scanner at the kiosk. Last, everyone who got involved in the evaluations and spared their time to provide feedback about the system that was under development (André Freitas, Drª Cláudia Gonçalves and Drª Silvia Dias).

My mother and father in law for supporting me and my family in everything we need to succeed, always taking our side.

To my mother and my sister who supported me in the good and bad times.

To my children, Camila and André, the joy of my life.

Last, to my wife who supported me with all her affection and love otherwise this dissertation would not be accomplished.

Thank you all.

# ABSTRACT

This project aimed to create a communication and interaction channel between Madeira Airport and its passengers. We used the pre-existent touch enabled screens at the terminal since their potential was not being utilised to their full capacity. To achieve our goal, we have followed an agile strategy to create a testable prototype and take advantages of its results. The developed prototype is based on a plugin architecture turning it into a maintainable and highly customisable system. The collected usage data suggests that we have achieved the initially defined goals. There is no doubt that this new interaction channel is an improvement regarding the provided services and, supported by the usage data, there is an opportunity to explore additional developments to the channel.

# KEYWORDS

Touch technology

Windows presentation foundation

Airport infrastructure

Plugin system architecture

Prototype

User experience

# ACRONYMS

WPF – Windows Presentation Foundation

MVVM – Model View ViewModel

C# - C Sharp

PHP – Hypertext Preprocessor

RAD – Rapid Application Development

DB – Database

UI – User interface

XAML – Extensible Application Markup Language

XML – Extensible Markup Language

LAMP – Linux Apache MySQL PHP

IDE – Integrated Development Environment

ATAM - Architecture Tradeoff Analysis Method

QAW – Quality Attribute Workshop

API – Application Programming Interface

MVC – Model View Controller

DLL – Dynamic-link library

ADO – ActiveX Data Object

E-R – Entity Relationship

CRUD – Create Retrieve Update Delete

IATA –International Air Transport Association

ICAO – International Civil Aviation Organization

SOAP – Simple Object Access Protocol

JSON – JavaScript Object Notation

HTTP – HyperText Transfer Protocol

# RESUMO

Este projeto teve como objetivo criar um canal de comunicação e interação entre o Aeroport da Madeira e os seus passageiros. Utilizamos os monitores de toque existentes no terminal do aeroporto uma vez que o seu potencial não estava a ser aproveitado da melhor forma. Para atingir o nosso objetivo, seguimos uma estratégia ágil para criar um protótipo facilmente testável e aproveitar os seus resultados. O protótipo desenvolvido segue uma arquitetura baseada em plugins facilitando assim a manutenção e personalização do sistema. Os dados de utilização recolhidos até à data sugerem que conseguimos alcançar os nossos objetivos definidos inicialmente. Este novo canal de interação é, sem dúvida, um grande avanço e melhoramento em relação ao sistema previamente instalado do ponto de vista dos serviços proporcionados e, suportado pelos dados de utilização, existe uma oportunidade de explorar desenvolvimentos adicionais para o canal criado.

# PALAVRAS CHAVE

Tecnologia de toque

Windows presentation foundation

Infraestrutura aeroportuária

Arquitetura de plugins

Protótipo

Experiência do utilizador

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF CHARTS

# 1 INTRODUCTION

Madeira Airport is definitely the most important port of entry for people to Madeira, whether they be residents or tourists visiting the island. As an infrastructure, the airport offers a wide range of services to a very diverse group of stakeholders that includes, but is not limited to, the flight companies and their passengers. In fact, the concept of an airport has changed in a number of important respects in recent years. The shopping space and other kinds of business-oriented services are shaping the layouts of airports and affecting the routes that passengers have to follow either to reach the boarding gates or to leave the airport. Although the time that passengers spend in the shopping areas has become important, at the same time everything should be organized in such a way that it does not affect the core business, which is aviation.

The companies that manage this infrastructure have to address a very demanding set of requirements, under very strict security conditions. They are allowing for more freedom of movement, but at the same time they have to ensure that passengers are at their boarding gate on time. Consequently, communication with passengers is a crucial aspect of the managing of such infrastructures, demanding the creation of new channels of interaction in order to improve the quality of the services provided. Smartphone applications are an effective way to interact with regular passengers, but this does not apply in the case of more occasional visitors to Madeira, such as tourists. Alternatively, devices like public touchscreens could be a good way to address this type of user.

## 1.1 Motivation

The airport has to provide solutions to the needs that have arisen. As the most important stakeholder of the business, the passenger deserves special attention at the airport.

In order to fill this gap, the airport must provide new ways of interacting with its passengers so as to provide valuable services that the passenger feels are important. Using this new method of interaction we can assist the passengers by helping the airport operation with regard to passenger arrivals and/or departures by providing flight information and locations at the terminal.

The task of building a system to provide this new interaction was given to the author of this thesis by the airport IT manager. As a master's student, a request was made to Professor Leonel Nóbrega and to Eng. Duarte Ferreira (airport general manager) to integrate the project into the master dissertation. The request was accepted and the project was ready to begin.

## 1.2 Problem

Madeira Airport already had digital equipment, 50-inch touch-screen monitors, to provide services to its passengers. The problem was that the services offered from this platform were not adequate for the passenger's needs as only one service was provided and the potential was clearly much greater.

The software installed in this system resembled a typical website regarding its look and feel. It was built using JOOMLA which is CMS (Content Management System) software used for building websites. It did not offer any kind of system usage data in order to identify possible future improvements. The only service provided by the platform was a virtual directory to assist passengers to ascertain their position inside the airport.

To provide new and better services for the airport passengers we have developed a whole new system to achieve better usage of the existing digital equipment. We had to

rethink the entire set of services to understand what would be more useful to airport passengers.

## 1.3 Approach

In order to develop the system, we followed an agile process in order to figure out which would be the most useful services that we could provide, in an incremental and interactive manner, to the passengers.

We have developed a fully functional prototype to be deployed in a real context in order to collect useful usage data to better understand passenger needs.

## 1.4 Contributions

The system is a clear upgrade from the previous one. Now there is not only one service provided at the kiosk, but six valuable services for Airport passengers. New services can be seamlessly integrated without any breaking changes. Anyone, even from outside the company, can develop a new service and deploy it very easily.

The developed prototype will be used as a tool for benchmark tests to determine new services to provide to the passengers. At the same time, it has to be a very robust prototype to be deployed in a real context by a small development team with a tight deadline.

This is a valuable diagnosis and evaluation tool that the airport plans to use for providing for new services to the passengers.

## 1.5 Dissertation structure

As stated, this project aimed to create a communication channel between the Madeira Airport and its passengers. Using the existing touch equipment at the airport we developed a prototype as a tool to benchmark new ideas by following a development process. Lastly, we analysed usage data, by means of charts, in order to understand how

users are interacting with the developed services. The dissertation is structured as follows:

- Chapter 2 - State of the Art: Study of the technology used to develop the project. In addition, research on available software in the field of large touch screen applications.
- Chapter 3 - Process: Presents the development process followed. Integration between RAD and Agile methodologies as well as all the steps taken to follow the process.
- Chapter 4 - Development: In this chapter we present the architecture, quality attributes and implementation decisions of the whole project. Implementation details are given showing snippets of code and figures to better illustrate the overall system. We have split this chapter into three parts regarding the overall system, the management application and the kiosk software itself.
- Chapter 5 - Prototype: A prototype guide is presented from the end user perspective. All aspects of the interfaces developed are shown and explained in detail. We cover the developed template as well as all the developed plugins.
- Chapter 6 - Evaluation: Usage data is analysed to understand how users are interacting with our system and how frequently.
- Chapter 7 - Conclusions: We present the conclusions taken from the process, development and evaluation. Furthermore, the contributions of this project are mentioned as well as improvements which could be implemented. Also, directions for future developments are provided.

# 2 STATE OF THE ART

In this chapter we will provide information concerning two goals. Firstly, we address the historical evolution of the touch-device technology, since it is the main interaction technology provided on the large multi-touch screens used in the developed system. Secondly, we analyse some previous well-known touch-based interactive systems in order to get insights on how this technology could be used to support a better and more natural interaction.

With regard to this project, smartphones and tablets are considered to be out of scope, since we are mainly interested in large touch screens.

## 2.1 Touch technology history

The touch-enabled devices are certainly not a new technology as they date back to the 1960s. In fact, in 1965 Eric Arthur Johnson published an article where he described a capacitive touch screen. The screen consisted of a monitor with wires across it that were sensitive to the touch of a finger [1]. The same author published another article in 1967 where he dug deeper into the touch screen concept, explaining his prototype through diagrams and photographs [2]. In 1968 he published another article demonstrating a use case for his prototype applied to air traffic control [3].

In 1972 CERN engineers developed a touch screen with resistive technology that was transparent and launched in 1973. The main goal was to build a piece of equipment that

would allow the management of the SPS (Super Proton Synchrotron) with only three command units. This equipment was active until 2008 [4].

In the same year an optical touch screen called PLATO IV was developed. These monitors consisted of a matrix of 16x16 infrared sensors. It was widely used by students to answer questions by touching the answer on the monitor [5].

In the 1980s there were several touch screen monitors developed with new technologies that are still in use today. Today's technology is an improvement on the technology developed in the 1970s and 1980s. The improvements provided more accurate and precise touch screens enabling a better experience for the user. One major improvement was the possibility of identifying multiple simultaneous touches, known as multi-touch technology. Multi touch allows users to perform gestures that would feel more natural when interacting with the system, thus providing a better experience such as browsing a digital book by swiping the screen.

### 2.1.1 Resistive

This touch technology consists of a bottom glass layer and a conductive film as the top layer [6]. Both layers face each other with a small gap between them. When the user touches the monitor the two layers make contact at that point resulting in a voltage change, which makes it possible to determine the point of contact [7].

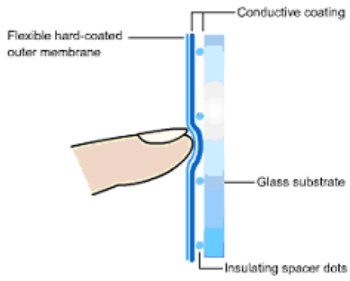



Figure 1 Resistive touch technology (available at http://www.elotouch.com/Technologies/compare_resist.asp on February 2016)

It is the preferred solution if the application requires durability and reliability. The touch screen still works even if the top layer is damaged.

### 2.1.2 Surface Capacitive

In this technology a transparent electrode layer is placed on top of a glass layer and covered by a protective layer. When a finger touches the monitor it reacts to the human body's static electrical capacity and some of the charge is transmitted to the user. This loss of capacitance is detected by the sensors located at the four corners of the screen making it possible to determine the point of contact [7].

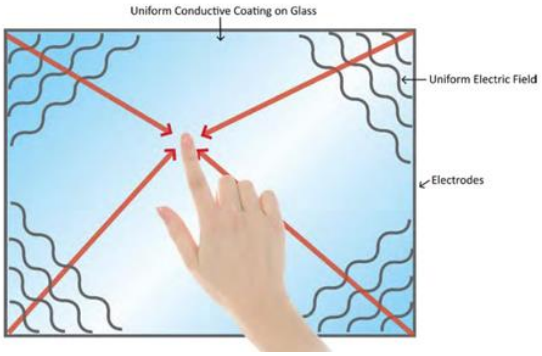



Figure 2 Surface Capacitive (available at https://www.touchsystems.com/surfacecapacitive on February 2016)

### 2.1.3 Projected capacitive

This technology consists of a matrix (or field of rows and columns) controlled by a microcontroller. It works in a similar way to the previous technology but it creates an electromagnetic field that gets distorted when a finger touches the monitor, enabling the detection of the position where the finger touched.

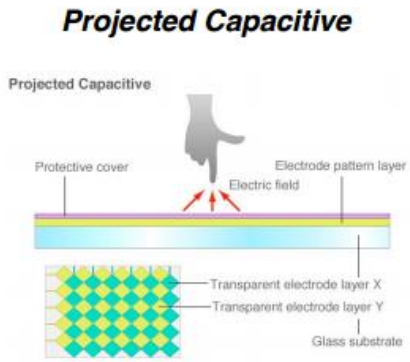


Figure 3 Projected Capacitive (available at http://www.eizoglobal.com/library/basics/basic_understanding_of_touch_panel/ on February 2016)

### 2.1.4 Surface acoustic wave

Wave reflectors are replaced around the monitor creating an invisible grid of ultrasonic waves using transducers placed in the monitor's corners. When the monitor is touched, a portion of the wave is absorbed allowing the computer to locate the touch position [7]. This is the technology used by the existing monitors at Madeira Airport [8], [9].

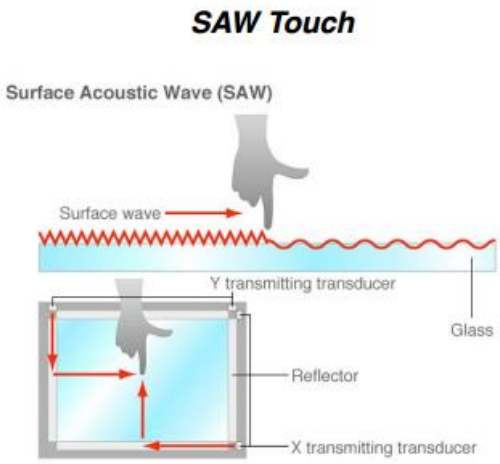



Figure 4 Surface Acoustic Wave (available at http://www.eizoglobal.com/library/basics/basic_understanding_of_touch_panel/ on February 2016)

### 2.1.5 Infrared

Using infrared emitters and receivers, this technology creates a grid of invisible light beams across the whole monitor. When an object interrupts the infrared light beams the sensors are able to identify the location where the interruption has occurred [7].

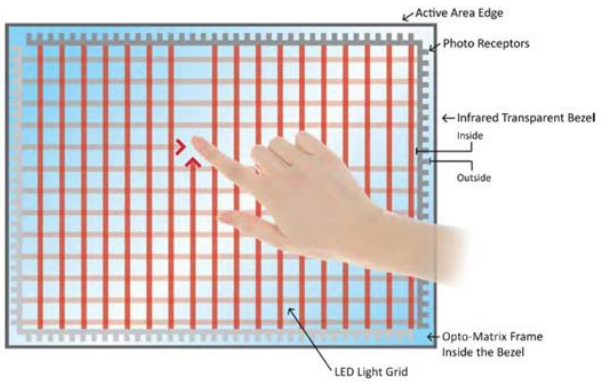



Figure 5 Infrared technology (available at https://www.touchsystems.com/infrared on February 2016)

### 2.1.6 Comparison

The table below shows advantages and disadvantages of the five mentioned touch technologies.

| | **Advantages** | **Disadvantages** |
|---|---|---|
| **Resistive** | Might be activated by any object; Low cost; Low consumption; Resistant to surface contamination; | Not the best image quality; Vulnerable to scratches; |
| **Surface Capacitive** | Good image quality; Longevity; Resistant to contamination and scratches; | Requires bare finger or capacitive equipment; Sensitive to electromagnetic field contamination; |
| **Projected Capacitive** | Excellent image quality; Resistant to scratches; Resistant to surface contamination; Multi touch (two points); | Activated only by bare finger or thin gloves (or cotton); |
| **SAW** | Excellent image quality; Scratch | Not activated by solid |

| | | |
|---|---|---|
| | resistant; Longevity; | objects as they create untouchable areas; Liquids can contaminate the surface; |
| **IR** | The best image quality; Longevity; Does not scratch; | False touches can be detected because the light beams are over the monitor. Vulnerable to contamination of the surface; High cost; Sensitive to light interference; |

Table 1 Touch technologies comparison [7]

### 2.1.7 The future of touch technology

The touch technology has been around for many years but the future is still to come. The possibilities are infinite and it is very hard to predict what is coming and what will make users happy and comfortable while using it.

Developments in this area are being made to bring futuristic features to today's touch technology. Haptic feedback will help users to interact with the touch screen, providing physical feedback on the screen surface by representing interface elements such as buttons and other controls.

Mobile gestural commands are under development. This technology is a mobile version of the Wii game console and the Xbox Kinect. A user will be able to perform gestural commands in a mobile fashion and with no need to actually touch the device.

Flexible touch interfaces are being developed, bringing a new world of tactile feeling when it comes to user interaction. Not only flexibility but also malleability is being developed, bringing the virtual 3D world closer to the users.

## 2.2 Multimedia kiosks

There are several distinct solutions for multimedia kiosks available in the market. These solutions include virtual directories, virtual receptionists, self-service in the food and beverage area, and aviation, amongst others. Firstly we will cover the history of this type of equipment. Lastly, we will briefly cover some of the available solutions such as:

- Alice Virtual Receptionist
- ShowPath Virtual Directory
- InfoGate

### 2.2.1 History

In several public spaces, from medium to large, we find assistive devices for visitors. These devices are intended to guide users during their visit to take full advantage of the space and find what they need quickly and effectively.

This project falls into the category of computer kiosks. These kiosks are now most commonly installed in shopping centres. Even so, according to [10] not many shopping centres use these kiosks to provide information and directions to their visitors nor do they see this technology being implemented in the coming years.

In spite of the data and study findings in shopping centres, the use of kiosks by the holders of these spaces is growing and being applied, not only in shopping centres but also in airports, catering etc.

### 2.2.2 Alice Virtual Receptionist

*Alice virtual receptionist* is a software/service built for business making it possible to be used by employees and visitors of the organisation [11]. This kiosk provides several functionalities such as:

1. Management, information centre, maps and check-in for its visitors.
2. Employees can monitor the kiosk area by means of its built-in camera.
3. Capture pictures of the visitors.

4. Generate reports concerning visitor's information and system usage.
5. Visitors and employees can talk and see each other through the kiosk.
6. Provide automatic welcome messages to its visitors when they arrive.
7. Widgets are provided to allow fast modification of its original look and feel.

This a very comprehensive service providing several out-of-the-box solutions and functionalities for business.

This software is a mature product aimed at businesses and corporations. From its success we can deduce that getting in touch with our visitors is a necessity in kiosk software. The easy customisation of the software management is another important point of this software as the organisation can quickly change the front-end design and the services provided by the system.

### 2.2.3 Show Path Virtual Directory

*ShowPath Virtual Directory* [12] is a wayfinding system that provides a map of the infrastructure in which it is installed. In addition, it provides information about all infrastructure locations indicated on the map. It also provides hardware with an attractive design and multi-touch support. Operating system locker software is also provided to prevent users from *jumping* out of the kiosk software.


Figure 6 Show Path Virtual Directory Software [12]

This kiosk software validates the development of a service that would help passengers to ascertain their location inside the airport terminal. By grouping the locations into categories the service is easier to use and understand from the point of view of an end user.

### 2.2.4 Airport context

The last two solutions described have a generalised scope, i.e. they are not specific to any type of use. They can be found in shopping centres, shops, businesses and other similar locations. But this project is limited to the airport context and therefore these solutions could work well and provide good service to passengers but without providing any specific information on airport particularities.

The *InfoGate* [13] software was developed to improve the quality of the service provided at Munich airport. Therefore, it is similar to the system proposed for this project, e.g. improve the services provided at the airport terminal. The InfoGate system can be installed in four different flavours: virtual service kiosk, mobile, interactive and digital signage.

In the virtual service kiosk and the interactive mode there are services that are a match to the services we implemented in this project for Madeira Airport. Services like wayfinding and flight information from the user's boarding pass are similar to services within our project.

This equipment provides customer information and advertising to the passengers in the terminal by reading information from the boarding pass and the location of the kiosk in the terminal. In addition to this, all information is stored in order to generate reports about system usage [13].

The InfoGate development team found that free internet access to its passengers would be essential to obtain higher usage from mobile devices. In Chapter 3 we discuss this finding as we have reached the same conclusion from our brainstorm meeting. With this finding, we decided that it would be essential to develop a plugin to provide our

passengers with all the necessary steps to connect to the free wi-fi network in the terminal building. By providing this service, we expect to encourage the passengers to interact with the system and discover the other services provided at the kiosk.

So far, the InfoGate software is state-of-the-art when it comes to kiosks in the airport terminal context. Before and during the development of our project we had no knowledge of the InfoGate software as was discovered by the team during the research undertaken to write this dissertation. In the end and as a coincidence our software has a couple of features and findings that match the InfoGate software, thereby validating our work.

## 2.3 Conclusions

Touch technology is here to stay. It is everywhere and in almost every recent device. Desktop computers, laptops, multimedia kiosks, hand-held and many other devices are currently using this technology.

By using this new way of interaction we can provide better and ‘easier to use’ services to end-users.

The usage of kiosk software is increasing and the owners of large surface areas are trying to get in touch with their visitors to provide useful services to them in order to make their visit as pleasant as possible. Combined with the touch technology, the kiosks are easy to use by visitors and provide greater satisfaction when compared to other input mechanisms.

# 3 PROCESS

In this chapter we outline the development process adopted in this project, focusing on artefacts produced in the early stages. In addition, we describe how the different phases of the process were conducted during the project realisation.

## 3.1 Process description

The development team for this project was small and we were not able to rigorously follow any methodology or software development process in order to build our functional prototype. Instead we used parts from Rapid Application Development and from Agile software development to construct the system [14].

The team broadly comprised three elements: one developer to produce the business logic code and architecture; one designer to identify and create the proper interfaces and interactions with the end users; and one project manager who guided the whole process of development and acted as a facilitator bringing together a wide range of stakeholders from inside and outside the business.

The two chosen methods/models are proven to produce high quality software in a short period of time with small teams.

To demonstrate the properties we have followed from each process model we will use the table below:

| RAD | Agile | Used |
|---|---|---|
| Based on prototypes that are improved into production code. | No prototyping is allowed. | RAD |
| Modify the first prototype and so on. | Break the solution into features. | Agile |
| Managed by project manager | Team members are self-managed | RAD |
| Developers work as individuals. | Focus on communication and group designing. | Agile |
| Not flexible for changes. | Continuous integration, making the team ready for changes. | Agile |
| Demonstrate prototypes, mockups, etc. | Demonstrate completed work, production ready. | Agile |

Table 2 Processess properties

The development process started with a brainstorm meeting involving a variety of stakeholders in order to identify key aspects of the project. To begin with, we tried to identify the users of the system and select the most important ones. Then we focused on identifying activities those users would most likely perform on the system and how frequent and important these activities would be.

In addition, we have conducted perspective-based UI inspection of the system. This inspection occurred during one of the prototype evaluations with the airport stakeholders. By using this method we were able to find solutions to interface problems and generate ideas and requirements.

The developer and the designer were free to make some decisions about the overall architecture and the code but everything else was done through project manager

decisions. Correspondingly, the project manager validated the work done and decided whether or not any changes were required.

Whenever a new service was developed it was rapidly integrated into the system and validated. Any changes to the developed service were restricted to the service itself, thus making the change process very flexible.

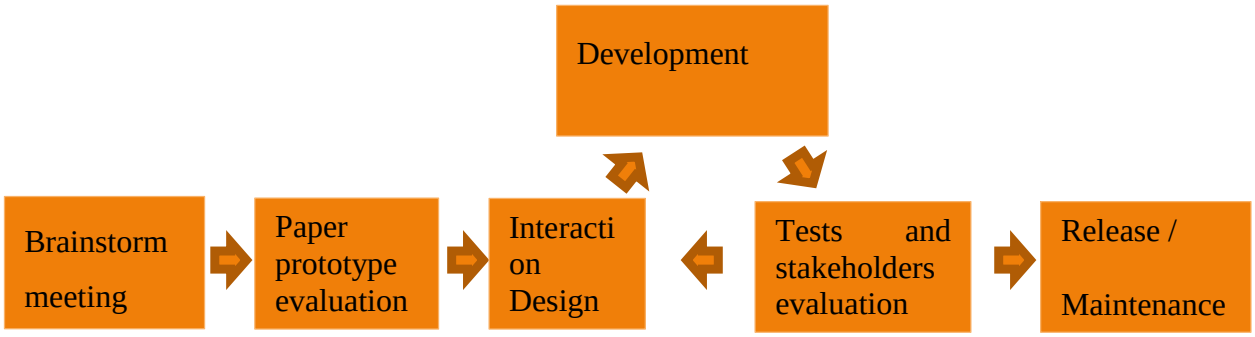



Figure 7 - Development process cycle

Weekly meetings took place at the university rectory to keep track of the work done and to assign new tasks to the team.

We can conclude that the mix of processes was beneficial for the development team as we were finally able to release high quality software and well-designed architecture.

## 3.2 Development tools

In order to speed up development and keep all the development team members up to date with the latest changes we decided to use a few tools to help us. These tools included IDE, data modelling tools, communication tools and frameworks.

For the kiosk software we chose the Microsoft Windows Presentation Foundation technology. This technology embraces the C# language and XAML (Extensible Application Markup Language) and as a desktop application it is easier to capture the user within the kiosk. In addition, the code is closer to the hardware improving performance. To develop a WPF windows desktop application we needed Visual Studio

(version 2013 Professional was used) and the .NET framework (version 4.5.2 used) which brought a set of useful classes for development.

For the database access in the kiosks we used the Entity framework as a data access layer. This framework enabled us to generate code from the database table structure. With a couple of clicks this updates the whole data model according to the database structure. It also provides powerful and easy-to-use methods to perform any required database operation.

Along with Visual Studio we used the ReSharper plugin. ReSharper is a productivity tool for Visual Studio. The main features include code inspections, refactoring and navigation via commands. This provides some tools to extract dependency graphs and class inheritance from the solution providing a good overview of the architecture. Code completion is also a time-saver and the code snippets it contains are very helpful.

Debugging the interface of a WPF application can be tricky when working with Data Binding. The visual designer from Visual Studio does not know the values that data can have in runtime leaving the space empty in design time. On the other hand, when in runtime, we needed a tool to inspect the interface and make changes to figure out errors. The tool we needed was *Snoop* (available at https://snoopwpf.codeplex.com/ ). This tool can enter the interface objects in runtime and allow us to change values to see the effect it has on the interface.

With regard to the management application we decided that the best solution would be a web application. There are good reasons for this decision. The main reason is that to access a web application a user needs only a web browser. Since browsers are installed on every computer today, everyone is able to access the application. Another reason is that an update is sent once to the web server and all the clients are updated immediately, thus there is no need to update the clients individually.

A web server was already prepared to host the application. It had a MySQL database server and an Apache web server running PHP (commonly known as LAMP stack). To make the development of this application easier, better and faster we decided to use a

PHP framework called Laravel (version 4.2, version 5 was released during the project development). This framework offers many useful features that speed up development. We did not need to worry about database connections, validation rules, etc. because the framework provided a really good layer for those features.

A very good piece of software that is shipped with *Laravel* is called *Artisan*. With artisan we were able to define a database structure in code and then run the migrate command. The real advantage of this tool is that the file containing the structure code for the tables is sent to the code repository and all team members can pull it and run the migrate command to update the local development database.

We have used a couple of different IDEs, because some features were better in one IDE than another. For example, for hot fixes on the master branch, the best and fastest way to achieve them was to use *Eclipse* together with *RSE* (Remote System Explorer) to make the fix on the running code and then push it to the master source. All the other branches then had to merge (rebase) the new master branch position. For local development, *NetBeans* was used as it better supports, in the opinion of the development team, the web languages used (PHP, HTML, CSS, and JavaScript).

For code sharing we created two distinct repositories, one for the kiosk application and another for the web management application. We have used Git as the repository and the code is hosted at BitBucket.org.

As a communication tool we created an account with *Trello.com* and a project to manage, organise and share information with the team. This tool is a board where we create columns and in these columns we add cards. This helped us to keep track of the design changes the designer was working on and the implementation features the developer was handling. It also allowed us to post comments and pictures on these cards so that updates to the team were made in real time.

## 3.3 Brainstorm meeting

At the very beginning of the development process a brainstorm meeting took place at the Madeira University rectory.

This meeting was intended to identify the system requirements along with the possible types of users of the system and describe the key features we could provide to the final users in order to increase their satisfaction when visiting Madeira Airport.

A wide range of perspectives was provided at this meeting. In this group were people from the technical department of Madeira Airport, one person representing Intertours (Tourism Company), both the designer and the developer from the application *HereWeGo* and a representative from the Regional tourism office.

Our agenda at this meeting was to identify the users we expected to interact with the system, the activities these users would be most likely to perform and to select the most important activities from those identified. Here, the contribution from everyone present at the meeting allowed us to understand what they expected from the system. There are some similarities from [15] where the users and their activities are identified in order to understand better the user experience of the system in use.

### 3.3.1 Identifying the users

Identifying the key user is crucial to knowing who is going to interact with the system. The result of this phase would have a great impact on the services, layout and key features to be implemented in the system.

At first, we asked all the participants to write on index cards the possible users of the system. There was no limit to the number of users to be identified by each participant, and of course there were no wrong answers as it was left entirely up to the participants.


Figure 8 Users ordered by importance

As a result of this activity, we were able to identify several different types of users of the system. Before ordering and selecting we removed any duplicates. Then, from the resulting set, we asked the participants to order the index cards in two different ways: firstly to order the cards from the most important user to the least important; and then to order them from the most frequent to the least frequent. After ordering, we had a clear picture of the real users that would be interacting with the system.

| User | Importance | Frequency |
|---|---|---|
| **Tourist** | 1 | 1 |
| **Traveller** | 3 | 2 |
| **Emigrant** | 6 | 3 |
| **Nature lover** | 4 | 4 |
| **Assistance** | 5 | 5 |
| **Reduced mobility people** | 8 | 6 |
| **Families** | 7 | 7 |
| **Travel agents** | 9 | 8 |
| **Visitors** | 2 | 9 |

| | | |
|---|---|---|
| **Surrounding population** | 10 | 10 |

Table 3 System users

The key user identified as being expected to interact with the system was the tourist.

Afterwards, we asked the participants to order the identified users based on the arrival and departure index cards. This task revealed that the key users that were identified by importance and frequency were closer to departure than arrival. This result was important because we could then build the system based on features that would be more useful to users departing from the Airport. Actually, giving the possible fear of flying or even the stress experienced by some passengers due to the whole check-in process, security control and passport control, we concluded that the best moment to capture a user's attention was immediately after security control (x-ray) or, where necessary, passport control. At that moment, the passenger will have passed through all the necessary checks and can spare some minutes to explore the system.


Figure 9 Users ordered by arrival or departure

### 3.3.2 Identifying activities

Later we asked the participants to identify the activities a passenger performs at the terminal during his visit. For this task, we took into account the results from the previous task so the activities were identified with the tourist in mind, as the most important user of the system.


Figure 10 Identifying user activities

We kept the same model used to identify the users by asking participants to write on index cards the main activities a tourist could perform at the terminal taking into consideration whether it was an arrival or departure activity.

| | | |
|---|---|---|
| Internet access | Events | Entertainment |
| Visit infrastructures | Get updates | Search for transportation |
| Search for information | Search for bathroom | What to do? |
| Check-in | Flight schedule | Available services |
| Weather information | Destination information | Assistance |
| Where to eat? | Shopping | Information |
| Kill time | Play | |

Table 4 User activities

### 3.3.3 Selecting activities

At this point, the key user and the main activities he could perform have been identified. With that information in mind, the participants were asked to place the most important features around the professor's iPad.


Figure 11 Activities selection

In the image above, the iPad is not present because it was used to take the picture. It is also clear that the Internet Access is not present either because it was placed right on top of the iPad as this was identified as the most likely activity to attract the user's attention.

| | | |
|---|---|---|
| Internet access * | Events * | Entertainment * |
| Visit infrastructures | Get updates * | Search for transportation |
| Search for information * | Search for bathroom | What to do? * |
| Check-in | Flight schedule * | Available services |
| Weather information * | Destination information * | Assistance * |
| Where to eat? | Shopping | Information |
| Kill time | Play | |

Table 5 Activities selection by importance (* represents higher importance)

The activities that were nearest to the professor's iPad were the most important to the system and are marked on the previous table with an asterisk (*). As mentioned before, the Internet access was put on top of the device representing the most important activity for a passenger to start interacting with the system.

At the time this meeting took place, Madeira Airport did not have free wi-fi access to the internet, but while this dissertation was being written the airport implemented free

internet access for its visitors. We can therefore relate our findings about the wi-fi access being a key feature for our system to the finding made by [13] the InfoGate system, but their finding goes beyond ours and states that wi-fi access is key to getting in touch with the mobile user, a fact that did not apply to our system.

### 3.3.4 Results

As a result of this meeting (workshop) we were able to identify several aspects related to the system development. The internet access was selected as being the most important feature to attract users to the kiosk.

Lastly, we were able to identify a set of services to be implemented based on the identified activities a passenger might perform at the airport, such as:

- Wi-fi
- News
- Weather
- Flight data
- Direction information
- Media gallery
- Events Schedule
- Where to eat

In the next step we will filter this list in order to fit the time box selected for the project. The number of services to be implemented should be around five or six depending on the complexity required for each service.

It is also a point of interest that the shopping activity was not selected as being one of the most important activities despite being mentioned very forcefully by some participants during the meeting.

The participants suggested that the system should have two distinct moments: an attraction state and an interaction state. The attraction state is where the system is

showing the available services and random information in order to attract the passenger's attention. We embraced that suggestion and followed it through.

## 3.4 Paper prototypes

Later, we conducted interviews with pre-selected users and presented a paper prototype with pieces that, when combined, constituted the interface of the system.

We prepared a document with some tasks that the user had to perform in the system, but these tasks were not exclusive and allowed the user to explore the system freely and present suggestions of changes to the interface.

### 3.4.1 Tasks

The tasks were prepared considering the two-state system: an attraction state, where the system is showing the available services as tiles that present information about the service; and a usage state where the user has selected a service and the interface changes to display the service content, rearranging the layout.

At the first point (attraction state) we asked the user how to select a service, "What would you do to select an application?" In addition to this we have asked for proposals for the layout of that mode by showing some interface alternatives.

At the second point (usage state) there were a few more tasks to be performed because, at this point, the system was active and waiting for interaction. The prepared tasks were:

- How would you change to another application?
- How to close an application?
- How to logout of your session?
- Where should important information be located?
- How to find a hidden application?
- Where should an advertisement go?
- Where should warning messages appear?
- Do you have suggestions for this system state (usage state)?

### 3.4.2 Paper prototype evaluation

During this phase three interviews were conducted and at the start of every interview, we prepared the interface with the pieces at random locations. In doing this we expected the users to present different alternatives for the interface and in the end we would merge the ideas into one final solution.

We used techniques like *Think Aloud* where the user, while interacting with the system, says what he is thinking. This helped us to understand what was on the mind of the user when interacting so we could prevent user errors.

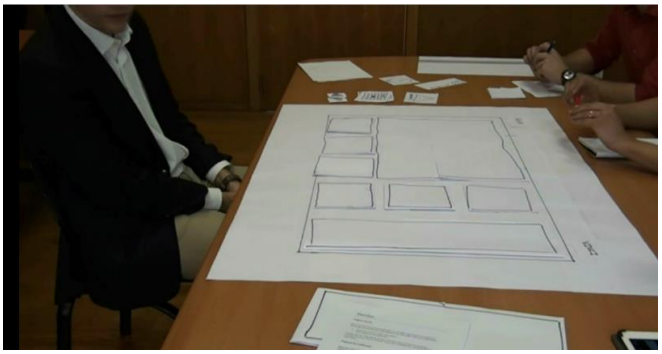


Figure 12 User interview

The prototype was designed to be close to the ergonomic of the real kiosk. The table-like ergonomic of the kiosk and the upward angle put some limitations and constraints on the arrangement of the action buttons on the interface. The size of the prototype was designed in scale to the real kiosk.


Figure 13 The kiosk equipment

During the interviews we noticed that if the action buttons were placed at the top of the screen, the user's arm stays in front of the rest of the application preventing them from seeing what is happening with the application. In addition, it is hard to reach the top of the screen by stretching the arm. Taller people would have no problem with this but shorter people would feel very uncomfortable trying to reach those action buttons. Furthermore, if we put the services on the left side of the screen, the user's arm stays, again, in front of the screen, thus blocking the interface.

### 3.4.3 Findings

After all the interviews, it was clear to everyone involved that the two-state system was the best course of action. One thing that was unexpected was that, for every user interview, they wanted to go back to the first interface without losing their session. This meant that a user could navigate back to the attraction mode at any time.

Placing the action buttons at the bottom of the screen was essential in order to give the users a comfortable experience and make the system easy to use.

The single touch to select an application was unanimous. Long tap was described as being a hidden feature and should be avoided.

Several suggestions were made regarding the layout of the active mode of the application during the interviews. By the end of all the interviews the users had come up with very similar suggestions for the layout. It was suggested that the selected application should have a dedicated area on the top part of the screen and the list of available applications should be placed at the bottom of the screen in miniature format.

Application-specific tools and actions should be placed at the bottom of the application area following the logic applied to placing the services at the bottom of the screen.

In the paper prototype we were unable to replicate the friction felt while performing drag operations on the real screen. The touch technology existent in the kiosk is outdated. What happens is that the friction on the monitor is so great that after performing a couple of drag operations, the finger gets hot and the user does not want to continue with the task that required the drag. As a result, all of the operations can be performed with a single tap.

It was suggested that advertising should be placed on the right of the screen. As noticed during the interviews, when the advertisement was placed at the right, the users centred themselves with the monitor to the left, where the selected application content was placed.

The languages menu was placed at the top of the screen, since it was agreed that this action would not be required frequently. Despite its having been said that the home button was important, interviewees did not use it as often as expected.

To logout of a user session, the application should be aware of the length of time the user has not interacted with the system and logout that session clearing all their data. Apart from the inactive time, the user can actively press a button to log out. The most important session data that should be cleared is the flight data. Leaving personal flight data on the screen could be seen as a security threat.

The persistent flight information data area was placed on the right side of the screen below the advertisements, following on from the finding about the user being left centred in respect to the monitor. Placing this information on the right, below the

advertisements, allows it to be permanently visible to the user. In this way, the user is easily alerted when his flight status changes so that he can take the appropriate measures.

We asked the participants to suggest the most important applications to implement given the list created during the previous phase. The services selected as the most important to be developed were:

- Flight Info
- Weather
- News
- Wi-fi
- Media gallery

To add to this list is a service with maps of the terminal and the locations marked with pins. This service is mandatory since it was a big concern for the airport manager.

## 3.5 Conclusions

Following a well-defined process such as the Agile or RAD models is very hard with a small team such as ours. We tried our best to follow a process adapted to our needs.

In summary, all the steps taken during the process were necessary to reach the end result.

The end result fits the needs of both the development team and the stakeholders of the system.

# 4 DEVELOPMENT

In this chapter we will discuss the architecture that emerged from the requirements obtained at the Quality Attribute Workshop in Chapter 3 and its development. We will also point out some quality attributes that the requirements have constrained.

The technology required to develop both the kiosk software and the management application will be investigated to provide an overall understanding of the communications and how they are related.

This chapter is focused on the engineering details of the platform and the use of diagrams and code snippets will be made in order to illustrate the architecture, strategies and decisions made during development.

## 4.1 A first attempt

Before the integration between the dissertation project and the airport assigned task was accepted, a first attempt was made to build a similar system for a couple of internal (not public) kiosks.

The developed software was a single application, one project. As a result, whenever the software needed modification, the whole code had to change (or be commented out) leading to an overhead regarding its management.

As a result of this attempt, the plugin system was firmly decided as the best way for the system to be developed. It is definitely a challenging architecture to build but in the long run, we gain in flexibility, maintainability and modifiability.

## 4.2 The overall system

The system is composed of two main applications: the web management application; and the kiosk software. Together they form the platform developed for this project.

The kiosk software is the public application that runs on the touch-screen monitor and is responsible for booting the template and plugins with respect to the monitor on which it is running.

The web management application is where the properties of each monitor, plugins and templates can be configured and the data for the developed plugins can be edited as well. It is possible to select which template each monitor is using, the plugins the monitor has to pass data to the plugin, setup default language, and to set the monitor location in the terminal, among others.

### 4.2.1 Quality Attributes

Given the context in which the system is installed, we had some major quality attributes to address. The quality attributes mentioned in this chapter are inherited by both developed applications, as they must guarantee the quality level for the entire solution. We will cover each application later in this chapter.

#### 4.2.1.1 Security

Solution-wide security must be granted in order to avoid interference with airport operations. The main concern was to place the solution on a separate network to the one used for airport operation and staff. This way, the solution will be confined to a small network created to hold only digital signage traffic.

#### 4.2.1.2 Availability

The solution is available as long as both applications are up and running. Internet access must be granted in order to download live data when needed. Data caching is performed in both applications to prevent data overload which could crash the system.

#### 4.2.1.3 Maintainability

The maintainability of the solution must be guaranteed for all its parts. Changes to the network or the database server used by the solution should be easily configurable in each application.

### 4.2.2 Architecture

We did not follow any software architecture methodology like ATAM (Architecture Tradeoff Analysis Method) to develop the system. Instead, we performed a very light QAW (Quality Attribute Workshop), described in Chapter 3, in order to ascertain the main requirements of the system.

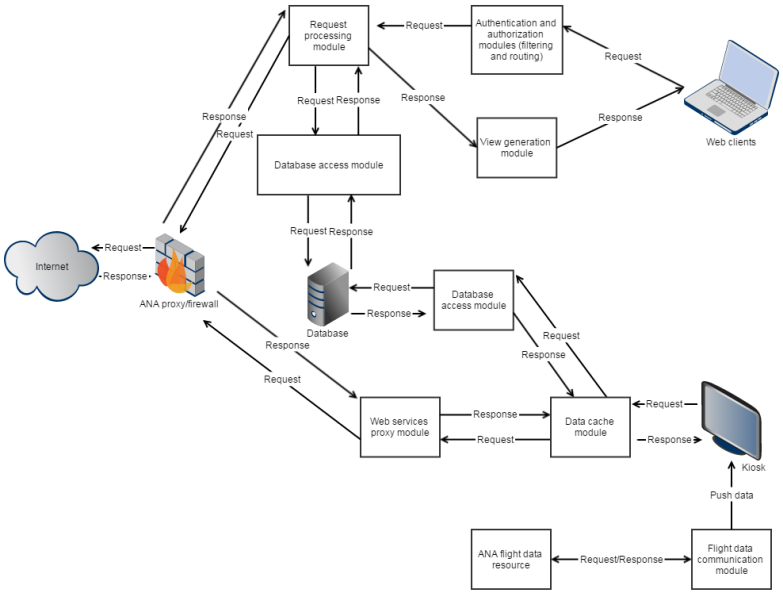



Figure 14 Overall architecture

Overall, the system can be broken down into modules that represent implementation decisions. Each module may represent a small feature or a full set of features. In Figure 14 all modules that comprise the system are shown in high-level detail.

For instance, the flight data communication module represents a soap server, a soap client, a web socket push server and a web socket client. We implemented a soap client and a server for the ANA flight data resource implementation that requires this architecture. The push server grabs the data from the ANA flight data resource and broadcasts to all the registered web socket clients.

The data cache module represents a single class that must verify the age of the last request, if it existed, and decide if a data refresh is needed. If so, the module should then perform either the desired database query or gather the data from the internet.

The monitors do not have internet access, so data from the internet must be gathered from a web proxy that performs that request. We included all the necessary services into our web server as an API service that collects the data from the internet and returns it to the kiosk clients. All internet access from inside the airport is done through a proxy that also acts as a firewall. As a result, we have configured our web server to use that proxy server to access the internet.

The database access module for the web application uses the Laravel framework Eloquent module that maps the database into classes and facilitates the access. At the kiosk, the same mapping is done but this time using the EntityFramework technology for C#. The database is a MySQL engine.

The other three modules work together to accomplish the web application. They were developed by the Laravel team and we make use of those components to achieve the quality attributes for that application.

In the next few chapters we dig deeper into both applications and the developed plugins, focusing on the architecture and the technology stack used for each.

## 4.3 Web management application

The web management application, as the name says, is a web application that allows users to configure the monitors, plugins and templates individually and perform CRUD (create, retrieve, update, delete) operations on the plugin configuration data.

This application not only allows the plugins configuration, it also allows managing the actual data that the plugins provide. We will cover this part later in this chapter.

### 4.3.1 Quality attributes

At the start of the development of this application we identified a set of quality attributes to be considered.

#### 4.3.1.1 Security

In our context, this attribute has two sensitive points: authorisation and authentication.

Authentication is the way in which the user of the system can be identified.. An authenticated user is a user that the systems knows. The method of fulfilling this need was a secure login system where the user is authenticated by entering their email and password. The passwords are stored in the database ciphered with AES-128.

Authorisation is the method by which an authenticated user is granted permission to perform operations in the application. To achieve this property we have added a feature that associates users and roles. A user is created during installation and assigned a role that allows him to manage the users and their roles. The role of managing users and roles is assignable to other users. With this system we can give access to new users but authorise them to perform only certain actions.

However, it is very unlikely an unauthorised user or hacker could gain access to the application since it is inside a private network managed by the airport IT department.

#### 4.3.1.2 Availability

We have to guarantee that the server (web and database) is up and running with very high levels of availability and we need ways to identify and be notified if the server

goes down. To this effect the web server is being monitored by the *cacti* software (available at http://www.cacti.net/). This software sends email reports if the server goes down or encounters other errors such as disk failure or high load average, using the SNMP protocol.

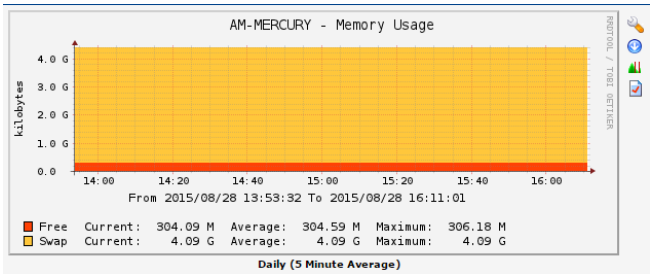



Figure 15 Cacti monitoring tool

Figure 15 shows the memory usage graph on the system server.

The server is a virtual machine so its availability is highly dependent on the virtualisation system implemented at the airport. The application is available as long as the web server is available.

In addition, we have a software package called NimSoft installed at the airport that can ping all our systems and send notifications via email and SMS when a system goes down. It consists of an agent installed on a machine that has access to all networks. It is in BETA mode but the results are quite satisfactory.

#### 4.3.1.3 Maintainability

The PHP framework used for this project comes with a lot of potential for maintainability. The framework provides a means to define modules in order to isolate functionality, to find and correct defects and their causes, or easily replace the entire modules.

From the MVC architecture we acquire the power to include business logic in the controllers, data models in the Models and the interface in the Views. One Model corresponds to one PHP file, one controller to one PHP file and the views can be spread

over several files and included in another view file using the Blade system (an HTML pre-processor) to facilitate the process and reuse HTML code.

Laravel provides a single file (routes.php) where we place all the routes the application has. In this file we can manage the services the application provides, making it easy to remove or add routes. Together with the route definition, we define the filters to be applied to each route (similar to aspect-oriented programming).

There is another mechanism to isolate modules called *Service Providers*. With this mechanism we isolate the modules/services inside reusable classes completely independently from the business logic application.

### 4.3.2 Architecture

The overall architecture of a web application is always the same: a web browser; a web server to process the request; and, in most cases, data storage. The way the web server handles the requests can differ from one project to another.

The architecture of this application follows the architecture of the chosen PHP framework. The MVC pattern is fully embraced, as are all other mechanisms that are shipped with Laravel.

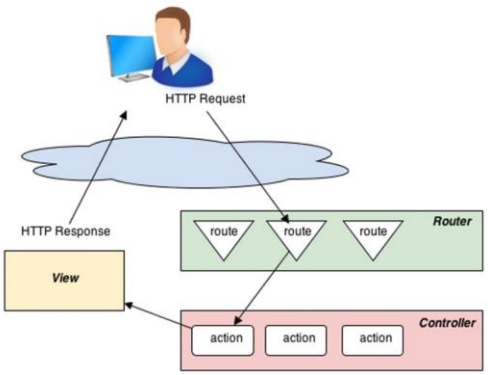



Figure 16 Laravel 4.2 Architecture (available at http://laravel-recipes.com/recipes/52/understanding-thE-Request-lifecycle on February 2016)

We have also included two of the most popular front-end development frameworks: JQuery and Twitter Bootstrap. These frameworks are very handy when developing the front end of a web application.

In the scope of this project we have developed three more components within the Laravel framework: a push notification service to send arbitrary data to the clients when needed via web sockets; a soap client to access soap services; and a soap server to handle soap requests. These three modules were specially designed to work with the flight plugin.

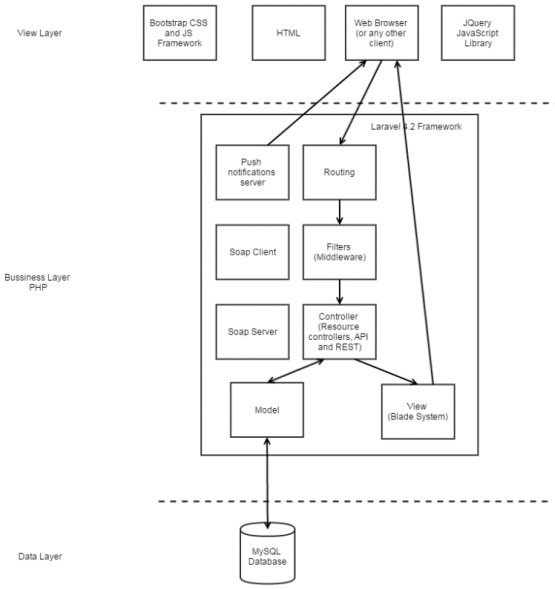



Figure 17 Technology stack

### 4.3.3 Strategies and implementation decisions

As we started the project using the Laravel framework the main concern at first was the data model and not the MVC architecture of the application as that is provided off-the-shelf by the framework.

The data model for the solution should reflect the core of the system. The system is composed of monitors that should have one template each and a list of plugins. The

template or a plugin could have dependencies (zero or more external libraries) for proper execution. A monitor can be displayed in many languages and have one as default.

With these concerns in mind, the core of the platform requires a few entities to start with. The entities are:

- Monitor
    - A data model representing one physical monitor.
- Plugin
    - Representing a single plugin.
- Template (Initialpage)
    - Representing the available window managers the monitor has
- Dependencies
    - Dependencies that a plugin or a template has.
- Languages
    - Languages that the system has.
- Airport (Weathercities)
    - The airport the monitor is located at.

With these entities we form the base to implement our plugin system configuration. A couple of steps are necessary to start using the system.

To start from a clean database a dependency needs to be created. It is necessary to create all the dependencies that the template and the plugins could use indicating the name of the DLL file of every dependency. After that, the dependencies are available to be associated to the plugins and/or templates.

The second step is to add at least one language to the system and the airport where the platform is going to be installed.

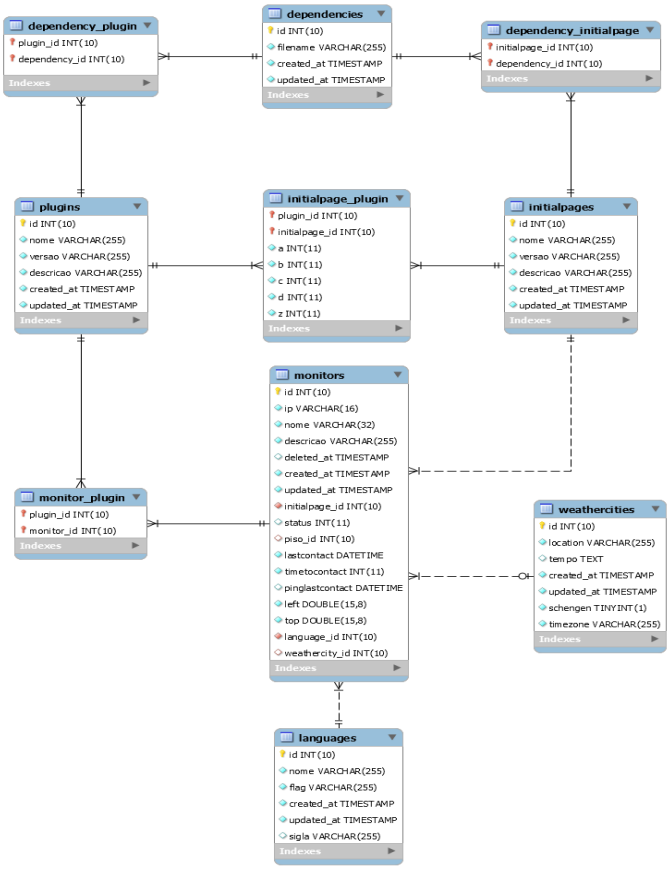



Figure 18 Database structure

Later, we must create the template we have developed by indicating the DLL file name that contains it and assigning the dependencies it was implemented with, previously created in the application.

The next step is to create the plugins we have developed, in the same way we created the template (the file name), but this time it is necessary to complete the name in every language added to the system, and indicate its dependencies.

At this point, we have everything ready to create the monitor. When creating the monitor we indicate its IP address, the template, the default language and the airport, among other things.

The IP address is the way in which we uniquely identify a particular monitor, as IP addresses are unique inside a network. As we will see later, the kiosk software retrieves the computer IP address and queries for its data (including template, language, plugins, etc.).

Before launching the application we need to configure the plugin with respect to each template, i.e. set the necessary data for the plugins to launch correctly in that template.

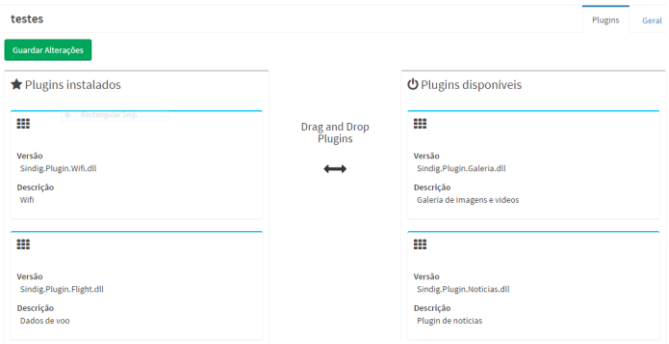



Figure 19 Managing plugins

Figure 19 shows the interface developed to install or uninstall a previously created plugin into a monitor.

The templates are represented in the database as initialpages. The name was created in the early stages of development and changing it later is not recommended. The same happened with the airport data model stored in the weathercities table. Initially that table was used only by the weather plugin but later on we found that it would be a good idea to identify the airport where the monitor is, as a generalisation technique.

The user roles implementation consists of a many-to-many relationship between the user and the role tables. In the back office we manage these roles and assign them to the users allowing them to perform the actions for which they are authorised. Then, in the

code, we use a filter to protect the routes by fetching the authenticated user roles and checking whether the user has the necessary role to access that specific route.

It cannot be emphasised enough that security is a major concern in this application. Authentication and authorisation were developed with special care and the development of new services must use the tools developed in order to protect the routes from undesired access and management.

## 4.4 Kiosk application

This is the software destined to be executed on the touch screen located at the airport. We have chosen the Microsoft WPF technology to develop the system. It is responsible for loading the correct template and installed plugins of the monitor. At its core it is a bootstrap code that uses reflection in order to dynamically load all the dependencies, the template and the plugins configured at the web application.

### 4.4.1 Quality attributes

This software had a few constraints regarding quality attributes as it is for general public usage and it resides inside an airport network.

#### 4.4.1.1 Security

This is a very sensitive factor as the application is executed inside an airport infrastructure. If for any reason a user of this system can access confidential information, the project will have failed.

The first step to ensuring the safety of the software and the airport was the decision that the monitors and all necessary servers should be (or at least have a network interface) in a network separate from the airport operational network. For this reason, a sub net was created to isolate the whole system from possible unwanted data access.

Next, we decided that the monitors should not have internet access. This decision resulted from previous bad experiences where users were able to get out of the kiosk system and open a web browser to surf the web. Two problems arise from free and

uncontrolled internet access. Firstly, users could consume a lot of the internet bandwidth if they accessed the browser and navigated to YouTube for example. Secondly, inappropriate content could be displayed, causing embarrassment to the airport. To address this constraint we built our web server with two network interfaces, one to support the monitors and the other in the employee network which has internet access. This decision helped us later when we needed to perform some caching of data to provide faster response times to improve performance and general user experience. For this reason, the web server acts as an internet proxy and all data that the kiosk needs to access from the internet needs to be requested to the server, the server fetches the request from the internet and sends the data back to the kiosk.

#### 4.4.1.2 Availability

As it is in a general public area, it is very important that the application is up and running, is continuously available and that it never closes. There are two concerns regarding application availability. Firstly, the network state of the monitor needs to be monitored and the management team needs to be notified if something happens. The second concern is to know if the application is being executed or if it is closed.

For the first point, we developed a component that performs a ping operation every ten minutes to every monitor registered in the management application. If the ping fails, the management team receives an email notification so that the correct action may be taken. If everything fails, the system blocks all operations and shows a message that says that the network connection is down. Even if a fatal exception occurs for any reason, the kiosk software restarts itself.


Figure 20 Email notification

The second concern is a bit harder to detect. How do we identify whether an application is running in windows or not? The first idea was to implement a windows service that continuously checks to see whether the process is being executed. But this did not solve the problem because even if the process is running it does not mean that the application is up. So we decided that every X minutes (configurable in the monitor's editing page) the monitor will update a field with the current date and time. It works the same way as the ping component. Every hour a PHP script is executed to test the dates and times the application last updated the database field. If the monitor did not update the field when it should have done, an email notification is sent to the management team.

#### 4.4.1.3 Maintainability

This software was created from scratch without any framework (except from the .NET framework) so it was important to define the interfaces and classes before implementing the code.

Since we have implemented the platform as a plugin system, any changes to the running software can be made without changing the current code unless we modify the existing interfaces. New plugins and templates are easily created in separate projects.

The plugin system really helps maintainability but this is just the base where the plugins and templates sit. We needed a way to separate concerns like models, views and business logic.

This is where the MVVM pattern comes into play. With this pattern, the models, viewmodels and views are separate. This pattern makes good use of the .NET framework and WPF mechanisms like Commands and Databinding.

Commands are central points where we attach a method to be executed if it can be executed, decoupling the event handlers in the code-behind file from the views and moving the logic into classes that can be reused. A couple of implementations of the *ICommand* interface were created because some command methods needed to be executed asynchronously and others synchronously.

The data binding mechanism is achieved by implementing the *INotifyPropertyChanged* interface. We have created a base class, *ObservableObject*, implementing this interface and every time we want a class to expose properties that should notify all the bindings to it, we just extend it and raise the property changed event on the *set* method of that property.

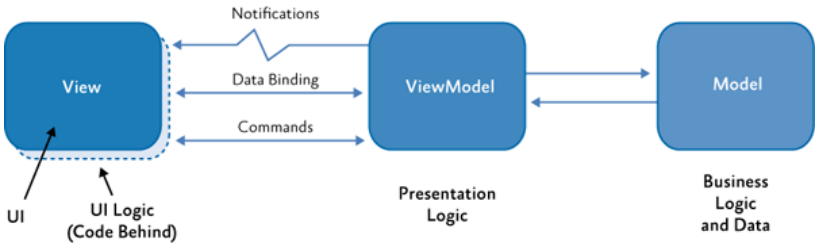



Figure 21 MVVM Architecture (available at https://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx on February 2016)

Sometimes, it is necessary to set the monitor into a maintenance state. That is, if for some reason we need to perform some server maintenance operation or any other maintenance we need to tell our users we are out of service. For this action we have created a maintenance mode on the host application. In the management web application we can tell all monitors (individually) to go to maintenance state and as soon as the monitor is inactive for at least 10 minutes, the application will restart and the monitor enters the maintenance state. When maintenance is finished, we can send

the restart command to all monitors via the web application and the monitors will restart out of the maintenance mode.

### 4.4.2 Architecture

A WPF application uses XAML (extensible application markup language) to define the interfaces (Views), C# (or Visual Basic) for the business logic and the .NET framework as a basis for writing high performance and better code.

As already mentioned, the architecture of this application is based on a plugin system where we can install/uninstall plugins at any moment via the management application without requiring changes in code. The system should be thought of as a platform on which the plugins (applications) are running. Thus the client application (the kiosk software) is just a host application for the plugins.

This application accesses the same database as the management application but has a different user with restricted permissions for reading only the data required to boot the system. The software was developed using the MVVM architectural pattern to provide a clear separation of concerns between the business, data and view layers and making it easy to change interfaces without changing the classes (maybe just extending them for new functionalities), easy collaboration with the developer/designer and really decoupled code as the viewmodels have no knowledge about the views but the latter uses the viewmodels properties for the data binding mechanism.

The start-up project is called *Sindig.Boot.* The base code of this project is responsible for reading all the data related to the monitor where it is being executed including which plugins, template and dependencies should be loaded. After this, it copies the plugins and template files (and their dependencies) from the plugins repository located in the network shared folder and places it in a local folder, loads the assembly into memory and instantiates an instance of each plugin and the template, passing to the latter a list containing the plugin instances.

A plugin has some data associated with a template. This data is used by the template to set the arrangement of the plugins on the interface or perform any other operation it finds necessary. This is the behaviour of the template developed for this project but the use to which the template puts that data is completely up to the template developer.

### 4.4.3 Strategies and implementation decisions

In order to accomplish a fully functional plugin system it was necessary to create a method where two entities with no knowledge of each other would be able to communicate, execute methods, handle and raise events in both ways. This is done via an intermediate interface of which both entities are aware. To achieve this, we have created a new project called *Sindig.Core* where all the interfaces reside and should be referenced by the host application (the kiosk is a C# project called *Sindig.Boot*) and by the plugins (the templates also reference this application but they are not considered to be plugins; they are window managers).

The template works as a window manager and is responsible for displaying the plugins and how the interaction works. Architecturally speaking, the host instantiates the plugins and the template is instantiated by passing the plugins list to its constructor.

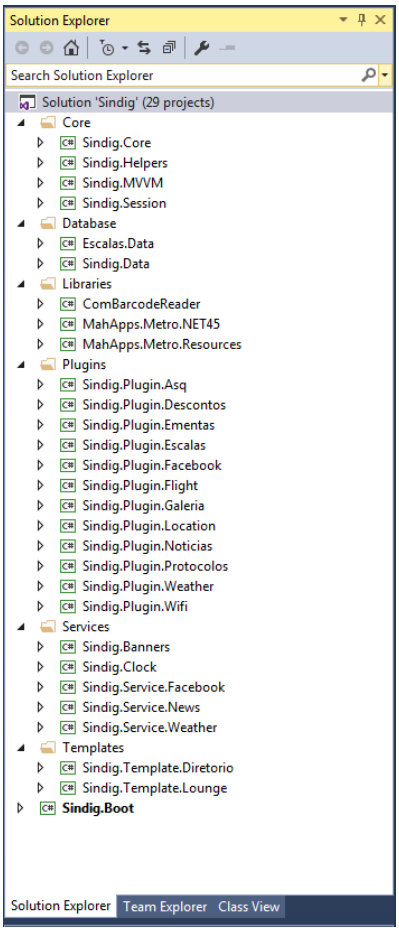



Figure 22 Solution organisation

In order to instantiate new instances of unknown types, the reflection technique was used. After loading the plugins and template symbols (contained into DLL files) we were able to instantiate the objects passing the arguments necessary to construct them. C# provides a number of useful classes to perform the loading. We open the DLL file that contains the symbols we want to load and try to find a class that implements a certain interface. If that class is found, a new instance is created and returned to the caller; if it is not found, null is returned.

The name given to that project is *Sindig.Boot* as, in its nature, it really is no more than a bootstrap code that joins the template and the plugins implementations via the common interfaces.

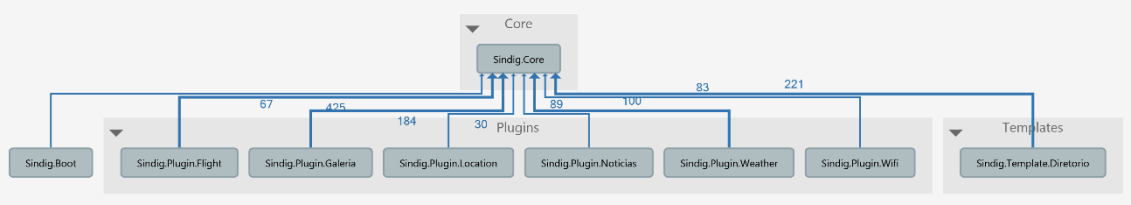



Figure 23 Dependency Graph

For this project, we have developed one template and six plugins. The decision to use the template system alongside the plugin system derives from fact that the airport owns other kiosks with different purposes but the management should be done in one single place, the web management application.

#### 4.4.3.1 Session data

During development it became clear that data such as the current language Id and abbreviation, the monitor Id and IP address and the monitor floor number and Id should be kept in a common place where it could be accessed by any component of the system.

For that purpose a static class was created to hold all this data.

```
namespace Sindig.Session
{
    public static class ApplicationData
    {
        public static string LanguageAbbr { get; set; }
        public static long LanguageId { get; set; }
        public static long MonitorId { get; set; }
        public static long MonitorPiso { get; set; }
        public static long MonitorPisoId { get; set; }
        public static long TemplateId { get; set; }
        public static string MonitorLocation { get; set; }
    }
}
```

Figure 24 Application data static class

We had to store any arbitrary data at fixed indexes and this data should raise the property changed event when its value changes. To achieve this behaviour we used an ObservableDictionary where we can store any object in any index represented by a string. Also, as this is a user session class, we store the user session Id represented by a hash that changes every time we detect a new user.

```
namespace Sindig.Session
{
    public class SessionData : ObservableObject
    {
        private static readonly SessionData instance = new SessionData();

        private IDictionary<string, object> _data;

        public string Guid { get; set; }

        private SessionData()
        {
            Data = new ObservableDictionary<string, object>();
        }

        public IDictionary<string, object> Data
        {
            get { return _data; }
            set
            {
                if(_data != value)
                {
                    _data = value;
                    RaisePropertyChanged("Data");
                }
            }
        }

        public static SessionData Instance
        {
            get
            {
                return instance;
            }
        }

        public void Logout()
        {
            Data.Clear();
        }
    }
}
```

Figure 25 Session data class

This class was developed as a singleton, because at any time, only one instance of the session should exist. Session data is cleared when we identify that the user has gone. When a new user is identified, we generate a new session id.

Finally, the logging mechanism works together with the session management. This mechanism takes care of logging all the user action and clearing their data at the correct time. The tactic we used was to record the last user action Timestamp and with a timer we check if it exceeds the five-minute maximum. If so, we clear the user data, logging the user out. To make it work we had to call a Touch method of the logging system for every user action on every plugin, updating the last action to the current date and time.

### 4.4.3.2 Database access

In this project we have used the MySQL database driver to store our data. The database resides on the same server as the management web application, so the network access is granted.

The database model was already generated in the management application, so we needed to map the database structure into classes in the kiosk software. Entity framework does this job for us by generating the models from a database so we will not explain how it is done.

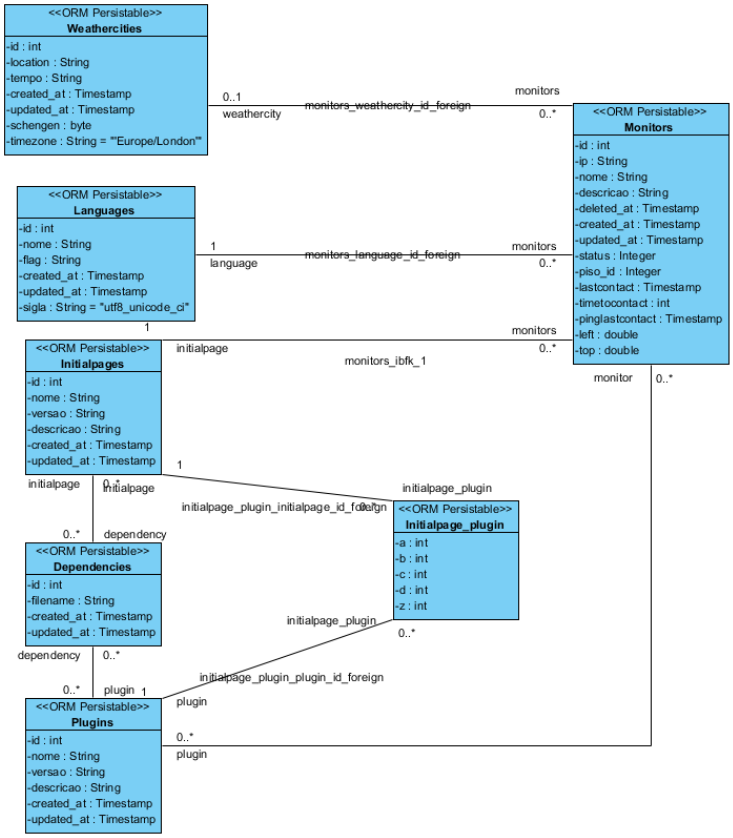



Figure 26 ER mapped to class diagram

In order to connect our kiosk software to the database we used the MySQL ADO connector for the .NET framework (version 6.9.6). With this connector installed we were able to write the code to access the database data.

In the code, we have used the entity framework 6 technology to access the data. This framework allows us to find the database address (along with username and password) and it automatically generates the database model (an E-R diagram), the classes and relationships that represent the database tables. The relationships are called navigation properties. For every table in the database, it generates a class and creates properties that represent the relationships.

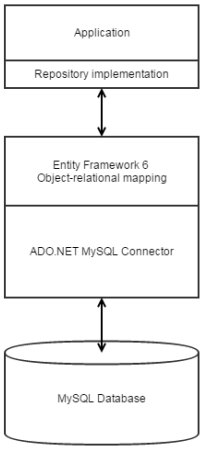



Figure 27 Database access architecture

After generating the classes that represent our database, it is possible to access the database through its context, provided by the entity framework, and obtain instances of the generated classes.

We have developed a repository design pattern to access the database. The code needed to access the database through the entity framework database context is very repetitive and takes a few lines of code. This pattern allowed us to encapsulate the CRUD logic in a single place providing specific methods to perform these operations. A repository is an implementation of the IRepository interface that we have created. By implementing an interface we have gained the ability, if needed in the future, to change the storage used for the project without changing the existing code but developing a new class for implementing the IRepository interface.

```
public interface IRepository<T> where T : class
    {
        IList<T> GetAll(params Expression<Func<T, object>>[] navigationProperties);
        T GetById(int id);

        IEnumerable<T> GetList(Expression<Func<T, bool>> filter = null,
Func<IQueryable<T>,
            IOrderedQueryable<T>> orderBy = null,
            params Expression<Func<T, object>>[] navigationProperties);

        T GetSingle(Func<T, bool> where,
            params Expression<Func<T, object>>[] navigationProperties);

        void Add(params T[] items);
        void Update(params T[] items);
        void Remove(params T[] items);
        void Delete(int id);

        Task<string> AddAsync(params T[] items);

        Task<T> GetSingleAsync(Expression<Func<T, bool>> where, params Expression<Func<T,
object>>[] navigationProperties);

        Task<IList<T>> GetAllAsync(params Expression<Func<T, object>>[]
navigationProperties);

        Task<List<T>> GetListAsync(Expression<Func<T, bool>> filter = null,
            Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,
             params Expression<Func<T, object>>[] navigationProperties);
    }
```

Figure 28 Repository interface

The IRepository interface provides the four main methods twice. That happens because we may need to access the database in an asynchronous fashion. If an asynchronous method calls the synchronous method of the repository, the method will block the execution until the data is obtained. Sometimes the synchronous method might be just what we need but most often, the asynchronous is a better call.

We still have another problem to deal with. The creation of the concrete repository implementation must be encapsulated so if we ever need to change the implementation there will be one single place to change the creation of the concrete type. To achieve this, we have encapsulated the creation inside a property in a factory class. Using a generic type “T” we can generate repositories for any database object in one single property.

```
public class Database<T> where T : class
{
    private readonly string _conStr;

    public Database(string conStr)
    {
        _conStr = conStr;
    }

    public IRepository<T> Repository => new Repository<T>(_conStr);
}
```

Figure 29 Database class

With the data access logic set we can now perform any operation in it with minimal coding and provide greater flexibility to the code and to future changes in the data store.

#### 4.4.3.3 Services

In addition, we have developed the system in a service-oriented fashion. For this reason we have services for weather, news and others. As a result, we can reuse these components and encapsulate the logic in a separate project. These services expose methods that take care of the whole logic to retrieve the desired information.

These services are:

- Banners
  they retrieve the banners given to the monitor where the application is being executed.
- Clock
  is a service with a property that updates a property every second, returning the current time. It uses the operating system time.
- News
  is used to retrieve the news from a web service. The implementation we used takes into account the monitor on which the application is being executed.
- Weather
  retrieves weather data from a web service.

Another advantage of using this service-oriented architecture is that we can send the objects between plugins acting as a communication pipe.

We will explain a couple of services later in this chapter.

#### 4.4.3.4 Binding text to resource key

Sometimes it is necessary for a TextBlock to have a dynamic value. This means that at some point the text displayed belongs to a key inside a resource file (XAML) and at some point we need to change the text to another key. To accomplish this we have created an attached property that we can bind to an element property, which returns the value of the key to be displayed.

To build this feature is fairly easy; we simply register the attached property and update the value of the TextBlock whenever the value of the binding changes.

An example usage of this class would be:

```
<TextBlock helpers:TextBlockResourceHelper.TextResourceKey="{Binding ObservationsKey}"></TextBlock>
```

Figure 30 Binding text to resource key

The resulting text displayed inside this TextBlock is the value of the resource where the key equals the value of the ObservationsKey property. Of course, it is necessary to raise the property changed event inside the class that owns the ObservationsKey property in order to update the text value.

This is not a technique used for the multi-language system. It is a technique to be used when we need a certain TextBlock to have multiple values in the same language.

With this approach we can isolate text messages inside the resource files, leaving our classes clearer, e.g. without the necessity of having to display different text.

#### 4.4.3.5 Plugin flip manager

Following principles to assign single responsibilities to classes we have developed a class to deal with the flipping action to be taken by the plugins. This class is responsible for managing how this feature works.

Every plugin that implements the IFlippable interface is registered inside this class by the template (if the template has the flip functionality). This class sets the time interval that a flip should occur in the interface, which plugins should flip and the maximum number of plugins that can flip at a time.

The selected plugins to be flipped are chosen at random and the number of plugins to flip is also random and it can be zero. When the timer elapses, we calculate how many tiles will flip. If the number of tiles to flip is greater than the maximum allowed, we flip the maximum. Next we randomly generate a list of indexes representing the position in the list of registered plugins, those will flip. With the list of plugins to flip, we only need to call the ChangeStateAsync method of the IFlippable interface for each plugin inside the generated list of indexes.

The flip mechanism is supposed to work only during the attraction mode, so when the template changes to the interaction state the flip mechanism should stop. To achieve this, we added two methods to this class, one to stop the timer and another to start it. This way, every time the template changes its state to the attraction mode we can call StartAnimating and when the template changes to the interaction mode we can call StopAnimating. Otherwise all the logic behind the flipping of every plugin would be called when the timer interval elapses and this logic could include database queries, API calls and so on, resulting in a high data load.

#### 4.4.3.6 The ITemplate interface

To develop a new template we need to add a class library project in Visual Studio, reference Sindig.Core, create a new class and implement the ITemplate interface.

```
namespace Sindig.Core.Template
{
    public interface ITemplate
    {
        Type View { get; }
        ITemplateViewModel ViewModel { get; }
        Task Initialize();
    }
}
```

Figure 31 ITemplate interface

The interface is very simple; it must return the type of the main view, a viewmodel of type ITemplateViewModel (interface contained inside the core project) that will be the data context of the type of view specified and a method called initialize.

The constructor receives a list of plugins so when implementing the class, the constructor must receive an object of type *List<IPlugin>*. This warning could have been avoided either by creating a base class or passing the plugins to the initialie method but at the moment this has not been implemented.

Another consideration is that the host application will call the template constructor from a thread different from the UI Thread, so it is very important that visual operations are executed in the initialize method, not in the constructor, i.e. every operation the template needs to execute on the UI thread must be done inside the initialize method, not the constructor otherwise it will throw an exception when booting.

Furthermore, the host creates a data template for the ITemplateViewModel with the Type View and sets the data context of the window to the template view model. As a result, the interface shows the View of the type specified and the data context is the template view model. From now on the template takes over as a window manager and manages the presentation, layout and template.

The ITemplateViewModel has no method or properties, it is just a way to retrieve the implementation that will be applied to the view.

In order to designate a view for a concrete class we need to create something called DataTemplate. This DataTemplate should be created at runtime because the classes and views are not known in compile time.

```
public static void CreateDataTemplateKey(Type vmType, Type viewType, FrameworkElement w)
        {
            var dt = new DataTemplate { DataType = vmType };
            var fef = new FrameworkElementFactory(viewType);
            dt.VisualTree = fef;
            var novo = new DataTemplateKey(vmType);
            w.Resources.Add(novo, dt);
        }
```

Figure 32 Create datatemplate key and add to view

A DataTemplate indicates that when the DataContext of a certain view (anything that derives from FrameworkElement) is of class A the view to be rendered should be of type viewA. A DataTemplate key is added to a parent View where this DataTemplate is going to be used. If in viewX we have a user control with DataContext set to ClassA, we need to create a DataTemplate key in viewX (the bottom most, but it could be in any other parent) indicating that when the DataContext of that user control is of type ClassA the viewA should be rendered in that place.

For instance, let us analyse what happens to the main window in the *Sindig.Boot* project, which is the loader project. After loading everything (all necessary DLL files and instances by reflection), a new window is created to host the template (which in turn will host the plugins). To make this window host the template, we create, in the code-behind file of that window, a DataTemplate key indicating that when the DataContext of the window is of the Type returned by the ViewModel property on the ITemplate, the view must be a new instance of the Type returned by the View property on the ITemplate interface as well. Then, all that remains is to set the DataContext of the window as the value returned by the ViewModel property of the ITemplate.

```
private void Initialize(ITemplate template)
        {
            SindigVisualHelper.CreateDataTemplateKey(template.ViewModel.GetType(),
template.View, this);
            DataContext = template.ViewModel;
        }
```

Figure 33 Bootstrap initialisation

Finally, there are two steps to dynamically assign a view to a concrete class. Firstly, create the DataTemplate key (assigning a class type to a view type) we are going to use. Secondly, set the DataContext of the element where the view will be rendered.

After this process, the ViewModel of the template is the owner of the window. It does not have any knowledge about the view but it has properties that the view can bind in order to display itself correctly.

#### 4.4.3.7 Template implementation

The template developed for this project is an implementation of the ITemplate interface, so all the properties and methods were implemented.

We already know that the window manager is the template view model and not the template itself. So all the logic behind the template is located at the template ViewModel. The template itself is only the entry point to the application where we assign responsibilities. For this reason, the initialize method of the template is just referred to as the initialize method in the ViewModel. This method is responsible for attaching event handlers for the plugins events, changing the language to the default language set at the management application and setting the current page.

As mentioned in the process chapter, the interface of this template should have two distinct moments. One is an attraction mode where random data are displayed in order to catch the attention of the airport passengers and the other would be an interaction mode where the user is interacting with the system. To develop a paging system with WPF we must create two classes, one for each page, make them implement a common interface and create a property that returns an instance of that interface in the class that links to the current view where the pages are about to be displayed. To change page we set the property, called CurrentPage, to be an instance of the concrete class that represents that page. In addition, we need a DataTemplate key for each page we want to display.

In the ITemplateViewModel implementation we have added a property called CurrentPage that returns an object of type IPluginPage and two other properties, one for each page that requires the returning of an object of type IPluginPage, but where each one has a different class.

```
public IPluginPage InactivePage { get; set; }

public IPluginPage ActivePage { get; set; }

public IPluginPage CurrentPage {
    get
    {
        return _currentPage;
    }
    set
    {
        if (_currentPage != value)
        {
            _currentPage = value;
            RaisePropertyChanged("CurrentPage");
        }
    }
}
```

Figure 34 Multi page implementation

So, to change from one page to another, all we need to do is set the CurrentPage to the page we want and raise the property changed event to tell the views to evaluate the new value of that property. As we have already seen, we need a DataTemplate for each page we want to display. As these views are known at compile time, we can just create it in XAML.

```
<UserControl.Resources>
    <ResourceDictionary>
        <DataTemplate DataType="{x:Type vms:ActiveViewModel}">
            <views:ActiveView></views:ActiveView>
        </DataTemplate>
        <DataTemplate DataType="{x:Type vms:InactiveViewModel}">
            <views:InactiveView></views:InactiveView>
        </DataTemplate>
          …
</UserControl.Resources>
```

Figure 35 Create datatemplete key in XAML

After the DataTemplate is added to the view, we create the element that we will set the Content property bound to the CurrentPage property in the ViewModel. Setting the Content property is equivalent to setting the DataContext.

Now this element will display the view inside the element according to the concrete type returned by the CurrentPage property.

```
<mah:TransitioningContentControl Grid.Row="0" Grid.Column="0" Grid.RowSpan="2"
Content="{Binding CurrentPage}"></mah:TransitioningContentControl>
```

Figure 36 - Setting element content in XAML bound to view model

The same strategy was used to set the current plugin to be displayed. There is a property called CurrentPlugin that is bound to the content property of an element at the ActiveView.xaml. For the plugins, however, the DataTemplate keys had to be inserted in the code-behind file because it is not known, at compile time, what concrete types are needed to create the DataTemplate.

```
        private void ActiveView_Loaded(object sender, DependencyPropertyChangedEventArgs
e)
        {
            var dc = DataContext as ActiveViewModel;
            if (dc == null || dc.App.CurrentPage != dc)
                return;

            foreach (var plugin in dc.App.Plugins)
            {
                Type view = plugin.AppView;
                SindigVisualHelper.CreateDataTemplateKey(plugin.AppViewModel.GetType(),
view, this);
            }
        }
```

Figure 37 Dynamic datatemplate key creation

This strategy is very close to the state design pattern if we identify each page as a state of the system and those states can access the object that created it in order to change the state or execute other methods.

To build a multi-language system, we have implemented a Command that is bound in the template view. This Command executes a method responsible for changing the language of the system. As far as the template is concerned, it needs to change the language of the whole system and tell the plugins that the current language has changed. The plugins need to perform their own actions to change to the chosen language.

All the services that are language dependant should update their interfaces as well. For all the plugins we just call the set language method indicating that the language has changed but do not load unnecessary data immediately. This strategy avoids loading the data from all the plugins when the only plugin we need to load data is the current plugin, if set.

This template includes an area where persistent data is displayed. When the monitor has installed the flight data plugin, then an animation should be shown telling the users that if they scan the boarding pass, the flight data will be available; if the flight plugin is not

installed, a simple welcome message is displayed. This area is also available for the plugins to set persistent and important information that should be available at all times. In the case of this project, only the flight plugin sends this information containing the user's flight data and is updated with the ANA airports resource broker (the same data provider of the fids system).

This feature was developed using the same DataTemplate strategy mentioned before. The plugin sends the view type and the object that will be its DataContext. The template takes care of setting the property bound to the animation element of the object sent by the plugin. If the plugin itself (instance of the class that implements IPlugin) is sent, then the template changes the current plugin to the one sent by the plugin.

```
async void plugin_OnSmallReceived(object sender, SmallEventArgs e)
{
    if (e.SmallViewModel != null)
        CurrentSmallViewModel = e.SmallViewModel;
    if (e.Plugin != null)
    {
        await ChangePlugin(e.Plugin).ConfigureAwait(false);
        await ChangePage(ActivePage);
    }
}

public IPluginViewModel CurrentSmallViewModel
{
    get { return _currentSmallViewModel; }
    set
    {
        if (_currentSmallViewModel != value)
        {
            _currentSmallViewModel = value;
            RaisePropertyChanged("CurrentSmallViewModel");
        }
    }
}
```

Figure 38 Persistent information event handling and property

There are three more events that the template should be aware of that are raised by the plugins. There is the logout event that a plugin can raise to tell the template that a logout was requested. There is also a toast event where the plugins can send any string data to be displayed as a toast message by the template (this feature is not in use but is ready if future design requires it).

There is one more event that is very important, the ChangeToPlugin event. This event is raised by a plugin when it wants the template to change to another plugin with some

data formatted in such a way that the plugin to change to can understand and take the necessary action.

The last feature we will cover regarding this template concerns the logging system. This template is responsible for logging every action a user takes (the plugins can log data as well). For every major user action a new row is inserted in the log table in the database.

#### 4.4.3.8 The IPlugin interface

A new plugin should create a class that implements this interface. The constructor must have one argument of the type List<int>. This list contains the data that the plugin has associated with the current template and has six values (five set in the management application and the last is passed by the host application when instantiating and is the Id property), that should be returned by the properties DataA, DataB, DataC, DataD, DataZ and Id respectively. The template will access these properties in order to make its calculations regarding positioning or other operations.

Just like the ITemplate, this interface must have one property to return the Type of the plugin view and another returning an IPluginViewModel. The process is the same as that of the template, a data template is created relating the type of view and the concrete type that implements the IPluginViewModel and, when the plugin is selected, the data context of the plugin area is set to the value returned by the IPluginViewModel property. As a result, the plugin area is a responsibility of the current plugin.

This interface has three events that the plugin can raise. These events are a logout command to be sent to the template, a Toast message that the template will display as a notification and another one to tell the template to change to a different plugin and optionally initialize with parameters.

The tile colour is just a hex value of the background of the plugin miniature (tile). The Version is the full name of the project including the extension (.DLL) and the Icon is a string that will be searched in the resources that represents the icon to display in the tile when in miniature mode.

The Name property is the name of the plugin. Owing to the multi-language nature of the system, it is imperative that plugins that have different names in different languages to implement the INotifyPropertyChanged interface or reference our *Sindig.MVVM* project and make it a subclass of ObservableObject in order to update the name on the user interface when a language change command is executed and the name changes by raising a property changed event.

There are also a few optional methods to implement. The SetLanguage for example is made to change the language and receives a language abbreviation. If the plugin is implemented in one language only, this method can be left blank.

#### 4.4.3.9 Plugin implementation

We will demonstrate how to develop a fully functional plugin from start to finish. As an example we will develop the wi-fi plugin, as it is the simplest plugin developed for this project.

We start off by creating a new class library project in Visual Studio and give the project a name. It is good practice to keep the namespacing convention for plugins. Following the naming convention we will call this project *Sindig.Plugin.Wifi.*

The next step to develop this plugin is to add the necessary references. The only required reference to be added is to the *Sindig.Core* project where the interface we are about to implement resides. Although the only required reference is already added, we will reference the *Sindig.MVVM* project because we will develop this plugin following the MVVM pattern and this project contains classes (asynchronous and synchronous command implementations and INotifyPropertyChanged implementations for databinding purposes) to implement this pattern.

Now we are ready to start developing the plugin logic. Create a new class and name it wifi.cs at the root of the project and make it extend ObservableObject (INotifyPropertyChanged implementation), implement the IPlugin interface and implement another interface we have not covered yet called *ITile*. The *ITile* interface provides properties to develop the plugin as a tile. We are building this plugin as a

single small square tile that displays a wi-fi symbol while the monitor is in the attraction mode.

At this point we have assembled everything we need to start developing our plugin. The wifi.cs file should look like Appendix B.

The code is very simple and straightforward. All we are doing is setting things up for the plugin to work correctly as it is only the entry point to our application, providing properties the template can use in order to display the plugin information.

As we have seen, the template displays the plugin tiles when it is in the attraction mode but when a plugin is selected the template allocates an area on the screen dedicated to the selected plugin. This is achieved by setting the data context of that area as the plugin AppViewModel property and as a result the interface is the AppView (because of the DataTemplate created by the template) and the plugin logic is contained inside the AppViewModel which controls the plugin behaviour.

It is now time to develop the plugin logic inside the AppViewModel. As this plugin contains only static information there is nothing to implement inside that class except the IPluginViewModel interface LoadDataAsync method.

```
public class WifiAppViewModel : ObservableObject, IPluginViewModel
    {
        public async Task LoadDataAsync()
        {

        }
    }
```

Figure 39 Wi-fi app view model implementation

All that remains for this plugin are the user interfaces. These are built as XAML user controls and do not need to implement any interface because the only information the template needs is the Type (the full namespace of the user control class and the name of the class, “Sindig.Plugin.Wifi.Views.WifiAppView” is the wi-fi AppView Type returned by the AppView property). As an example, the wi-fi small tile can be seen at Appendix A.

This is just standard XAML markup, but there is one trick. To make this plugin multilingual we cannot hard code any text inside the view; instead we need to bind the text of the TextBlock to a string as a DynamicResource (a resource that can change in time). To do this we add all the text we need to localise to an XAML resource file (one per language).

What happens is, the template detects that the user is trying to change the language so it notifies all plugins to change their current language. The plugin then loads the XAML file for that language and inserts it into the application resource. We need to be careful to give the same name for the text in every language file.

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:system="clr-namespace:System;assembly=mscorlib">
  <system:String x:Key="Wifi-TileText">Descobre como aceder à internet sem fios no seu dispositivo preferido.</system:String>
……
  <system:String x:Key="Wifi-Instructions">grátis e ilimitada</system:String>
</ResourceDictionary>
```

Figure 40 Wi-fi Portuguese resource file

This is an example of the Portuguese file so the English file would be exactly the same except for the string values inside the tags. This way we are able to bind from the view to a key of the resource file and when the current file is changed, the value of the TextBlock is changed.

Now the created plugin is ready for production and with support for multiple languages.

#### 4.4.3.10 The Flight Plugin

The flight plugin was the most challenging plugin developed for this system. It is composed of many pieces that work together in order to display the correct flight information to the passengers at the terminal.

The plugin was built following the same principles as the example above. The difference is its complexity as it makes use of dependencies in order to work correctly.

We required a way to read the passenger boarding card in order to provide customised information about the flight. To achieve this functionality we have used the existent bar code reader (the same used at the check-in counters by the handlers) and developed a library to read and parse that data. The data is formatted following the IATA standards in Appendix D. This library is one of the dependencies we need to assign to this plugin in the management application.

Figure 41 Barcode scanner

The barcode scanner had to be configured to perform in the intended way. We wanted the scanner to be in idle mode until it identified an object to be scanned. To configure the scanner we used the *MetroSet2* software as shown in Appendix L. We chose the scanner to operate in *Presentation Mode* and to add a suffix with an *ETX* character allowing us to identify where the boarding pass information stops. Also, a short range was selected in order to prevent false activation of the scanner.

In the web application we developed a soap server to receive calls from the ANA resource broker. They perform two distinct calls to our server. The first call is a CheckConn call to make sure our application is up. The second call is ResourceDataAvailable that tells us that the resource broker has new data for us. When we receive that call we use our soap client to make a call to the resource broker method called GetDataExtended to fetch the new data. When we receive the data we store it in the database to allow the kiosk to display a list of the flights (arrivals and departures).

We also push this data to all the clients (kiosks with the plugin installed), through our web socket server, so they can update the flights that have new information including, if selected, the user flight.

At the kiosk we display two lists of flights, arrivals and departures, allowing any user to see detailed information about any flight with live feed. These lists are built with database information and updated in real time through the web socket.

When the plugin starts we create a web socket connection to the web socket server. We create only one connection for the plugin and cycle the flights to update their information.

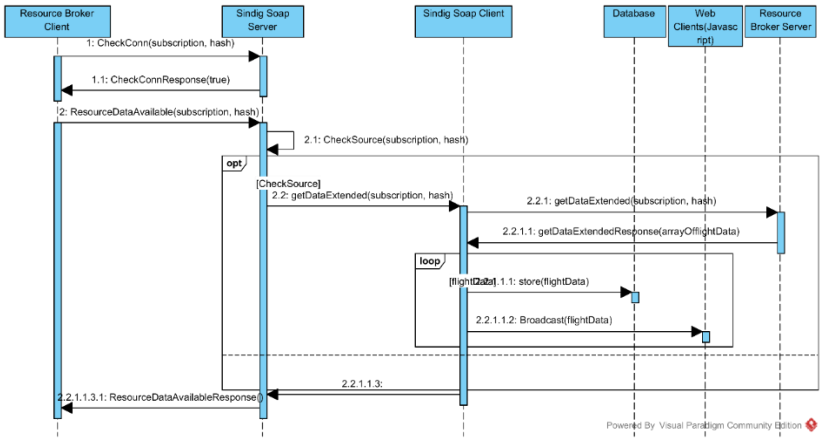



Figure 42 Flight plugin sequence diagram

When a boarding card is read, we first check if the data is valid according to the IATA specifications, then query the database for the latest entry about that flight. We then set the current flight as the created one and change the page to the flight page displaying all the flight information we have (we do not store or display any personal information like name or seat number for security reasons) and the current flight is updated when new data is received through the web socket.


Figure 43 Flight plugin departures list

The web socket receives data in the JSON format that needs to be parsed before it can be used. To that effect we have used another dependency called JSON.NET which is a library built to make it easy to work with JSON objects.

This plugin also makes use of the small space reserved by the template to display persistent information. When a user selects a flight or scans the boarding pass we send to the template the user flight in order for it to be displayed all the time while the user is interacting with the system. This information is also updated with live feed data which can guarantee the passenger that he is not missing his flight.

As a result, we have developed an interactive flight information panel and created a way in which the user can interact with the system but stay informed about his flight, lowering the stress level (if any) while using the system.

#### 4.4.3.11 The Weather Plugin

Together with the flight plugin, the weather plugin was identified at an early stage as useful to be included in the system, leading us to its development.

To any given passenger it could be useful to know the weather and the forecast, at the destination. In addition, from a curiosity perspective, a worldwide weather forecast could be provided to the passengers at the terminal.

To begin with, we needed to subscribe to a weather service in order to obtain live and up-to-date weather information from around the world. Also, it had to be free and provide a worldwide weather forecast. Accordingly, we have subscribed to the service found at http://www.worldweatheronline.com/ to obtain the necessary data.

There is, however, a limitation on this service regarding the number of requests per day. However, as the monitors do not have internet access the request to the weather service should be done through our internet proxy module, thereby reducing the number of requests.

Besides avoiding too many requests to the weather service, this strategy gave us the ability to cache the weather data of each of the locations accessed to reduce even more the number of requests per day and the network bandwidth usage, thereby increasing the overall plugin response speed. In addition to caching the weather data on the server, we cache data on the monitors, eliminating another unnecessary request.

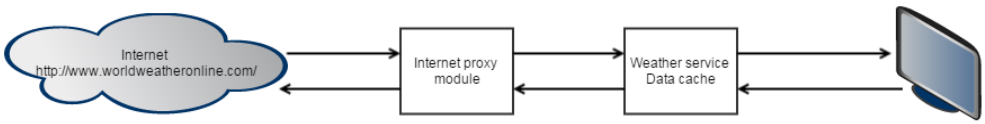



Figure 44 Weather data access architecture

Due to the nature of this system, we have designed the weather cities to correspond to the airport codes. As a result, the provided weather forecast corresponds to the weather at the airport of the selected city.

As regards implementation, we have developed a weather service that takes care of all the weather requests and data caching for the plugin itself. In addition it allows us to use the service from another plugin, for example the flight plugin, to display the weather at the destination.

Finally, the weather plugin itself gathers the data from the weather service and displays it according to the designed user interface and interactions following the same MVVM pattern as the other plugins.

### 4.4.3.12 The News Plugin

The strategy followed to develop this plugin is an exact match of the method used to implement the weather plugin. The only difference is the data source; in this case the data source used to implement the news service are the RSS feeds provided by the news companies.

The subscribed weather service provides the data in the JSON format. By contrast, the proxy module for the news plugin receives the data formatted in XML according to the RSS specification. Therefore, the RSS is parsed and cached in JSON format at the time it is gathered and sent back to the news service, avoiding extra parsing apart from that of JSON.

With regard to the interfaces of the plugin, a couple of WPF techniques were used to develop the desired result. WPF has built-in panels to display items but none provided the functionality we needed. Furthermore, none could be found on the internet that were ready to use.

The behaviour of the interface was as follows: for one channel, the news should be arranged horizontally with fixed width but variable height. The news on the second row and beyond should fit close to the news on the previous row in the same column.

In other words, the rows created and inserted should not be dependant on the height of the the previous row.

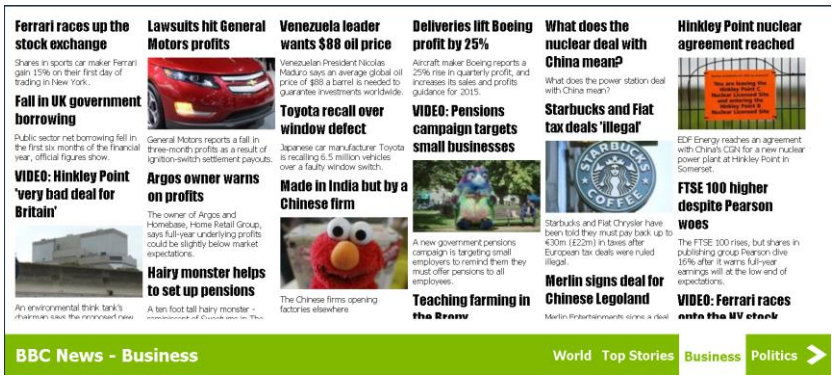



Figure 45 News plugin result interface

Developing a WPF panel from scratch requires comprehensive knowledge of the technology. Moreover, it is a complex task and takes quite some time to understand and use correctly. Put differently, as we did not have the time to go over and learn how to create panels from scratch, we decided instead to use the knowledge already acquired.

As a result, we approached the problem from a different perspective. We achieved the desired result by using the existent panels in WPF. The idea is very basic. Start by setting a width for the news item. Then, from the total width for displaying the channel we check if it fits by dividing it by the width of the news and summing the necessary margins between each news item ending up with the number of columns. If it does not fit, we make the width smaller by one and try again recursively. For each column we add a WrapPanel oriented vertically and go through all the news from the channel and add one at each WrapPanel at a time and back to the first.

In addition, the news property of the view was created as a dependency property in order to update the view only when it changes, it is not empty and the selected plugin is the news plugin. That is, when we update the News property to contain the news of the selected channel we need to update the view to display the correct data.

The problem is that the plugin Selected property is only set after the News property is set and the code does not get executed. So we had to create a dependency property in the view called Selected and bind the value of the plugin Selected property and execute the same code as that which is executed when the News property changed.

All in all, the development of this plugin was quite challenging due to the complexity of the interface. The end result is successful and does exactly what we needed. The same technique was used in the gallery plugin but this time we inverted the situation, creating a dynamic Grid with a variable number of rows and added one WrapPanel oriented horizontally in each row and added the media to each WrapPanel until it finished.

#### 4.4.3.13 Other considerations

All the methods that have the signature:

Task DoSomething ();

Must be implemented like this:

async Task DoSomething ();

The reason for this is that an interface cannot contain the async keyword, only the return type.

Inside these methods, it is expected that at least one method call is executed with the await operator. This operator will make the method execute asynchronously in a multi-threaded fashion, which is better for user feedback, not blocking the UI while loading data for example [16], [17].

The use of async await operators is expected all the way down (or up) so that no method blocks the UI during a data loading or any other time-consuming operation. Do not forget to call the method ConfigureAwait (false) to every async method unless you really need to store the context before that call (only in rare situations is it not well suited).

## 4.5 Perspective based UI inspection

During one of the airport stakeholder evaluations we had conducted a perspective based UI inspection [18], [19]. The inspectors (the stakeholders in this case) were given tasks to perform on the system, given a well-defined perspective.

The given perspective was: you are a passenger that has already checked-in and passed the security control. Now you are in the departure lounge waiting for your gate to open and you find this (our prototype) system.

The gender and age of the prospective passenger was the same as the evaluator since this allowed for a good variety of personal data. Playing the role of a facilitator was the project manager, who conducted the whole process.

Moreover, it occurred in an evaluation meeting where most of the plugins had already been completed. Those inspections were documented by means of video footage in order to allow a later examination by the team.

One of the inspectors suggested a new arrangement for the tiles on the attraction state regarding the size and colour of the gallery plugin. The suggestion consisted of increasing the size of the tiles in order to fill the gaps not filled in the interface grid. The suggestion was accepted and as a result the attraction state looks more complete.

Another inspector suggested that when a user touches an advertisement it should display some information about the advertised product. This suggestion represents the development of a new plugin that will be revisited in the near future.

The results from this evaluation served to validate the findings from Chapter 3. It did help us to discover issues regarding small pieces of the interface such as colours and tuning the layout arrangement and alignment. Also, the stakeholders were made to feel that they belonged to the project.

# 5 PROTOTYPE

In this chapter we will focus on the end user perspective of the most interesting features of implementation and design regarding the finished prototype. We will describe the most important pieces of the interfaces and correlate them to our findings during the development process.

## 5.1 The template

The developed template has two modes, the attraction mode and the interaction mode. With regard to the attraction mode, we did not spend much time on it as it is just the entry to our application and its layout was initially well defined. On the other hand, the interaction mode was not clear from the start, resulting in a couple of interactions needed to define its layout.


Figure 46 Template attraction mode

The attraction mode follows the look and feel used nowadays on touch devices. We tried to follow these lines but with our own implementation.


Figure 47 Template interaction mode guide

Figure 47 shows how we mapped the findings from the process with the end result. As suggested by our findings, we placed the plugins at the bottom of the screen to facilitate navigation and prevent the user's arm from covering the current application area. The advertisement to the right resulted in a left oriented position from our user, representing no interference with usage. The application area occupies the largest area of the template and is the responsibility of the selected plugin. Persistent information, or animation, stays below the advertisement and is always visible.

## 5.2 Flight

This is the main plugin developed for this project, thus it was target of many interface proposals and changes.

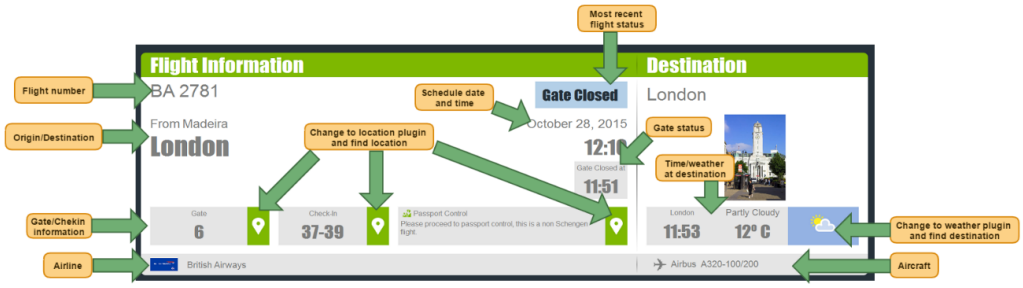



Figure 48 Flight data guide

Figure 48 shows every piece of information the user can obtain by selecting his flight or reading his boarding pass. Additionally, it shows every interaction the user can perform with this information such as finding the gate or check-in counter.

Apart from the flight data interface, we have developed the layout for the small area of the template with persistent information.

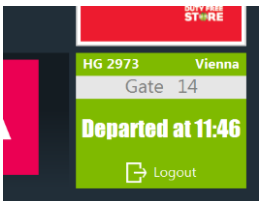



Figure 49 Persistent flight information

The space left in this template area is quite small and cannot contain too much information. As a consequence, we have reduced the amount of data it should display leaving only the minimum required by our users in order to keep them up to date with the latest changes in their flight.

Lastly, the flight lists for arrivals and departures, were also re-designed several times. In the end, we decided to follow the layout used by the airport terminal flight information panels to maintain consistency at the airport.


Figure 50 Flights arrivals/departures pages

The barcode scanner was installed at the bottom right corner of the kiosk equipment. The kiosk structure constrained the positioning of the scanner. The kiosk did not allow

us to install the scanner horizontally due to its internal structure. However the result was very satisfactory.

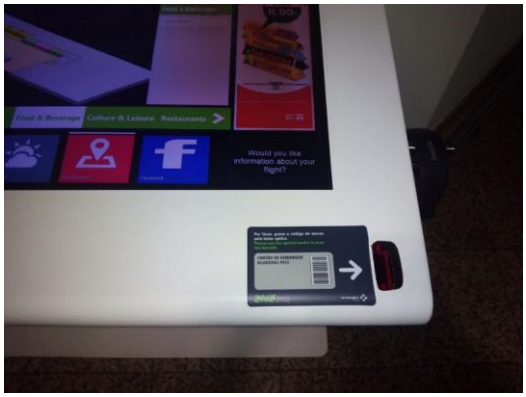


Figure 51 Barcode scanner installed

In addition, a sticker was placed on the side of the scanner to indicate to the passengers where to scan their boarding pass.

## 5.3 Gallery

In the development chapter we discussed how the interfaces were developed; at this stage we are only interested in the end result.

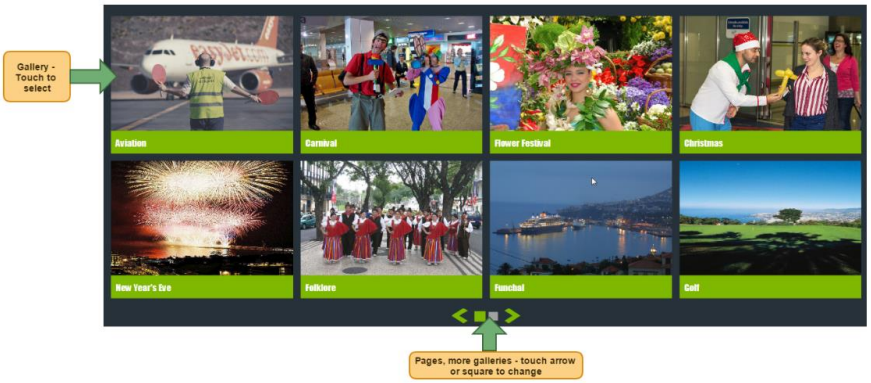



Figure 52 Galleries page guide

This plugin works as a flow of actions. First, the user selects the gallery from which he would like to see the media.

Then, the media thumbnails of the selected gallery are presented and the user can touch to see the full size picture. A toolbar is displayed to make it possible to navigate back to the galleries page.

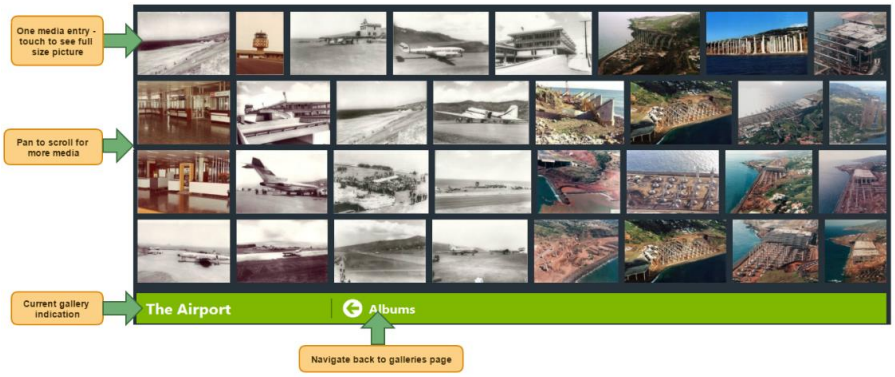



Figure 53 Gallery thumbnails guide

Finally, the selected image is shown in full size and the toolbar displays actions regarding the current media selected.


Figure 54 Display media guide

Also, small thumbnails are displayed to allow the user to change the current selected media. The galleries support media in image or video format.

## 5.4 News

It was very challenging to figure out how to create a good interface for this plugin. Thus, we created many proposals to this end. In the weekly meetings we have discussed at length all proposals from the designer to find the appropriate solution.

We followed what news web sites are doing by arranging the news entries vertically in columns.

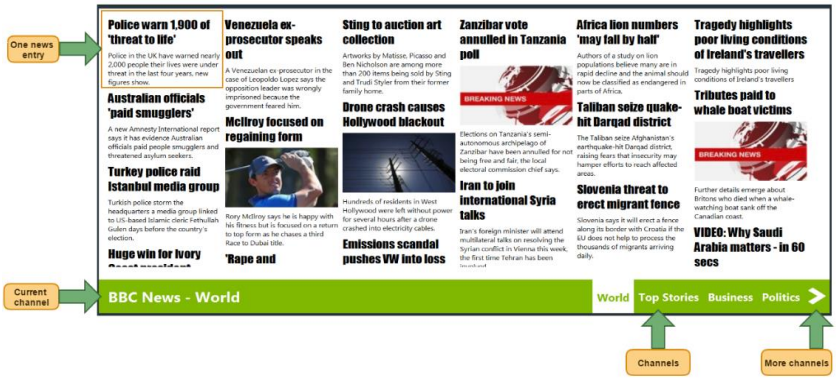



Figure 55 News interface guide

The bottom bar follows the same guidelines as the other plugins. This toolbar allows the user to perform actions such as changing the current news channel and browsing for more channels as they are arranged in pages.

## 5.5 Weather

Like all other plugins we had a couple of design proposals for this plugin. The core idea behind these interfaces was kept as the end result; we have just adjusted the way the weather information is displayed in the main area.


Figure 56 Weather plugin guide

## 5.6 Location

Although this plugin was not identified as an important feature in the system, we did not disregard its interfaces and interaction.


Figure 57 Location plugin guide

This plugin allows the user to navigate through the terminal floors, select a category such as restaurants or shopping and select the location from within the selected category from a list. Additionally, it is possible to locate the user position in the terminal.

The map categorises certain areas of the terminal which are included in the legend to help users identify what each colour means. Also, the interface allocates an area where information about the selected location is displayed.

# 6 EVALUATION

In this chapter we will focus our attention on the evaluation methods we have followed. We will talk about the inquiries made to real users of the system and present usage charts related to a week.

## 6.1 Usage charts

The developed prototype is equipped with a logging mechanism that records on a database table every user interaction with the kiosk software. With this data, we are able to extract useful information about the usage of the system, in order to present findings, conclusions and recommendations for future development and decision-making.

The logging mechanism has one problem. If a user leaves the kiosk without explicitly logging out of his session, we need to terminate the session automatically. As explained earlier, we use a timer to count the inactivity time of the kiosk to log the user out after a few minutes. The problem occurs when a user leaves an object on top of the screen, resulting in a never-ending activity and the session remains active continuously. Consequently, we had to improve our session management by storing a Timestamp of the last activity regarding our commands and logout if this time was greater than five minutes.

Furthermore, we are ignoring sessions with time lower than two seconds. The reason is that those sessions represent a single interaction and not real usage of the system as no further interactions are detected.

The data we used for this analysis corresponds to the week of 24 - 31 October 2015.

### 6.1.1 Sessions

A session corresponds to a user using the system. Different user means different session. During this week we were able to identify 290 distinct sessions. We will now analyse what happened this week during the sessions, taking into account the day of the week and the time range (morning, night, afternoon and dawn).

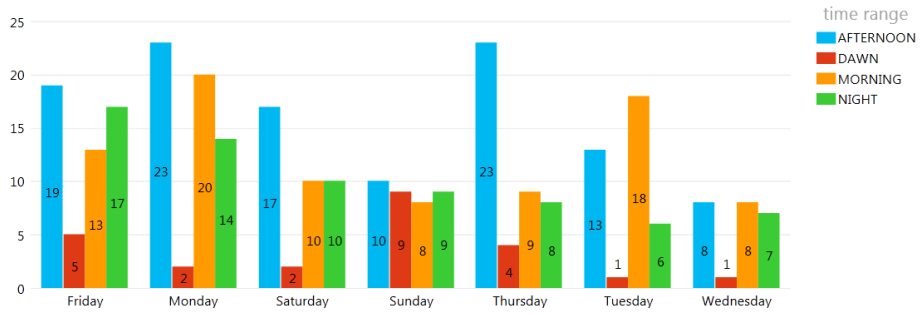



Chart 1 Distinct sessions by day of week and time range

Chart 1 is composed of three axes: the day of the week, the time range and the total distinct sessions. It represents an overall view of the sessions during this week but it can be difficult to see the day of the week or the time range with the most sessions. We shall split the above chart into two to have a better view.

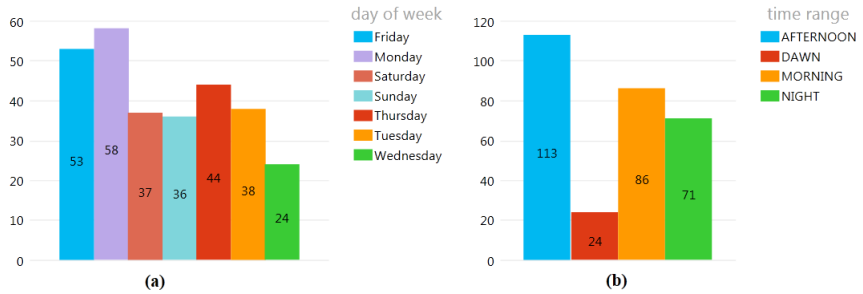



Chart 2 (a) Distinct sessions by day of week and (b) Time range

From Chart 2 we can conclude that, regarding the day of the week, Mondays and Fridays have the most usage. Monday is not a surprise since Monday is the busiest day at the airport. From (b) we can see that during the afternoon is when most users interact with the system.

The presented charts take into account only the identified sessions but do not tell us for how long a user interacted with the system. The overall average session time during this week was 4:45 minutes (four minutes and forty five seconds), with the maximum being 27:37 minutes (twenty seven minutes and thirty seven seconds).

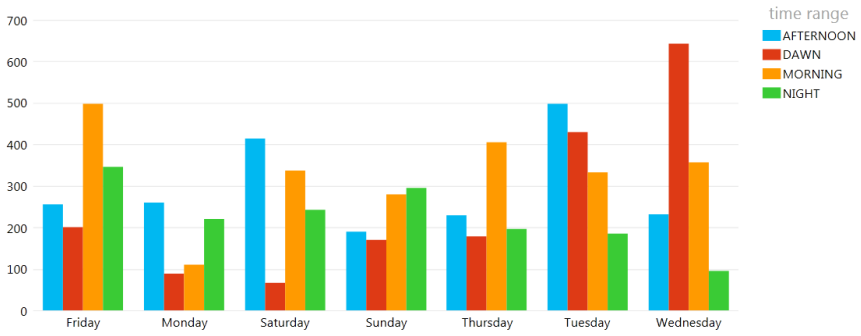



Chart 3 Average session time by day of week and time range

If we go back to Chart 1 Distinct sessions by day of week and time range, we can see that Tuesday and Wednesday at dawn had only one session each. As a result, on the chart above on Tuesday and Wednesday at dawn the average only considers those single sessions resulting in a chart that maybe does not correspond to reality.

We have broken Chart 3 into two results in the following charts.

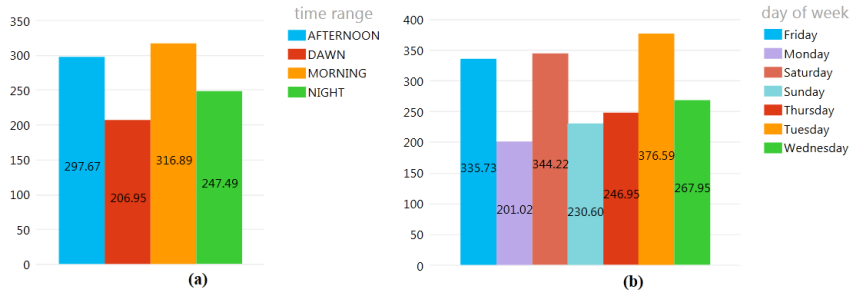



Chart 4 (a) Average session time by time range and (b) Day of week

Chart 4 looks better as **(a)** shows that the dawn average session time is much lower than the other time ranges and that mornings and afternoons are the time ranges where the users spent more time during the week.

From **(b)** we can draw a few more conclusions. The Wednesday average session time has dropped but not that of Tuesday. Tuesday, Saturday and Friday were the days of the week with higher session times. An interesting point is that on Monday, despite its being the day of the week with the most identified sessions, the average session time is very low. This observation might be in line with our initial thoughts about this project. As a public facility, on busy days competition to interact with the system might occur, resulting in lower session times.

### 6.1.2 Plugins

The logging system also stored information about the plugin usage during this week. We were able to create charts representing that data. During this week we collected 1641 plugin changes (each represents a user selecting a plugin).

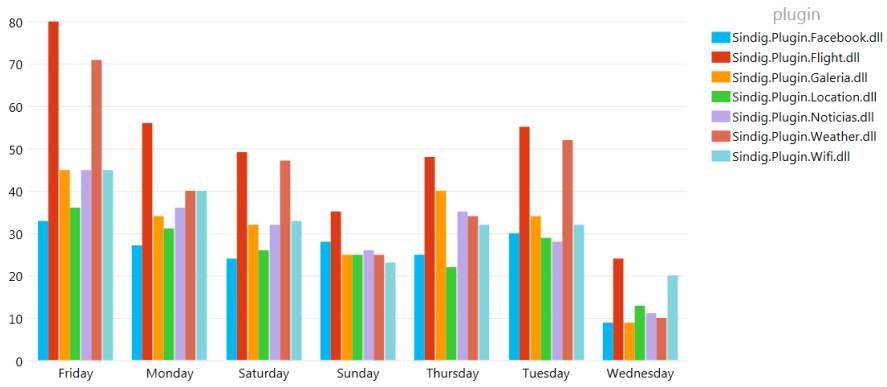



Chart 5 Plugin usage by day of week

No matter what day of the week, the plugin most navigated to is the flight plugin. It tells us that this plugin is the most attractive for our users.

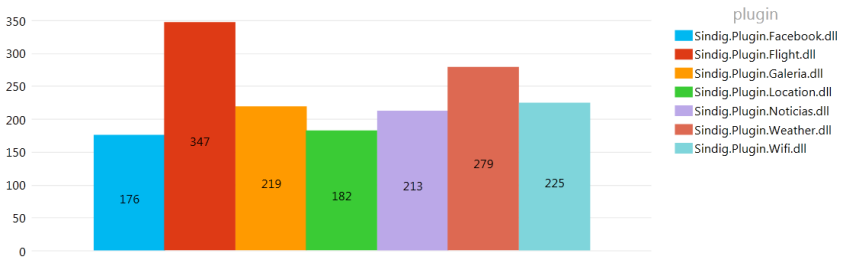



Chart 6 Plugin usage

The chart above ranks our plugins by the times they were selected by the users. There are no surprises from the previous chart. The flight plugin comes in first place with the weather in second.

Validating our initial thoughts about the location plugin, it is one of the least used plugins just ahead of the Facebook plugin. The poor result for the Facebook plugin might be related to the fact that it is the airport Facebook page being displayed, or else people are afraid that the plugin might use their Facebook account (which is not the case).

We will now summarise what happened with the plugin usage regarding the time range.

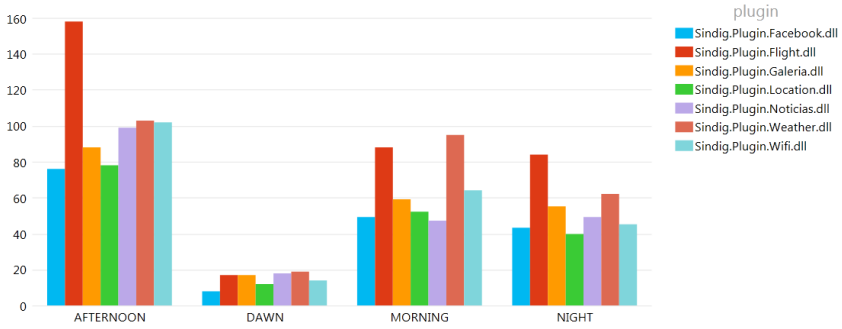



Chart 7 Plugin usage by time range

There are no surprises on the chart above. Relating this chart to the sessions by time range, we know that afternoons have more sessions so it is expected that more plugin changes occur during that period.

On the next charts we will analyse the data regarding average time of usage from each plugin. The average usage time of the plugins is one minute and thirty eight seconds.

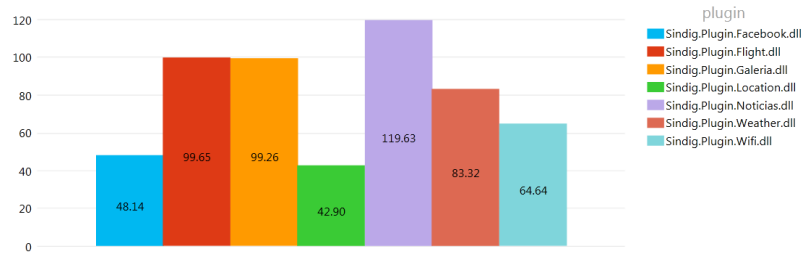



Chart 8 Average plugin time

The chart above presents us with some very interesting results. As discovered from previous charts, the Flight and Weather plugins are the most used plugins. However, on this chart we notice that they are not where the users spend most of their time. As surprising as it seems, the News plugin has an average of nearly two minutes compared to one and a half minutes from the flight and gallery plugins. Additionally, the gallery plugin has an advantage over the others, standing out with nearly the same time of usage as the flight plugin.

By contrast, the location and Facebook plugins have really low usage time on average. Adding the fact that they are not commonly used we can say that these services are not valued by our users.

The next graph shows the maximum time spent by users on each plugin. We notice that it follows the statistics from the previous chart. It is noteworthy that, despite not having the highest average, the flight plugin had the maximum plugin time, followed by the news plugin.

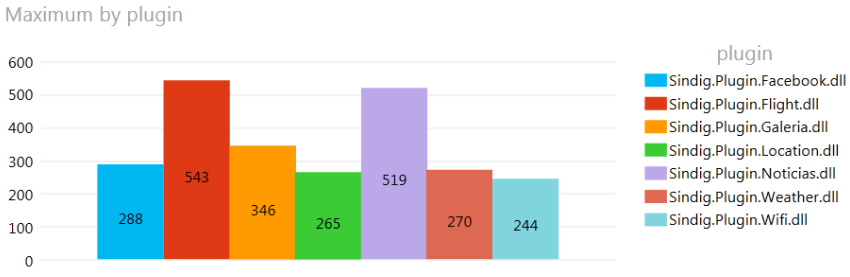



Chart 9 Maximum plugin time

In summary, when a user goes to the news or gallery plugin, it is likely that he will spend some time on these services. Moreover, it consolidates our findings that these services would be useful to our passengers if included in the platform, as they were.

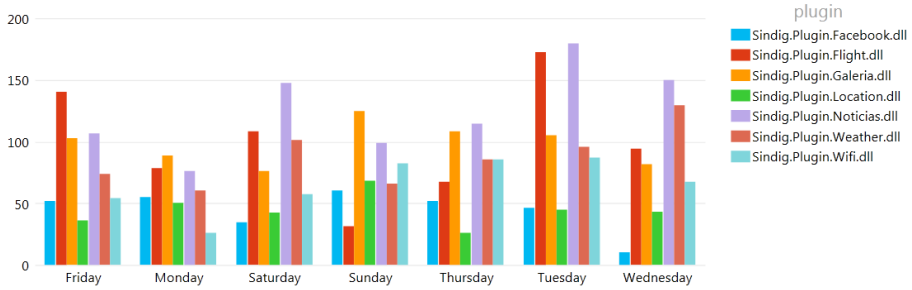



Chart 10 Plugin usage time by day of week

The chart above gives us a good overview of how the news and the gallery plugin usage times progressed through the week. The flight plugin is still a valuable service. Nonetheless, the gallery and news plugin bars stand out in the chart as having increasing usage throughout the week.

Likewise, we can compare the average usage time of the plugins regarding the time range.

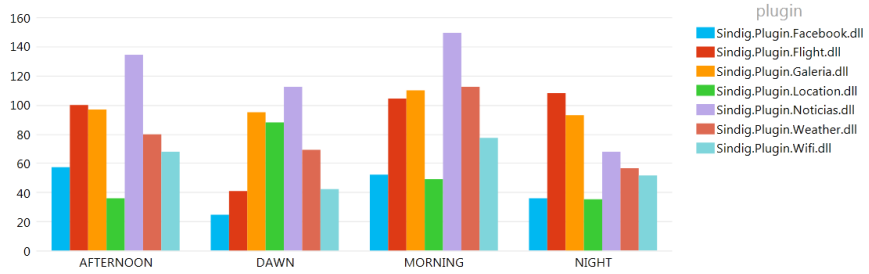



Chart 11 Plugin usage time by time range

With respect to time range we can identify the morning and afternoon as being the periods of the day when the users spend their time on the platform. Yet, we can safely say that during the night period, users do not stay for long periods on the platform. Despite having identified only 24 distinct sessions during the dawn period, we see that these users spent a significant amount of time interacting with the system. Correspondingly, some plugins have more usage time than other time periods. It is

interesting that users do not spend much time on the flight plugin during the dawn period.

### 6.1.3 Boarding pass scan

The system is able to read boarding cards in order to show the user information about their flight. The usage of this feature was also a target of analysis, having registered 261 successfully scanned boarding passes in the period from 1 October to 28 October (four weeks exactly). The chart below shows the usage of this feature taking into account the time range and the day of the week.

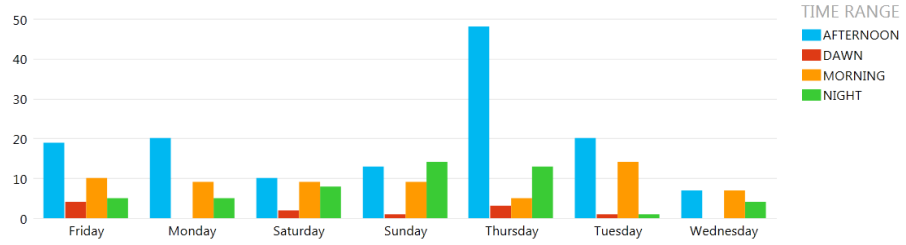



Chart 12 Boarding pass scan by time range and day of week

From the chart above the Thursday afternoon bar really stands out while the other bars follow an even distribution.

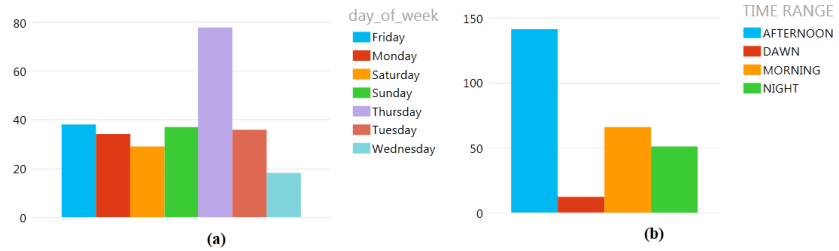



Chart 13 (a) Scan by day of week and (b) Scan by time range

The two charts above confirm that Thursday afternoons are the periods where users most often scan their boarding passes. This might be due to a weekly flight at that period where the users are not afraid to test this functionality. The decision to try scanning the boarding pass might be related to country of origin, age or another factor.

We will try to find some relationship later in this chapter when we analyse the flights that occurred during a week.

The distribution throughout the other days of the week are quite similar. The same happens in the time range analysis where Thursday stands out. By comparison, the distribution is not even during the dawn period. As this period has few distinct sessions it is reasonable to say that the more users we have, more boarding passes could be scanned.

### 6.1.4 Flight data

In order to understand better the scanned boarding passes we created the following charts to try to find some relation between the results from Thursday. We used the same period as the plugin and sessions analysis.

There were 182 flights during that week summing up to a total of 28210 available seats. We will assume that the flights were full when they departed.

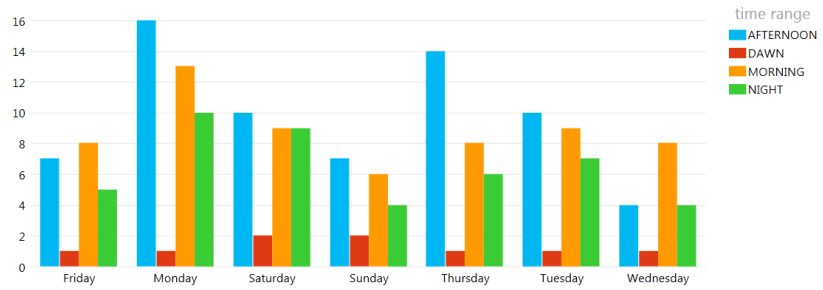



Chart 14 Flights by day of week and time range

The chart below shows the available seats on the aeroplanes that operated during the time range for this analysis. We can see the same distribution from both charts, as expected.

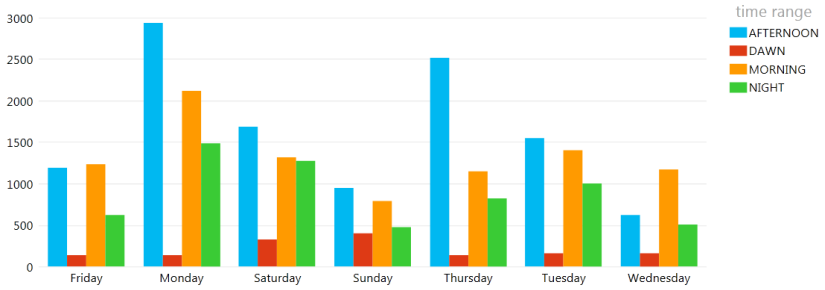



Chart 15 Available seats by day of week and time range

Regarding the result from Thursday afternoon, we have analysed more data to draw a conclusion.

By analysing the flight data, we noticed that during that week, there were two flights with destinations that did not occur on any other day of the week. Those destinations are Vienna and Vilnius. However, the extracted flight data from that week did not register any boarding pass scan on Thursday afternoon (28 October).

In order to understand what happened we analysed the data from all Thursdays in that range. On Thursday afternoons there were 48 boarding pass scans. On 1 October in the afternoon there were two scans, on 8 October 17 scans, on 15 October 23 scans and on 22 October 6 scans. Most of the scans happened between 1500 and 1700 validating the fact that the two flights mentioned previously had no influence on the result as they departed at 1400 and 1500.

The 48 boarding pass scans came from 28 different sessions. Two sessions scanned five passes each, four sessions scanned three passes each, another four sessions scanned two passes and 18 scanned one boarding pass making the total of 48.

Moreover, if we consider only one boarding pass scan per session, we end up with 28 scans. This consideration relates to the fact that we noticed that some of the scans happened in a short period of time (one or two minutes difference).

We can conclude that the data from Thursday afternoon was influenced by the scans performed on 15 October. During that afternoon, 23 boarding passes were scanned from 15 different sessions (15 scans considering one scan per session).

We cannot draw any conclusion from what happened during that Thursday afternoon. Maybe social influence among the passengers led to a lack of trust in the passengers which prevented them from scanning their boarding passes. Anyway, we will treat this result as an outlier.

## 6.2 System usability scale

In order to gain a better understanding of the system usability we have gathered information from some users after they interacted with the system with an SUS (system usability scale) questionnaire. SUS is a reliable tool for measuring usability by asking participants to answer ten questions with five response options from strongly agree to strongly disagree (Appendix M. )[20].

We asked nine users to answer the SUS questionnaire regarding the developed system.

As a result, we were able to measure the usability of the system in a 0 to 100 scale.

| Participant | q1 | q2 | q3 | q4 | q5 | q6 | q7 | q8 | q9 | q10 | SUS Score |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **p1** | 5 | 1 | 5 | 1 | 5 | 1 | 5 | 1 | 5 | 1 | **100.0** |
| **p2** | 4 | 2 | 5 | 1 | 5 | 1 | 5 | 1 | 5 | 1 | **95.0** |
| **p3** | 3 | 3 | 4 | 2 | 4 | 2 | 4 | 3 | 3 | 2 | **65.0** |
| **p4** | 4 | 2 | 4 | 2 | 4 | 2 | 4 | 2 | 4 | 2 | **75.0** |
| **p5** | 4 | 3 | 4 | 3 | 3 | 3 | 4 | 2 | 3 | 3 | **60.0** |
| **p6** | 3 | 2 | 4 | 1 | 3 | 3 | 4 | 2 | 4 | 1 | **72.5** |
| **p7** | 5 | 2 | 4 | 3 | 4 | 3 | 3 | 2 | 4 | 2 | **70.0** |
| **p8** | 4 | 1 | 5 | 1 | 5 | 1 | 5 | 1 | 5 | 1 | **97.5** |
| **p9** | 5 | 2 | 5 | 1 | 4 | 2 | 5 | 1 | 5 | 1 | **92.5** |

Table 6 System usability scale results

The table above shows the results from all the participants in this inquiry. The minimum score was 60 and the maximum was 100. The average result from SUS is 80.83 which is a good result. According to [21], the average SUS score is 68 leaving us in a comfortable position. Even though our sample is quite small, the same study demonstrates that the final result from SUS should fall around the same values from large or small samples.

In conclusion, our sample was small but it does not invalidate the results obtained from it. The average result is very impressive and indicates that the system achieves its initial goal regarding usability.

## 6.3 Conclusions

As far as the system usability is concerned, we can safely say that we have achieved our goals by implementing an easy-to-use system. Due to the context where the system is installed, users will only use the system once and for short periods of time. Being easy to use they will not need to spend time learning how to use the system.

The usage data analysis brought some unexpected results. It was interesting that the flight plugin and the weather plugin were the most frequently accessed, but regarding total time spent on a plugin, the news plugin really stood out. Also, the media gallery made an impression on this axis.

As far as usage count is concerned, the result is fairly impressive. In one week we had 290 distinct sessions (different users). In four weeks 261 boarding passes were scanned from 180 different sessions (users).

The usage data is not to be treated as 100 % exact for reasons we explained earlier about the difficulty of identifying different sessions. We do not know if our data would be better or worse if it were more reliable.

# 7 CONCLUSIONS

In this project we aimed to develop a new interaction channel between Madeira Airport and its passengers. Following an adapted process from RAD and Agile we were able to identify and implement a set of valuable services that create a better experience for the airport visitors. Using the touch technology on a large screen is a challenge that falls outside the scope of the new personal devices such as smartphones and tablets as it is completely different ergonomically. A solid platform was developed from the ground up offering a number of possible customizations and extensions.

The main conclusion we can draw from the project is that, even in a small airport like Madeira Airport (2.6 million passengers in 2015 as opposed to Lisbon airport with 20 million passengers in 2015 for example), when passengers approach devices such as the touchscreens used in the project, they are mainly looking for additional information. This is aligned with our initial findings from the brainstorm session we held and supported by the data collected during the use of the system in real settings. The fact that touch screens are identified with the airport brand enforces the user expectations regarding information related to the airport services and flights. Also, it creates a trust relationship and gives credibility to the information acquired from the system.

The results obtained from the project clearly show that the use of such devices is an alternative way to interact and communicate with airport users, giving them more control and easy access to relevant information. Like self-check-in machines, the

establishment of this interaction channel could release airport staff to deal with more important requests and increase the overall quality of service.

From a technical point of view, the touch technology existent on the monitors is, given the technology nowadays, quite outdated. That is, the support for multi-touch is not present, preventing us from exploring gestures from users. Gestures could have brought a completely different interaction design to the end result and created a better experience for our users. Furthermore, we noticed that the friction between the user finger and the touch monitor glass was quite uncomfortable preventing us from exploring drag operations on a larger scale.

With regard to the architecture developed a highly adaptable system was achieved. It is a fully functional plugin system, very easy to customise and maintain and works as it should, having been in use for three months so far without crashing, with precise session logging, even in situations where there is no connectivity from the server to the internet or from the kiosk to the server. During the first week of production a bug in the logging mechanism was detected leading to false session times which resulted in incorrect usage data and, in some cases, crashed the system due to a null pointer exception. The bug was detected and fixed timeously. New plugins can be easily developed by anyone, simply adding a reference to our assembly (core DLL file), without the need to change any of the core code. As far as the management application is concerned, the system has a user and roles system with a fine-grained solution to provide specific users access to specific areas of the application and protecting against inappropriate use.

The most challenging plugin built for this project was the flight data plugin because a lot of components had to be developed. This plugin involves a barcode scanner for reading the data from the user boarding passes and with that data, finding their flight and updating the data in a live manner. It required a lot of research and hard work to make all the pieces work together.

We started the project without a designer as he was only included after the project had begun. However, the timing could not have been better as it was a critical point for designing the interfaces and interactions. As a matter of fact, the author of this project learned a valuable lesson about including designers in projects as they bring new ways of thinking, resulting in a better end result. The improvements to the look and feel from all the user interfaces are clear.

We did not follow a web-based approach. A touch-based approach was a better choice given the available hardware and the nature of the system. In essence, our approach involved sizing and positioning with touch in mind and the size of the screen used.

From the airport management company the project brings a renewable and more useful use of the touch screens which were already in place. More importantly, we believe that the project opens new business opportunities for the company, if further development of the system is strategically decided.

We have developed a small set of services that reflects the findings from our development process but other services could be explored. Marketing is one of the strands where the airport could generate income from its shops and other partners by using a fixed spot to display advertisements with an associated cost. Furthermore, the system can be used by passengers to buy and collect their purchased items at some specific point in the airport terminal.

The architecture of the system can be improved at any time to provide a more extensive API for the plugins. At the moment, the API includes features that were needed by the plugins developed for this project.

Other uses of the developed technology are being considered and could bring the project to other contexts. These new challenges will certainly show the quality of the system and further improve all aspects developed so far. The system can be deployed in production on any surface area where visitors need additional information, for example, shopping centres, museums, amusement parks etc.

All in all, it is just another brick in the wall, we have left the door open for the emergence of new ideas that could add value to the platform and provide greater experiences to Madeira Airport passengers.

# 8 REFERENCES


[1] E. A. Johnson, “Touch display—a novel input/output device for computers,” *Electronics Letters*, vol. 1, no. 8, p. 219, 1965.

[2] E. A. JOHNSON, “Touch Displays: A Programmed Man-Machine Interface,” *Electronic Letters*, vol. 10, no. 2, pp. 271–277, 1967.

[3] N. W. Orr, V. D. Hopkin, and R. I. O. A. M. F. (England), *The Role of the Touch Display in Air Traffic Control*. Defense Technical Information Center, 1968 [Online]. Available: https://books.google.pt/books?id=ksusNwAACAAJ. [Accessed: 18-Feb-2016]

[4] G. Dalakov, “Touch Screen.” [Online]. Available: http://history-computer.com/ModernComputer/Basis/touch_screen.html. [Accessed: 18-Feb-2016]

[5] “Touchscreen.” [Online]. Available: https://en.wikipedia.org/wiki/Touchscreen. [Accessed: 18-Feb-2016]

[6] R. Downs, “Using resistive touch screens for Human/Machine interface,” 2005. [Online]. Available: http://www.ti.com/lit/an/slyt209a/slyt209a.pdf. [Accessed: 19-Feb-2016]

[7] T. V. Monitors, “The 5 Types Of Touch Screen Technology.” [Online]. Available: http://www.tru-vumonitors.com/images/Touch_Screen_Basics.Comparisons.pdf. [Accessed: 18-Feb-2016]

[8] Kiosystem, “Zetta Kiosystem.” [Online]. Available: http://www.kiosystem.pt/website/images/stories/zetta/Zetta.pdf. [Accessed: 18-

Feb-2016]

[9] E. Touch, “IntelliTouch Plus Surface Acoustic Wave Specifications,” 2014. [Online]. Available: http://media.elotouch.com/pdfs/datasheets/Technologies/DS000045_IntelliTouch_Plus_a.pdf. [Accessed: 18-Feb-2016]

[10] R. Clodfelter and J. Overstreet, “Technological Profile Of Shopping Centers ,” *Journal of Shopping Center Research*, 2015 [Online]. Available: http://jrdelisle.com/JSCR/IndArticles/Clodfelter_N196.pdf. [Accessed: 18-Feb-2016]

[11] “Alice Virtual Recepcionist.” [Online]. Available: http://alicereceptionist.com/. [Accessed: 18-Feb-2016]

[12] oemKIOSKS, “ShowPath Virtual Directory.” [Online]. Available: http://www.oemkiosks.com/store/datasheets/2010/Catalogo_ShowPath_UK.pdf. [Accessed: 18-Feb-2016]

[13] M. Zaddach, “InfoGate - The benefits of digitalizing the contact with the passenger,” 2015. [Online]. Available: https://www.sita.aero/globalassets/microsites/atis-2015/insight-sessions-presentations/michael-zaddach-insight-sessionb-atis2015.pdf. [Accessed: 18-Feb-2016]

[14] A. Phillips, “The Need for Rapid Prototyping in an Agile Age,” 2013. [Online]. Available: http://www.developer.com/design/the-need-for-rapid-prototyping-in-an-agile-age.html. [Accessed: 18-Feb-2016]

[15] J. Teixeira, L. Patrício, N. Nunes, and L. Nóbrega, “Customer Experience Modeling: Designing Interactions for Service Systems,” in *Human-Computer Interaction – INTERACT 2011*, vol. 6949, Campos, Pedro and Graham, Nicholas and Jorge, Joaquim and Nunes, Nuno and Palanque, Philippe and Winckler, Marco, Ed. Springer Berlin Heidelberg, 2011, pp. 136–143 [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23768-3_11

[16] G. Hall, *Pro WPF and Silverlight MVVM*. Springer, 2010.

[17] A. Nathan, *WPF 4.5 Unleashed*. Pearson Education, 2013.

[18] V. B. Zhijun Zhang and B. Shneiderman, “Perspective-based Usability Inspection: An Empirical Validation of efficacy,” *Department of Computer Science, University of Maryland*, 1998.

[19] C. Wilson, “User Interface Inspection Methods: A User-Centered Design Method,” 2014.

[20] J. Brooke, “SUS-A quick and dirty usability scale,” *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[21] “10 Things To Know About The System Usability Scale .” [Online]. Available: http://www.measuringu.com/blog/10-things-SUS.php

[22] B. Ink, “Rapid Application Development,” *Blue Ink, Automated Architecture*, 2005. [Online]. Available: http://www.blueink.biz/RapidApplicationDevelopment.aspx. [Accessed: 18-Feb-2016]

[23] “Rapid application development.” [Online]. Available: https://en.wikipedia.org/wiki/Rapid_application_development. [Accessed: 18-Feb-2016]

[24] “Rapid Application Development.” [Online]. Available: http://www.ftms.edu.my/images/Document/IMM006%20-%20RAPID%20APPLICATION%20DEVELOPMENT/Chapter%202nnote.pdf. [Accessed: 18-Feb-2016]

# 9 APPENDICES

# Appendix A. WIFI SMALL TILE IMPLEMENTATION

```
<UserControl x:Class="Sindig.Plugin.Wifi.Views.SingleTileNormalView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d" Background="Transparent" d:DesignHeight="300" d:DesignWidth="300">
    <UserControl.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="pack://application:,,,/MahApps.Metro.Resources;component/Anam.xaml" />
                <ResourceDictionary Source="pack://application:,,,/MahApps.Metro.Resources;component/Hugo.xaml" />
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </UserControl.Resources>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"></ColumnDefinition>
            <ColumnDefinition Width="Auto"></ColumnDefinition>
            <ColumnDefinition Width="*"></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <Rectangle Grid.Row="1" Grid.Column="1" Fill="White" Width="140" Height="100">
            <Rectangle.OpacityMask>
                <VisualBrush Visual="{StaticResource Wi_Fi_Icon}" Stretch="Fill" />
            </Rectangle.OpacityMask>
        </Rectangle>
        <TextBlock Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="3" VerticalAlignment="Bottom"
Margin="0,0,0,10" Text="{DynamicResource Wifi-Instructions}" HorizontalAlignment="Center"
Foreground="White" FontSize="{StaticResource XXX-Small-Font}"></TextBlock>
    </Grid>
</UserControl>
```

# Appendix B. PLUGIN IMPLEMENTATION

```
public class Wifi : ObservableObject, IPlugin, ITile
  {
    private readonly List<int> _data;
    private bool _selected;
    public Wifi(List<int> data)
    {
      _data = data;
    }
    #region IPlugin implemetation
    public event EventHandler<PluginEventArgs> OnChangeToPlugin;
    public event EventHandler<EventArgs> OnLogout;
    public event EventHandler<StringMessageEventArgs> OnToast;
    public int DataA => _data[0];
    public int DataB => _data[1];
    public int DataC => _data[2];
    public int DataD => _data[3];
    public int DataZ => _data[4];
    public string Icon => "Wi_Fi_Icon";
    public string TileColor => "#FF339933";
    public string Versao => "Sindig.Plugin.Wifi";
    public int Id => _data[5];
    public string Nome => "Wifi";
    public Type AppView { get; } = Type.GetType("Sindig.Plugin.Wifi.Views.WifiAppView");
    public IPluginViewModel AppViewModel { get;  } = new WifiAppViewModel();
    public bool Selected
    {
      get { return _selected; }
      set{ _selected = value; RaisePropertyChanged("Selected"); }
    }
    public async Task LoadData() => await AppViewModel.LoadDataAsync().ConfigureAwait(false);
    public async Task SetLanguage(string sigla)
    {
      // add the template dictinary to the application resources
      var dict = new ResourceDictionary();
      string uri = @"pack://application:,,,/Sindig.Plugin.Wifi;component/languages/wifi" + sigla + ".xaml";
```

```
        dict.Source = new Uri(uri, UriKind.Absolute);
        Application.Current.Resources.MergedDictionaries.Add(dict);
    }
    public void Exit() {/* No action */}
    public async Task InitializeWithParameters(object param) {/* No action */}
    public void Logout(){/* No action */}
    #endregion
    #region ITile interface
    public Type CustomTileView { get; }
    public Type HorizontalTileView { get; }
    public Type SingleTileView { get; } = Type.GetType("Sindig.Plugin.Wifi.Views.SingleTileNormalView");
    public Type SquareTileView { get; }
    public Type VerticalTileView { get; }
    #endregion
}
```

# Appendix C. IPLUGIN INTERFACE

```
namespace Sindig.Core
{
    public interface IPlugin
    {
        Task LoadData();
        event EventHandler<EventArgs> OnLogout;
        event EventHandler<StringMessageEventArgs> OnToast;
        event EventHandler<PluginEventArgs> OnChangeToPlugin;
        int Id { get; }
        string Versao { get; }
        string TileColor { get; }
        string Icon { get; }
        int DataA { get; }
        int DataB { get; }
        int DataC { get; }
        int DataD { get; }
        int DataZ { get; }
        string Nome { get; }
        IPluginViewModel AppViewModel { get; }
        Type AppView { get; }
        Task SetLanguage(string sigla);
        void Exit();
        void Logout();
        Task InitializeWithParameters(object param);
        bool Selected { get; set; }
    }
}
```

# Appendix D. BARCODE FLIGHT DATA ENCODED



### 5.2.2. Encoding one flight leg

The M format is used for single leg or multiple legs. Encoding one leg will set the value of 'number of legs encoded' to '1'. The example below (see fig. 23) uses a fixed-length field with 60 positions, according to the BCBP standard published in June 2009.

| | Item number | Element Description | Field Size | Unique / repeated | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mandatory items | 1 | Format Code | 1 | U | M | | | | | | | | | | | | | | | | | | | | |
| | 5 | Number of Legs Encoded | 1 | U | 1 | | | | | | | | | | | | | | | | | | | | |
| | 11 | Passenger Name | 20 | U | D | E | S | M | A | R | A | I | S | / | L | U | C | | | | | | | | |
| | 253 | Electronic Ticket Indicator | 1 | U | E | | | | | | | | | | | | | | | | | | | | |
| | 7 | Operating carrier PNR Code | 7 | R | A | B | C | 1 | 2 | 3 | | | | | | | | | | | | | | | |
| | 28 | From City Airport Code | 3 | R | Y | U | L | | | | | | | | | | | | | | | | | | |
| | 38 | To City Airport Code | 3 | R | F | R | A | | | | | | | | | | | | | | | | | | |
| | 42 | Operating carrier Designator | 3 | R | A | C | | | | | | | | | | | | | | | | | | | |
| | 43 | Flight Number | 5 | R | 0 | 8 | 3 | 4 | | | | | | | | | | | | | | | | | |
| | 46 | Date of Flight | 3 | R | 2 | 2 | 6 | | | | | | | | | | | | | | | | | | August 14th |
| | 71 | Compartment Code | 1 | R | F | | | | | | | | | | | | | | | | | | | | |
| | 104 | Seat Number | 4 | R | 0 | 0 | 1 | A | | | | | | | | | | | | | | | | | |
| | 107 | Check-in Sequence Number | 5 | R | 0 | 0 | 2 | 5 | | | | | | | | | | | | | | | | | |
| | 113 | Passenger Status | 1 | R | 1 | | | | | | | | | | | | | | | | | | | | |
| | 6 | Field size of following variable size field | 2 | R | 0 | 0 | | | | | | | | | | | | | | | | | | | 0 in Decimal = 00 in Hexidecimal |
| Conditional items - Flight segment #1 | 8 | Beginning of version number | 1 | U | | | | | | | | | | | | | | | | | | | | | |
| | 9 | Version number | 1 | U | | | | | | | | | | | | | | | | | | | | | |
| | 10 | Field size of following structured message - unique | 2 | U | | | | | | | | | | | | | | | | | | | | | |
| | 15 | Passenger Description | 1 | U | | | | | | | | | | | | | | | | | | | | | |
| | 12 | Source of check-in | 1 | U | | | | | | | | | | | | | | | | | | | | | |
| | 14 | Source of Boarding Pass Issuance | 1 | U | | | | | | | | | | | | | | | | | | | | | |
| | 22 | Date of Issue of Boarding Pass | 4 | U | | | | | | | | | | | | | | | | | | | | | |
| | 16 | Document Type | 1 | U | | | | | | | | | | | | | | | | | | | | | |
| | 21 | Airline Designator of boarding pass issuer | 3 | U | | | | | | | | | | | | | | | | | | | | | |
| | 23 | Baggage Tag Licence Plate Number (s) | 13 | U | | | | | | | | | | | | | | | | | | | | | |
| | 17 | Field size of following structured message - repeated | 2 | R | | | | | | | | | | | | | | | | | | | | | |
| | 142 | Airline Numeric Code | 3 | R | | | | | | | | | | | | | | | | | | | | | |
| | 143 | Document Form/Serial Number | 10 | R | | | | | | | | | | | | | | | | | | | | | |
| | 18 | Selectee indicator | 1 | R | | | | | | | | | | | | | | | | | | | | | |
| | 108 | International Documentation Verification | 1 | R | | | | | | | | | | | | | | | | | | | | | |
| | 19 | Marketing carrier designator | 3 | R | | | | | | | | | | | | | | | | | | | | | |
| | 20 | Frequent Flyer Airline Designator | 3 | R | | | | | | | | | | | | | | | | | | | | | |
| | 236 | Frequent Flyer Number | 16 | R | | | | | | | | | | | | | | | | | | | | | |
| | 89 | ID/AD Indicator | 1 | R | | | | | | | | | | | | | | | | | | | | | |
| | 118 | Free Baggage Allowance | 3 | R | | | | | | | | | | | | | | | | | | | | | |
| | 4 | For individual airline use | Var | R | | | | | | | | | | | | | | | | | | | | | |
| Security | 25 | Beginning of Security Data | 1 | U | | | | | | | | | | | | | | | | | | | | | |
| | 28 | Type of Security Data | 1 | U | | | | | | | | | | | | | | | | | | | | | |
| | 29 | Length of Security Data | 2 | U | | | | | | | | | | | | | | | | | | | | | |
| | 30 | Security Data | Var | U | | | | | | | | | | | | | | | | | | | | | |

*Figure 23 - Format M example one flight leg*

# Appendix E. SOAP SERVER FOR ANA RESOURCE BROKER

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:s="http://www.w3.org/2001/XMLSchema"
             xmlns:s0="http://www.outsystems.com"
             xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:tns="http://Sindig.Master/ResourceBroker/"
             xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
             xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
             targetNamespace="http://Sindig.Master/ResourceBroker/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
      <s:schema elementFormDefault="qualified"
         targetNamespace="http://www.outsystems.com">
          <s:element name="CheckConn">
             <s:complexType>
                <s:sequence>
                   <s:element minOccurs="1" maxOccurs="1"
         name="Subscription" type="s:int" />
                   <s:element minOccurs="0" maxOccurs="1"
         name="Secret" type="s:string" />
                </s:sequence>
             </s:complexType>
          </s:element>
          <s:element name="CheckConnResponse">
             <s:complexType>
                <s:sequence>
                   <s:element minOccurs="1" maxOccurs="1"
         name="Result" type="s:boolean" />
                </s:sequence>
             </s:complexType>
          </s:element>
           <s:element name="ScoreDataAvailable">
              <s:complexType>
                 <s:sequence>
```

```
            <s:element minOccurs="1" maxOccurs="1"
      name="Subscription" type="s:int" />
            </s:sequence>
          </s:complexType>
        </s:element>
        <s:element name="ScoreDataAvailableResponse">
          <s:complexType />
        </s:element>
        <s:element name="ResourceDataAvailable">
          <s:complexType>
            <s:sequence>
              <s:element minOccurs="1" maxOccurs="1"
      name="Subscription" type="s:int" />
            </s:sequence>
          </s:complexType>
        </s:element>
        <s:element name="ResourceDataAvailableResponse">
          <s:complexType />
        </s:element>
      </s:schema>
    </types>
    <message name="CheckConnSoapIn">
      <part name="parameters" element="s0:CheckConn" />
    </message>
    <message name="CheckConnSoapOut">
      <part name="parameters" element="s0:CheckConnResponse" />
    </message>
    <message name="ScoreDataAvailableSoapIn">
      <part name="parameters" element="s0:ScoreDataAvailable" />
    </message>
    <message name="ScoreDataAvailableSoapOut">
      <part name="parameters" element="s0:ScoreDataAvailableResponse" />
    </message>
    <message name="ResourceDataAvailableSoapIn">
      <part name="parameters" element="s0:ResourceDataAvailable" />
    </message>
    <message name="ResourceDataAvailableSoapOut">
```

```
    <part name="parameters" element="s0:ResourceDataAvailableResponse" />
  </message>
  <portType name="ResourceBrokerSoap">
    <operation name="CheckConn">
      <input message="tns:CheckConnSoapIn" />
      <output message="tns:CheckConnSoapOut" />
    </operation>
    <operation name="ScoreDataAvailable">
      <input message="tns:ScoreDataAvailableSoapIn" />
      <output message="tns:ScoreDataAvailableSoapOut" />
    </operation>
    <operation name="ResourceDataAvailable">
      <input message="tns:ResourceDataAvailableSoapIn" />
      <output message="tns:ResourceDataAvailableSoapOut" />
    </operation>
  </portType>
  <binding name="ResourceBrokerSoap" type="tns:ResourceBrokerSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
    <operation name="CheckConn">
      <soap:operation
        soapAction="http://Sindig.Master/ResourceBroker/CheckConn"
        style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
    <operation name="ScoreDataAvailable">
      <soap:operation
        soapAction="http://Sindig.Master/ResourceBroker/ScoreDataAvailab
le" style="document" />
      <input>
        <soap:body use="literal" />
      </input>
```

```
            <output>
              <soap:body use="literal" />
            </output>
          </operation>
          <operation name="ResourceDataAvailable">
            <soap:operation
              soapAction="http://Sindig.Master/ResourceBroker/ResourceDataAvai
lable" style="document" />
            <input>
              <soap:body use="literal" />
            </input>
            <output>
              <soap:body use="literal" />
            </output>
          </operation>
        </binding>
        <service name="ResourceBroker">
          <port name="ResourceBrokerSoap" binding="tns:ResourceBrokerSoap">
            <soap:address
              location="http://192.168.18.147:8888/resourcebroker.asmx" />
          </port>
        </service>
      </definitions>
```

# Appendix F. ANA RESOURCE BROKER SOAP SERVER DEFINITION

<wsdl:definitions xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://ResourceBroker/ResourceData/" xmlns:s0="http://www.outsystems.com" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://ResourceBroker/ResourceData/">

<wsdl:types>

<s:schema elementFormDefault="**qualified**" targetNamespace="**http://www.outsystems.com**">

<s:element name="getDataExtended">

<s:complexType>

<s:sequence>

<s:element minOccurs="**1**" maxOccurs="**1**" name="**Subscription**" type="**s:int**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**Secret**" type="**s:string**"/>

</s:sequence>

</s:complexType>

</s:element>

<s:element name="getDataExtendedResponse">

<s:complexType>

<s:sequence>

<s:element minOccurs="0" maxOccurs="1" name="Data" type="s0:ArrayOfFlightResourceExtended"/>

</s:sequence>

</s:complexType>

</s:element>

<s:complexType name="ArrayOfFlightResourceExtended">

<s:sequence>

<s:element minOccurs="0" maxOccurs="unbounded" name="FlightResourceExtended" nillable="true" type="s0:FlightResourceExtended"/>

</s:sequence>

</s:complexType>

<s:complexType name="FlightResourceExtended">

<s:sequence>

<s:element minOccurs="**1**" maxOccurs="**1**" name="**Id**" type="**s:int**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**Airport**" type="**s:string**"/>

<s:element minOccurs="**1**" maxOccurs="**1**" name="**URNO**" type="**s:int**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**FlightType**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**MovType**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Registration**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**AirlineCode**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**FlightNr**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**StandCode**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**ArrivalBelt**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Gate**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**CheckInCounters**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**FlightDestination**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**FlightOrigin**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**EstimatedDateTime**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**ScheduleDateTime**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**ControlDateTime**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**ActualDateTime**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**CallSign**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Via**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**CodeShare**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**BlockDateTime**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**BeginBoardingDateTime**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**EndBoardingDateTime**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**ServiceType**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**AircraftType**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**DelayCodes**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**ExitDoor**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Chute**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**BoardingStatus**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**ScheduleDateTimeUTC**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**EstimatedDateTimeUTC**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**ControlDateTimeUTC**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**ActualDateTimeUTC**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**BlockDateTimeUTC**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**BeginBoardingDateTimeUTC**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**EndBoardingDateTimeUTC**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**GateStatus**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**CheckInCounters2**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Gate2**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**ArrivalBelt2**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**GateStatus2**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**BeginBoardingDateTime2**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**EndBoardingDateTime2**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**BeginBoardingDateTimeUTC2**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**EndBoardingDateTimeUTC2**" type="**s:string**"/>
<s:element minOccurs="**1**" maxOccurs="**1**" name="**Mixed**" type="**s:boolean**"/>
<s:element minOccurs="**1**" maxOccurs="**1**" name="**RotationKey**" type="**s:int**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**CreationDateTime**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**OriginOrDestination**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Status**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Nature**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**OperationType**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Classification**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Runway**" type="**s:string**"/>
<s:element minOccurs="**1**" maxOccurs="**1**" name="**SeatNumber**" type="**s:int**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Vias**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**AirlineCodeICAO**" type="**s:string**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**ContraMarca**" type="**s:string**"/>
</s:sequence>
</s:complexType>
<s:element name="**getData**">
<s:complexType>
<s:sequence>
<s:element minOccurs="**1**" maxOccurs="**1**" name="**Subscription**" type="**s:int**"/>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Secret**" type="**s:string**"/>
</s:sequence>
</s:complexType>
</s:element>
<s:element name="getDataResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**Data**" type="**s0:ArrayOfFlightResource**"/>
</s:sequence>
</s:complexType>
</s:element>
<s:complexType name="ArrayOfFlightResource">
<s:sequence>

<s:element minOccurs="0" maxOccurs="unbounded" name="FlightResource" nillable="true" type="s0:FlightResource"/>

</s:sequence>

</s:complexType>

<s:complexType name="**FlightResource**">

<s:sequence>

<s:element minOccurs="**1**" maxOccurs="**1**" name="**Id**" type="**s:int**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**CreationDateTime**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**FlightNr**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**MovType**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**ScheduleDateTime**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**OriginOrDestination**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**Via**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**AircraftType**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**ServiceType**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**Status**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**StandCode**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**ExitDoor**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**ArrivalBelt**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**Registration**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**CheckInCounters**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**Chute**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**Gate**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**EstimateDateTime**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**ActualDateTime**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**BlockDateTime**" type="**s:string**"/>

<s:element minOccurs="**0**" maxOccurs="**1**" name="**CodeShare**" type="**s:string**"/>

</s:sequence>

</s:complexType>

</s:schema>

</wsdl:types>

<wsdl:message name="getDataExtendedSoapIn">

<wsdl:part name="parameters" element="s0:getDataExtended"/>

</wsdl:message>

<wsdl:message name="getDataExtendedSoapOut">

<wsdl:part name="parameters" element="s0:getDataExtendedResponse"/>

</wsdl:message>

<wsdl:message name="**getDataSoapIn**">

<wsdl:part name=**"parameters"** element=**"s0:getData"**/>
</wsdl:message>
<wsdl:message name=**"getDataSoapOut"**>
<wsdl:part name="parameters" element="s0:getDataResponse"/>
</wsdl:message>
<wsdl:portType name=**"ResourceDataSoap"**>
<wsdl:operation name=**"getDataExtended"**>
<wsdl:input message="tns:getDataExtendedSoapIn"/>
<wsdl:output message="tns:getDataExtendedSoapOut"/>
</wsdl:operation>
<wsdl:operation name=**"getData"**>
<wsdl:input message=**"tns:getDataSoapIn"**/>
<wsdl:output message=**"tns:getDataSoapOut"**/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ResourceDataSoap" type="tns:ResourceDataSoap">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name=**"getDataExtended"**>
<soap:operation soapAction="http://ResourceBroker/ResourceData/getDataExtended" style="document"/>
<wsdl:input>
<soap:body use=**"literal"**/>
</wsdl:input>
<wsdl:output>
<soap:body use=**"literal"**/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name=**"getData"**>
<soap:operation soapAction="http://ResourceBroker/ResourceData/getData" style="document"/>
<wsdl:input>
<soap:body use=**"literal"**/>
</wsdl:input>
<wsdl:output>
<soap:body use=**"literal"**/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:binding name="ResourceDataSoap12" type="tns:ResourceDataSoap">

<soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="**getDataExtended**">
<soap12:operation soapAction="http://ResourceBroker/ResourceData/getDataExtended" style="document"/>
<wsdl:input>
<soap12:body use="**literal**"/>
</wsdl:input>
<wsdl:output>
<soap12:body use="**literal**"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="**getData**">
<soap12:operation soapAction="http://ResourceBroker/ResourceData/getData" style="document"/>
<wsdl:input>
<soap12:body use="**literal**"/>
</wsdl:input>
<wsdl:output>
<soap12:body use="**literal**"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="**ResourceData**">
<wsdl:port name="ResourceDataSoap" binding="tns:ResourceDataSoap">
<soap:address location="http://reb.ana.pt/resourcebroker/ResourceData.asmx"/>
</wsdl:port>
<wsdl:port name="ResourceDataSoap12" binding="tns:ResourceDataSoap12">
<soap12:address location="http://reb.ana.pt/resourcebroker/ResourceData.asmx"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>

# Appendix G. ASYNCHRONOUS COMMAND IMPLEMENTATION

```
namespace Sindig.MVVM
{
   public interface IAsyncCommand : IAsyncCommand<object>
   {
   }
   public interface IAsyncCommand<in T> : IRaiseCanExecuteChanged
   {
      Task ExecuteAsync(T obj);
      bool CanExecute(object obj);
      ICommand Command { get; }
   }
   public class AwaitableDelegateCommand : AwaitableDelegateCommand<object>, IAsyncCommand
   {
      public AwaitableDelegateCommand(Func<Task> executeMethod)
         : base(o => executeMethod())
      {
      }
      public AwaitableDelegateCommand(Func<Task> executeMethod, Func<bool> canExecuteMethod)
         : base(o => executeMethod(), o => canExecuteMethod())
      {
      }
   }
   public interface IRaiseCanExecuteChanged
   {
      void RaiseCanExecuteChanged();
   }
   // And an extension method to make it easy to raise changed events
   public static class CommandExtensions
   {
      public static void RaiseCanExecuteChanged(this ICommand command)
      {
         var canExecuteChanged = command as IRaiseCanExecuteChanged;
         if (canExecuteChanged != null)
            canExecuteChanged.RaiseCanExecuteChanged();
```

```
    }
}
public class AwaitableDelegateCommand<T> : IAsyncCommand<T>, ICommand
{
    private readonly Func<T, Task> _executeMethod;
    private readonly DelegateCommand<T> _underlyingCommand;
    private bool _isExecuting;
    public AwaitableDelegateCommand(Func<T, Task> executeMethod)
        : this(executeMethod, _ => true)
    {
    }
    public AwaitableDelegateCommand(Func<T, Task> executeMethod, Func<T, bool> canExecuteMethod)
    {
        _executeMethod = executeMethod;
        _underlyingCommand = new DelegateCommand<T>(x => { }, canExecuteMethod);
    }
    public async Task ExecuteAsync(T obj)
    {
        try
        {
            _isExecuting = true;
            RaiseCanExecuteChanged();
            await _executeMethod(obj).ConfigureAwait(false);
        }
        finally
        {
            _isExecuting = false;
            RaiseCanExecuteChanged();
        }
    }
    public ICommand Command { get { return this; } }
    public bool CanExecute(object parameter)
    {
        return !_isExecuting && parameter is T && _underlyingCommand.CanExecute((T)parameter);
    }
    public event EventHandler CanExecuteChanged
    {
```

```
        add { _underlyingCommand.CanExecuteChanged += value; }
        remove { _underlyingCommand.CanExecuteChanged -= value; }
      }
      public async void Execute(object parameter)
      {
        await ExecuteAsync((T)parameter).ConfigureAwait(false);
      }
      public void RaiseCanExecuteChanged()
      {
        _underlyingCommand.RaiseCanExecuteChanged();
      }
    }
}
```

# Appendix H. SYNCHRONOUS COMMAND IMPLEMENTATION

```
namespace Sindig.MVVM
{
   public class RelayCommand : ICommand
   {
      #region Fields
      protected readonly Action<object> _execute;
      protected readonly Predicate<object> _canExecute;
      #endregion // Fields
      #region Constructors
      /// <summary>
      /// Creates a new command that can always execute.
      /// </summary>
      /// <param name="execute">The execution logic.</param>
      public RelayCommand(Action<object> execute)
         : this(execute, null)
      {
      }
      /// <summary>
      /// Creates a new command.
      /// </summary>
      /// <param name="execute">The execution logic.</param>
      /// <param name="canExecute">The execution status logic.</param>
      public RelayCommand(Action<object> execute, Predicate<object> canExecute)
      {
         if (execute == null)
            throw new ArgumentNullException("execute");
         _execute = execute;
         _canExecute = canExecute;
      }
      #endregion // Constructors
      #region ICommand Members
      [DebuggerStepThrough]
      public virtual bool CanExecute(object parameters)
      {
```

```
            return _canExecute == null || _canExecute(parameters);
        }
        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }
        public virtual void Execute(object parameters)
        {
            _execute(parameters);
        }
        #endregion // ICommand Members
    }
}
```

# Appendix I. OBSERVABLEOBJECT

```
namespace Sindig.MVVM
{
   public class ObservableObject : INotifyPropertyChanged
   {
      public event PropertyChangedEventHandler PropertyChanged;

      public void RaisePropertyChanged(string propertyName)
      {
         var handler = PropertyChanged;

         if (handler != null)
         {
            handler(this, new PropertyChangedEventArgs(propertyName));
         }
      }
   }
}
```

# Appendix J. RAPID APPLICATION DEVELOPMENT

Prior to the RAD process the development teams were following processes such as the waterfall which resulted in software that did not meet the clients' needs because the application took so long to be built that the requirements had changed before the system was completed. For larger projects, these processes resulted in completed but unusable systems [22]. The problem was that the iteration could not be started before the previous has been finished completely. As a result the requirements changes were not updated during a cycle invalidating all the development that was based on the old requirements.

This process model is based on prototyping and iterative development putting less emphasis on planning tasks and put more emphasis on development [23] with small teams of highly qualified, motivated and experienced members.

RAD is a concept where products can be produced faster and with higher quality. The requirements are gathered through workshops or focus groups and early testing of the prototypes. User testing of design occurs reiterative which means user inputs happens during the whole software development process. The reuse of components is a major factor to reduce overall time of development and improve quality as the components can be target of improvements. RAD works better for projects where the scope is small or the work can be broken in pieces.

The core elements of RAD are [24]:

- Prototyping – should be constructed quickly so end user feedback could be gathered fast and early.
- Iterative development – create incremental versions of the product in short development cycles. In every cycle, the user has an opportunity to provide feedback about the systems being constructed.
- Time boxing – define the time for each cycle and the time for the whole system. This results in cutting off features to finish the project inside the time box.

- Team Members – usage of small experienced teams (2 – 6 max).
- Management approach – Active and involved management to mitigate risks of long development cycles, client misunderstandings and missed deadlines. Must focus on team member selection, motivation and obstacles clearing.
- RAD Tools – software to aid the development activity reducing the amount of time needed for development.

This process consists of four lifecycle stages [24]:

- Requirements planning – users, development team and stake holders discuss the properties of the project such as needs, constraints, scope and requirements. After that, the team is allowed to continue. At the end of this stage, the project estimation should be considered.
- User Design – Development team and end users meet on workshops to analyze in details the requirements. Develop the system structure, build data model (entity relation diagrams), formalize business rules, develop test plans, select appropriate construction approach for the system and prepare a work plan defining the next steps such as transition of the system, the effort required to perform these steps, and a schedule by which these steps can be completed.
- Construction – Development of the system in iterative cycles of development, testing, requirements refining, and development again until the system is completed. Convert the data model into a functional database. Prepare the system for transition, setup for production environment.
- Transition – Install the system in production environment. Train end user to use the new system.

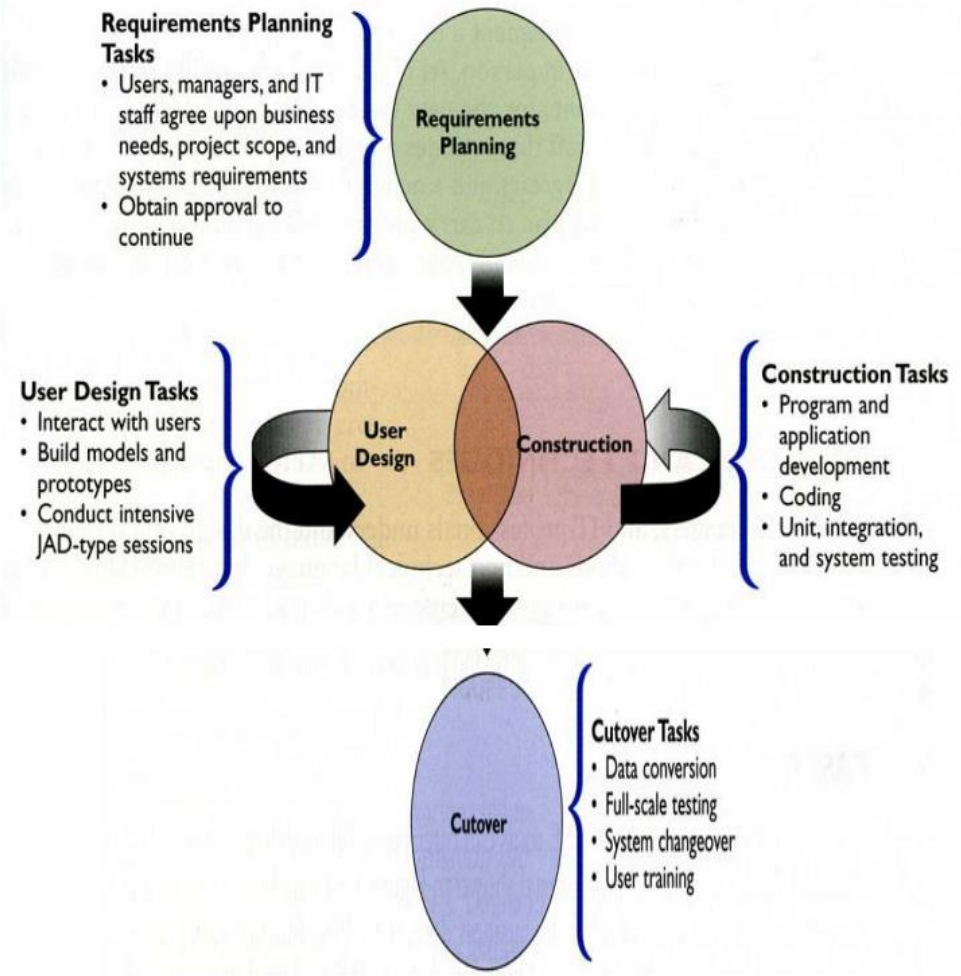



Figure 58 Four phases of the RAD model

The RAD methodology was designed to take full advantage of the tools available to help developers and designers to create the system speeding up development time. Those tools can be requirements gathering tools, computer aided software engineering tools (such as data modelling tools), IDE's, testing tools and communication tools [22].

It is a big risk to freeze requirements in time and develop the systems based on it for a long time. Requirements change, and they change fast and it is our job to be on top of those changes as soon as they happen. During development, it is extremely important

that we identify as much as possible areas where changes might happen and prepare our code to cope with the possible new requirements.

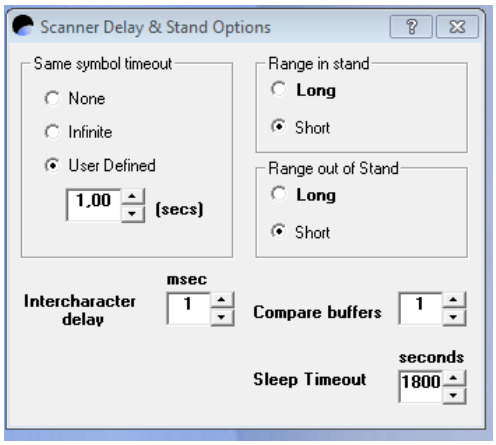
In order to follow the RAD process we had to adapt it to fit our needs. We were a team of one designer one developer and one facilitator, so we were a small team. The software had to be broken into pieces so that the work could divided between both the designer and the developer so we decided that the system should be constructed as a platform that hosts plugins where the plugins could be done independently by implementing a common interface. The team is experienced in web development and general programming, design patterns and design. The one aspect we had to adapt from RAD was the short time to deliver because the team was very small and a plugin system requires a much more complex understanding and logic to be implemented which required more time to develop the system as a whole but not lengthening the development cycles.

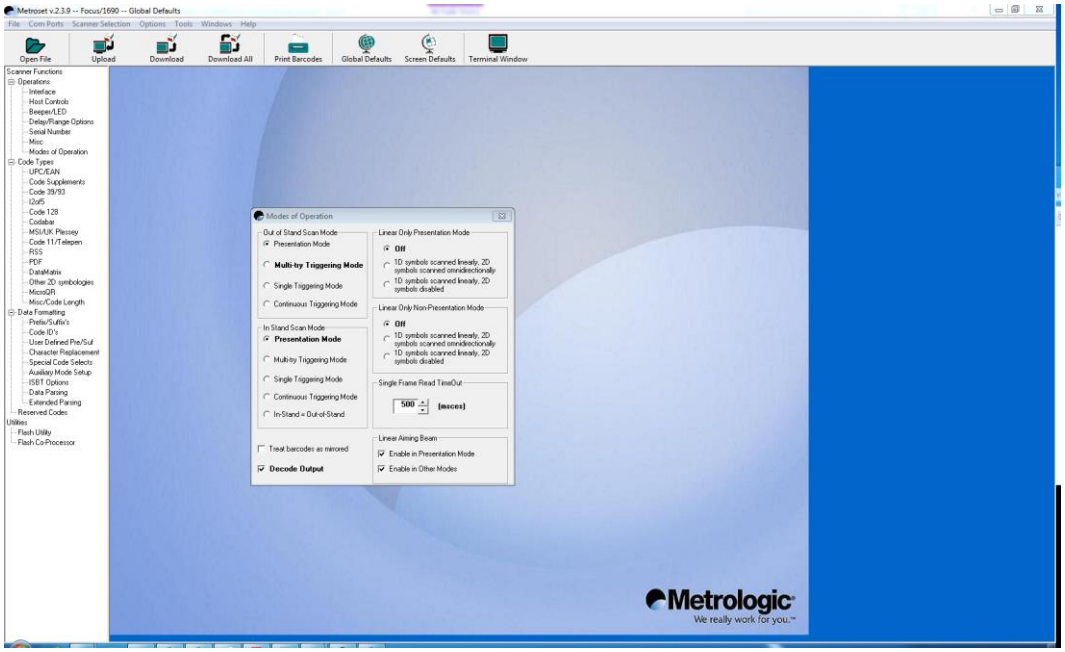
# Appendix K. REFLECTION CLASS

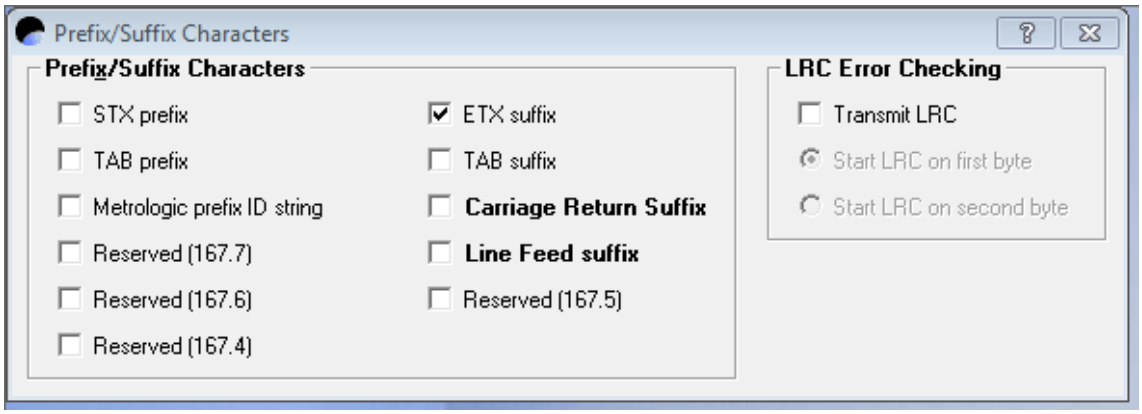
```
namespace Sindig.Boot.Classes
{
   public class DependencyLoader
   {
      public T Add<T>(string path, string matchInterface, params object[] parameters)
      {

         Assembly assembly = JustLoad(path);

         T instance = default(T);

         foreach (Type type in assembly.GetTypes())
         {
            if (type.IsPublic == false) continue;

            if (type.IsAbstract) continue;

            Type tinterface = type.GetInterface(matchInterface);
            if (tinterface == null) continue;

            instance = (T)Activator.CreateInstance(assembly.GetType(type.ToString()), parameters);

            if (instance == null) continue;
            break;
         }
         return instance;
      }
      public Assembly JustLoad(string path)
      {
         var file = LoadFileFromStream(path);

         return Assembly.Load(file);
      }
      private byte[] LoadFileFromStream(string path)
      {
```

```
        byte[] file = null;
        try
        {

            const int bufferSize = 1024;
            using (FileStream fileStream = File.Open(path, FileMode.Open))
            {
                using (var memoryStream = new MemoryStream())
                {
                    var buffer = new byte[bufferSize];
                    int readBytesCount = 0;
                    while ((readBytesCount = fileStream.Read(buffer, 0, bufferSize)) > 0)
                        memoryStream.Write(buffer, 0, readBytesCount);
                    file = memoryStream.ToArray();
                }
            }
        }
        catch
        {
            Console.WriteLine("Failed to load from stream.");
        }
        return file;
    }
  }
}
```

# Appendix L. BARCODE SCANNER CONFIGURATION




Prefix/Suffix Characters
Prefix/Suffix Characters
STX prefix
TAB prefix
Metrologic prefix ID string
Reserved (167.7)
Reserved (167.6)
Reserved (167.4)
ETX suffix
TAB suffix
Carriage Return Suffix
Line Feed suffix
Reserved (167.5)
LRC Error Checking
Transmit LRC
Start LRC on first byte
Start LRC on second byte

# Appendix M. SYSTEM USABILITY SCALE QUESTIONNAIRE

***System Usability Scale***