Università della Svizzera italiana

Faculty of Informatics

Bachelor Thesis

June 19, 2017

# Design Interactive Display Applications

for Active and Walk-by Contact Personalization

Francesco Saverio Zuppichini

*Abstract*

Nowadays the majority of public displays follow a closed model by showing static and passive content. People have started to ignore them due to their low utility. The problem is identified in their lack of user personalisation. Opening them to interactive application that supports user personalisation could create a new communication channel for the 21st century. In the last years, the interest into this field has increased leading to the development of different systems that uses digital signage to deliver personalised content. Our project takes inspiration from previous researches, creating a new architecture that supports users active and walk-by content personalisation. Active contact personalisation allows the viewers to temporally take control of the screen using a mobile application. Walk-by personalisation is the possibility to access the personalised content by walking nearby a display. Our system is composed by five elements: *Map Provider*, a *Proxy*, *Mobile Application* and two display's interactive applications. These applications are: *Transport* and *Upcoming Classes*. In this project we present our prototype implementation of the architecture, the relative design studied and an overview of its main functionalities.

Advisor
Prof. Marc Langheinrich
Assistant
Dr. Ivan Elhart

Advisor's approval (Prof. Marc Langheinrich): Date:

# Contents


**1 Introduction** **2**
1.1 Motivation . . . 2
1.2 State of the Art . . . 3
1.3 Results . . . 3

**2 Architecture** **3**
2.1 Overview . . . 3
2.2 Architecture Elements . . . 4
2.2.1 Map Provider . . . 4
2.2.2 Display Client . . . 5
2.2.3 Transport App . . . 5
2.2.4 Classes App . . . 6
2.2.5 Mobile Application . . . 6
2.2.6 Proxy . . . 7
2.3 Architecture Interactions . . . 7
2.4 Architecture Technologies . . . 9
2.4.1 Back-End . . . 9
2.4.2 Front-end . . . 9
2.5 Architecture Scalability . . . 10

**3 Design Interface** **10**
3.0.1 Display Applications . . . 10
3.0.2 Smartphone . . . 12

**4 Conclusion** **14**

# 1 Introduction

## 1.1 Motivation

Public displays have emerged as a wide used mass communication channel in the last years. They are commonplace in public places such as squares, supermarket, and in our case, universities. Traditionally they were used as a common broadcast medium pushing static and passive content to the audience adopting a closed model. Nowadays, this model is still the most used. This approach can be observed in Figure 1 (a) that shows the New Yorks Times Square while (b) paints the same city in the later half of the 20th century. It can be noticed that displays and posters are used for the same goal: advertisement.

Due to a decreasing of people's interest towards them, displays stakeholders have tried a different approach to gathering back the audience's attention. One of the most common is to just increase the number of screens. This strategy has lead to a ubiquitous deployment of public displays making viewers ignore them even more.


**(a)** Times Square


**(b)** New York in the 60s

**Figure 1**

The major problem is the lack of users personalisation. In the last years, the interested in this field increased and lost of studied were proposed in order to solve the problem. Recently, researches have developed system where the users can ask the displays through a mobile application to show custom content such as map, transit timetables or schedules. Most of them follow the Mark Weiser's [6] vision in which computers becomes part of the environment disappearing by fading in the background. Some examples may be the *e-Campus* [1] in the University of Lancaster that shows interactive content such as Youtube videos, *Tacita* used in this university where students can access an array of campus' applications, PriCal[4] an ambient calendar display that displays a user's schedule and *Instant places* [3] where users can pin their favourite soccer team. By taking advantages of the last technologies advancements, that had brought the touch interface to the displays, we can create interactive applications that allows users to surf content directly on the display making them more attractable.

Our project aims to design an architecture that supports interactive displays' application and active and walk-by contact personalisation. Active, meaning that the user can take temporary control of the display deciding which content will be shown on the screen by selecting preferences through a mobile interface. Walk-by signifies users are able to see their preferences by just walking nearby a display thanks to Bluetooth beacon's places nearby them. This gives an array of advantages, for instance, the viewer can select its content once and just walk to display when he needs it. Moreover, public displays are usually placed in very accessible places making even easier for the audience to find them.

Usually, public displays "*have [...] been investigated from the perspective of single user interaction paradigm*" [2]. In this model, a user can access a rich array of applications and services from the screen. However, since public displays are used by a homogenous audience, multi-user support must be the provider. In our project, we support booth single user and multi-user model by taking advantages of GUI techniques to improve the services' accessibility. One interested case study was conducted in this university [5] where emerged the importance of using a clear design for booth a mobile application and display "*so that people may associate the content they see on the displays with the application generating that content*" [5]

Other studies in this field demonstrate the important of user's privacy in pervasive display networks. A solution was proposed[2] to solved the problem by creating an architecture that "*support displays personalisation without allowing a display network to track and profile users views*". In our project, we took inspiration from these researches in order to also make users' preferences anonymous by using a colour based approach in the system so no other viewers can know whose preferences are.

### 1.2 State of the Art

User users personalisation in pervasive display networks is not a new idea. A number of previous project have been suggest in order to solve the lack of users interactions. However, only a small number of them supports multi users and multi displays stakeholders, such as *e-Campus* [1], *Tacita*, *PriCal* [4] and *Instant Places* [3].

*e-Campus* is a wide public displays network in the Lancaster University that supports interactive application such as Youtube, Flicker and campus-based information visualisation. The system counts thirty displays representing a successful example for our project.

*Tacita* is the current architecture used by *Universita della Svizzera Italiana*. The network supports a range of application that focus on providing useful information to the faculty student. For example, is it possible to get information about bike sharing, exams or public transportation. One limitation is the lacking of users preference.

*PriCal* is a smart interactive calendar display application. The main idea is to let the audience see their schedules directly into the display. A context-adaptive privacy is provided by detecting the viewers and adapting their events visibility according to their preferences. One interesting aspect of this project is the architecture implementation. Their system supports both registered users as well as unknown persons providing a quality network and facilitates interaction by passers-by.

*Instant Places* is a web-based display platform using the metaphors of Pins and Posters. In their study, they created an *application that displays content associated with the soccer teams supported by place visitors*[3]. A pin is created for each team and when a user selects it, the displays shows associated content. At the same time, the application displays a global view of others selected pins giving a feedback to the user about people's preferences.

### 1.3 Results

With our project, we created a network that supports users personalisation into public displays by developing a new architecture with two interactive applications.

The audience can see their preferences by just walking nearby a screen while interacting with the mobile application that may also run in background. Also, information can be retrieved directly from the display using a touch interface, For instance, is possible to see buses' schedules by clicking on a station.

In order to store the state of displays, applications and beacons, a *MAP Provider* were developed. Also, it exposes a web socket is used to broadcast real-time notification through all the system notifying the presence of a user near a screen.

The applications developed are *Transport* and *Upcoming Classes*. *Transport* allows users to select their bus schedules as preferences and access useful content directly on the screen, like see directions from the display to the stations. *Upcoming Classes* shows the semester schedules supporting custom queries in order to discover specific content.

A colour based approach is used to identifier users' preference by only the creator. The user can select a custom colour in the mobile application that will be showed in the display.

## 2 Architecture

### 2.1 Overview

Our system is composed by five elements: *Map Provider*, a *Proxy*, *Mobile Application* and two display's interactive applications. These applications are: *Transport* and *Upcoming Classes*. We structure the web servers using a *Micro Service* architecture in order to decompose the service into different smaller services to improve modularity and make the application easier to understand, develop and test. The components communicate using REST calls and Web Sockets that are proxied to the clients in order to serve all the applications from the same domain as is shown in Figure 2. Each of them exposes an API that can be called in order to get the required data in JSON format.

The *Map Provider* stores the state of displays, beacons, users and the applications' information with the purpose of making them available to the system. A web socket is used to broadcast real-time notification through all the clients to provide real-time feedback about the global network state.

We developed two display's application, *Transport* and *Upcoming Classes*, using the same common interface abstracting them into the *Application Layer* (Figure 3.). Is it possible to individually add new applications by making them conform to our API interface for preference personalisation. Therefore, each application can be added from everywhere allowing developers to use their favourites technologies and hosting platforms. Since they follow a common structure, we will refer to them as the *Application Layer*.

A simple *Proxy* is used to proxied both HTTP request and web sockets for serving each applications from the same Domain avoiding cross-origin requests. Our network supports both HTTP and HTTPS, for security reason, the sockets only works in secure mode.

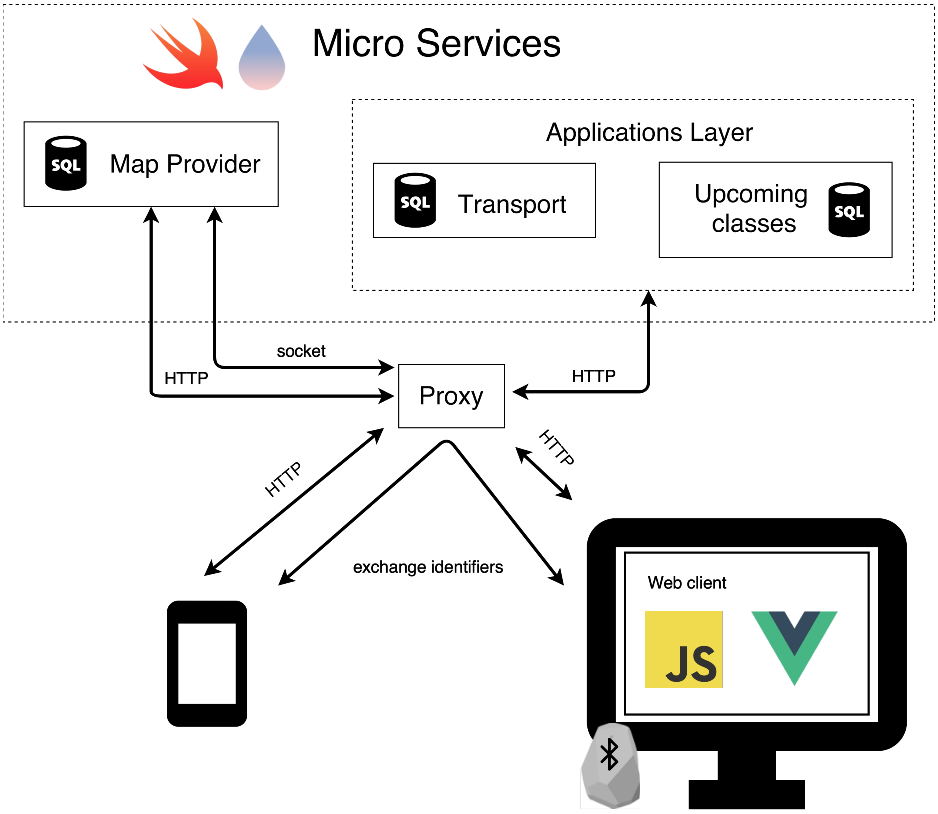



**Figure 2.** Architecture Elements

## 2.2 Architecture Elements

### 2.2.1 Map Provider

The *Map Provider* stores in a database informations about displays, beacons and available applications in order to make it possible for the client to access to the *Application Layer* and have a global knowledge of the current network state. A many to one relation is created between the Display and the Application (Figure 3) table, since, a single screen can run multiple application at the moment. In our project we limited to one main application for screen, but, using a responsive design approach, is it possible to maintain a pleasant look even if one is shrunk due to displays size limitation. Since the *Map Provider* allows to enable and disable custom services, a many to many pivots is used to keep track of local user's applications state. If an application is turned off and a user is near to a display running that application, then no interaction between the two will happen. In our design, this check, is done on the mobile application in order to send fewer information into the socket.

The physical device that makes the in-walk communication possible is the Bluetooth beacon produced. They are produced by Estimote. In order to link them to a screen, a many to one relation between Beacon and Display, since many beacons can be linked to a screen, but no the other way around. The model stores the beacon's mac address as unique identifier. These devices must be putted really close to the machine we want identify in order to increase the location accuracy. Moreover, thanks to our API design, they can be changed at any time by just send the correct request to the server.

A User model is stored with basic information such as email and favourite colour. To know which application was enabled a one to many relations is stored in the Application table.

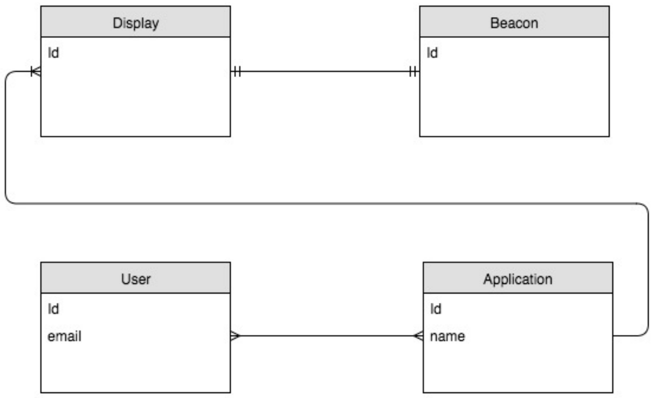



**Figure 3.** Relations in the Map Provider

#### 2.2.2 Display Client

In each Display runs a web browser in which applications, provided by each service server, are shown. For our project, we used a touch display provided from the university with an Intel NCU running Windows 10 connect to it, but our architecture can support any operating system, since it only needs a web client to run it. Since the applications are *interactive* is it possible for the audience to interact with them using a touch interface to click on the Applications' page.

Bluetooth beacons are physically placed nearby to make them visible from the mobile application and create walk-by contact personalisation. They are placed into key points of the university, such as, *mensa* or in entrance of the faculty. Is it very important to accurately study where to place them to maximise the number of viewers. Users can temporally control them by creating preferences that are shown when they walk-by. Also, is it possible to dynamically change the running application by making a REST call to the *Map Provider* that will notify the screen through sockets and make it updates its state.

#### 2.2.3 Transport App

The *Transport* application allows users to navigate the nearby bus station and create preferences through the mobile interface. We decided to gather the data from the well designed Opendata API that allows fetching all kind of transportation information, in our case we used only use buses. However, due to the limit number of request we can make, fixed to three per second, and the necessity to have some custom endpoint, we cloned them.

We also have integrated Google Maps API into the screen's front-end. So, after fetching the exactly display's position thanks to web browser geo localisation, we can show the estimated time to get to a specific station by walk and the correct directions. With this information we can give real-time visible feedback on screen when it is going to be too late to catch a specific bus.

Since we are using a built-in web browsers feature, we can easily scale our system. The image you need to move the display to one location to another, with our architecture there is no need to update its state into the server. Since the information is fetched client side, its position is updated as soon as it is moved, as well as with all the other functionalities related to it, no external operation is needed.

Even if it can be argued that by cloning Opendata API we loose all information about transports delays, we notice that the delay field is always null. So, we have to assume that even them can not have access to such detailed information or it is not implemented yet. For our project, we cloned them every 6 hours, but the schedule can be changed in virtual no time. A the mobile interface is used to manage the preferences. To do so, the user must select the correct station, buses and direction.

Figure 4 shows how a preference is created. From left to right, the user presses a floating button and selection station page appears. Using a step by step approach the preference is created, and, after confirming the creation in the modal, it is added to the user's preferences.

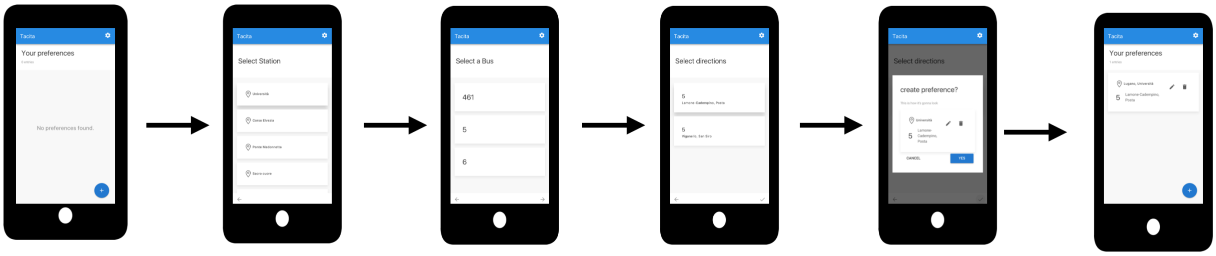



**Figure 4.** UI flow in *Transport*

### 2.2.4 Classes App

The second application is *Upcoming Classes*. It displays information about classes such as the course's schedule for the next days. Is it possible to select preferences or to create queries for specific courses from the screen. All the courses are displayed on the calendar in the correct order supporting booth month and week view. Similarly to the Transport application, we had issues with the API, in our case, provided by the University itself. A single API call to know all the schedules takes more than 1500*ms* as showed in Table 1. It gets worse if you try to get all the courses for a faculty: Therefore they cannot be properly used in a real application. Even if, from the client, we always cache the request,

| Request | Time |
|---|---|
| http://search.usi.ch/api/courses/35255488/schedules | 1952ms |
| http://search.usi.ch/api/faculties/1/courses | 9294ms |

**Table 1**

we cannot avoid waiting for the first time. The reason why they are so slow is the response size. Their response is heavy due to a poorly model population; for each course that is sent back, tonnes of unused field are provided. Also, by inspecting a response for class schedules, we can notice the same huge course object appears, unnecessary, for each schedule object. Therefore, again, we needed to clone all the API in order to just sent the right amount of information, by doing that, the previous schedule request now needs just 18ms. Table 2 shows these results.

| Request | Time |
|---|---|
| http://tacita/classes/api/course/246/schedules | 18ms |
| http://tacita/classes/api/faculty/1/courses | 1000ms |

**Table 2**

The display's front end application is divided into two main part easily identifiable, the calendar and the query engine next to it. The calendar is create using *fullcalendar* jQuery library that does all the dirty work of render and setting up all the events into the corrects slots. The courses can be selected thanks to the query engine on the right part of the screen. As soon as the user clicks on the button representing the faculty, it is guided in order to create a valid query using a step by step approach; even if it may be not the faster way, it is the safest since no wrong request can be generated. The procedure is shown in the following storyboard:

As we did for the Transport Application, we also created a smart phone interface in order to create, remove and edit preferences. For consistency reason, we decide to keep to use the same UI design we used for the transport application.

### 2.2.5 Mobile Application

A mobile application was developed in order to let users select their preferences and custom settings. In our project, each user can select a colour that is shown as a little label near the preference on the screen makes it easier to identify. Since only the content owner knows is colour, anonymity is maintained in the system. The home page shows the applications list. By clicking the application's preferences page will appear allowing the user to manage the preferences.

Is possible to quickly enable/disable content flow for each application by changing the state of a toggle present on each service. When an application is disabled, no information will be sent to the *Map Provider*. However, since we are using socket connection, when an application is enabled the new state will be immediately broadcast to the system making screens react to it.

When a Bluetooth beacon is noticed, a floating button appears making possible to access the display page where detail information about the discovered display is showed. Automatically, users preferences will be displayed on the screen. The interactions are deeply explained into section 2.3. Figure 5 shows a usage example. From left to right, the application is opened, then the *Transport* application is selected and its preference page is open. Later a new colour is picked, in our case blue. After we can observe that a display was found by clicking on the floating button that has just appeared. In the last picture, our preferences are labelled into the display with the chosen colour.

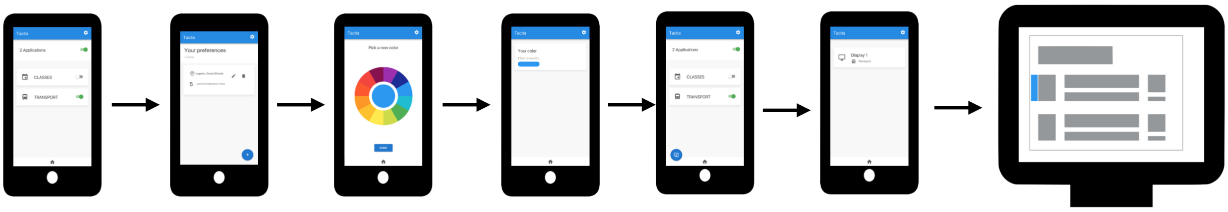


**Figure 5.** Mobile Application UI flow

### 2.2.6 Proxy

A simple *Proxy* was created in order to proxy all the HTTP/HTTPS and web socket request. In this way, a single domain is exposed to the client so no CROS origin request is needed. Due to security reason, sockets are only supported into the secure mode; so it is highly recommended to access our servers with a secure connection in order experience all the features of our project

## 2.3 Architecture Interactions

In the previous part, we define in detail each element of our architecture without giving a global overview of how each part collaborate with the all system. In this section we are going to analyse all the interaction, especially between user and display, showing and explain each message that the entities exchange in order to communicate. From a User point of view, the first action that can happen is walking into a display. As soon as the mobile application detects the beacons, it gets a list of them sorted by distance (action 1. of Figure 6). The first one is fetched and its id is used to make a request to Map Provider (2.) in order to know which display is associated with it. If there is one, a message containing user's id and display's is pushed into the socket (3.) only if the screen application is enabled. An example message:

```
{
  action: 'USER_NEARBY',
  payload: {
    user_id: '1',
    display_id: '1'
  }
}
```

When the display receives the notification, it checks if it has the same id. If it gets the user's preference from the back-end running application (4.). Finally, the user's content is showed on the screen (5.).

Similarly, when a user walks away it lost its connection with the beacon and a new message is pushed into the socket in order to remove its preferences from the display.

```
{
  action: 'USER_EXITS',
  payload: {
    user_id: '1',
    display_id: '1'
  }
}
```

To avoid processing the same information two or more times, the displays uses a in memory cache. Also, a life duration is set for each of them in order to remove them if no exit event is detected.

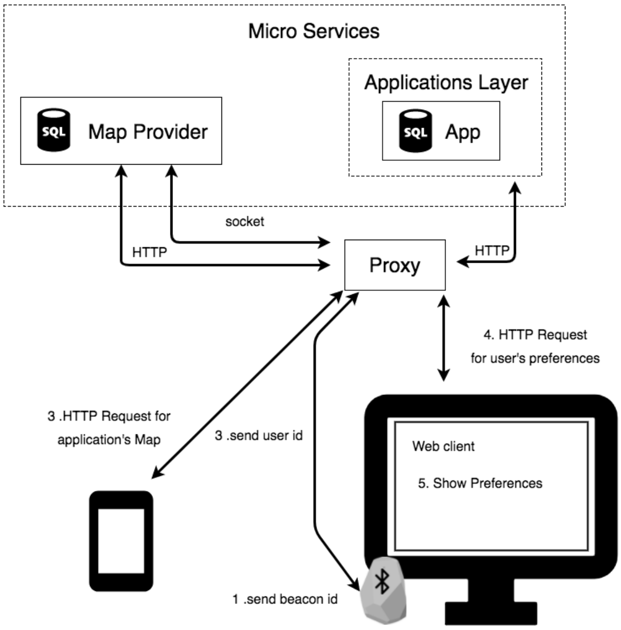



**Figure 6.** Architecture Interactions, User point of view

When the display's web client is connect to one of the applications (action 1. Figure 7), it does some initial setup (2.). Depending on the local case, it may ask for the geo localisation, used in the *Transport application* or create some data structures. After the boot phase is done, a request for the data is made to its provider in order to show informations (3.). In the mean time, the screen sends the applications that its running to the *MAP Provider* (5.) in order to update its state. Then the *Provider* broadcasts to all the clients the new information using sockets so each element has a global real-time view of all the displays state. In our project, the mobile application, when it receives such information and if the new application running is enabled, it sends again the user's identifier in order to show the personalised content into the new application without reloading the view.

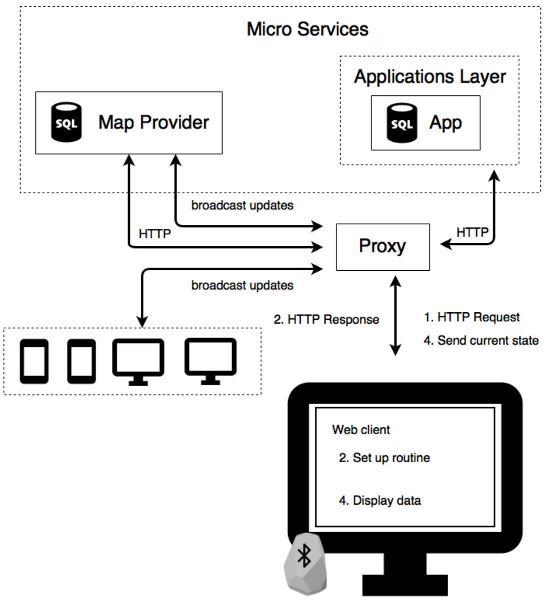



**Figure 7.** Architecture Interactions, Display point of view

At each screen is associated a unique id. Therefore, before deploying a display, it must create a new entity in our database by making the correct request to the *Map Provider*. The id is passed to the display's client by adding it in the end of the URL that is processed by the front-end software.

## 2.4 Architecture Technologies

In this section, we talk about the technologies that we used for the back-end and the front-end. For this project, we choose the latest technologies and programming languages in order to provide a service that will last without being obsolete in a long term.

### 2.4.1 Back-End

We firstly started to develop the first application using Django, a widely used Python web framework, but we encored problems with web socket that were not supported "out of the box". So, we decide to completely switch both language and framework by choosing Swift 3 and Vapor. Swift is a "powerful and intuitive" open source programming language developed by Apple in 2015 for booth MacOS X and Linux. It is a Protocol Oriented Programming Languages. Unlike classes, the fundamental of Protocol Oriented Programming, POP, is Value Type encouraging flat and not nested code. This benefit is reflected in its performance and flexibility. The syntax is concise yet expressive, it uses labels and spaces to improve the code readability making possible to write complete sentences only with code. Apple wanted to create a product that was at the same time, feast and beauty.

After selecting a programming language, we need a framework to create our applications. We choose Vapor, a powerful, beautiful and easy to use MVC framework to build web servers. In our project it powers all micro services. Vapor is easy to start with thanks to the clear APIs that, not also speed up the developer work, but also make the code more maintainable and more understandable. One of his strength points is the performance inherited from Swift. Vapor uses a Model View Controller design pattern that divides an application into three decoupled interconnected parts in order to separate internal representations of information. A *Model* directly manages the data and logic about a database entry such as display or beacon. The *View* outputs a representation of an information, in our case, the web page. The this part, the *Controller* acts on booth *View* and *Model*. It controls the data flow into the model, updating its state, and change the view whenever is needed.

### 2.4.2 Front-end

In order to archive the best results in term of scalability and performance, we choose Vue.js as main front-end framework. Vue.js is a library for building interactive web interfaces, it uses web components to provide a convenient way to organise an application by decoupled its element into blocks. A web component is similar to an Object Oriented programming classes, it encapsulates all the logic behind a certain functionality of the application. In Vue, a component, is formed by three parts: *template*, *script* and *style*.

The *template* allows the developer to write HTML and include variables, conditions and loops. Also, thanks to built-in loaders, is it possible to easily use any pre-processing HTML library such as Pug. The second part, the *script*, is the core of each component. As the name may suggest, it is the code part. Each component exposes a Javascript object allowing Vue to grab it and render it in the proper way. The last tag, *style* is were is possible to custom style a component using CSS or css-processors such as Less or Sass. One of the main advantages of Vue over other front-end framework is *reactivity*. Reactive programming is a synchronous programming paradigm concerned with data streams and the propagation of change. It uses Observers in order to trigger events every time a change of state is detected; therefore Vue can automatically call a render only in the part that was actually mutated minimising the DOM access and increasing speed. A deep comparison with other frameworks can be fount at: https://vuejs.org/v2/guide/comparison.html.

As we said, each component represent a feature, a part of the web application; it may happen that one of them need to communicate with another, maybe after a change of state or a user event. In order manage the data flow we used the Flux pattern. The main idea behind this pattern is that the state can be mutated thought actions that are reduced into the stores. Flux is composed of four parts: *Dispatcher*, *Store*, *Actions* and *View*. The *Dispatcher* is a singleton that receives them and dispatches to every store that have registered with it; its important to highlight that every store receives every action. The *Store* is only source of truth in the applications; it holds its state and manages the logic behind it. The data must only be mutated by responding to and action by emitting a "change" event. *Actions* are the internal API of each application, they define all the possible interaction that may happen. They are plain javascript object composed of a type field and some data. *Views* displays store's data; in our application, a single view is a Vue component.

Even if there already exist a fantastic Flux library, Vuex, for Vue created by its author, we decide to create a new one from scratch. Our library, called Flue, aims to provide a better object oriented approach than the existing one. You can find examples and documentation at our Github repository: https://github.com/FrancescoSaverioZuppichini/Flue.

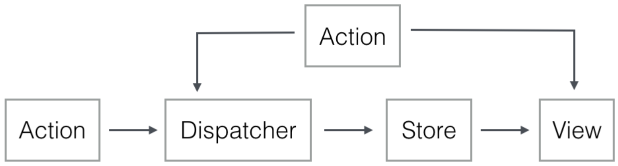



**Figure 8.** Flux data flow

## 2.5 Architecture Scalability

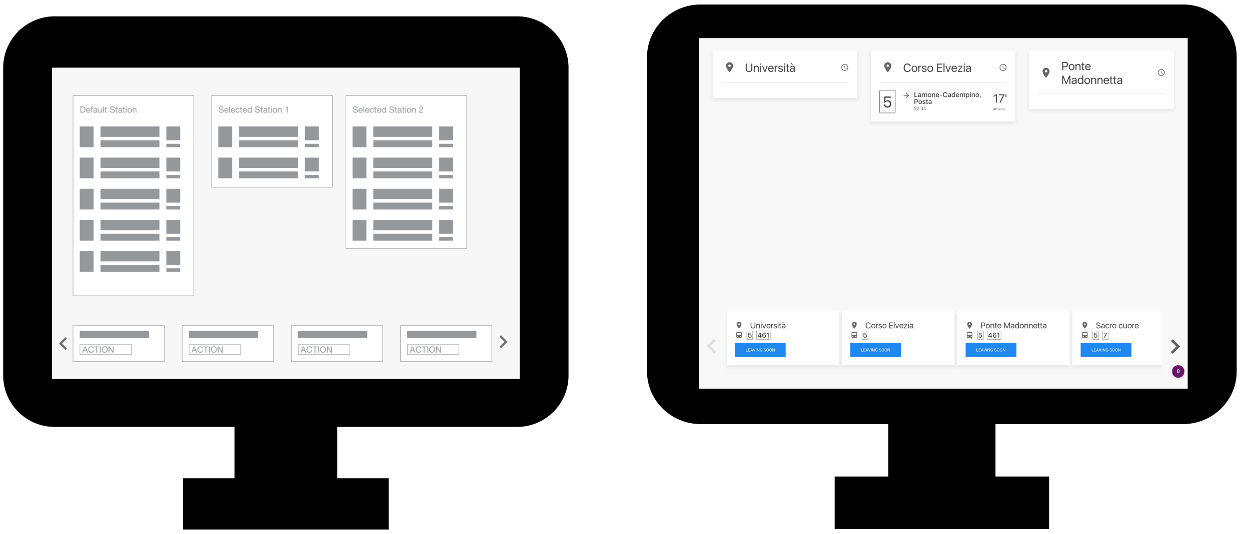
Scalability is the ability of a system to grow larger with minimum changes. In the previous sections we often talk about how some design choices lead to an improvement in term of scalability, such as adopting a micro services approach. In our specific case, our system state can change very quickly at any moment due to external decisions, for example if the provided buy new displays or new applications are available. In order to make it possible to deploy independently from the network a new application, we abstract the its concept creating a layer of services that can be expanded very easily. The developers can create a new applications by just creating a web server being conform to our generic interface and expose his url. Another possible problem that can may appear is when a display is moved away. We explained before how each screen, when its connected, fetch its position and use it in order to communicate with Google Maps API to show real direction indications. If such problem appears, very easily, the provider just need to refresh the screen and the front-end software will automatically update itself. Also displays can be removed or added by simply using *Map Provider* REST API. Imagine screen breaks, then the associated beacon can be moved to another screen or the display may be replaced. The last case is the simplest, since we can just turn on the screen and use the last display's id, if no new device is available then the beacons can be quickly be linked to the a new screen.

# 3 Design Interface

Our application is exposed to a heterogeneous audience, therefore a clear and effective interface is mandatory to ensure a global usability. We have also to provide a no blocking content flow in the displays' applications, meaning that an interaction should not prevent, or block, another. Moreover all the state changing must be displayed using design techniques, such as well targeted animation, in order to make the user understanding what is happening around him. We decide to follow a *content first* strategy by making the content clear and visible using a minimalistic approach; no unused element is present in any application. We choose cards as an only way to organise information. A card is a sheet of material that serves as an entry point to more detailed information proving a convenient way to display content composed of different elements.

### 3.0.1 Display Applications

The main challenge was to design for a big touch interface. We started by keeping in mind that every information must be quickly reachable and accessible; the user must understand in a fraction of second what he needs. By keeping that constrains in mind. Our design process starts with wireframes; representing the skeleton of the application exposing its main functionalities. They are terribly useful to get a first general look of what the application is going to be. For what It concern the Transport service, we begin by discarding every not necessary element following, as we said, a minimalistic approach. In the following picture, you can see the first wireframe's muck up and the final product




**(a)** wireframe **(b)** final product

**Figure 9.** Transport Application

If you look at the left image you can notice that all the functionalities are intuitively recognisable. In our design, studied to be used in public displays, we focused on the interactive part. At the bottom of the application there is a carousel used to change the station on the fly. Two huge arrows allow viewers to easily select mores stations by clicking on them. Moreover, all buttons are big enough to be easily touched from everyone without any problems. We used animation to increase the user experience. For example, when a station is selected is it pushed on the right part using a slide-in in animation making more visible. Also, an open station shakes when a user tries to re-open it, giving a strong visible feedback of where it is on the display.

In the Class application we used the same approach, we started by sketching the wireframes and defining the main functionalities. In this specific case, we need two components: a Calendar and a query engine to search courses.


**(a)** wireframe **(b)** final product

**Figure 10.** Classes Application

### 3.0.2 Smartphone

We develop a *mobile application* to organise the user preferences, choose customisable settings and manage interactive applications. We design it from sketch to be accessible for everyone from everywhere. In main page of the application is possible to see all the existing application exposed by the *Application layer*. In each of them is possible to use a toggle button to enable them or disable. Skeletron loaders were chosen instead of classic spinner to create a more comfortable user experience. Therefore, in case of slow internet, is possible to guess the content of a page. Figure 11 shows this strategy. As you may notice, is it easy to understand how the content is going to appear on the application. We took advantages of such design pattern in almost all of our applications.

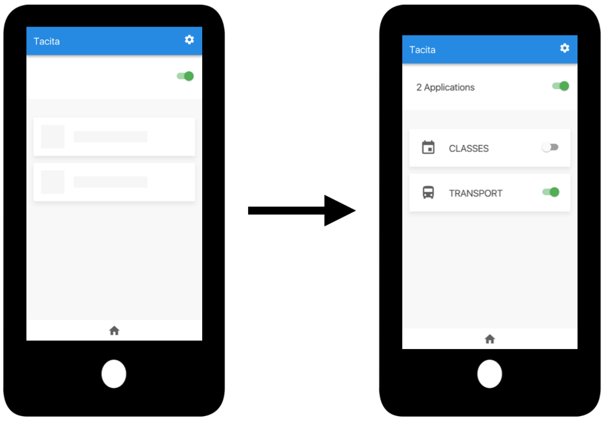



**Figure 11.** Smartphone home page

For each application, a toggle button is present on the right of the container to enable and disable it. Moreover, a global toggle button is available on the top of the page allowing to enable of disable all the applications in one click. In the case that a provider adds a description for an application, then it is possible to read it using the arrow at the bottom of each card. In figure 13 is present the UI flow that occurs when a User walk nearby a screen. A floating button pops up making the displays page available. These buttons represent a great way to quickly add a functionality to a view, we decided to create a pulse animation to increase its visibility in the page. Display page shows all the discovered displays from the detected beacon and, since, we are using socket connection, if the display changes the running application, the client will be notified updating its view.

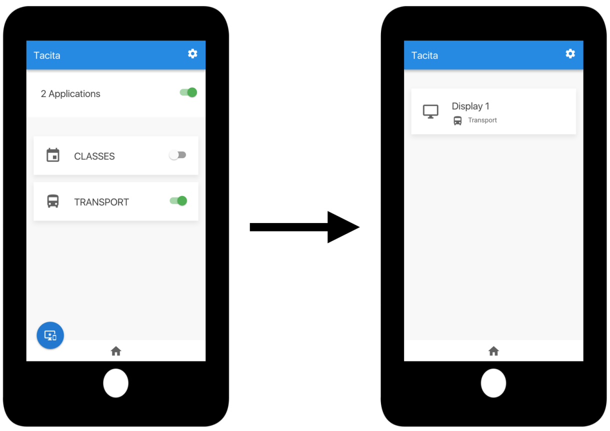



**Figure 12.** Display found

As we said before, We also provide a convenient way to select a custom personalisation. We choose a colour based approach in order to highlight the visibility of an own content. The user can pick up a colour from the wheel and use it to find his preferences on the screens. Moreover, our system can easily support any other identifier such as username or avatar.

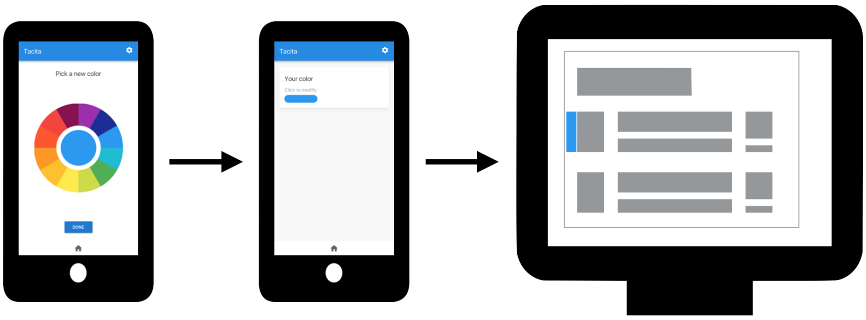



**Figure 13.** Preference on display

In Figure 13, an user selects a "blue" colour as preferences' identifier by clicking on the colour wheel and press "done". Then, we a screen displays the personal content, a little label is added on the left part of preferences. If one of more users walk in, then the label is split as is it shown in Figure 14.

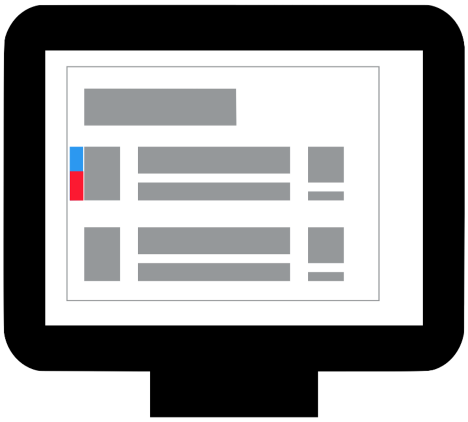


**Figure 14.** Preferences on display

# 4 Conclusion

In this work, we proposed an architecture that supports users personalisation by creating two interactive applications. A *Map Provider* web server provides a global overview of the state of beacons, displays and applications. Through its API is possible to dynamically link and unlink beacons to displays, manage applications and stores basic users information. Our system supports realtime notification that is pushed into sockets and broadcasted to the clients updating their information accordingly. For example, if the displays change its running application then a message containing the new application is dispatched to the others devices in order to always provide a correct feedback to the user.

With a mobile application the viewers can both manage their preferences and discover applications. Specifically, is it possible to quickly enable and disable applications without changing the preferences related to them. Also, a colour can be selected as preference's identifier. This information is stored into the *Map Provider* and used as only way to recognise the owned content. Since only the creator knows the colour, then anonymity is maintained in the system. Moreover, the mobile applications display real-time information about surrounding screens that have been discovered, such as the running applications.

In order to support users personalisation we develop two interactive applications: *Transport* and *Upcoming Classes*. They implement the same common interface providing the same API structure in order to be easily integrated into the system. *Transport* allows users to select their bus schedules as preferences and to navigate content directly on the screen. For example, is possible to get the directions from the local display to one station, or to see the arriving buses timetable. *Upcoming Classes* shows the semester schedules supporting custom queries in order to discover specific content. It uses an event based calendar that supports both week and month views giving the ability to quickly discover all the schedules for a course. Each of them serves two single page web application, one for the display, and one for the mobile application. They were developed using Vue.js a front-end framework that uses the paradigm of web components to create reusable code. Therefore, main of our code is shared across the applications. All the applications support multi users model providing a *non blocking* content flow where each user can discover information without been prevent by others viewers interactions.

One key feature of the project are *walk-by* contact personalisation. In this model, the users can pull its content into the system by just walking nearby a display in which a Bluetooth beacon is placed. The device is able to detect the mobile application's Bluetooth and dispatch its identifier in order to be discovered by the system. This methodology brings some advantages. For instance, it is not required to open the application nearby a display since it can run in background. One limitation is the hardware itself, we had some problems with the beacons. Since a connection delay is added by the manufacture to the them, it may happen that a viewer has to wait some seconds before the mobile application is able to restore the connection with the beacon.

Since our system address a heterogeneous audience, all the applications must be accessible. By keeping this consideration in mind we adopted a minimalistic content first design approach. In each UI the contents are shown into cards keeping logically clear the meaning of each component. Moreover, icons are wide used to provide a fast way to identify elements on the screen. In order to serve all the request from the same domain, a *Proxy* was added to the system to avoid cross-origin requests.

In future work more applications and customisable preferences can be added. For example, an avatar could be also used to identify preferences on the screen. We also plan to integrate the proposed architecture into the current working university system. Due to the system scalability, deploy a new application is relatively easy since it only requires abstract constraints such as API endpoints design. Therefore, a new service can be implemented with any technology and host anywhere. We strongly believe that public displays may become a new communication medium for the 21st century. With this project, we propose possible architecture design to supports users preferences that can be used as a starting point for future works.

# References


[1] S. Clinch, N. Davies, A. Friday1, and C. Efstratiou2. *Reflections on the Long-term Use of an Experimental Digital Signage System*. 2014.

[2] N. Davies1, M. Langheinrich, S. Clinch, I. Elhart, A. Friday, T. Kubitza1, , and B. Surajbali. *Personalisation and Privacy in Future Pervasive DisplayNetworks*. 2014.

[3] B. S. Rui JoseÌĄ, HeÌĄlder Pinto and A. Melro. *Pins and Posters: Paradigms for Content Publication on Situated Displays*. 2014.

[4] F. Schaub, B. K. ÌĹonings, P. Lang, B. ÌĹorn Wiedersheim, C. Winkler, and M. Webe. *PriCal: Context-adaptive Privacyin Ambient Calendar Displays*. 2014.

[5] C. Taivan, R. JosÃľ, B. Silva, I. Elhart, and J. Cardoso. *Design Considerations for Application Selection and Control in Multi-User Public Displays*. 2014.

[6] M. Weiser. *The Computer for the 21st Century*. 2014.