
# Development of Cross Platform Mobile Applications using HTML5

Jade University

Pedro Henrique Fialho Santos, 6007614
Tayla Simões Santos, 6007613
Study Program: Telecommunications Engineering
Bachelor Thesis, 2014
Pr. Dr.-Ing. Heiner Köster, Dipl.-Ing. Udo Willers
Wilhelmshaven, Germany

# Table of Contents

[P] – Written by Pedro Henrique Fialho Santos;
[T] – Written by Tayla Simões Santos;
[P,T] – Written by both.

# Table of Figures and Tables

# 1 Introduction

Developing Cross-Platforms applications is a big deal for developers. On a scenario that you must develop a mobile application, and it must run in iPhone, iPad, iPod, as well as in any Android or Windows devices, definitely a cross-platform programming language can be very helpful. As known, iOS applications use Objective-C, and just switched to Swift; Android uses Java, and Windows uses C#. Therefore, 3 (three) applications must be developed, in 3 (three) different programming languages; and this is without any doubt time wastage.

Most of companies have to have different teams, one for each platform they want to develop, and also good designers for developing each image in different resolutions for each screen size. This will not work if you have a small company with few developers.

That's why Cross Platforms programming languages are becoming more and more useful nowadays. A Kendo UI's survey [4] with 5000 developers global around revealed that when developing apps for multiple platforms, HTML was the technology of choice for the majority.

With HTML5 it is possible to build an application for iOS, Android, Windows Phone and Blackberry at once, with the native look of each one of those. Also, the version control becomes easy, because only one code must be changed.

The power of HTML5 with CSS3 and JavaScript, such as jQuery Mobile is impressive, and is being improved every day. Today, only few features that native applications can make are still missing in cross-platform mobile applications.

There are several IDEs that can be used for developing HTML5 applications, and a lot of JavaScripts as well. We decided to use the Intel XDK environment, which is an IDE that contains its own JavaScript for developers who wish to use their HTML5 expertise to build hybrid HTML5 apps for mobile devices.

This thesis is divided in two parts. The first one is the topics 1 to 4, which is the smaller. This part is a guide for new Intel XDK users. It presents also some HTML5, JavaScript and CSS3 concepts; furthermore, it contains a tutorial for the development of 3 small

applications. All these applications created during this guide and their source codes are attached to this project in the appendix.

The second part of this thesis regards a full operating solution, containing a Web Service and a cross-platform Mobile Application that was developed for the International Office of Jade University.
The Web Service contains a front-end page where the International Office is able to do several things, such as creating informations for the students. It developed with Java language, and the JSP for the pages. The cross-platform Mobile Applications was developed with HTML5, CSS3, and Java Scripts. We used Intel XDK's JavaScript as well as jQuery and jQuery Mobile.

The code from the full operating Web Service and the cross-platform Mobile Application developed with HTML5 in the Intel XDK environment can be found in the appendix.

Moreover, the full thesis, all the source codes, images, libraries, and the "ready to run programs" are contained in a CD that accompanies this thesis.

# 2 Intel XDK

## 2.1 About Intel XDK

This chapter is about clarifying why Intel XDK was used and what are the benefits of it. It also presents other options for HTML5 developers.

### 2.1.1 What is Intel XDK?

Intel XDK is a HTML5 cross-platform solution. It is an integrated development environment (IDE) for cross-platform apps, where you can develop, emulate, test on devices and build apps. It enable developers to write web and hybrid apps once, and deploy across many app stores and form factor devices.

Currently, Intel XDK is available as a free download for Windows 7 & 8, Apple OS X, and Ubuntu Linux.

### 2.1.2 Features

C [5]

**Development Tools:** Built-in intuitive tools that let you design engaging HTML5 responsive apps in less time.

- Quick start theme templates, samples, and demos to speed app design
- App Designer UI builder, a drag & drop round-trip capable tool
- Supports jQuery Mobile, App Framework, Twitter Bootstrap, Topcoat
- Brackets editor with auto-completion and syntax-highlighting
- Apache Cordova 3.x API support for Android and Windows 8 Store
- Integrated GIT source code repository support with the editor
- Use your favorite JS or CSS library
- Web RTC, Vimeo YouTube data stream widget support, plus Twitter feeds within Intel XDK

**Start with Samples:**

- Scalable & Responsive
- Javascript Frameworks, Web Services, Device hardware access & more
- Platform Agnostic & Platform Specific
- Editor & App Designer/App Starter compliant

**Emulate, Test, and Debug Tools:** Emulators, debuggers, and profilers to enhance app performance and quality

- Emulator debugger, JS debugger, and JS remote debugger for Android devices
- App Preview, On-Device App Tester, and On-Device App Debugger
- Live Development for simultaneous previewing of app edits with automatic syncing on an Android Device
- Performance Profiler for Android, plus Xlint compatibility static checker support
- Emulator app testing in 17 different device views with GPS, broadband, accelerometer, and network simulation
- Crosswalk web runtime for Android extends Hybrid app capabilities, enables access to device and native platform features, plus run-time build & packaging service support

**Build and Publish to App Stores:** Write once, deploy across many App stores and form factor devices

- Cloud-based build system simplifies packaging of Web and Hybrid built apps for various app stores
- Publish to many app stores, across many form factors easily
- Apple App Store
- Google Play
- Windows Store
- Tizen App Store
- Nook Store (Android)
- Amazon Store (Android)
- Chrome Store (web app)
- Facebook Store (web app)

### 2.1.2 Why Intel XDK?

With Intel XDK is easy to developers to check the look and feel of their apps with on-screen emulation on a wide variety of devices (sizes and resolutions). The App Tester allows you to test on a physical device.

It has free packaging services for HTML5 apps and Apache Cordova apps, for Android, Microsoft Windows, Apple iOS, Facebook, Amazon Kindle and other App Stores.

Intel® XDK streamlined interface is built on Web technologies HTML, CSS, JavaScript, and Node-Webkit back-end. It runs on Windows, OS X, and Ubuntu Linux, without browser or Java dependencies, and it is free.

- Easy-to-use: Streamlined workflow from design to app store
- Develop faster: Integrated design, test, and build tools
- Deploy simply: Across more app stores, and form factors

### 2.1.3 Other IDEs Options

There are plenty of IDEs that can be used for developing HTML5 applications. But only few of them will let you run and debug your application on mobile simulator as Intel XDK. Some of them are very confortable for developers and save you a lot of time. They also vary on price.

- Monaca: http://www.monaca.mobi/en/
- Eclipse: https://www.eclipse.org/
- NetBeans: https://netbeans.org/
- Adobe Dreamweaver CC: http://www.adobe.com/products/dreamweaver.html
- JetBrains WebStorm: http://www.jetbrains.com/webstorm/
- Aptana Studio: http://www.aptana.org/
- Komodo IDE: http://komodoide.com/
- Sublime Text: http://www.sublimetext.com/

### 2.1.4 Other Frameworks Options

Intel XDK supports Intel AppFramework, jQuery Mobile, Twitter Bootstrap. Currently there are plenty of options available. You can always import the best framework and use it in your project with Intel XDK.

The best option depends on the user familiarity with JavaScript and HTML5.

Here is listed some of them:

- Intel App Framework – http://app-framework-software.intel.com/
- jQuery Mobile – http://jquerymobile.com
- Kendo UI – http://www.kendoui.com/
- Twitter Bootstrap 3 – http://getbootstrap.com
- Sencha Touch – http://www.sencha.com/products/touch/
- TopCoat – http://topcoat.io/

The first three are the most used ones. Here is summarized some advantages and disadvantages of each of them. On Gajotres website[9] there is a complete comparison between them:

| HTML 5 Framework | Pros | Cons |
|---|---|---|
| Intel App Framework | 1) Fast and reliable;<br>2) Optimized for Android and iOS;<br>3) Small and light;<br>4) Excellent native wrapper controller;<br>5) Very easy to use;<br>6) Excellent theme support;<br>7) Has MVC support. | 1) Very bad documentation compared to other frameworks;<br>2) The framework is a rewrite of jQuery Framework, but there are missing things;<br>3) Does not support every HTML5 browser. |
| jQuery Mobile | 1) Most commonly used, which means a lot of third path information;<br>2) Extremely easy to use;<br>3) Good documentation;<br>4) Support every HTML5 browser;<br>5) Native look depending on mobile platform;<br>6) Many third party plugins. | 1) Sluggish on mobile devices (not optimized like Intel App Framework);<br>2) Not out of the box MVC Support;<br>3) Sluggish even more when combined with PhoneGap; |
| Kendo UI | 1) jQuery based framework, which also means a lot of third path information;<br>2) Complete package, framework, UI and MVC at once;<br>3) Great documentation;<br>4) Native look depending on mobile platform; | 1) Commercial license and support;<br>2) Lack of third party support, mainly due commercial license. |

*Table 1 – Frameworks comparison*

# 3 Preparing the Environment

## 3.1 Before get started

In this chapter the installation of Intel XDK will be shown with all steps that should be follow, since find it on the internet until have it installed on your computer.

### 3.1.1 Install Google Chrome Browser

One of the benefits of developing an HTML 5 application, is the possibility to use the same code for the mobile application and also the website application. We recommend downloading and installing Google Chrome Browser to emulate your website, and also check how it behaves with the css and javascript properties.

### 3.1.2 Install App Preview on your Device

In order to preview or install an application you developed on your device you must download an application called App Preview (from Intel) on your device. For that, use the official store of your device operating system. (Apple Store for iOS based devices, and Google Play for Android based devices).

After downloading and installing the application, on the first time you open it, you will be prompt whether you want to create an account, press "yes", and fill the data correctly. Later on we will synchronize your device with your computer to allow the simulation.

## 3.2 Downloading and Installing Intel XDK

### 3.2.1 Downloading

In order to download the Intel XDK program on your computer, it is necessary to follow the following instructions:

1. Open Intel XDK official website (http://xdk-software.intel.com). Right after you will come across the page shown on Figure 1:


*Figure 2 – Website for Dowloading Intel XDK*

2. Click on the blue button called Download Intel XDK;
3. A new page will appear, recognizing automatically your operating system and preparing the download, as shown in Figure 2 and after five seconds it will automatically start downloading:

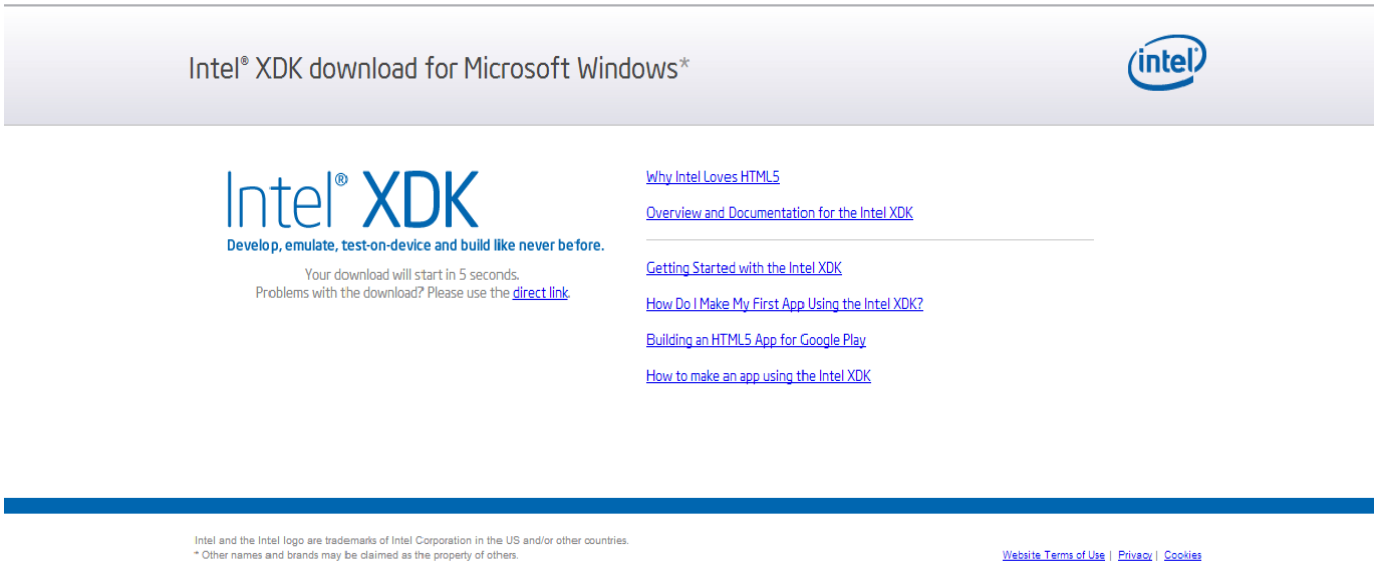


*Figure 3 - Dowload Preparation*

### 3.2.2 Installing

To install the Intel XDK successfully it is necessary to follow these steps:

1. Click on your completed download and the following window will be opened:


*Figure 4 – Openning Intel XDK*

2. After loading the necessary complements it will be ready to start. This is the Welcome page and the next step is to click on the button called Next;


*Figure 5 – Welcome Page of Intel XDK*

3. The Options page will be opened and now you can choose a destination folder by typing the place or by clicking on the button called Browse and choosing the right place, then click on the button called OK. When the destination folder is chosen, click on the button called Next;

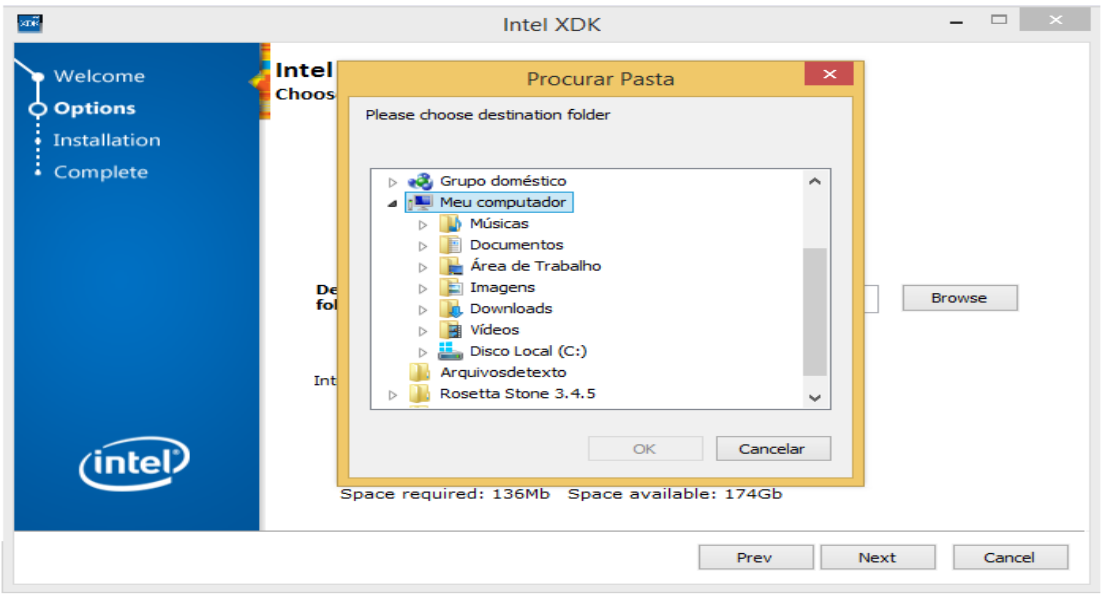


*Figure 7 – Choosing Destination Folder*

4. The following confirmation page will be opened, showing the destination folder(s) and the components that are going to be installed. Check the informations. If you decide to change the destination folder just click on the button called Prev and change the information, otherwise click on the button called Install:

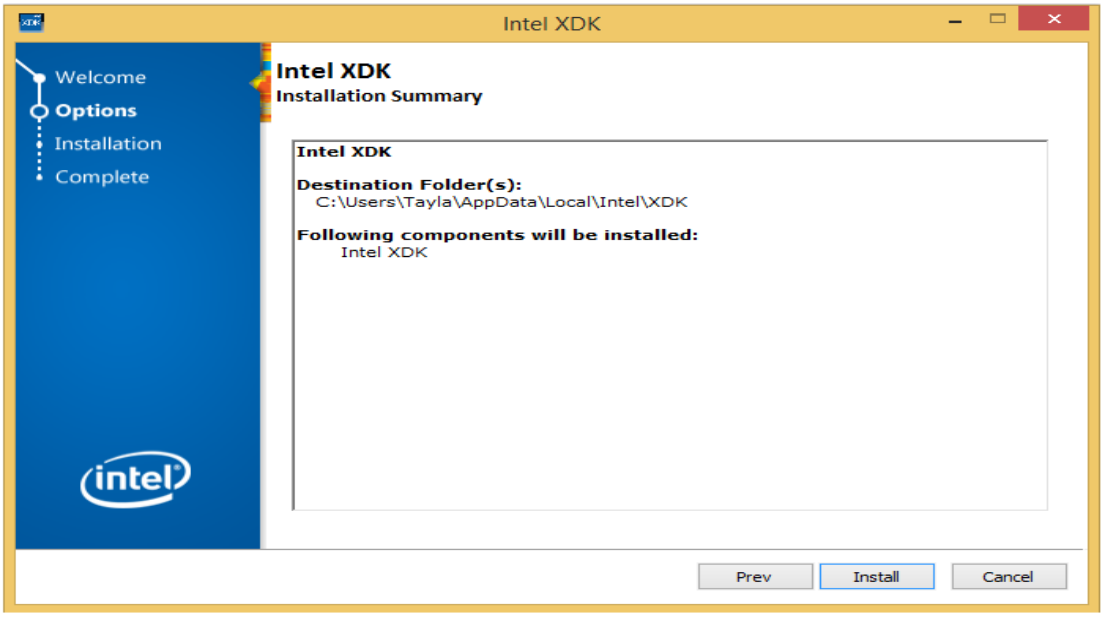


*Figure 6 – Confirmation of Destination Folder*

5. The Installation window will be opened and start installing:

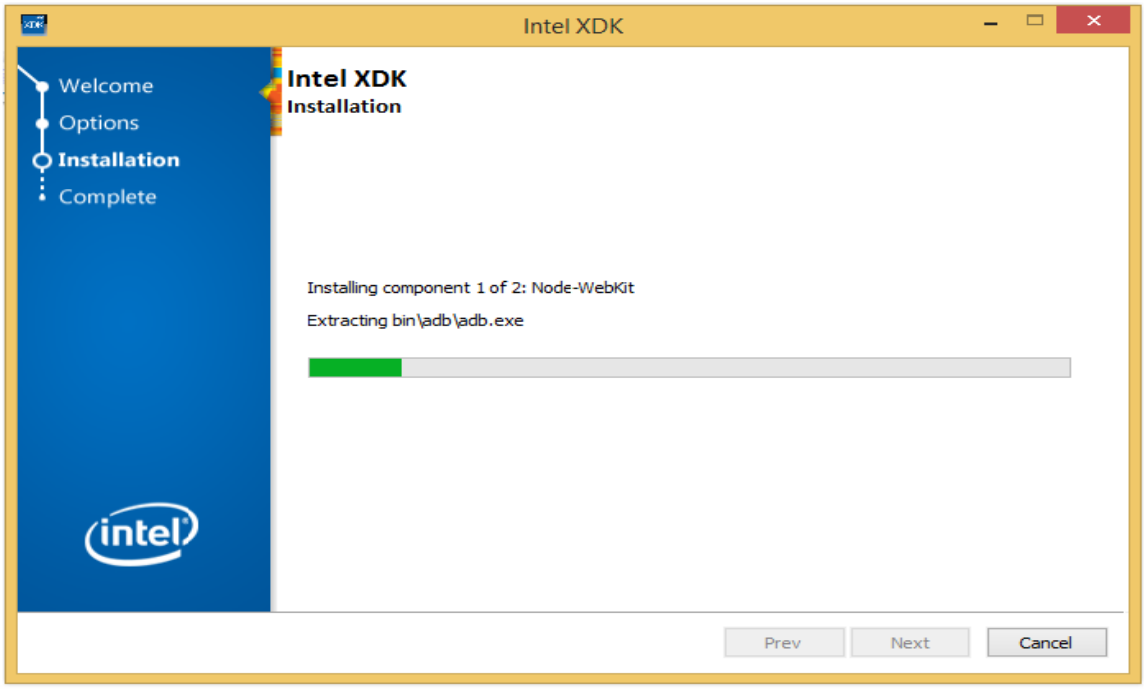


*Figure 8 – Start Installation*

6. After installing all the components, the Complete page will appear showing that the Intel XDK is now installed. When you are ready click on the button called Finish, after that the window will be closed.

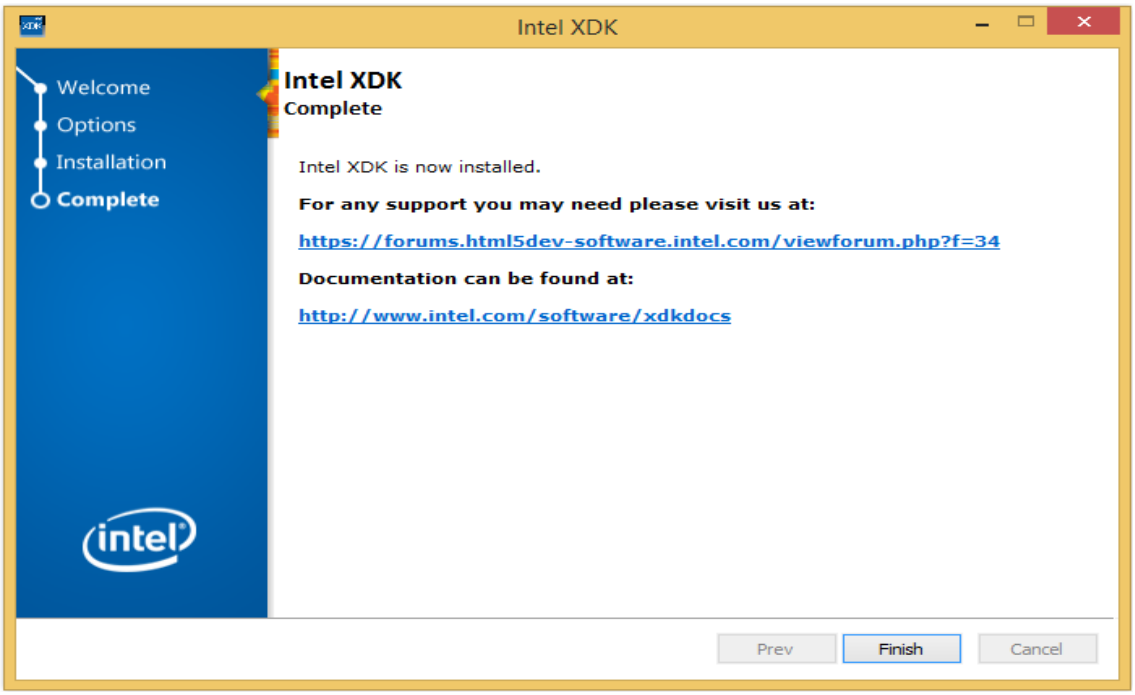


*Figure 9 – Completion of Intel XDK Intallation*

# 4 Getting Started

The purpose of this chapter is to present the Intel XDK IDE and to make your first application using it.

## 4.1 Presenting the XDK IDE

Intel XDK provides a training page which new users can get started with the XDK environment and learn some HTML5, CSS3 and JavaScript. There are plenty of videos and tutorials about each feature of Intel XDK. They are split into "HTML5 Overview Videos" and "Getting Started Videos":

https://software.intel.com/en-us/html5/training

## 4.2 Hello World

This part will show how to create, simulate and open in Browser an application. Follow the steps and enjoy it!

### 4.2.1 Creating the APP

1. When you open the Intel XDK, this first page will appear:

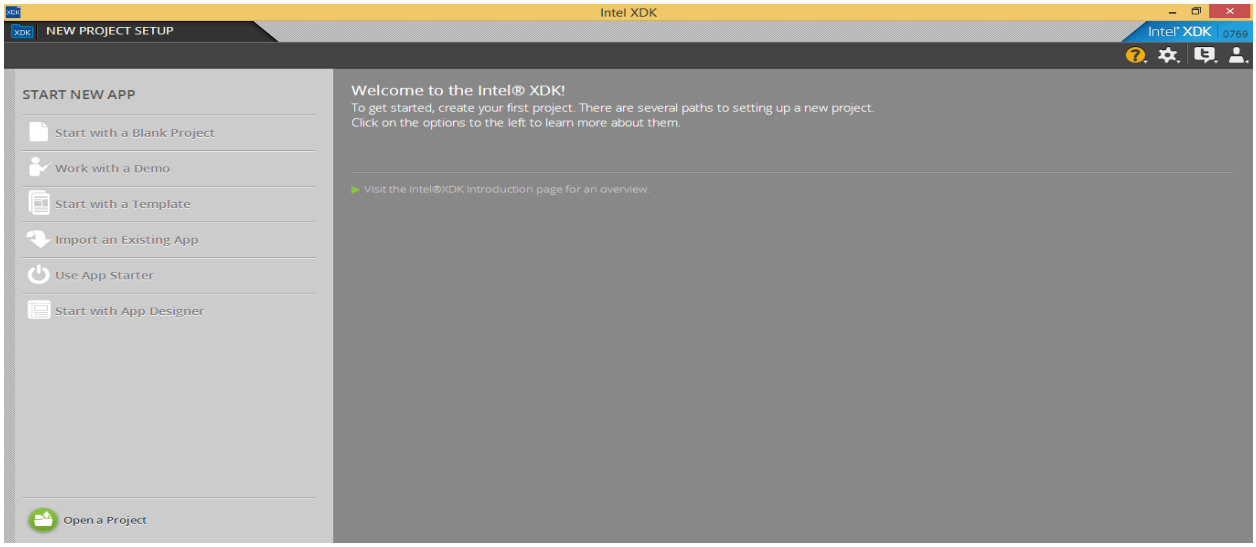


*Figure 10 – First Intel XDK Program Page*

Here you can choose the way you want to start your new application.

2. Now it is a good idea to logon your account, in order to do that you have to go on the right side of the program window click on the button called Account, near settings.

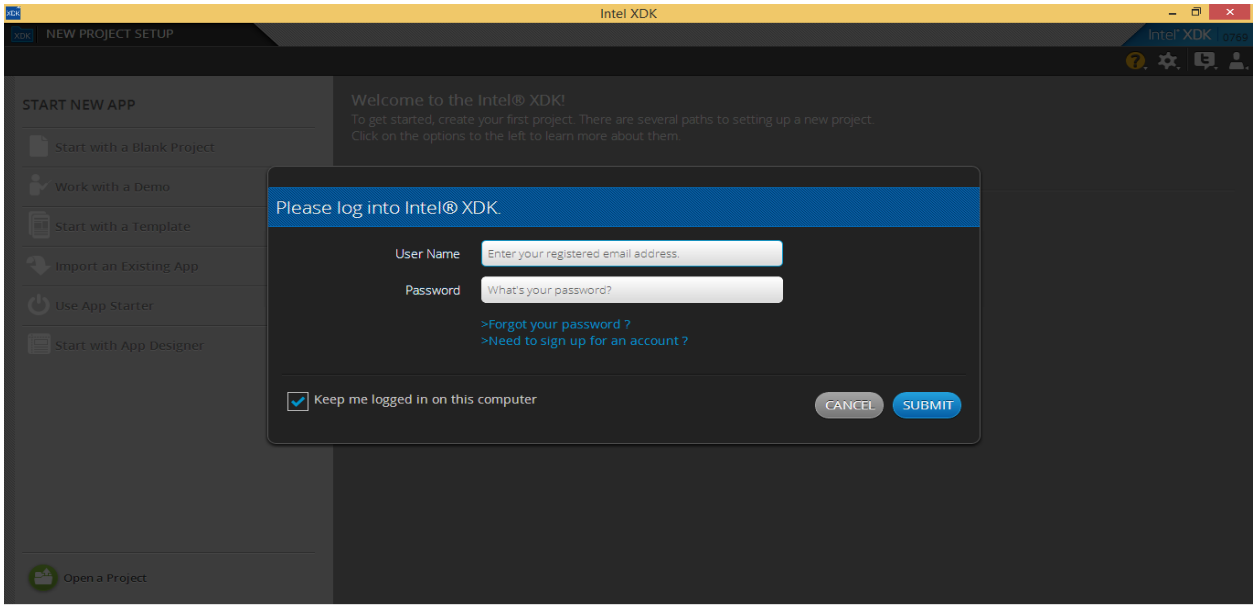


*Figure 11 – Login Box*

Enter with your user name and password you have already defined on your mobile, otherwise click on Need to sign up for an account and create an account. After that click on the button called SUBMIT.

3. You will choose the option called Use App Starter, so click on it:

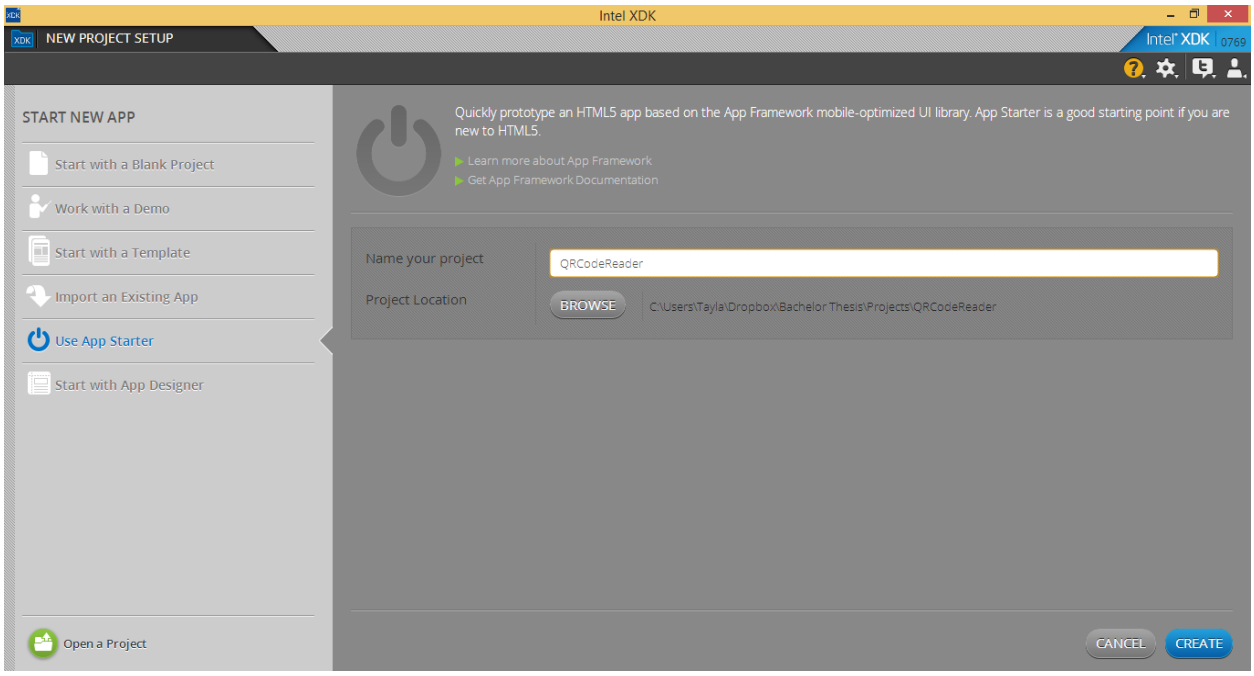


*Figure 12 - Use App Starter Page*

Now, name your project as QRCodeReader, and then select the project location by clicking on the button called BROWSE and selecting the right place. After that click on the button called CREATE.

4. At this point your project is created and the program offer you a quick tour where it will show some functions and the meaning of some components, which you are going to use thereafter. To accept that, click on the button called LET'S DO IT, otherwise click on the button called NO THANKS.

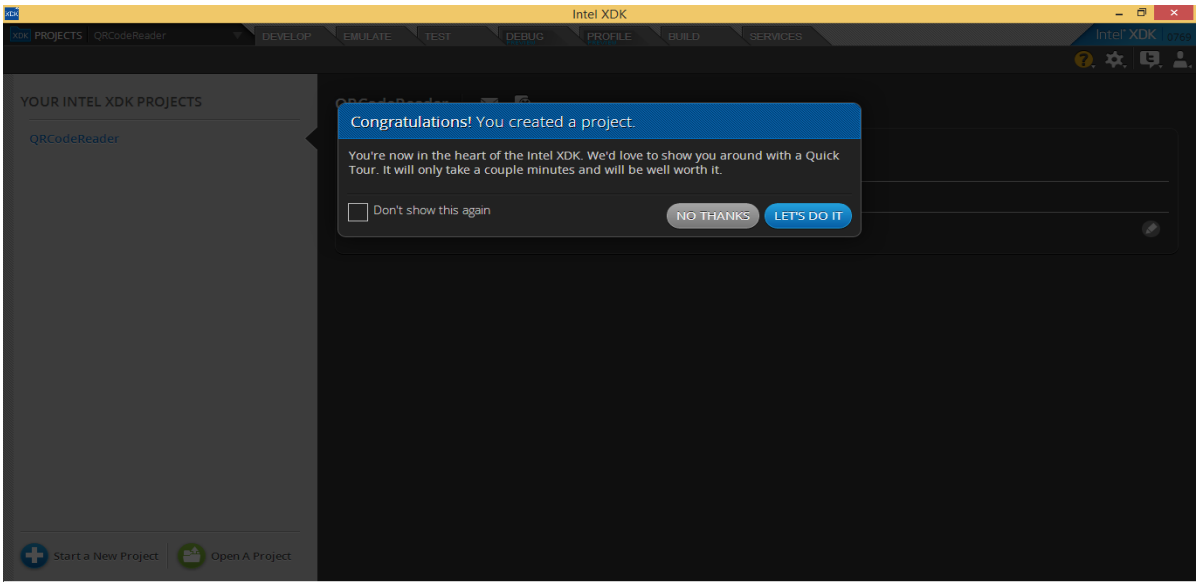


*Figure 12 – Confirmation of Project Creation*

5. So here, you can see a preview of your program with a button called Hello World.

If you click on Hello World, you can change some styles of it on the window called Button Styles that will appear on the right side of the program.

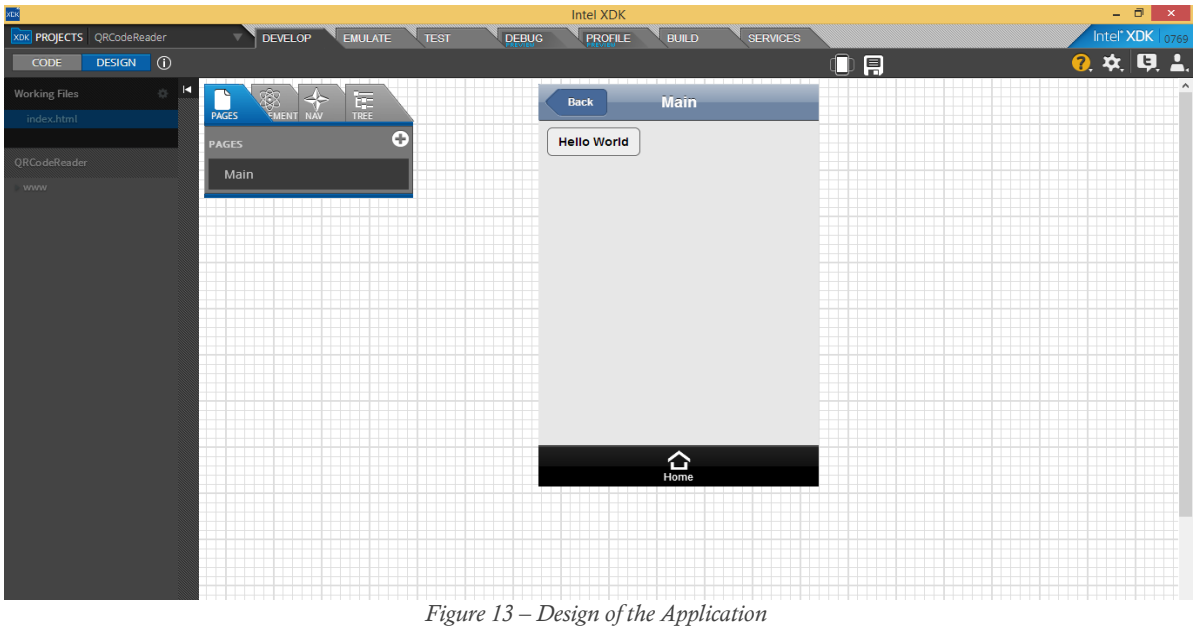


*Figure 13 – Design of the Application*

### 4.2.2 Simulating the APP

Now it is time to simulate it and see the results of your work. To do so, click on the tab called EMULATE and it will automatically simulate it for you.

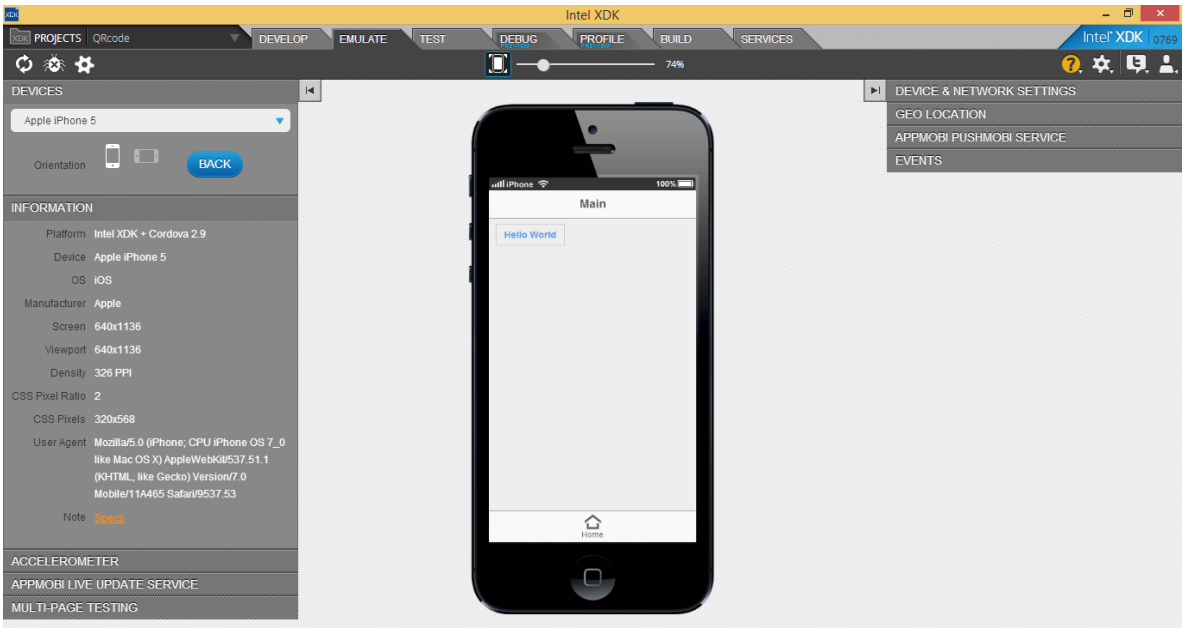


*Figure 13 – Emulate Page*

On the left side there is a button called DEVICES, there you can choose the device of your preference to simulate your app on it. Try it out!

### 4.2.3 Emulating on your Mobile

First of all, you have to click on the tab called TEST and select the option Run Intel XDK local server to enable test over Wi-Fi.

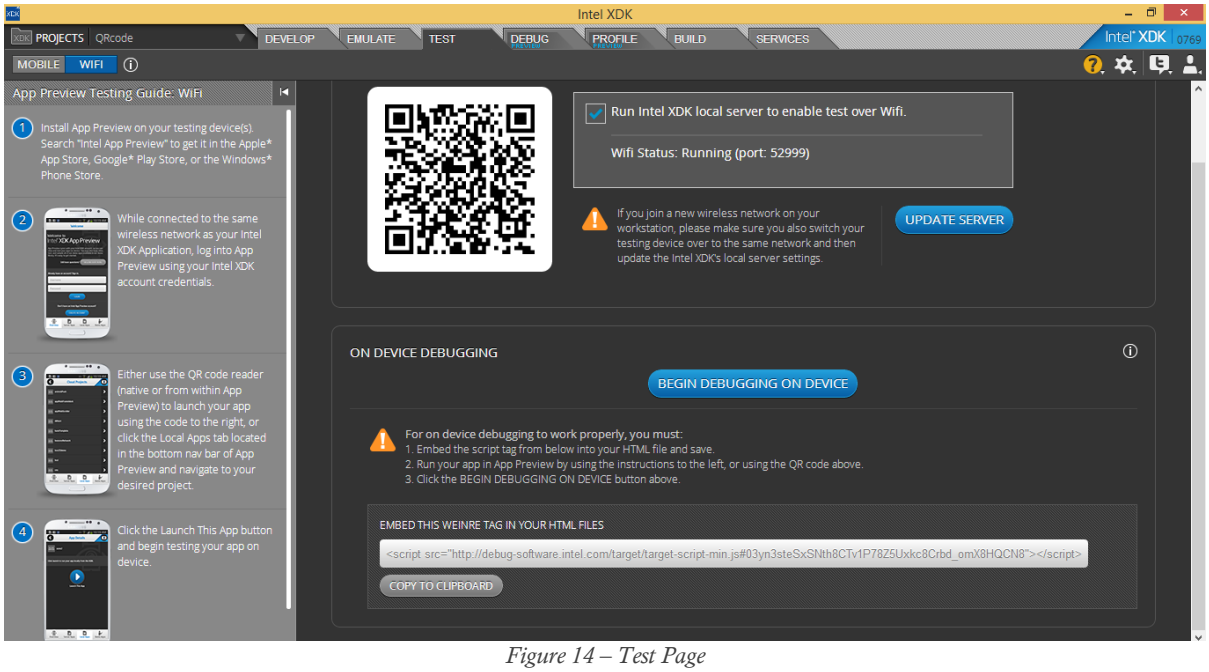


*Figure 14 – Test Page*

Now open the application App Preview on your mobile and click on the button called Local Apps (make sure your device and the Intel XDK are in the same wireless network).

See that it already recognized your Hello World program and by clicking on it, a blue button called Launch This App will appear, so click on it and then you will see the result of your program.

Now click on your mobile button to go back and wait for the next simulation.

## 4.3 QR Code

Here we will create an application that open the camera of a device and read a QR Code by editing the Hello World program. The source code of this program is available on the attachment 1.

### 4.3.1 Editing the last APP

1. Click on the tab called DEVELOP, then double click the button called Hello World that you created.

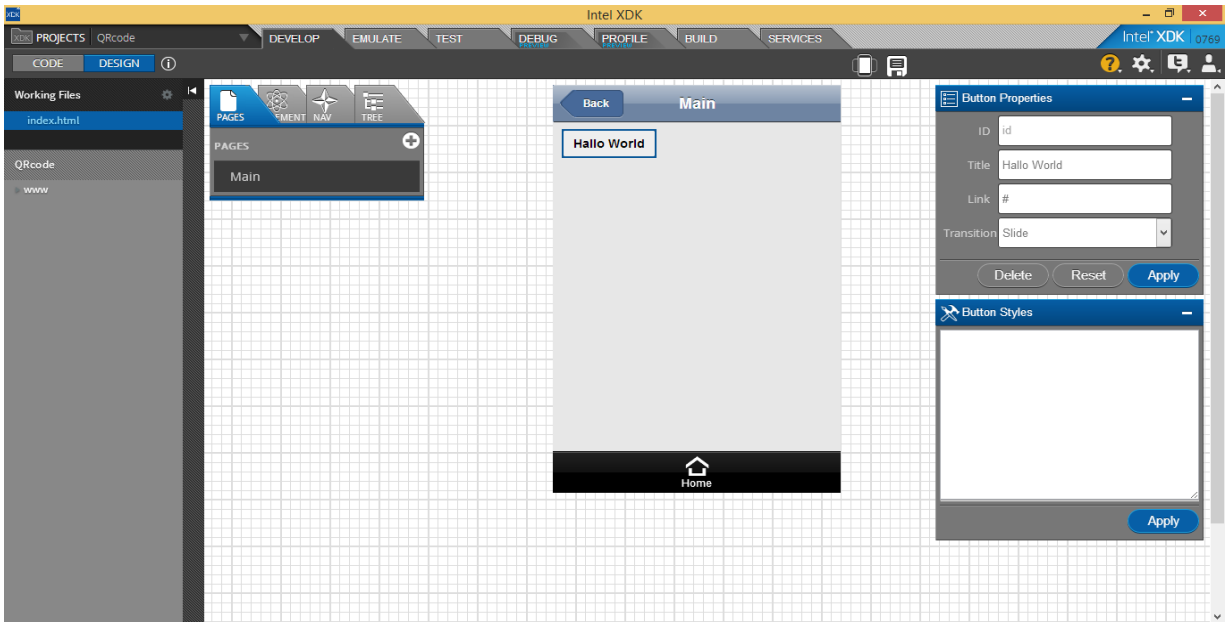


*Figure 15 – Button Edition*

Notice that two small windows appeared on the right side of the program window, please go there, click on the button called Delete and then confirm the action.

2. Click on the tab called ELEMENTS on the left side of the program, click, hold the button called Text (T) and drag it to the previewer (Main). Do the same with the button called Button (GO).

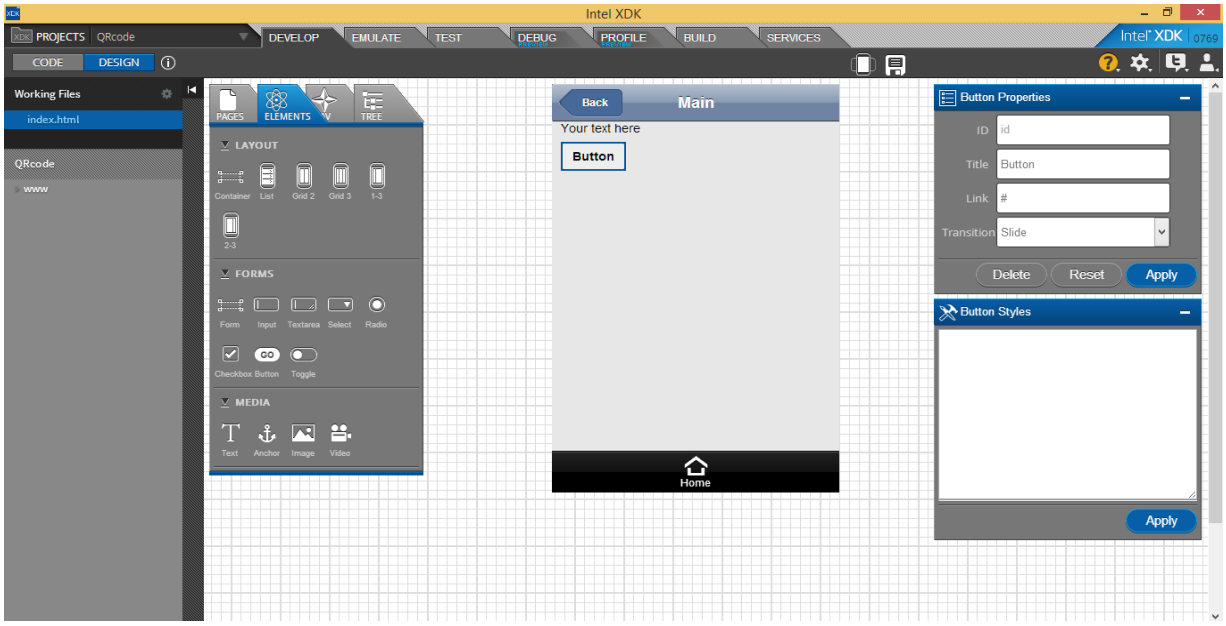


*Figure 16 – Elements Tab*

3. Now let's stylize them a little bit! Click on the first button called Your text here, go to the Text Properties window and rename it as in the followed image, then click on the button called Apply. After that go to the Text Styles window, add the respective styles and then click on the button called Apply.

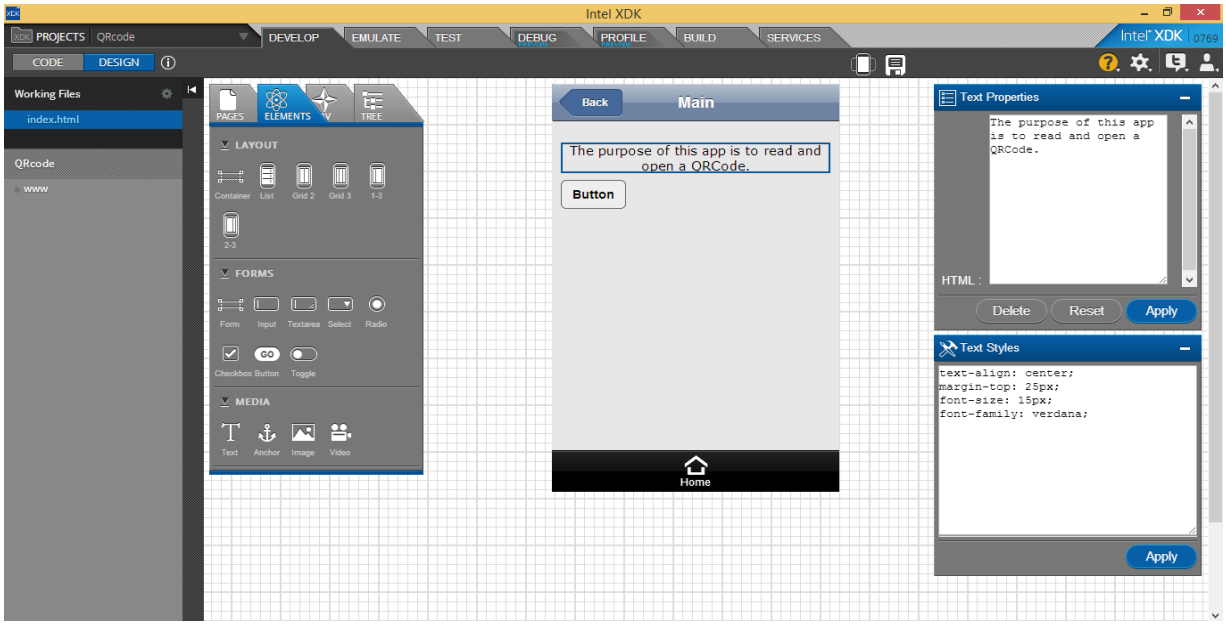


*Figure 17 – Button (Your text here) Edition*

4. Click on the second button called Button, go to the Button Properties window and rename it as in the followed image, then click on the button called Apply. After that go to the Button Styles window, add the respective styles and then click on the button called Apply.

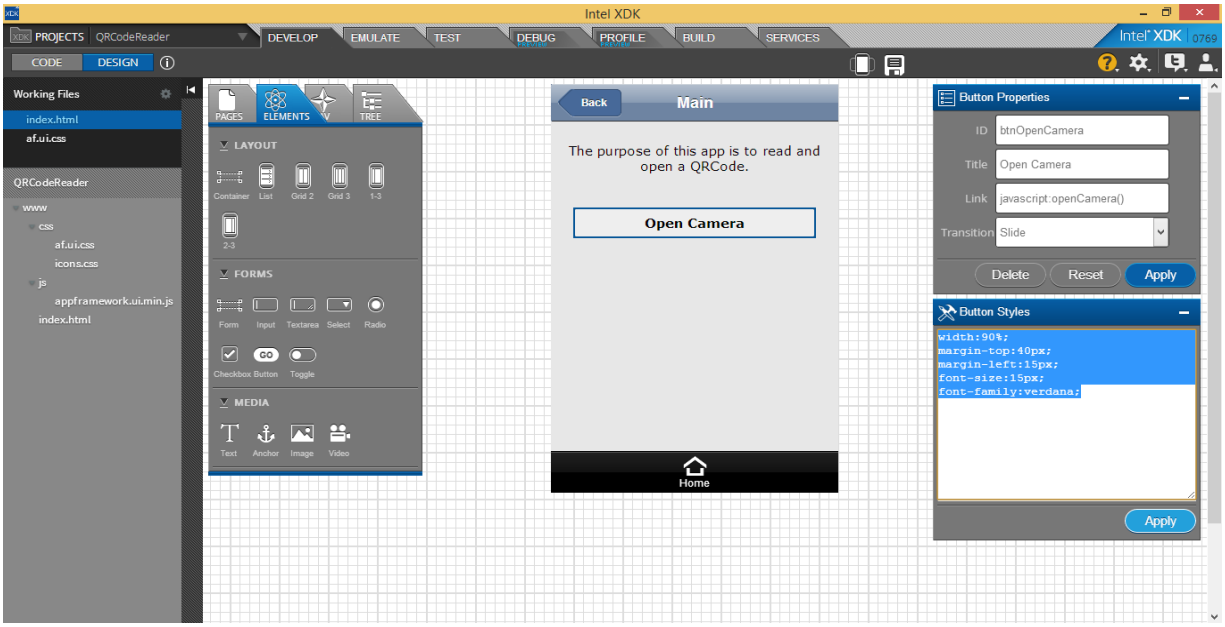


*Figure 18 - Button (Button) Edition*

It looks very good! However, you still have to open the camera and read a QRCode, so let's continue.

5. Go to the upper-left corner and click on the button called CODE, then add the following script below the last script and above the links on index.html:

```
1  <script>
2      function openCamera() {
3              intel.xdk.device.scanBarcode();
4          }
5  </script>
```

When this script is called, it is activated. In line 2 it is possible to see the creation of a function called openCamera and within it there is a scanBarcode method, which is a XDK method used to open a QR Code reader in the application.

6. After implementing the first script, it is necessary to create an event that will be listening the environment and when something happens with bar code reader, it is activated. For doing so, add the following event within the function called onDeviceReady, which is inside the second script on index.html:

```
1   //Adding a listener
2   document.addEventListener("intel.xdk.device.barcode.scan",
function(evt){
3         if (evt.success == true) {
4                 //successful scan
5                  intel.xdk.notification.beep(1);
6                  alert(evt.codedata);
7                  window.open(evt.codedata);
8          }
9           },false);
```

In line 2 it is possible to see the listener event, which means that when something happens with the bar code reader it is activated. Its designation is to verify with an "if" condition, as shown on line 3 whether the application captured a QR Code image successfully or not. If it was successful, it will make a beep sound, as shown on line 5, then it will show an alert message with the respective website obtained by the image, as shown on line 6 and finally it will open the page from the respective website, as shown on line 7. If the application does not capture the image, nothing happens.

### 4.3.2 Simulating the APP

Click on the tab called "EMULATE" and it will automatically simulate it for you as in the first program.

Once you selected the option Run Intel XDK local server to enable test over Wi-Fi on the Test tab you do not have to do it again. So let's open it on your device.

### 4.3.3 Emulating on your Mobile

Open the application App Preview on your mobile and click on the button called Local Apps as you did before.

It will recognize your QRCodeReader program and by clicking on it, a blue button called Launch This App will appear, so click on it and test your new app by clicking on the button called Open Camera and aim the camera in a QR Code, if it redirect you to the webpage it was supposed to, your program is working fine.

### 4.3.4 Generating APK for Android

How your app is working fine, it is time to create an APK for Android, so that you can open and visualize it, as it would really be if you publish in Google Play. To do so, it is necessary to follow the following steps:

1. Click on the Build tab and then in the Android square, click on the button Build:

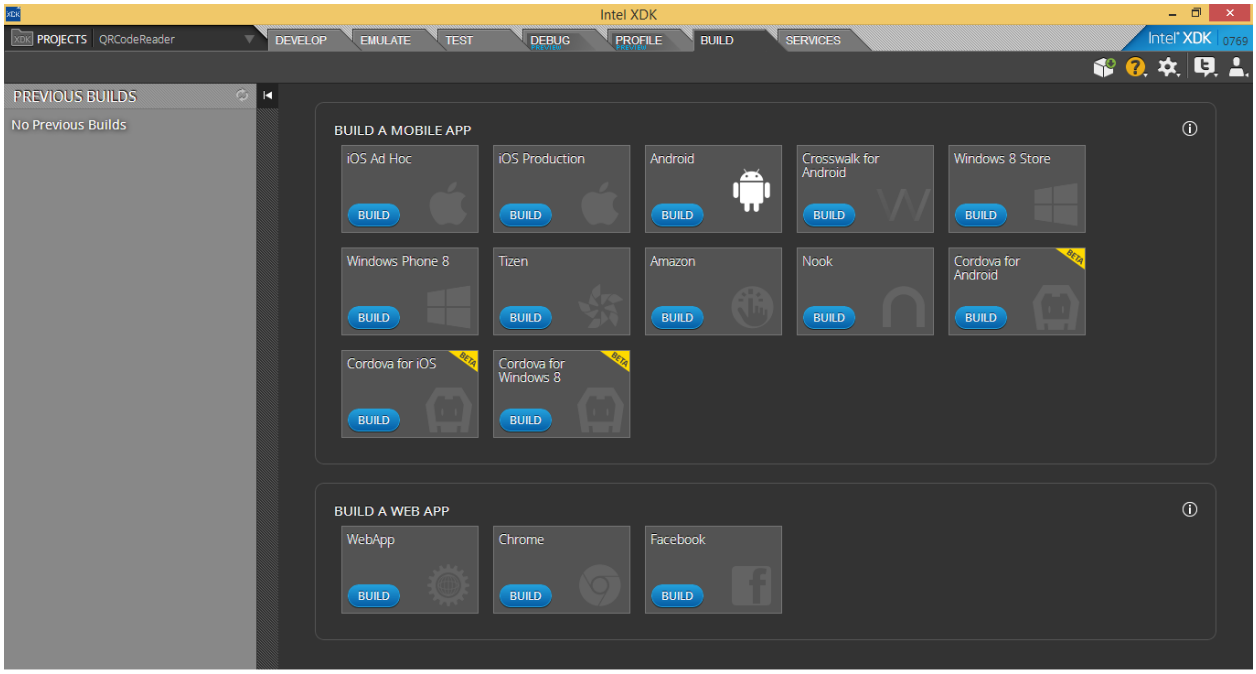


*Figure 20 – Build Page*

2. On this page you can see on the right side the informations about the Android Development Build. On the left side it is possible to see App details, Assets, Plugins, Credentials and Push, so if you have the need of changing the configuration of any of these options, click on the button called edit right in front of the respective option. Otherwise, if everything is right, click on the green button called Build App Now:

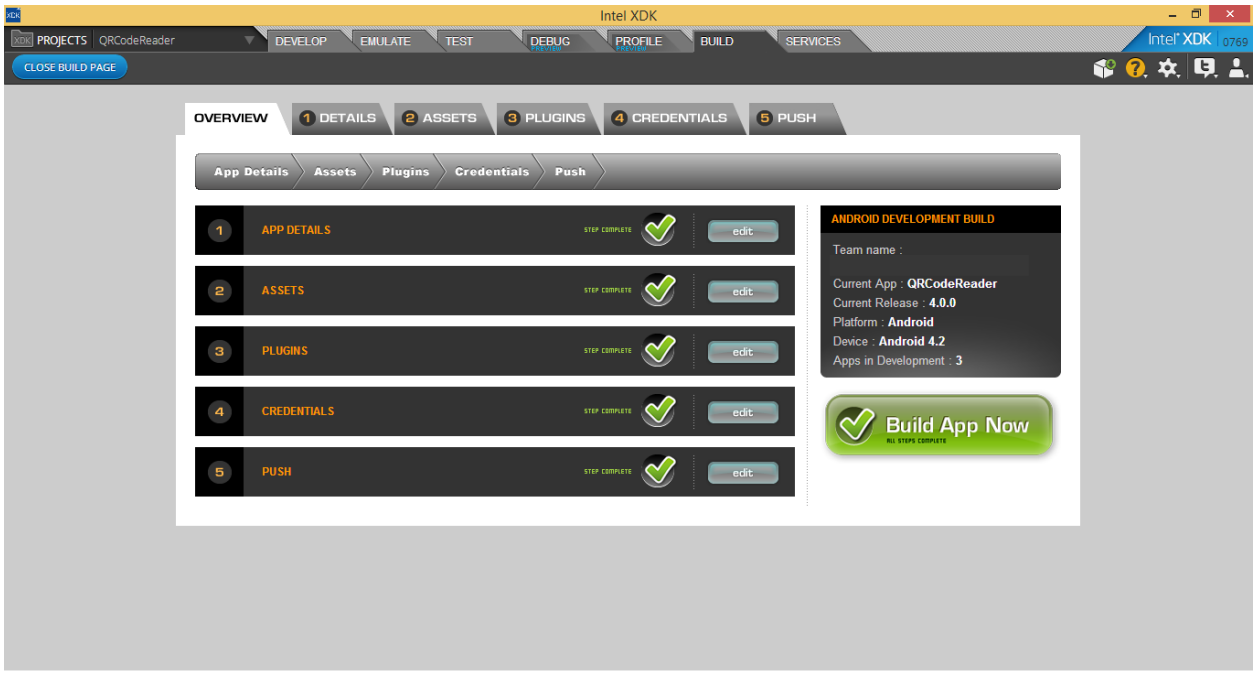


*Figure 21 – Build Overview*

3. It will take some minutes for building the application and when it finishes building, you will see a confirmation screen. In order to download your application on your computer, click on the green button called Download Build:

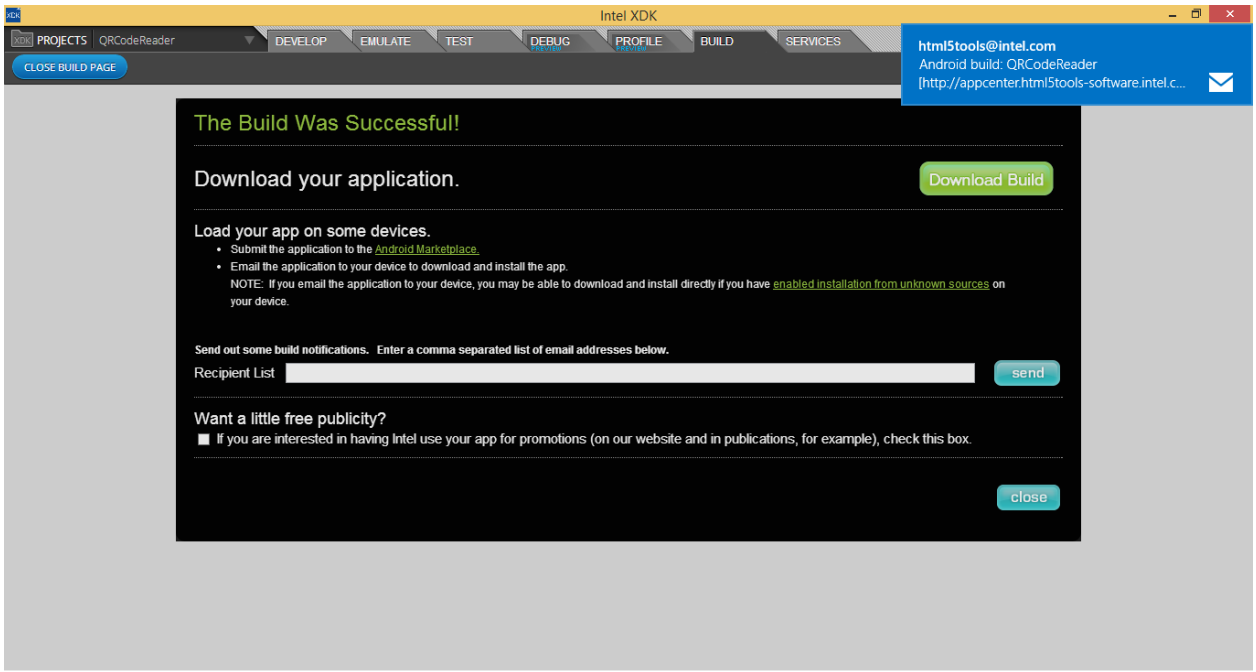


*Figure 22 – Build Confirmation*

You will receive an email as well, saying that the Android build for QRCodeReader was successful and you have to click on a link called "**here**" in order to download the APK on your device. After that the APK will be downloaded and when it completes you have to click on it. In some devices the Install can be blocked for security reasons, if it happens you have to go to setting and check Unknown sources box, which allow installation of apps from sources other than the Play Store. After doing that, a permission screen will appear, if you agree with all terms click on the button called Install.

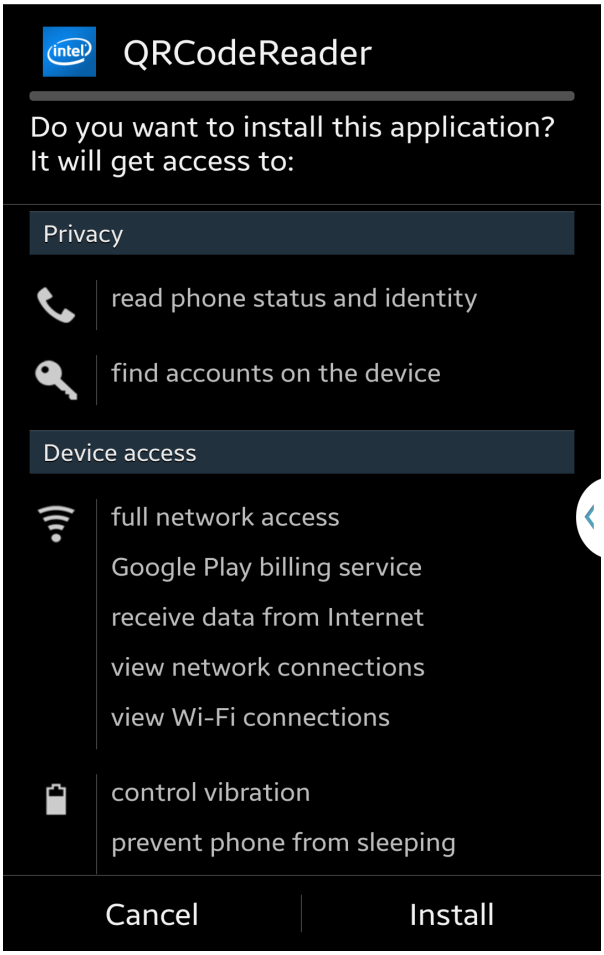


*Figure 23 – Device Permission Screen*

After that, your App will be installed and when it completes, click on the button called Open and then it will be opened and you can enjoy your Application, testing it and see the original result!

# 4.4 Working with a Demo

The source code of this program is available on the attachment 2.

## 4.4.1 Types of Demo

Intel XDK provides some Demos that can be seen in this page:
https://software.intel.com/en-us/html5/projects

The purpose of these demos is to facilitate the development of applications that uses one template that was already developed by someone else. One very useful demo is the "IQM TAB NAV", which uses tab bar for a tab-based navigation.

## 4.4.2 Selecting and Emulating a Demo

1. In your Intel XDK program select "Start a new Project", and choose the option "Work with a demo". Search for the demo "jQM Tab Nav" and choose it by click on the button called NEXT.

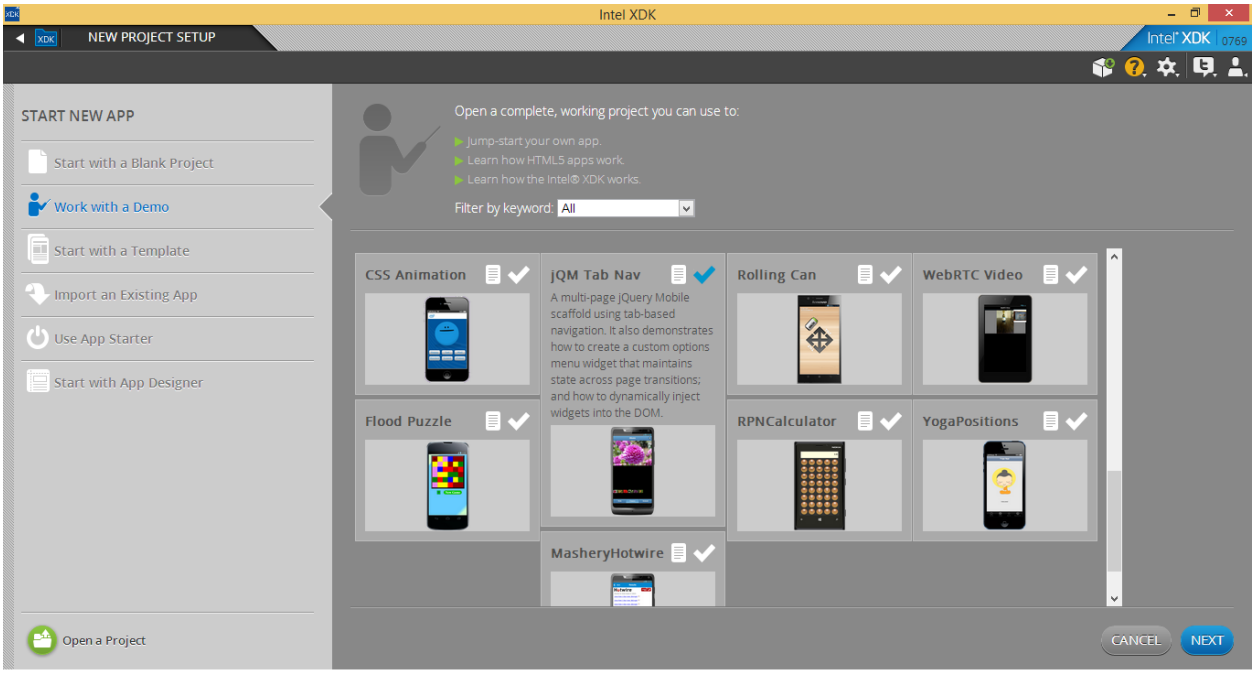


*Figure 24 – Work with a Demo Page*

2. Now name your project and click on the "CREATE" button.

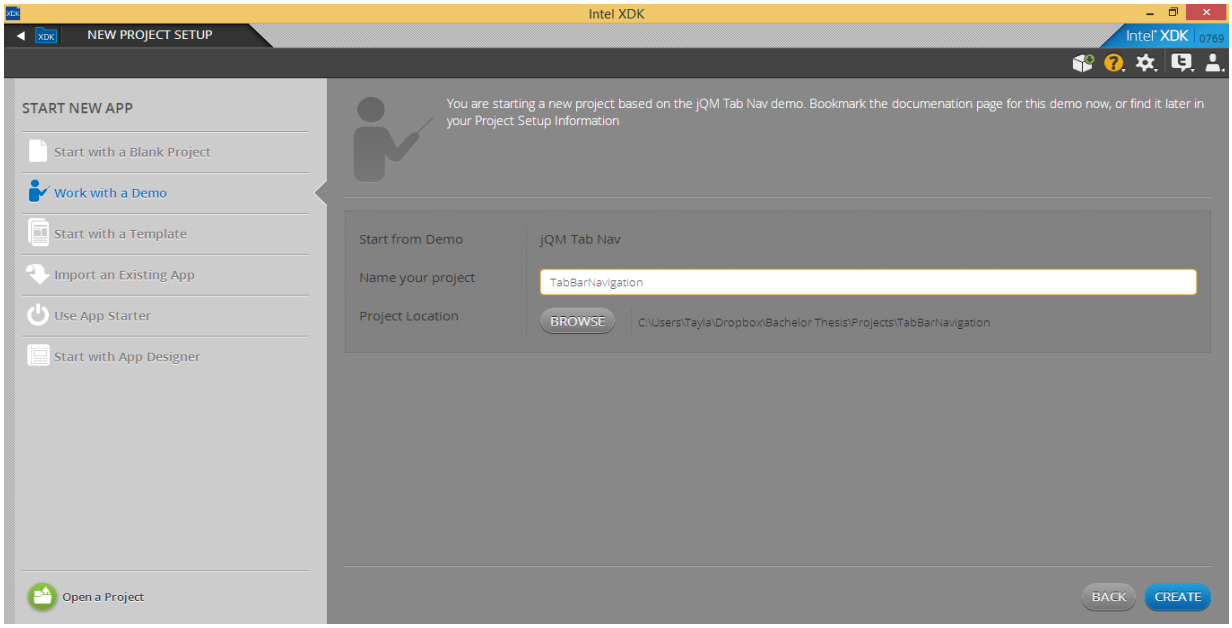


*Figure 25 – Project Nomination*

As you can see, there are already some code in your index.html and some JavaScripts and CSS documents as well.

3. Choose "emulate" and see the behavior of this demo. It is a simple tab-based application with the feature of choosing thumbnails images.

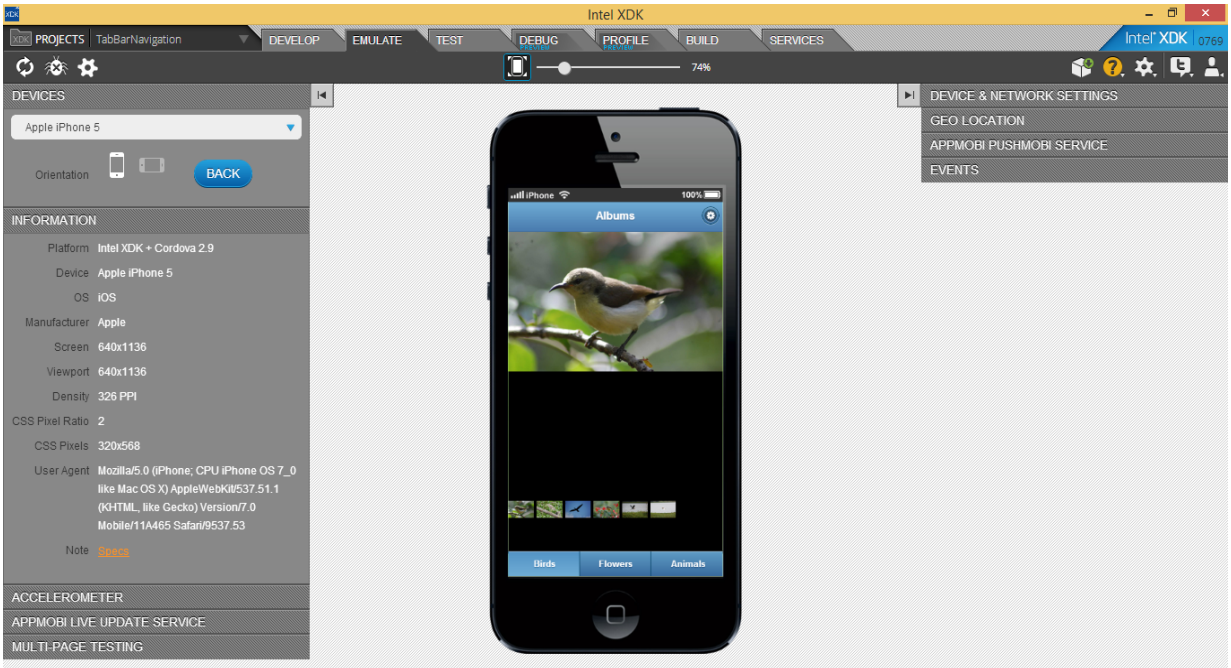


*Figure 26 – Emulate Page*

### 4.4.3 Understanding the Code

Taking a look at the source code, we can notice that there is only one page, called "index.html", as always. So how there are three tabs on the application? The answer to this question is on the source code.

We can notice that for each tab, there is one div that incorporates its contents, and is identified with the data-role="page". An id is also attributed for it in order to choose which page the user wants to see, for example:

```
1   <div data-role="page" id="animals">.
```

### 4.4.4 Creating a Tab and Taking Pictures

Let's edit the code of this template and add another tab with some content.

1. For doing so, it is necessary to add this code in **all** the <ul></ul> that are inside the navbar:

```
1   <li><a href="#camera" data-theme="b">Camera</a></li>
```

2. Now, after the third page (id="animals"), add the following code:

```
1   <!-- page 4  -->
2   <div data-role="page" id="camera">
3
4   <!-- header (same data-id for persistent toolbar) -->
5   <div data-role="header" data-id="tabnav-header" data-
    position="fixed" data-theme="b">
6           <h1>My Camera Page</h1>
7       </div>
8
9          <!-- content -->
10      <div data-role="content" class="content_div">
11
12          <div class="mainimage_div" data-role="content" data-
position="fixed">
13              <img id="camImage" class="mainimage"/>
14              <button onclick="javascript:takePicture()" data-
theme="b">Take Picture</button>
15              <button onclick="javascript:importPicture()" data-
```

```
theme="b">Select Picture</button>
16          </div>
17      </div>
18
19      <!-- footer -->
20      <div data-role="footer" data-id="tabnav-footer" data-
position="fixed">
21          <div data-role="navbar">
22              <ul>
23                  <li><a href="#birds" data-theme="b">Birds</a></li>
24                  <li><a href="#flowers" data-
theme="b">Flowers</a></li>
25                  <li><a href="#animals" data-
theme="b">Animals</a></li>
26                  <li><a href="#camera" data-theme="b" class="ui-btn-
active ui-state-persist">Camera</a></li>
27              </ul>
28          </div>
29      </div>
30
31  </div>
```

On line 2 it creates another page with `id="camera"`, which will be the page where we will be able to take pictures. On line 5 we are creating a tool bar, like the other pages, where it is possible to select the background color for the application.

On line 10 we attributed to the content the same class as the other page's contents. On line 12 we add a div that will keep the elements we are going to use on the screen. On line 13 it takes the main image, which means the last image we clicked on and fix it on the screen, like in the other pages. On lines 14 and 15 we added two buttons, the first one, when the user click on it, it will call a JavaScript function that will open the camera, so that the user can take pictures. The second button, when the user click on it, it will call another JavaScript function to import pictures that the user has already taken with the app or even pictures that are stored in his gallery. Both JavaScripts will be detailed explained in next two topics.

In the footer part, we defined divs that will allocate the navigation bar with the same style as in the other pages, as we can see on lines 20 and 21. On page 22 we created a list that will contain all the tab bars, just like the other pages; and each of them are links to its respective page.

3. Emulate it again. Now you can notice that a new Tab was created, and there are two buttons inside of it. But nothing happens when clicking on them. That happens, because there are no functions takePicture() and importPicture() defined yet. In order to do so, let's create a JavaScript document inside the folder called app (where all JavaScript documents are placed), so click on it with the right button of the mouse, select the option New File and name it as **myscript.js**:

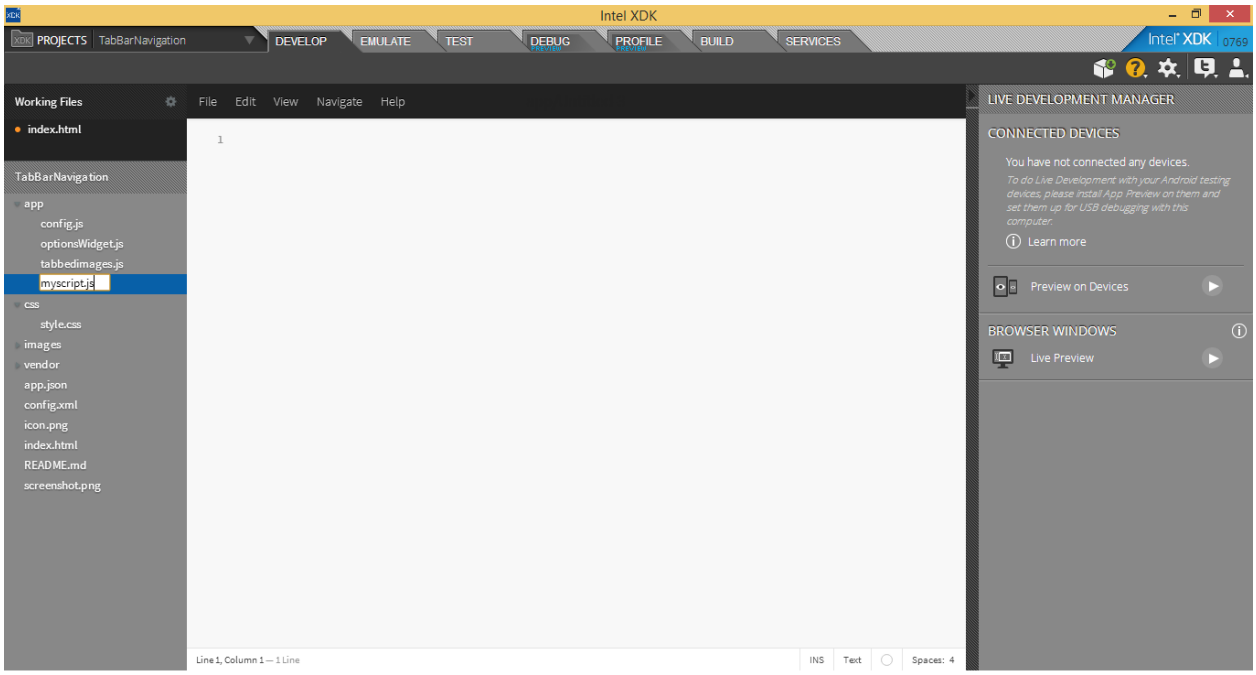


*Figure 27 – Adding JavaScript Element*

4. Inside of it, place the following code:

```
1     //Event listener for camera
2   document.addEventListener("intel.xdk.camera.picture.add",
   onSuccess);
3   document.addEventListener("intel.xdk.camera.picture.busy",
onSuccess);
4   document.addEventListener("intel.xdk.camera.picture.cancel",
onSuccess);
```

```
5
6
7   function onSuccess(event) {
8
9       if (event.success === true) {
10          var imagesrc =
intel.xdk.camera.getPictureURL(event.filename);
11          var inputImg = document.getElementById('camImage');
12          inputImg.src = imagesrc;
13      } else {
14          if (event.message !== undefined) {
15              alert(event.message);
16          } else {
17              alert("Error capturing picture!");
18          }
19      }
20   }
21
22  function takePicture() {
22      intel.xdk.camera.takePicture(80, true, "jpg");
23  }
24
25  function importPicture(){
26     intel.xdk.camera.importPicture();
27  }
```

Let's understand the code. First we defined some events handled by Intel XDK, as shown on lines 2, 3 and 4, and when they happen, the method "onSuccess" will be called. After that, we defined the function "onSuccess", as shown on line 7, and it has the parameter "event", which is a kind of struct that contains the property called "success", "message" and "filename". Then we checked if the event was successful, as shown on line 9, if positive we can retrieve the image from the camera by using the XDK method "`intel.xdk.camera.getPictureURL`", as shown on line 10. After that, we searched for the img component we defined on the index.html as "camImage", as shown on line 11, and set its source to the source of the picture returned by the event, as shown on line 12.

If the event was not successful, it checks whether the error is undefined or not, if negative it shows an event error message, which comes from XDK. If positive, it displays a default message.

We also defined two functions, the first one is called takePicture and it uses the method from Intel XDK to take a picture:

```
1   intel.xdk.camera.takePicture(80, true, "jpg");
```

The first argument is the quality, the second one is if you want the taken picture to be saved on the library and the third one is the picture type.

For importing a picture from the library, we only need to use a method without parameter:

```
1   intel.xdk.camera.importPicture();
```

Now, all we have to do in order to make it work is to reference the JavaScript document into the index.html file. It can be done with the following two lines; a good idea is to place them with the other scripts in the head:

```
1   <script type="text/javascript" src="app/myscript.js"></script>
2   <script src="intelxdk.js"></script>
```

Done! Now you can emulate and test it on your device!

# 5 Complete Application Development

Normally, a fully application does not run by itself. Either it retrieves data from a Web Service or it sends data to somewhere. During this chapter we will create and understand a fully working application. The chosen topic was a service for the International Office of universities, where the office has one webpage with the option to create warnings, such as informations of events or concerning to the studies. The students with their device are able to see all these informations and filter them by the city.

A database is necessary to store the informations, created in the webpage and retain it.

A Web Service is necessary to connect to the database and exhibit the data for the device. The format chosen was jSON.

## 5.1 Database Structure

The relational database management system (RDBMS) chosen was MySQL, because it is easy to handle and to support. There are several Interfaces for drawing structure and managing the data contained with the database, such as MySQL Workbench, for instance.

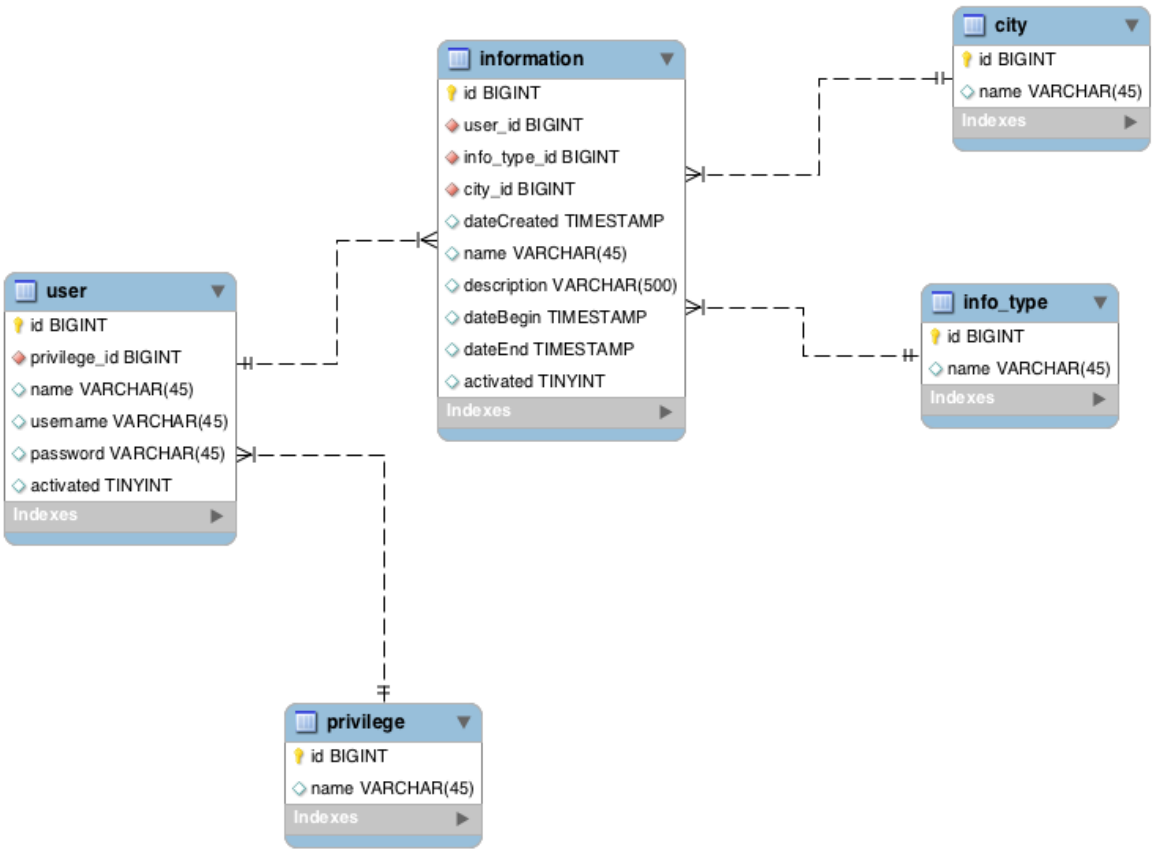


*Figure 28 – Database Structure*

The structure developed for the application was the one as shown on figure 28.

The create statement for such a database can be found in the Appendix.

For understanding this image, some SQL knowledge must be known. First of all there are 5 tables, these being **user**, **privilege**, **information**, **info_type** and **city**. Let's take a look on privilege table first. This table contains 2 attributes, being them an ID, which is a unique identifier for each privilege contained in this table. Therefore it is a BIGINT, marked as PK (primary key), NN (not null), AI (auto increment). The other attribute is a name, marked as a VARCHAR(45), meaning it contains text as long as 45 characters.

Two privileges are used in this application, the **Grant All** and the **Reduced** privileges. The Reduced means that the user does not have access for everything; in the application context it means that he is only able create and edit informations. The Grant All means that the user has access to everything, meaning that he can add/edit/remove other users, cities and types of informations.

Now let's take a look on our user table, and understand how does it works. In the user table, there is the ID- the primary key (unique identifier for each user) as well, but also has a privilege_id. This privilege_id means that user and privilege are related somehow. In this case, this relation is called ManyToOne, meaning that one privilege can have many users. The user have also other attributes, such as name, username, password, and a activated flag that represents whether the user is active or not. This is very useful, because you can mark a user as not activated without deleting it.

The password stored in the database is an hash of the user real password. This is a security feature, for making sure that if an attacker successfully get into the database, he/she will not be able to read the passwords in plain text.

## 5.2 Web Service

For the web service, Java Web Application was chosen to handle the request and connect to the database. We used hibernate to handle the connection with the database.

As known, java is an object-oriented language; and in well-modeled object-oriented applications there is a good separation between the responsibilities of each part of the application.

We chose to develop it in different and independent layers: Presentation, Business (service), Data Access Object (DAO), and Domain. The structure is as in the following image:

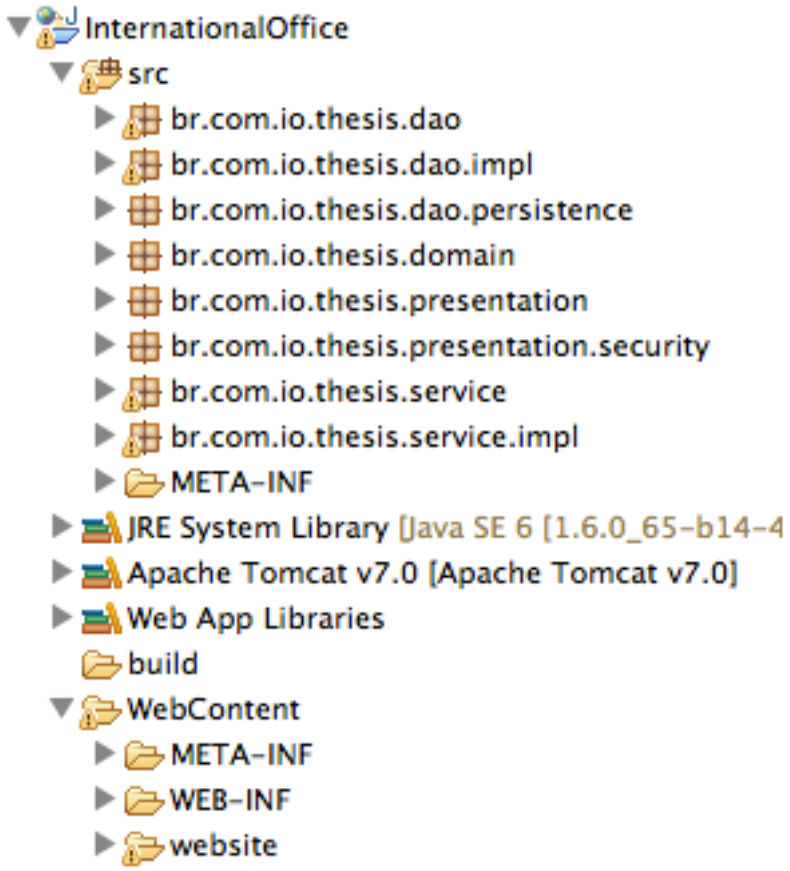


*Figure 29 – Project Structure*

### 5.2.1 Data Access Object (DAO) Layer

The data access layer or the persistence layer is one of the most interesting. It is the application part responsible for communicating with the database, or the persistence framework, which means that it is responsible for extract, insert and update informations. It is also responsible for transforming models in Object Relational models, such ORM (Object Relational Mapping), since in many cases we deal with relational databases.

**Interface DAO**: It is a good idea to specify DAO through an interface instead of a concrete class. Thus, not only the work to implement new mechanisms for persistence are facilitated, but also, we prevent that create references to concrete classes, reducing the coupling to a particular mechanism.

Another great benefit of using interfaces is the fact that it is known exactly what methods are available in our implementation without actually knowing how or how these methods will be available.

As explained earlier, the main objective of the data access layer is to abstract access to information, so the proposal should contain interface methods to create, read, update and delete (CRUD) information in the database.

When starting any project, the first step is to think which entities (objects) should be stored and what information will be stored within each one. In a real life application, these entities correspond to the tables of the database; in our case they correspond to **user**, **information**, **privilege**, **city** and **info_type** tables.

For example, on our user interface, the goal is storing data of a user. By observing our table, we can notice that it must store the following informations: **id**, **privilege_id**, **name**, **username**, **password** and **activated**. By modeling this information we have the following result.

```
1  public class User {
2
3    private Long id;
4    private String name;
5    private String username;
6    private String password;
7    private Privilege privilege;
8    private Boolean activated;
9
10   //Constructors gets and sets suppressed.
11  }
```

Of course, there is also a Privilege class as well as, **Information**, **City** and **InfoType**, containing their own attributes.

The above class is responsible for encapsulating the data in an entity, in this case "User". Every class that encapsulates any type of information should contain methods (gets and sets) in order to recovery and update the encapsulated informations. To facilitate the visualization, these methods have been suppressed, but are available in the Appendix, which contains the full source code of the project.

With our entity-class ready, we can now develop our DAO interface. Recalling that the first step is to build the interface and soon after its implementation.

As all the entities contains the CRUD methods, we created a so called GenericDAO, which is responsible to provide these functions:

```
1  public interface GenericDAO<T> {
2
3    public void insert(T entity);
4    public void delete(T entity);
5    public T update(T entity);
6    public T findByID(Long entityID);
7    public List<T> findAll();
8  }
```

So, in our user interface, we need only to add the necessary methods, that are exclusive for the User, which in our case is finding the user by its login.

```
1  public interface UserDAO extends GenericDAO<User>{
2    public User findByLogin (User usr);
3  }
```

The interface determines which methods should be implemented in the DAO layer and which methods are available to the business layer.

**Hibernate DAO:** Hibernate is a framework for mapping relational objects that facilitates the mapping of entities, their attributes and relationships. The goal of Hibernate is to decrease the complexity between the Java-based object model that need to work with a database programs. In particular, the development of queries and data updates. Its main feature is the transformation of Java classes for data tables (and Java data types to SQL). Hibernate assembles the SQL calls and relieves the developer from manual work converting the resulting data, keeping the portable to any SQL database program. Hibernate assembles the SQL calls and relieves the developer from manual work converting the resulting data, keeping the portable to any SQL database program.

For generating a series of facilities and resources, Hibernate is not a small project. In addition to the Hibernate own library, a series of libraries that form the auxiliary dependencies of the solution will be required. Below there is a list of the files required:

- **hibernate-3.6.10.jar** [14]: Basic hibernate library;
- **antlr-2.7.6.jar** [15]: Build tool;
- **commons-collections-3.1.jar** [16]: Basic library collections;
- **dom4j-1.5.2.jar** [17]: XML parser;
- **javassist-3.12.0.GA.jar** [18]: Assistant handling of byte codes;
- **jta-1.1.jar** [19]: Java Transaction API;
- **slf4j-api-1.5.8.jar** [20]: Abstraction layer logs;
- **slf4j-log4j12-1.7.2.jar** [21]: Integrating logs to log4j;
- **log4j-1.2.17.jar** [22]: Helper library for generating logs.

As we are using MySQL, the MySQL connector jar is also required:

- **mysql-connector-java-3.0.16-ga-bin.jar** [23]: basic library for connecting with MySQL.

In a Java Web Project, all these libraries must be on the /WebContent/WEB-INF/lib folder.

**JPA DAO:** Java Persistence API (JPA), is a standard Java API for data persistence. JPA defines a way of object-relational mapping for Java objects, called entity beans.

There are several implementations in the market following specifications JPA. We can cite Hibernate, OpenJPA, EclipseLink, among others.

Although Hibernate count on a complete solution for data persistence, his group of developers have implemented all the JPA layer, thus making Hibernate also a JPA resources-provider, known as Providers. By using Hibernate just as a JPA Provider, we gain in the future not only the freedom to change database, but also the freedom to change the JPA implementation's provider, without jeopardy the codes that were already developed.

To add JPA support to hibernate, one library is required:

- **hibernate-jpa-2.0-api-1.0.1.Final.jar** [24].

Much of the JPA was based on concepts created by Hibernate, so the way you work and required settings structure are quite similar.

The first step in the implementation of our project is to inform the JPA the entities that we persist, and what attributes each one carries within itself. With the inclusion of the Annotations feature in the Java platform since its 1.5 version, this procedure has become much easier. In many cases the use of Annotations avoids creating configuration files, especially XML files. To the programmer, the configuration becomes much more transparent since it can be done directly in the source code of the application. Annotations are easily identified because they are always preceded with the sign "@".

For mapping of our organization, we made use of this feature instead of using XML as follows below:

```
1   @Entity
2   @Table(name = "user")
3   public class User {

4   @Id
5   @GeneratedValue(strategy = GenerationType.AUTO)
6   private Long id;
7
8   @Column(name = "name")
9   private String name;
10
11  @Column(name = "username")
12  private String username;
13
14  @Column(name = "password")
15  private String password;
16
17  @ManyToOne
18  @JoinColumn(name = "priviledge_id")
19  private Priviledge priviledge;
20
21  @Column(name="activated", columnDefinition="tinyint")
22  private Boolean activated;
23
24  //Constructors gets and sets suppressed.
25  }
```

As we can see, the use of Annotations is quite transparent to the developer and simplifies day-to-day of it.

JPA will use these Annotations to generate the necessary settings, just as could be done with an XML file. Although you can configure the entities directly in the source code of the application, the configuration of the database information in an XML file is still required.

By default the file must be in the /src/META-INF/ folder, labeled with the name of **persistence.xml**, as we see below:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2
3   <persistence version="2.0"
4   xmlns="http://java.sun.com/xml/ns/persistence"
5   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
7
8       <persistence-unit name="InternationalOfficePU" transaction-
type="RESOURCE_LOCAL">
9
10          <provider>org.hibernate.ejb.HibernatePersistence</provider>
11
12          <class>br.com.io.thesis.domain.Information</class>
13          <class>br.com.io.thesis.domain.User</class>
14          <class>br.com.io.thesis.domain.InfoType</class>
15          <class>br.com.io.thesis.domain.City</class>
16          <class>br.com.io.thesis.domain.Privilege</class>
17
18          <exclude-unlisted-classes>true</exclude-unlisted-classes>
19
20          <properties>
21
22              <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://127.0.0.1:3306/internationalOffice" />
23              <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver" />
24              <property name="javax.persistence.jdbc.user"
```

```
value="root" />
25              <property name="javax.persistence.jdbc.password"
value="addon9rozMYSQL" />
26
27              <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect" />
28              <property name="hibernate.connection.shutdown"
value="true" />
29               <property name="hibernate.hbm2ddl.auto"
value="validate" />
30              <property name="hibernate.show_sql" value="true" />
31              <property name="hibernate.format_sql" value="false"/>
32
33          </properties>
34
35      </persistence-unit>
36
37  </persistence>
```

At this point it already understandable the great advantage of JPA. The mapping of the object does not depend on our database. In future, if any change in the database supplier is required, no change in the configuration of the entity, as well as in our project source will be required. The entire interface between our project and the database is resolved by Hibernate/JPA.

Fully configured with JPA using Hibernate as Provider, now it is possible to implement the DAO classes using the features of the framework. More implementation details can be found in the source code of the project, which is on the Appendix.

Therefore, we conclude that:

1) The data access layer must disregard the way the data are obtained
2) The use of interfaces facilitates the integration between the layers
3) Changes in the data access layer should not affect the other layers
4) The data access layer should provide resources to change and query data

**Transactions:** Transaction is a basic concept of any database system. The transaction's essential point is the comprising of several steps into a single operation of "all or nothing".

The intermediate states between the steps are not seen by other concurrent transactions, and if a fault that prevent the transaction to reach its end occurs, then, none of the intermediate steps will affect the database. The integrity of a transaction depends on four properties, known as ACID (Atomicity, Consistency, Isolation and Durability).

All of them are important, but let's take a look at the atomicity property:

**Atomicity:** All actions that make part of the transaction must be successfully completed in order to be effective. If during the transaction any action fails, the entire transaction must be undone (rolled back). When all actions are performed successfully, the transaction can be made and kept at the bank (commit).

As we can see, transactions are extremely important to maintain the integrity of the stored data. The transaction is a functionality provided by the database, therefore, it should be noted whether it supports this technology.

To provide control of the transactions in terms of software, the JPA provides some methods so that our software has control over transactions. The transaction object can be recovered within the **EntityManager**.

```
1   EntityTransaction transaction = null;
2
3   EntityManagerFactory emf =
Persistence.createEntityManagerFactory("InternationalOfficePU");
4
5   EntityManager em = emf.createEntityManager();
6   try {
7     em.clear();
8     transaction = em.getTransaction();
9     transaction.begin();
10    em.persist(obj);
11    transaction.commit();
12    return obj;
13  }catch(Exception e){
14    e.printStackTrace();
15     if(transaction != null){
16          transaction.rollback();
17      }
18  }
```

**transaction.begin()** : start of the transaction.
**transaction.commit()** : make changes to the database data.
**transaction.rollback()** : abandoning the transaction.

In order to facilitate de Transactions management, we created a class called PersistenceContext, which contains a EntityManager along other methods, such as:

```
1   public static EntityTransaction beginTransaction(){
2     EntityTransaction tx = getCurrentTransaction();
3     tx.begin();
4     return tx;
5   }
6
7   public static EntityTransaction commitTransaction(){
8     EntityTransaction tx = getCurrentTransaction();
9     tx.commit();
10    return tx;
11  }
12
13  public static EntityTransaction rollbackTransaction(){
14    EntityTransaction tx = getCurrentTransaction();
15    tx.rollback();
16    return tx;
17  }
18
19  public static void closeTransaction(){
20    transaction.set(null);
21    closeCurrentManager();
22  }
```

Thus, for implementing the UserDAO, we have a class UserDAOImpl, which extends a GenericDAOImpl that contains the CRUD methods, but also implements the specific methods from the UserDAO, in our case, the method findByLogin.

```
1   public class UserDAOImpl extends GenericDAOImpl<User>
implements   UserDAO{
2
3     public UserDAOImpl() {
4          super(User.class);
5     }
6
7
8     @Override
9     public User findByLogin (User usr) {
10         EntityManager em = PersistenceContext.getCurrentManager();
11         Query q = em.createQuery("SELECT p FROM User p WHERE
p.username = :username AND p.password = :pass AND p.activated=1");
12         q.setParameter("username", usr.getUsername());
13         q.setParameter("pass", usr.getPassword());
14
15         usr = (User) q.getSingleResult();
16
17         return usr;
18    }
19
20  }
```

As we can notice, we access a static method from PersistenceContext, which is called "getCurrentManager()", which returns a EntityManager object, used for making the transaction. The full source code of the PersistenceContext class is found at the Appendix.

### 5.2.2 Business Layer

The purpose of the business layer is to implement the application logic, exposing this logic to the presentation layer for applications or other remote clients, for example, mobile clients, through well-defined interfaces.

**Design Patterns:** When you seek for a design pattern solution for the business layer, usually you will find quite similar solutions, with similar approaches. Among them we can mention BusinessObject, Façade, ServiceBean, etc.

**ServiceBean:** We decided to use in our application the ServiceBean design pattern. Each module in the system will have a ServiceBean that implement the corresponding logic for that entity. Therefore, on a project that has as entities: User, City, Information, we may expect to find in the business layer classes as UserService, CityService and InformationService, for example.

Each ServiceBean must provide methods capable of performing the operations that we desire in each module. To access the data of each module, it will have an instance of the respective DAO.

Just as done with the DAO, each ServiceBean starts with an interface that contains methods that will be implemented in the "Impl", which are the methods that will be available for the presentation layer. We also develop a GenericService, which contains basic methods that all the other services will need, such as:

```
1  public interface GenericService<T> {
2
3    public void insert(T entity);
4    public void delete(T entity);
5    public T update(T entity);
6    public T findByID(Long entityID);
7    public List<T> findAll();
8
9  }
```

So, in order to develop our UserService, which will contain all the methods from the Generic service, we use the inheritance concept, which in java means that our UserService extends from our GenericService. Furthermore we defined the specific methods.

```
1  public interface UserService extends GenericService<User> {
2    public User findByLogin (User usr);
3  }
```

In our implementation class (UserServiceImpl), we also extend it from the GenericServiceImpl, which implements all the methods from the GenericService, but we also implement the specific method from the UserService, which in our case is findByLogin (User usr).

```
1     public class UserServiceImpl extends GenericServiceImpl<User>
implements UserService {
2
3     private UserDAO usrDAO;
4
5     public UserServiceImpl() {
6           usrDAO = new UserDAOImpl();
7     }
8
9     @Override
10    public User findByLogin (User usr) {
11          try {
12                PersistenceContext.beginTransaction();
13                usr = this.usrDAO.findByLogin(usr);
14                PersistenceContext.commitTransaction();
15          }catch(Exception e){
16                PersistenceContext.rollbackTransaction();
17          }finally{
18                PersistenceContext.closeTransaction();
19          }
20          return usr;
21    }
22
23  }
```

As we can notice, the methods from the PersistenceContext are again needed for beginning and closing the transaction. Furthermore, we can notice that if everything goes right, the transaction committed, and if anything goes wrong, it is rolled back. All the operations are surrounded with the beginning and the committing of the transaction, thus ensuring that the **atomicity** property is respected.

We can also notice in this example that the only assignment of the business layer is to pass the data to the DAO layer. No other type of logic is required in this case. Another important point that we can observe is that **the business layer has no idea how the data is persisted**, it only makes use of the methods defined on the DAO class.

### 5.2.3 Presentation Layer

The purpose of the presentation layer is to make the interaction between man and machine. The layer should provide to the user **data input resources**, usually the keyboard and mouse events, and **data output resources**, usually in the form of visual interfaces, or even sounds.

Developing user interfaces is not an easy task. Developing a layer where the human factor is the main one makes the work harder. When we worked in the previous layers (Business and Data) our interaction was always linked to some software or application, which made our job easier, because in all of them there are protocols and rules that we must follow, and if we do everything right, everything should work as expected. In the presentation layer things are not so simple. Several aspects must be observed and often the main objective is simplification. This goal is always a big task for the developers, because for making the easier it is for the user, the harder it is for the developer.

While developing this project, we noticed that most of the time. Most of the hours of work were linked in some way of the presentation layer. In addition to easy, the application should be intuitive and the workflow should be as natural as possible, which means, the closer to the user's work reality.

**MVC -** MVC is a design pattern widespread in the web developer community. Although widespread in applications that run on the Internet, it can be used on any platform type, such as web, desktop, mobile, etc.

MVC is a design pattern from the presentation layer. Considering so, this abstraction hinders integration and application constructions errors. We will now understand the meaning of each letter of the acronym:

**M - Model**: The model is used to define and manage information's domain. It is a detailed representation of information that the presentation layer has access.

**V - View**: The view shows the model in a format appropriate to the user in the data output. Different views can exist for a single model for different purposes.

**C - Controller**: The controller receives the input data, validates it and than initiates response to the user invoking the business layer objects, which ends up creating or altering the objects of the model layer.

Graphically the MVC solution can be represented as in figure 30.

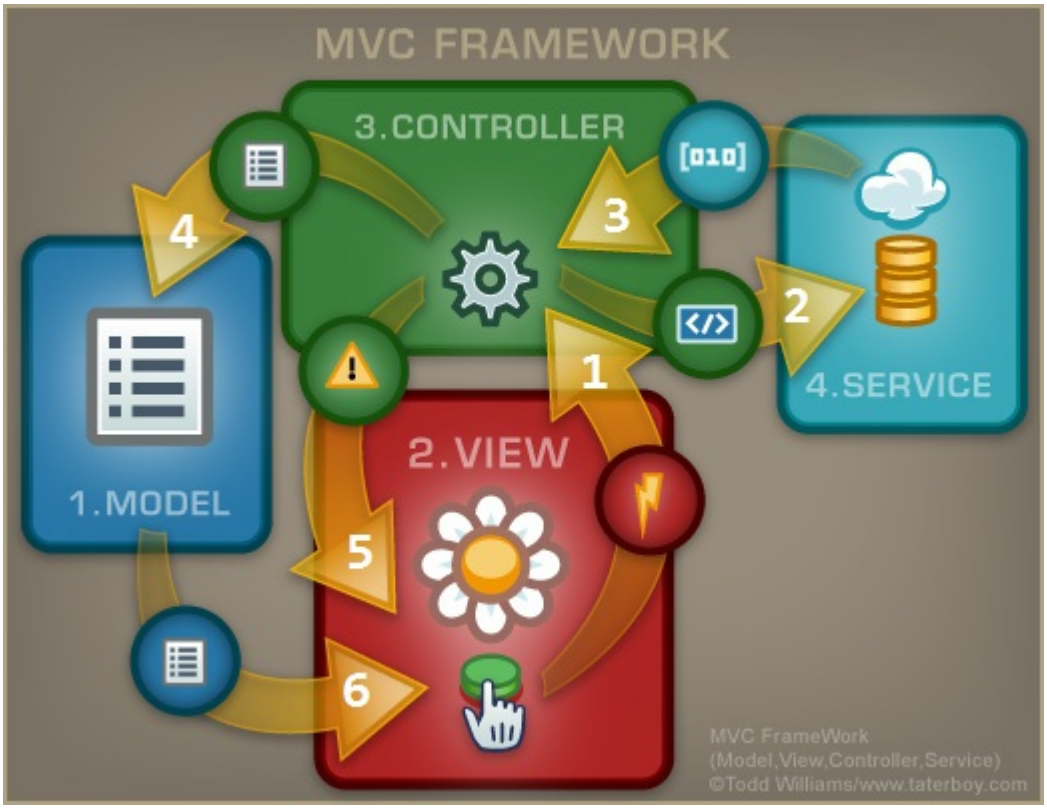


*Figure 30 – MVC Framework* [12].
*http://www.taterboy.com/blog/images/xMVC_graphics01.jpg.pagespeed.ic.bXcSUdTS85.jpg*

We can observe that the user action will generate 6 (six) events within our presentation layer. To illustrate our example, we will see how the users' control system works.

In such a system we have routines for listing and registering users. Following we find a detailed analysis of each step of the request.

**User listing routine**

1) The user clicks on a button or link from view called "List Users";
2) The controller receives the request and asks for the service object a listing of all registered users;
3) The service returns a collection of objects with all registered users;
4) The controller stores the data received from the service into the model;
5) The controller requests a view to display registered users;
6) The requested view uses the data from the model to populate a table with customers.

**User registering routine**

1) The user fills the required data in a form and click on button called "Add";
2) The controller receives the request, retrieves the data entered in each field, encapsulates the data in a "User" object and send a request for registration to the service;
3) The service returns the same object now filled, or some exception can be triggered;
4) The controller interprets the response from the service, and saves in the model an error or success message;
5) The controller requests a view for sending a message to the customer;
6) The requested view uses the data from the model to show the message to the user.

As we can see, the presentation is much more organized and if a problem occurs, we will know exactly where the problem is.

**Servlet**: We developed the application in the Java EE platform, using Servlet and JSP. The approach used in this case is known as Action-based. When the data that needs to be available for the View are "pushed" from the Controller, it is said that we are using a MVC style called MVC Push. Currently, approaches that follow this strategy are known as Action Based.

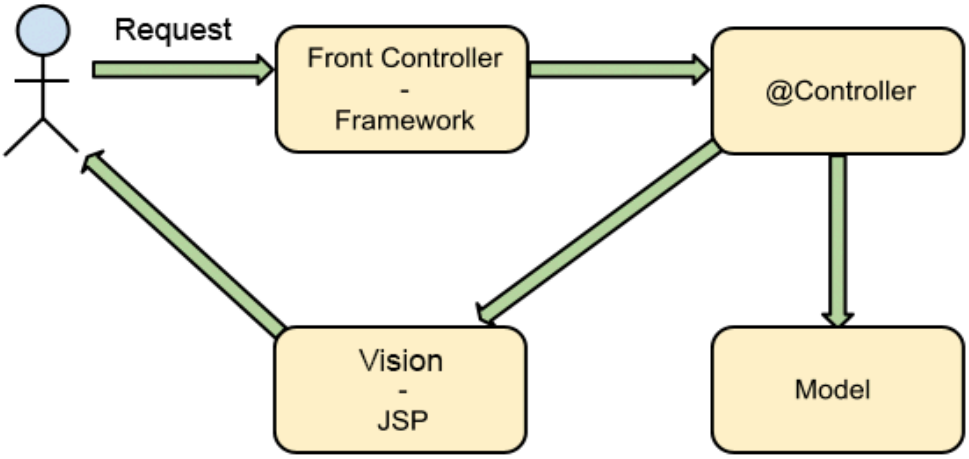


*Figure 31 – Action based Servlet*
*@Controller tells which "Vision" will be returned to the user.*

Our application implements the controller class within the servlet and the views in JSP files. Importantly, to run the application, it is necessary to install a supported web container such as Tomcat 7.0, for example.

On our UserController, we can notice this piece of code:

```
1   @WebServlet(value = "/user.mdl", loadOnStartup = 1)
2   public class UserController extends HttpServlet {
3
4     private UserService userService;
5     private PriviledgeService privService;
6
7     public UserController(){
8           this.userService = new UserServiceImpl();
9           this.privService = new PriviledgeServiceImpl();
10    }
11   //And so on
12   }
```

When any request comes for the servlet, independently whether it is GET or POST, it is handled by a method called handleRequest, as it follows:

```
1   @Override
2   protected void doGet(HttpServletRequest request,
3           HttpServletResponse response) throws ServletException,
IOException {
4           handleRequest(request, response);
5     }
6
7     @Override
9     protected void doPost(HttpServletRequest request,
10              HttpServletResponse response) throws
ServletException, 12   IOException {
13          handleRequest(request, response);
14  }
```

The handleRequest method receives the request and response and can check what the user want to do, by retrieving the "action" parameter from request. Then, it is possible to call the corresponding method according with the content of the action parameter:

```
1   protected void handleRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {

2           String action = request.getParameter("action");
3
4           if(action==null) {
5                this.showLoginPage(request, response);
6           } else if (action.equals("makeLogin")) {
7                this.makeLoginAjax(request, response);
8           } else if(action.equals("register")) {
9                this.register(request,response);
10          } else if(action.equals("list")) {
11               this.list(request,response);
12          }else if(action.equals("edit")){
13               this.edit(request,response);
14          } else if (action.equals("logout")) {
15               this.logout(request,response);
16          }
17
18    }
```

The "magic" of transforming the request from the user into a "User" object, is inside a method called requestToBean:

```
1   private User requestToBean(HttpServletRequest request) {
2
3     User bean = new User();
4
5     String _strID = request.getParameter("user_id");
6     String _strPriviledge = request.getParameter("user_priviledge");
7     String _strActivated = request.getParameter("user_activated");
8
```

```
9     if (_strID != null) {
10          bean.setId(new Long(_strID));
11    }
12    if (_strPriviledge != null) {
13          bean.setPriviledge(this.privService.findByID(new
Long(_strPriviledge)));
14    }
15    if (_strActivated != null) {
16          bean.setActivated(new Boolean(_strActivated));
17    }
18
19    bean.setName(request.getParameter("user_name"));
20    bean.setPassword(request.getParameter("user_password"));
21    bean.setUsername(request.getParameter("user_username"));
22
23    return bean;
24
25  }
```

We can notice that on line 3, an object called bean, which is an instance of the User class is created, and it is filled out with the informations that comes from the JSP.

All the JSPs are stored in the WebContent/website folder. Let's see an example of how the client listing is done.

First, in the menu, there is a link for listing clients:

```
1   <li><a href="user.mdl?action=list">List users</a></li>
```

As we can see, the action is a parameter, and list is its value. The handleRequest notices that, and call the method "list":

```
1   this.list(request,response);
```

This method is implemented later as:

```
1   private void list(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
2     RequestDispatcher rd =
```

```
this.getServletContext().getRequestDispatcher("/website/listUser.jsp")
;
3
4     List<User> userList = this.userService.findAll();
5     request.setAttribute("users", userList);
6
7     rd.forward(request, response);
8   }
```

On line 4, we create a List of Users, called userList. This list is instantiated with the method findAll from the userService. Therefore, after this call it contains all the users registered on the database. Than, this userList is put as an attribute into the request, and a forward is done for showing the "listUser.jsp" page.

The "listUser.jsp" looks like this:

```
1   <%@ page import="br.com.io.thesis.domain.*"%>
2
3   <%
4     List<User> users = (List<User>) request.getAttribute("users");
5   %>
```

First of all, we import all the classes within the domain package, which mean the User, Priviledge, City, Information and InfoType classes. Than, we create a List of users, called "users" and instantiate with the attribute that came from the Servlet, called also "users". At this moment, this local variable from the Jsp called users, contains all the users and their informations.

Putting these informations on a table, is now an easy job:

```
1   <table>
2    <thead>
3     <tr>
4           <th>ID</th>
5               <th>Name</th>
6               <th>Username</th>
7               <th>Password</th>
8               <th>Priviledge</th>
```

```
9               <th>Activated</th>
10              <th>Action</th>
11    </tr>
12   </thead>
13
14     <tbody>
15    <%
16    for (User usr: users) {
17          out.print("<tr>");
18               out.print("<td>" + usr.getId() +  "</td>");
19               out.print("<td>" + usr.getName() +  "</td>");
20               out.print("<td>" + usr.getUsername() +  "</td>");
21               out.print("<td>" + usr.getPassword() +  "</td>");
22               out.print("<td>" + usr.getPriviledge().getName() +
"</td>");
23               out.print("<td>" +usr.getActivated() +  "</td>");
24               out.print("<td> <a
href=\"?action=edit&id="+usr.getId()+ "\"> [EDIT] </a> </td>");
25          out.print("</tr>");
26    }
27
28    %>
29     </tbody>
30  </table>
```

Some libraries were very usefull for developing and designing the front-end pages.

- angular.js [30] – Great tool for extending HTML Vocabulary to the application;
- bootstrap.min.js [31] – Very powerfull HTML, CSS and JavaScript framework for developing web/mobile projects;
- jquery-1.10.2.min.js [32] – Very powerfull JavaScript library;
- jquery.dataTables.js [33] – Plug-in for jQuery for handling tables.

For de styling:

- bootstrap.min.css [31] – The corresponding style for the bootstrap JavaScript.

- bootstrap-theme.min.css [31] – Some pre-defined beautifull themes that can be used with bootstrap;
- jquery.dataTables.css [33] – The corresponding style for the dataTables JavaScript.

### 5.2.4 Creating a Firewall

One crucial point for the webserver is the security. How to ensure that a user that have a "Reduced" privilege will not be able to insert/edit another users, while a "Grand all" user can do everything? In order to understand that, it is necessary to take a look at the package "br.com.io.thesis.presentation.security", which contains a so-called **PresentationServletFirewall** file. When the user makes the login, an objet called usr, that contains the user, is stored in the session.

Our PresentationServletFirewall contains a annotation, showing java compiler that it is a filter:

```
1   @WebFilter(filterName="firewall",urlPatterns={"*.mdl","*.html"})
```

Before any page that ends with .mdl or .html be called, it passes through this firewall, which will grant or refuse access to the page.

The first thing to be done, is to create a list that contains all pages that are free to any one, being logged or not, being "Reduced" or "Grant All" privilege; this is called in the code **freePermissions**. The second step is to define also a list that contains all pages that are free for the "Reduced" privilege. A list for the "Grant All" is not necessary, taking in account that it can access everything; this is called in the code **reducedPermissions**. Thus, we need 2 lists, as we can see:

```
1   public ArrayList<String> freePermissions;
2   public ArrayList<String> reducedPermissions;
```

After creating these arrays, we are ready to populate then with the actions that the user can do. These actions are mapped in such a way:

| Link | Permission name |
|---|---|
| /InternationalOffice/information.mdl?action=list | information.list |
| /InternationalOffice/information.mdl?action=register | information.register |
| /InternationalOffice/user.mdl?action=list | user.list |
| /InternationalOffice/user.mdl?action=register | user.register |

*Table 2 – Firewall Permissions*

And so on. As we can see, the permission name is created using the requested page. First is the name of the module, followed by a "." and than the value of the action parameter.

So, there is the code it would look something like this:

```
1   //Free Permissions
2   freePermissions = new ArrayList<String>();
3   freePermissions.add("user.null");
4   freePermissions.add("user.makeLogin");
5   freePermissions.add("information.responseMobile");
6
7   //Reduced Permissions
8   reducedPermissions = new ArrayList<String>();
9   reducedPermissions.add("information.list");
10  reducedPermissions.add("information.register");
11  reducedPermissions.add("user.logout");
```

All the other pages, such as user.list, user.register, and so on, are only available for users with the "Grant All" privilege.

When a request come for an ".mdl" or an ".html", the method doFilter is called:

```
1   @Override
2   public void doFilter(ServletRequest req, ServletResponse res,
FilterChain chain) throws IOException, ServletException {
3
4     HttpServletRequest request = (HttpServletRequest)req;
5     HttpServletResponse response = (HttpServletResponse)res;
6     HttpSession session = request.getSession();
7
8        String local = getName(request);
9
10       User usr = (User) session.getAttribute("usr");
11
12     if(freePermissions.contains(local)){
13    if (local.equals("user.null") && usr!=null && usr.getId() !=0) {
```

```
14          response.sendRedirect("information.mdl?action=register");
15     }
16     else {
17          chain.doFilter(req,res);
18     }
19       }
20       else if (reducedPermissions.contains(local) && usr!=null &&
usr.getPrivilege().getId().equals(Privilege.PRIV_GRANT_REDUCED)) {
21          chain.doFilter(req,res);
22       }
23       else if (usr!=null &&
usr.getPrivilege().getId().equals(Privilege.PRIV_GRANT_ALL)) {
24     chain.doFilter(req,res);
25       } else {
26          response.sendRedirect("user.mdl");
27       }
28
29  }
```

On the line 8, the method getName returns the permission name, as we could see on Table 1. Than, we retrieve the user object from the session and check whether it privilege fulfills the requirements of the page he requested to see. First of all, if the freePermissions contains the local, we do a further check, to see if the user attribute retrieved from the session contains a user. If it does, instead of showing the login page again, we show the Information Registering page; otherwise, the chain.doFilter method shows the desired page requested by the user.

If it is not a freePermission, we check whether it is a reducedPermission. If so, than we check if the user has already logged in (therefore, the usr attribute from the session must not be null), and if he has the "Reduced Privilege". This privilege is declared as a static final long variable on the Privilege class, as we can see on the following piece of code:

```
1   public static final Long PRIV_GRANT_ALL = new Long(1);
2   public static final Long PRIV_GRANT_REDUCED = new Long(2);
```

If none of these criteria have been fulfilled, then it is for sure page that requires a "Grant All Privilege", so we use the same procedure for checking whether the user have this privilege or not.

### 5.2.5 Creating a jSON Response for the Mobile

Even though the creation of the jSON response for the mobile is done in the presentation layer, we decided to create a separated chapter for that, taking on account that it is very crucial for this application. There are several ways of creating a jSON, the simplest way is to use a library in order to create a jSON. Its library has also some dependencies:

- **json-lib-2.4-jdk15.jar** [25]: Java library for transforming beans, maps, collections, java arrays and XML to JSON;
- **ezmorph-1.0.2.jar** [26]: Simple java library for transforming an Object to another Object;
- **commons-beanutils-1.8.0.jar** [27]: BeanUtils provides an easy-to-use but flexible wrapper around reflection and introspection;
- **commons-lang-2.6.jar** [28]: Commons Lang, a package of Java utility classes for the classes that are in java.lang's hierarchy, or are considered to be so standard as to justify existence in java.lang;
- **commons-logging-1.0.4.jar** [29]: Commons Logging is a thin adapter allowing configurable bridging to other, well known logging systems;

The link for checking the jSON response, is the following page:

/InternationalOffice/information.mdl?action=responseMobile

When the user access this page, it access the InformationController class, which is a servlet, because it is mapping the "information.mdl".

```
1   @WebServlet(value = "/information.mdl", loadOnStartup = 1)
2   public class InformationController extends HttpServlet {
```

Than a further check is made for retrieving the action from the requested, which in our case was responseMobile. Therefore, a method also called "responseMobile" is called. This method is responsible for creating the jSON response and printing it out. The following piece of code shows how it is done:

```
1   private void responseMobile(HttpServletRequest request,
HttpServletResponse response) throws IOException {
```

```
2
3     JSONObject json = new JSONObject();
4
5     Date today = new Date();
6     List<Information> infoList =
this.informationService.findActivesInTime(today);
7
8
9     for (Information info:infoList)  {
10
11            JSONObject information = new JSONObject();
12            information.put("name", info.getName());
13            information.put("description", info.getDescription());
14            information.put("type", info.getInfoType().getName());
15            information.put("city", info.getCity().getName());
16
17          json.accumulate("informations", information);
18
19    }
20
21    response.addHeader("Access-Control-Allow-Origin", "*");
22    response.setContentType("text/html; charset=UTF-8");
23    response.setCharacterEncoding("UTF-8");
24    PrintWriter out = response.getWriter();
25    out.println(json.toString(2));
26
27  }
```

First of all, we have a `informationService` object, that is from the type `InformationService`, and instantiated as a `new InformationServiceImpl()`.

As we want to show only informations marked as active, and within a date interval (information date begin < TODAY < information date end), we created a method on the InformationDAO that does this job, and it can be called by the informationService. More details about how it was done can be found in the class **InformationDAOImpl** in the Appendix.

After retrieving all the active informations within this time interval, we create a JSON Object called "information" for each information in the list, and put the name, description,

type and city parameter values into it. After that, it is accumulated in the main jSON.

After iterating the entire array, the jSON is printed, and appear as something like that:

```
{"informations": [
    {
    "name": "French Evening",
    "description": "Dear students from Jade University. On the next Tuesday, will have the an event on the evangelic church. It will be the french evening and everybody can have a lot of fun!",
    "type": "Events",
    "city": "Wilhelmshaven"
  },
    {
    "name": "Brazilian's Goodbye Party",
    "description": "Dear students from the international office. On the 22nd of July, the Brazilians are going back to Brazil. Lets celebrate with than! You are all invited to a party in their house at the 20th July at 9PM.",
    "type": "Events",
    "city": "Wilhelmshaven"
  },
    {
    "name": "Learning Agreement",
    "description": "Dear students, the deadline for delivering the Learning Agreement from your university is on the 08/20. It should be delivered in the International Office.",
    "type": "Formalities",
    "city": "Wilhelmshaven"
  }
]}
```

As we can notice, we have 3 informations and the response contains its name, description, type and city. Now we are ready for developing our application!

## 5.3 Application

The development of the I.O. Jade Hochschule application was possible with the support of Intel XDK features combined with HTML5, CSS3 and JavaScript, including specific libraries such as jQuery and jQuery Mobile. This topic highlights the main parts of the application development, explaining their respective functions.

### 5.3.1 Creating the APP

With our experience with Intel XDK, we realized that work with a demo or even with a template is very helpful and easier when developing an application, however these attributes make the app heavier than a blank page app, because they add unnecessary functions. For this reason we decided to make a blank page app, even though it requires more time, it is easier to handle, it has a cleaner text and it is easier to support.

### 5.3.2 Libraries

To support the functionalities of our application, we used some important libraries, such as: jQuery [32] and jQuery Mobile [34] with their respective CSS.

### 5.3.3 Application Purpose

As known, Jade Hochschule has six departments split in 3 cities: Wilhelmshaven, Oldenburg and Elsfleth, so our application can be used for foreign students of any of these cities, because it has an option to choose the respective city and then they can have access to the informations.

In an interview with an international office member, was established that we would work with 4 types of informations in the application, such as: events, formalities, informations and news. When the application is started, the latest announcements will appear over the old ones, so that the user will be always updated. However if the application has already started and then the International Office post a new announcement, the user has to scroll down the page in order to receive it, updating the application.

All the announcements will appear as a list, so if their texts are longer than three lines, the user has to click on them and a new page will appear, displaying the whole content.

### 5.3.4 Development

Inside the head part of the code, we entitled our application as I.O. Jade Hochschule.

We used the utf-8 character model in case of having different characters, like the German ones, so it can interpret all kind of characters:

```
1   <meta http-equiv="Content-type" content="text/html; charset=utf-8">
```

Some scripts were necessary to use in order to support the application. The first script we used, was the following:

```
1    <script src='intelxdk.js'></script>
```

Which is basically the Intel XDK JavaScript. It contains a lot of functions, such as detecting when the application was launched, making Ajax requests, creating tab bars and so on.

We also opt to use the latest version of jQuery [32] (1.11.0) as well as jQueryMobile [34] (1.4.2) for making the complete application:

```
1    <script src='scripts/jquery-1.11.0.min.js'></script>
2    <script src='scripts/jquery.mobile-1.4.2.min.js'></script>
```

This was optional, because for our small application, the Intel XDK JavaScript probably provided everything that was necessary. Nevertheless, we opt to use these 2 JavaScripts, because they are easier to handle, and we had previous knowledge of them.

In order to create the "push to update" functionality, which basically updates the informations when the view is pushed down, we used:

```
1    <script src='scripts/jquery.scrollz.min.js'></script>
```

But we also had to create a second script:

```
1    <script src='scripts/jsoniojadehochschule.js'></script>
```

The first one contains the properties and the call to be done when the view is pushed, while the second one is the responsible for querying the Web Service, and interpreting the jSON response. Furthermore, it also creates dynamically the views (div and its sub views) containing the messages and its contents in the main view.

For styling our application, we use the corresponding style for the jQuery Mobile [34] JavaScript:

```
1    <link href="style/jquery.mobile-1.4.2.min.css" rel="stylesheet"
type="text/css">
```

The same procedure was made for the scroll view, which contains its own CSS.

```
1    <link href="style/jquery.scrollz.css" rel="stylesheet"
type="text/css">
```

Furthermore, we created our own style sheet, containing the style properties for every element, classes and ids that we choose:

```
1    <link href="style/style.css" rel="stylesheet" type="text/css">
```

As mentioned previously the application is composed by two pages. The first page displays all the messages retrieved from the Web Service, and the second one displays the whole content of the chosen message, when a specific message is clicked.

For doing so we divided the body into two parts. The first part contains a div with the data-role “page” and the id “mainPage”:

```
1   <div data-role="page" id="mainPage">
```

The second part also contains a div with data-role “page” as well as the id, which the value is set to “infoPage”:

```
1   <div data-role="page" id="infoPage">
```

Data-role is a jQuery attribute that must be always within a div. It has the function of dividing the application into pages. Therefore, each div that contains the data-role attribute equal to page, the jQuery knows it consists in a different page; that’s why we need to have an id for each page, so that we can stylize it however we want.

When the application is launched the jQuery shows only the first page, which means, the first div with data-role equal to “page”, hiding the others. When a different page is called, jQuery hides the first one and displays the chosen page.

The “mainPage” id is used to stylize the background of the first page, defined by a radial gradient function that creates an image with a defined shape (circle or ellipse) and its colors start from the center until the end of the image:

```
#mainPage {
        background:
        radial-gradient(black 15%, transparent 16%) 0 0,
        radial-gradient(black 15%, transparent 16%) 8px 8px,
        radial-gradient(rgba(255,255,255,.1) 15%, transparent
20%) 0 1px,
        radial-gradient(rgba(255,255,255,.1) 15%, transparent
20%) 8px 9px;
        background-color:#282828;
        background-size:16px 16px;
}
```

It also contains a div with the “myheader” id:

```
<div id="myheader"></div>
```

Its CSS consists in a fixed position, which means that when the user scroll down or up the page the header will not move. Its width occupies 100% of the screen, and the text-align attribute is used to centralize the header. The z-index attribute was used in order to it be displayed always in front of the messages while they are scrolled.

```
#myheader{
    height: 50px;
    position: fixed;
    top: 0px;
    width: 100%;
    z-index: 99999;
    text-align: center;
}
```

This div should contain the name of the application that is “I.O. Jade Hochschule”, inside an h2 element. However, during the development we found a bug that happens in Android devices: if a heading element, such as h2 is placed within the header div, its contents is not displayed. Moreover, we found that when the “text-align:center” is removed, it displays normally, but not centered, of course. However, as we wanted the header element to be displayed on the center of the header div, we developed a workaround; adding dynamically the heading element via JavaScript (jQuery) after the application is launched did the trick:

```
1   $("#myheader").html("<h2>I.O. Jade Hochschule</h2>");
```

Within the first page, there is another div, which will have the id “total”:

```
1   <div id="total">
```

This div is composed by all the other divs of the first page, except for the header. The “total” id is composed by a defined margin, used to place a select bellow the application title:

```
1   #total{
2       margin-top: 50px;
3    }
```

It contains a div with the “allMessages” id:

```
1   <div id="allMessages"></div>
```

As explained before, the content of this div is empty, because our application works with dynamic messages and not static. The application will receive always messages with different titles, texts and footer texts. For this reason, it is necessary to put the whole content of it inside a JavaScript function, which receives and interprets the jSON received from the webserver. For each information contained in the jSON response, it creates dynamically inside the “allMessages” div another div (with class = “messages”) with its content.

Lets take a look on the piece of code of our “jsoniojadehochschule.js” script that does this:

```
1   var code = "<div class='messages'><div class='title'><span>" +
info.name + "</span></div><p>" + truncatedInfo + "</p> <input
type=‘hidden’ value=‘“ + info.description +”’ ><a><input type=‘hidden’
value=‘“ + info.city +”’ ></a><span class='bottom'>" + info.type +
"</span></div>";
```

The full source code can be found in the Appendix.
For the class “messages” we defined background color, margin, padding and height:

```
.messages{
    background-color: #F0F0F0;
    margin: 11px;
    padding: 4px 5px;
    height: 126px;
 }
```

The “messages” div contains a div with a “title” class, which is composed by a span element so that it can have a different style from other texts. The title is centralized, has a specified color, font size and font weight:

```
.title{
    text-align: center;
    color: #DD0A20;
    font-size: 115%;
    font-weight: bold;
 }
```

Inside the “messages” div, there is a  element, where the text of the real message will be displayed. It has a defined margin, height and color:

```
.messages p {
    margin-top: 5px;
    margin-bottom: 10px;
    height: 60px;
    color: #484848;
 }
```

It also contains also a span with a “bottom” class, which is also stylized differently of the other texts and the it will display the message type. Color, font size and font style are defined as:

```
.bottom{
    color: #DD0A20;
    font-size: 110%;
```

```
        font-style: italic;
        padding-left: 5px;
    }
```

Furthermore, we handled the event when the person clicks on the div that has the class "messages". When this happens, we fill the next page elements with the value of the elements contained in the message clicked, and redirect the user to the second page, where he is able to read the full message:

```
    $('#allMessages').delegate(".messages", "click", function() {
                var info = $(this).children('input').val();
                var type = $(this).children('span').text();
                var title =
$(this).children('div').children('span').text();

                $('#actualTitle').text(title);
                $('#actualDescription').text(info);
                $('#actualType').text(type);


               document.location="#infoPage";
            });
```

The second page is the one that contains the whole content of the chosen message, such as:

1. A div with data-role="page" and an "infopage" id:

```
    <div data-role="page" id="infopage">
```

2. A div with the "actualMessage" id, and inside some elements, which have content fulfilled when the user clicks on any of the messages:

```
    <div id="actualMessage">
        <div class="title"><span id="actualTitle"></span></div>
        <p id="actualDescription"></p>
        <span id="actualType"></span><br/>
    </div>
```

3. A link to go back to the first page:

```
    <a href="#infopage">Go Back</a>
```

In order to create the city filtering, we created a select where the user is able to choose his respective University's city. The possible options are Wilhelmshaven, Oldenburg and Elsfleth:

```
<select id='citySelect'>
     <option value='Wilhelmshaven'>Wilhelmshaven</option>
     <option value='Oldenburg'>Oldenburg</option>
     <option value='Elsfleth'>Elsfleth</option>
</select>
```

After that we created a function using JavaScript with jQuery, that listens to the event "change" of the select. When it happens, it iterates all the messages, and compares its input that contains the city with the city selected on the options. If they are equal the message is displayed, otherwise it is hidden:

```
$( "#citySelect" ).change(function() {
    var cityFromSelect = $( "#citySelect" ).val();
    $( ".messages" ).each(function() {
        var cityFromMessage =
$(this).children('a').children('input').val();
        if (cityFromMessage == cityFromSelect) {
             $(this).show();
        } else {
            $(this).hide();
        }
    });
 });
```

### 5.3.5 Simulating

This is the simulation of our application, so that it is possible to have an idea of how it will really be:

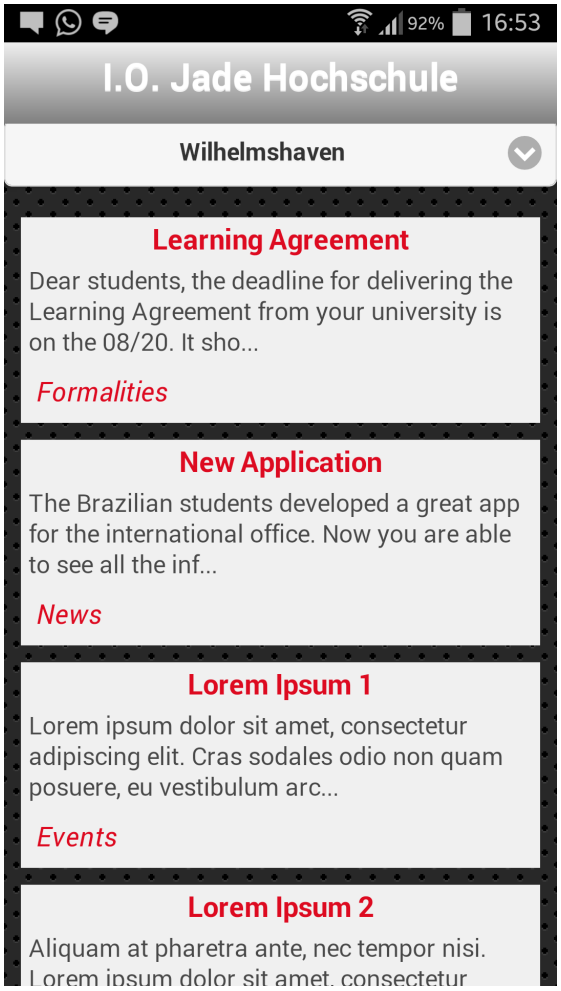


*Figure 32 – Application First Page*

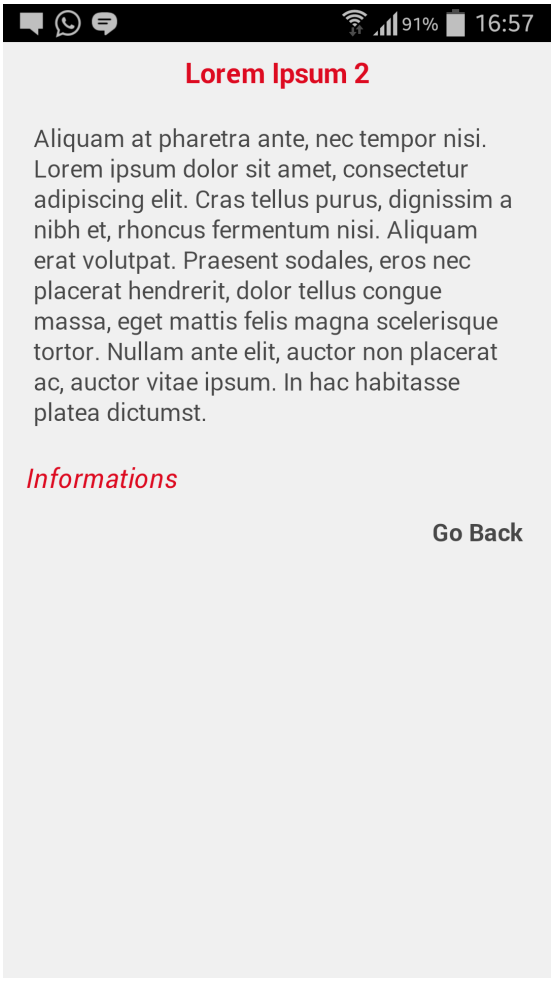


*Figure 33 – Content of Message*

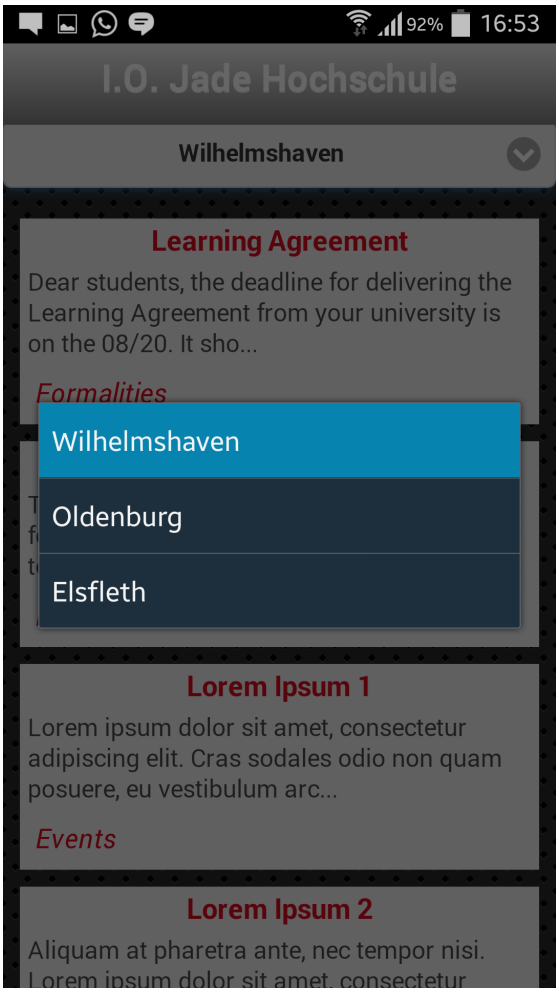


*Figure 34 – Android Button Style*

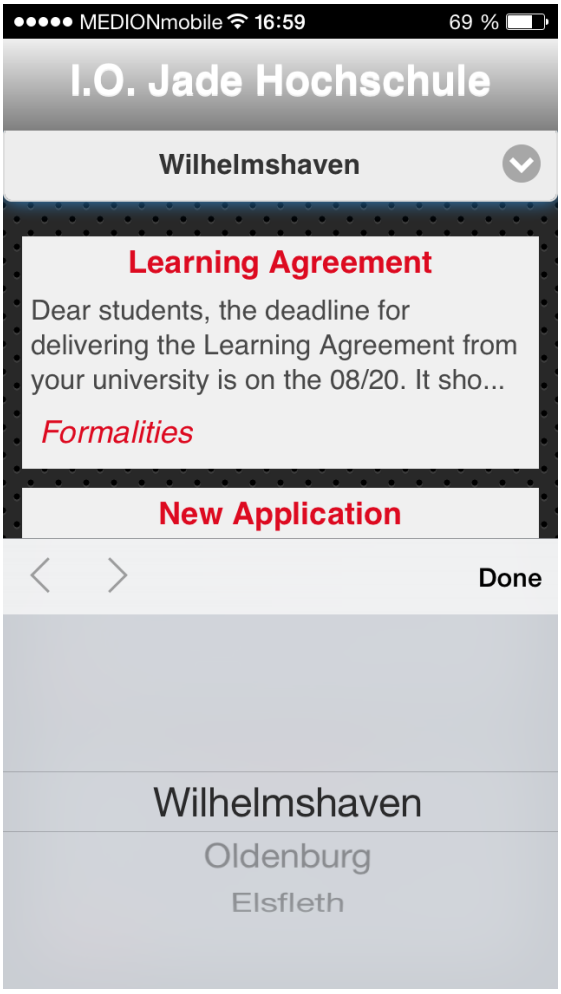


*Figure 35 – Apple Button Style*

## 5.4 Further Considerations

1- As we used design patterns to develop the source code of the Web Service, it is divided in several layers, represented as packages in java, as shown in Figure 29. Each DAO, DAOImpl, Service, ServiceImpl and Presentation package contains one class, with a well-defined purpose for each Table containing the database: City, Information, InformationType, Privilege and User. Therefore there are 6 classes, such as City, CityDAO, CityDAOImpl, CityService, CityServiceImpl, CityPresentation, for each one of these tables.

However, as we the concept of inheritance was used, a so called "**Generic**" class was created for each one of these layers. Thus, a class "CityDAO" for instance, only needs to inherit itself from GenericDAO, without any further implementation. That way, the code of these classes are very short. In this sense, the only exceptions on the Web Service are the Information and User classes, which have specific functions beyond the Generic classes. This can appear a little bit unnecessary at first, but these patterns exists for a reason: Using the layer abstraction it is possible to know exactly where a problem is locatted, and also, when a Project beggins to grow quickly, they are very helpfull because each class has one very defined purpose.

2- Unfortunately, we did not have time to develop everything that we wanted. The application and the Web Service do what they are supposed to do, but there are some points that will be added in future versions, such as:

- Possibility for the International Office to create events, and link than with a Facebook album;
- Creation of a Tab Bar in the application, where it will show all registered events, and their respective photos;
- Give the possibility to the students to upload these photos directly to the Facebook album regarding an event, and the International Office to approve or not these photos.
- Storing the password in MD5 hashes is a great thing for security, but can be improved by also salting it.

3- Moreover, there are also some well-known small bugs in the application:

- In Galaxy S4, there is a delay for selecting an option on the select button, in our case regarding to the city. This is an Intel XDK problem, and probably will be solved in the next version;
- Users do not have the permission for changing their own password;

- The date-picker for the information does not work in every browser. In some tests we notice that only a simple input with a date type, such as: <input type="date" name="bday"> will only work properly in Chrome or in Opera. In further research, we found out that this is one of the 12 HTML5 new Input Type, and for those, not all browsers can work properly yet, as can be seen in w3schools website: http://www.w3schools.com/html/html5_form_input_types.asp

# 6 Conclusion

The development of cross-platform applications is an important subject. The number of developers that are using it is only increasing, and there is a great open market for it.
Although it have limitations comparing to native applications, developing cross-platform applications using HTML5 + CSS3 and JavaScript is much easier and more efficient, as it spares a great amount of time.

There is a huge amount of frameworks such as Intel XDK and jQuery Mobile that makes the development of HTML5 applications very easy. The advantages of using these frameworks are countless, and can be summarized in one sentence: "write less, do more". With these frameworks, it is possible to design a single application that will work on all smartphone devices, regardless its platform. Furthermore, it gives your application the "look and feel" of the user device, without you have to write any more code.

Even though it possible to make full applications, games, and a huge amount of things within the application itself, normally, most of the applications regardless whether it is cross-platform, relies on a so called Web Service, which is basically where all the informations can be accessed, inserted or removed. In this regard, the great advantage of using Web Service for an application, is that the data is not stored inside the user's device (which could be space consuming), but on a server that can be accessed from everywhere.

On our studies, we came to the conclusion that developing applications with a Web Service can be very difficult for new developers. However, if you know the right toolkits (APIs) or frameworks, and follow design patterns as well as its standard, you will notice that it becomes much more easier.
As a matter of fact, the architecture of an application is basically dictated by the framework you are using. Sometimes following standards it is time consuming and seems not efficiently, while creating your own for your "small application" seems a better idea. However, when the application begins to grow quickly, you will notice that the actual model does not support all the requirements anymore, and therefore a change is necessary.
Despite learning and following protocols seems boring, it is strongly recommended that you do so, rather than just start to code.

# 7 Bibliography


**Literature**

[1] Pilgrim, M. (2010). *HTML5: up and running*. " O'Reilly Media, Inc.".

[2] LISBOA, F. (2010). *Introdução ao Demoiselle Framework: Uma abordagem comparativa de Utilização do padrão MVC para o desenvolvimento de aplicações Web em Java orientada ao reuso*. Brasília. SERPRO.

[3] O'Neil, E. J. (2008, June). Object/relational mapping 2008: hibernate and the entity data model (edm). In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 1351-1356). ACM.

**Research**

[4] Kendo UI, 2013. *Global Developer Survey 2013*, [online] Available at <http://www.telerik.com/whitepapers/kendo-ui/global-developer-survey-2013> [Accessed 04 June 2014].

**Internet Sources**

[5] Intel, *Intel® XDK Html5 tools*, [online] Available at <https://software.intel.com/en-us/html5/tools> [Accessed 01 June 2014].

[6] Intel, December 2013. *XDK Documentation*, [online] Available at <https://software.intel.com/en-us/html5/xdkdocs> [Accessed 03 June 2014].

[7] jQuery Foundation, 2014. *jQuery API Documentation*, [online] Available at <http://api.jquery.com/> [Accessed 15 June 2014].

[8] jQuery Foundation, 2014. *jQuery Mobile 1.4 API Documentation*, [online] Available at <http://api.jquerymobile.com/> [Accessed 18 June 2014].

[9] Gaić, Dragan, July 2013. *TOP 7 MOBILE APPLICATION HTML5 FRAMEWORKS*, [online] Available at <http://www.gajotres.net/top-7-mobile-application-html5-frameworks> [Accessed 28 June 2014].

[10] Eckstein, Robert, March 2007. *Java SE Application Design With MVC*, [online] Available at <http://www.oracle.com/technetwork/articles/javase/index-142890.html> [Accessed 23 June 2014].

[11] Almeida, Adriano, July 2012. *Entenda os MVCs e os frameworks Action e Component Based*, [online] Available at <http://blog.caelum.com.br/entenda-os-mvcs-e-os-frameworks-action-e-component-based> [Accessed 25 June 2014].

## Figures

[12] Figure 30. MVC Framework, *Todd Williams; MVC FrameWork*; Accessed June. 27, 2014 from Todd Williams website <http://www.taterboy.com/blog/images/xMVC_graphics01.jpg.pagespeed.ic.bXcSUdTS85.jpg>.

## Libraries

[14] http://hibernate.org/orm/downloads

[15] http://www.antlr.org/download.html

[16] http://commons.apache.org/proper/commons-collections/download_collections.cgi

[17] http://dom4j.sourceforge.net/dom4j-1.6.1/download.html

[18] http://www.javassist.org

[19] http://www.oracle.com/technetwork/java/javaee/jta/index.html

[20] http://www.slf4j.org/download.html

[21] http://www.slf4j.org/legacy.html

[22] http://logging.apache.org/log4j/1.2/download.html

[23] http://www.mysql.com/products/connector

[24] http://mvnrepository.com/artifact/org.hibernate.javax.persistence/hibernate-jpa-2.0-api/1.0.0.Final

[25] http://json-lib.sourceforge.net

[26] http://ezmorph.sourceforge.net

[27] http://commons.apache.org/beanutils

[28] http://commons.apache.org/lang

[29] http://commons.apache.org/logging

[30] https://angularjs.org

[31] http://getbootstrap.com

[32] http://jquery.com

[33] http://www.datatables.net

[34] http://jquerymobile.com