



Higher Institute of Applied Sciences And Technology Sousse

Summer Internship Report for
First Year engineering
by
Oussema Hidri

Create Hybrid Mobile Application With MobileFirst 7.1



Table of Contents

Introduction	1
I. Presentation of the Company.....	2
II. Hybrid Mobile Application.....	3
a. What's an Hybrid Mobile App?	4
b. How it is built?	4
c. Why choosing Hybrid app?	5
1. AngularJs.....	6
a. Directive	6
b. Module	8
c. Controller	10
d. Other	11
2. Apache Cordova.....	12
3. Ionic.....	14
III. MobileFirst.....	15
1. Architecture.....	16
2. Adapters.....	17
a. Create Adapter	18
b. HTTP Adapter	20
c. SQL Adapter	23
d. Test and Invoke Adapters	25

3. Authentication.....	28
a. Concept of MobileFirst Authentication	29
b. Configuring the Authentication	32
4. File Structure of MobileFirst.....	34
IV. Overview of the Application.....	37
1. The Database	38
2. Home page	41
3. Signup	42
4. Login	43
5. Saved News	44
6. Search for News	46
7. Weather	47
8. Menu	53
V. Improvements.....	54
Conclusion.....	56

Introduction

As a first year engineer student at *Higher Institute of Applied Sciences and Technology Sousse*, I needed to do a minimal one month summer internship to enhance my both soft and technical skills and to prepare me for my future professional life.

After a resume postulation and an interview, I was accepted as an intern in Proxym-Group.

I was very happy to work in very prestigious company and excited to learn new things and make new connections.

In this two months internship, I was able to fully develop an hybrid mobile application.

I have also learned to work in a professional environment as well to feel like I was like one of the engineers at this great company.

I. Presentation of the Company:



Proxym Group is an international IT group operating in Europe, Middle East and North-West Africa with offices in Dubai, Paris and Sousse.

They help their customers leverage IT innovation and mobile-enable and upgrade their organizations by providing competitive and innovative IT Know-how, solutions and services.

They are at the crossroad of Mobile, Information System and Web.

They provide services and industry solution. Their +100 skilled enthusiast engineers have successfully built over 230 projects and delivered more than 120 mobile apps for customers of various sectors and sizes.

II. Hybrid Mobile Application:

Mobile applications are the hot thing in our days. There are billion of application that provide multiple functionalities for users.

Being able to create a mobile application is an amazing thing, and you will have a feeling of satisfaction going through the process of developing, testing and finishing the application; at least that's what I felt.

I was lucky enough to learn and to develop an Hybrid Mobile App in this two months internship.

At first I was skeptical about the capabilities of such app, and also hesitant to devote 2 months time to create a full app, but when I started learning about this new field I was eager to learn, to test new things, and to see the final result.

I also realized that hybrid apps are a perfect choice, if you are good at web technologies and you don't have the time to learn creating mobile applications with native languages.

But first, let's go through three basic questions.

a. What's an Hybrid Mobile App?

Hybrid mobile apps are like any other apps you will find on your phone. The key difference is that, they are hosted inside a native application that utilizes a **mobile platform's WebView**.

WebView: is like a chromeless browser window configured to run fullscreen.

Using WebView enables the application to access device capabilities that are often restricted to access from inside mobile browsers.

A well-written hybrid app should not look or behave any differently than its native equivalent.

b. How it is built?

Hybrid mobile applications are built in a similar manner as websites. Both use a combination of web technologies:

- HTML
- CSS
- JavaScript

However, instead of targeting a mobile browser, hybrid apps target the WebView.

This enables them to do things like access hardware capabilities of the mobile device.

Today, most hybrid mobile apps leverage Apache Cordova, a platform that provides a consistent set of APIs to access device capabilities through plug-ins - more on that later.

c. Why choosing Hybrid app?

Now, we know what an Hybrid app is and how it is made, *what are the motivation to go Hybrid?*

Hybrid mobile app development enables developers to target more than one platform.

Typically, this approach is chosen because it offers a path to create a full application in a faster time.

But the downside is, you can find yourself targeting the features of a mobile platform only to discover that they are inaccessible because the plug-ins for them are out-of-date, unreliable, or missing.

Finally, it is important to recognize that hybrid is not the ultimate and only approach for mobile applications, but for sure the path for web developer savvies.

1. AngularJS



Angularjs is a client-side JavaScript Framework for adding interactivity to HTML. It is the most known framework for building “one page” application.

The official website of AngularJS angularjs.org, contains valuable resources and all necessary documentation to learn, to develop, and to master this rich framework.

In the following few pages, I will list some rudimental notions of AngularJs.

a. Directive

Directives are HTML annotations that trigger specific AngularJs behaviors.

AngularJS provides a ton of built-in directives for different usage and you can also create your own directive if you want to.

I will list some of the used directive in my project, and I am sure you will get the idea behind it:

- ng-app: Attach the application Module to HTML page.
- ng-controller: Attach a controller to HTML page.
- ng-model: Binds the form element value to a property.
- ng-show, ng-hide: Display or hide a section based on an Expression.
- ng-repeat: Repeat a section for each item in an array.
- ng-src: specify the source for HTML elements.
- ng-click: Trigger a function when the HTML element is clicked.
- ng-init: Allows to evaluate an expression in the current scope.
- ng-class: Append the class name if the expression is validated.
- ng-submit: Trigger a function when form is submitted.
- novalidate: Turn off default HTML validation used in <form>
- required: Mark required fields.

To match the directives to HTML element, it suffice to:

<code><input ng-model="foo"></code>
HTML code

I will be using them through the other components of AngularJs, since they are, as you can see, the linkers between JavaScript and HTML code.

b. Module

A module is the container for the different parts of your app. It is the main method that wires together the different parts of the app. It is also a way to make your code more maintainable, testable, and readable.

What I meant is you can write your JavaScript code into multiple files, and then use them in one file, using what called dependencies injection.

As you write your functions and procedures in your JavaScript files, you link them to your HTML page with the directive `ng-app`.

Let's see how to use a module:

```
var app = angular.module('store', []);  
// angular: AngularJS object  
// 'store': Application name  
// []: List of Dependencies
```

JavaScript File

```
<div ng-app="store">  
  <p>  
    {{ "Hello" + "You" }}  
  </p>  
</div>
```

HTML File

And for the dependency injection:

```
var app = angular.module('store1', []);
```

JavaScript File

```
var app = angular.module('store2', ['store1']);
```

JavaScript File

The code is self explanatory, the function of 'store1' can be used in the 'store2'.

c. Controller

Angular controllers have the same notion as Objects.

After linking a controller to a HTML element `ng-controller` with directive, and when that section get rendered or invoked by the app, an object will be created and will contain all variables and function defined in that controller.

```
app.controller('StoreController', function() {  
    var gem = {  
        name: 'Ruby',  
        price: 2.95,  
        description: 'Shiny and Pretty'  
    };  
});
```

JavaScript File

```
<div ng-controller="StoreController as store">  
    <h1>{{store.gem.name}}</h1>  
    <h2>${{store.gem.price}}</h2>  
    <p>{{store.gem.description}}</p>  
</div>
```

HTML File

Beware of the scope of your controllers.

If you try to print a value outside the `<div>` that contains the controller, it will never work.

As I was developing the app, I created multiple controllers, to separate the app's sections. Since not all the sections are similar, dividing your app

into multiple controllers will keep your code easier to maintain and to improve upon it in the future.

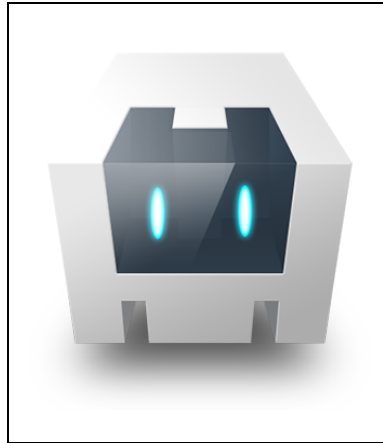
d. Other

AngularJs framework has too many functionalities and features to mention in my internship report.

I will just list the other important features that I used in this project, and you can find them very clear in their documentation:

- Services
- Expressions
- Filters
- Forms
- Models

2. Apache Cordova



Cordova, formerly called as “Phone Gap” is a platform to build mobile applications using web technologies.

It is responsible for making the magic happen, it lets you use the native device functionality like:

- Camera
- GPS
- Accelerometer
- Contacts
- etc

The official website cordova.apache.org, provides good documentation and many reliable plug-ins

With more than 1500 plugins and their wide community, you will not need to write any plugin by yourself.

So, Cordova takes care of packaging the HTML app as a native app on your device.

But if you simply take an existing website and package it as a mobile app, you will literally have a mess rather than an app.

Ionic is the missing piece of the hole story. Do not worry it will be introduced to you in the next section.

3. Ionic



Ionic is a beautiful, open source framework that offers a library of mobile-optimized HTML, CSS and JS components for building native and progressive web apps.

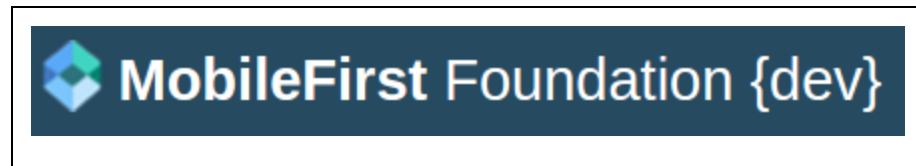
The core of Ionic is Apache Cordova, and it uses AngularJs as a way to handle its components, for this reason, you need to learn AngularJs and get use to it so you will be able to manipulate this framework easily.

To start playing with Ionic you need:

1. Install Node.js
2. Install the latest Cordova.
3. Install Android/iOS platform dependencies.
4. Start a project using one of the templates, or a blank page.
5. Develop the application.
6. Run it on the wanted platform or even your pc browser.

Go to ionicframework.com/docs, for all documentation and resources. With its rich documentation, Ionic is easy and fast to learn.

III. IBM MobileFirst:



IBM MobileFirst previously known as Worklight is an enterprise solution to develop, test, maintain, and execute your mobile application in different environments.

The official website ibm.com/mobilefirst contains all the necessary tutorials to understand and to be able to start working with MobileFirst.

It took me sometime to understand it and to start working efficiently with it, but as soon as I got the notion of it, I kind of enjoyed the fast pace and the consistency of MobileFirst.

What I have end up doing is downloading the MobileFirst Development Environment and installing it on top of eclipse.

It also have a CLI, but this way is more user friendly and easier to understand and use.

1. Architecture:

MobileFirst has its own architecture that is simple to understand.

MobileFirst comes with its his own **server**, the server is “*Websphere*” and you do not have to configure anything unless you are switching your server or exporting it somewhere else your localhost.

When you preview or run the application the server will be started automatically.

MobileFirst has two great features that I have used in my project, and I will go through the both of them in the next few page:

- Adapters
- Authentication

2. Adapters

Adapters are a key feature of the MobileFirst.

They are used to transfer and retrieve information from back-end systems to client applications and cloud services.

Each adapter consists of:

- XML file:

Describes the connectivity options and lists the procedures that are exposed to the application or other adapters.

- JS file:

Contains the implementation of procedures that are declared in the XML file.

- XSL files (they are optional):

Contain a transformation scheme for retrieved raw XML data.

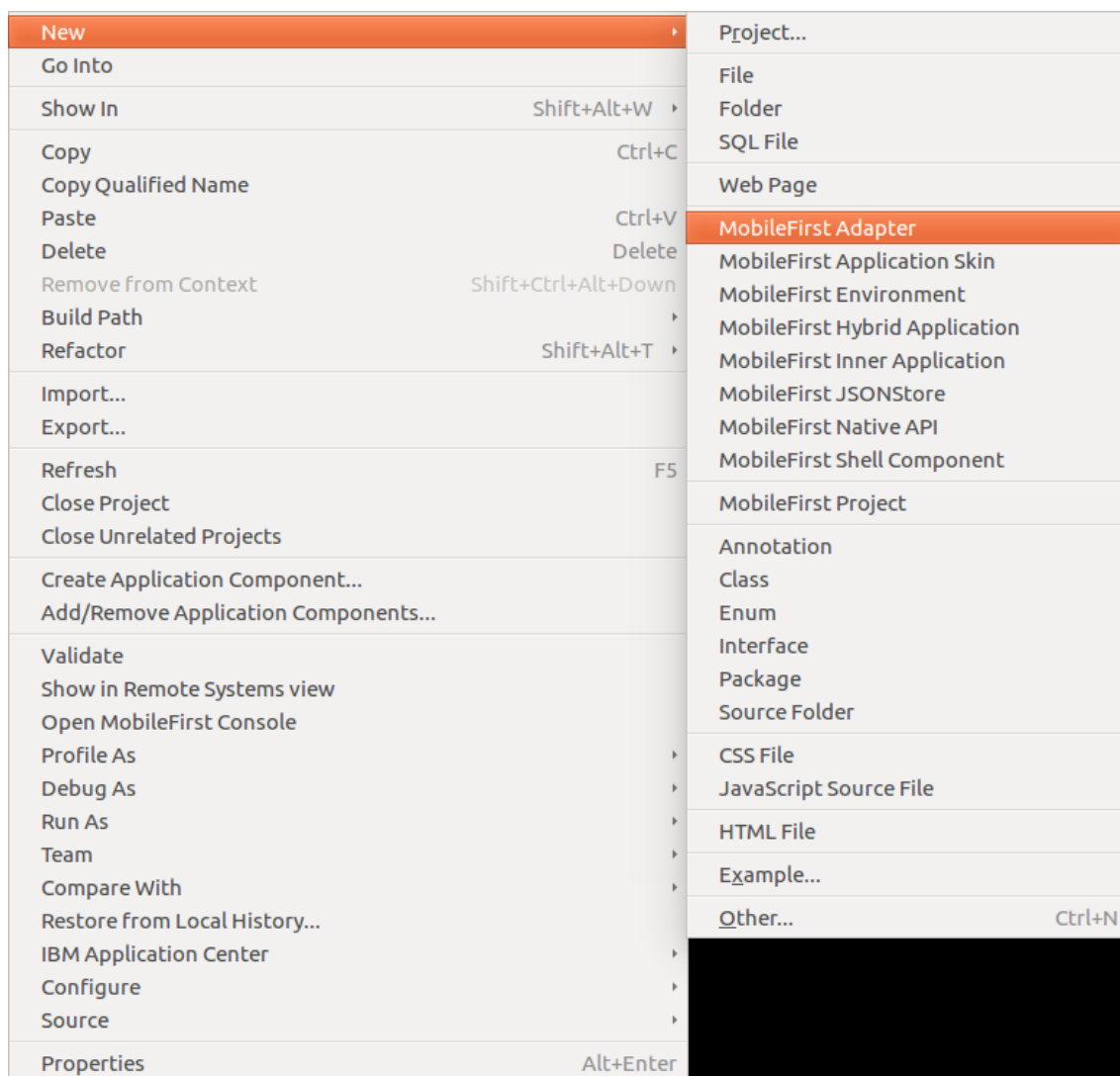
Data that is retrieved by an adapter can be returned raw or preprocessed by the adapter itself.

In either case, it is presented to the application as JSON object.

a. Create an adapter

Creating an adapter is made easy with the GUI on eclipse, all you need to do is to follow this few steps:

- i. Right-click on your MobileFirst project
- ii. Left-click on New> MobileFirst Adapter



iii. In the new popued window; select the type of the Adapter, and enter a valid name.

New MobileFirst Adapter

MobileFirst Adapter
Create a new adapter.

Project name: FirstNewsFirst

Adapter type: HTTP Adapter

Adapter name: TutoAdapter

☐ Create procedures for offline JSONStore

Retrieve JSON data with:

Add JSON data with:

Replace JSON data with:

Remove JSON data with:

☐ Create procedures for USSD enablement

Cancel Finish

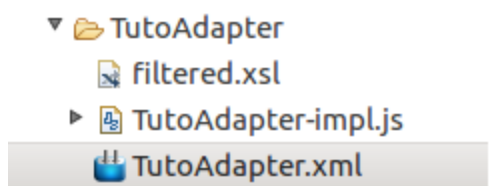
You will find a list of Adapter types, there will be seven of them.
What I have used through my project are *HTTP Adapter* and *SQL Adapter* and therefore I will explain only those two.

b. HTTP Adapter

HTTP adapters work with RESTful and SOAP-based services and can read structured HTTP sources such as RSS feeds.

You can easily customize it with simple server-side JavaScript code.

After the discussed above steps, you will end up with this new created folder that contains the 3 files:



Let's start by having a look on the xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Licensed Materials - Property of IBM
5725-I43 (C) Copyright IBM Corp. 2011, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<wl:adapter name="TutoAdapter"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wl="http://www.ibm.com/mfp/integration"
  xmlns:http="http://www.ibm.com/mfp/integration/http">

  <displayName>TutoAdapter</displayName>
  <description>TutoAdapter</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>https</protocol>
      <domain>developer.ibm.com</domain>
      <port>443</port>
      <connectionTimeoutInMilliseconds>30000</connectionTimeoutInMilliseconds>
      <socketTimeoutInMilliseconds>30000</socketTimeoutInMilliseconds>
      <maxConcurrentConnectionsPerNode>50</maxConcurrentConnectionsPerNode>
      <!-- Following properties used by adapter's key manager for choosing specific certificate from key store
      <sslCertificateAlias></sslCertificateAlias>
      <sslCertificatePassword></sslCertificatePassword>
      -->
    </connectionPolicy>
  </connectivity>

  <procedure name="getFeed"/>
  <procedure name="getFeedFiltered"/>
</wl:adapter>
```

I will explain the most important elements that you need to understand so you will be able to configure the adapter.

XML element	Usage
name	Mandatory. Name of the adapter that will be used in the invocation of its procedures.
displayName description	Optional. Additional information about the adapter.
connectivity	Mandatory. Defines the connection properties and load constraints of the back-end system.
procedure	Declares the name of an adapter's procedure. One entry for each adapter's procedure.

I have listed In the above table, the most important XML element for configuring *HTTP Adapter*.

Our work is not finished yet, we now need to configure the element inside the ***connectivity*** XML element.

connectivity elements	Usage
domain	Sets the URL of the back-end system.
protocol	Sets the protocol of specified URL. <ul style="list-style-type: none">● http

	<ul style="list-style-type: none"> ● https
port	Sets the port of specified protocol: <ul style="list-style-type: none"> ● 80 for http ● 443 for https
Rest of elements	The rest of element in the connectivity XML element, is configured for connection timeout and maximum connection nodes.

Now to the JavaScript file; I will display an example of JavaScript implementation:

```
function getFeeds() {
    var input = {
        method: 'get',
        returnedContentType: 'xml',
        path: 'rss.xml',
        parameters: {
        }
    };

    return WL.Server.invokeHttp(input);
}
```

JavaScript implementation file

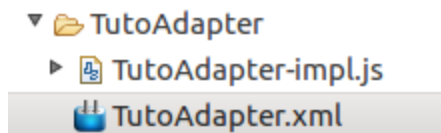
The returnedContentType can be JSON, XML, or even HTML.

And that's pretty much all you have to know to use *HTTP Adapter*.
Let's move to the second types of adapters, *SQL Adapter*.

c. SQL Adapter

The second type of adapters I have used in my project is SQL adapter. SQL adapter, is a way to connect to your database - wherever it is - and to execute any query you want to that database.

If you selected SQL Adapter, when creating a new adapter, you will end up with this new folder:



As you can see, there is not XSL file, since we are not handling any XML raw data.

Let's have a look on the XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Licensed Materials - Property of IBM
5725-I43 (C) Copyright IBM Corp. 2011, 2013. All Rights Reserved.
US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<wl:adapter name="TutoAdapter"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wl="http://www.ibm.com/mfp/integration"
  xmlns:sql="http://www.ibm.com/mfp/integration/sql">
  <displayName>TutoAdapter</displayName>
  <description>TutoAdapter</description>
  <connectivity>
    <connectionPolicy xsi:type="sql:SQLConnectionPolicy">
      <!-- Example for using a JNDI data source, replace with actual data source name -->
      <!-- <dataSourceJNDIName>java:/data-source-jndi-name</dataSourceJNDIName> -->

      <!-- Example for using MySQL connector, do not forget to put the MySQL connector library in the project's lib folder -->
      <dataSourceDefinition>
        <driverClass>com.mysql.jdbc.Driver</driverClass>
        <url>jdbc:mysql://localhost:3306/mydb</url>
        <user>myUsername</user>
        <password>myPassword</password>
      </dataSourceDefinition>
    </connectionPolicy>
  </connectivity>

  <!-- Replace this with appropriate procedures -->
  <procedure name="procedure1"/>
  <procedure name="procedure2"/>
</wl:adapter>
```

Let's directly see how the elements of connectivity change, since the other main elements remained the same.

connectivity elements	Usage
driverClass	According to your database, you need to indicate the right driver class for your database. By default, the driver is for MySQL database. You need to include the driver's jar under <i>server>lib</i> . Go to sql-workbench.net/manual/jdbc-setup for a list of drivers and their jars.
url	The URL of your database. It can be in localhost or in a remote place.
user	Specify the username of the database.
password	Specify the password for the username.

Now to the JavaScript implementation file:

```
var preparedState =
    WL.Server.createStatement(
        "SELECT * FROM table_name WHERE id=?;"
    );
function executeQuery(id) {
    return WL.Server.invokeSQLStatement({
        preparedStatement: preparedState,
        parameters: [id]
    });
}
```

JavaScript implementation file

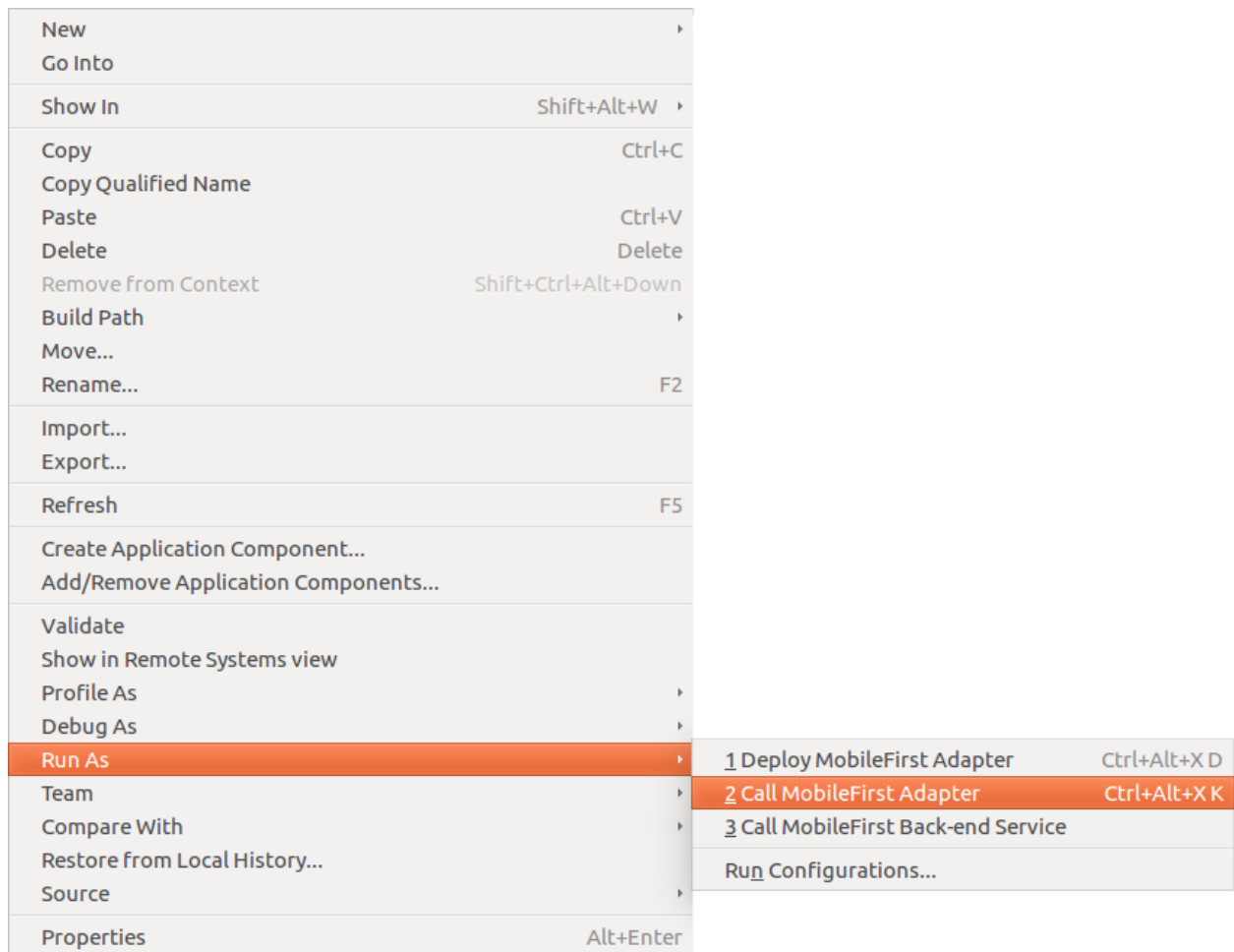
d. Test and Invoke Adapters:

What's the purpose of creating an adapter without knowing how to use it.

Test is a crucial step in development, it makes sure that your reasoning and your code is correct and ready to go.

Thus, let's test our adapter's procedure before invoking it:

Right-click on your adapter's folder, then:

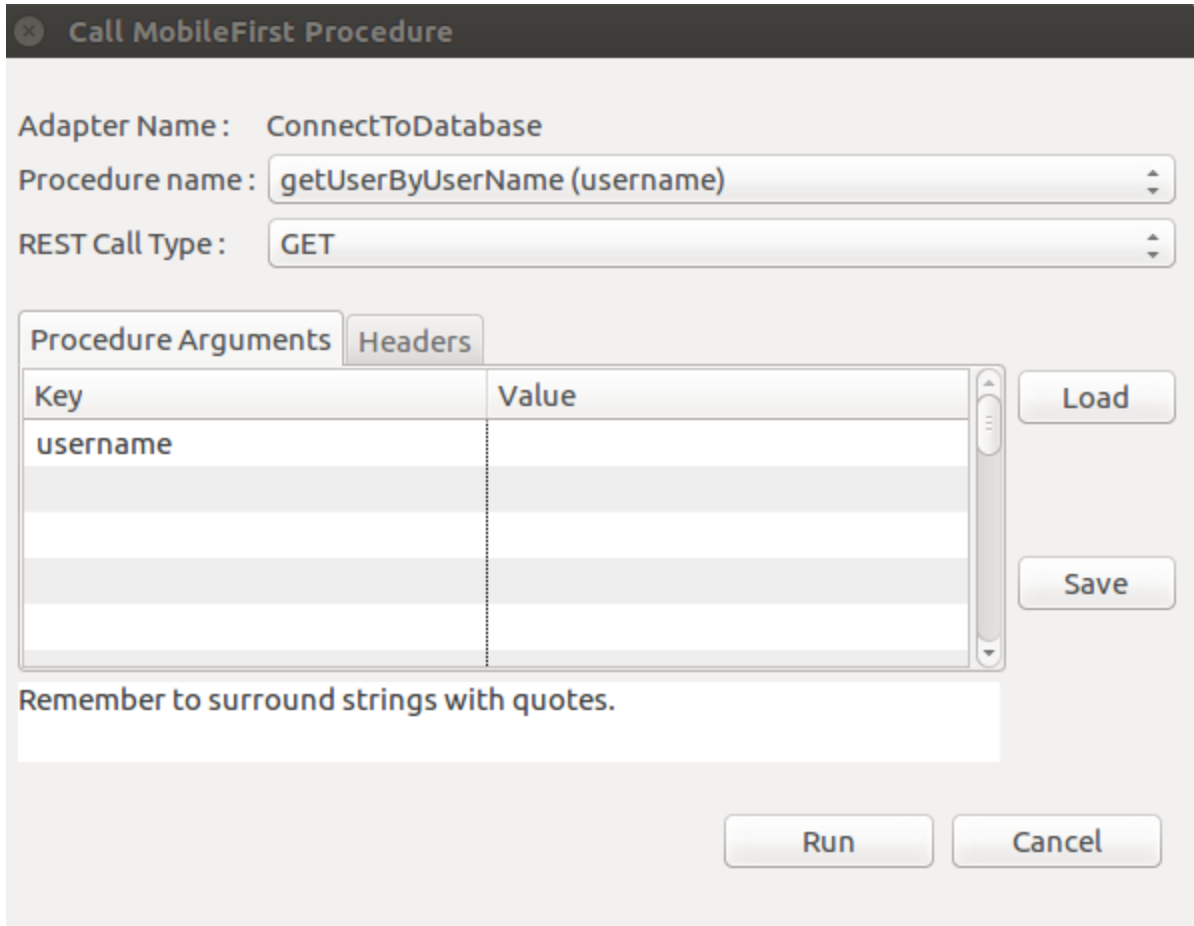


A new window will appear after clicking on that menu.

You will select the procedure that you want to test, type any parameters the procedure needs to be executed and hit *Run*.

Easy, right?

What we have in the following image, is a test of a procedure of one of the adapters used in my project.



Call MobileFirst Procedure

Adapter Name: ConnectToDatabase

Procedure name: getUserByUserName (username)

REST Call Type: GET

Procedure Arguments Headers

Key	Value
username	

Load

Save

Remember to surround strings with quotes.

Run Cancel

As you can see, we have *Adapter Name*, *Procedure name*, and *Procedure Arguments* three main fields you need to adjust for your call.

After done testing your adapter's procedures, you need to call them in your application code.

Basically, we will either invoke the adapter's function in the client-side or in the server-side.

```
// parameters: list of parameters that the procedure
// needs to be executed
var invocationDataConfig = {
    adapter: 'nameOfAdapter',
    procedure: 'nameOfProcedure',
    parameters: []
};

// successFunction: success callback of the procedure
// failureFunction: failure callback of the procedure
WL.Client.invokeProcedure(invocationDataConfig, {
    onSuccess: successFunction,
    onFailure: failureFunction
});
```

Client-side JavaScript call

As you can see, this adapter call will work Asynchronously.

```
function callAdapterProcedure(parm) {
    return WL.Server.invokeProcedure({
        adapter: 'nameOfAdapter',
        procedure: 'nameOfProcedure',
        Parameters: [parm]
    });
}
```

Server-side JavaScript call

This adapter call will work Synchronously.

I have personally used the *server-side adapter call* in other adapters.

3. Authentication

Another feature MobileFirst have is **Authentication Realms**.

It is a powerful security measurement, that can be used to protect all MobileFirst entities:

- Applications
- Adapter procedures
- Static resources

Each realm consist of:

- One Authenticator
- One Login Module

And they are both server-side components.

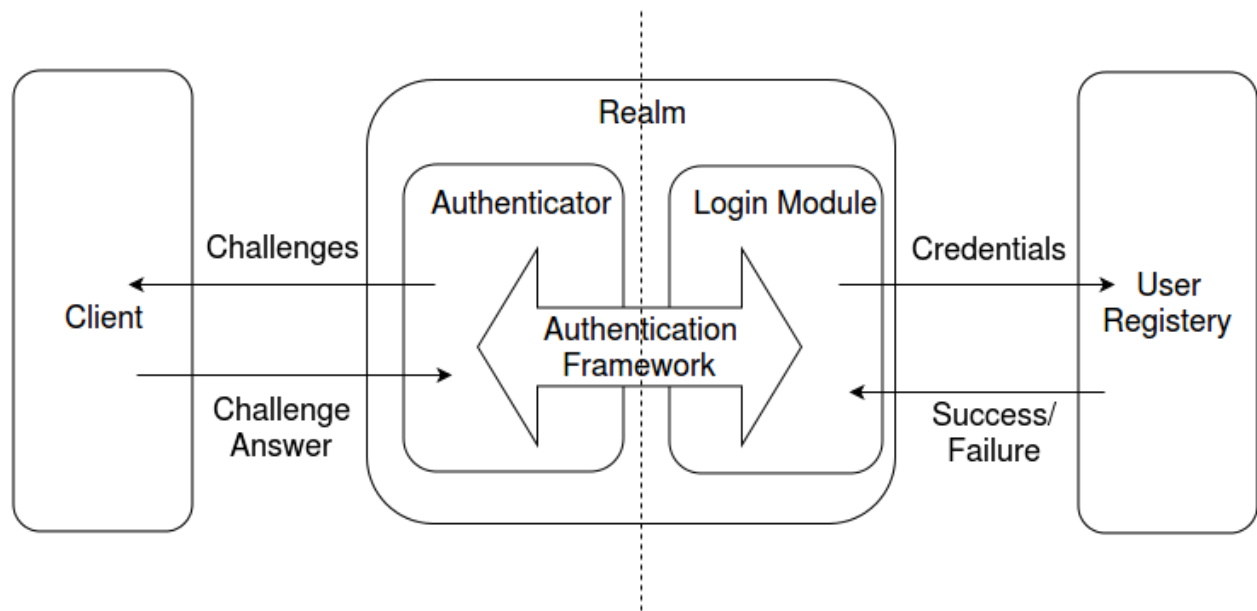
And each realm requires a **challenge Handler** on the client-side.

Wow, all of these concepts...

I know, it can be daunting at first, but I will try to explain some of it.

a. Concept of MobileFirst Authentication

Before start explaining, look at this following scheme, it will help you understand the hole idea.



- Authentication Realm:

Each authentication realm defines its authentication flow.

Each realm must have a corresponding challenge handler in the client application.

- Challenge Handler:

A challenge handler is a client-side entity that controls a specific challenge.

It is used to detect the authentication challenges of the server and handle them.

After a challenge handler detects an authentication challenge that is returned from the server, it is responsible for collecting the required credentials and for sending them back to the server.

- Authenticator:

It is a server-side entity that is responsible for collecting the credentials from the client application, then sending them to Login Module.

Mobile first comes with different types of authenticators:

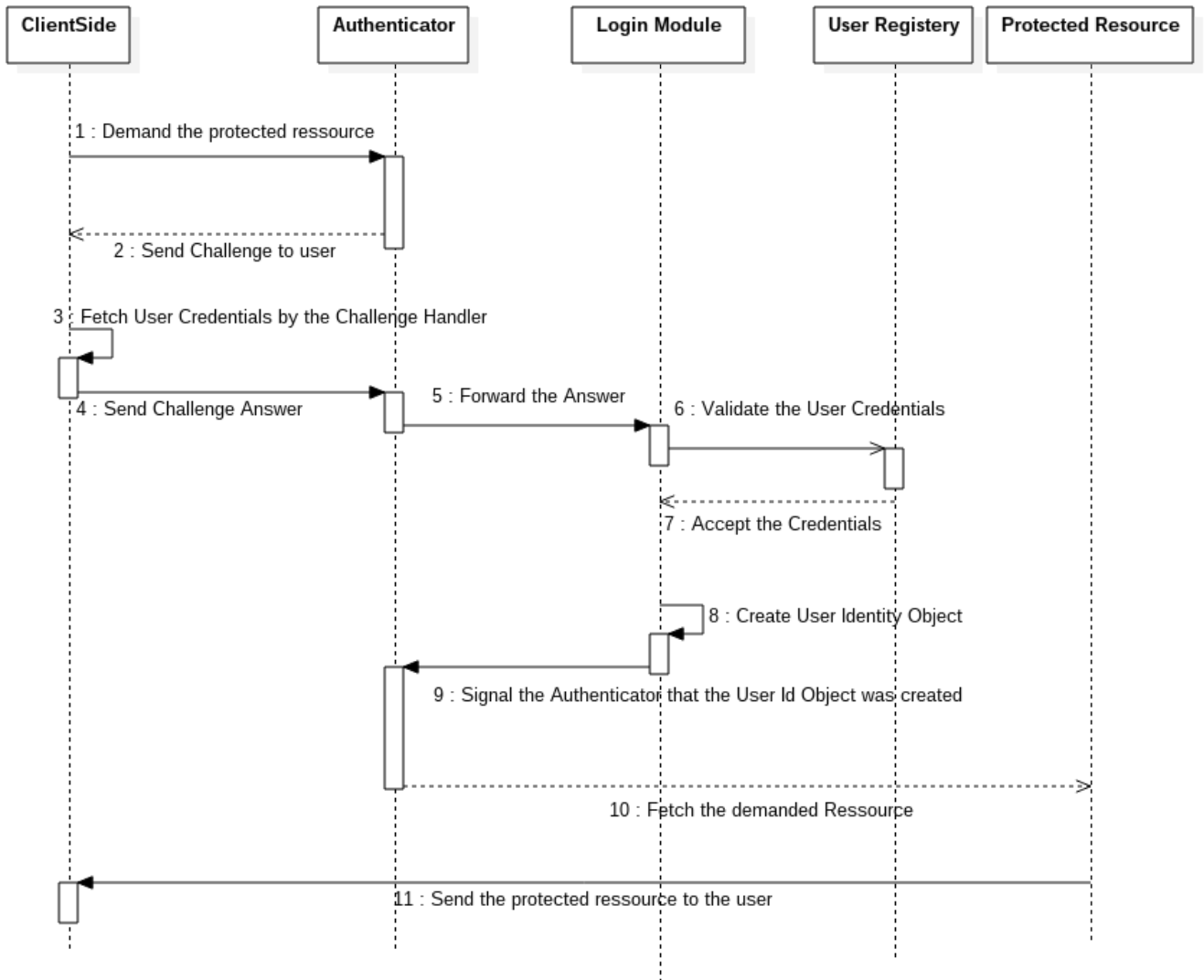
- Form-based Authenticator
- Adapter-based Authenticator
- Header-based Authenticator
- Custom Authenticator

- Login Module:

It is a server-side entity that is responsible for verifying the user credentials and for creating a user identity object, which holds the user properties for the remainder of the session.

A login module destroys the user identity object when the authenticated session terminates (logout or timeout).

The next scheme will be a sequence diagram of the routine done when a client request a protected resource; it is self explanatory:



b. Configuring the Authentication

Luckily all the settings are configured in *server\conf\authenticationConfig.xml* file. So, all we have to do is to open that XML file and add corresponding elements:

```
<realms>
  <realm name="SampleAppRealm" loginModule="StrongDummy">
    <className>com.worklight.core.auth.ext.FormBasedAuthentication</className>
    <realm>
      ...
    </realm>
  </realms>
```

Defining Realm

```
<loginModules>
  <loginModule name="StrongDummy">
    <className>com.worklight.core.auth.ext.NonValidLoginModule</className>
    </loginModule>
    ...
  </loginModules>
```

Defining Login Module

After you set up authentication realms, you need to define the security test to be used.

MF comes with three types of tests that can be defined:

- **webSecurityTest:**
Test that enables default web security-related realms.

```
<webSecurityTest name="SampleWebSecurityTest">
  <testUser realm="SampleRealm" />
</webSecurityTest>
```

- mobileSecurityTest:

Test that enables default mobile security-related realms.

```
<mobileSecurityTest name="SampleMobileSecurityTest">
  <testUser realm="SampleRealm">
</mobileSecurityTest>
```

- customSecurityTest:

Does not contain any default realms.

```
<customSecurityTest name="SampleCustomSecurityTest">
  <test realm="SampleRealm1" step="1" />
  <test realm="SampleRealm2" step="2" />
  <test realm="SampleRealm3" step="3" />
</customSecurityTest>
```

As you can see, a security test can include more than one *Realm*.

You can define which realm to be used as a user realm by

`isInternalUserId="true"`.

4. File Structure of MobileFirst:

MobileFirst has its own file organization that is worth mentioning.

I will go through its file structure, you can guide yourself with the image I have included of my own project.

- adapters

This folder is where all your created adapters live.

It will contain several folders, each for every adapter you have created for your project.

- apps

This folder will contain your created apps.

You can have more than one app in your MobileFirst Project.

As you can see I have only one, called “*News_app*”.

Under my main application, I have multiple files and folders, the most important one, or at least the one I used is:

- common

This folder will contain the main parts of your application.

If you noticed, the folder contains as many folders as a website:

- css: folder for your .css files.
- images: folder for your images resources.
- js: folder for your JavaScript files.
- lib: folder for all used libraries.
- templates: folder for all your .html files.
- index.html: the main file that will load when your app gets opened.










































- server

This folder will basically contain all files required to configure the MobileFirst server.

- lib: This folder will contain any libraries that your server will use. For instance, it will contain any SQL drivers if you used a SQL adapter.
- conf: As its name suggests, this folder contains a bunch of configuration files.

The ones I have used are authenticationConfig.xml which is a file for authentication see and worklight.properties which is a file that can hold any wanted variable that will be used in your project.

For example, I used that file to keep my APIs' keys and URL paths.

- ▼  FirstNewsFirst
 - ▶  Java Resources
 - ▶  JavaScript Resources
 - ▼  adapters
 - ▶  ConnectToDatabase
 - ▶  FetchNews
 - ▶  GoogleLocation
 - ▶  JSDataAdapter
 - ▶  newsAPI
 - ▶  nyTimesAPI
 - ▶  OpenWeatherAPI
 - ▶  PasswordAdapter
 - ▶  SingleStepAuthAdapter
 - ▶  TheGuardianAPI
 - ▼  apps
 - ▼  News_app
 - ▶  android
 -  certificates
 - ▼  common
 - ▶  css
 - ▶  images
 - ▶  js
 - ▶  lib
 - ▶  templates
 -  index.html
 - ▶  common_bak
 - ▶  legal
 - ▶  mobilewebapp
 -  application-descriptor.xml
 -  build-settings.xml
 - ▶  bin
 - ▶  externalServerLibraries
 - ▼  server
 - ▼  conf
 -  authenticationConfig.xml
 -  login.html
 -  mfp-default.keystore
 -  SMSConfig.xml
 -  worklight.properties
 - ▶  java
 - ▶  lib

IV. Overview of the Application

The moment I have been waiting for, the application.

In this part, I will go through the application that I have been working on.

It has simple User Interface, and simple functionalities.

The application is a news application, that will fetch different news from different sources and print them to the user, he can also save them for future usage.

It will also have a weather section, that will detect the user location and display today weather as well as next 15 days.

The user have to create an account to be able to use fully the application.

The application is divided into six main pages plus the menu:

- i. Home Page
- ii. Sign up
- iii. Login
- iv. Saved News
- v. Search for News
- vi. Weather

Anyway, I hope you like it as much as I do.

1. The Database

Before going through the main pages, I will go through the database and the design I have implemented since it is the backbone of the application.

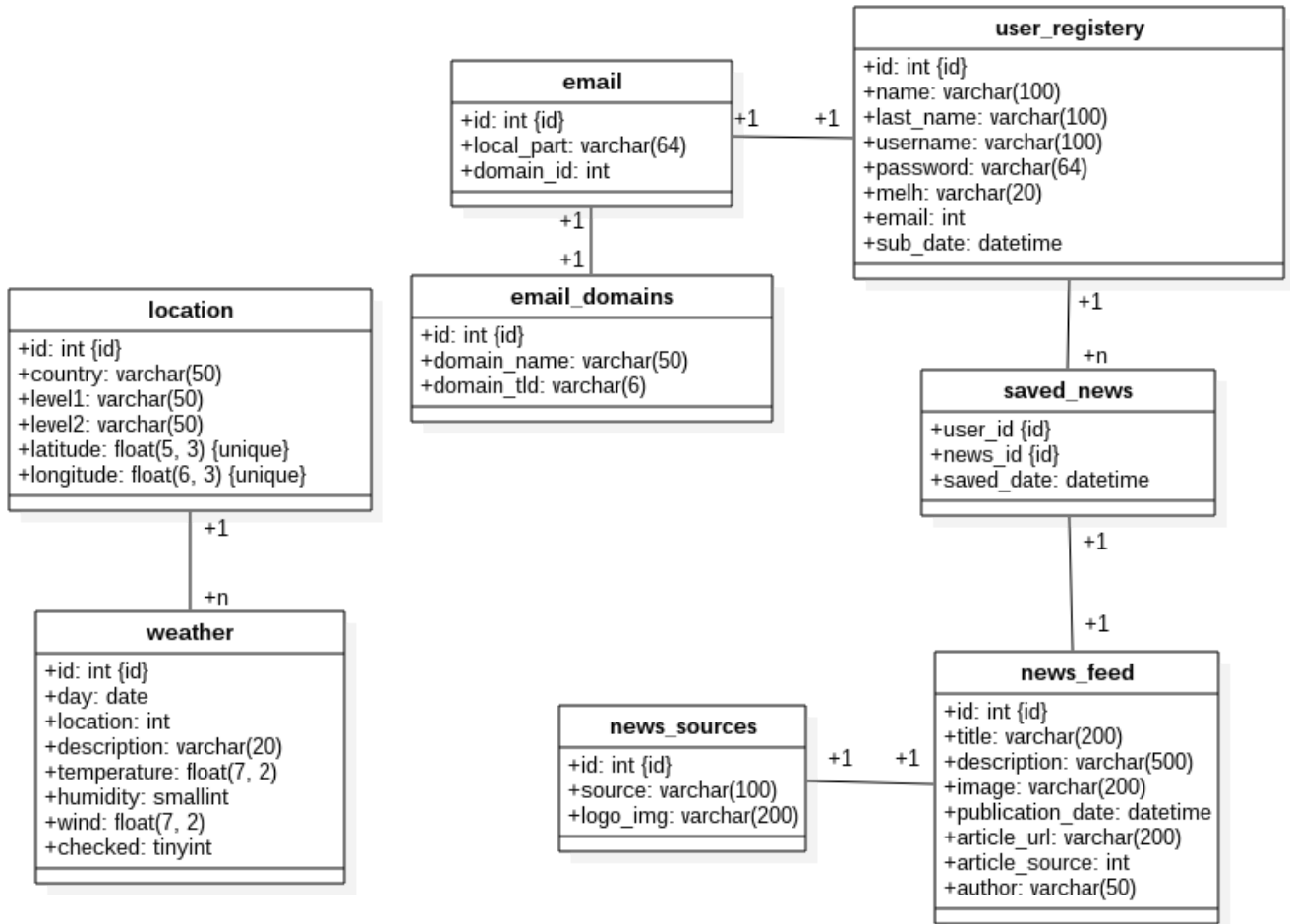
I have used a **MySQL** database on my localhost that I have connected to the application with a *SQL Adapter*.

I have also created multiple adapter's procedures that will execute different queries.

I will list all the tables that I have used in my database, as I will explain some of them and the design behind of it:

- user_registery
- email
- email_domains
- saved_news
- news_feed
- news_sources
- location
- weather

This is the whole database with all its tables and variables:



The **user_registry** table is responsible of storing data for users.

It will save a hashed password with SHA-256 function.

A random generated salt will be also saved for each user and used to determine the right password.

When a user sign up, he will enter an email that will be saved into **email** and **email_domains** tables and linked to in the **user_registry** table.

When a user save a news it will be saved in the **saved_news** table.
I also have a table for all **news_sources** that is used to determine the source of news that are saved in **news_feed**.

All the fetched news with different APIs will be saved obviously in **news_feed** table.

The **location** table will save the geo-coordinates as well as the country, level1 and level 2 administrative names.

This table is used to minimize the request upon the *Google API*.

The last mentioned table will be linked to **weather** table, that will contain data of weather for a specific location.

All these tables, you might wonder!

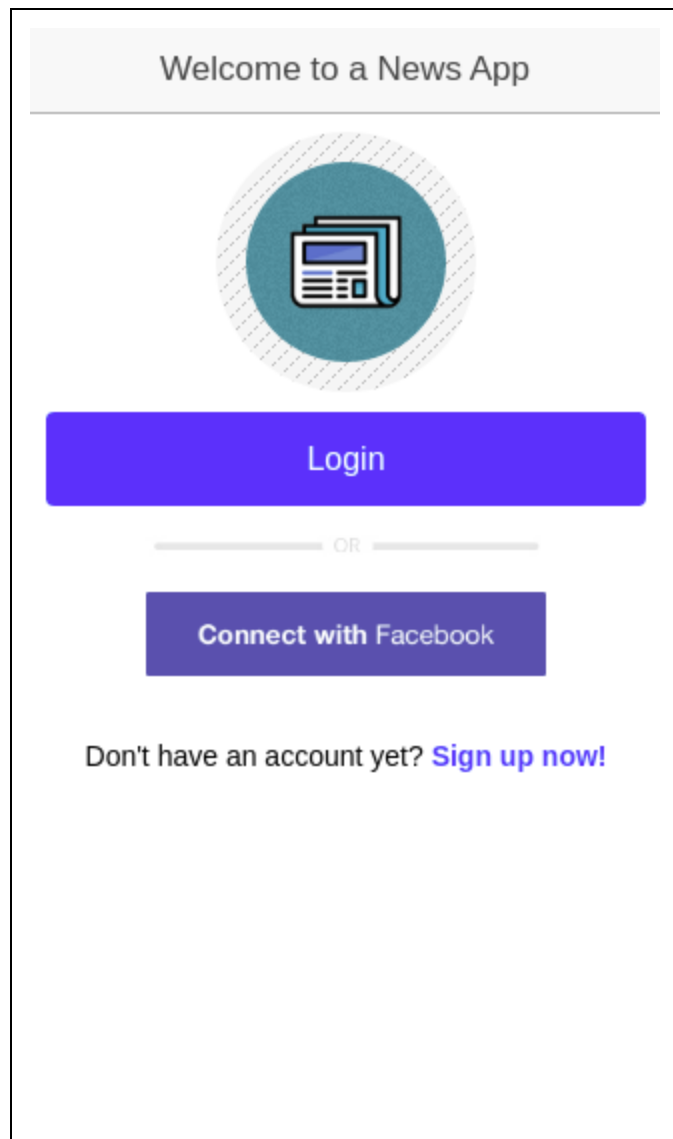
Let me explain myself.

Since I am using different APIs, and I only have an *API Keys* for a developer, the number of requests for any API will be limited.

Thus, I am using **news_feed** table to store all the data fetched from news APIs, **weather** table to store data received from *open weather API*, **location** table to store data received from *Google API*.

And now, you know why.

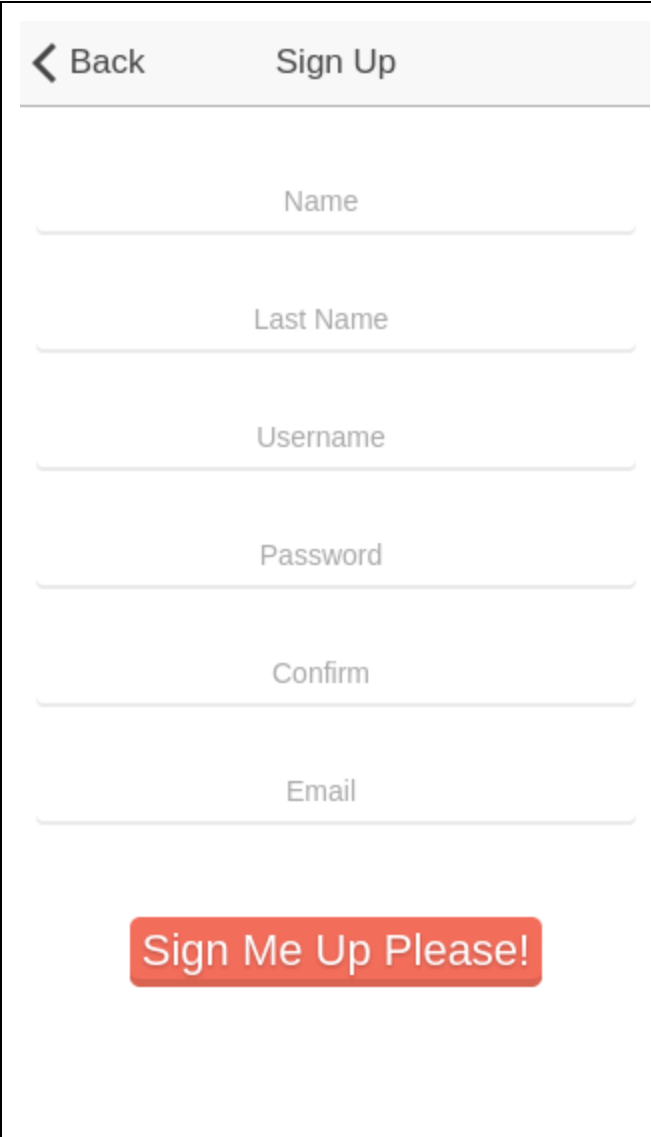
2. Home Page



The first page you will be able to see is this Login/Sign up page. It has the App logo and buttons to login and to sign up to the app.

The application is protected with an “*Adapter Based Authentication*”. So to be able to go inside the application you need to have an account and pass the security routine of MobileFirst Authentication process.

3. Sign Up



The image shows a mobile application interface for a sign-up page. At the top, there is a header bar with a back arrow and the text "Back" on the left, and "Sign Up" on the right. Below the header, there are six input fields, each with a label above it: "Name", "Last Name", "Username", "Password", "Confirm", and "Email". Each input field is represented by a horizontal line with rounded ends. At the bottom of the form, there is a red button with the text "Sign Me Up Please!" in white.

The Signup page will contain a form that the user is required to fill to be able to create an account.

The **Password** field must be confirmed in the **Confirm** input.

The password will be hashed before it will be saved in the table, and each time the user will login, the new entered password will also be hashed and then compared to that saved password.

The user must also specify an **Email** to be able to sign up.

4. Login

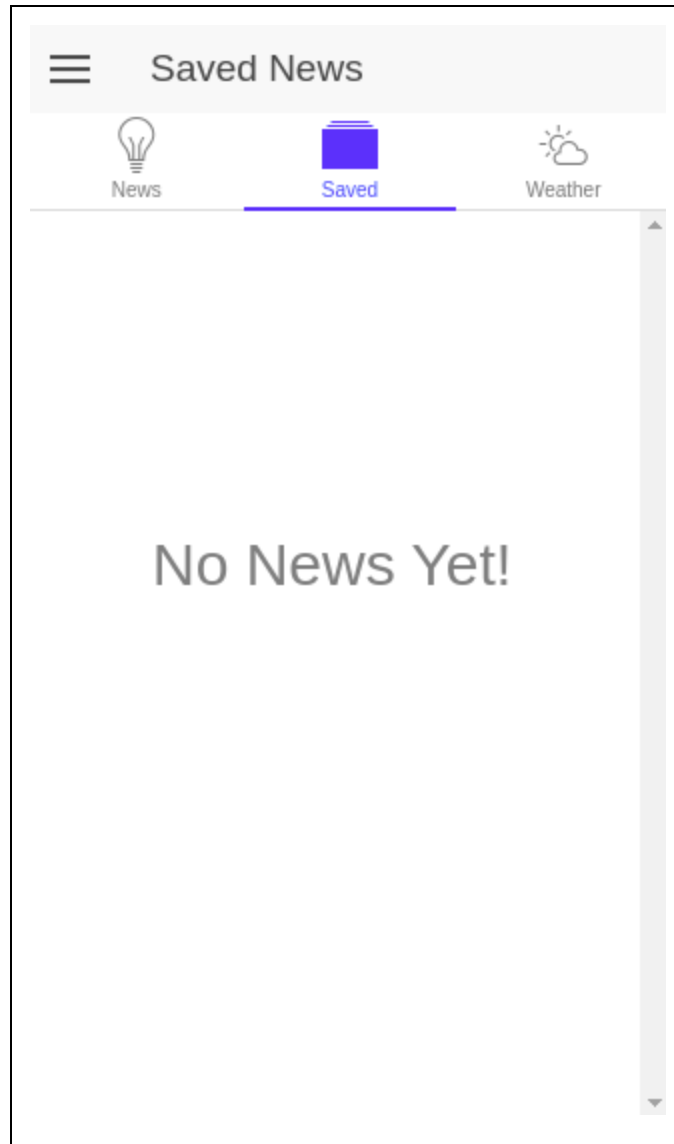
<div><div>< Back</div><div>Login</div></div> <div><div>What's your username?</div><div></div></div> <div><div>Enter your password</div><div></div></div> <div><div>Connect me to News!</div></div>	<div><div>< Back</div><div>Login</div></div> <div><div>root</div><div></div></div> <div><div>.... </div><div></div></div> <div><div>Connect me to News!</div></div>
---	--

Typical stuff right here.

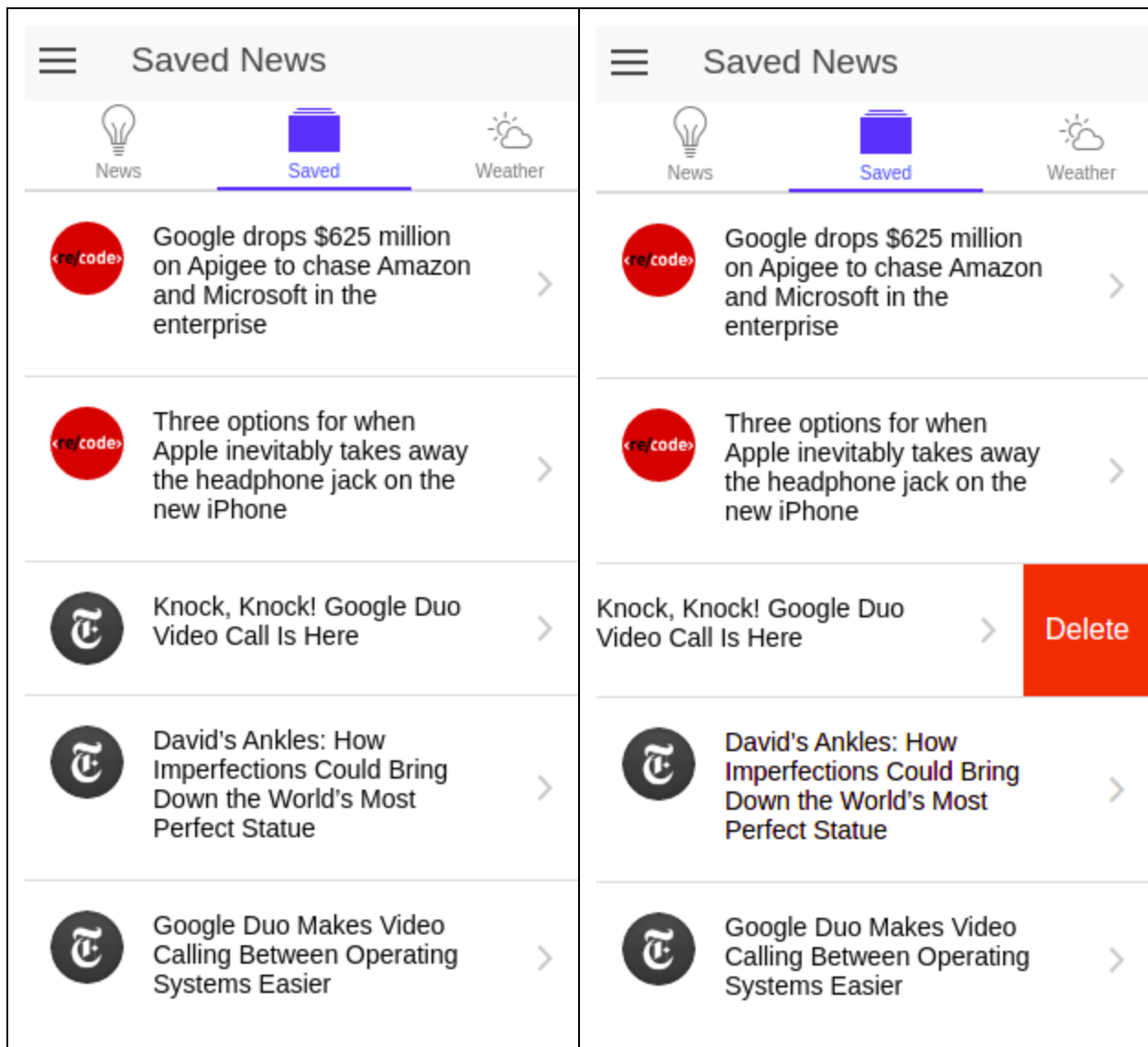
To be able to access the application you need to login with an account.

To login you need to enter your **username**, and a valid **password**.

5. Saved News



When you login for the first time, this is the page that will popup for you. It is the **Saved News** section, that will contain all you saved news but when you have no saved news it will just display “No News Yet!”.

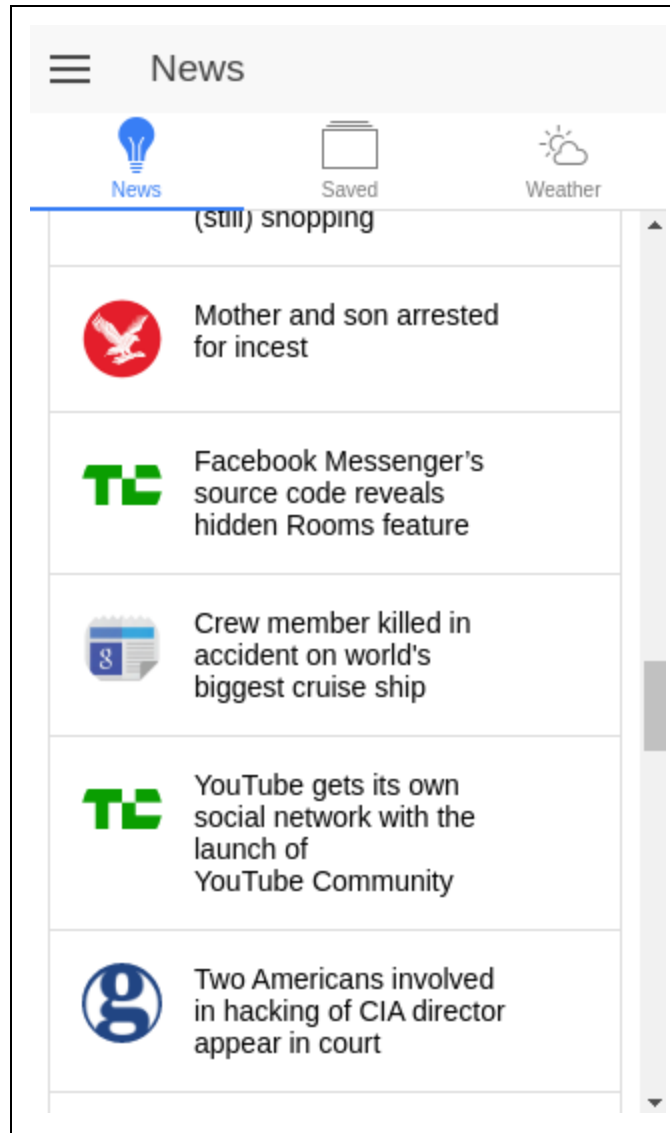


This is a view of a filled *Saved News* page.

As you can see, there is an option for deleting saved news.

When you click on a specific news, you will be able to read a brief description of that news, and if you want to read more you will be redirected to the article page on your **mobile browser**.

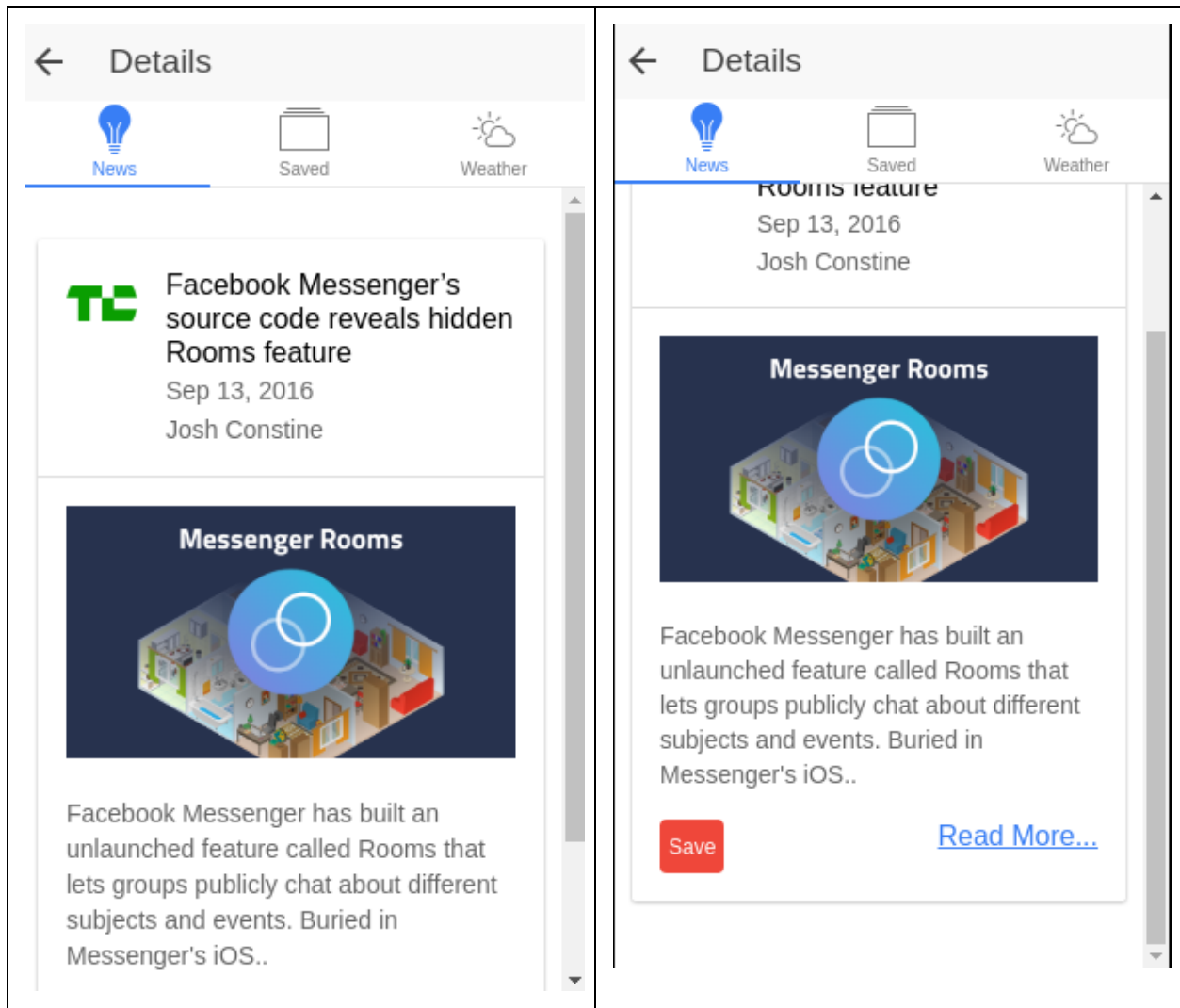
6. Search for News



This is the **News** page. It will contain different news from different sources and they all be sorted depending their publication date.

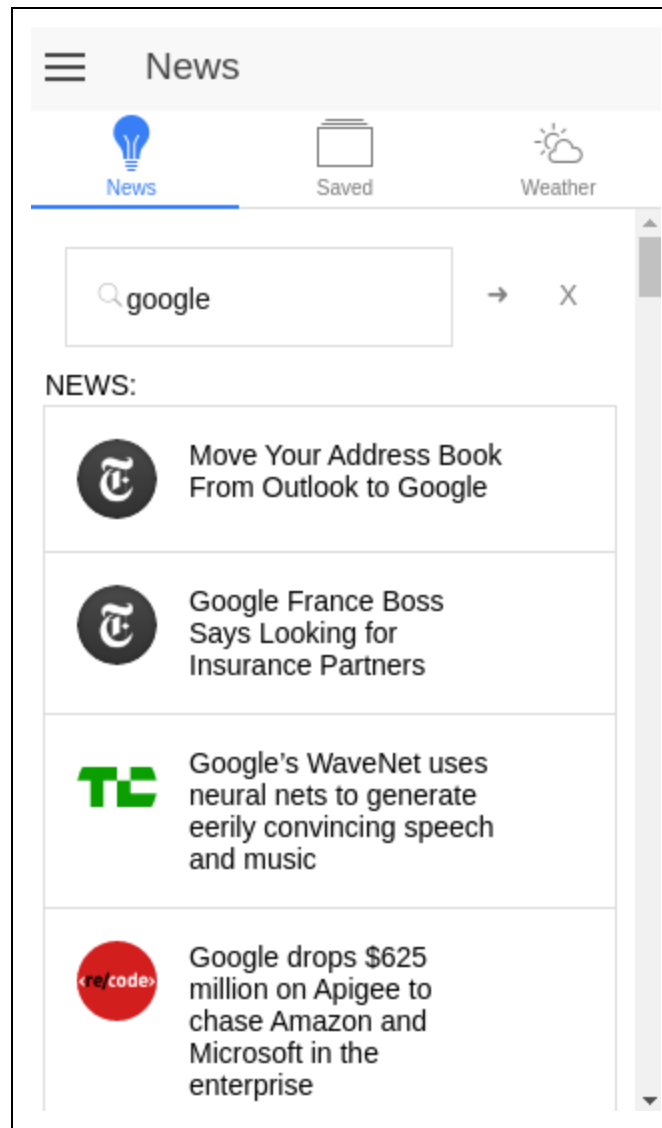
As long as the user scrolls down, the news will be added to the list due to **infinite scroll**.

So, when you click on a specific news, you will be redirected to the details page:

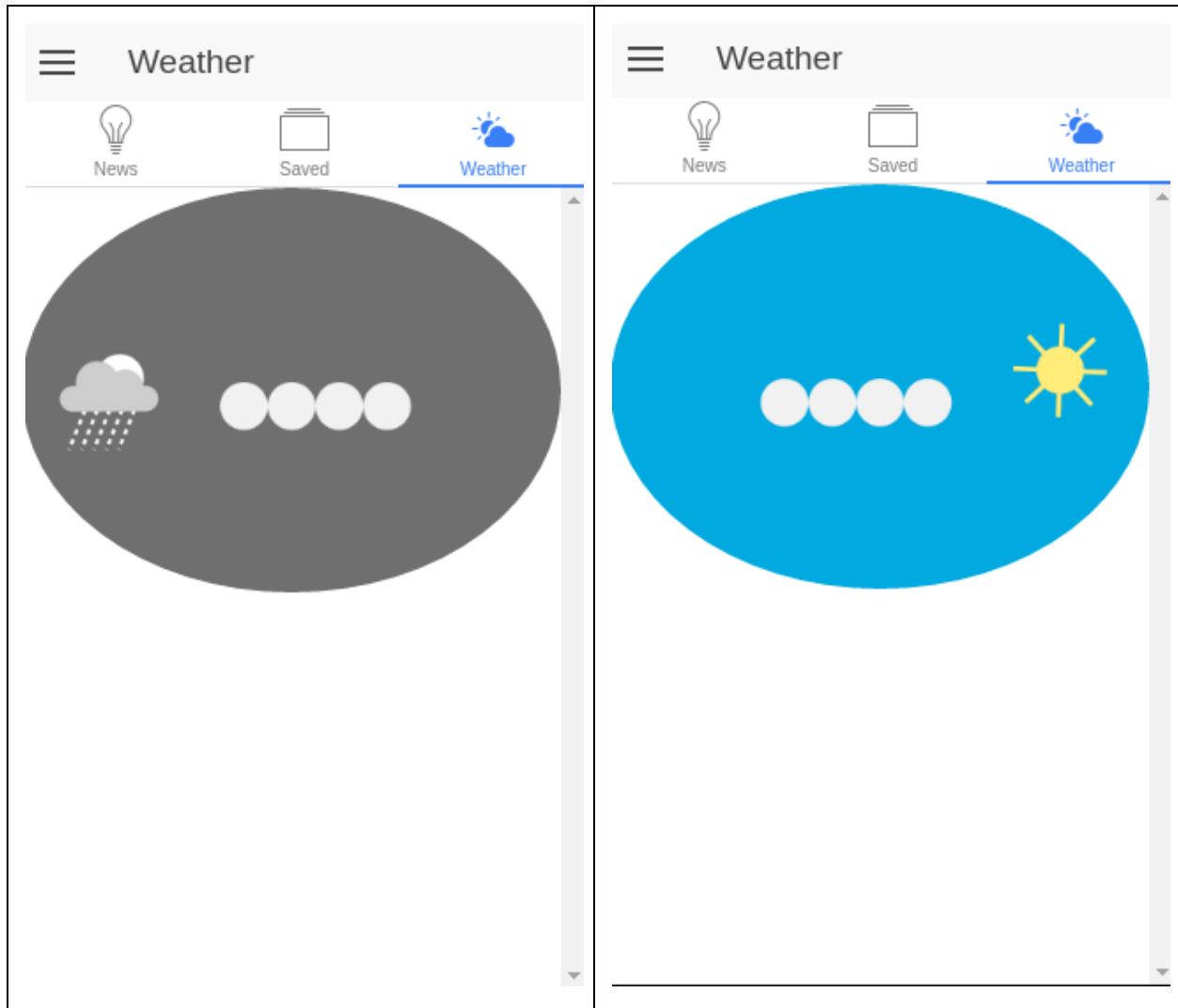


The user can be satisfied with this description or want to *Read More...*, and a browser linking to that article source will popup. He will have the option to *save* that news for future usage.

The user will also have the ability to search for specific terms in the search bar. And when he hits the 'x' button, he will be returned to the date ordered list.

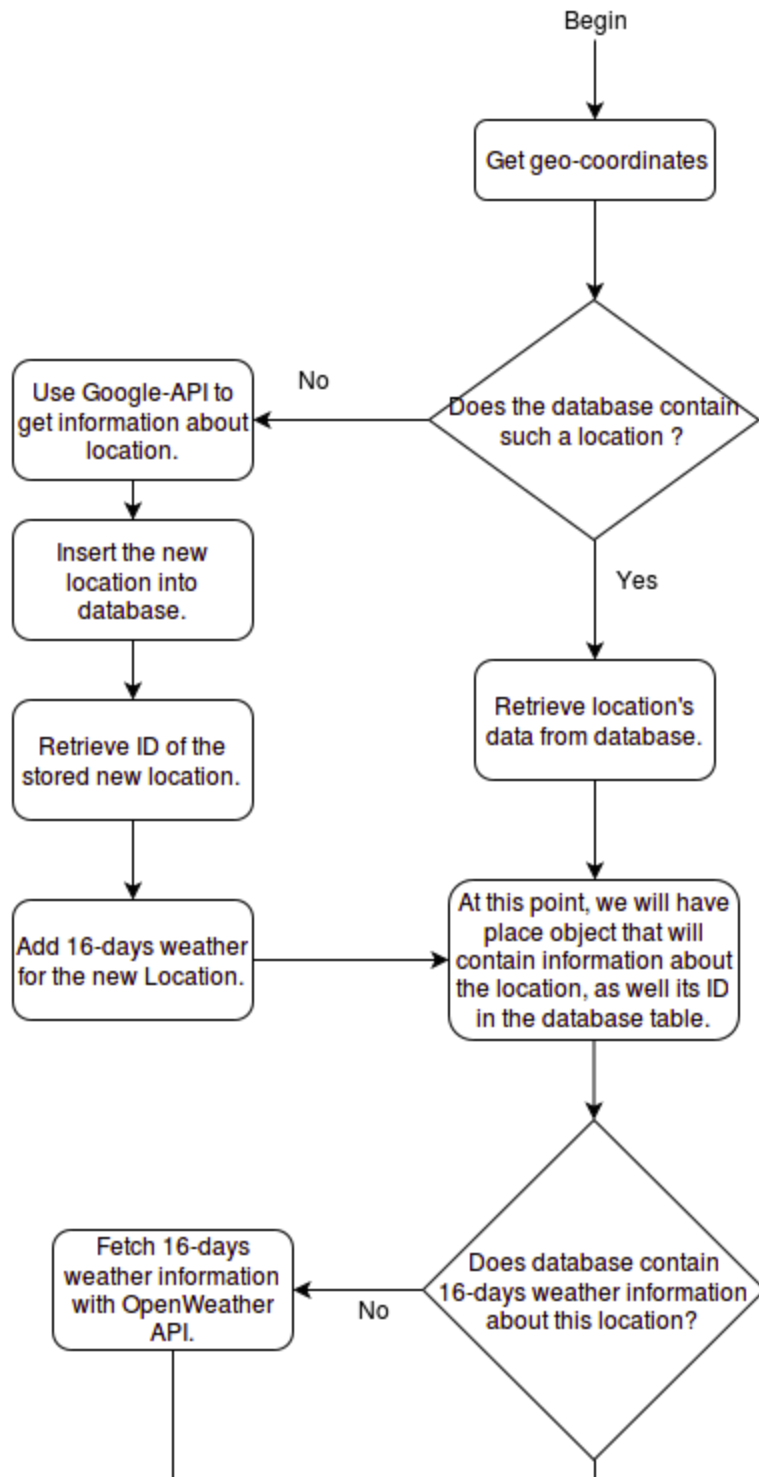


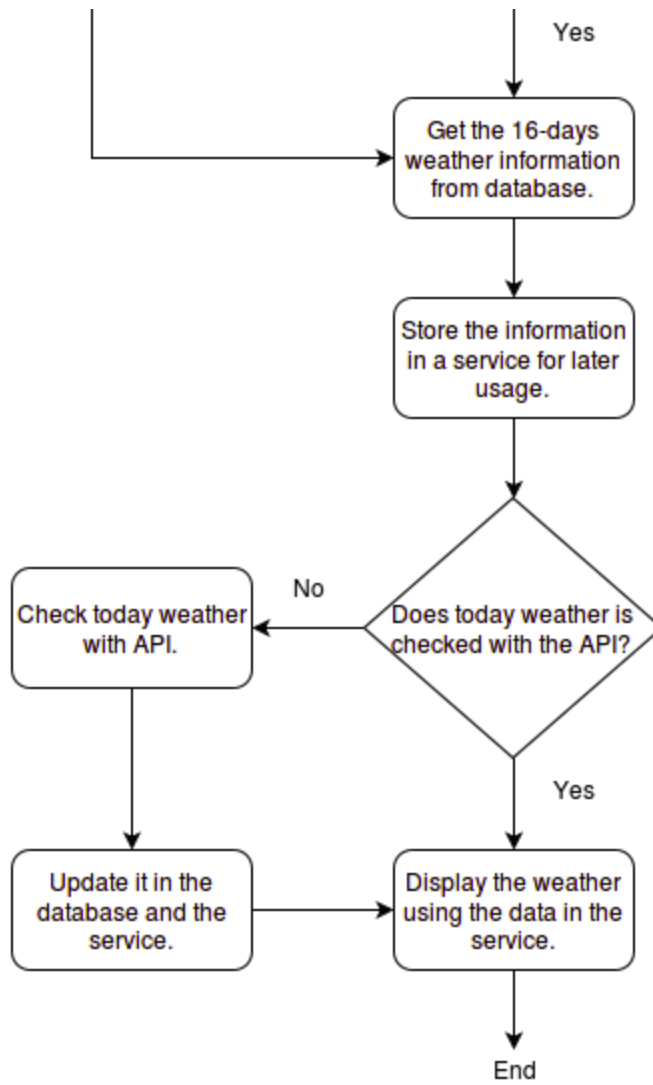
7. Weather



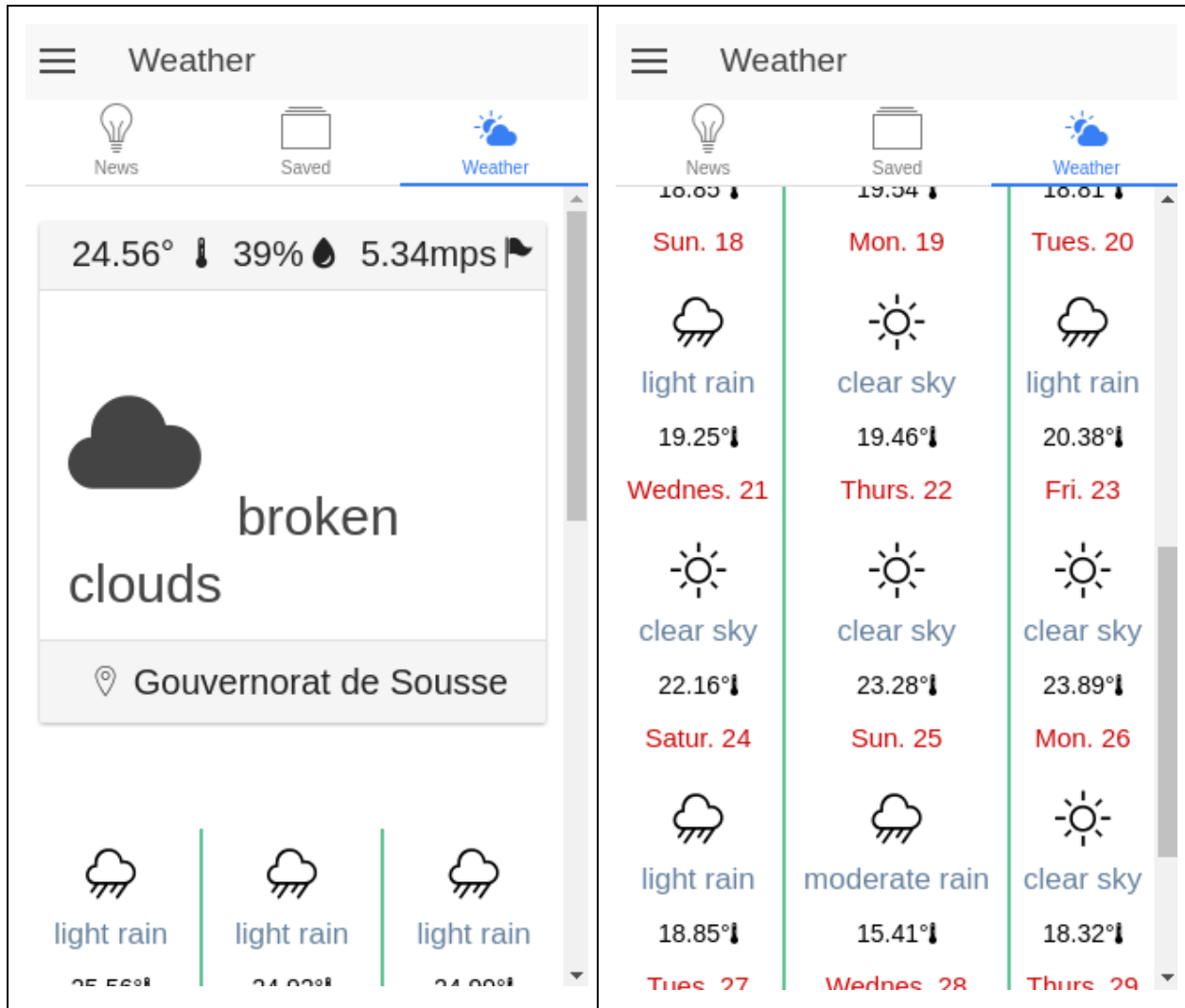
When you open the weather page, you will go through a specific algorithm that tries to minimize the request for all users, this is way you will see a little waiting display until the algorithm finish fetching data for a specific location.

This is the algorithm that the application will go through to display the weather:

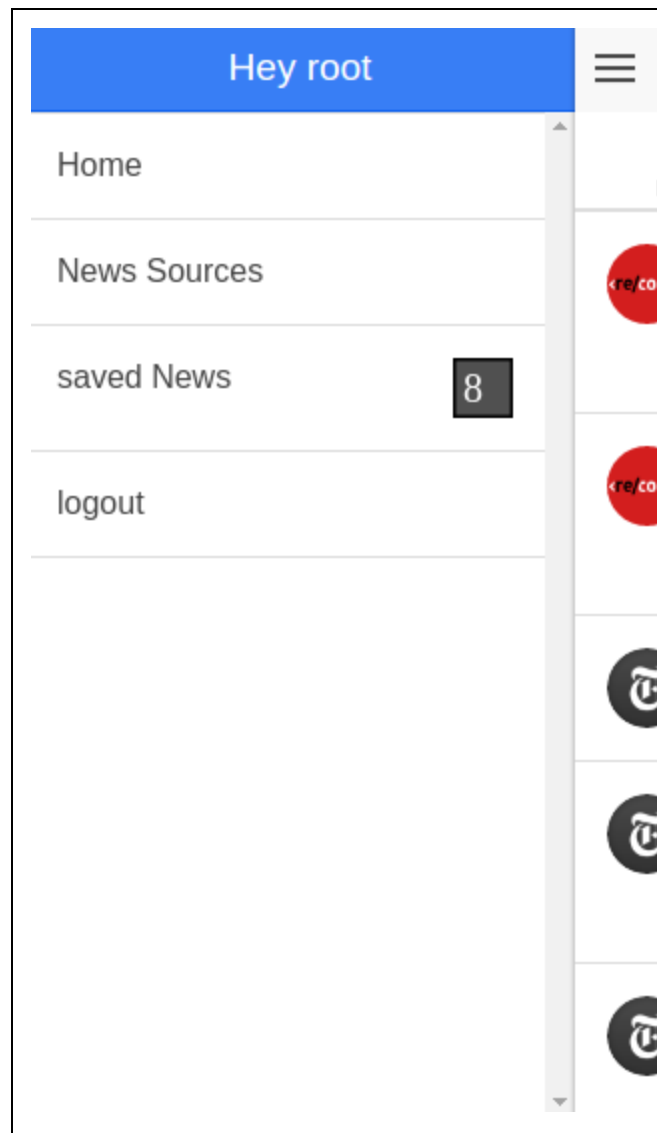




When all the data gets fetched, this is result that will get printed to the user:



8. Menu



When the user swipes to the right, a left menu will show up.

The menu will contain four buttons:

- Home
- News Sources
- Saved News
- logout

You can figure out what each one will do.

V. Improvements:

My application is miles away of being perfect.

As I were developing the application, I have always found things that it needed to be improved. Some of them got that little change or little fix that were needed but other I left or could not finish due to time limit.

I will mention some of them, and how they can be improved:

❖ RAM usage:

The RAM in SmartPhone is a very valuable resource. We do not have enough of it, and therefore we need to use it wisely.

The first thing I have noticed is that the consumption of RAM is out of control which is due to my saving of news and weather data in *services* which are a kind of arrays.

I think the solution to this problem is to fetch only what the user needs, so instead of saving all the information of every news, the application can only fetch the title of the news and when the user wants more information the application will get it for him.

❖ Facebook Connection:

To make the user sign up or login through Facebook connection is really a neat feature, but to make that through you need to manipulate their API.

Facebook API is a little complex to use, I have manipulated it and I was able to make a website to connect with Facebook, but unfortunately I had not enough time to make it to my Hybrid Mobile Application.

❖ News Sources in Menu:

If you did not figure out what the “News Sources” button can do, you are in the right place.

The idea behind that button, was to print all news sources, and make the user add or remove news sources.

Again, due to time limit I was not able to finish it.

❖ Weather page:

The weather part can be a separate application due to the amount of work that it requires.

The weather is a variable thing through the day, so the application needs to fetch the weather each every period of time and update the table in the database.

That what my application misses.

Conclusion

In this two months I was able to understand, develop and finish this project.

It was a pleasant journey as well very educatif.

Being able to develop an hybrid application is very useful as it is the hot thing of this century, and as a future software engineer, this is skill will be very useful in my career.