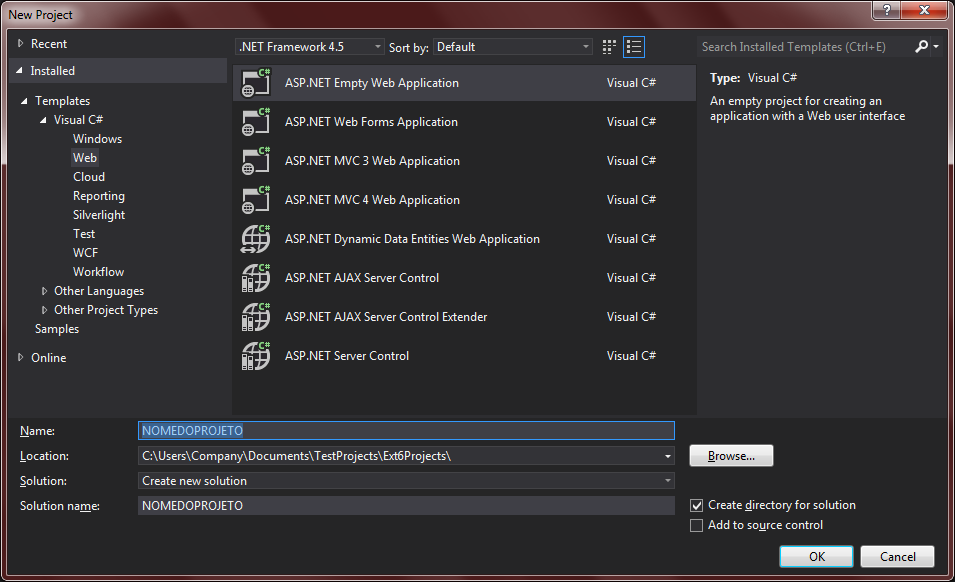
ExtJS 6 – Step by step

*Version: 1.0.0*

# Create project in Visual Studio

1. Choose **ASP.NET Empty Web Application;**
2. Give the project a name and select the location where it will be located;



# Generate app by Sencha Cmd

1. Navigate to the ExtJS 6 sdk folder:

cd C:\Users\Company\Documents\Ext-6.0.2

1. Run the following command, as described below (since the project generated by VS creates a subfolder):

Sencha generate app **PROJECTNAME** C:\Users\Company\Documents\TestProjects\Ext6Projects\ **PROJECTNAME\PROJECTNAME**

1. Navigate to the generated app folder:

cd C:\Users\Company\Documents\TestProjects\Ext6Projects\ **PROJECTNAME\PROJECTNAME**

# Next steps

1. Go to Visual Studio, click on show hidden files and uncheck some files in the generated extjs application.

**NOTE:** If you just include index.aspx in the project, and run the project, the default extjs page will already be working.

* In index.aspx, you can see that there is an Ext.beforeLoad function that identifies which device is being used and applies the type of theme/framework accordingly.
* The types are used through the **modern** and **classic folders** . Usually classic for desktop and modern for mobile.

# Installing cordova in the project

1. Then, open cmd, navigate to the project folder and type the command:

sencha cordova init

The above command will create a cordova folder in the root folder of the application. By default, it will create the applicationID as “ **com.domain.WebApplication1** ”. This can be seen in the config.xml file in the cordova folder. This explanation is given after executing the above command.

**~~NOTE~~** ~~: If you already know the Application ID, enter it like this (but it is recommended to use the code above and then edit it manually):~~

~~sencha cordova init com . mycompany.WebApplication1 WebApplication1~~

1. Open the app.json file in the project root;
2. Insert the cordova code into the builds area. The “builds” area should look like this:

**NOTE:**

* **Be careful with the commas. Otherwise, when you try to run a build in cmd, the system will warn you that it cannot read the JSON.**
* **On MAC, check if the quotation marks (“”) are correctly placed like the others, as the keyboard input pattern may be different. It is advisable to copy and paste the quotation marks that are already in the file.**

"builds" : {

"classic" : {

"toolkit" : "classic" ,

"theme" : "theme-triton"

},

"modern" : {

"toolkit" : "modern" ,

"theme" : "theme-cupertino" ,

**"packager" : "cordova" ,**

**"cordova" : {**

**"config" : {**

**"platforms" : "android" ,**

**"id" : "com.domain.WebApplication1"**

**}**

**}**

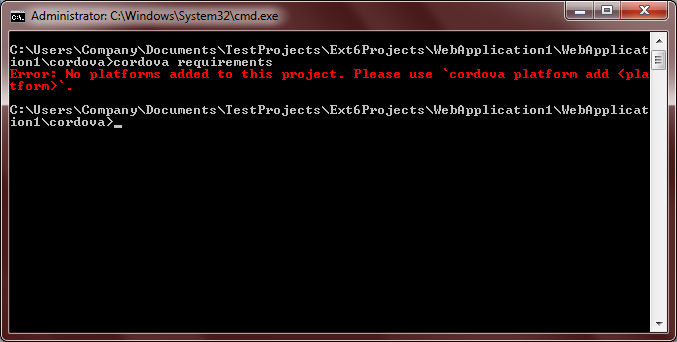
}

}

* For platforms, you can use **ios android . Read the** [Platforms](http://docs.sencha.com/cmd/guides/cordova_phonegap.html#cordova_phonegap_-_platforms) section

## Adding Platforms in Cordova

**cordova requirements** command , we might get this result:



What we need to do is add the platforms. In this case, I will only show you how to add the Android platform because I don't have a Mac.

See the link: <https://cordova.apache.org/docs/en/latest/reference/cordova-cli/index.html#examples>

The page explains how to add additional features such as the device camera.

### In short

1. Install android platform in cordova:

cordova platform add android --save --fetch

1. Install the camera plugin;

cordova plugin add cordova-plugin-camera --save --fetch

1. Check if everything is ok:

cordova requirements android

# Example – Creating a screen that loads ICG users into a form

In this example, a screen will be created in the desktop version (classic toolkit) and in the mobile version (modern toolkit).

In the desktop version, we will create a grid that brings the ICG users. In the mobile version, we will create a form that when pressing the **Load button** , the system randomly loads the users into a form.

**~~NOTE~~** ~~: I tried to create a simple view along with the List.js example in the modern or classic folder and got an error – ExtJS couldn’t load a new .js file at all. So, a “Teste.js” file was created in the “app/view/main” folder and it worked.~~ ( ***Solved –*** [***Read Compiling Sencha Cmd classes***](#_Compilando_as_novas) )

**OBS2** : Take some precautions such as: the modern theme uses “ extend : 'Ext.grid.Grid' , ” for grids and does not have “ extend : 'Ext.grid.Panel' , ”.

## Creating Store and Model

Stores and models must be created in the path “ ***myApp/app/store*** *”* and “ ***myApp/app/store*** *”* .

* In the app folder, create a store called UsersICG:

Ext . define ( 'WebApplication1.store.UsersICG' , {

extend : 'Ext.data.Store' ,

indeed : 'store.UsersICG' ,

autoLoad : true ,

storeId : 'UsersICG' ,

model : 'WebApplication1.model.UsersICG'

});

* In the app folder, create a UsersICG model:

Ext . define ( 'WebApplication1.model.UsersICG' , {

extend : 'Ext.data.Model' ,

requires : [

'Ext.data.Field'

],

//fields: [

//'ds\_user', 'username'

//],

config : {

useCache : false ,

fields : [

'ds\_user' , 'username'

]

},

proxy : {

type : 'ajax' ,

headers : { 'Content-Type' : 'application/json' , 'Accept' : 'application/json' },

timeout : 5 \* 60 \* 1000 ,

api : {

//read: '../../../../services/Usuarios.asmx/GetUsuarios'

read : 'http://192.168.0.11:8383/UsuariosICG/GetUsuarios'

},

reader : {

type : 'json' ,

rootProperty : 'date' ,

successProperty : 'd.success' ,

messageProperty : 'd.msg' ,

totalProperty : 'd.total'

},

noCache : false

}

});

## Creating a View

In Extjs 6, we noticed that there are 2 toolkits: *classic*and *modern* , where *classic* refers to applications that run in the browser and *modern* on mobile. At the beginning of this example, there is **OBS2** which explains the reason for the separation of views, as there are differences in the structure of both, that is, one has a component that the other does not have in the same way/syntax.

### Classic Toolkit

* Create a view for the *classic toolkit* (browser client), in the path *MyApp/* ***classic*** */src/view/main* :

Ext . define ( 'WebApplication1.view.main.UsersICG' , {

extend : 'Ext.panel.Panel' ,

xtype : 'UsersICG' ,

requires : [

'WebApplication1.store.UsersICG' ,

'WebApplication1.model.UsersICG'

],

title : 'ICG User Panel' ,

id : 'id\_panel\_usuariosICG' ,

items : [

{

xtype : 'button' ,

itemId : 'loadUsers' ,

listeners : {

click : 'onAddClick'

},

text : 'To load'

},

{

xtype : 'grid' ,

title : 'ICG Users Grid (test)' ,

store : {

type : 'UsersICG'

},

columns : [

{ text : 'Name' , dataIndex : 'ds\_user' , width : 100 },

{ text : 'User' , dataIndex : 'username' , width : 230 }

]

}

]

});

### Modern Toolkit

* Create a view for the *modern toolkit* (browser client), in the path *MyApp/* ***modern*** */src/view/main* :

Ext . define ( 'WebApplication1.view.main.UsersICG' , {

extend : 'Ext.panel.Panel' ,

xtype : 'UsersICG' ,

requires : [

'WebApplication1.store.UsersICG' ,

'WebApplication1.model.UsersICG'

],

title : 'ICG User Panel' ,

id : 'id\_panel\_usuariosICG' ,

items : [

{

xtype : 'button' ,

itemId : 'loadUsers' ,

listeners : {

tap : 'onAddClick' ,

click : 'onAddClick'

},

text : 'To load'

},

{

xtype : 'grid' ,

title : 'ICG Users Grid' ,

store : {

type : 'UsersICG'

},

columns : [

{ text : 'Name' , dataIndex : 'ds\_user' , width : 100 },

{ text : 'User' , dataIndex : 'username' , width : 230 }

]

},

{

xtype : 'formpanel' ,

//fullscreen: true,

items : [

{

xtype : 'textfield' ,

name : 'name' ,

id : 'id\_name' ,

label : 'Name'

},

{

xtype : 'textfield' ,

name : 'username' ,

id : 'id\_username' ,

label : 'Username'

},

{

xtype : 'passwordfield' ,

name : 'password' ,

label : 'Password'

}

]

}

]

});

### Compiling the new views created in Sencha Cmd

Any inclusion/addition of new files in the extjs project, the framework needs to compile into a global .JSON. To do this, the following commands are executed, each according to the toolkit:

* **It is important to have the index.html, otherwise it will not work;**

Sencha app watch classic

Sencha app watch modern

**NOTE:** When executing any of the commands above, Sencha Cmd will give a message as if it is loading or waiting for something ( *Waiting for changes* ). But this is normal, the process has already finished and the command has already been executed. Now just test that it will work on both platforms/toolkits.

**NOTE 2:** This .json that is compiled can be seen in the project root folder. There is ***modern.json*** and ***classic.json*** . To ensure that the above commands were executed successfully, an alternative to checking manually is to open these files and search for the name of the file you just included.

### Loading Screen...

To add a loading screen, before loading the page or opening the application, simply insert these codes in index.html.

* ***style*** tag :

< style type = "text/css" >

/\*\*

\* Example of an initial loading indicator.

\* It is recommended to keep this as minimal as possible to provide instant feedback

\* while other resources are still being loaded for the first time

\*/

html , body {

height : 100% ;

background-color : #5fa2dd

}

#appLoadingIndicator {

position : absolute ;

top : 50% ;

margin-top : -15px ;

text-align : center ;

width : 100% ;

height : 30px ;

-webkit-animation-name : appLoadingIndicator ;

-webkit-animation-duration : 0.5s ;

-webkit-animation-iteration-count : infinite ;

-webkit-animation-direction : linear ;

}

#appLoadingIndicator > \* {

background-color : #FFFFFF ;

display : inline-block ;

height : 30px ;

-webkit-border-radius : 15px ;

margin : 0 5px ;

width : 30px ;

opacity : 0.8 ;

}

@-webkit-keyframes appLoadingIndicator {

0% {

opacity : 0.8

}

50% {

opacity : 0

}

100% {

opacity : 0.8

}

}

</ style >

* ***body*** tag :

< body >

< div id = "appLoadingIndicator" >

< div ></ div >

< div ></ div >

< div ></ div >

</ div >

</ body >

Done, loading screen added!

## Using the Camera (version: 1.0 *works, but maybe not a “good practice”* )

**NOTE:** One way found to make the camera work in extjs was to manually copy the framework folders and paste them into the modern folder. (See step 3).

1. Navigate to the cordova folder and check the cordova plugins that are installed:

cordova plugins

1. If necessary, install the camera plugins:

cordova plugin add cordova-plugin-camera

1. In the **MyProject\ext\packages\legacy\modern\src\device folder** , copy the “camera” and “communicator” folders and the “Camera.js” and “Communicator.js” files.
2. **MyProject \ext\modern\modern\src** folder , create a new folder called **device** and paste the two folders and the two files inside this folder.

**Warning: VS publish did not copy the files when this implementation was done (so it was copied manually).**

1. In MainController, add the events:

attachImage : function ( btn ) {

var me = this ;

var actionSheet = Ext . create ( 'Ext.ActionSheet' , {

items : [

{

text : 'Take a picture with the Camera' ,

handler : function () {

me . captureImage ( 'camera' , actionSheet );

}

},

{

text : 'Choose image from album' ,

handler : function () {

me . captureImage ( 'album' , actionSheet );

}

},

{

text : 'Cancel' ,

ui : 'decline' ,

handler : function () {

actionSheet . hide ();

Ext . Viewport . remove ( actionSheet );

}

}

]

});

Ext . Viewport . add ( actionSheet );

actionSheet . show ();

},

captureImage : function ( src , actionSheet ) {

var me = this ;

Ext . device . Camera . capture ({

success : function ( image ) {

console . log ( image );

//var attachmentsForm = me.getNewFalhaForm().down('#attachments');

var img = Ext . create ( 'Ext.Img' , {

src : image ,

height : 48 ,

width : 64

});

//attachmentsForm.insert(0, img);

var field = Ext . create ( 'Ext.field.Hidden' , {

name : 'attachments' ,

value : image

});

//annexesForm.insert(0, field);

actionSheet . hide ();

Ext . Viewport . remove ( actionSheet );

},

source : src ,

encoding : 'jpg' ,

quality : 75 ,

width : 640 ,

height : 480 ,

destination : 'file'

});

}

1. In the UsersICG view of the modern folder ( **WebApplication1\modern\src\view\main\UsersICG.js** ), add the require “ **Ext.\*** ” and the button that will trigger the event for the functions above.

requires : [

'Ext.\*',

'WebApplication1.store.UsersICG' ,

'WebApplication1.model.UsersICG'

],

........................................

{

xtype : 'button' ,

listeners : {

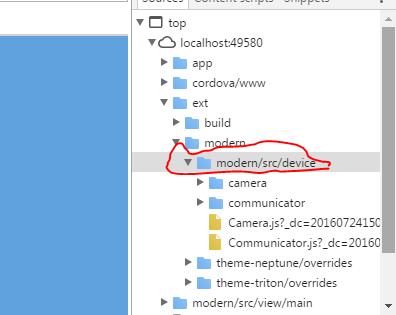
tap : 'attachImage'

},

text : 'Attach Image'

},

This will make the following appear in the browser:



1. Test in the PC browser if the Ext.device component is working.
2. [Build Android](#_Build_do_projeto)
3. Done, access to the camera and album will be working.

# Creating services / webservices

**Important:**

* When making a connection to the database through the store/model, there is a path passed that refers to the location where the webservice and its function are.
* To perform local testing, you must open the firewall port (same port as IIS) for the Android client to consume the webservices.
* Same thing for production, you must have an application running on a hosting/url or simply the webservices, because the mobile application is just a client, it does not compile the entire ASP.NET application. Remember that Cordova only packages the extjs part and not the entire ASP.
* Check if the **IIS pool's .NET framework version is 4.0** (if not, it will not run any type of service, neither .asmx nor .svc)

Two ways of creating a webservice will be demonstrated. With .asmx and/or .svc:

## .asmx:

With .asmx, the model should look like this:

read : 'http://192.168.0.11:8383/services/UsuariosICG.asmx/GetUsuarios'

using System ;

using System . Collections . Generic ;

using System . Linq ;

using System . Web ;

using System . Web . Script . Services ;

using System . Web . Services ;

namespace WebApplication1 . services

{

/// <summary>

/// Summary description for Users

/// </summary>

[ WebService ( Namespace = "http://tempuri.org/" )]

[ WebServiceBinding ( ConformsTo = WsiProfiles . BasicProfile1\_1 )]

[ System . ComponentModel . ToolboxItem ( false )]

// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.

[ System . Web . Script . Services . ScriptService ]

public class Users\_Old : System . Web . Services . WebService

{

[ Serializable ]

public class User {

public virtual string ds\_user { get ; set ; }

public virtual string username { get ; set ; }

}

[ WebMethod ( EnableSession = true )]

[ ScriptMethod ( UseHttpGet = true , ResponseFormat = ResponseFormat . Json )]

public object GetUsers ()

{

IList < User > listUsers = new List < User > ();

User userClass ;

var usersICG = Library . Data . Wrappers . Users . FindAllByID\_CLIENT ( 203 );

foreach ( var item in usersICG )

{

userClass = new User

{

ds\_user = item . DS\_USER ,

username = item . USERNAME

};

listUsers . Add ( userClass );

}

return listUsers ;

}

}

}

## .svc:

For .svc, a change is required in Global.asax, which will change the webservice URL route:

In application\_start, insert this code:

RouteTable . Routes . Add ( new ServiceRoute ( "ICGUsers" , new WebServiceHostFactory (), typeof ( WebApplication1 . services . UsuariosICG )));

**NOTE : If necessary,** manually add the “ System.ServiceModel.Activation ” and “ System.ServiceModel.Web” dlls to the references .

With that, the model could look like this:

read : 'http://192.168.0.11:8383/UsuariosICG/GetUsuarios'

In the folder that was created for the services, add a WCF Service, which will create a **IUsuariosICG.cs and UsuariosICG.svc** .

### ServiceName.svc:

using System ;

using System . Collections . Generic ;

using System . Linq ;

using System . Runtime . Serialization ;

using System . ServiceModel ;

using System . ServiceModel . Activation ;

using System . Text ;

namespace WebApplication1 . services

{

// NOTE: You can use the "Rename" command on the "Refactor" menu to change the class name "Usuarios" in code, svc and config file together.

// NOTE: In order to launch WCF Test Client for testing this service, please select Usuarios.svc or Usuarios.svc.cs at the Solution Explorer and start debugging.

[ AspNetCompatibilityRequirements ( RequirementsMode = AspNetCompatibilityRequirementsMode . Required )]

public class ICGUsers : ICGUsers

{

public UserReturn GetUsers ()

{

List < UserModel > listUsers = new List < UserModel > ();

UserReturn ret = new ReturnUser ();

UserModel userClass ;

var usersICG = Library . Date . Wrappers . Users . FindAllByID\_CLIENTE ( 203 ) . Take ( 50 );

foreach ( var item in usersICG )

{

userClass = new UserModel

{

ds\_user = item . DS\_USER ,

username = item . USERNAME

};

listUsers . Add ( userClass );

}

ret . date = listUsers ;

return ret ;

}

}

}

### ICGUsers:

using System ;

using System . Collections . Generic ;

using System . Linq ;

using System . Runtime . Serialization ;

using System . ServiceModel ;

using System . ServiceModel . Web ;

using System . Text ;

namespace WebApplication1 . services

{

// NOTE: You can use the "Rename" command on the "Refactor" menu to change the interface name "IUsuarios" in both code and config file together.

[ ServiceContract ]

public interface ICGUsers

{

//[OperationContract]

//void DoWork();

[ OperationContract ]

[ WebGet (

UriTemplate = "/GetUsers" ,

RequestFormat = WebMessageFormat . Json ,

ResponseFormat = WebMessageFormat . Json )]

UserReturn GetUsers ();

}

[ DataContract ]

public class UserModel

{

[ DataMember ( Name = "ds\_user" )]

public string ds\_user ;

[ DataMember ( Name = "username" )]

public string username ;

}

[ DataContract ]

public class UserReturn //: ReturnJsonBase

{

[ DataMember ]

public List < UserModel > date ;

}

}

# APK and Project Build

* Navigate to the project root folder.
* **It is important to have the index.html, otherwise it will not work;**

For the first build, it is necessary to run the command below:

Sencha app build

This command will also build for Android.

Then, to generate only the mobile version, run the command below:

Sencha app build modern

# Versioning

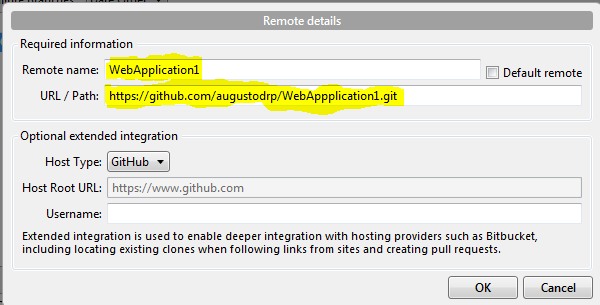
## .GitHub

### GitHub website

* Create repository on GitHub website;

### SourceTree

* In sourcetree, click on “ ***Clone / New” > New repository*** ;
* Put the Http address of git ( <https://github.com/augustodrp/WebAppplication1.git>)
* Then, go to ***Repository > Repository Settings;***
* ***Remote*** tab , click on add, and fill in as follows:



* Select the ***Unstaged files*** and commit;
* Click ***Push*** ;
* After clicking push, you will be taken to a login screen. This authentication works like this:
* Username: ***github user*** *(* ***https://github.com/augustodrp*** *)* ;
* Password: ***github password***
* Done! The commit should work. Remember that the initial commit takes a few minutes, as it uploads the entire project.

## GitIgnore – Configuration

### First commit without setting ignoreFile

Not all of them need to be versioned, as many are local files. Visual Studio, for example, creates many files such as ***.suo*** , ***.csproj.user*** , /bin folder and /obj folder.

If you made the first commit without having configured the ignore file before, git will commit all the files in the project and if you only configure the ignorefile to ignore these files, it will not work because git does not ignore files that are already versioned ( *confirm this statement* ).

**Important** : Delete all debug files. (In the /obj folder there is also /obj/debug).

The fact is, we then need to delete these files from the repository before configuring the ignore file.

After that, commit these deletions. To get an idea, [see a deletion commit](https://github.com/augustodrp/WebAppplication1/commit/2dee49064805954eb91add09370ca243c5f42f30) .

Then, set up .gitignore like this:

\*.suo

\*.log

\*.log.\*

WebApplication1/obj

WebApplication1/bin

WebApplication1/WebApplication1.csproj.user

WebApplication1/\*.log

# Debug Android Device on Chrome (via USB)

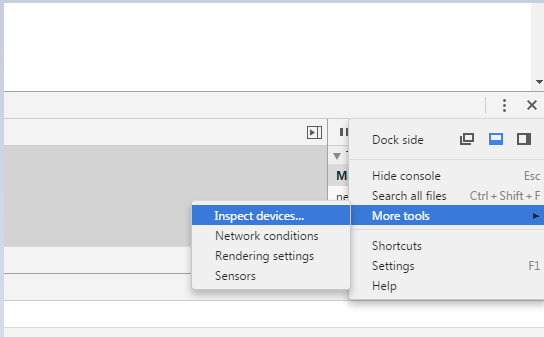
Documentation: <https://developers.google.com/web/tools/chrome-devtools/debug/remote-debugging/remote-debugging>

Important: The AndroidManifest.xml file (in cordova) must have debug enabled ( **android:debuggable="true"** ). (Read more: <https://developer.android.com/studio/run/device.html>)

The TRF path, for example, is: **C:\Users\Company\TRF2\TRF2\cordova\platforms\android\** **AndroidManifest.xml**

To configure, you must follow these steps:

* Configure Android to allow USB debugging;
* Connect the android device to the PC USB;
* Open chrome on your PC;
* In the developer tool, click “:” (3 dots - next to the “x” button) > More Tools > Inspect Device;



* This screen will open, which means you need to authorize this connection on Android:



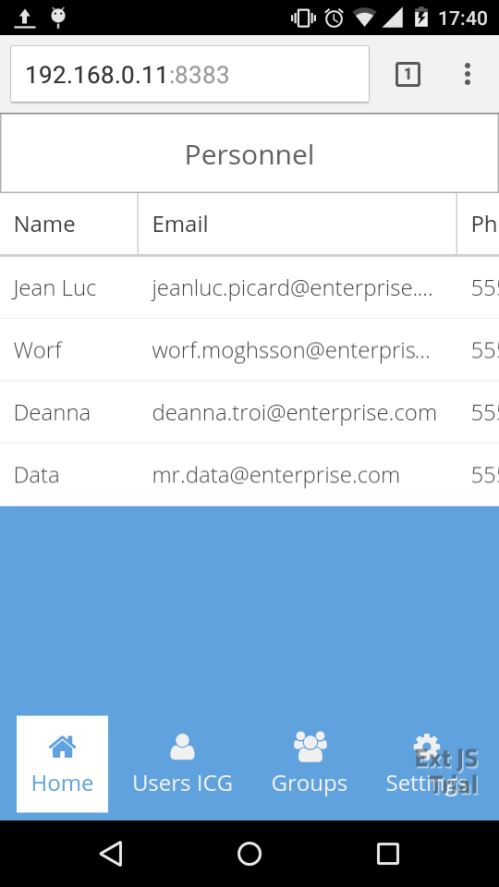
* Authorize on android:



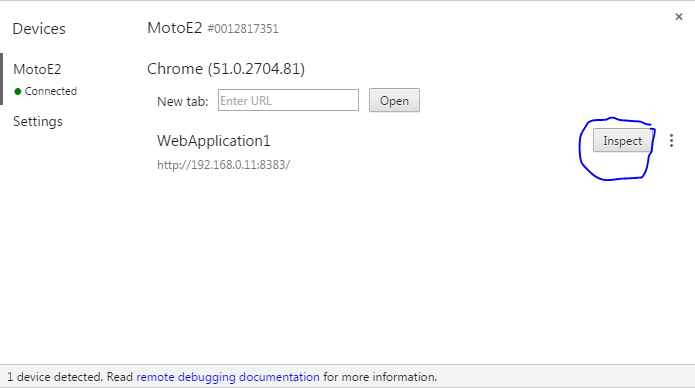
* After authorizing on the Android screen, you will be connected:



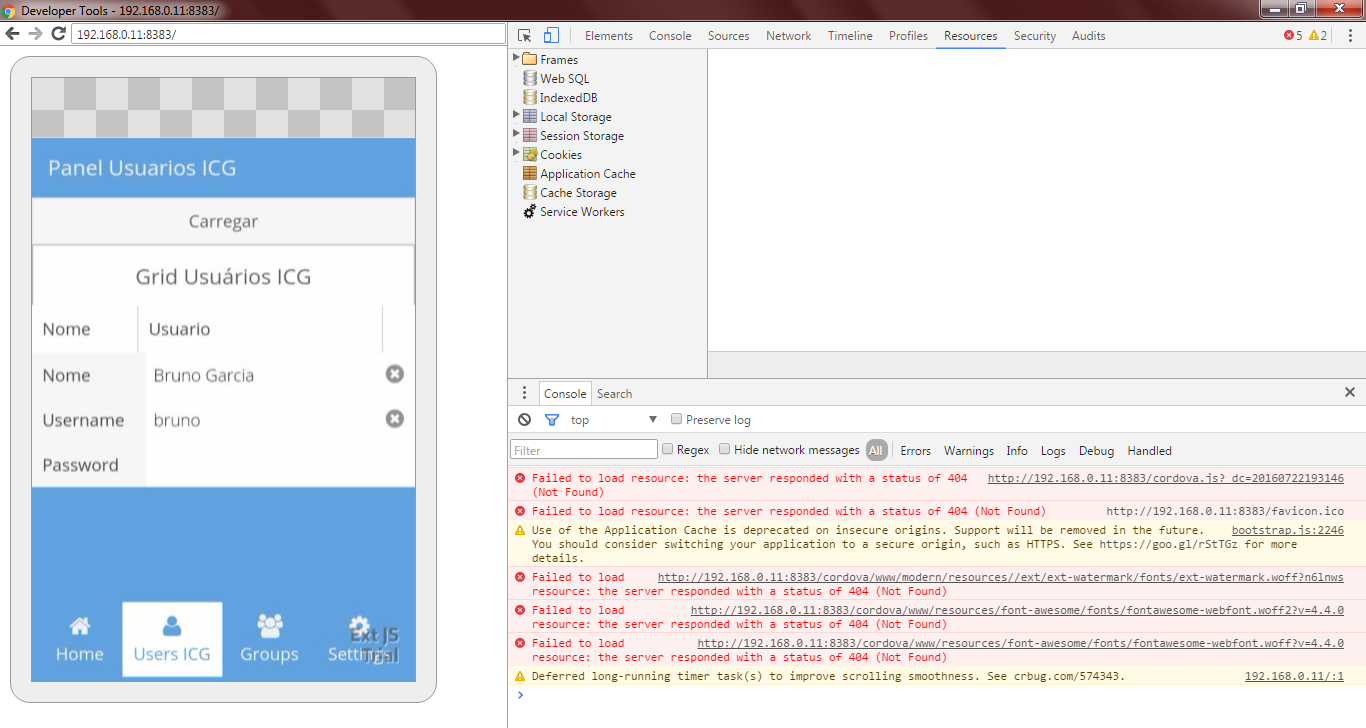
* Access the desired URL via Android:



* Returning to the developer tool, you can see the accessed URL, click on Inspect:



* Okay, now you can debug on your Android device:



# Publish Visual Studio

**First: Read the next topic.**

To make a local publish, it is important to remember that **VS only publishes files that are included in the project.** So for this, it is necessary to take some precautions, for example, it is essential that the ***classic.json* and *modern.json*** filesare included in the project, as sencha compiles all the application classes into these .json files. ( [See more about this](#_Compilando_as_novas_1) ).

# IIS – Visual Studio Configuration

To facilitate testing alongside development, it is possible to configure IIS to host the folder where the application under development is located, without the need to publish as per the previous topic. To do this, follow the steps below:

1. Grant permissions on the root folder where the application is located, allowing all access to “ **Everyone** ”;
2. In IIS, add a website and point it to the application folder;
3. Go to **Application Pools** and change the framework version to 4;
4. Now, the application should be running, but without a connection to the database. To do this, follow the next step;
5. Include the following lines in WEB.CONFIG:

< system.webServer >

< handlers >

<!-- LINES BELOW WERE ADDED BECAUSE OF IIS

(BUG THAT WAS OCCURRING: IIS WAS ALREADY CONNECTING TO THE BANK, HOWEVER, FOR EXAMPLE, WHEN LOGINING IT WAS GIVING AN ERROR)

NOW ACCESS FROM OUTSIDE TO PORT 9292 WORKS -->

< add name =" ScriptHandlerFactory " verb =" \* " path =" \*.asmx " preCondition =" integratedMode " type =" System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35 " />

< add name =" ScriptHandlerFactoryAppServices " verb =" \* " path =" \*\_AppService.axd " preCondition =" integratedMode " type =" System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35 " />

</ handlers >

1. For external testing, open the advanced firewall and add the application port created in IIS to Inbound Rules;
2. Ready!

# DEBUG IIS in Visual Studio

## [Attach to the ASP.NET application from the Visual Studio computer](javascript:void(0))

* Run Visual Studio as **Administrator** ;
* On the Visual Studio computer, open the **MyMVC** solution.
* In Visual Studio, click **Debug / Attach to Process** .
* ~~Set the Qualifier field to~~**~~<remote computer name>:4020~~** ~~.~~ (in this case it will not be remote. So just click on “Find..” directly)
* You should see some processes appear in the **Available Processes** window.
* Check **Show processes from all users** .
* Look for **w3wp.exe** and click **Attach** .
* Open the remote computer's website. In a browser, go to **http://<remote computer name>** .
* ~~You should see the ASP.NET web page. Click~~**~~About~~** ~~.~~

~~The breakpoint should be hit in Visual Studio.~~

Link: <https://msdn.microsoft.com/en-us/library/mt621540.aspx>

# PUBLISH GOOGLE PLAY

First of all, it is worth remembering that it will be published in production. Therefore, first follow the steps below before going further:

* Check if **android:debuggable="false"** is actually false. (See [Debug Android Device on Chrome (via USB)](#_Debug_Android_Device) );
* Change the app version in the cordova **config.xml file;**
* Check if the application URL (located in the **index.html file** ) is pointing to the production link;

Then, assuming there is already an existing key, follow the instructions below.

* [Generate the apk using cordova](#_APK_e_Build)
* Replace the file “ **android-debug-unaligned.apk** ” with “ **TRF.apk** ”.
* Open CMD as administrator;
* Navigate to the folder **C:\Program Files (x86)\Android\android-sdk\build-tools\23.0.3** (required for **zipalign** )
* Run the commands:

jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore C:\Users\Company\TRF2\TRF2\TRF.keystore C:\Users\Company\TRF2\TRF2\cordova\platforms\android\build\outputs\apk\TRF.apk "TRF"

* TRF Keystore Password: **SECRET\_PASSWORD**
* The above command will generate a size error. So open this apk with winrar, delete the META-FILES folder;
* **Now sign up for it:**

jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore C:\Users\Company\TRF2\TRF2\TRF.keystore C:\Users\Company\TRF2\TRF2\cordova\platforms\android\build\outputs\apk\TRF.apk "TRF"

* **Now zip:**

zipalign -f -v 4 C:\Users\Company\TRF2\TRF2\cordova \platforms\android\build\outputs\apk\TRF.apk C:\Users\Company\TRF2\TRF2\cordova\platforms\android\build \outputs\apk\TRF-Release.apk

* Now upload the TRF-Releaase.apk file

# Debug ExtJS (Enable/Disable)

* In the app.json file at the root of the project, look for **debug** .
* There will be only two locations (in the classic area and in the modern area);
* Read the instructions in the code, and if you agree, comment out that line in the file.