# CRIAX-SDK, aplicación móvil Memo

Nilmar Sánchez Muguercia


Memos
Memo ejemplo
26/02/2014 (10:50:30)
CRIAX
Chromeless Rich Internet Ap
by Qooxdoo

# Sobre esta edición

**Fecha de Publicación:** 20/03/2014

Este libro ha sido publicado con easybook v4.8, una herramienta libre, de código abierto para la publicación de libros. Ha sido desarrollada por Javier Eguiluz, usando varios componentes de Symfony2 y disponible de forma gratuita en easybook-project.org (http://easybook-project.org/) .



**Poder hacer consultas directas al autor:** cualquier parte del libro, tanto dudas sobre ejemplos, capítulos, códigos se responderán normalmente en un período de 24hs.

**Acceso a todos los fuentes:** de todos los codigos fuentes del libro, revisados y comentados por el mismo autor, estarán publicados en el sitio de Firefoxmania (http://firefoxmania.uci.cu/) , en el subdominio de desarrollo (http://developer.firefoxmania.uci.cu/) .

**Actualizaciones:** tanto correcciones como ejemplos o hasta capítulos nuevos, estarán publicados en el sitio de Firefoxmania (http://firefoxmania.uci.cu/) , en el subdominio de desarrollo (http://developer.firefoxmania.uci.cu/) .

**Cambia el contenido del libro:** si consideras que algún capítulo, código o imágen podría mejorarse, o algún tema que ves no se encuentra tratado en el libro, tu sugerencia será recibida y tenida en cuenta para la próxima actualización del libro.

**Contactar autor:** (nilmar@uci.cu) , (nilmar@isla-paradiso.net)

# Licencia

*A mi familia, por confiar en mí, por apoyarme con amor infinito en todos los retos que he afrontado en mi carrera. A mi princesita Bublic, por todo el tiempo dispensado, que era parte de su tiempo. A todas aquellas personas que me quieren y me desean bien. A toda la comunidad de Mozilla y en especial a Firefoxmania (Comunidad de Mozilla en Cuba), por brindarme un espacio, a los desarrolladores de los proyectos Qooxdoo y B2G. A todos aquellos que mejoran la web cada día.*

# Agradecimientos

Agradezco al destino, por como ha sido conmigo, pues las situaciones que se me presentaron son las que me han llevado a este momento, ya que después de mucho pensar, trato de sacar algo positivo de lo negativo que pudo suceder en el camino.

A Javier Eguiluz y Andre Alves Garzia, por la herramienta y magnifica guía rápida respectivamente, pues sirvieron de punto de partida para este pequeño libro.

En especial a Martin Fowler, pues fue quien realmente me abrió los ojos al mundo profesional del desarrollo de software, gracias Martin pues me enseñastes lo que realmente debía aprender.

A los técnicos del Docente 4, Facultad 6, por brindarme el laboratorio 201 para que pudiera trabajar con tranquilidad. A su jefa Nohemy por todas las veces que la molesté pidíendole la PC.

A todos ellos, gracias

# Sobre el autor

Mi nombre es Nilmar Sánchez Muguercia, actualmente tengo 27 años y nací en La Habana, Cuba. Interesado en Artes Marciales, Sherlock Holmes y la Web.

En este libro encontrarás muchas partes en las que expreso mi opinión personal y tomo decisiones que pueden ser diferentes a las de otros programadores. Si existe algún error en lo que expreso por favor hacérmelo llegar para futuras versiones del libro, ya sea por correo (nilmar@uci.cu) , (nilmar@isla-paradiso.net) o por Twitter @criaxsdk (https://twitter.com/criaxsdk) .

# Índice de contenidos

# Lista de figuras

# Prefacio

Recientemente volví a leer esa magnífica guía rápida (http://leanpub.com/quick-guidefirefoxosdevelopment) que hiciera Andre Garzia para desarrollar aplicaciones para Firefox OS. Es entonces que me vino la idea de por qué no hacer esta aplicación en CRIAX-SDK, así se podría mostrar el funcionamiento de la plataforma para el desarrollo de aplicaciones móviles, e introducir el primer manual de la misma.

# Introducción

# Para quien es este libro

Este libro es para aquellos desarrolladores web, que estén interesados en ampliar su espectro en la construcción de software para plataformas web y móvil. En especial, para aquellos que se sientan atraídos por la creación de webapp para Firefox OS.

# Buenas prácticas

Desarrolladores experimentados, notarán que durante el desarrollo de la aplicación no suelo seguir las buenas prácticas. Esto se debe a una razón: la simplicidad y poder mantener el código lo más esclarecido posible. Todo el código presente funciona correctamente, a medida que se vayan realizando actualizaciones del libro, se irán implementando mejores prácticas de manera tal, que la complejidad pueda ir aumentando gradualmente.

# Opiniones, críticas, quejas y sugerencias

Este es un libro gratis y con la intención de introducir el uso de la plataforma CRIAX-SDK en el desarrollo de aplicaciones móviles para Firefox OS, se aceptan todas las opiniones, críticas, quejas y sugerencias. Gracias de antemano a todos aquellos que con su opinión contribuyan a mejorar el libro.

# Intenciones

Las intenciones de este libro van más allá de centrarse en el aprendizaje de HTML, CSS, Javascript, Firefox OS o el simulador de B2G.

# Revisores

- **Jany Ramos Mosqueda**

Jany Ramos Mosqueda, nacida en La Habana, Cuba el 25 de junio de 1987. Graduada de Técnico en Informática el 24 de julio de 2006. Trabajó como traductora de episodios de Tras la Huella en 2009. Graduada de Ingeniería en Ciencias Informáticas el 19 de julio de 2011. Se ha desempeñado como analista de sistemas, líder de proyecto y desarrolladora en el centro Telemática de la Universidad de las Ciencias Informáticas. Entre sus publicaciones científicas se encuentran: "Plataforma de gestión de servicios telemáticos para GNU/Linux. Módulo DNS" y "Herramienta de monitorización del despliegue para centros de emergencias 171".

# Historial de versiones

## Versión 0.1

Esta es la primera versión del libro, si contiene algún error gramatical, de concordancia u ortográfico, por favor perdónenme.

# Como leer este libro

El libro está describe el desarrollo de una aplicación para tomar notas, todo el proceso desde la concepción de la idea hasta la instalación de la aplicación en un teléfono.

Consta de 3 partes:

- La primera explica cómo crear una webapp para Firefox OS con la plataforma, así como los primeros pasos en el desarrollo de la aplicación, llegando hasta el listado de los memos que se encuentran guardados.

- La segunda parte explica cómo llevar a cabo la gestión de memos (insertar, actualizar y eliminar).

- La tercera parte explica como dejar lista la aplicación para su distribución, como ejecutarla en el simulador de B2G y como instalarla en un teléfono con Firefox OS.

# La plataforma

## Nacimiento

Cuántas veces como desarrollador Web has querido hacer algo para el escritorio con los conocimientos que posees? Bueno creo que a muchos les ha pasado. Por estos deseos, sueños, han nacido plataformas que nos permiten sin tener conocimiento de la programación en el entorno de escritorio, crear aplicaciones que corran en él, utilizando tecnologías y lenguajes creados inicialmente para la Web.

Ejemplos de este tipo de software hoy día tenemos: AppJs, TideSDK, Adobe Air, entre otros. Plataformas que de forma simple ponen a nuestro servicio la posibilidad de creación de aplicaciones RIA embebidas en el escritorio. Es en este grupo donde clasifica CRIAX-SDK (Chromeless Rich Interner Applications by Qooxdoo). CRIAX es una plataforma que permite la creación de aplicaciones de escritorio con tecnologías Web, así como webapp para Firefox OS.


**Figura .1** "Identidad de CRIAX-SDK"

**NOTA:**El diseño de la identidad de CRIAX fue desarrollado por Pedro Enrique Palau Isaac (pepalau@uci.cu).

Producto de la fusión de 2 proyectos, Chromeless y Qooxdoo, pretende ser una alternativa libre, multiplataforma y sencilla para la creación de este tipo de aplicaciones. Entre sus principales características se pueden encontrar:

**Interfaz:**

- Paquetes de iconos listos para ser utilizados
- Configuración de temas
- Creación de interfaces tipo Qt o Swing

- Personalización de temas sin tener conocimientos profundos de CSS o HTML
- Homogeneidad entre los distintos Sistemas Operativos

**Persistencia:**

- Interacción con SQLite
- Carga de datos a partir de Json
- Carga de datos a partir de YQL (Yahoo Query Languaje para web services)
- Consultas a web services

**Interacción con el sistema:**

- Lectura, escritura de archivos
- Ejecución de comandos
- Clipboard
- Otros

**Toolchain:**

- Generador de API
- Testrunner para los test
- Revisión del código
- Almacenamiento en cache de clases precompiladas
- Generación de distribuible
- Compresión y optimización de código final

**Arquitectura y diseño:**

- POO (interfaces, herencia, mixin, clases abstractas, clases estáticas, clases singleton)
- Lenguaje Javascript
- Uso de Namespace
- HMVC para la presentación
- Fácil integración con librerías de terceros

**Otras:**

- Contenedor de servicios (json)
- Generador de código (aun en desarrollo)
- ORM (mapeo json)
- Internacionalización
- Configuración de aplicación (json)
- Autocarga de clases

Quizás muchas de estas características resulten familiares de las plataformas mencionadas anteriormente y otras sean totalmente nuevas en este tipo de software.

# La plataforma

## Instalación

Lo primero es descargar CRIAX-SDK, para el directorio que más nos guste.

**NOTA:**traten de que la ruta no contenga espacios en blanco.

En Linux

```
$ /home/tecnico/
```

En Window

```
$ D:\
```

Como próximo paso tenemos, descompactar el .zip y ya está, listo CRIAX-SDK para empezar a desarrollar. Ahora que ya tenemos la plataforma funcional en todo su esplendor, vamos a crear el **workspace**, para las aplicaciones que iremos desarrollando. Simplemente creamos un directorio donde más nos guste y allí dentro estarán todas las aplicaciones creadas con CRIAX-SDK.

En Linux

```
$ /home/tecnico/CriaxProyects/
```

En Window

```
$ D:\CriaxProyects\
```

# I PARTE: La aplicación

Capítulo 1

# Creando la aplicación

La aplicación a desarrollar es muy simple, una Memo para tomar notas. La aplicación tiene 2 pantallas, una para mostrar el listado de los memos y otra donde podremos crear, editar o borrar los mismos. A diferencia de la aplicación original no utiliza **IndexedDb** para almacenar la información sino un sistema JSON, que puede ser accedido a través de consultas SQL.

Lo primero que haremos será crear el skeleton de la aplicación llamada `memo`. Para ello nos paramos en el directorio raíz de la plataforma y tecleamos por consola:

```
$ cd Criax
$ create-application (.bat o .sh) -n memo -o
.../CriaxProyect/ -t firefoxos
```

Como se puede observar al comando se le pasan 3 parámetros, el nombre de la aplicación `-n`, la ruta donde se creará la aplicación `-o` y el tipo de aplicación a crear `-t`.

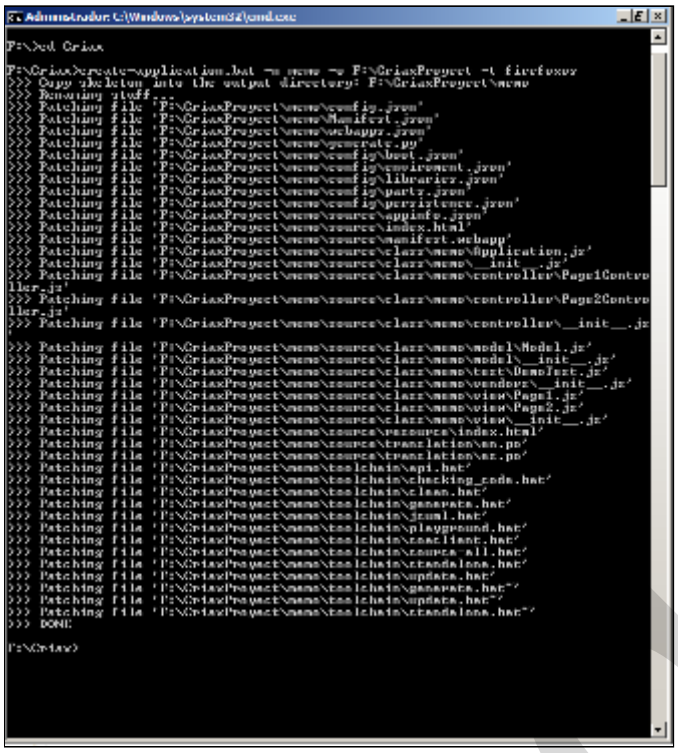


**Figura 1.1** "Creación de la aplicación"

Una vez creada la aplicación el siguiente paso es generarla para pre compilar las clases y dejarlas listas para futuro desarrollo de otras aplicaciones, para ello nos vamos hasta el directorio raíz de la aplicación creada y tecleamos en la consola:

```
$ cd memo
$ toolchain/generate(.bat o .sh)
```

Quizás demore un poquito pues esto pre compila todas las clases de la plataforma. Una vez que termine la generación se nos mostrará una ventana como la siguiente indicándonos que la aplicación esta lista para comenzar a ser desarrollada.


**Figura 1.2** "Generación de la aplicación: página 1."

Este skeleton posee 2 páginas.


**Figura 1.3** "Generación de la aplicación: página 2."

Capítulo 2

# La programación

Ahora pasemos a lo importante, la funcionalidad de nuestra aplicación, para ello abrimos el vendor Criax-IDE `Criax/vendor/Criax-IDE/spket` y creamos un nuevo proyecto, a diferencia de lo que se piensa la ruta del proyecto en el IDE no será el directorio raíz de la aplicación sino `memo/source/class/memo/`.

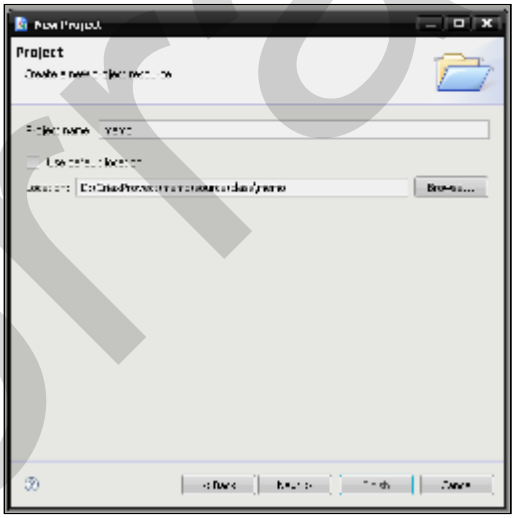


**Figura 2.1** "Creando un nuevo proyecto"

Una vez listo el proyecto en el IDE, y generado, lo primero que haremos será crear un modelo que se encargará de trabajar con la persistencia de los memos. Para ello en el IDE seleccionamos el directorio `model` y allí dentro creamos un archivo llamado `Query.js`. Con la ayuda de los snippets creamos una clase modelo.

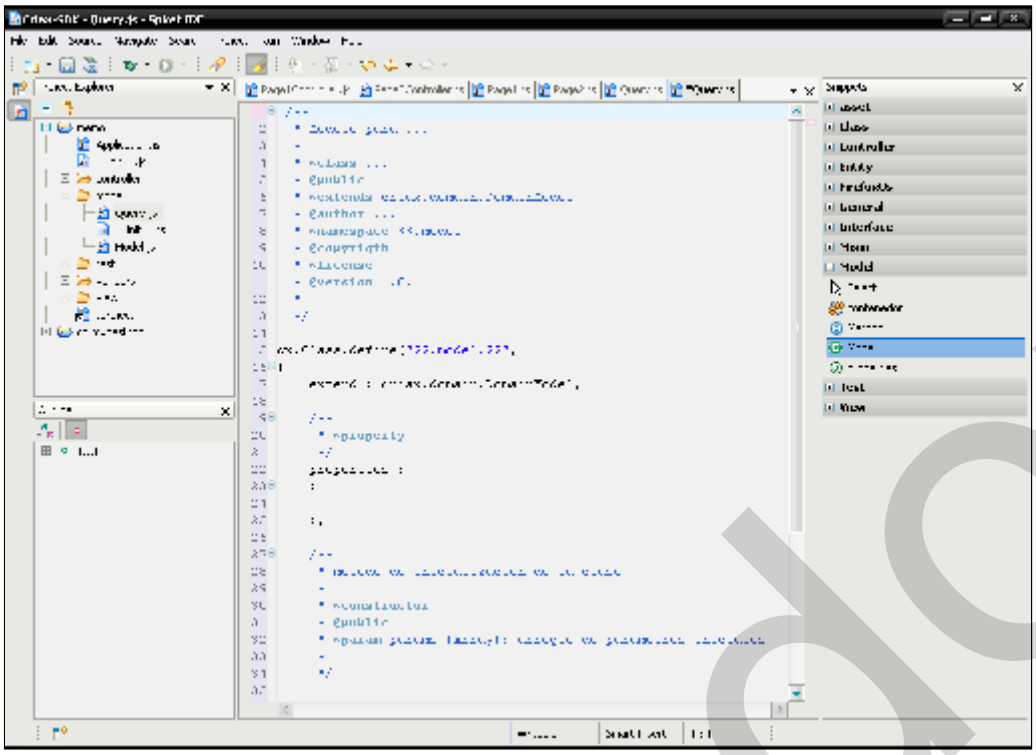

**Figura 2.2** "Nuevo modelo a partir de snippet"

Dicha clase será `singleton`, pues de esta manera siempre utilizaremos el mismo recurso para acceder a la información almacenada de los memos, quedando la clase inicial de esta manera:

```
/**
  * Modelo para Query
  *
  * @class Query
  * @public
  * @extends criax.domain.DomainModel
  * @author Nilmar Sanchez Muguercia
  * @namespace memo.model
  * @copyrigth
  * @license
  * @version 1.0.1
  *
  */
qx.Class.define("memo.model.Query",
{
    extend : criax.domain.DomainModel,
    type : "singleton",
```

```
    /**
     * @property
     */
    properties :
    {

    },

    /**
     * metodo de inicializacion de la clase
     *
     * @constructor
     * @public
     * @param params {Array}: arreglo de parametros
iniciales
     *
     */
    construct : function(params)
    {
        this.base(arguments);
    },

   /**
     * @method
     */
    members :
    {

    }
});
```

Dicha clase contendrá un atributo, el encargado de hacer las consultas al JSON.

```
/**
 * @property
 */
properties :
{
    /**
     * propiedad para objeto Json
```

```
        *
        * @name __trimQuery
        * @private
        * @type {cormx.dbal.driver.Json}
        *
        */
       __trimQuery : {}
},
```

Ahora debemos establecer el esquema para la persistencia de datos que no es más que una definición en forma de JSON de la base de datos. El esquema contiene una sola tabla:

```
"Memo" : {
        "id":{"type":"String"},
        "titulo":{"type":"String"},
        "contenido":{"type":"String"},
        "creado":{"type":"String"},
        "modificado":{"type":"String"}
        }
```

Seguidamente vamos al archivo de configuración de persistencia `memo/config/persistence.json`, en la sección de SCHEMA el valor de `criax.persistence.map.name` es la clave que contendrá el esquema (lo veremos más adelante) por defecto `criax.entities`. En la sección de CONNECTION el valor de la clave `driver` es `json`, el valor de `name`, será el nombre con el que se identificará la base de datos, y el `path` el medio donde almacenaremos la información para persistir. Quedando la configuración de la persistencia de la siguiente manera:

```
{
    "export" : ["build"],
    "jobs" :
    {
        "build" :
        {
            "environment" :
            {
                //SCHEMA
                "criax.persistence.map.dir" : "",//ruta/
```

```
por_defecto/resource/persistence/ en moviles no hce falta
                "criax.persistence.map.name" :
"criax.entities", //nombre de la constante de entorno
para tomar el esquema, pueden haber varios esquemas

                //
                "criax.persistence.db.driver" :
"json",//sqlite por defecto json para app moviles
                "criax.persistence.db.name" : "Db",
                "criax.persistence.db.path" :
"localStorage"//ruta/por_defecto/resource/persistence/ o
lugar donde persistir
            },

            //AUTOLOAD
            "use" :
            {
                "cormxdesktop.Application" :
                [

                ]
            }
        }
    }
}
```

Una vez configurada la persistencia, queda escribir el esquema en su archivo, este se encuentra en `memo/source/resource/persistence/schema.json`, en el valor de la clave `criax.entities`, escribiremos dentro de un `String`, el json compactado, **cuidado con las comillas**:

```
{
    "export" : ["build"],
    "jobs" :
    {
        "build" :
        {   "environment" :
            {
                "criax.entities" : '"Memo" :
{"id":{"type":"String"}, "titulo":{"type":"String"},
```

```
"contenido":{"type":"String"},
"creado":{"type":"String"},
"modificado":{"type":"String"}}'
            }
        }
    }
}
```

Como se puede apreciar se guardará la información del título, el contenido del memo, la fecha de creado y la fecha de la última modificación.

Seguidamente creamos un objeto del driver de persistencia para JSON que contiene CRIAX él se encargará de leer de la configuración el nombre de la base de datos así como el esquema para trabajar con la misma. Quedando el constructor de la clase de la siguiente manera:

```
/**
 * metodo de inicializacion de la clase
 *
 * @constructor
 * @public
 *
 */
construct : function()
{
    this.base(arguments);
    this.__trimQuery = new cormx.dbal.driver.Json();
},
```

Ahora si, a la codificación importante, vamos a comenzar necesitando un método para obtener todos los memos disponibles. Si seguimos más o menos un desarrollo basado en TDD primero debemos crear el test y después programamos.

**NOTA:**seguir al pie de la letra el Desarrollo Guiado por Pruebas **TDD**, si que resulta.

Capítulo 3

# El test

## 3.1 Entorno de test

Para realizar test en nuestras aplicaciones debemos modificar un poco la configuración de la misma. Lo primero será ir al directorio de configuración `memo/config/` y en el archivo de configuración `test.json` en la sección de `enviroment`, le ponemos `true` a `criax.test` y `criax.test.console.application`. El primero con el objetivo de que la aplicación sepa que correrá en modo de test y el segundo para mostrar el resultado de los test además de por consola, también en la aplicación.

```
{
    "export" : ["build"],
    "jobs" :
    {
        "build" :
        {
            "environment" :
            {
              "criax.test" : true,
              "criax.test.console.application" : true
            },
            //AUTOLOAD
            "use" :
            {
                "criax.unit.TestRunner" :
                [
```

```
                    "memo.test.DemoTest"
                ]
            }
        }
    }
}
```

**NOTA:** La sección `use` la veremos mas adelante.

Lo segundo será ir al directorio de los servicios `memo/source/resource/services`, abrir el archivo principal `services.json` y descomentariar la última línea que es la que carga el servicio de test.

```
/***** IMPORTADOR DE SERVICIOS *********/
{
    services: [
        //importar los servicios de persistencia
        //{include:"persistence"}
        //importar los servicios de los test
        {include:"test"}
    ]
}
```

Una vez configurada la aplicación corremos los test, para ello utilizamos el script en consola:

```
$ cd memo
$ toolchain/test(.bat o .sh)
```

Nos saldrá una pequeña ventana con los test que se ejecutaron, en este caso solo los que vienen por defecto al crear una aplicación.

**Figura 3.1** "Ejecutando los test"

## 3.2 Agregar un test

Para agregar un test a nuestra aplicación lo primero será: en el IDE seleccionar el directorio `test/unit`, allí crearemos un archivo llamado `QueryTest.js`, el mismo contendrá la clase para testear los métodos del modelo `Query`. Por convención las clases de los test se llamarán igual que su modelo, pero agregando detrás `Test`. Nos auxiliamos de los `snippets`, en la sección de test, para crear los mismos.

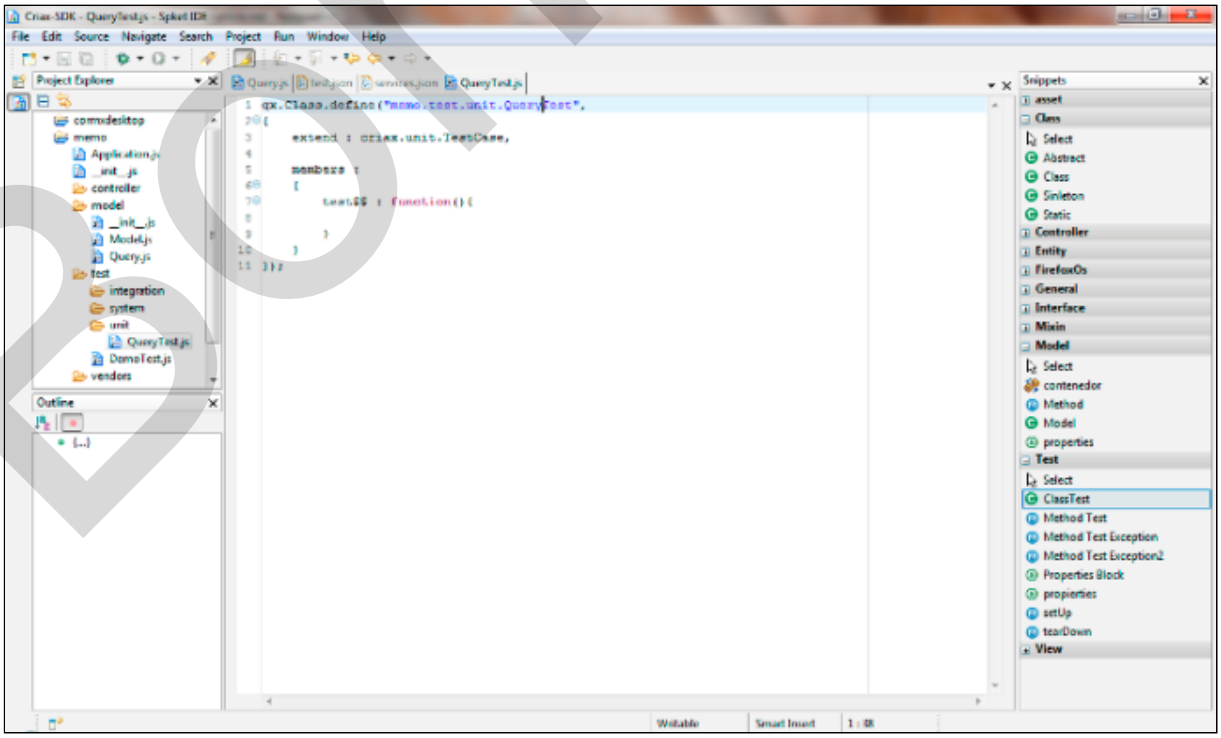

**Figura 3.2** "Creando un clase de test"

Ahora como inicialmente no hay ningún memo disponible, al inicio la aplicación creará un memo automático, y por lo tanto cuando listemos los memos la cantidad será uno. De esta manera podemos crear un test que devuelva `true` cuando la cantidad de memos iniciales de la aplicación sea uno. Para ello creamos un método llamado `testGetAllMemos`, si seguimos una convención los métodos de test se llamarán igual que los métodos a testear con el prefijo `test`.

```
testGetAllMemos : function(){
    var memos = ["memoinicial"];
    this.assertEquals(1,memos.length);
}
```

Para agregar esta clase de test, vamos al archivo de configuración de los test `memo/config/test.json`, y en la sección `use`, agregamos el namespace de la clase.

```
{
    "export" : ["build"],
    "jobs" :
    {
        "build" :
        {
            "environment" :
            {
              "criax.test" : true,
              "criax.test.console.application" : true
            },
            //AUTOLOAD
            "use" :
            {
                "criax.unit.TestRunner" :
                [
                    "memo.test.DemoTest",
                    "memo.test.unit.QueryTest"
                ]
            }
        }
```

```
    }
}
```

Esto se hace con el objetivo de que CRIAX cargue esa clase antes de la clase de ejecuta los test. El siguiente paso será agregar la clase de test al servicio de test, para ello abrimos el archivo de servicio de test en `memo/source/resource/services/test.json`.

```
/****** SERVICIOS PARA LOS TEST *********/
{
    services : [
        {
            id:"demo",
            clazz:"memo.test.DemoTest",
        },
        //servicio del Testrunner (debe ser el ultimo
servicio de los test)
        {
            id:"testrunner",
            clazz:"criax.unit.TestRunner",
            //arguments:["true"],//si se para al primer
error
            call:{
              1:["addClass","@demo"],//agregar una
clase a testear

2:["skipMethods","testAdvanced"]//saltarce este metodo
para no testear
                }
        }
    ]
}
```

Y según la convención de los test **ver archivo README.txt**, agregamos la nueva clase de test. A la misma se le establece un identificador y se pone el namespace. Cuando se agrega al TestRunner, se le establece un número para el orden de ejecución del test.

```
/****** SERVICIOS PARA LOS TEST *********/
{
```

```
    services : [
        {
            id:"demo",
            clazz:"memo.test.DemoTest",
        },
        {
            id:"query",
            clazz:"memo.test.unit.QueryTest",
        },
        //servicio del Testrunner (debe ser el ultimo
servicio de los test)
        {
            id:"testrunner",
            clazz:"criax.unit.TestRunner",
            //arguments:["true"],//si se para al primer
error
            call:{
                1:["addClass","@demo"],//agregar una
clase a testear

2:["skipMethods","testAdvanced"],//saltarce este metodo
para no testear
                3:["addClass","@query"]
                }
        }
    ]
}
```

Para correr el test nuevamente ejecutamos:

```
$ toolchain/test(.bat o .sh)
```

**Figura 3.3** "Test agregado"

Si el fondo del test sale en azul, es que todo se ejecutó correctamente, en amarillo significa que se saltó ese test, no lo ejecutó, y en rojo que es el test falló.

Ahora creamos un método real que podamos probar.

## 3.3 Testear código real

Volvemos al modelo `Query` y allí creamos un método para obtener todos los memos disponibles, lo llamaremos `getAllMemos` y como inicialmente no van a existir memos, insertaremos uno por defecto. Para ello nos auxiliamos de los snippets del IDE en la sección de modelos y creamos el método:

```
/**
 *  metodo para obtener todos los memos
 *  inserta un memo por defecto
 *
 * @method getAllMemos
 * @public
 * @return {Array} todos los memos
 *
 */
getAllMemos : function(){

}
```

Como se puede observar los snippets vienen con el comentario también, esto es para que al terminar la aplicación se pueda generar el API de la misma, para futuras mejoras. Para facilitar el trabajo podemos abrir el API de `cormx` y buscar la clase `cormx.dbal.driver.Json`. La misma se encuentrá en el directorio de la plataforma `.../Criax/documentation/API/cormx/index.html`.

Lo primero en el método será verificar que no exista ningún memo, para poder insertar el memo por defecto, para ello utilizamos el método `existDataInStorage`.

```
if(!this.__trimQuery.existDataInStorage()){

}
```

En caso de no existir memo, entonces insertamos un memo por defecto, con el método `insert`, donde le pasamos el nombre de lo que podemos llamar tabla y un esquema con el valor de las columnas o atributo.

```
var initData = {
                "id":"1",
                "titulo":"Memo ejemplo",
                "contenido":"Contenido memo ejemplo",
                "creado":"26/02/2014 (10:50:30)",
                "modificado":"26/02/2014 (10:50:30)"
               };
this.__trimQuery.insert("Memo",initData);
```

Por último retornamos todos los memos existentes con `all`, pasándole el nombre de la tabla.

```
return this.__trimQuery.all("Memo");
```

De esta manera el método queda:

```
/**
 *  metodo para obtener todos los memos
 *  inserta un memo por defecto
 *
 * @method getAllMemos
 * @public
 * @return {Array} todos los memos
```

```
 *
 */
getAllMemos : function(){
    if(!this.__trimQuery.existDataInStorage()){
        var initData = {
                "id":"1",
                "titulo":"Memo ejemplo",
                "contenido":"Contenido memo ejemplo",
                "creado":"26/02/2014 (10:50:30)",
                "modificado":"26/02/2014 (10:50:30)"
               };
        this.__trimQuery.insert("Memo",initData);
    }
    return this.__trimQuery.all("Memo");
}
```

Ahora que programamos un método que nos debe devolver un listado de memos, y en este caso hay uno, entonces verificamos que esto sea cierto. Para ello vamos al método del test y enlazamos el test con el modelo:

```
testGetAllMemos : function(){
    var query = memo.model.Query.getInstance();
    var memos = query.getAllMemos();
    this.assertEquals(1,memos.length);
}
```

Como el modelo `Query` es singleton, obtener la instancia del mismo se hace mediante el método estático `getInstance`. Nuevamente corremos los test con el comando `test`:

```
$ toolchain/test(.bat o .sh)
```

Si todo se ejecutó correctamente entonces pasamos a construir la interfaz visual, en caso de algún fallo revisar el código, *ojo con las comas en los JSON*.

Capítulo 4

# La presentación

Lo siguiente es crear la presentación para mostrar la lista de memos, para ello primero creamos un controlador llamado `MemoController` seleccionando el directorio `controller`, y nuevamente nos auxiliamos de los snippets en la sección de `Controller` -> `Mobile Controller`. El controlador es quién se encarga de manejar los eventos que ocurran en la vista; también toma el rol de punto de entrada para mostrar la vista y pasarle los datos al modelo donde está la lógica de negocio.

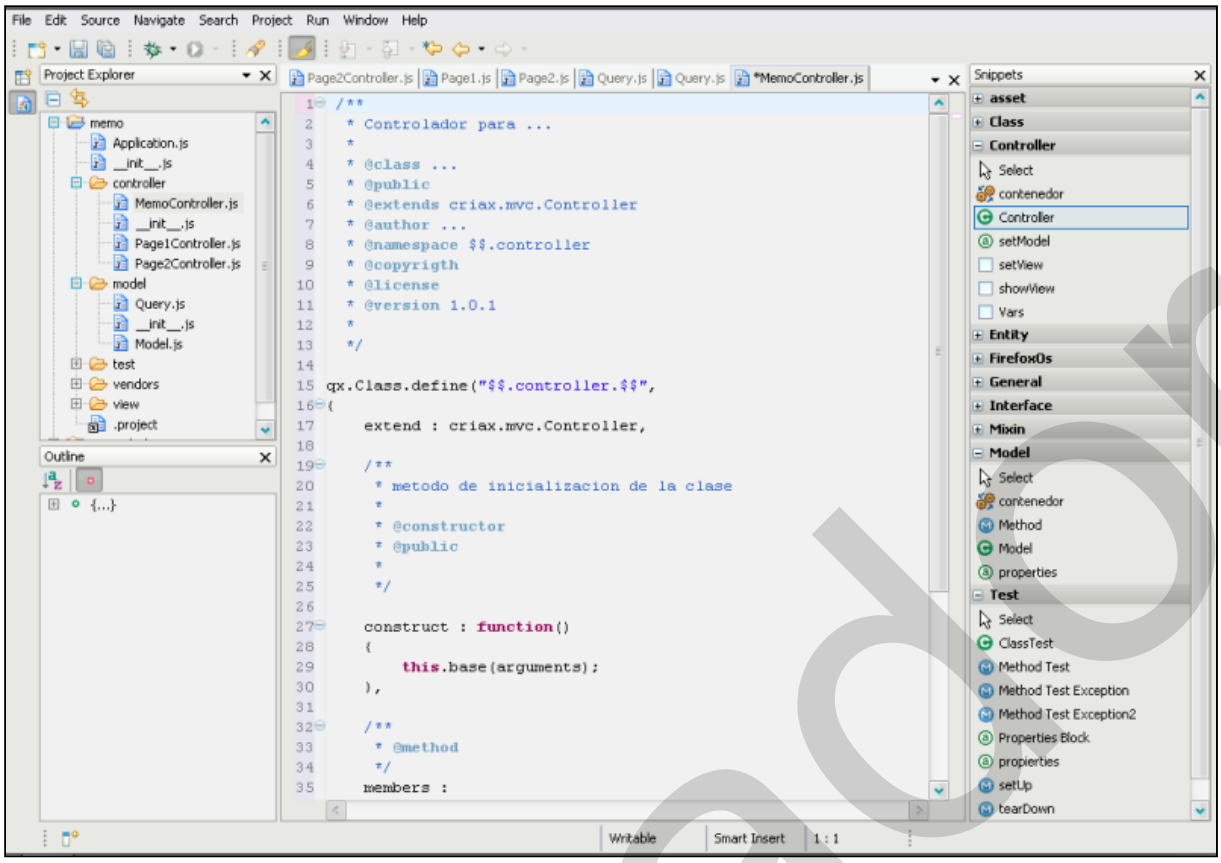


**Figura 4.1** "Crear controlador"

El código del controlador inicialmente queda:

```
/**
  * Controlador para Memo
  *
  * @class MemoController
  * @public
  * @extends criax.mobile.mvc.Controller
  * @author Nilmar Sanchez Muguercia
  * @namespace memo.controller
  * @copyrigth
  * @license
  * @version 1.0.1
  *
  */
qx.Class.define("memo.controller.MemoController",
{
    extend : criax.mobile.mvc.Controller,
```

```
    /**
     * metodo de inicializacion de la clase
     *
     * @constructor
     * @public
     *
     */
    construct : function()
    {
        this.base(arguments);
    },

    /**
     * @method
     */
    members :
    {
        /**
         * metodo inicial del controlador
         *
         * @method index
         * @public
         *
         */
        index : function(){

        }
    }
});
```

El método `index` es el punto de entrada del controlador para mostrar la vista. Ahí llamamos al método del modelo que nos devuelve todos los memos y creamos una variable para pasar esta información a la vista.

```
/**
 * metodo inicial del controlador
 *
 * @method index
 * @public
 *
```

```
 */
index : function(){
    var query = memo.model.Query.getInstance();
    var memos = query.getAllMemos();
    this.setVar({"memos":memos});
}
```

Lo siguiente es crear la clase de la vista, para ello seleccionamos en el IDE el directorio `view`, creamos un archivo llamado `Memo.js` y utilizamos los snippets en la sección de la `View` -> `Mobile Page`.

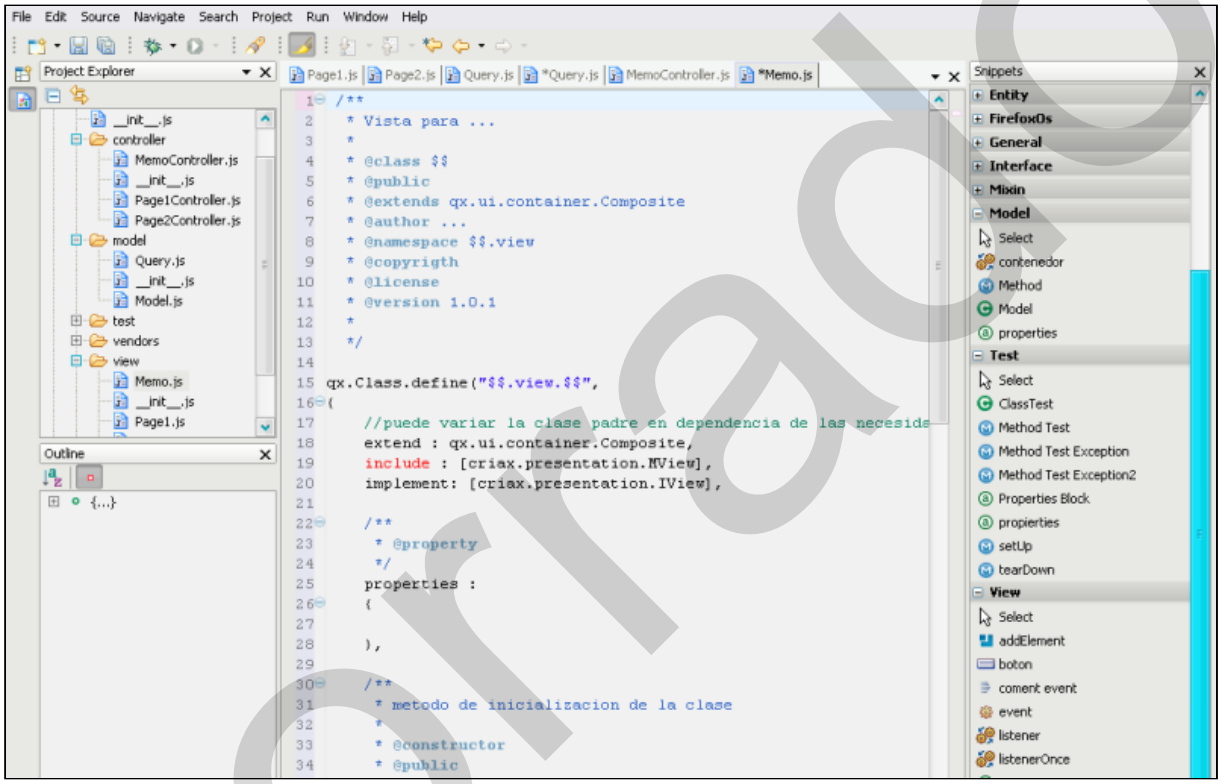


**Figura 4.2** "Crear vista"

La vista será una clase que contendrá elementos visuales, el código inicial queda:

```
/**
 * Clase para la vista de Memo
 *
 * @class Memo
 * @public
 * @extends criax.mobile.page.NavigationPage
 * @author Nilmar Sanchez Muguercia
 * @namespace memo.view
 * @copyrigth
```

```
   * @license
   * @version 0.0.1
   *
   */
qx.Class.define("memo.view.Memo",
{
    extend : criax.mobile.page.NavigationPage,
    include : [criax.mobile.presentation.MView,
qx.locale.MTranslation],
    implement: [criax.mobile.presentation.IView],

    /**
     * @property
     * */
    properties :
    {

    },

    /**
     * metodo de inicializacion de la clase
     *
     * @constructor
     * @public
     *
     */
    construct : function()
    {
        this.base(arguments);
        this.setTitle("Title");
    },

    /**
     * @method
     * */
    members :
    {
        /**
         * metodo de inicializacion de los componentes de
```

```
la vista
         *
         * @method components
         * @public
         *
         */
        components : function(){

        },

        /**
         * metodo para poner los componentes en la vista
         *
         * @method _initialize
         * @protected
         *
         */
        _initialize : function(){
            this.base(arguments);
            this.setOrangeBackground(true);
        }
    }
});
```

Para que todo sea más organizado, en el método `components` se inicializan cada uno de los componentes de la vista y en el método `_initialize` se configuran y se agregan los mismos a la vista.

Como esta vista y este controlador serán los que deben salir cuando cargue la aplicación, debemos configurar la misma para que los tome como vista y controlador de inicio. Para ello vamos al directorio de configuración y abrimos el archivo `boot.json`.

```
{
    export : ["build"],
    jobs :
    {
        build :
        {
            //INIT CLASS
```

```
            environment :
            {
                criax.init.controller :
"memo.controller.Page1Controller",
                criax.init.view : "memo.view.Page1",
                criax.init.domain :
"criax.domain.ModelFacade"
            },
            //AUTOLOAD
            use :
            {
                memo.Application :
                [
                    "memo.controller.Page1Controller",
                    "memo.view.Page1",
                    "criax.domain.ModelFacade"
                ]
            }
        }
    }
}
```

En la sección `environment`, se establecen los valores del controlador inicial, la vista inicial y la clase que servirá de fachada para el manejo de los modelos de dominio en caso de necesitarla. Ahí establecemos el valor de:

```
criax.init.controller : 'memo.controller.MemoController'
criax.init.view : 'memo.view.Memo'
```

En la sección `use`, se establecen los namespace de las clases que la aplicación necesita cargar antes de iniciar, por lo tanto modificamos los namespace del controlador inicial y de la vista inicial de manera tal que queden los de la aplicación Memo. Quedando finalmente:

```
{
    export : ["build"],
    jobs :
    {
        build :
        {
```

```
            //INIT CLASS
            environment :
            {
                criax.init.controller :
"memo.controller.MemoController",
                criax.init.view : "memo.view.Memo",
                criax.init.domain :
"criax.domain.ModelFacade"
            },
            //AUTOLOAD
            use :
            {
                memo.Application :
                [
                    "memo.controller.MemoController",
                    "memo.view.Memo",
                    "criax.domain.ModelFacade"
                ]
            }
        }
    }
}
```

Como la aplicación esta en modo de `test`, debemos volverla a poner en modo normal, por lo cual llevamos a cabo el proceso inverso: abrimos el archivo de configuración de los test y ponemos `false` en los valores de `criax.test` y `criax.test.console.application`. Luego comentariamos en el archivo principal de los servicios, el servicio que carga los tests. Una vez hecho esto, actualizamos la aplicación:

```
$ toolchain/update(.bat o .sh)
```

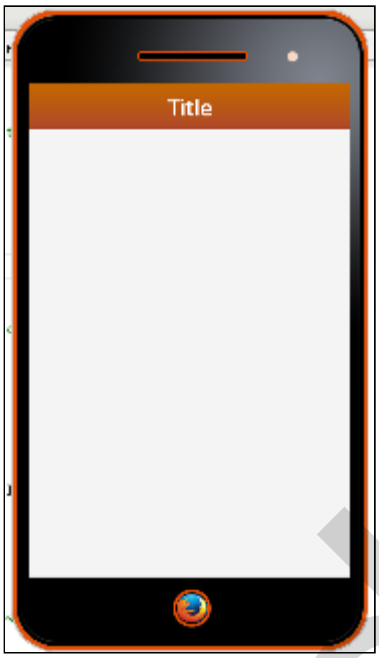


**Figura 4.3** "Nueva página de navegación"

En la vista mostraremos los memos en un elemento de lista, para ello primero creamos un atributo de la vista que contendrá el elemento de la lista:

```
/**
 * @property
 * */
properties :
{
    /**
     * propiedad para el listado de memos
     *
     * @name memosList
     * @public
     * @type {criax.mobile.list.List}
     *
     */
    memosList : {}
},
```

Lo inicializamos en el método `components`:

```
/**
 * metodo de inicializacion de los componentes de la vista
 *
```

```
     * @method components
     * @public
     *
     */
    components : function(){
        this.memosList = new criax.mobile.list.List();
    },
```

Y finalmente lo agregamos a la página:

```
    /**
     * metodo para poner los componentes en la vista
     *
     * @method _initialize
     * @protected
     *
     */
    _initialize : function(){
        this.base(arguments);
        this.setOrangeBackground(true);

        this.addContent(this.memosList);
    }
```

Como la lista esta vacía no mostrará nada apenas una rayita.

Para llenar la lista debemos configurar cada uno de sus elementos o `items`. Como la información que devuelve el método `getAllMemos` es un arreglo de objetos, debemos configurar un método que se encargará de establecer el valor de los distintos elementos configurables de un `item` de la lista, como su label, sublabel, si mostrará una flecha o no y su icono, este método tendrá por parámetro un objeto para configurar la función sobre la que se delega esta opción.

```
this.memosList.setConfigureItem(this.__createItemList);
```

Después se establece el modelo:

```
this.memosList.setArrayModel(this.vars.memos);
```

Quedando el método de inicialización:

```
/**
 * metodo para poner los componentes en la vista
 *
 * @method _initialize
 * @protected
 *
 */
_initialize : function(){
    this.base(arguments);
    this.setOrangeBackground(true);

this.memosList.setConfigureItem(this.__createItemList);
    this.memosList.setArrayModel(this.vars.memos);
    this.addContent(this.memosList);
},
```

Ahora queda desarrollar el método sobre el cual se delega la responsabilidad de configurar los `items` de la lista:

```
/**
 *  metodo para crear los elementos de la lista
 *
 * @method __createItemList
 * @private
 * @param item {Object} un elemento del componente de la
lista
 * @param data {Object} un elemento del arreglo de datos
 * @param row {Integer} numero de la fila
 *
 */
__createItemList : function(item, data, row){
    item.setName(data.id);
    item.setTitle(data.titulo);
    item.setSubtitle(data.modificado);
    item.setShowArrow(true);
}
```

Para poder poner un icono delante del título del elemento de la lista agregamos un `asset` a la clase. Los assets no son más que los recursos que se utilizarán en la aplicación: imágenes, flash, entre otras cosas, se pone la ruta a partir de los

recursos de la aplicación o a partir de los recursos de las librerías a utilizar. En el presente caso utilizaremos algunos iconos de la interfaz Gaia de FirefoxOs, que se encuentran dentro de los recursos de Criax. Los assets, se ponen dentro de los comentarios de la clase como se puede observar a continuación:

```
/**
 * Clase para Listado de memos
 *
 * @class Memo
 * @public
 * @extends criax.mobile.page.NavigationPage
 * @author Nilmar Sanchez Muguercia
 * @namespace memo.view
 * @copyrigth
 * @license
 * @version 0.0.1
 *
 * @asset(criax/icon/gaia/edit.png)
 */
```

Para usar el asset, en el método de configuración de los items, agregamos:

```
item.setImage("criax/icon/gaia/edit.png");
```

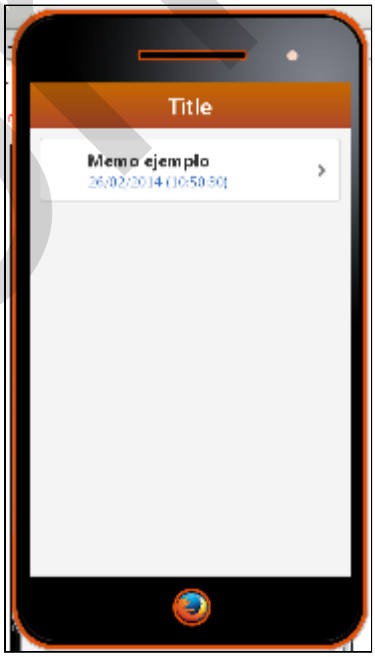


**Figura 4.4** "Listado de memos"

Ahora configuremos de manera más acorde la página, poniéndole el título y un botón para la opción de agregar un nuevo memo. Estas acciones las llevamos a cabo en el constructor de la vista:

```
/**
 * metodo de inicializacion de la clase
 *
 * @constructor
 * @public
 *
 */
construct : function()
{
    this.base(arguments);
    this.setTitle("Memos");
    this.setShowButton(true);
    this.setButtonIcon("criax/icon/gaia/add.png");
},
```

El icono por supuesto primeramente tenemos que agregarlo como `asset`. Si dejamos apretado el memo de ejemplo el fondo se pondrá azul y mostrará a la izquierda el icono de editar el mismo. Para que el mismo se vea siempre pondremos las listas con un fondo negro para eso como segundo parámetro al crear la lista le pasamos la constante BLACK de la clase lista, en el método `components`:

```
this.memosList = new
criax.mobile.list.List(null,criax.mobile.list.List.BLACK);
```


**Figura 4.5** "Listado de memos: diseño 2"

Ahora ya tenemos completa la página con el listado de los memos. Pasamos a la siguiente parte, crear una página, donde podamos gestionar los memos, entiéndase, agregar un memo, eliminar un memo y modificar la información de un memo.

# II PARTE: Gestionando memos

Capítulo 5

# Agregando un memo

## 5.1 Inicio de gestión

Para la gestión que se llevara a cabo en los memos, haremos el clásico CRUD. Las acciones de inserción actualización y eliminación se llevaran a cabo en la misma página para poder reutilizar los elementos que se irán construyendo poco a poco a medida que avanza el tutorial. Algunos elementos de la primera parte son omitidos, debido a su temprana explicación, es el caso de los assets y algunas líneas de código que componen los métodos, por esta razón para mostrar código de varios métodos al mismo tiempo se separaran los mismos, por **...**.

## 5.2 Test

Por supuesto que lo primero que haremos será crear un test para un método que agregue un memo a la lista, ya la vía la conocemos, configuramos la aplicación para que pueda ser testeada, editamos el archivo de configuración de los test `memo/config/test.json` y el archivo principal de los servicios `memo/source/resource/services/services.json`. Abrimos la clase de test `QueryTest` y agregamos un método llamado `testInsert` y al resultado del listado de memos le agregamos un objeto con los datos del memo. Esta sería la forma más simple para verificar que nuestro test funciona correctamente.

```
testInsert : function(){
    var query = memo.model.Query.getInstance();
    var memos = query.getAllMemos();
    var newMemo = {
            "id":"2",
            "titulo":"Nuevo Memo",
```

```
                "contenido":"Contenido del nuevo memo",
                "creado":"12/03/2014 (13:25:00)",
                "modificado":"12/03/2014 (13:25:00)"
            };
        memos.push(newMemo);
        this.assertEquals(2,memos.length);
    }
```

Corremos los tests y verificamos que todo funcionó correctamente con el nuevo test.

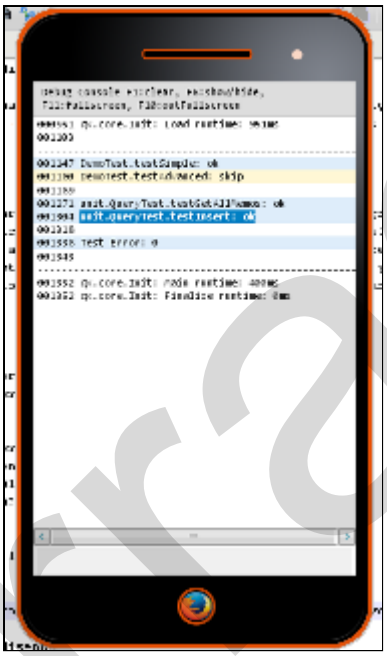

**Figura 5.1** "Test de inserción"

Ahora vayamos al código real, en el modelo crearemos un método llamado `save` para insertar cada uno de los memos nuevos. En el mismo utilizaremos el método `insert` de `JSON`, pasándole como parámetros el nombre de la tabla y el objeto con la información del nuevo memo. Como el nombre de la tabla es el mismo, el método `save` tendrá un solo parámetro el objeto con la información:

```
/**
 *  metodo para adicionar un test
 *
 * @method save
 * @public
 * @param newMemo {Object} informacion del nuevo memo
 *
```

```
 */
save : function(newMemo){
    this.__trimQuery.insert("Memo",newMemo);
}
```

Ahora el método del test lo modificamos para que utilice el nuevo método del modelo.

```
testInsert : function(){
    var query = memo.model.Query.getInstance();
    var memos = query.getAllMemos();

    var newMemo = {
            "id":"2",
            "titulo":"Nuevo Memo",
            "contenido":"Contenido del nuevo memo",
            "creado":"12/03/2014 (13:25:00)",
            "modificado":"12/03/2014 (13:25:00)"
        };
   query.save(newMemo);
   var newListMemos = query.getAllMemos();
   this.assertEquals(2,newListMemos.length);
}
```

Corremos los tests y verificamos que todo haya salido correctamente.

## 5.3 La presentación

Ahora crearemos la presentación para poder adicionar un memo. Primeramente creamos el controlador de la vista `ManageMemoController`:

```
/**
  * Controlador para ManageMemo
  *
  * @class ManageMemoController
  * @public
  * @extends criax.mobile.mvc.Controller
  * @author Nilmar Sanchez Muguercia
  * @namespace memo.controller
  * @copyrigth
  * @license
```

```
   * @version 1.0.1
   *
   */
qx.Class.define("memo.controller.ManageMemoController",
{
    extend : criax.mobile.mvc.Controller,

    /**
     * metodo de inicializacion de la clase
     *
     * @constructor
     * @public
     *
     */
    construct : function()
    {
        this.base(arguments);
    },

    /**
     * @method
     */
    members :
    {
        /**
         * metodo inicial del controlador
         *
         * @method index
         * @public
         *
         */
        index : function(){

        }
    }
});
```

Y seguidamente la vista `ManageMemo`:

```
 /**
  * Clase para ManageMemo
  *
  * @class ManageMemo
  * @public
  * @extends criax.mobile.page.NavigationPage
  * @author Nilmar Sanchez Muguercia
  * @namespace memo.view
   * @copyrigth
  * @license
  * @version 0.0.1
  *
  */
qx.Class.define("memo.view.ManageMemo",
{
    extend : criax.mobile.page.NavigationPage,
    include : [criax.mobile.presentation.MView,
qx.locale.MTranslation],
    implement: [criax.mobile.presentation.IView],

    /**
     * @property
     * */
    properties :
    {

    },

    /**
     * metodo de inicializacion de la clase
     *
     * @constructor
     * @public
     *
     */
    construct : function()
    {
        this.base(arguments);
        this.setTitle("Title");
```

```
    },

    /**
     * @method
     * */
    members :
    {
        /**
         * metodo de inicializacion de los componentes de
la vista
         *
         * @method components
         * @public
         *
         */
        components : function(){

        },

        /**
         * metodo para poner los componentes en la vista
         *
         * @method _initialize
         * @protected
         *
         */
        _initialize : function(){
            this.base(arguments);
            this.setOrangeBackground(true);
        }
    }
});
```

Ahora queremos que cuando demos un toque al símbolo + en la barra de título nos vaya a esta nueva página donde podamos agregar el memo, para ello en la vista inicial `Memo`, agregamos el evento `action` a la página, el mismo se dispara al tocar el botón derecho de la barra de título. Por supuesto como esto no es un componente el evento lo agregamos en el método `_inicilize` :

```
//en la vista inicial Memo.js
this.addListener("action",
this._controller.newMemo,this._controller);
```

**NOTA:** este evento lo debemos poner antes de agregar los componentes a la vista.

Como explicamos anteriormente los controladores manejan los eventos que se produzcan en la vista de esta forma tenemos los eventos de la vista agrupados y la clase misma se torna más limpia encargándose solamente de la configuración de los widgets. Por lo tanto en el controlador de la vista `Memo`, creamos un método llamado `newMemo`, que se ejecutará cada vez que se toque el botón derecho de la barra de título de la página.

```
/**
 * evento para agregar un nuevo memo
 * @event newMemo
 * @type action
 * @public
 *
 */
newMemo : function(objEvent) {

}
```

El mismo tiene un parámetro `objEvent`, este es el objeto del evento ejecutado. Este método ejecutara un código que nos permitirá pasar a la siguiente página.

```
this.assambler(memo.controller.ManageMemoController,
memo.view.ManageMemo);
```

Si se observa la API de este método se podrá verificar que lleva otros parámetros como el método a ejecutar los parámetros a pasar y alguna animación que se quiera poner en la transición entre páginas. Por defecto el método a ejecutar es `index` y no lleva parámetros.

Salimos de la configuración de test y actualizamos

```
$ toolchain/update(.bat o .sh)
```

Cuando corramos la aplicación damos un toque al signo de + y nos llevara a la página nueva.


**Figura 5.2** "Página para la gestión de memos"

Configuraremos la página, agregando un botón a la izquierda de la barra de títulos para volver al listado de memos, un botón a la derecha para la información sobre la aplicación. También le cambiaremos el título.

```
/**
 * metodo de inicializacion de la clase
 *
 * @constructor
 * @public
 *
 */
construct : function()
{
    this.base(arguments);
    this.setTitle("Add Memo");
    this.setShowButton(true);
    this.setButtonIcon("criax/icon/gaia/menu.png");
    this.setShowBackButton(true);
    this.setBackButtonIcon("criax/icon/gaia/back.png");
},
```

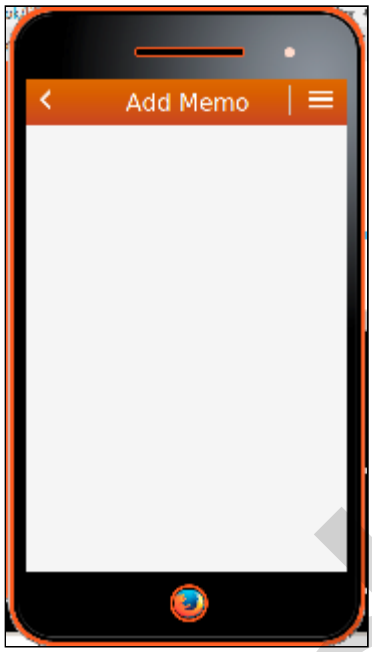


**Figura 5.3** "Configuración de la nueva página"

También agregaremos una barra de herramientas en la parte de abajo de la página, que nos permitirá trabajar con los memos. La barra de herramientas ya si es un componente por lo tanto tenemos que crear un atributo de la clase para la misma e inicializarlo en el método `components`:

```
//EN LAS PROPIEDADES
/**
 * propiedad para la barra de herramientas
 *
 * @name toolBar
 * @public
 * @type {qx.ui.mobile.toolbar.ToolBar}
 *
 */
toolBar : {}
...
//EN EL METODO COMPONENTS
this.toolBar = new qx.ui.mobile.toolbar.ToolBar();
...
//EN EL METODO _INICIALIZE
this.add(this.toolBar);
```

Aquí podemos ver un detalle y es que para poner la barra de herramientas se utiliza el método `add` en vez de `addContent`, esto se hace así para que la barra

de herramientas pueda situarse en la parte inferior de la pantalla y no interfiera con los scroll de la página.

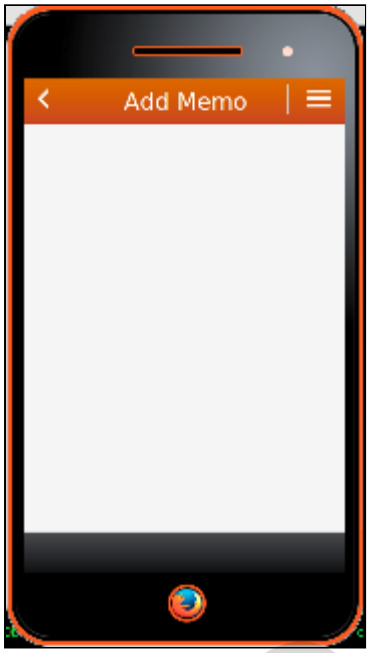


**Figura 5.4** "Página para gestionar los memos"

Ahora, barra de herramientas al fin, debe tener varias opciones con las que trabajar, para ello a la barra de herramientas le agregamos botones, que se encargaran de las diferentes labores, por el momento solo tendremos una, la opción de eliminar. Creamos un botón de tipo toolbar y lo agregamos a la barra. Dicho botón no tendrá testo sino una imagen.

**NOTA:** recordar que la ruta de las imágenes hay que agregarlas al asset en el comentario de la clase.

```
//EN LAS PROPIEDADES
/**
 * propiedad para el boton de eliminar memo
 *
 * @name buttonDelete
 * @public
 * @type {qx.ui.mobile.toolbar.Button}
 *
 */
buttonDelete : {}
...
//EN EL METODO COMPONENTS
this.buttonDelete = new
```

```
qx.ui.mobile.toolbar.Button(null,"criax/icon/gaia/
delete.png");
...
//EN EL METODO _INICIALIZE
this.toolBar.add(this.buttonDelete);
```

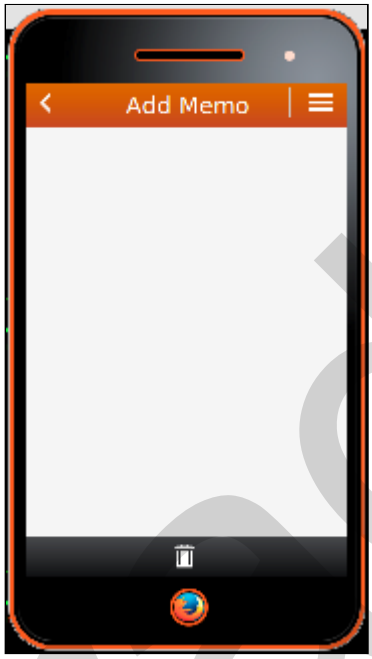


**Figura 5.5** "Página con barra de herramientas"

Como hay un solo botón, se centra y ocupa todo el ancho de la página, pero a medida que se vayan agregando botones los mismo se van acomodando de manera tal que el espacio que ocupan sea parejo.

## 5.4 La configuración

Ahora pasemos a cosas más importantes, nos falta que cuando demos en la flecha de la izquierda de la página vaya al listado de memos y cuando toquemos el botón de menú se despliegue la información sobre la aplicación. Comencemos por crear el evento para ir a la pantalla de inicio, para ello a la página le ponemos una escucha para el evento `back`:

```
this.addListener("back",this._controller.back,this._controller);
```

Y dicho evento será manejado en el controlador de la vista. Creamos un método en el controlador llamado `back` y al mismo le agregamos el simple código:

```
this.assambler(memo.controller.MemoController,
memo.view.Memo, "index", [], {reverse:true});
```

Aquí se establece el controlador, vista y método para el cual se navegará, y también una animación para la transición. De esta manara cuando pinchemos en la flecha de la izquierda iremos a la lista de memos. Ahora despleguemos un popup con la información de la aplicación en el icono de menú. Para ello crearemos 2 atributos en la vista un popup y un label con el testo de la información, seguidamente inicializaremos estos 2 componentes y le diremos al popup que su contenido es el label y estará asociado al botón derecho de la barra de navegación, por último le estableceremos un título al popup.

```
//CREANDO LOS COMPONENTES EN LAS PROPIEDADES
/**
 * propiedad para el popup de informacion
 *
 * @name popupInfo
 * @public
 * @type {Popup}
 *
 */
popupInfo : {},

/**
 * propiedad para el label con la informacion
 *
 * @name labelInfo
 * @public
 * @type {Label}
 *
 */
labelInfo : {}
...
//INICIALIZANDO LOS COMPONENTES
this.labelInfo = new qx.ui.mobile.basic.Label("CRIAX Memo
application");
this.popupInfo = new
qx.ui.mobile.dialog.Popup(this.labelInfo,
this.getNavegationBarRightButton());
...
//EN EL METODO _INITIALIZE
this.popupInfo.setTitle("Information");
```

Los componentes de popup tienen una particularidad, los mismos no se agregan a la página sino que se mantienen ocultos y cuando se quieran mostrar, se llama su método `show`. En este caso, queremos mostrar el popup con la información cuando toquemos el botón derecho, para ello ponemos una escucha en el evento `action` de la página que ejecutara un método del controlador llamado `info`.

```
this.addListener("action", this._controller.info,
this._controller);
```

En el controlador creamos el método y mostramos el popup:

```
/**
 * evento para mostrar la informacion de la aplicacion
 * @event info
 * @type action
 * @public
 *
 */
info : function(objEvent){
    var view = this.getView();
    var popup = view.popupInfo;
    popup.show();
}
```

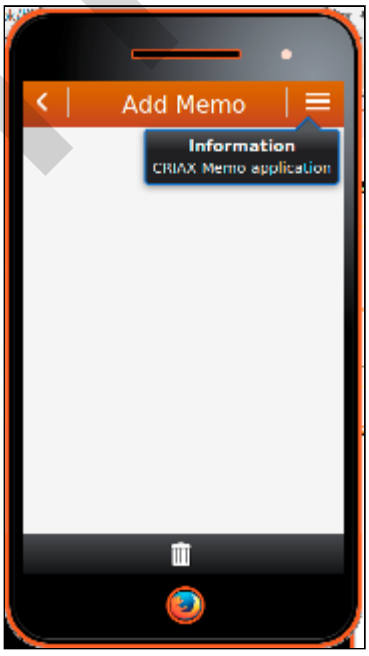


**Figura 5.6** "Popup"

Capítulo 6

# El formulario

En este punto solo nos falta un formulario para poder adicionar la información del memo, aquí utilizaremos uno de los nuevos componentes experimentales aun en desarrollo, de la versión 1.1 de CRIAX-SDK, los formularios. Los mismo se crearon para dar más facilidad a la hora de trabajar con ellos.

Lo primero será crear dentro del directorio de las vistas, un paquete que contendrá los formularios llamado `forms` y dentro del mismo una clase exclusivamente para este componente, la misma se llamara `ManageMemo`, la misma hereda de la clase de formulario `criax.mobile.form.Form`.

```
/**
 *  Clase para el formulario de gestion de memos
 *
 * @class ManageMemo
 * @extends criax.mobile.form.Form
 * @author Nilmar Sanchez Muguercia
 * @namespace memo.view.forms
 * @copyrigth
 * @license
 * @version 1.0.1
 *
 */
qx.Class.define("memo.view.forms.ManageMemo",
{
    extend : criax.mobile.form.Form,
    /**
```

```
     * @property
     */
    properties :
    {

    },

    /**
     * metodo de inicializacion de la clase
     *
     * @constructor
     * @public
     * @param view {MVIew} vista que contiene el
formulario
     * @param header {String} testo de encabezado del
formulario
     *
     */
    construct : function(view,header)
    {
        this.base(arguments,view,header);
    },

   /**
     * @method
     */
    members :
    {

    }
});
```

De manera sencilla cada campo del formulario será un atributo de esta clase y el formulario contendrá un campo para el título del memo y otro para el contenido. Las fechas se gestionan de manera automática.

```
/**
 * @property
 */
properties :
```

```
{
    /**
     * propiedad para el campo de titulo
     *
     * @name title
     * @public
     * @type {criax.mobile.form.text.TextField}
     *
     */
    title : {},

    /**
     * propiedad para el campo de contenido
     *
     * @name content
     * @public
     * @type {criax.mobile.form.text.TextArea}
     *
     */
    content : {}
},
```

Seguidamente crearemos un método llamado `input` donde configuraremos cada campo de formulario, su validación, filtro, etc.

```
/**
 *  metodo para configurar los campos del formulario
 *
 * @method input
 * @public
 *
 */
input : function(){
    this.title = new criax.mobile.form.text.TextField();
    this.title.setLabel("Title:");
    this.title.setRequired(true);
    this.content.addValidation("presence","Title
empty",true);
    this.title.addValidation("alpha","Letters and
numbers",true);
```

```
    this.title.addValidation("max-length","Title too
large",true,[20]);

    this.content = new criax.mobile.form.text.TextArea();
    this.content.setLabel("Content:");
    this.content.setRequired(true);
    this.content.addValidation("presence","Content
empty",true);
    this.content.addValidation("max-length","Content too
large",true,[255]);
}
```

En el caso del campo de título, le decimos que es requerido y lo validamos para que solo se puedan poner letras y números (alpha) y tenga una longitud máxima de 20 caracteres, en el caso del contenido es muy parecido excepto que se puede escribir cualquier cosa mientras no se deje vacío.

Seguidamente creamos un método llamado `form` que se encargará de agregar los campos al formulario así como la configuración del mismo. Lo primero será decirle al formulario que se valide solamente cuando sea enviado, de esta manera los campos no seran validados mientras se escribe, tambien debemos decirle que filtros debe aplicar a sus campos.

**NOTA:** como las mismas características son aplicadas a los 2 campos (la validacion solo cuando se envia el formulario y los filtros), cuando sucede esto se pueden aplicar desde el formulario.

Se le agregan sus campos y el nombre con el que se identificaran los mismos. Seguidamente agregamos un botón para adicionar el memo y uno para cancelar. A ambos botones se le debe pasar un callback que será el código a ejecutar cuando se dispare su evento `execute`.

```
/**
 *  metodo para agregar los campos al formulario y
configurar el mismo
 *
 * @method form
 * @public
 *
 */
form : function(){
```

```
        this.validateOnlyOnSubmit();
        this.addFilters(["first-up","clean"]);

        this.addField(this.title,"title");
        this.addField(this.content,"content");

        this.setSaveButton("Add");
        this.setResetButton("Cancel");
    }
```

Ahora nos queda mostrar el formulario creado, para ello vamos a la página de gestión y creamos un atributo que representará el formulario. Inicializamos ese atributo en `components`, pasándole el objeto de la vista a la cual estará asociado y el testo de encabezado del formulario y lo configuramos en `_initialize` llamando a los métodos `input` y `form`. Por último que nos queda mostrar el formulario renderizado. En las aplicaciones móviles existen 2 tipos de renderizados, el `SINGLE` que nos mostrará los campos con su label uno debajo del otro, y el `PLACEHOLDER`, que omitirá las etiquetas labels de los campos y pondrá esta información en el placeholer de cada campo, para ir entendiendo primeramente veremos la primera opción y más adelante configuraremos para utilizar la segunda.

```
//EN LAS PROPIEDADES
 /**
 * propiedad para el formulario de agregar memo
 *
 * @name formMemo
 * @public
 * @type {Forms}
 *
 */
formMemo : {}
...
//EN COMPONENTS
this.formMemo = new memo.view.forms.ManageMemo(this,"Add
Memo");
...
//EN _INITIALIZE
this.formMemo.setGroupBox();
this.formMemo.input();
```

```
this.formMemo.form();
this.addContent(
this.formMemo.getRender(criax.mobile.form.Form.SINGLE));
```

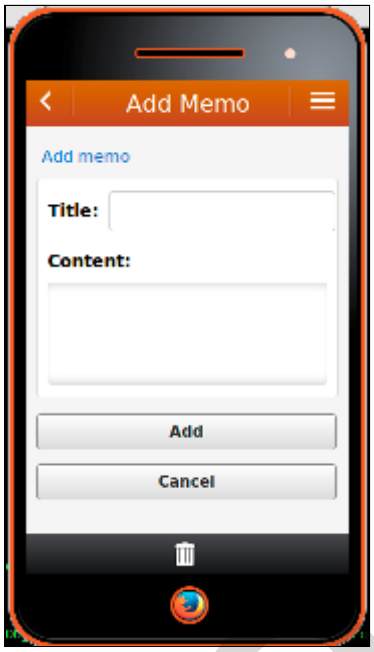


**Figura 6.1** "Página con formulario"

Así no se ve mal, pero realmente podría verse mejor. Quitemos el encabezado y pongamos los campos sin label. Para ello en el método `_initialize` cambiamos por:

```
this.formMemo.input();
this.formMemo.form();
this.addContent(
this.formMemo.getRender(criax.mobile.form.Form.PLACEHOLDER));
```

Por otro lado en la clase de formulario a los botones les ponemos un estilo negro.

```
this.setSaveButton("Add",null,criax.mobile.form.Button.BLACK);
this.setResetButton("Cancel",null,criax.mobile.form.Button.BLACK);
```

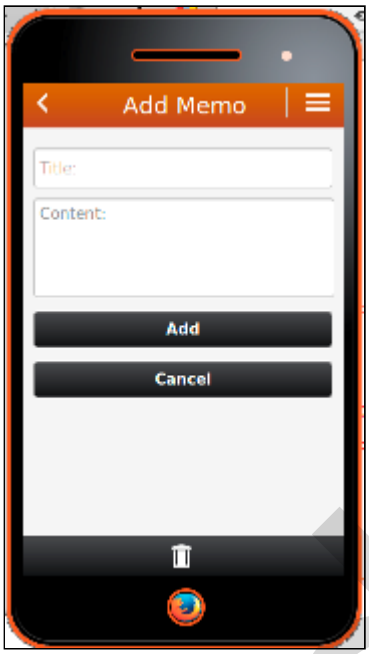


**Figura 6.2** "Página con formulario mejorado"

Ahora nos queda persistir la información obtenida del formulario. Para ello agregamos un método llamado `submit` en el controlador, que se ejecutara al dar tocar el botón de aceptar, pasamos ese método como callback al método `form`, del formulario y el contexto en el que se desarrollara esta función:

```
//EN EL CONTROLADOR DE LA VISTA
/**
 *  metodo para enviar y validar los datos del formulario
 *
 * @method submit
 * @public
 *
 */
submit : function(){

}
...
//EN LA VISTA
this.formMemo.form(this._controller.submit,this._controller);
...
//EN EL FORMULARIO
/**
 *  metodo para agregar los campos al formulario y
configurar el mismo
```

```
     *
     * @method form
     * @public
     *
     */
    form : function(execution,contest){
        this.validateOnlyOnSubmit();
        this.addFilters(["first-up","clean"]);

        this.addField(this.title,"title");
        this.addField(this.content,"content");

        this.setSaveButton("Add", null,
    criax.mobile.form.Button.BLACK, execution, contest);
        this.setResetButton("Cancel", null,
    criax.mobile.form.Button.BLACK);
    }
```

Por defecto el botón de cancelar limpiara los campos del formulario. En el controlador de la vista, en el método `submit`, lo primero es verificar que el formulario es válido y en caso de serlo, entonces obtenemos la información del mismo filtrada. Formateamos la fecha (para ello creamos un método en el modelo `Query`, llamado `formatDate`):

```
    /**
     *  metodo para formatear la fecha
     *
     * @method formatDate
     * @public
     * @return {String} fecha formateada
     *
     */
    formatDate : function(){
        var date = new Date();
        return
    date.getDate()+"/"+(date.getMonth()+1)+"/"+date.getFullYear()+"
    ("+date.getHours()+":"+date.getMinutes()+":"+date.getSeconds()+")";
    }
```

Para persistir los datos:

```
/**
 *  metodo para enviar y validar los datos del formulario
 *
 * @method submit
 * @public
 *
 */
submit : function(){
    var view = this.getView();
    var form = view.formMemo;
    if(form.validate()){
        var filtrateFormInfo = form.getFiltrateModel();
        var query = memo.model.Query.getInstance();
        var memoData = {};
        memoData.id = "2";
        memoData.titulo = filtrateFormInfo.getTitle();
        memoData.contenido =
filtrateFormInfo.getContent();
        memoData.creado = query.formatDate();
        memoData.modificado = query.formatDate();
        query.save(memoData);
    }
}
```

Como se puede observar el id no siempre puede ser `2`, resolvamos esto de manera sencilla. Para ello creamos un método en el modelo `Query` llamado `__autoIncrement`.

```
/**
 *  metodo para poner el id autoincrementado
 *
 * @method __autoIncrement
 * @public
 * @return {Integer} numero incrementado
 *
 */
__autoIncrement : function(){
    var lastMemo = this.__trimQuery.execute("SELECT *
FROM Memo");
    if(lastMemo.length == 0){
```

```
            return 1;
        }
        var init = parseInt(lastMemo[0].id);
        for(var i=0;i<lastMemo.length;i++){
            var id = parseInt(lastMemo[i].id);
            if(id > init){
                init = id;
            }
        }
        init++;
        return init;
    }
```

De esta manera cuando agreguemos un memo nuevo, el valor del `id` lo podemos dejar vacío. En caso de que la lista de memos este vacía simplemente devolvemos 1. Solo queda aumentar el id en el memo agregado:

```
save : function(newMemo){
    newMemo.id = this.__autoIncrement();
    this.__trimQuery.insert("Memo",newMemo);
},
```

Para terminar al agregar el memo, que nos vaya a la página del listado, para ver el memo agregado. Recuerdan en el controlador `ManageMemoController`, habíamos creado un evento `back`, para cuando se tocara la flecha izquierda fuera al listado, bueno simplemente, reutilizando código tenemos que llamar este método al final del método submit y ya está.

```
/**
 *  metodo para enviar y validar los datos del formulario
 *
 * @method submit
 * @public
 *
 */
submit : function(){
    var view = this.getView();
    var form = view.formMemo;
    if(form.validate()){
        var filtrateFormInfo = form.getFiltrateModel();
```

```
        var query = memo.model.Query.getInstance();
        var memoData = {};
        memoData.titulo = filtrateFormInfo.getTitle();
        memoData.contenido =
filtrateFormInfo.getContent();
        memoData.creado = query.formatDate();
        memoData.modificado = query.formatDate();
        query.save(memoData);
        this.back();
    }
}
```

Capítulo 7

# Actualizar un memo

Para actualizar un memo, lo primero será seleccionar el memo de la lista, por ello agregamos un evento de escucha a la lista de memos para cuando se seleccione uno vaya a la página del formulario de edición. Obtenemos la posición del elemento seleccionado en la lista, y enviamos a la otra página el objeto de memo con su información.

```
//EN EL _INITIALIZE DE MEMO
this.memosList.addListener("changeSelection",
this._controller.updateMemo, this._controller);
...
//EN EL CONTROLADOR DE LA VISTA MEMO
/**
 * evento para modificar un memo
 * @event updateMemo
 * @type changeSelection
 * @public
 *
 */
updateMemo : function(objEvent) {
    var view = this.getView();
    var pos = objEvent.getData();
    var listModel = view.memosList.getModel();
    var item = listModel.getItem(pos);
    this.assambler(memo.controller.ManageMemoController,
```

```
memo.view.ManageMemo, "index", [item]);
}
```

Como es editar el memo seleccionado tenemos que pasar la información del mismo. Como se utiliza el mismo punto de entrada para la otra presentación modificamos el método `index` del controlador de gestión. De esta manera si se va actualizar se pasa la información del memo, si se va a insertar se deja vacío. En el caso de la actualización el título de la página será el título del memo y el botón del formulario en vez de decir `Add`, dirá `Modified`.

```
//EN EL CONTROLADOR DE LA VISTA DE GESTION
/**
 * metodo inicial del controlador
 *
 * @method index
 * @public
 *
 */
index : function(data){
    var data = data || null;
    var dataView = {};
    dataView.title = "Add Memo";
    if(data != null){
        dataView.title = data.titulo;
    }
    dataView.item = data;
    this.setVar(dataView);
},
...
//EN _INITIALIZE EN LA VISTA DE GESTION
this.setTitle(this.vars.title);
```

Hicimos algo muy simple cambiar el título de la ventana cuando editemos un memo, ahora inicializaremos el formulario con la información del memo, para ello cuando inicializamos el formulario le pasamos un 3 parámetro que será el objeto del memo a editar.

```
//EN EL METODO COMPONENTS DE LA VISTA
this.formMemo = new
memo.view.forms.ManageMemo(this,null,this.vars.item);
```

Esto nos lleva a crear otro atributo en la clase de formulario cuyo valor por defecto será nulo. En el caso de la edición los componentes de formulario tomarán como valor inicial la información del memo.

```
//CREANDO EL ATRIBUTO
/**
 * propiedad para los datos del memo a editar
 *
 * @name memoData
 * @public
 * @type {Object}
 *
 */
memoData : {}
...
//INICIALIZANDO EL ATRIBUTO
this.memoData = memoData || null;
...
//EN LOS COMPONENTES DE FORMULARIO
input : function(){
    this.title = new criax.mobile.form.text.TextField();
    this.title.setLabel("Title:");
    this.title.setRequired(true);
    this.title.addValidation("presence","Title
empty",true);
    this.title.addValidation("alpha","Letters and
numbers",true);
    this.title.addValidation("max-length","Title too
large",true,[20]);

    this.content = new criax.mobile.form.text.TextArea();
    this.content.setLabel("Content:");
    this.content.setRequired(true);
    this.content.addValidation("presence","Content
empty",true);
    this.content.addValidation("max-length","Content too
large",true,[255]);

    if(this.memoData != null){
        this.title.setValue(this.memoData.titulo);
```

```
            this.content.setValue(this.memoData.contenido);
        }
    },
    ...
    //EN LA CONFIGURACION DEL FORMULARIO
    form : function(execution,contest){
        this.validateOnlyOnSubmit();
        this.addFilters(["first-up","clean"]);

        this.addField(this.title,"title");
        this.addField(this.content,"content");

        var buttonSave = "Add";
        if(this.memoData != null){
            var buttonSave = "Modified";
        }
        this.setSaveButton(buttonSave, null,
    criax.mobile.form.Button.BLACK, execution, contest);
        this.setResetButton("Cancel", null,
    criax.mobile.form.Button.BLACK);
    }
```


**Figura 7.1** "Actualizar la información de un memo"

Al momento de establecer el cambio, debemos confirmar que los datos nuevos del memo son correctos, esto lo haremos de manera simple con una ventanita

de confirmación. Como reutilizaremos el mismo método submit, la ejecución del callback del botón de aceptar, la modificaremos para poder pasarle a los datos del memo a actualizar.

```
//CONFIGURACION DEL BOTON DE ENVIO DEL FORMULARIO
this.formMemo.form(this.__callbackSubmit,this);
...
//METODO QUE SERVIRA PARA CALLBACK DE ENVIO
/**
 * metodo para el callback de envio
 *
 * @method __callbackSubmit
 * @private
 *
 */
__callbackSubmit : function(){
    this._controller.submit(this.vars.title);
}
```

Esto implica realizar un cambio en el método `submit` del controlador, pues si se pasa la información del memo, quiere decir que se va actualizar.

```
submit : function(memoData){
    var view = this.getView();
    var form = view.formMemo;
    if(form.validate()){
        var filtrateFormInfo = form.getFiltrateModel();
        var query = memo.model.Query.getInstance();
        if(memoData == undefined){
            var memoData = {};
            memoData.creado = query.formatDate();
            this.__saveMemo(memoData, filtrateFormInfo,
query);
        }else{
            if(window.confirm("Modified memo data")){
                this.__saveMemo(memoData,
filtrateFormInfo, query);
            }
        }
        this.back();
```

```
    }
}
```

Como se puede observar creamos un método nuevo `__saveMemo` para no tener que repetir código:

```
/**
 *  metodo para salvar la informacion de un memo
 *
 * @method __saveMemo
 * @private
 * @param memoData {Object} datos del memo
 * @param filtrateFormInfo {Object} modelo del formulario
con los datos filtrados
 * @param query {memo.model.Query} instancia del modelo
Query
 *
 */

__saveMemo : function(memoData,filtrateFormInfo,query){
    memoData.titulo = filtrateFormInfo.getTitle();
    memoData.contenido = filtrateFormInfo.getContent();
    memoData.modificado = query.formatDate();
    query.save(memoData);
},
```

Debido a que cuando se actualiza el memo este ya contiene un id, en el método `save` del modelo `Query`, debemos modificar este método para verificar el id del memo y en dependencia de ello realizar una acción u otra.

```
save : function(memoData){
    if(memoData.id == undefined){
        memoData.id = this.__autoIncrement();
        this.__trimQuery.insert("Memo",memoData);
    }else{
        this.__trimQuery.update("Memo",memoData,
{'id':memoData.id});
    }
},
```

Capítulo 8

# Eliminar un memo

Por último nos queda la opción de eliminar un memo, que será muy parecido a la anterior con la diferencia de que simplemente seleccionamos el memo que deseamos eliminar y lo eliminamos, siempre con una pregunta de confirmación.

La mitad ya la tenemos adelantada, nos queda poner un evento de escucha al `tap` del botón de la barra de herramientas con el cestico de basura, por otro lado crearemos un evento para poder pasar al controlador el dato del memo.

```
//EN EL METODO _INITIALIZE
this.buttonDelete.addListener("tap",this.__delete,this);
...
//EVENTO DE ESCUCHA PARA TAP
/**
 * evento para para eliminar el memo seleccionado
 * @event __delte
 * @type tap
 * @private
 *
 */
__delete : function(objEvent) {
    this._controller.deleteMemo(this.vars.item);
}
```

Ahora en el controlador ponemos la confirmación de la eliminación y pasamos el dato del memo al modelo.

```
/**
 *  metodo para eliminar un memo
 *
 * @method deleteMemo
 * @public
 * @param memoData {Object} datos del memo
 *
 */
deleteMemo : function(memoData){
    var query = memo.model.Query.getInstance();
    if(window.confirm("Delete memo")){
        query.remove(memoData);
    }
    this.back();
}
```

En el modelo nos queda desarrollar una funcionalidad para eliminar un memo en dependencia de su id.

```
/**
 *  metodo para eliminar un memo
 *
 * @method remove
 * @public
 * @param memoData {Object} memo a eliminar
 *
 */
remove : function(memoData){
    this.__trimQuery.deleteRow("Memo",{'id':memoData.id});
}
```

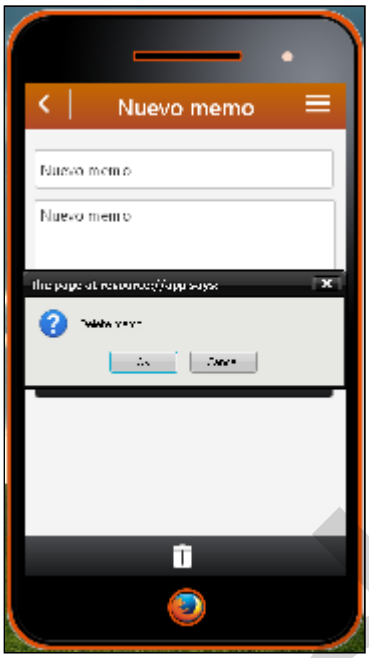


**Figura 8.1** "Eliminar memo"

## 8.1 Un detalle

Aunque parece este el final, nos queda un detalle muy simple, cuando vamos agregar un memo también se muestra el botón de eliminar de la barra de herramientas, ocultemos este en caso de que no se haya pasado ningún memo para gestionar. Para ello no agregamos la barra de herramientas a la página, al final del método `_initialize` de la vista `Memo`.

```
...
if(this.vars.item != null){
    this.add(this.toolBar);
}
```

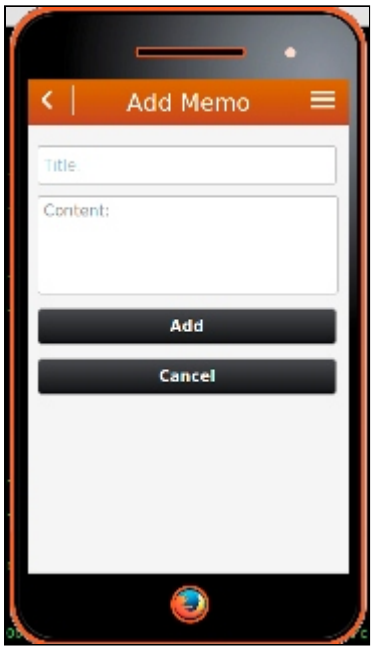


**Figura 8.2** "Insertar mejorado"

# III PARTE: El instalador

Capítulo 9

# Creando el distribuible

Ahora lo que queda es generar un distribuible de la aplicación.

## 9.1 La configuración

Una vez terminada la aplicación queda la configuración de la misma con los datos del autor, icono que la identificará, entre otras cosas. Para ello lo primero es abrir el archivo `memo/source/manifest.webapp`, allí configuraremos las características de la webapp, como el nombre, la versión, iconos, descripción. Por defecto CRIAX, inserta la información más básica.

```
{
    "name": "memo",
    "version": "0.1",
    "description": "First memo application by CRIAX-SDK",
    "launch_path": "/index.html",
    "icons": {
        "16": "/resource/icons/16.png",
        "22": "/resource/icons/22.png",
        "32": "/resource/icons/32.png",
        "48": "/resource/icons/48.png",
        "64": "/resource/icons/64.png",
        "128": "/resource/icons/128.png"
    },
    "developer": {
        "name": "Nilmar Sanchez Muguercia",
        "url": "http://developer.firefoxmania.uci.cu"
```

```
    },
    "installs_allowed_from": ["*"],
    "locales": {
        "es": {
        "description": "Primera aplicacion memo hecha con
CRIAX-SDK",
        "developer": {
            "url": "http://developer.firefoxmania.uci.cu"
        }
    }
  },
  "default_locale": "en"
}
```

## 9.2 Preparando el entorno

Una vez configurado el manifest, para generar el distribuible, tenemos que decirle a la configuración del entorno de la aplicación que estará en producción, para ello vamos al archivo `memo/config/enviroment.json` y en la sección de `enviroment` buscamos la clave `criax.production`, y le ponemos `true`, a los debugs, `false`, `false` y `0`.

```
{
    "export" : ["build"],
    "jobs" :
    {
        "build" :
        {
            "environment" :
            {
                "qx.debug" : false,
                "qx.debug.dispose" : false,
                "qx.disposerDebugLevel" : "0",

                "criax.production" : true,

                //SERVICES
                "criax.services.dir" : "services",
                "criax.services.default" : "services",

                //RESOLUTION
```

```
                    "criax.width" : 356,
                    "criax.height" : 640,

                    //LANGUAGE
                    "criax.i18n" : "es"
                }
            }
        }
    }
```

## 9.3 Generando el distribuible

Ahora la aplicación si esta lista para generar el distribuible, para ello ejecutamos el comando standalone:

```
$ cd memo
$ toolchain/standalone(.bat o .sh)
```

Esta script, creará un distribuible de la aplicación listo para subir al market y además mostrará la aplicación corriendo en B2G (versión de escritorio). Navegamos por las páginas del simulador y nos encontraremos nuestra aplicación:

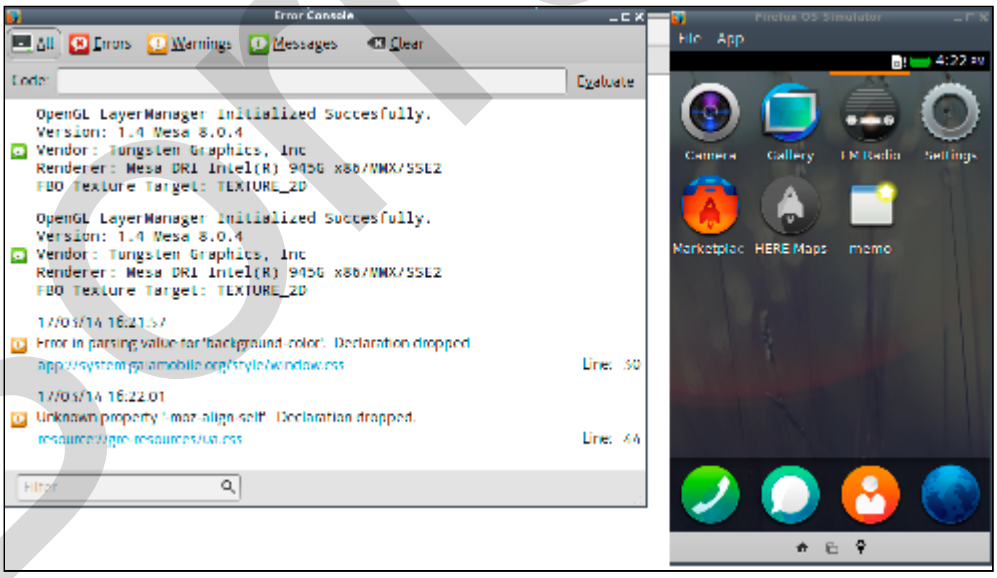


**Figura 9.1** "Aplicación en el simulador 1"

Si tocamos el icono de la misma, se abrirá y podremos comenzar a interactuar con la ella.

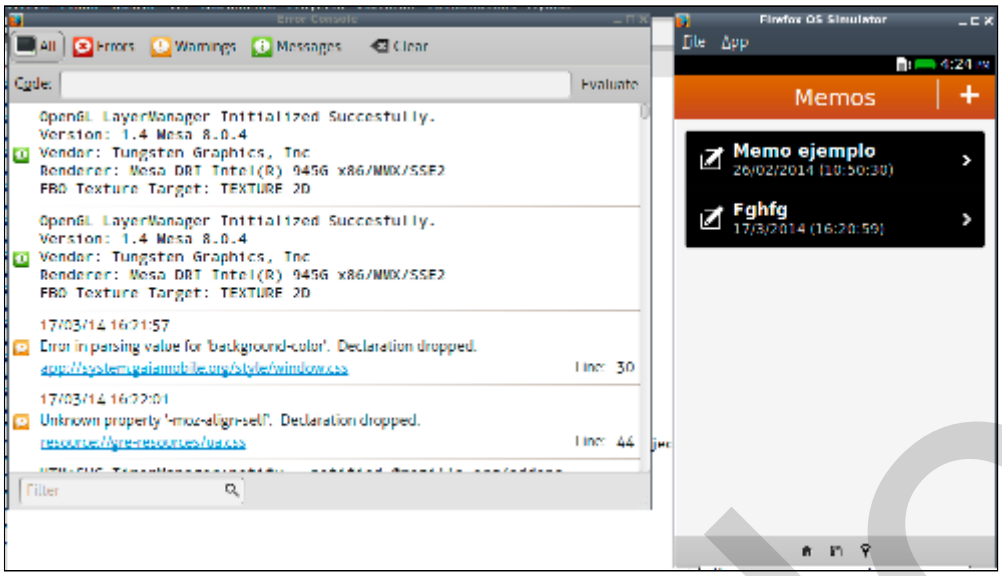


**Figura 9.2** "Aplicación en el simulador 2"

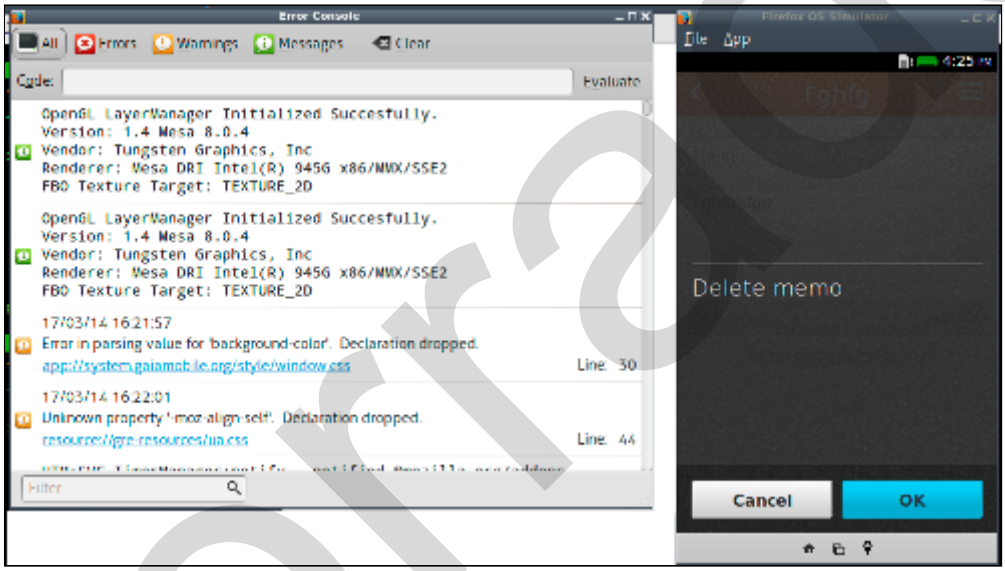


**Figura 9.3** "Aplicación en el simulador 3"

Capítulo 10

# El simulador

Si queremos utilizar el simulador del navegador, interactuamos con el directorio `build` que se encuentra en el directorio raíz de la aplicación. Primeramente instalamos los complementos `adbhelper-S.O-latest.xpi` y `fxos_1_2_simulator-S.O-latest.xpi`. Una vez instalado en el navegador vamos a `herramientas->desarrollador web->administrador de aplicaciones`.

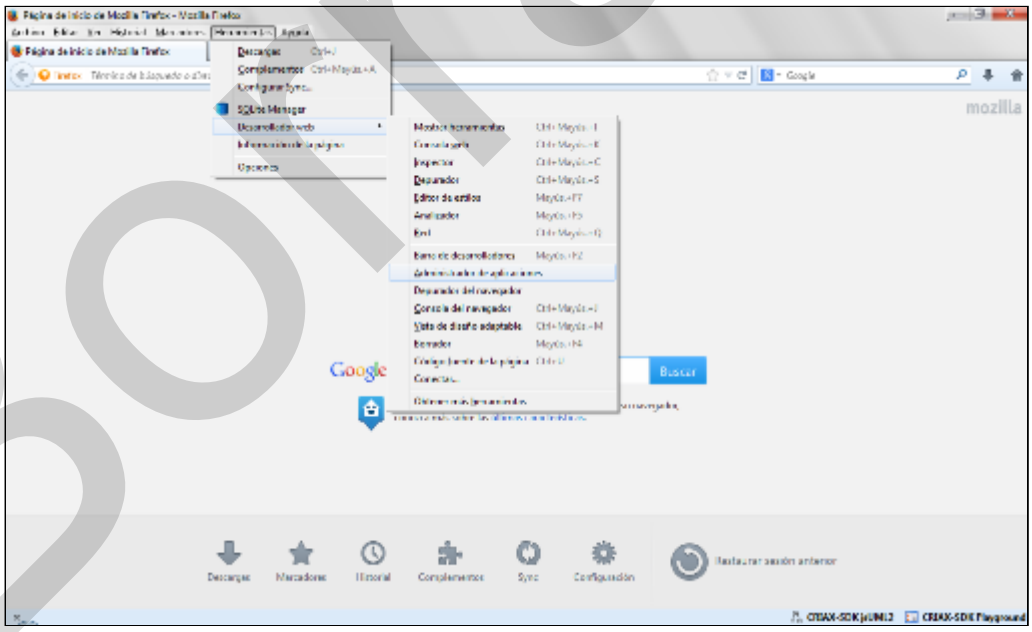


**Figura 10.1** "Como iniciar el administrador de aplicaciones"

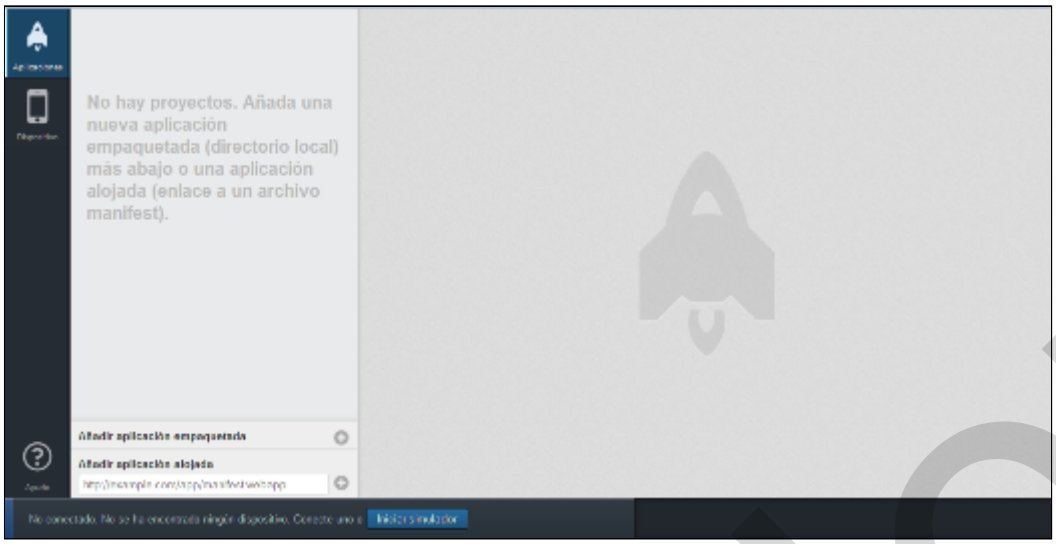


**Figura 10.2** "Administrador de aplicaciones 1"

Una vez abierto tenemos que agregar la aplicación al mismo, para ello vamos a la opción de **Añadir aplicación empaquetada** y buscamos y seleccionamos la carpeta `build`, del directorio raíz de nuestra aplicación, de esta manera la aplicación queda lista para correr en el simulador.

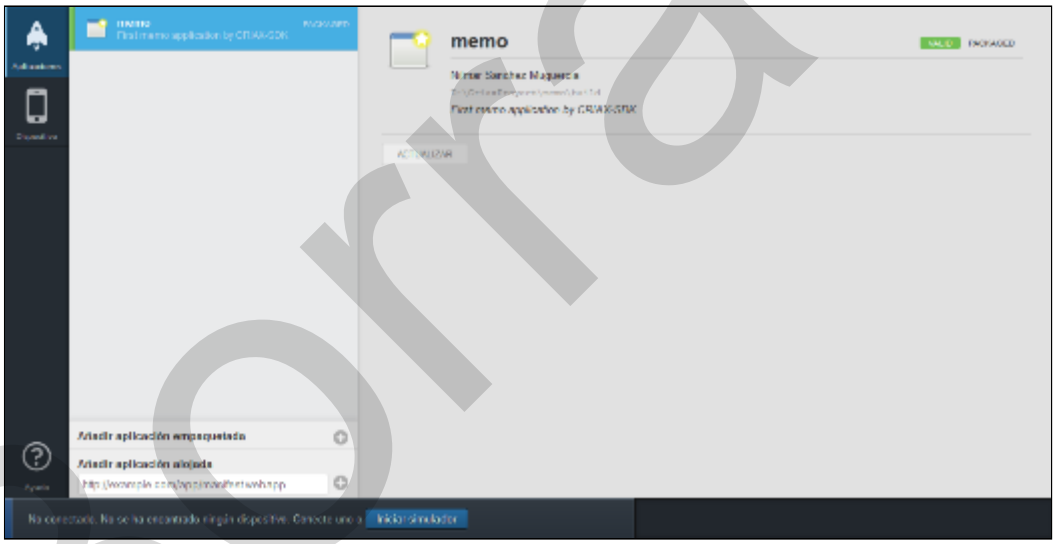


**Figura 10.3** "Administrador de aplicaciones 2"

Nos queda simplemente echar andar el simulador, damos click en el botón azul de abajo `Iniciar simulador` y luego click en el botón `Firefox OS 1.2` o `Firefox OS 1.3` en dependencia de la versión que estemos usando.

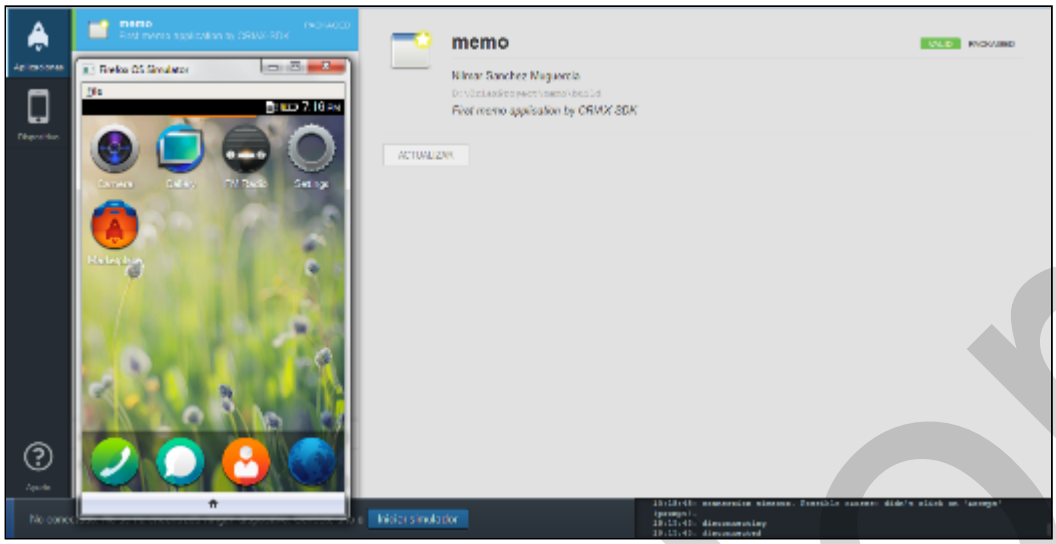


**Figura 10.4** "Simulador Firefox OS 1.2"

Para agregar nuestra aplicación al simulador, damos click en el botón de actualizar del panel derecho y automáticamente la aplicación se agregará al simulador, ahí podremos interactuar con ella.

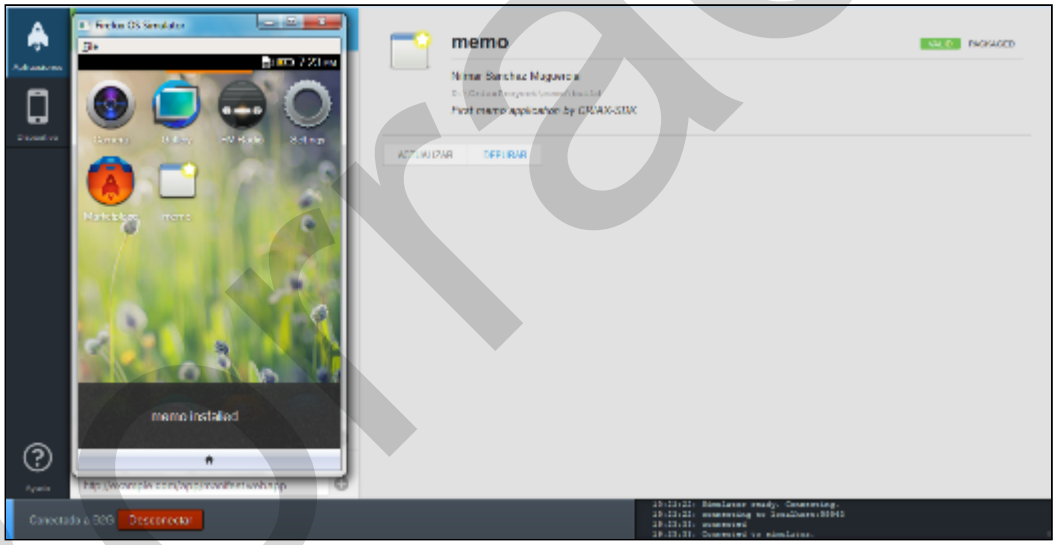


**Figura 10.5** "Aplicación en el simulador 1"

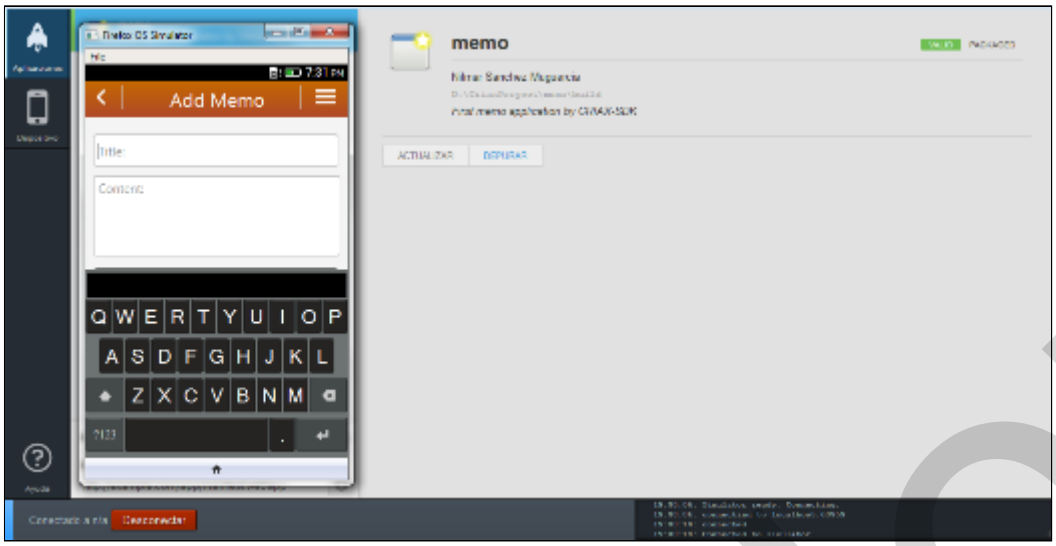


**Figura 10.6** "Aplicación en el simulador 2"

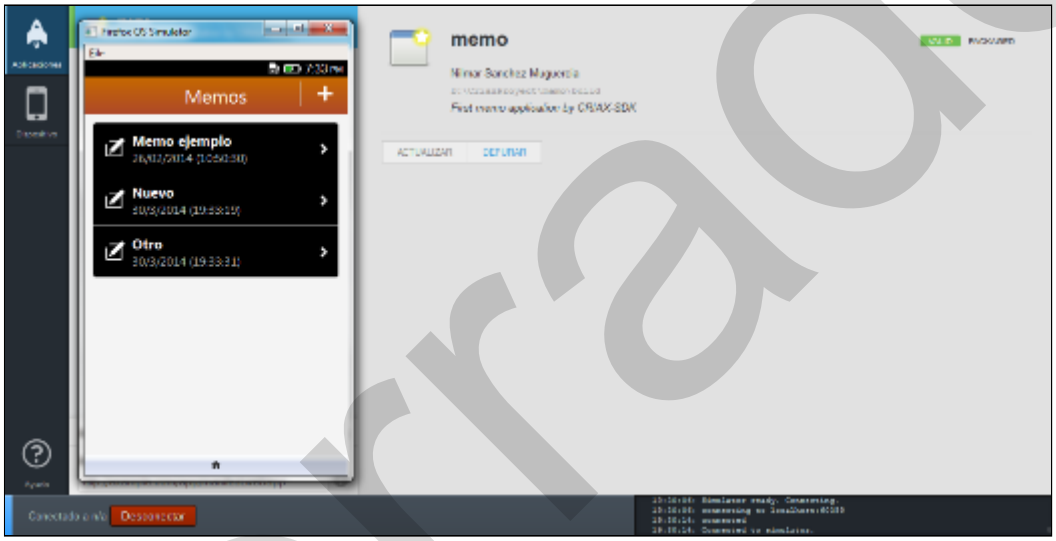


**Figura 10.7** "Aplicación en el simulador 3"

Capítulo 11

# El teléfono

**NOTA:** primero que todo cabe destacar que este tutorial, está siendo ejecutado en un terminal Geeksphone Keon, con actualización Firefox OS 1.5


**Figura 11.1** "Captura de las características del dispositivo"

## 11.1 La depuración remota

Primero que todo debemos asegurarnos que la depuración remota está activada. Para ello en el teléfono vamos a: `Settings > Device information > More information > Developer` y marcamos la opción de `Remote Debugging`.

## 11.2 Drivers en window

**NOTA:** para los que trabajen en Linux la cosa es más sencilla, para los que trabajen en Windows, aquí les va la ayuda (solo este epígrafe).

Lo primero por supuesto es tener el dispositivo. Una vez con él, engañaremos al sistema operativo Windows 7 o inferior para que piense que el teléfono que se conectó es un teléfono con Android. Para ello tenemos que utilizar los drivers que se encontrarán junto con el código de la aplicación. Descompactamos `fos_usb_driver_window.zip` y dentro encontraremos los archivos necesarios. Conectamos el teléfono a la PC y esta por supuesto no lo debe reconocer. Vamos al administrador de dispositivos y nos debe salir un nuevo dispositivo desconocido con un signo de interrogación en amarillo.

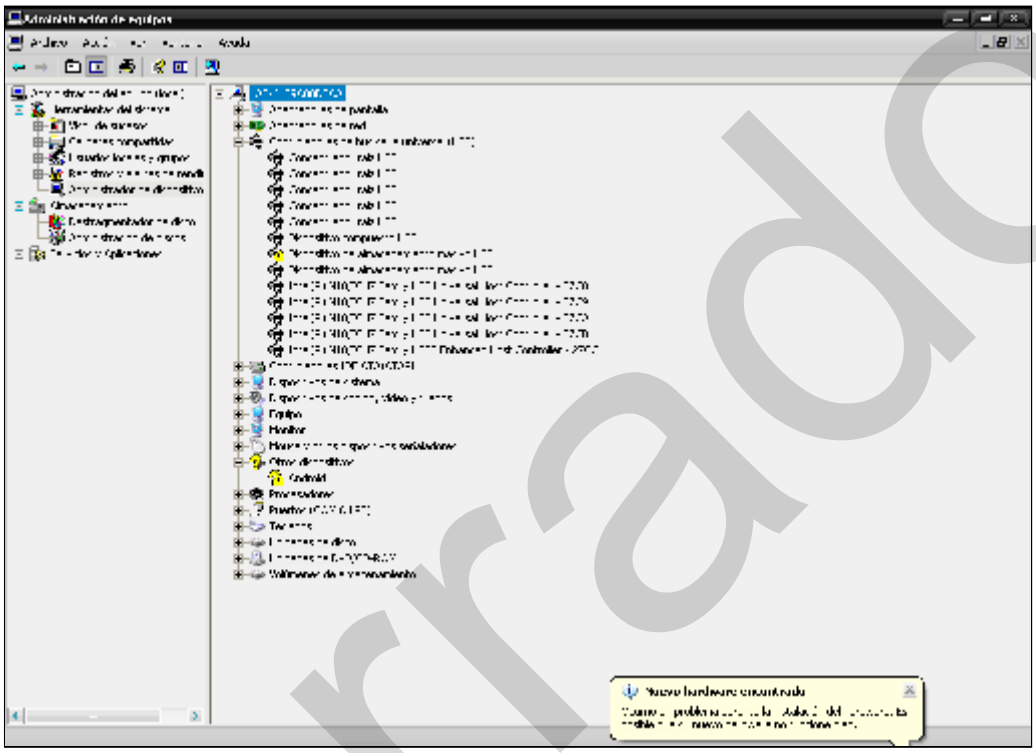

**Figura 11.2** "Dispositivo desconocido 1"

Damos click derecho encima y seleccionamos `Actualizar controlador`. En el asistente seleccionamos `Si solo esta vez` y en la siguiente ventana `Instalar desde una lista o ubicacion especifica`.

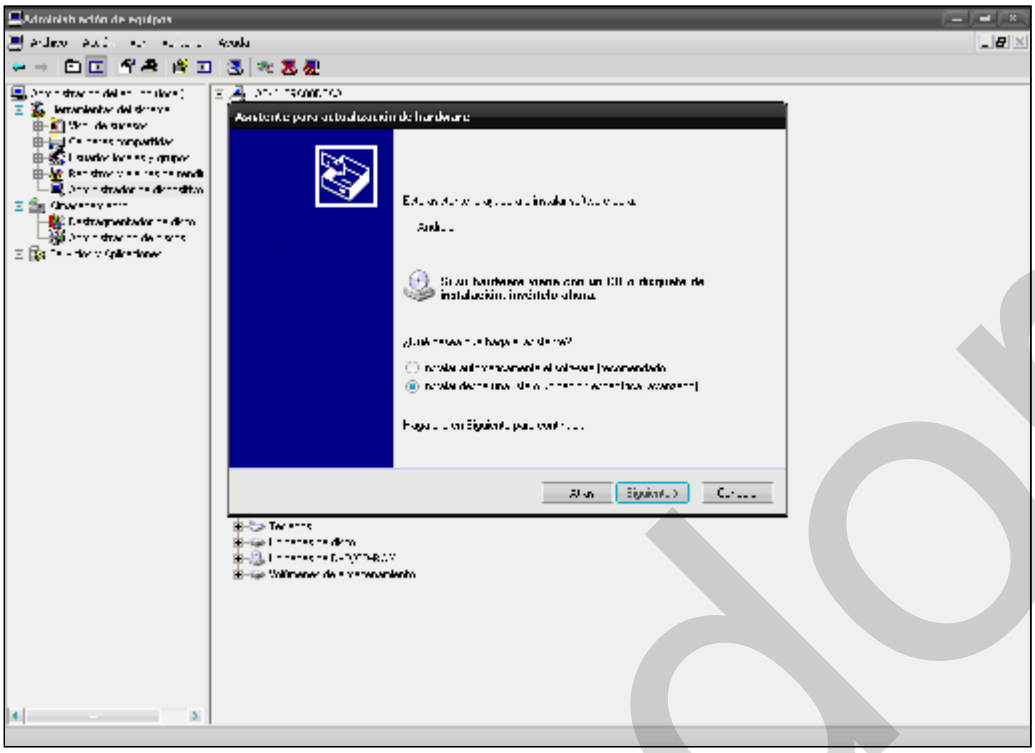

**Figura 11.3** "Dispositivo desconocido 2"

Examinamos y buscamos el contenido descompactado de `fos_usb_driver_window.zip` y damos siguiente. El asistente instalará los drivers, no se asusten si al final da un error. Finalizamos el asistente y ahora la PC pensara que tiene conectado un dispositivo con Android.

## 11.3 El teléfono en el simulador

Ahora podemos abrir el simulador, que contendrá ya nuestra aplicación y debajo podremos observar que al lado del botón `Iniciar simulador`, aparece un botón para conectar con el teléfono. Damos click en él y el simulador se conectará al teléfono. Como podrán observar el teléfono pedirá que aceptemos la conexión entrante.

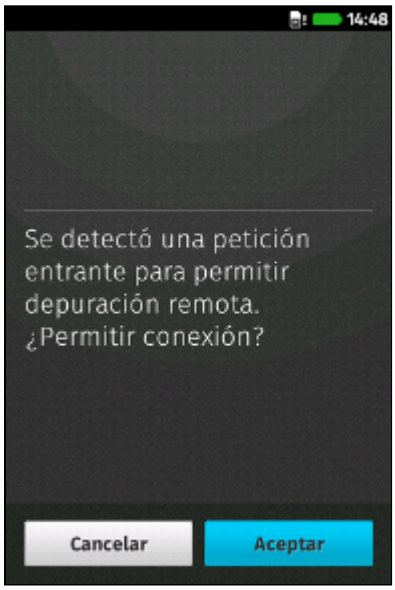


**Figura 11.4** "Captura desde el dispositivo: aceptar conexión entrante"

Ahora ya estamos conectados al dispositivo

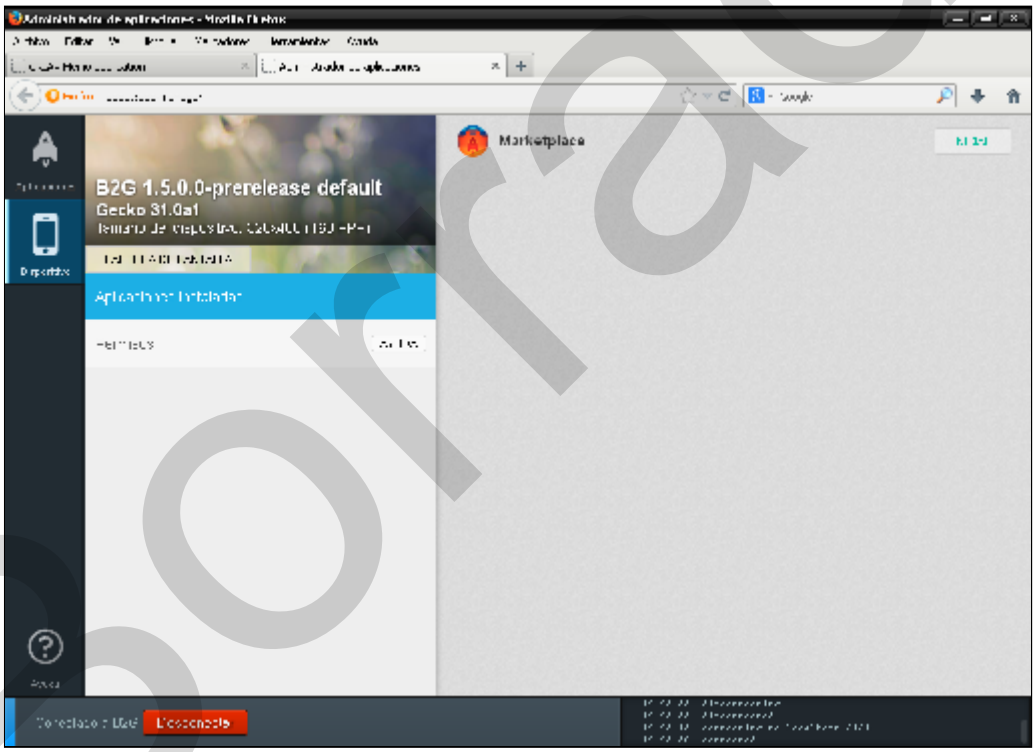


**Figura 11.5** "Dispositivo conectado"

Simplemente nos queda subir la aplicación al teléfono. Para ello señalamos la aplicación y le damos `Actualizar`. Inmediatamente en el teléfono indicará que la aplicación se instaló correctamente y la mostrará.

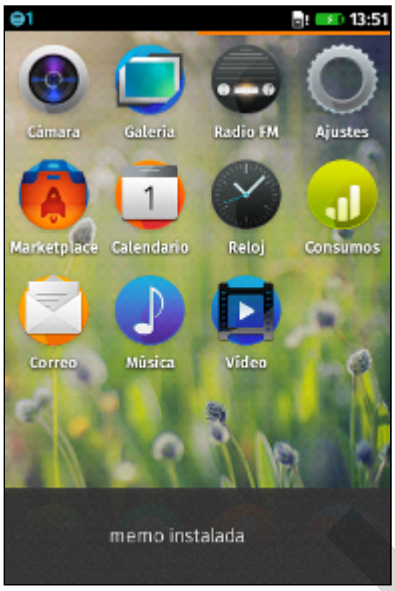


**Figura 11.6** "Captura desde el dispositivo: aplicación instalada 1"

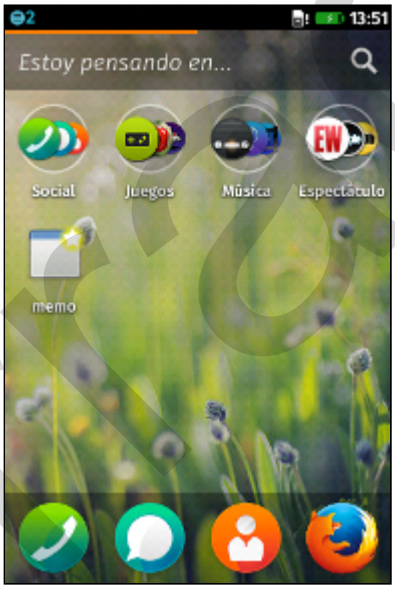


**Figura 11.7** "Captura desde el dispositivo: aplicación instalada 2"

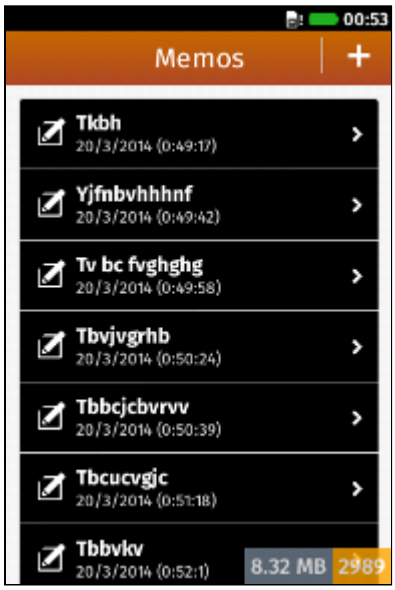


**Figura 11.8** "Captura desde el dispositivo: listado de memos"

# Glosario

- **AppJs:**Proyecto para desarrollar aplicaciones de escritorio con tecnologias web.
- **Adobe Air:**Plataforma de Adobe para el desarrollo de aplicaciones RIA.
- **Chromeless:** proyecto de Mozilla para que el usuario pudiera desarrollar su propio navegador web con lenguages de HTML5.
- **CSS:** Hojas de Estilo en Cascada, lenguage para maquetar en la web.
- **Firefoxos:**Sistema Operatico de Mozilla para dispositivos moviles.
- **HTML:**Lenguage Marcado de Hipertexto, lenguage de etiquetas para desarrollar las paginas web.
- **HMVC:**Metodologia Modelo Vista Controlador Jerarquico.
- **JSON:** Notacion de objetos de Javascript, formato ligero para el intercambio de datos.
- **Javascript:** Lenguage de programacion que se utiliza para crear paginas dinamicas del lador del cliente.
- **Qooxdoo:** Framework de Javascript para el desarrollo de paginas web.
- **Qt:**Libreria para desarrollar interfaces de usuario en C++.
- **RIA:**Aplicaciones Enriquecidas de Internet.
- **Swing:**Libreria de Java para el desarrollo de interfaces de usuario.
- **SQLite:** Liberia programada en C que funciona como un sistema de base de datos relacionales.
- **Twitter:**Plataforma de microbloggin.
- **TideSDK:**Plataforma para el desarrollo de aplicaciones de escritorio con tecnologias web.
- **Webapp:**Aplicaciones web instalables.