UNIVERSIDAD
REY JUAN CARLOS

INGENIERÍA EN TELECOMUNICACIONES Y
LICENCIATURA EN ADMINISTRACIÓN Y DIRECCIÓN DE
EMPRESAS

Curso Académico 2016/2017

Trabajo Fin de Carrera

MY APP ANALYZER

Autor : Alexandra Ortega Martín

Tutor : Dr. Gregorio Robles

Proyecto Fin de Carrera

My App Analyzer

Autor : Alexandra Ortega Martín

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2017, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 20XX

Dedicado a

mi familia / mi abuelo / mi abuela

II

Agradecimientos

Aquí vienen los agradecimientos... Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.

Resumen

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este capítulo se introduce el proyeto. Debería tener información general sobre el mismo, dando la información sobre el contexto en el que se ha desarrollado.

No te olvides de echarle un ojo a la página con los cinco errores de escritura más frecuentes[1].

1.1.

1.1.1. Estilo

Sobre el uso de las comas[2]

```
From gaurav at gold-solutions.co.uk  Fri Jan 14 14:51:11 2005
From: gaurav at gold-solutions.co.uk (gaurav_gold)
Date: Fri Jan 14 19:25:51 2005
Subject: [Mailman-Users] mailman issues
Message-ID: <003c01c4fa40$1d99b4c0$94592252@gaurav7klgnyif>

Dear Sir/Madam,
How can people reply to the mailing list?  How do i turn off
this feature? How can i also enable a feature where if someone
replies the newsletter the email gets deleted?
Thanks
```

[1] http://www.tallerdeescritores.com/errores-de-escritura-frecuentes

[2] http://narrativabreve.com/2015/02/opiniones-de-un-corrector-de-estilo-11-recetas-par

html

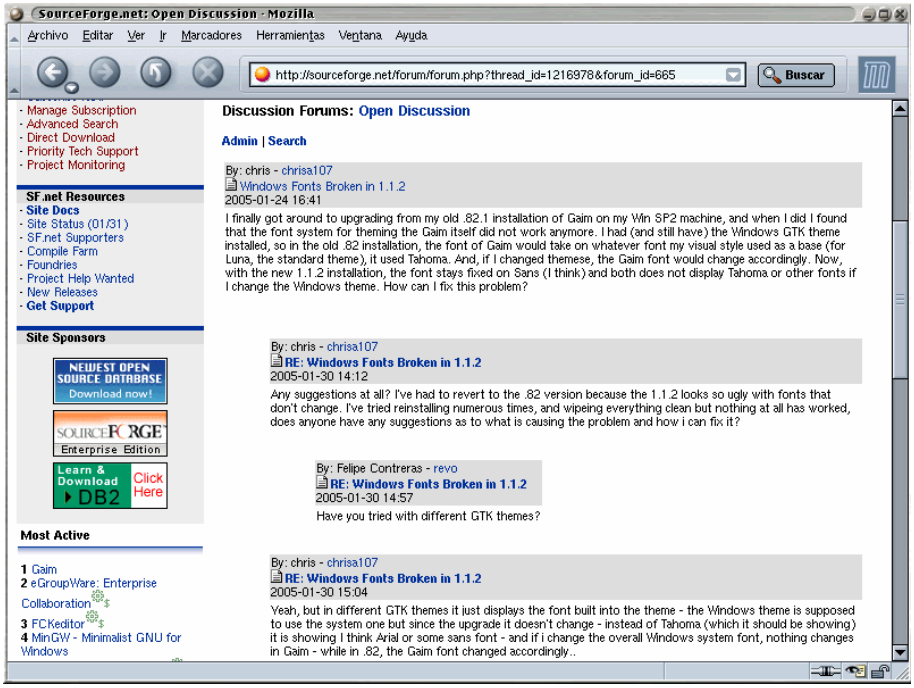

Figura 1.1: Página con enlaces a hilos

```
From msapiro at value.net  Fri Jan 14 19:48:51 2005
From: msapiro at value.net (Mark Sapiro)
Date: Fri Jan 14 19:49:04 2005
Subject: [Mailman-Users] mailman issues
In-Reply-To: <003c01c4fa40$1d99b4c0$94592252@gaurav7klgnyif>
Message-ID: <PC17302005011410485105780lb04d55@msapiro>

gaurav_gold wrote:
>How can people reply to the mailing list?  How do i turn off
this feature? How can i also enable a feature where if someone
replies the newsletter the email gets deleted?

See the FAQ
>Mailman FAQ: http://www.python.org/cgi-bin/faqw-mm.py
article 3.11
```

1.2. Estructura de la memoria

En esta sección se debería introducir la esctura de la memoria. Así:

- En el primer capítulo se hace una intro al proyecto.
- En el capítulo 2 se muestran los objetivos del proyecto.
- A continuación se presenta el estado del arte.
- ...

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo general de este proyecto es crear una plataforma donde los nuevos programadores puedan evaluar su código de forma que no sólo puedan conocer sus puntos fuertes y débiles sino que además puedan obtener nuevos retos para mejorar sus habilidades computacionales.

Niños y jóvenes con interés por la programación conforman el público objetivo al que va destinada la aplicación. Por ello se han adaptado el diseño, funcionalidad y lenguaje para que resulten la experiencia de usuario sea lo más atractiva y lúdica posible.

En el sistema de clasificación hemos huido de las puntuaciones númericas, convirtiéndolas en tres niveles representados por los colores del semáforo: alto (verde), medio (amarillo) y bajo (rojo). De esta manera convertimos la experiencia en un juego, enmarcándola en un contexto más positivo para el usuario y alejándonos de las puntuaciones númericas utilizadas de los exámenes.

Toda la aplicación se relaciona con el usuario a través de un lenguaje coloquial, sencillo y en inglés. Se eligió este idioma porque es importante familizarizar a los niños con el mismo y reforzar sus conocimientos a través del juego.

2.2. Objetivos específicos

- Analizar las posibilidades de personalización y bloques computacionales que ofrece App Inventor como herramienta para crear programas.

- Definir las habilidades y capacidades del usuario a analizar. De los múltiples enfoques considerados y de cara a una simplicación de la información final se optó por crear tres grandes bloques de análisis: componentes, programación y usabilidad.

- Crear una aplicación Django donde se recoja la lógica anterior y poder ofrecer al usuario la información de manera clara y resumida. El usuario podrá además tener un registro de los proyectos guardados con anterioridad para poder revisar su clasificación.

- Unir las tecnologías Django, Bootstrap y Ajax para mejorar la capa de *Front End* y hacer la experiencia de usuario mucho más dinámica.

2.3. Planificación temporal

La planificación temporal del proyecto ha sido definida en función de los objetivos específicos marcados siguiendo un orden que permitiera avanzar en todos los aspectos de la aplicación:

- **Fase I**: búsqueda de documentación sobre App Inventor y creación de una cuenta en su web [1]donde poder analizar los bloques disponibles para el usuario, el contenido del archivo comprimido (*.aia*) en que se guardan los programas y cómo se representa la información dentro del mismo.

- **Fase II**:

  - Creación de una aplicación Django con una web simple donde subir el fichero con el proyecto, descomprimirlo y guardar su información en base de datos.
  - Añadir la posibilidad de tener una cuenta de usuario en la aplicación donde se almacenen los diferentes proyectos subidos de cara a futuras consultas.

- **Fase III**: definición de los puntos a analizar en cada proyecto subido e implementar la lógica de evaluación y clasificación.

- **Fase IV**: incorporar a la web las tecnologías Bootstrap y Ajax para añadir dinamismo y mejorar su aspecto.

[1] `http://appinventor.mit.edu/explore/`

Capítulo 3

Estado del arte

En este capítulo se explicarán brevemente las principales tecnologías utilizadas en el proyecto.

Puedes citar libros, como el de Bonabeau et al. sobre procesos estigmérgicos [1].

También existe la posibilidad de poner notas al pie de página, por ejemplo, una para indicarte que visite la página de LibreSoft[1].

3.1. App Inventor

(Contenido)

3.2. Python

(Contenido)

3.3. Django

(Contenido)

[1] `http://www.libresoft.es`

3.4. SQLite

(Contenido)

3.5. HHTP

(Contenido)

3.6. CSS

(Contenido)

3.7. HTML5

(Contenido)

3.8. Bootstrap

(Contenido)

3.9. Ajax

(Contenido)

3.10. Javascript

(Contenido)

Capítulo 4

Diseño e implementación

En este capítulo se detallarán la arquitectura y el diseño implementados en My App Inventor, relacionando las diferentes tecnologías utilizadas y explicando el flujo interno del programa.

4.1. Arquitectura general

La arquitectura de la aplicación se basa en el modelo cliente-servidor descrito en la figura 4.1. Una vez que el usuario ha entrado en su cuenta dentro de la web, tendrá dos opciones: volver a analizar un proyecto existente o subir el fichero comprimido *.aia* que previamente ha descargado de la web de App Inventor[1] y analizarlo. Ambas peticiones serán recogidas por el manejador Django que internamente realizará las acciones necesarias para obtener una evaluación del programa. Los resultados finales serán renderizados por BootStrap y mostrados al usuario en su navegador web.

Como se puede observar en la siguiente figura tenemos dos componentes principales:

- Servidor: compuesto por una aplicación Django y una base de datos SQLite. Mientras que Django realizará la interacción con la capa de Front End y realizará el análisis de los datos, la base de datos nos ayudará a almacenar la información relativa a los usuarios y sus proyectos.

- Cliente: el usuario accede a la aplicación mediante un navegador web desde el que se realizan las peticiones GET o POST (Http Request) y mostrará los datos recibidos desde el

[1] `http://appinventor.mit.edu/explore/`

Servidor (HTTP Response). El desarrollo de esta parte del Front End se realiza a conjuntamente con Django y Bootstrap de manera que la web se adapta al dispositivo utilizado por el usuario (ordenador o móvil)

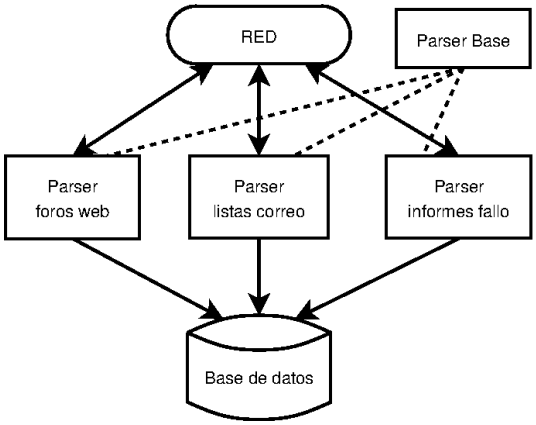

Figura 4.1: Arquitectura Cliente-Servidor

4.2. Diseño e implementación del servidor

Como hemos visto, la arquitectura del Servidor consta de dos grandes bloques: la lógica de Django y el almacenamiento de SQLite.

4.2.1. Lógica servidor: Django

La lógica del proyecto My App Inventor está organizada en los siguientes directorios:

- Projecto Django - **analyzeMyApp**: dentro nos encontraremos con los ficheros de configuración. En *settings.py* indicaremos los parámetros principales del programa (directorios, idioma, base de datos ...), mientras que en *wsgi.py* especificaremos las reglas de comunicación entre un servidor web y nuestra aplicación cuando despleguemos en producción.

- Aplicación - **myAnalyzer**: en este directorio guardaremos la lógica de la aplicación, es decir, qué hacer cuando recibimos una petición, cómo será nuestro modelo de datos, qué pasos seguir al analizar un fichero recibido.... En las siguientes secciones veremos más en detalle este apartado.

- Ficheros estáticos - **static**: donde están las imágenes, estilos css y pequeños Javascript para utilizar con Bootstrap.
- Plantillas - **templates**: incluye todas las plantillas HTML que formarán parte de la interacción del usuario con la aplicación.
- Proyectos guardados - **saved_projects**: además de guardar la información más relevante en base de datos, guardaremos una copia de seguridad de cada proyecto subido por el usuario en disco, así en caso de posibles ampliaciones de la lógica de evaluación, se podrá actualizar el resultado de forma transparente al usuario sin necesidad de que vuelva a cargar su programa de nuevo.

```
/
├── manage.py
├── analyzeMyApp
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── myAnalyzer
│   ├── __init__.py
│   ├── admin.py
│   ├── scoreMyAppMessages.py
│   ├── admin.py
│   ├── forms.py
│   ├── models.py
│   ├── urls.py
│   ├── apps.py
│   ├── scoreMyApp.py
│   ├── parser.py
│   ├── views.py
│   ├── errors.py
│   └── tests.py
├── static
├── templates
└── saved_projects
```

Figura 4.2: Directorio de archivos My App Inventor

En líneas generales, el flujo de la aplicación sigue los pasos de la Figura 4.3 y se define dentro del directorio **myAnalyzer**. Cuando un usuario registrado realiza una petición a la aplicación, se diferencia entre guardar un nuevo proyecto o cargar uno preexistente. Tras este paso,

se procede al anáĺisis del programa y como paso final, se devuelve una respuesta con la clasificación obtenida. Durante todo el proceso, si se recibe una petición de un usuario no registrado, se redirige al mismo a la página de inicio.

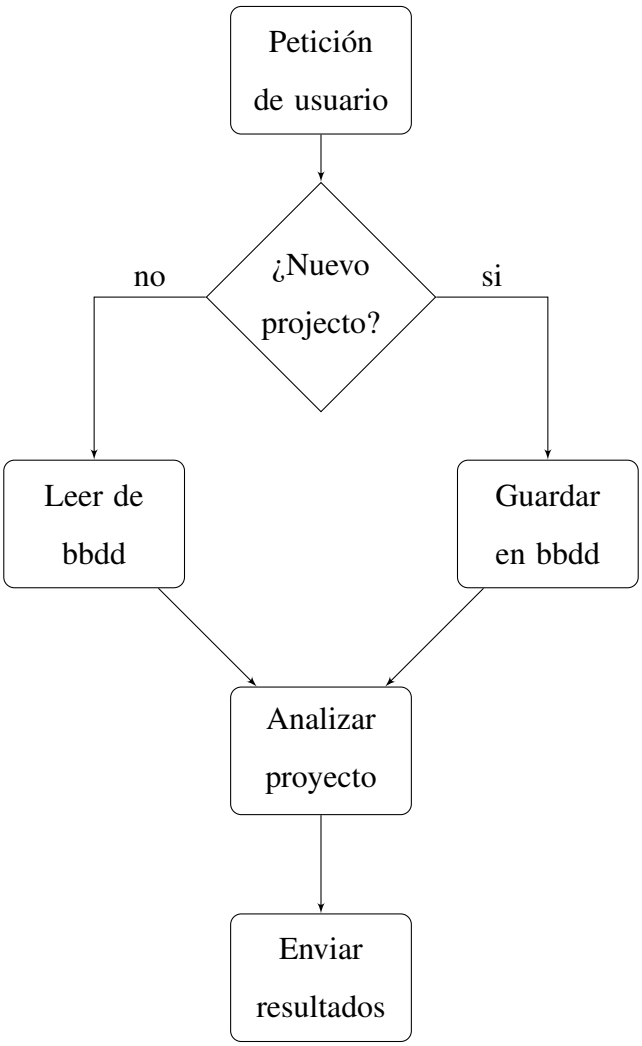

Figura 4.3: Flujo principal My App Inventor

En el momento en que la aplicación recibe una petición HTTP del usuario con una url concreta, ésta se mapea en el fichero *urls.py*. Éste a su vez redige la información a un procedimiento concreto de *wsgi.py* en función de la operación a realizar: iniciar o cerrar sesión, actualizar datos de perfil, cargar un nuevo programa, ver los anteriormente guardados . . . No todas las urls son accesibles al usuario mediante la petición GET; por ejemplo, sólo podremos acceder a la vista que analiza los proyectos del usuario,*views.showUserAnalyzeProjectsPage*, mediante una petición de tipo POST insertada en un formulario de la aplicación para mante-

ner la integridad de los datos recibidos y controlar su contenido. Lo mismo ocurre con la vista *views.showDownloadPage* en la que se accede a la petición POST a través de un formulario.

```
urlpatterns = [
                url(r'^login/', views.showLoginPage),
                url(r'^logout/', views.showLogoutPage),
                url(r'^createprofile/', views.showCreateProfilePage),
                url(r'^userprofile/', views.showUserProfilePage),
                url(r'^download/', views.showDownloadPage),
                url(r'^updateprofile/', views.showUpdateProfilePage),
                url(r'^userprojects/', views.showUserProjectsPage),
                url(r'^analyze/', views.showUserAnalyzeProjectsPage),
                url(r'^$', views.showLoginPage),
                ]
```

Veamos en más detalle los diferentes procedimientos implicados en el proceso global que forman parte de la aplicación.

Comunicación

Como hemos comentado anteriormente, si un usuario no inicia sesión en la aplicación, se le devolverá siempre a la página inicial. Entre todas las librerías (o *views*) que incorpora Django, utilizaremos el Sistema de Autenticación [2] para complementar las vistas que nos ayudarán a realizar las operaciones más comunes como iniciar o cerrar sesión (*views.showLoginPage, views.showLogoutPage*), mantener la misma entre peticiones al servidor o comprobar si un usuario está conectado.

Todas las peticiones se realizarán a través de los objetos HttpResponse [3] de Django que manejan solicitudes GET y POST según corresponda. Por ejemplo, en la vista de inicio de sesión, se diferencia entre la solicitud de *loggin* (GET), donde respondemos al usuario con un formulario en el que introducir sus datos de registro, y la recepción de los mismos en el método POST para autenticarle.

[2] https://docs.djangoproject.com/en/1.11/topics/auth/default/
[3] https://docs.djangoproject.com/en/1.11/ref/request-response/

Tras el inicio de sesión, el usuario será redirigido a la página principal donde podrá evaluar su código en función de si ya está subido a la aplicación o no, como vimos en la Figura 4.3.

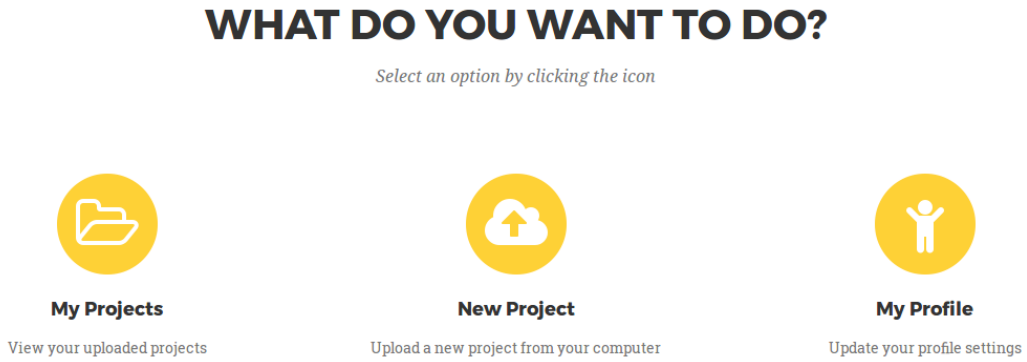

Figura 4.4: Opciones disponibles

Primero veamos el caso donde un programador quiere analizar su código por primera vez. La función *views.showDownloadPage* devuelve en la primera petición (GET) un formulario en el que el usuario podrá buscar en su disco el fichero comprimido *.aia* que previamente se ha descargado de su cuenta en App Inventor. Tras pulsar el botón Enviar, el navegador enviará una petición POST con el fichero al Servidor, donde se comprobará que no existe previamente para este usuario y decomprimirá, para posteriormente almacenar la información más relevante en base de datos.

La estructura en la que App Inventor guarda la información sigue el esquema descrito en la figura 4.5. En la carpeta *assets* se almacenan los ficheros estáticos de la aplicación, como las imágenes y sonidos. En *project.properties* se especifica la configuración del proyecto: versión del código, primera pantalla, tamaño.... Y finalmente en el último nivel del directorio *src* se encuentran la información relativa a los bloques utilizados (*.scm) y las lógicas que los relacionan (*.bky) por pantallas. Los datos más importantes para My App Inventor se encuentran en este último nivel y por ello serán los que guardemos en base de datos para su posterior análisis.

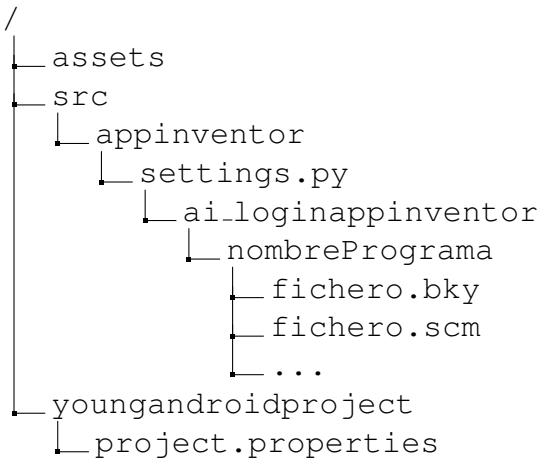

Figura 4.5: Directorio App Inventor

Si en un primer caso partíamos de un nuevo fichero de código, nuestra aplicación también permite al usuario guardar los proyectos guardados para poder volver a analizarlos en el futuro. Al llamar a la función *views.showUserProjectsPage* el Servidor listará todos los programas almacenados para el usuario y éste podrá volver a obtener su clasificación.

Tanto si partimos de un código nuevo como de uno ya existente, la última operación que hace My App Inventor es analizar el código. ¿Y por qué analizar algo que ya está puntuado? Pensando en futuras mejoras e implementaciones del algoritmo de análisis, de esta forma hacemos posible que el usuario siempre tenga su disposición la última versión del mismo con la clasificación más actualizada. Así no es necesario preprocesar todos los programas en base de datos en caso de actualización ya que ésta se realiza bajo demanda.

Análisis de código

El análisis de los programas del usuario se realiza a tres niveles: componentes, programación y usabilidad, cada uno con sus correspondientes subniveles. Como habíamos comentado en la sección anterior, en base de datos tendremos la información principal de las pantallas que forman cada proyecto. Para cada pantalla tenemos:

- Fichero **.scm**: contiene los bloques añadidos al proyecto y sus propiedades en formato Json (JavaScript Object Notation)[4], un formato de representación de objetos con una estructura sencilla basada en relaciones nombre-valor. En función del bloque utilizado ten-

[4] `http://www.json.org/`

dremos unas propiedades u otras: posición (*AlignHorizontal*), fondo (*BackgroundColor*), tamaño (*Width*) . . .


```
#|
$JSON
{"authURL":["ai2.appinventor.mit.edu"],"YaVersion":"159","Source":"Form","Properties":
{"$Name":"LogInScreen","$Type":"Form","$Version":"20","AlignHorizontal":"3","AlignVertical":"2","AppName":"Multifunctionalaplication"
[{"$Name":"HorizontalArrangement1","$Type":"HorizontalArrangement","$Version":"3","BackgroundColor":"&H00FFFFFF","Width":"-2","Uuid":
[{"$Name":"VerticalScrollArrangement1","$Type":"VerticalScrollArrangement","$Version":"1","AlignHorizontal":"3","BackgroundColor":"&H
[{"$Name":"LogInBox","$Type":"TextBox","$Version":"5","Width":"-2","Hint":"Username, e-mail or phone","Uuid":"1160039573"},
{"$Name":"LogInPasswordtBox","$Type":"PasswordTextBox","$Version":"3","Width":"-2","Hint":"Password","Uuid":"827897089"}]},
{"$Name":"VerticalArrangement1","$Type":"VerticalArrangement","$Version":"3","AlignVertical":"2","BackgroundColor":"&H00FFFFFF","Heigl
[{"$Name":"SignInBtn","$Type":"Button","$Version":"6","BackgroundColor":"&H00FFFFFF","Text":"Sign In","Uuid":"-756982208"}]}]},
{"$Name":"HorizontalArrangement2","$Type":"HorizontalArrangement","$Version":"3","BackgroundColor":"&H00FFFFFF","Width":"-2","Uuid":"
[{"$Name":"VerticalScrollArrangement2","$Type":"VerticalScrollArrangement","$Version":"1","BackgroundColor":"&H00FFFFFF","Width":"-2"
[{"$Name":"FullNameSU","$Type":"TextBox","$Version":"5","Width":"-2","Hint":"Full Name","Uuid":"1890435708"},
{"$Name":"EmailSU","$Type":"TextBox","$Version":"5","Width":"-2","Hint":"E-mail","Uuid":"-1325954197"},
{"$Name":"PhoneSU","$Type":"TextBox","$Version":"5","Width":"-2","Hint":"Phone number","NumbersOnly":"True","Uuid":"592322586"},
{"$Name":"UsernameSU","$Type":"TextBox","$Version":"5","Width":"-2","Hint":"Username","Uuid":"-604227488"},
{"$Name":"PasswordSU","$Type":"PasswordTextBox","$Version":"3","Width":"-2","Hint":"Password","Uuid":"-1827733330"},
{"$Name":"CPasswordSU","$Type":"PasswordTextBox","$Version":"3","Width":"-2","Hint":"Confirm password","Uuid":"-710498798"},
{"$Name":"AgeSU","$Type":"TextBox","$Version":"5","Width":"-2","Hint":"Age","NumbersOnly":"True","Uuid":"-595745446"},
{"$Name":"HorizontalArrangement4","$Type":"HorizontalArrangement","$Version":"3","AlignHorizontal":"3","AlignVertical":"2","Backgroun
[{"$Name":"HorizontalArrangement5","$Type":"HorizontalArrangement","$Version":"3","AlignHorizontal":"3","AlignVertical":"2","Backgrou
[{"$Name":"Label1","$Type":"Label","$Version":"4","Text":"Gender:","Uuid":"1447773381"}]},
{"$Name":"GenderSpinner","$Type":"Spinner","$Version":"1","ElementsFromString":"Male, Female","Uuid":"-592189594"}]}]},
{"$Name":"VerticalArrangement2","$Type":"VerticalArrangement","$Version":"3","AlignHorizontal":"3","AlignVertical":"2","BackgroundCol
[{"$Name":"SignUpBtn","$Type":"Button","$Version":"6","BackgroundColor":"&H00FFFFFF","Text":"Sign Up","Uuid":"-2024365814"}]}]},
{"$Name":"HorizontalArrangement3","$Type":"HorizontalArrangement","$Version":"3","AlignHorizontal":"2","BackgroundColor":"&H00FFFFFF"
[{"$Name":"SignBtn","$Type":"Button","$Version":"6","BackgroundColor":"&H00FFFFFF","Text":"Sign Up","Uuid":"1042598961"}]},
{"$Name":"TinyWebDB1","$Type":"TinyWebDB","$Version":"2","ServiceURL":"l","Uuid":"-378396360"},
{"$Name":"TinyDB1","$Type":"TinyDB","$Version":"1","Uuid":"708141961"},{"$Name":"Clock1","$Type":"Clock","$Version":"3","Uuid":"18109
{"$Name":"Notifier1","$Type":"Notifier","$Version":"4","Uuid":"-1240548683"}]}}
|#
```


Figura 4.6: Ejemplo fichero .scm

- Fichero **.bky**: en él se almacena en formato XML (Extensible Markup Language)[5] todas las relaciones entre bloques activos y su funcionamiento. Será por tanto el principal fichero del que extraeremos la información estructural y funcional a analizar.


```
<xml xmlns="http://www.w3.org/1999/xhtml">
  <block type="component_event" id="1" x="867" y="-418">
    <mutation component_type="Button" instance_name="SignUpBtn" event_name="Click"></mutation>
    <field name="COMPONENT_SELECTOR">SignUpBtn</field>
    <statement name="DO">
      <block type="controls_if" id="2" inline="false">
        <mutation else="1"></mutation>
        <value name="IF0">
          <block type="logic_negate" id="3" inline="false">
            <value name="BOOL">
              <block type="lists_is_in" id="4" inline="false">
                <value name="ITEM">
                  <block type="text_trim" id="5" inline="false">
                    <value name="TEXT">
                      <block type="component_set_get" id="6">
                        <mutation component_type="TextBox" set_or_get="get" property_name="Text" is_generic="false" instance_name="UsernameSU"></
mutation>
                        <field name="COMPONENT_SELECTOR">UsernameSU</field>
                        <field name="PROP">Text</field>
                      </block>
                    </value>
                  </block>
                </value>
                <value name="LIST">
                  <block type="lexical_variable_get" id="7">
                    <field name="VAR">global userByUsername</field>
                  </block>
                </value>
              </block>
            </value>
          </block>
        </value>
        <statement name="DO0">
          <block type="lists_add_items" id="8" inline="false">
```


Figura 4.7: Ejemplo fichero .bky

[5] https://www.w3.org/XML/

La función *views.showUserAnalyzeProjectsPage* será la encargada de recibir la identificación del proyecto a analizar y responder al usuario con su evaluación. Tras consultar el contenido de las pantallas en base de datos, llamará a la función *scoreMyApp.getScore* que será la encargada de analizar cada nivel individualmente. En este proceso, las funciones llamadas se han organizado en diferentes clases según su propósito:

- **scoreMyApp**: contiene todas las funciones relativas a analizar el XML y obtener las estadísticas. Su función principal es *scoreMyApp.getScore* y en esta sección nos centraremos en su análisis.

- **scoreMyAppMessages**: procesa las clasificaciones y genera mensajes personalizados para el usuario.

Dentro de *scoreMyApp.getScore* analizaremos cada nivel pantalla por pantalla, comparando los resultados en cada iteracción de forma que la estructura final contenga toda la información del programa.

```
if componentLevels['Score'] > generalScore['ComponentLevels']['Score']:
# Update
generalScore['ComponentLevels']['Score'] = componentLevels['Score']

G1 = generalScore['ComponentLevels']['L1_components']
S1 = componentLevels['L1_components']
generalScore['ComponentLevels']['L1_components'] = returnUnion(G1,S1,0)
```

La clasificación se almacenará en un diccionario en el que guardaremos las estadísticas de cada nivel. Para obtener las puntuaciones individuales se obtiene una media con los niveles de las clasificaciones, mientras que la clasificación final es una media ponderada de los tres puntos a analizar. Los componentes utilizados y las habilidades de programación tienen un peso del 45 % cada uno en la media mientras que la usabilidad de la aplicación se lleva el 10 % restante al considerarse una característica a evaluar pero que no debe tener la misma importancia en la nota final.

Nivel	**Subniveles**	**Ponderación**
ComponentLevels	Score,L1_components,L2_components,L3_components	45 %
ProgrammingLevels	Score,Flow,Data,Variable,Generalization	45 %
ScreensLevels	Score,Screens	10 %

Figura 4.8: Estructura de la clasificación

Para la evaluación de los **componentes** se han clasificado en tres niveles todos los bloques disponibles en App Inventor en función de su nivel de complejidad, independientemente de su naturaleza. Por ejemplo, en el nivel Bajo nos encontramos con bloques tipo cuadro de texto y relojes, mientras que en el nivel Alto tenemos componentes de Lego Mindstorms o bases de datos experimentales como FirebaseDB.

Subnivel	**Bloques**
Bajo	**InterfazUsuario** [Button CheckBox DatePicker Image Label ListPicker ListView Notifier Slider Spinner TextBox TimePicker] **Diseño** [HorizontalArrangement HorizontalScrollArrangement TableArrangement VerticalArrangement VerticalScrollArrangement] **Media** [ImagePicker] **Dibujo** [Ball Canvas ImageSprite] **Sensores** [Clock]
Medio	**InterfazUsuario** [PasswordTextBox WebViewer] **Media** [Camcorder Camera Player Sound SoundRecorder SpeechRecognizer TextToSpeech VideoPlayer YandexTranslate] **Sensores** [AccelerometerSensor BarcodeScanner OrientationSensor Pedometer] **Social** [ContactPicker EmailPicker PhoneCall PhoneNumberPicker Sharing Texting Twitter] **Almacenamiento** [File TinyDB] **Conectividad** [ActivityStarter BluetoothClient]
Alto	**Sensores** [GyroscopeSensor LocationSensor NearField ProximitySensor] **Almacenamiento** [FusionTablesControl TinyWebDB] **Conectividad** [BluethoothServer Web] **Lego** [NctDrive NctColorSensor NxtLightSensor NxtSoundSensor NxtTouchSensor NxtUltrasonicSensor NxtDirectCommands Ev3Motors Ev3ColorSensor Ev3GyroSensor Ev3TouchSensor Ev3UltrasonicSensor Ev3Sound Ev3UI Ev3Commands] **Experimental** [FirebaseDB]

Figura 4.9: Componentes: subniveles

programación

usabilidad

Envío de resultados

4.2.2. Modelo de datos

4.3. Diseño e implentación del cliente

Capítulo 5

Resultados

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

6.2. Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TFG/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

1. a
2. b

6.3. Lecciones aprendidas

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. a

2. b

6.4. Trabajos futuros

Ningún software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFMs.

6.5. Valoración personal

Finalmente (y de manera opcional), hay gente que se anima a dar su punto de vista sobre el proyecto, lo que ha aprendido, lo que le gustaría haber aprendido, las tecnologías utilizadas y demás.

Apéndice A

Manual de usuario

Bibliografía

[1] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Articial Systems*. Oxford University Press, Inc., 1999.