



**INGENIERÍA DE TELECOMUNICACIÓN Y LICENCIATURA  
EN ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS**

Curso Académico 2016/2017

Trabajo Fin de Carrera

**MY APP ANALYZER**

**Fomentando la programación de móviles entre los  
jóvenes**

Autor : Alexandra Ortega Martín

Tutor : Dr. Gregorio Robles



# Proyecto Fin de Carrera

## My App Analyzer: Fomentando la programación de móviles entre los jóvenes

**Autor :** Alexandra Ortega Martín

**Tutor :** Dr. Gregorio Robles Martínez

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

## **Presidente:**

## **Secretario:**

## Vocal:

y habiendo obtenido la siguiente calificación:

## **Calificación:**

Fuenlabrada, a de de 2017



*Dedicado a  
mi familia  
gracias por estar siempre ahí*



# Agradecimientos

Siempre he visto lejano este momento y aunque parecía que nunca iba a llegar, aquí estoy, a punto de poner punto final a mi primer gran paso. Reconozco que tomé la decisión de estudiar la doble licenciatura de forma un poco precipitada, me gustaban demasiadas carreras y no tenía muy claro por qué rama decantarme. Así que cuando vi la posibilidad de unir una ingeniería en la que formarme en varias especialidades con una licenciatura donde me enseñarían cómo funciona la estructura financiera en la que vivimos, supe que era para mí. Por qué no, soy una persona a la que le gustan los retos y el cambio. A día de hoy puedo decir que fue la mejor decisión que he tomado, no sólo porque he cumplido mi objetivo de saber un poquito más de todo, sino porque el perfil multidisciplinar que he desarrollado me ha ayudado muchísimo en mi trayectoria profesional.

Pero no ha sido fácil, ha sido un camino duro del que hubiera salido de no ser por el apoyo de mi familia, son ellos los que han conseguido que hoy pueda estar escribiendo estas líneas. Gracias papá y mamá por esforzados y ayudarme a estudiar en la universidad, sin vosotros no lo hubiera conseguido. Gracias por empujarme en los momentos de flaqueza, aunque fuera a regañadientes por mi parte. A mi hermano, que con los años hemos descubierto otro punto más en común y siempre me ha animado a seguir mi ‘vena tecnológica’. Y a mi pareja, quien ha estado conmigo desde el principio, siendo un pilar fundamental en todos estos años y es quien ha sufrido el sprint final. Todas esas noches delante de los apuntes, programando a altas horas de la madrugada o intentando descifrar los problemas de clase han merecido la pena. Ya llegó el final y con él, un nuevo comienzo.

Por último quería agradecer a mi tutor Gregorio Robles la oportunidad de realizar el proyecto con él. Durante la carrera he tenido multitud de profesores, pero al final sólo te acuerdas del nombre de unos pocos, los que realmente han conseguido que su asignatura no sea sólo una más y que realmente te apetezca seguir aprendiendo. Gracias por tu cercanía y paciencia.



# Resumen

**My App Inventor** es una aplicación web que permite analizar y clasificar proyectos realizados en App Inventor, una plataforma de desarrollo de aplicaciones móviles para el sistema operativo Android.

A través de un diseño atractivo y una navegación sencilla, el **objetivo** de este proyecto es orientar a los futuros programadores y mejorar su pensamiento computacional. Analizaremos tres aspectos clave de las aplicaciones creadas con App Inventor: componentes utilizados, técnicas de programación aplicadas y usabilidad, y mostraremos los resultados al usuario de forma que no sólo obtenga su nivel, sino que también conozca cómo mejorarlo.

Las múltiples **tecnologías** utilizadas para llevar a cabo este proyecto están orientadas al desarrollo web. Desde el lado del servidor, como base de la aplicación se han utilizado el lenguaje Python 2.7 y su framework Django 1.9., junto con la base de datos SQLite. Centrándonos en la capa *Front end* donde se encuentra el cliente, además de HTML y CSS se ha añadido dinamismo a la web con Bootstrap, JavaScript y AJAX, adaptándola a las nuevas tendencias de maquetación.



# Summary

**My App Inventor** is a web application that analyze and classify projects done in App Inventor, a development platform focused in mobile applications based on Android.

Through an attractive design and easy navigation, the **objective** of this project is to guide future developers and improve its computational thinking. We analize three key points of the apps created with App Inventor: componentes included, applied programming techniques and schedule, displaying the results to the users so that they don't only know their level but also offer tips to improve it.

The multiple **technologies** used in this project are oriented towards web development. At the server side, Python 2.7 and its framework Django 1.9 have been used together with data base SQLite. In the Front end, where the client is located, in addition to HTML and CSS, Bootstrap, JavaScript and AJAX have been included to dinamize the templates and adapt them to the new layout trends.



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Marco General . . . . .	1
1.2. Estructura de la memoria . . . . .	3
<b>2. Objetivos</b>	<b>5</b>
2.1. Objetivo general . . . . .	5
2.2. Objetivos específicos . . . . .	5
2.3. Planificación temporal . . . . .	6
<b>3. Estado del arte</b>	<b>9</b>
3.1. App Inventor . . . . .	9
3.1.1. App Inventor Designer . . . . .	10
3.1.2. App Inventor Blocks . . . . .	12
3.2. Python . . . . .	13
3.3. Django . . . . .	15
3.4. SQLite . . . . .	15
3.5. HTTP . . . . .	16
3.6. HTML5 . . . . .	16
3.7. CSS . . . . .	17
3.8. JavaScript . . . . .	18
3.9. Bootstrap . . . . .	18
3.10. AJAX . . . . .	19
<b>4. Diseño e implementación</b>	<b>21</b>
4.1. Arquitectura general . . . . .	21

4.2. Diseño e implementación del servidor . . . . .	22
4.2.1. Lógica servidor: Django . . . . .	22
4.2.2. Modelo de datos . . . . .	39
4.3. Manual de Usuario . . . . .	42
<b>5. Resultados</b>	<b>51</b>
5.1. Nivel Alto: Multifunctional Application . . . . .	51
5.2. Nivel Medio: Android Mash . . . . .	54
5.3. Nivel Bajo: YOUB Analogic Clock . . . . .	57
<b>6. Conclusiones</b>	<b>61</b>
6.1. Consecución de objetivos . . . . .	61
6.2. Aplicación de lo aprendido . . . . .	62
6.3. Lecciones aprendidas . . . . .	63
6.4. Trabajos futuros . . . . .	63
<b>Bibliografía</b>	<b>65</b>

# Índice de figuras

1.1.	Programación con Logo . . . . .	2
1.2.	Dr. Scratch . . . . .	2
3.1.	Web App Inventor . . . . .	9
3.2.	Web App Inventor . . . . .	10
3.3.	Web App Inventor . . . . .	11
3.4.	Web App Inventor . . . . .	12
3.5.	Lenguajes más importantes en 2016. Fuente: <a href="http://www.codeeval.com">www.codeeval.com</a> . . . . .	14
3.6.	Directorio de archivos Bootstrap . . . . .	18
3.7.	HTML simple VS. con Bootstrap . . . . .	19
4.1.	Arquitectura Cliente-Servidor . . . . .	22
4.2.	Directorio de archivos My App Inventor . . . . .	23
4.3.	Flujo principal My App Inventor . . . . .	24
4.4.	Opciones disponibles . . . . .	26
4.5.	Directorio App Inventor . . . . .	27
4.6.	Ejemplo fichero .scm . . . . .	28
4.7.	Ejemplo fichero .bky . . . . .	28
4.8.	Estructura de la clasificación . . . . .	30
4.9.	Componentes . . . . .	31
4.10.	Programación . . . . .	32
4.11.	Resultados del análisis . . . . .	35
4.12.	Resultados de la categoría Componentes . . . . .	36
4.13.	Resultados de la categoría Programación . . . . .	37

4.14. Resultados de la categoría Usabilidad . . . . .	37
4.15. Versión móvil con Bootstrap . . . . .	39
4.16. Petición de Bootstrap desde el navegador . . . . .	39
4.17. Modelo de datos . . . . .	41
4.18. Pantalla de Bienvenida . . . . .	42
4.19. Pantalla de Inicio de Sesión . . . . .	42
4.20. Pantalla Servicios . . . . .	43
4.21. Pantalla ‘Acerca de’ . . . . .	43
4.22. Link para crear una nueva cuenta . . . . .	44
4.23. Creación de cuenta de usuario . . . . .	44
4.24. Mensaje de error por contraseña inválida . . . . .	45
4.25. Mensaje de confirmación al registrarse . . . . .	45
4.26. Opciones disponibles en vista de usuario . . . . .	46
4.27. Actualización de datos de usuario . . . . .	46
4.28. Mensaje de confirmación de actualización de datos de usuario . . . . .	47
4.29. Añadir un nuevo proyecto . . . . .	47
4.30. Resultados del análisis de nuestro proyecto . . . . .	48
4.31. Ver desglose del análisis por categoría . . . . .	48
4.32. Lista de proyectos guardados . . . . .	49
4.33. Botón para salir de sesión . . . . .	49
4.34. Mensaje de confirmación de cierre de sesión . . . . .	50
5.1. Multifunctional Application: diseño . . . . .	51
5.2. Multifunctional Application: clasificación general . . . . .	52
5.3. Android Mash: diseño . . . . .	54
5.4. Android Mash: clasificación general . . . . .	55
5.5. Analogic Clock: diseño . . . . .	57
5.6. Analogic Clock: clasificación general . . . . .	57

# **Capítulo 1**

## **Introducción**

En una época donde las nuevas tecnologías están totalmente integradas en nuestro día a día y en el que los niños cada vez aprenden antes a usar ordenadores y móviles, es importante desarrollar en los más jóvenes aptitudes lógicas que les conviertan de simples consumidores a creadores. Como futuros informáticos, médicos, profesores o dibujantes, el pensamiento computacional les ayudará a pensar de una forma organizada, metódica y secuencial, capacidades que les serán de ayuda sea cual sea su profesión. Pero ¿cómo podemos enseñarles informática cuando son tan pequeños? Las metodologías clásicas de aprendizaje en las que la principal barrera de entrada es un lenguaje complicado y abstracto han dado paso a nuevas formas de enseñar a programar donde aplicaciones sencillas y con un diseño atractivo permiten a los jóvenes ‘aprender jugando’.

### **1.1. Marco General**

Ya en la década de los 60 se detectó la necesidad de acercar la tecnología a la sociedad con Logo<sup>1</sup>, un lenguaje simple donde por medio de instrucciones básicas se guiaba a una tortuga para crear formas geométricas en la pantalla. Logo fue el precursor de los nuevos lenguajes y aplicaciones como Blockly<sup>2</sup> y su metodología de interconexión de bloques, Scratch<sup>3</sup>, App Inventor o Alice<sup>4</sup>, donde podemos crear animaciones en 3D.

---

<sup>1</sup>[http://el.media.mit.edu/logo-foundation/what\\_is\\_logo/history.html](http://el.media.mit.edu/logo-foundation/what_is_logo/history.html)

<sup>2</sup><https://developers.google.com/blockly/>

<sup>3</sup><https://scratch.mit.edu/>

<sup>4</sup><http://www.alice.org/index.php>

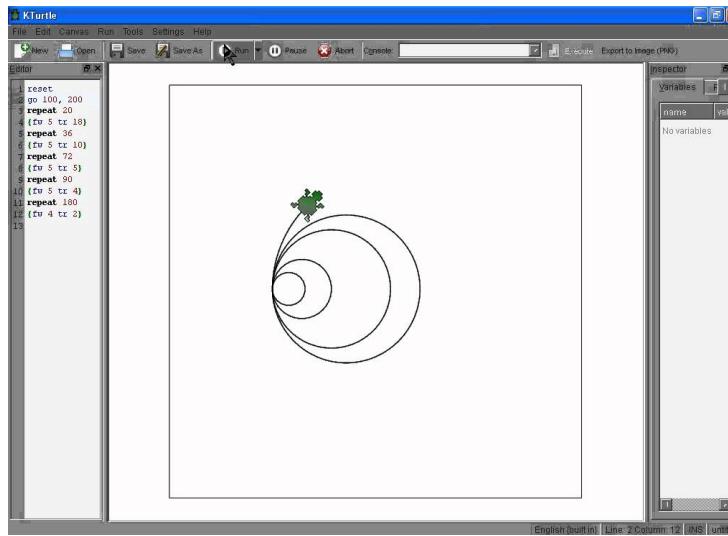


Figura 1.1: Programación con Logo

Siguiendo la idea de mis compañeros de carrera y su proyecto Dr. Scratch<sup>5</sup>, mi tutor Gregorio Robles me propuso crear una línea paralela de trabajo donde poder continuar con la labor educativa de esta aplicación pero centrándonos en otro modelo de programación. Tras analizar las alternativas, elegimos App Inventor, que mantiene la metodología de programación por bloques de Scratch o Blockly enfocándola a un público de mayor edad.

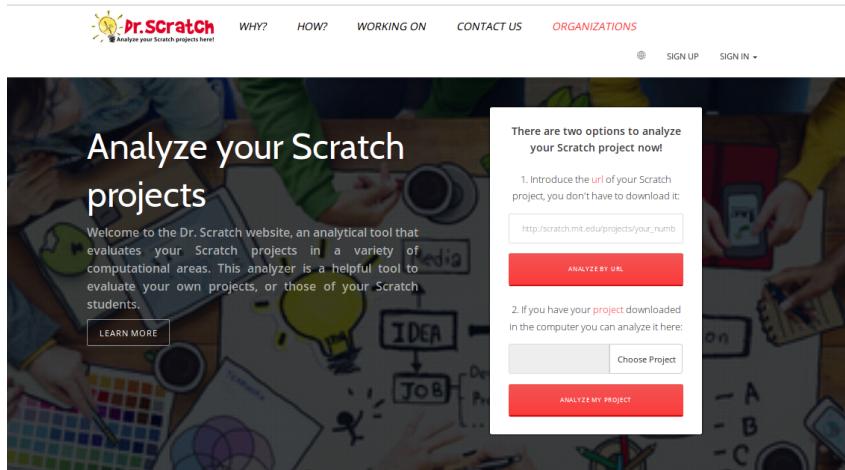


Figura 1.2: Dr. Scratch

Dr. Scratch es una aplicación web que analiza programas creados en Scratch. Tomando como base la librería Hairball<sup>6</sup>, muestra información sobre aspectos del pensamiento compu-

<sup>5</sup><http://www.drscratch.org/>

<sup>6</sup><https://github.com/ucsb-cs-education/hairball>

tacional tales como pensamiento lógico, control de flujo, abstracción, paralelismo, sincronización... Ofreciendo una puntuación al usuario que le servirá de guía para conocer el nivel de su programa. Los puntos clave del análisis vienen determinados por las posibilidades que ofrece Scratch respecto a bloques y configuración, que varían respecto a los que nos encontramos en App Inventor. Otra diferencia notable entre Scratch y App Inventor la encontramos en la salida que proporcionan. Mientras que el primero está enfocado a aplicaciones web, el segundo se centra en las aplicaciones para móviles Android. Además, como hemos comentado anteriormente, App Inventor está dirigido a un público joven, ofreciéndole la posibilidad de utilizar el hardware de sus *smartphones*, como la cámara o el *Bluetooth*, e incluso interactuar con robots Lego Mindstorms.

## 1.2. Estructura de la memoria

A continuación veremos la estructura de esta memoria y sus capítulos principales:

- Introducción: presentación del trabajo y del contexto social en el que se desarrolla, además de la motivación para afrontarlo.
- Objetivos: desarrollo de qué es lo que buscamos con nuestra aplicación y los pasos seguidos para obtener el resultado final.
- Estado del arte: breve introducción a las principales tecnologías utilizadas.
- Diseño e implementación: arquitectura de My App Inventor, funcionalidades y cómo se han utilizado las diferentes tecnologías. Incluye un manual de usuario.
- Resultados: análisis del resultado final del proyecto.
- Conclusiones: reflexión sobre los resultados obtenidos, consecución de objetivos y problemas encontrados.



# **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 numéricas, 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 numéricas 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 familiarizar 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 simplificación 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<sup>1</sup> 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. Comencé esta fase en agosto de 2016.
- **Fase II:** entre septiembre de 2016 y febrero de 2017 realicé las siguientes tareas:
  - 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. Esta parte del proyecto fue desarrollada entre febrero y marzo de 2017.

---

<sup>1</sup><http://appinventor.mit.edu/explore/>

- **Fase IV:** incorporar a la web las tecnologías Bootstrap y AJAX para añadir dinamismo y mejorar su aspecto. La última fase ocupó los meses de abril y mayo, donde comencé a escribir la memoria.



# Capítulo 3

## Estado del arte

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

### 3.1. App Inventor

MIT App Inventor es un entorno de desarrollo web para la creación de aplicaciones móviles basadas en el sistema Android. Su principal objetivo es democratizar la programación y acercarla al público más joven transformándolo de consumidor de tecnología, a creador de la misma.

*'Anyone Can Build Apps That Impact the World'*



Figura 3.1: Web App Inventor

Gracias a su entorno gráfico sencillo e intuitivo, permite la construcción gratuita de programas reduciendo la curva de aprendizaje y eliminando barreras de entrada ya que para utilizarlo

sólo es necesario un ordenador con internet y un móvil con sistema operativo Android. Pero además de las ventajas desde el punto de vista tecnológico también hay que añadir el componente social. En su web nos permiten no sólo subir el código sino que además se pueden ver y descargar gratuitamente los programas de otros usuarios para usarlos o modificarlos. Desde su creación en 2015 es utilizado en colegios para mejorar la visión lógica de los estudiantes y crear comunidades educativas que conectan a jóvenes de cualquier parte del mundo.

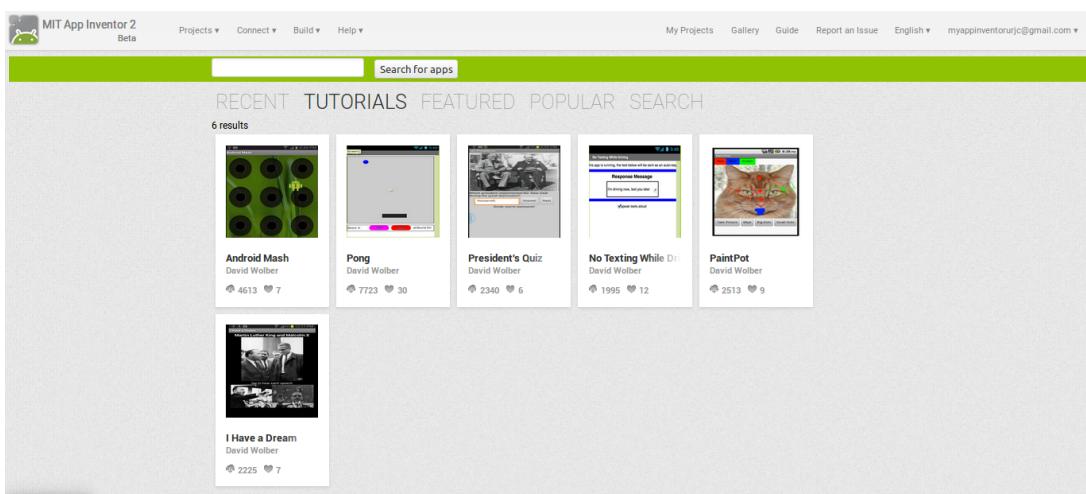


Figura 3.2: Web App Inventor

Para poder utilizarlo hay que tener una cuenta de Google y entrar en su web<sup>1</sup>. Tras iniciar sesión y seleccionar la aplicación, disponemos de dos herramientas: Designer y Blocks. Ambas funcionan con de la funcionalidad *drag & drop*, una vez decidido qué componente o bloque utilizar, lo arrastraremos hasta la previsualización de pantalla (*Viewer*) para situarlo en una zona determinada o conectarlo con otros elementos.

### 3.1.1. App Inventor Designer

En este apartado se construye la interfaz de usuario del programa, añadiendo y personalizando los componentes.

<sup>1</sup><http://appinventor.mit.edu/explore/>

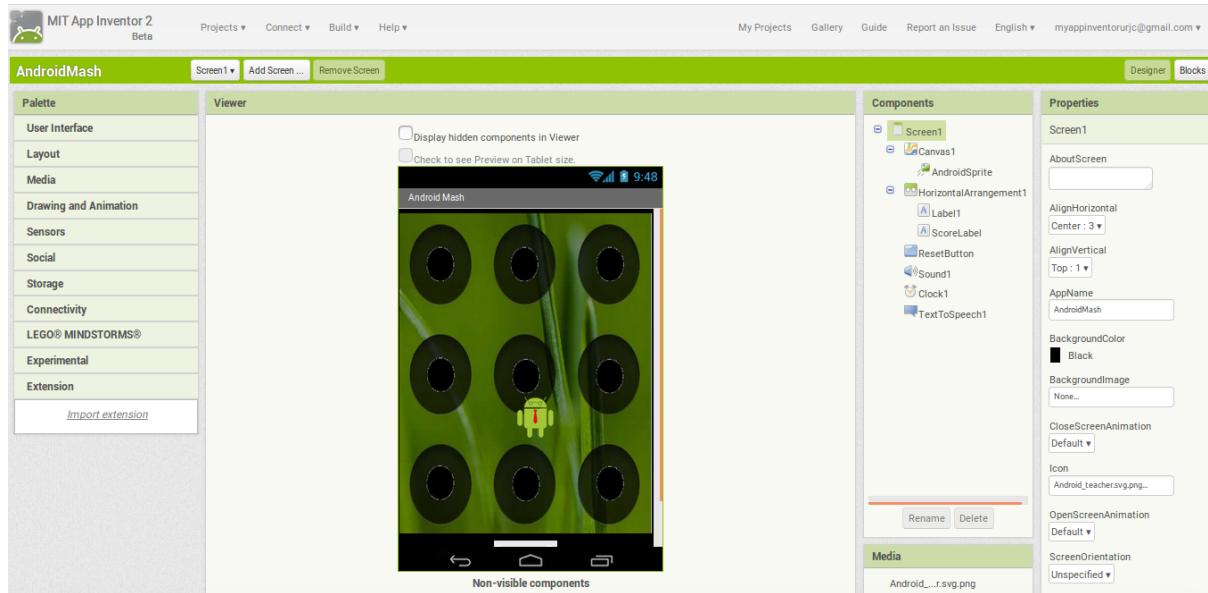


Figura 3.3: Web App Inventor

Incluye una amplia librería que nos permite escoger entre componentes de distinta complejidad y que serán objeto de análisis en nuestra aplicación como veremos en el siguiente capítulo:

- *User Interface*: botones, imágenes, casillas y todo tipo de elementos que forman parte de la interfaz de usuario y forman la estructura básica del programa.
- *Layout*: crea secciones horizontales o verticales en la pantalla.
- *Media*: cámara de fotos o vídeo, reproductor de audio, grabadora...
- *Drawing and animation*: paneles de dibujo o delimitador de zonas para interactuar con el usuario a través de la pantalla táctil.
- *Sensors*: reloj, geolocalizador, podómetro, sensor de proximidad...
- *Social*: permite acceder a la lista de contactos, compartir ficheros o texto e incluso conectarse a Twitter.
- *Storage*: con gestores de ficheros y bases de datos.
- *Connectivity*: creación de servidor/cliente de Bluetooth y posibilidad de realizar peticiones HTTP.

- *Lego Mindstorms*<sup>2</sup>: interfaz de alto nivel para controlar los componentes de los robots Lego.
- *Experimental*: como alternativa al almacenamiento de datos incluido en *Storage*, se puede incluir Firebase<sup>3</sup>, una base de datos creada por Google optimizada para dispositivos móviles.
- *Extension*: importador de extensiones de terceros.

### 3.1.2. App Inventor Blocks

Todo en App Inventor está causado por un evento: el click en un botón, la finalización de un reloj, el inicio de una conexión... Tras crear la máscara de la aplicación con los componentes, la dotaremos de lógica en la vista Blocks.

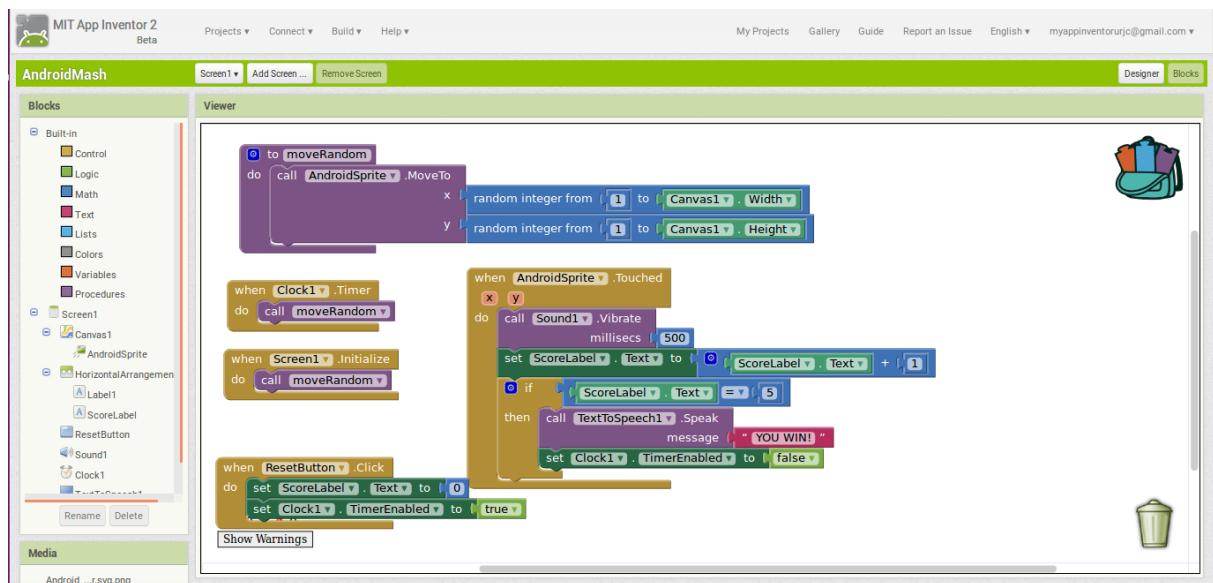


Figura 3.4: Web App Inventor

Por pantalla podremos insertar:

- *Built-in*: bloques lógicos predefinidos que podemos combinar entre sí:
  - *Control*: condicionales y bucles.

<sup>2</sup><https://www.lego.com/en-gb/mindstorms/>

<sup>3</sup><https://firebase.google.com/>

- *Logic*: comparadores, booleanos (*True/False*) y puertas lógicas (*and/or*)
  - *Math*: operaciones matemáticas algebraicas y trigonométricas.
  - *Text*: funciones para variables tipo texto: contiene, longitud, concatenación...
  - *Lists*: creación, consulta y modificación de listas de variables.
  - *Colors*: selector de color.
  - *Variables*: creación, inicialización y modificación de variables locales y globales.
  - *Procedures*: definición de procedimientos
- 
- *Screen*: gestión de eventos sobre los componentes añadidos en cada pantalla.
  - *Any Component*: gestión de eventos globales por componente. Por ejemplo, podemos definir una acción a ejecutar cuando se pulse un botón en cualquiera de las pantallas que componen el programa.

## 3.2. Python

Python es un potente lenguaje de programación de alto nivel entre cuyas características principales destaca una: la sencillez. Gracias a su facilidad de uso y de lectura, cualquier programador puede entender su sintaxis y adaptarla a sus necesidades. Además es de código abierto y multiplataforma, es decir, puede redistribuirse gratuitamente en cualquier sistema operativo: Linux, Unix, Windows o MAC OS.

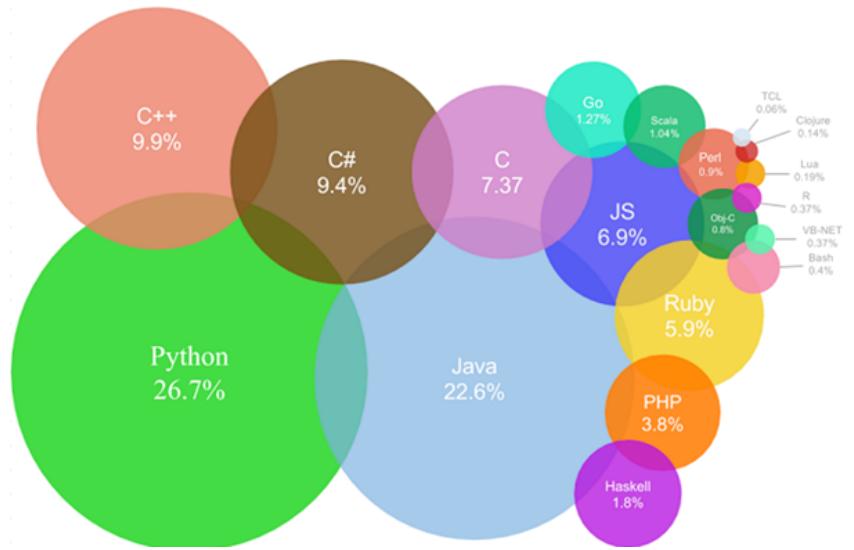


Figura 3.5: Lenguajes más importantes en 2016. Fuente: [www.codeeval.com](http://www.codeeval.com)

Algunas características a destacar:

- Sintaxis sencilla: como hemos comentado, se busca una escritura simple, reduciendo caracteres redundantes y favoreciendo su interpretación rápida. Como ejemplo de esto, la definición de los bloques por el nivel del sangrado (eliminando llaves o delimitadores de fin de instrucción) o la sustitución de los caracteres lógicos `&&`, `||` y `!` por palabras como *and*, *or* y *not*.
- Incluye multitud de librerías optimizadas para reducir el código y el tiempo de ejecución.
- Dinámicamente tipado: no es necesario definir los tipos de datos previamente sino que podemos ir añadiendo variables y automáticamente Python les asignará un tipo en función de su contenido.
- Fuertemente tipado: no convierte automáticamente unos tipos de datos en otros.
- Incluye un modo interactivo donde poder ver el resultado de las instrucciones inmediatamente sin necesidad de compilar previamente el programa.
- Soporta programación orientada a objetos.
- Puede extenderse añadiendo módulos implementados en otros lenguajes como C o C++.

### 3.3. Django

Django es un framework de alto nivel escrito en Python para construir aplicaciones web de forma gratuita, sencilla y rápida. Incluye todas las librerías necesarias que pueden utilizarse en el desarrollo web, de manera que el programador sólo tiene que preocuparse en diseñar su aplicación sin necesidad de ‘reinventar la rueda’.

Principales características:

- Rápido: las funciones están optimizadas para que su ejecución sea lo más rápida posible.
- Seguro: ayuda a los desarrolladores a evitar los errores más comunes en seguridad.
- Escalable: es flexible y fácilmente escalable, adaptándose a las necesidades de la aplicación.
- Código abierto: el uso de funciones predefinidas no excluye que el programador tenga acceso a su contenido para reutilizar o modificarlas. Todo el código es accesible para los usuarios.
- Versátil: gracias a Django podemos crear todo tipo de aplicaciones como servidores de contenidos ([www.openstack.org](http://www.openstack.org)) o redes sociales([www.instagram.com](http://www.instagram.com)).

### 3.4. SQLite

SQLite es un sistema de gestión de bases de datos basado en el estándar SQL. Sus principales características son:

- Transacciones atómicas, consistencia, aislamiento, y durabilidad ante fallos de sistema.
- No necesita configuración previa para poder utilizarlo.
- Multiplataforma: Android, BSD, iOS, Linux, Mac, Solaris, VxWorks y Windows
- No es una parte independiente del programa con el que se comunica sino que se integra con él, reduciendo la latencia de acceso. La base de datos se guarda como un fichero estándar dentro del sistema.
- De dominio público.

### 3.5. HTTP

HTTP son las siglas de Hypertext Transfer Protocol, un protocolo de comunicación web a nivel aplicación. Se utiliza en aplicaciones distribuidas y define los mensajes a intercambiar entre cliente y servidor dentro del protocolo pregunta/respuesta.

- No mantiene estado: no guarda información de conexiones anteriores. En caso de necesitar hacerlo se utilizan cookies, ficheros de información almacenados en el cliente por el servidor.
- Es colaborativo.
- Los mensajes enviados están escritos en texto plano y siguen la estructura:
  - Método de petición + URL del recurso + versión HTTP del cliente. Concretamente en nuestra aplicación usaremos los métodos GET para petición simple de recursos (solicitar una página web) y POST para envío de datos de cara a su procesamiento (envío de datos de formulario).
  - Cabecera con metadatos: idioma, tipo de servidor, fecha, cookie...
  - Cuerpo con datos a intercambiar.

### 3.6. HTML5

*HyperText Markup Language* o HTML es el lenguaje estándar de la World Wide Web (WWW) en el que se define el formato para elaborar las páginas web. Está basado en ficheros de texto plano donde la interpretación del código recaerá sobre el navegador del cliente, que será el encargado de solicitar los recursos especificados y mostrar el contenido correctamente.

Los ficheros HTML utilizan etiquetas y atributos para identificar cada elemento del documento:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>App Inventor Analyzer</title>
  </head>
```

```

<body>
  <p>Cuerpo de la web</p>
</body>
</html>

```

Actualmente se encuentra en la versión 5 cuyas principales novedades permiten mayor dinamismo en las páginas web: etiquetas para 2D y 3D, generación de tablas dinámicas, nuevos tipos de datos en formularios o funcionalidad Drag & Drop.

## 3.7. CSS

*Cascading Style Sheets* o CSS es un lenguaje de diseño gráfico utilizado para dar estilo a ficheros HTML y XML. A través de la separación entre contenido y formato permiten eliminar líneas redundantes en los documentos, realizar modificaciones globales más fácilmente y adaptar la visualización para diferentes métodos de renderizado. Para utilizar CSS se necesita:

- Fichero .css: en el se definen las reglas mediante pares selector - bloque de declaración.

```

header .intro-text .intro-heading {
  font-family: "Montserrat", "Helvetica Neue", Helvetica, Arial, sans-serif;
  text-transform: uppercase;
  font-weight: 700;
  font-size: 50px;
  margin-bottom: 25px;
}

```

- Fichero .html: donde se identifican los elementos a formatear con el selector especificado en el fichero .css.

```

<div class="intro-text">
  <div class="intro-lead-in">Welcome to MIT App Inventor Analyzer</div>
  <div class="intro-heading">It's Nice To Meet You</div>
  <a href="#services" class="page-scroll btn btn-xl">Tell Me More</a>
</div>

```

## 3.8. JavaScript

JavaScript es un lenguaje de programación interpretado utilizado principalmente en el lado del cliente para añadir dinamismo a las páginas web mostradas por el navegador. Al ser interpretado, no necesita ser compilado previamente ya que se interpreta a tiempo real, mejorando la eficiencia de las aplicaciones.

## 3.9. Bootstrap

Bootstrap es un framework de código abierto para el desarrollo de páginas y aplicaciones web creado por desarrolladores de Twitter. Este conjunto de herramientas une las tecnologías HTML, CSS, y Javascript para adaptar dinámicamente el formato de las webs al dispositivo utilizado (ordenador, tableta o móvil). La estructura básica de Bootstrap sigue el sistema de ficheros de la figura 3.6. En ella podemos ver cómo los ficheros CSS y JavaScript base vienen incluidos en su versión normal (\*.css) y minimizada (\*.min.css). Esta última es una compresión en la que se eliminan caracteres innecesarios en el código como espacios o saltos de línea, y su objetivo es reducir el tiempo de procesado de los ficheros por el navegador.

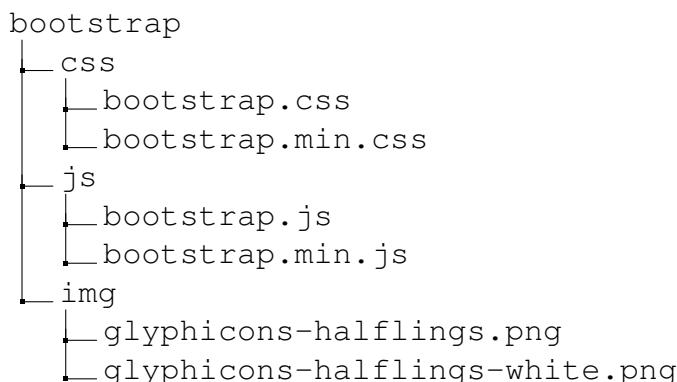


Figura 3.6: Directorio de archivos Bootstrap

---

HTML simple	HTML con Bootstrap
<pre>&lt;!DOCTYPE html&gt; &lt;html&gt; &lt;head&gt; &lt;title&gt;Bootstrap 101 Template&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;h1&gt;Hello , world !&lt;/h1&gt; &lt;script src="http://code.jquery.com/ jquery.js"&gt;&lt;/script&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<pre>&lt;!DOCTYPE html&gt; &lt;html&gt; &lt;head&gt; &lt;title&gt;Bootstrap 101 Template&lt;/title&gt; &lt;!-- Bootstrap --&gt; &lt;link href="css/bootstrap.min.css" rel="stylesheet" media="screen"&gt; &lt;/head&gt; &lt;body&gt; &lt;h1&gt;Hello , world !&lt;/h1&gt; &lt;script src="http://code.jquery.com/ jquery.js"&gt;&lt;/script&gt; &lt;script src="js/bootstrap.min.js"&gt;&lt;/ script&gt; &lt;/body&gt; &lt;/html&gt;</pre>

---

Figura 3.7: HTML simple VS. con Bootstrap

## 3.10. AJAX

*Asynchronous JavaScript And XML* o AJAX es una técnica de desarrollo asíncrona para actualización de páginas web bajo petición sin necesidad de recargarlas completamente, lo que mejora la eficiencia de las aplicaciones. Utilizando AJAX podemos utilizar varias tecnologías: HTML (datos), CSS (estilo), Document Object Model o DOM (actualización dinámica de contenido), XMLHttpRequest (petición al servidor) y XML (formato de envío de información).

En nuestra aplicación My App Inventor podemos ver un ejemplo de esta tecnología en la plantilla *userprofile.html* donde actualizaremos el contenido de la página en función de la opción escogida por el usuario: ver lista de proyectos, guardar un nuevo proyecto o actualizar perfil.

```
<a class="page-scroll" href="#usercontent" onclick="loadUserProjects()">
    My projects</a>
<a class="page-scroll" href="#usercontent" onclick="loadNewProject()"">
    Add New</a>
<a class="page-scroll" href="#usercontent" onclick="loadUpdateProfile()">
    Update profile </a>

...
<!-- Load UserProjects JavaScript --&gt;
&lt;script&gt;
function loadUserProjects() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 &amp;&amp; this.status == 200) {
            document.getElementById("usercontent").innerHTML =
                this.responseText;
        }
    };
    xhttp.open("GET", "/userprojects/", true);
    xhttp.send();
}
&lt;/script&gt;</pre>
```

# 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<sup>1</sup> 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

---

<sup>1</sup><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)

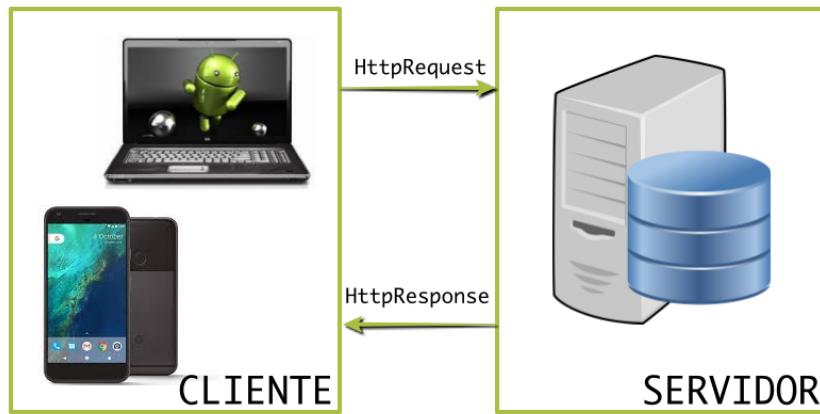


Figura 4.1: Arquitectura Cliente-Servidor

## 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:

- Proyecto 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.

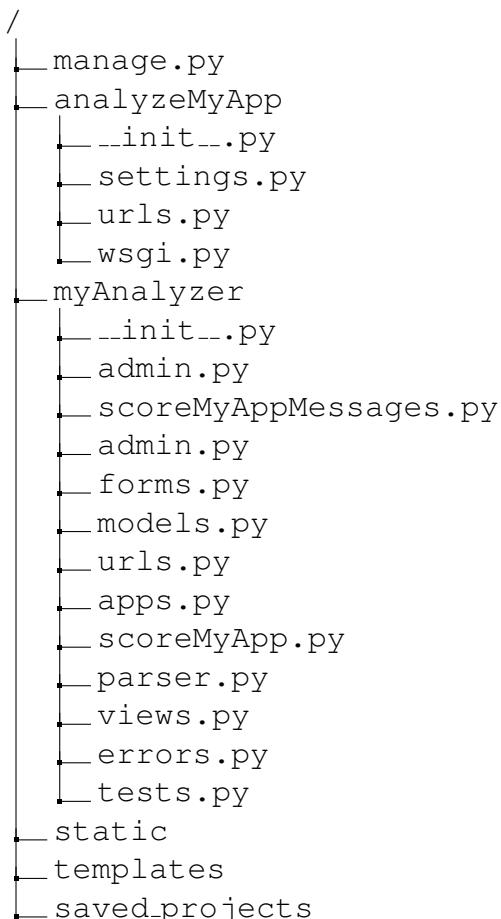


Figura 4.2: Directorio de archivos My App Inventor

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 análisis 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.

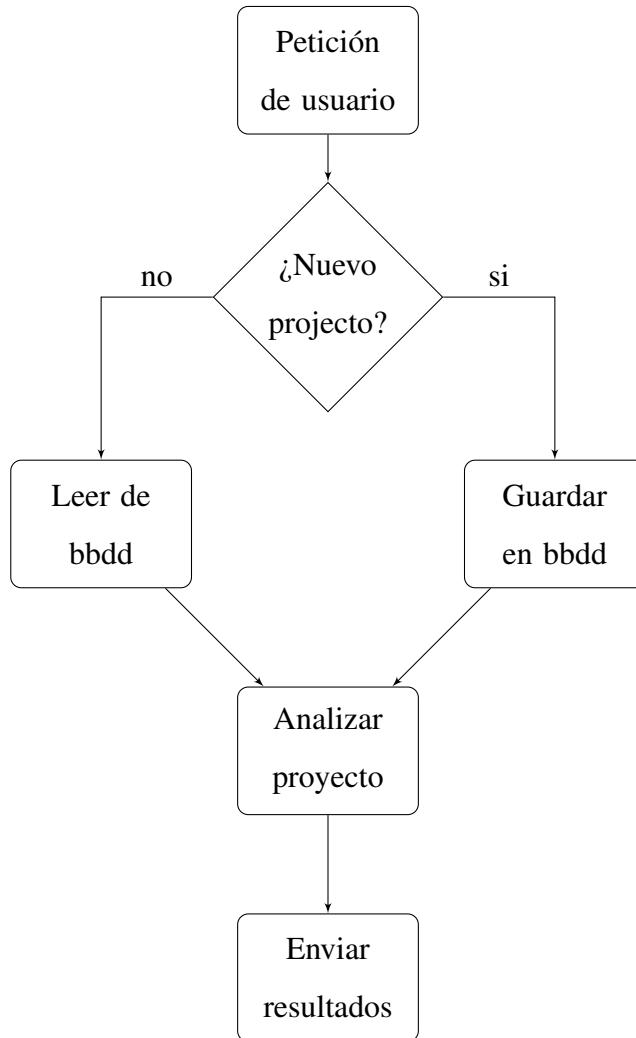


Figura 4.3: Flujo principal My App Inventor

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 redirige 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<sup>2</sup> 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`<sup>3</sup> 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 `login` (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.

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.

---

<sup>2</sup><https://docs.djangoproject.com/en/1.11/topics/auth/default/>

<sup>3</sup><https://docs.djangoproject.com/en/1.11/ref/request-response/>

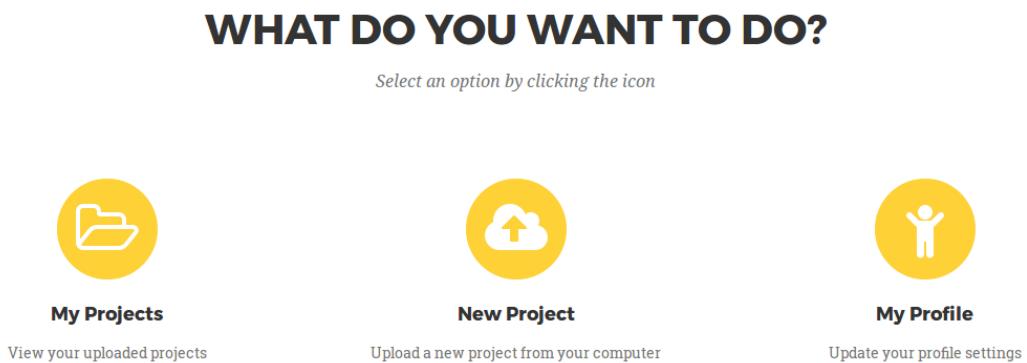


Figura 4.4: Opciones disponibles

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 descomprimirá, 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.

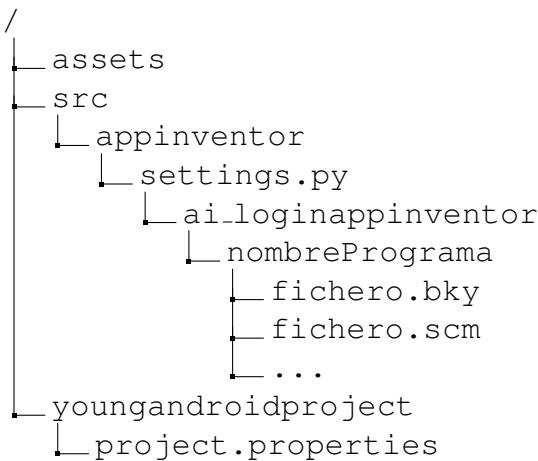


Figura 4.5: Directorio App Inventor

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)<sup>4</sup>, un formato de representación de objetos con una estructura sencilla basada en relaciones nombre-valor. En función del bloque utilizado

---

<sup>4</sup><http://www.json.org/>

tendremos 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": "Multifunctionalapplication"}, {"$Name": "HorizontalArrangement1", "$Type": "HorizontalArrangement", "$Version": "3", "BackgroundColor": "&H00FFFFFF", "Width": "-2", "Uuid": "160039573"}, {"$Name": "VerticalScrollArrangement1", "$Type": "VerticalScrollArrangement", "$Version": "1", "AlignHorizontal": "3", "BackgroundColor": "&H00FFFFFF", "Width": "-2", "Uuid": "827897089"}, {"$Name": "LoginBox", "$Type": "TextBox", "$Version": "5", "Width": "-2", "Hint": "Username, e-mail or phone", "Uuid": "1160039573"}, {"$Name": "LoginPasswordBox", "$Type": "PasswordTextBox", "$Version": "3", "Width": "-2", "Hint": "Password", "Uuid": "827897089"}, {"$Name": "VerticalArrangement1", "$Type": "VerticalArrangement", "$Version": "3", "AlignVertical": "2", "BackgroundColor": "&H00FFFFFF", "Height": "100"}, {"$Name": "SignInBtn", "$Type": "Button", "$Version": "6", "BackgroundColor": "&H00FFFFFF", "Text": "Sign In", "Uuid": "-756982208"}, {"$Name": "HorizontalArrangement2", "$Type": "HorizontalArrangement", "$Version": "3", "BackgroundColor": "&H00FFFFFF", "Width": "-2", "Uuid": "160039573"}, {"$Name": "VerticalScrollArrangement2", "$Type": "VerticalScrollArrangement", "$Version": "1", "BackgroundColor": "&H00FFFFFF", "Width": "-2", "Uuid": "827897089"}, {"$Name": "FullNameSU", "$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": "-60427488"}, {"$Name": "PasswordsU", "$Type": "PasswordTextBox", "$Version": "3", "Width": "-2", "Hint": "Password", "Uuid": "-1827733330"}, {"$Name": "CPasswordsSU", "$Type": "PasswordTextBox", "$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", "Background": "#000000", "Width": "100"}, {"$Name": "HorizontalArrangement5", "$Type": "HorizontalArrangement", "$Version": "3", "AlignHorizontal": "3", "AlignVertical": "2", "Background": "#000000", "Width": "100"}, {"$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", "BackgroundColor": "#000000", "Width": "100"}, {"$Name": "HorizontalArrangement3", "$Type": "HorizontalArrangement", "$Version": "3", "AlignHorizontal": "2", "BackgroundColor": "&H00FFFFFF", "Width": "100"}, {"$Name": "SignBttn", "$Type": "Button", "$Version": "6", "BackgroundColor": "&H00FFFFFF", "Text": "Sign Up", "Uuid": "1042598961"}, {"$Name": "TinyWebDB1", "$Type": "TinyWebDB", "$Version": "2", "ServiceURL": "1", "Uuid": "-378396360"}, {"$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)<sup>5</sup> 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="D00">
<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 name="D00">
<block type="lists_add_items" id="8" inline="false">
```

Figura 4.7: Ejemplo fichero .bky

<sup>5</sup><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 analizará 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 iteración 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 puntuación final es una media ponderada de los tres niveles 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 pues es un concepto más de diseño que de habilidad en la programación.

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

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

de datos experimentales como FirebaseDatabase. Una vez definidos los tres niveles, con la función *scoreMyApp.componentLevels\_Score* buscaremos todas las ocurrencias de cada tipo de bloque dentro de la pantalla, guardando los resultados en tres diccionarios:

```
L1_found = { 'UserInterface': UserInterface_L1_found ,
             'Layout': Layout_L1_found ,
             'Media': Media_L1_found ,
             'Drawing': Drawing_L1_found ,
             'Sensors': Sensors_L1_found }

L2_found = { 'UserInterface': UserInterface_L2_found ,
             'Media': Media_L2_found ,
             'Sensors': Sensors_L2_found ,
             'Social': Social_L2_found ,
             'Storage': Storage_L2_found ,
             'Connectivity': Connectivity_L2_found }

L3_found = { 'Sensors': Sensors_L3_found ,
             'Storage': Storage_L3_found ,
             'Connectivity': Connectivity_L3_found ,
             'Lego': Lego_L3_found ,
             'Experimental': Experimental_L3_found }
```

Tras contabilizar el número de componentes total en cada subnivel, se asignará la puntuación de este módulo en función de la nota máxima alcanzada y guardaremos tanto la puntuación como el análisis para poder utilizarlos más adelante.

```
if c3 > 0 : # High score for complex components
score = 3
elif c2 > 0: # Medium score for complicated components
score = 2
```

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

Figura 4.9: Componentes

```

else: # Low score for simple components
score = 1

results = { 'Score':score ,
            'L1_components':L1_found ,
            'L2_components': L2_found ,
            'L3_components': L3_found}

```

En la **programación** se revisarán competencias del usuario en distintas áreas de carácter abstracto como el control de flujo o el uso de funciones. Se han intentado agrupar a alto nivel las posibles etiquetas para poder organizarlas por funcionalidad y dentro de la misma, en los mínimos grupos, de manera que para cada subnivel tendremos siempre dos grupos globales que nos ayudarán a determinar la puntuación.

Subnivel	Identificadores
Flow	component_event_controls_
Data	lists component_set_get
Variable	global_declaration lexical_variable_get lexical_variable_set math_text_local_declaration_statement local_declaration_expression
Generalization	procedures_is_generic

Figura 4.10: Programación

- **Control de flujo** *scoreMyApp.flow\_control*: identifica las expresiones lógicas para ejecutar ciclos, estructuras condicionales o eventos que dependen del comportamiento de otro componente o variable.
  - Nivel alto: se incluyen expresiones lógicas y eventos dependientes.
  - Nivel medio: se incluyen expresiones lógicas o eventos dependientes.
  - Nivel bajo: no incluye ningún control de flujo.
- **Gestión de datos** *scoreMyApp.data\_control*: detecta el uso de listas para organización datos y asignación o modificación de valores a variables y componentes.
  - Nivel alto: se incluyen listas y modificadores de variables.

- Nivel medio: se incluyen listas o modificadores de variables.
  - Nivel bajo: no se utilizan etiquetas de gestión de datos.
- **Representación de variables** *scoreMyApp.variable\_control*: se tendrá en cuenta si el usuario utiliza declaraciones de variables tanto globales como locales y, en función del tipo de programa, expresiones matemáticas (sumas, potencias, operaciones de trigonometría...) o expresiones relacionadas con textos (unión, convertir a mayúsculas, contiene...).
- Nivel alto: se incluyen declaración de variables y funcionalidades de matemáticas o texto.
  - Nivel medio: se incluyen declaración de variables o funcionalidades de matemáticas o texto.
  - Nivel bajo: no se utilizan métodos referenciados a variables.
- **Generalización** *scoreMyApp.generalization\_control*: la optimización de código a través de funciones y la generalización de procesos para un mismo tipo de variable son características que todo buen programa debe tener ya que facilitan su lectura y permiten la reutilización procedimientos.
- Nivel alto: se incluyen procedimientos y eventos genéricos para un mismo tipo de componente.
  - Nivel medio: se incluyen procedimientos (en esta ocasión se ha considerado que el uso de procedimientos es una funcionalidad básica e imprescindible)
  - Nivel bajo: no se utiliza ningún tipo de mecanismo de generalización.

Al igual que en el nivel anterior, se guardarán todas las estadísticas e identificadores obtenidos en un diccionario además de la nota media de todas las subcategorías:

```
results = { 'Score' : score ,
            'Flow' : flowCtrl ,
            'Data' : dataCtrl ,
            'Variable' : variableCtrl ,
            'Generalization' : generalizationCtrl }
```

Por último analizaremos la **usabilidad** extrapolándola al número de pantallas que forman la aplicación. Una de las características de las aplicaciones actuales es su simplicidad para el usuario. Incluir gran cantidad de pantallería y opciones reduce su usabilidad al convertirla en poco intuitiva. Si queremos mejorar la experiencia de usuario conviene que nuestro programa tenga un número suficientemente bajo de pantallas para que resulte fácil de utilizar pero no sea excesivamente sencillo para que no pierda funcionalidad. Tras estudiar las aplicaciones existentes en el mercado, en este apartado se han definido los siguientes niveles dentro de la función *scoreMyApp.nScreens\_control*:

- Nivel alto: entre 3 y 10 pantallas distintas. Ejemplos: Twitter, Facebook, Instagram...
- Nivel medio: entre 1 y 3 pantallas. Ejemplos: Notas, Reloj, Calculadora
- Nivel bajo: una única pantalla o más de 10 pantallas.

Tras el análisis de los componentes, técnicas de programación y usabilidad para todas las pantallas que componen el programa a analizar, obtendremos el siguiente esquema de resultados recopilando la máxima información posible, como se puede ver en la figura 4.11.

### Envío de resultados

Volviendo a la vista encargada de obtener los resultados finales, *views.showUserAnalyzeProjectsPage*, tras el análisis de los ficheros que componen el programa, llamaremos a las funciones de la clase *scoreMyAppMessages*. En ella procesaremos toda la información y la mostraremos al usuario de forma intuitiva y simplificada a través de imágenes y mensajes de texto cortos.

Gracias a la función *scoreMyAppMessages.getScoreMsgs* obtendremos todos los datos necesarios que queremos mostrar al usuario a partir de los resultados vistos en el apartado anterior:

- Nivel global - *general\_score\_msg*: a partir de la puntuación general tendremos un mensaje que resumirá el nivel promedio de la aplicación.

```
# General Score message
if score[ 'Score' ] == 3:
    general_score_msg = 'Great job! Your app has a HIGH score!
    Check out the different skills to improve it even more'
elif score[ 'Score' ] == 2:
```

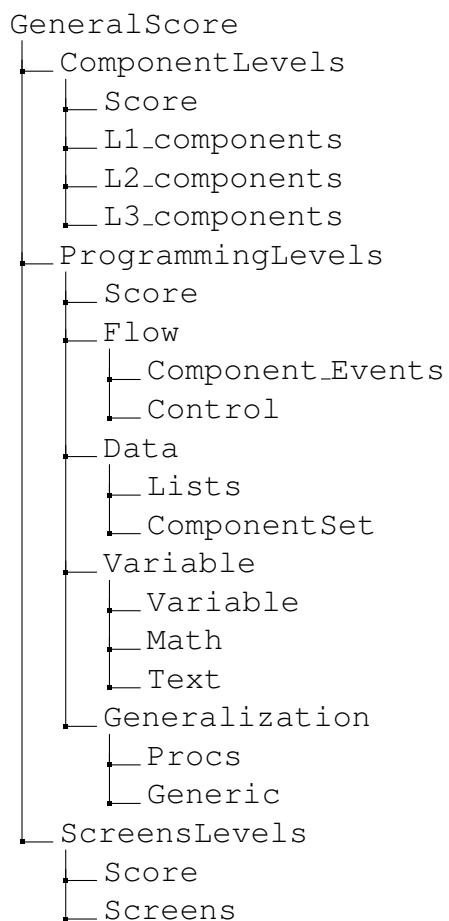


Figura 4.11: Resultados del análisis

```

    general_score_msg = 'Great job! Your app has a MEDIUM score!
Check out the different skills to reach the next level'
else:
    general_score_msg = 'Ups! Your app has a LOW score. Check out
the different skills to improve it'

```

- Nivel por categoría - *comp\_score\_msg, progr\_score\_msg, sched\_score\_msg*: componentes, programación y usabilidad mostrarán un mensaje personalizado en función de la puntuación media de sus subniveles.
- Desglose de resultados por categoría - *comp\_score\_info, progr\_score\_info, sched\_score\_info*: cada apartado incluirá la información más importante del mismo además de un consejo para mejorar el nivel actual. Dicho mensaje se mostrará siempre, pues aunque el usuario haya obtenido la máxima puntuación trataremos de motivarle para que incluya más complejidad a su programa o mayor diversidad de componentes.
- Componentes: listará los bloques utilizados para conseguir este nivel.

## COMPONENTS

*Due to the components used, you have a HIGH score. We challenge you to include more high level blocks*



To reach this score you have used these **high level** components:

• Storage: TinyWebDB

Unless you have the highest score, you can improve your app including components with medium or low level

Figura 4.12: Resultados de la categoría Componentes

- Programación: resumirá por categorías todos los identificadores añadidos al código junto con el número de veces que se utilizan de forma que el usuario pueda tener una visión global la estructura de su programa.

**PROGRAMMING**

*Your skills with the Built-in blocks have a HIGH score. We challenge you to improve even more the complexity of your app*



**VARIABLE CONTROL**

You have a **high** level in Variable Control due to the use of these blocks:

- Variable:
  - variable: 22
- Text:
  - text\_join: 2
- Math:
  - math\_subtract: 4
  - math\_compare: 2
  - math\_division: 4
  - math\_random\_int: 2
  - math\_add: 4
  - math\_number: 34

 You can add more of them in order to improve your skills

Figura 4.13: Resultados de la categoría Programación

- Usabilidad: muestra el número de pantallas que forman el programa.

**SCHEDULE**

*Your app usability has a HIGH score. Your number of screens is perfect*



You have included **3** screens in your app

 This schedule is perfect so ensure a good user experience

Figura 4.14: Resultados de la categoría Usabilidad

Una vez obtenidos todos los resultados, el siguiente paso será enviarlos al navegador del usuario a través de la página *analyzeAppCode.html*. En ésta se unen la mayoría de las tecnologías vistas en el capítulo Estado del Arte:

- HTML: como estándar para definir la estructura básica del archivo, legible e interpretable por el navegador. Identificado en la siguiente etiqueta dentro del propio fichero:

```
|||<!DOCTYPE html>
```

- CSS: antes de enviar el archivo HTML, parsearemos tanto los resultados del análisis como el estilo para integrarlos con el resto de valores de la página. Todos los formatos estarán definidos en diferentes ficheros .css almacenados estáticamente en el Servidor.

```
|||{ % load staticfiles %}
```

```
|||...
```

```
|||<link href="{ % static 'css/agency.min.css' %}" rel="stylesheet">
```

- JavaScript: gracias a este lenguaje de programación podemos implementar funciones dentro del HTML para que nuestra página sea más dinámica. Se utiliza por ejemplo, a la hora de ocultar o mostrar el menú:

```
|||...
```

```
// Highlight the top nav as scrolling occurs
$( 'body' ).scrollspy( {
    target: '.navbar-fixed-top',
    offset: 51
});

// Closes the Responsive Menu on Menu Item Click
$('.navbar-collapse ul li a').click( function() {
    $('.navbar-toggle:visible').click(); });

```

- Bootstrap: *framework* que aúna el resto de tecnologías (HTML, CSS, JavaScript) y nos permite que la página se adapte a dispositivos móviles como se muestra en la figura 4.5.

```
|||<!-- Bootstrap Core CSS -->
<link href="{ % static 'vendor/bootstrap/css/bootstrap.min.css' %}" rel="stylesheet">
```

||

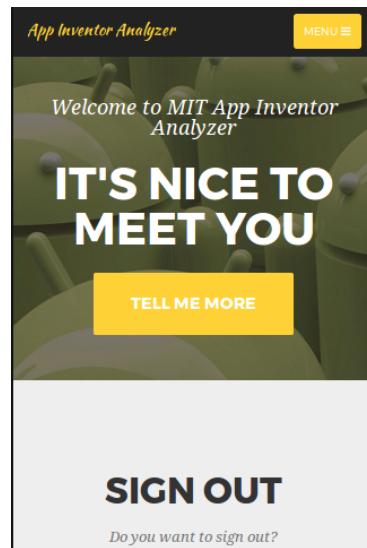


Figura 4.15: Versión móvil con Bootstrap

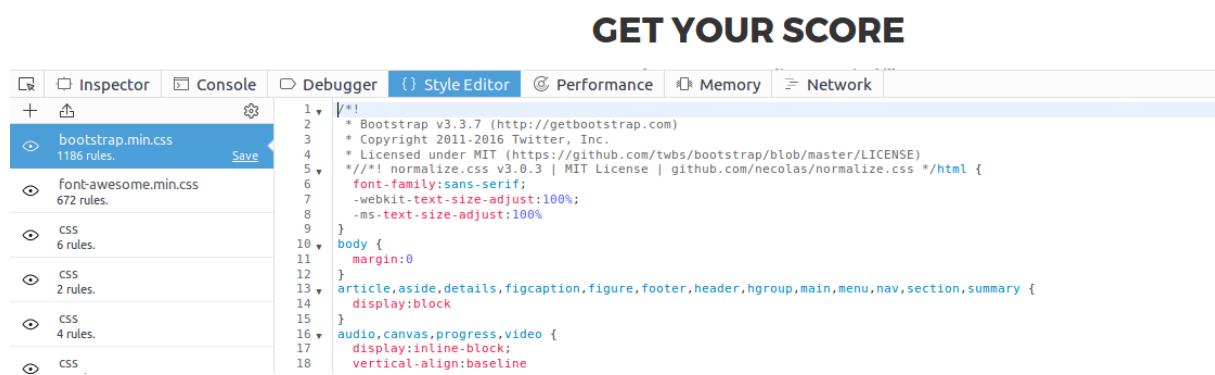


Figura 4.16: Petición de Bootstrap desde el navegador

## 4.2.2. Modelo de datos

En la organización y uso de la información se han utilizado dos modelos en función de las características de los datos a guardar:

- Contenido no persistente: variables locales creadas durante la ejecución de los procesos en el servidor.

- Contenido persistente: base de datos SQLite<sup>6</sup> y almacenamiento en memoria.

Como vimos en la sección anterior, los datos dinámicos donde se almacenan los resultados tras el análisis de código se guardan en una variable de tipo diccionario, donde se organizarán en pares clave-valor. En ella tendremos acceso tanto a las puntuaciones como a las prácticas de programación que las originan, ambas representadas en la figura 4.11.

Por otro lado tendremos el almacenamiento en memoria estática de los programas subidos por cada usuario. Para ello se ha creado un sistema de directorios donde ir descargando el código y así poder tener acceso a él en el futuro en caso de actualizaciones del sistema de análisis.

Los datos relativos a la información de usuario o los proyectos guardados por el mismo se guardarán en una base de datos, definida dentro de la aplicación en las clases de *models.py*. Para todo lo relativo a la creación y autenticación de usuarios, Django dispone de objetos predefinidos<sup>7</sup> que se han reutilizado añadiendo nuevas características. Para el caso del objeto User, que por defecto incluye como atributos primarios *username*, *password*, *email*, *first\_name* y *last\_name*, se ha usado de modelo base para la clase *UserProfile* extendiéndolo de forma que también incluya el campo *appinventorLogin*, necesario en la extracción de los ficheros comprimidos subidos por el usuario.

```
class UserProfile(models.Model):
    user = models.OneToOneField(User)

    # The additional attributes we wish to include.
    appinventorLogin = models.CharField(max_length=254)

    # Override the __unicode__() method to return out something meaningful!
    def __unicode__(self):
        return self.user.username
```

Para almacenar los usuarios existentes en la aplicación, sus proyectos y el contenido de los mismos se han creado las siguientes clases:

```
class Users(models.Model):
    userID = models.IntegerField(primary_key=True) # user_id
    projects = models.ManyToManyField('Projects') # The user can load and
                                                # analyze multiple projects
```

<sup>6</sup><https://www.sqlite.org>

<sup>7</sup><https://docs.djangoproject.com/en/1.11/topics/auth>

```

class Projects(models.Model):
    projectName = models.TextField(primary_key=True) # user.id_filename.
    name
    screens = models.ManyToManyField('Screens') # The project can include
    multiple screens
    projectProperties = models.TextField() # Project properties

class Screens(models.Model):
    scrID = models.TextField(primary_key=True) # user.id_filename.
    name_Screen number
    bky = models.TextField() # Blockly info
    scm = models.TextField() # Screen Description

```

De esta forma un usuario puede tener varios proyectos que a su vez estarán compuestos por una o varias pantallas asociadas:

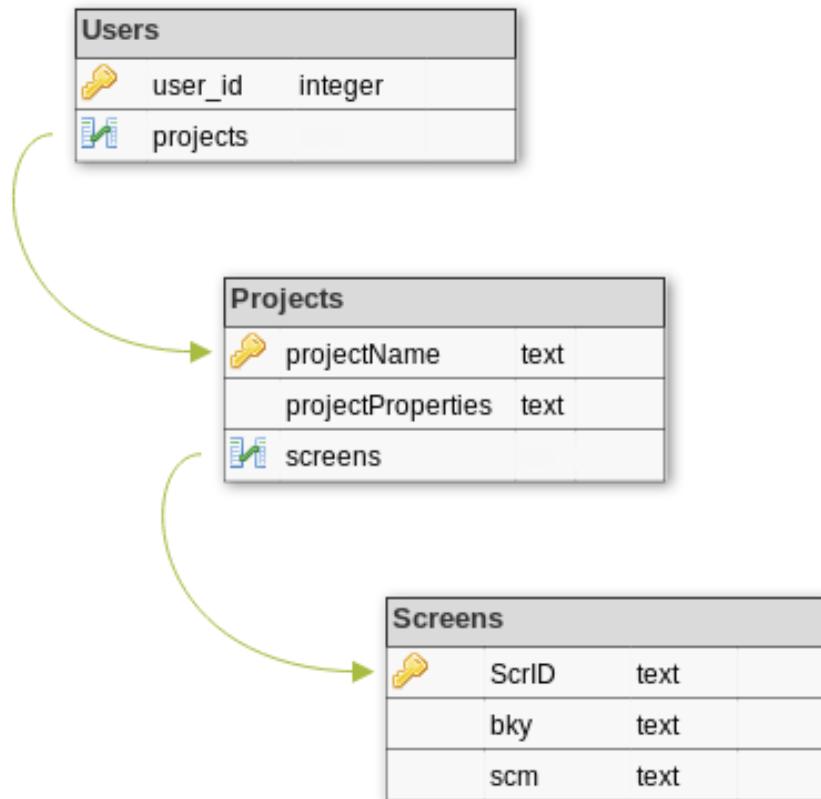


Figura 4.17: Modelo de datos

### 4.3. Manual de Usuario

Al entrar en la aplicación la primera vista que se muestra al usuario es la de Bienvenida. El resto de opciones e información de la página se podrán visualizar a través de los links situados en el menú superior o desplazándose hacia abajo.

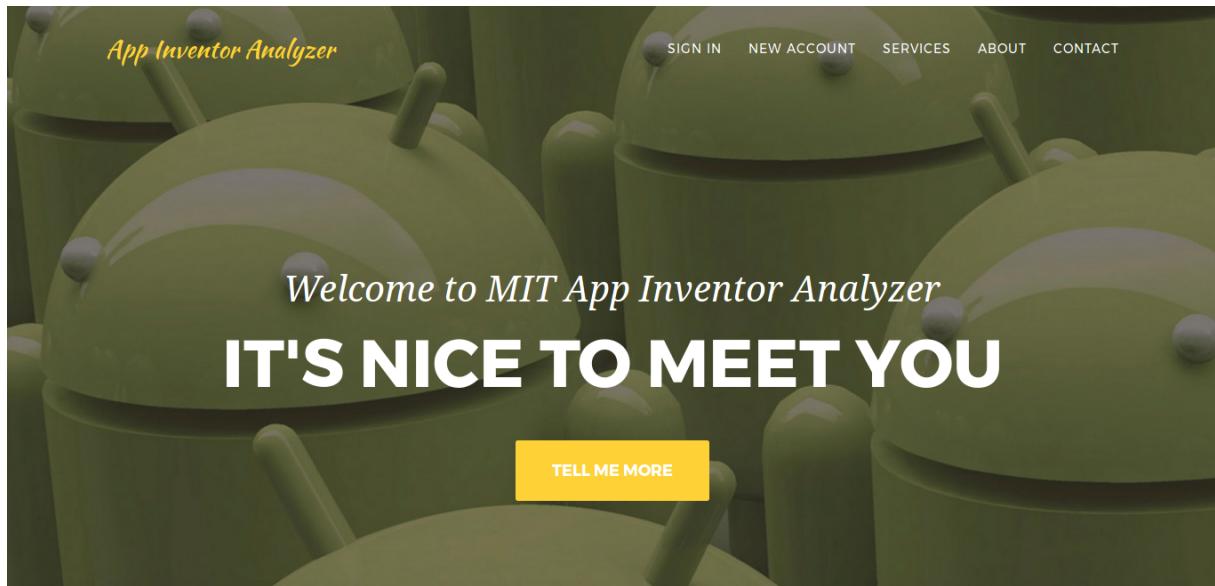


Figura 4.18: Pantalla de Bienvenida

En la pantalla de Inicio de Sesión el usuario podrá introducir su usuario y contraseña para poder acceder a su vista personal.

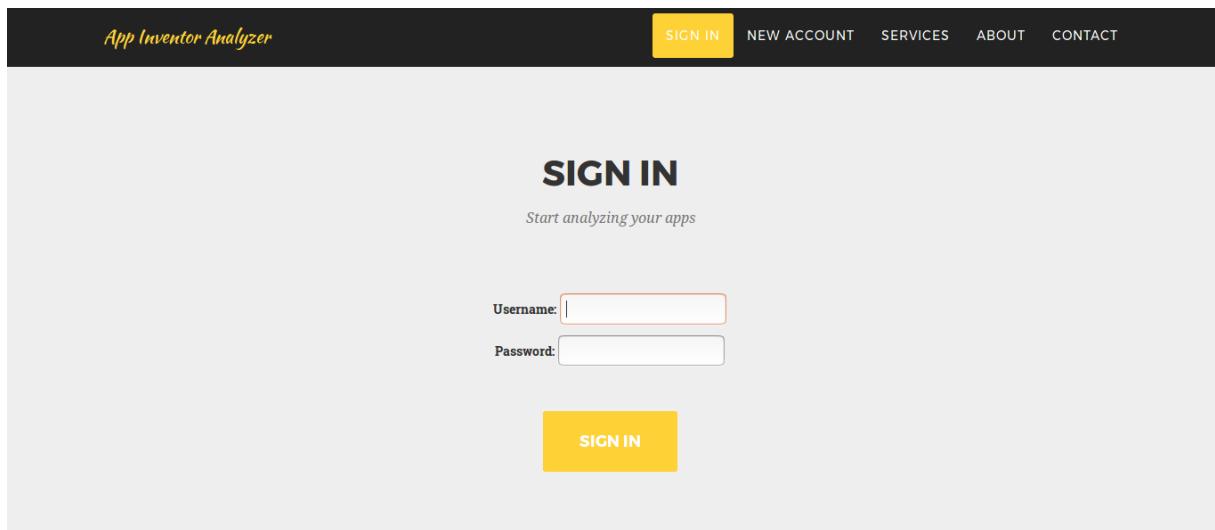


Figura 4.19: Pantalla de Inicio de Sesión

También se muestra información sobre los distintos aspectos que se estudian en la aplicación y un resumen de los pasos a seguir para analizar los proyectos:

The screenshot shows the 'Services' section of the App Inventor Analyzer. At the top, there's a navigation bar with links for SIGN IN, NEW ACCOUNT, SERVICES (which is highlighted in yellow), ABOUT, and CONTACT. Below the navigation, the title 'GET YOUR SCORE' is displayed in large, bold, black capital letters. A subtitle below it reads 'We analyze your app according to 3 main skills:'. Three circular icons represent these skills: 'Components' (yellow circle with a 3D cube icon), 'Programming' (yellow circle with a code editor icon), and 'Schedule' (yellow circle with a thumbs-up icon). Each skill has a brief description below it.

Component	Description
<b>Components</b>	Text labels, sensors, file servers... Depending on the components used, you'll get a level at this category. The most difficult the higher level
<b>Programming</b>	App Inventor include a high variety of blocks you can use in your app. How complex are the blocks you've added?
<b>Schedule</b>	Simple app or excessively complex for the users? Get the best schedule for your app with our tips

Figura 4.20: Pantalla Servicios

The screenshot shows the 'ABOUT' section of the App Inventor Analyzer. At the top, there's a navigation bar with links for SIGN IN, NEW ACCOUNT, SERVICES (highlighted in yellow), ABOUT, and CONTACT. Below the navigation, the title 'HOW IT WORKS' is displayed in large, bold, black capital letters. A subtitle below it reads 'Follow these easy steps to analyze your app'. The process is divided into four steps, each illustrated with a green Android robot icon in a circle:

- 1ST STEP...**  
**Download your app**  
Once you have your app in App Inventor export the .aia file to your computer
- 2ND STEP...**  
**Enter your account**  
Have you already created your account? Sign in or create a new one
- 3RD STEP...**  
**Upload your app**  
Once you are in your user's page, upload your new app to analyze it. Do you want to re-analyze an existing project? Skip this step :)
- 4RD STEP...**  
**See your apps**  
You can check apps you've previously uploaded. Go back and analyze them as many times as you want

At the bottom of the flowchart is a yellow circle containing the text 'IMPROVE YOUR APPS!'

Figura 4.21: Pantalla 'Acerca de'

Si queremos utilizar My App Inventor y aún no nos hemos registrado, el primer paso será crear una cuenta de usuario. Para ello iremos a la opción ‘New Account’ en el menú principal (Figura 4.22), lo que abrirá una nueva pestaña en el navegador:

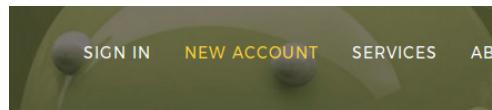


Figura 4.22: Link para crear una nueva cuenta

Esta vista incluye todos los datos necesarios para crear una cuenta: nombre, email, contraseña. . . Será necesario especificar también el nick de usuario en App Inventor para poder leer correctamente los ficheros comprimidos.

The screenshot shows a 'CREATE ACCOUNT' form. At the top, it says 'Start using the analyzer'. Below that, there are several input fields with validation messages:

- Username:** jmmurillo • This field is required.
- First name:** Jose Manuel • This field is required.
- Last name:** Murillo • This field is required.
- Email:** jm.murillo@email.com • This field is required.
- Password1:** ..... • This field is required.
- Password2:** ..... • This field is required.
- Appinventorlogin:** myappinventorurjc

At the bottom is a large yellow button labeled 'CREATE ACCOUNT'.

Figura 4.23: Creación de cuenta de usuario

Si el usuario ya está registrado o las contraseñas no coinciden, la aplicación tiene implementadas una serie de validaciones que muestran un mensaje de error antes de crear la cuenta.

The screenshot shows a registration form with the following fields and errors:

- Username:** jmmurillo (no error)
- First name:** Jose Manuel (no error)
- Last name:** Murillo (no error)
- Email:** jm.murillo@email.com (no error)
- Password:** (highlighted with a red border, indicating an error)
- Password2:** (highlighted with a red border, indicating an error)

A yellow header bar at the top displays the error message: "Passwords don't match".

Figura 4.24: Mensaje de error por contraseña inválida

También se informa si el registro ha finalizado correctamente:

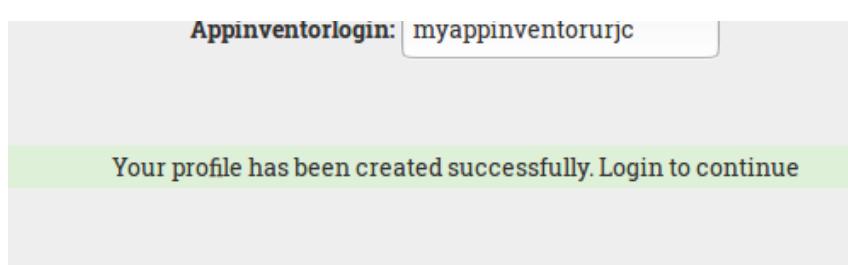


Figura 4.25: Mensaje de confirmación al registrarse

Volviendo a la página principal, iniciaremos sesión en el apartado ‘Sign In’, lo que redirigirá a la vista de la figura 4.26. Nada más entrar nos muestra tres opciones: ver proyectos guardados, subir un nuevo proyecto o modificar los datos de nuestra cuenta. Tras seleccionar una de ellas la página ofrecerá el nuevo contenido (gracias a la tecnología Ajax podemos actualizar asíncronamente esta sección de la web).

## WHAT DO YOU WANT TO DO?

Select an option by clicking the icon



**My Projects**

View your uploaded projects



**New Project**

Upload a new project from your computer



**My Profile**

Update your profile settings



Figura 4.26: Opciones disponibles en vista de usuario

Antes de comenzar a analizar un proyecto, veamos brevemente cómo actualizar nuestros datos de usuario. Seleccionando la opción ‘My Profile’ tendremos acceso a la información susceptible de ser modificada.

## YOUR PROFILE

Modify and update your data

First name:	<input type="text" value="Jose Manuel"/>
Last name:	<input type="text" value="Murillo"/>
Email address:	<input type="text" value="jm.murillo@email.com"/>
AppinventorLogin:	<input type="text" value="myappinventorurjc"/>



**SAVE**

Figura 4.27: Actualización de datos de usuario

Una vez realizado el cambio, nos aparecerá un mensaje de confirmación en la página principal.

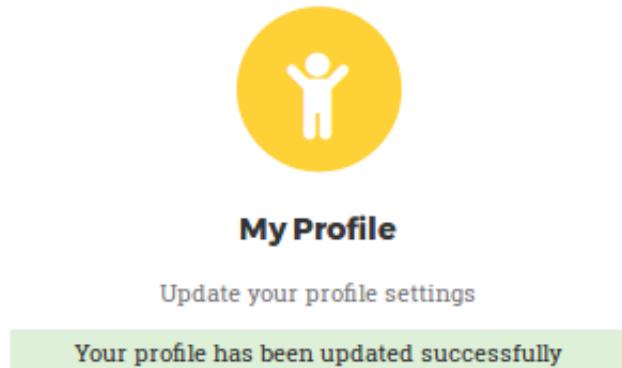


Figura 4.28: Mensaje de confirmación de actualización de datos de usuario

Ahora que ya tenemos una cuenta, podemos subir nuestro primer proyecto. Para ello debemos ir a la opción ‘New Project’ y buscaremos en nuestro disco el fichero .aia previamente descargado de nuestra cuenta de App Inventor. Pulsamos ‘Load and Analyze’ y ¡comienza el análisis!



Figura 4.29: Añadir un nuevo proyecto

La aplicación nos redirigirá en segundos a una nueva vista, donde nos muestra la valoración

general de los principales aspectos analizados.

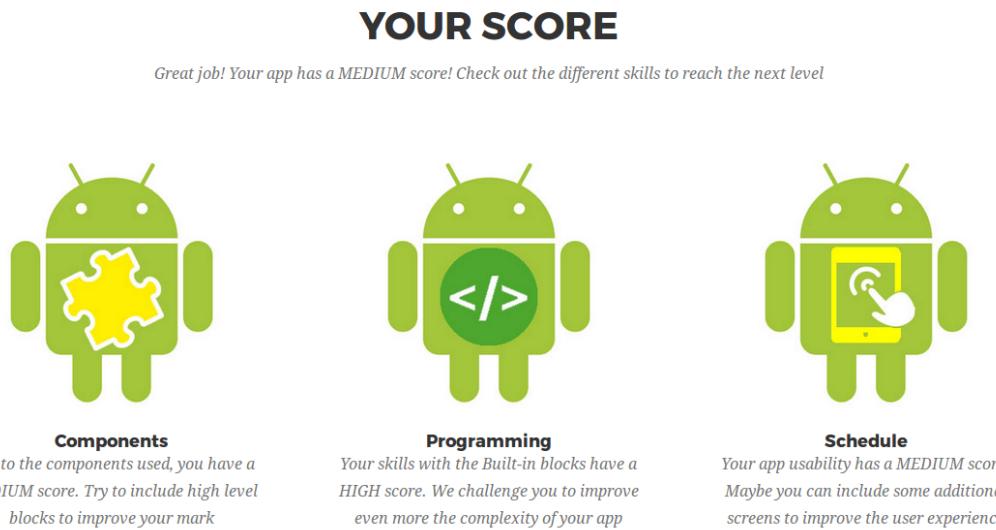


Figura 4.30: Resultados del análisis de nuestro proyecto

Si queremos ver el desglose de cada uno de ellos, simplemente haremos *click* sobre cada opción para abrir una ventana emergente que mostrará la información desglosada como vimos en la figura 4.9.



Figura 4.31: Ver desglose del análisis por categoría

En caso de querer consultar el análisis de un proyecto que ya tenemos guardado con anterioridad, tendremos que ir a la opción ‘My Projects’ del menú. En ella se listan todos los proyectos subidos por el usuario. Seleccionando uno de ellos y pulsando el botón ‘Lets Analyze’ se mostrará de nuevo la información de la figura 4.30.

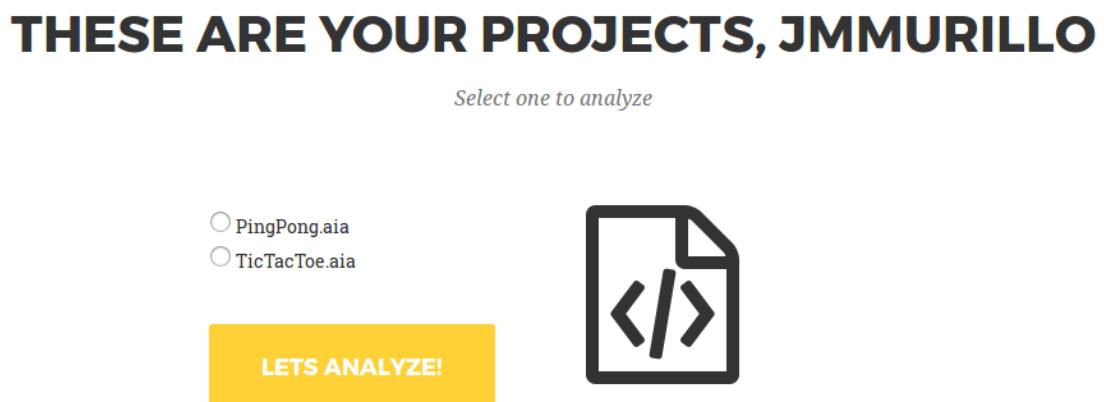


Figura 4.32: Lista de proyectos guardados

Para finalizar sesión y volver a la página general, basta con pulsar el botón ‘Sign out’ situado en el menú superior y la aplicación nos redirigirá automáticamente a la vista principal como se muestra en la figura 4.34.

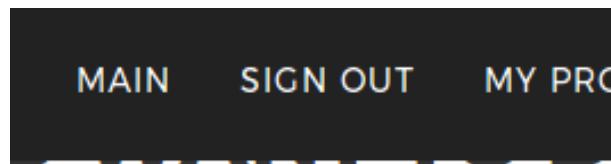


Figura 4.33: Botón para salir de sesión

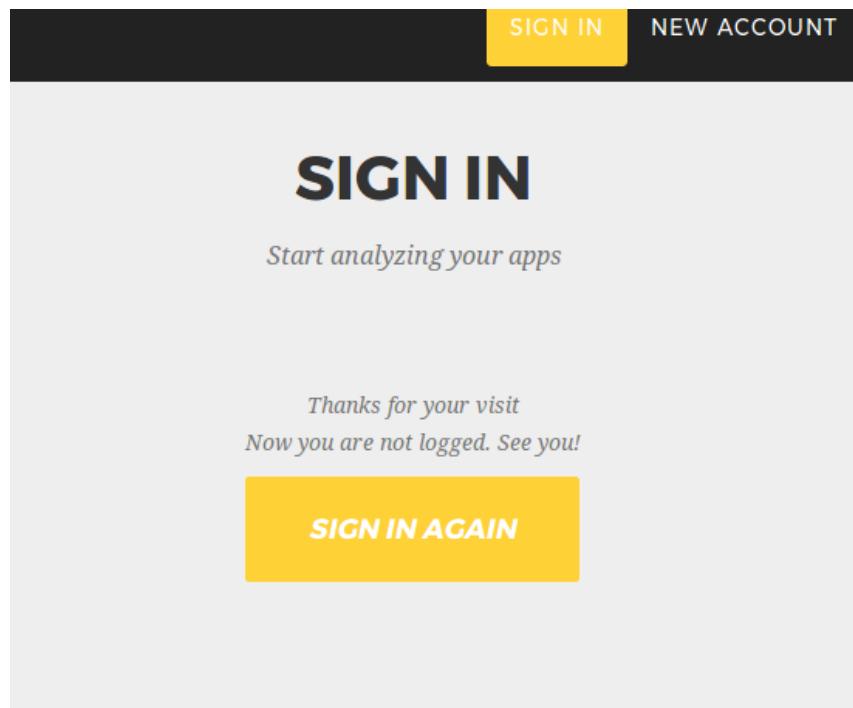


Figura 4.34: Mensaje de confirmación de cierre de sesión

# Capítulo 5

## Resultados

A continuación vamos a analizar tres proyectos reales pertenecientes a la galería de App Inventor para ver la salida que obtenemos con el analizador.

### 5.1. Nivel Alto: Multifunctional Application

*Multifunctional Application* es un programa que incluye control de acceso, uso de eventos para modificar elementos en múltiples pantallas y uso de la base de datos TinyWebDB para guardar información.

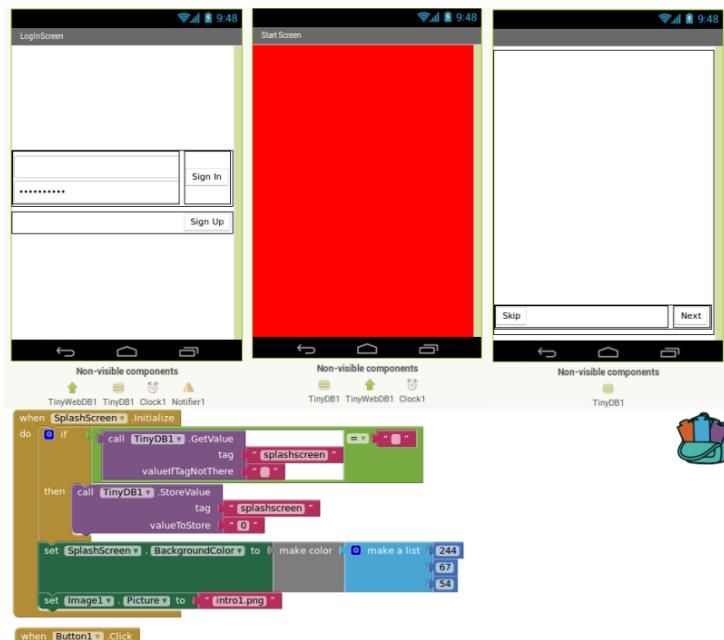


Figura 5.1: Multifunctional Application: diseño

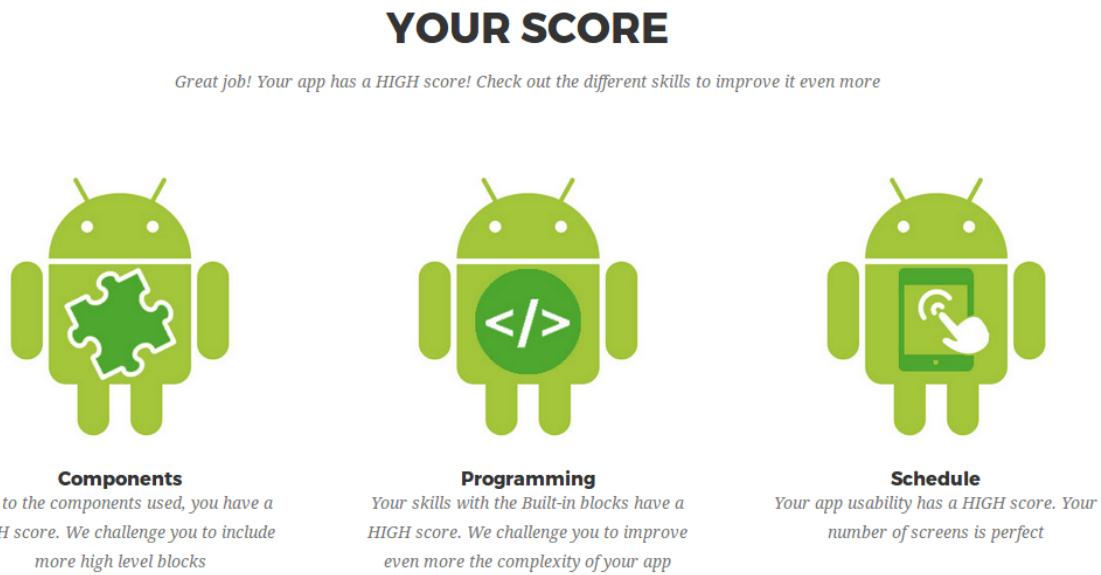


Figura 5.2: Multifunctional Application: clasificación general

Si analizamos el proyecto, My App Inventor le da una clasificación general Alta con el siguiente desglose de criterios:

- Nivel Alto en Componentes: aunque incluye otro tipo de componentes, es el uso del almacenamiento con TinyWebDB el que le otorga cierta complejidad.

To reach this score you have used these high level components:

- Storage : TinyWebDB

Tip: Unless you have the highest score , you can improve your app including components with medium or low level

- Nivel Alto en Programación: control de variables, gestión de datos y control de flujo consiguen una clasificación alta, mientras que el uso de funciones y la ausencia de procedimientos generales hace que en generalización tenga un nivel medio.

#### VARIABLE CONTROL

You have a high level in Variable Control due to the use of these blocks :

- Variable :
  - variable : 120
- Text :
  - text\_join : 8

- text\_trim: 20
- text\_isEmpty: 20
- Math:
  - math\_number: 16

Tip: You can add more of them in order to improve your skills

#### GENERALIZATION

You have a medium level in Generalization due to the use of these blocks:

- Procedures:
  - procedures\_callnoreturn: 8
  - procedures\_defnoreturn: 4
- Generalization: 0

Tip: Try to include both categories to improve your level

#### DATA MANAGEMENT

You have a high level in Data Management due to the use of these blocks:

- Component Set:
  - component\_set\_get: 212
- Lists:
  - lists\_create\_with: 20
  - lists\_select\_item: 16
  - lists\_is\_in: 16
  - lists\_is\_list: 8
  - lists\_add\_items: 8

Tip: You can add more of them in order to improve your skills

#### FLOW CONTROL

You have a high level in Flow Control due to the use of these blocks:

- Control:
  - controls\_if: 44
- Component\_Events:
  - component\_event: 28

Tip: You can add more of them in order to improve your skills

- Nivel Alto en Usabilidad: la inclusión de múltiples pantallas le otorga una puntuación alta.

You have included 3 screens in your app

Tip: This schedule is perfect so ensure a good user experience

## 5.2. Nivel Medio: Android Mash

*Android Mash* es un juego sencillo que interactúa con el usuario a través de la pantalla táctil. Forma parte de los tutoriales incluidos en App Inventor para enseñar a incluir elementos básicos como una imagen de fondo, un sensor táctil, temporizador y comportamiento aleatorio. Como vemos en la figura 5.3, está compuesto por una pantalla y pocos componentes que se relacionan a través de bloques condicionales y una función que genera números aleatorios.

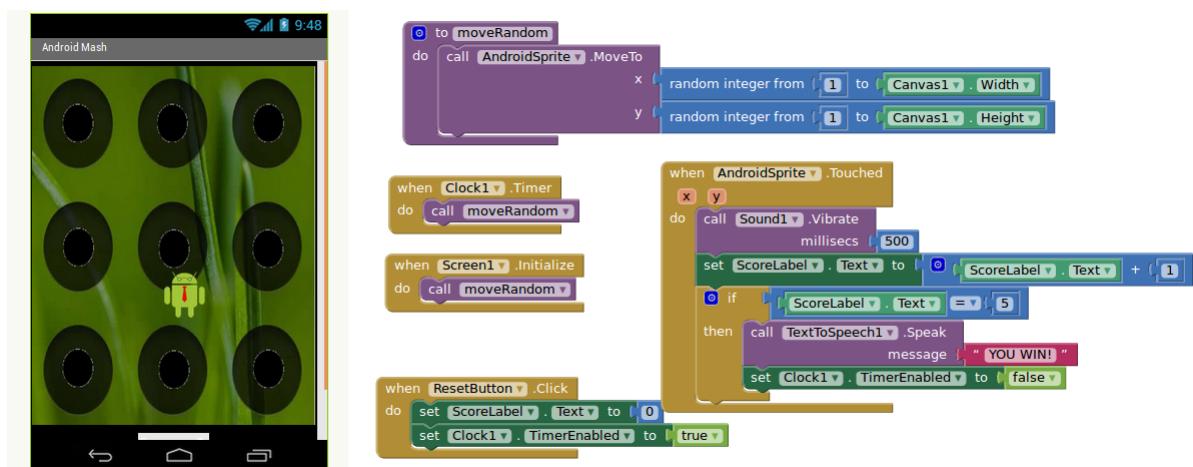


Figura 5.3: Android Mash: diseño

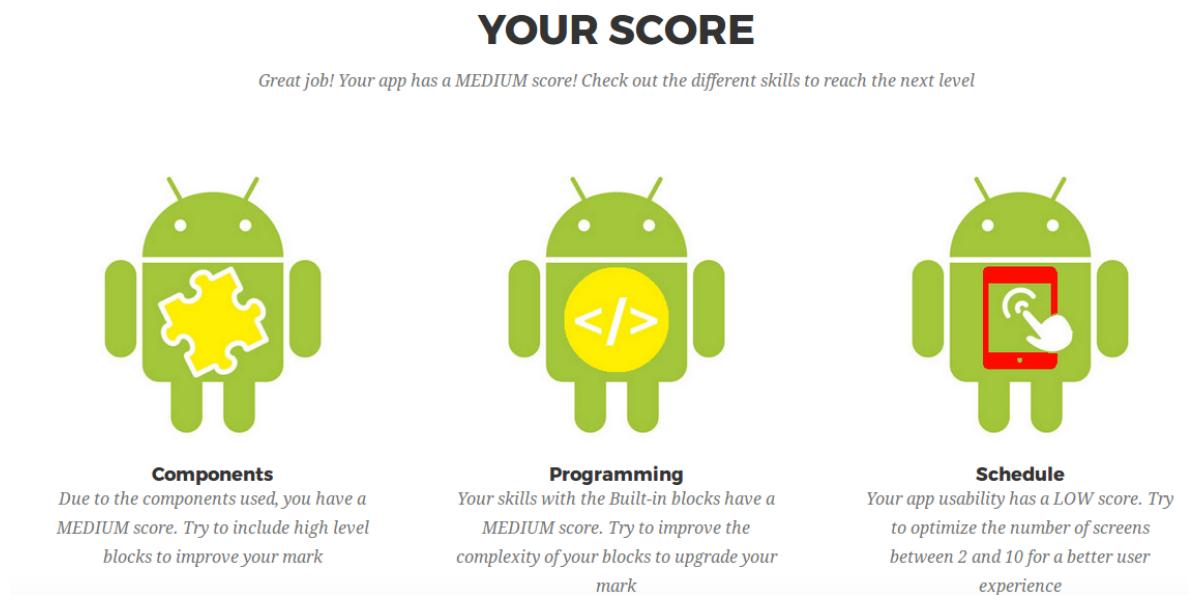


Figura 5.4: Android Mash: clasificación general

El análisis de este proyecto nos da un nivel Medio general y los siguientes valores en el resto de criterios:

- Nivel Medio en Componentes: aunque se trata de una interfaz sencilla, incluye elementos como sonidos y lectura de textos con voz lo que le dota de cierta complejidad a la aplicación.

To reach this score you have used these medium level components :

- Media: Sound ,TextToSpeech

Tip: Try to include highest level blocks to improve your mark .

- Nivel Medio en Programación: en criterios como control de variables, generalización y gestión de datos tiene un nivel medio ya que incluye al menos uno de los identificadores de bloque necesarios. En control de flujo obtiene un nivel alto ya que utiliza tanto expresiones lógicas como dependencia de eventos.

#### VARIABLE CONTROL

You have a medium level in Variable Control due to the use of these blocks :

- Variable :

  - variable : 0

- Text: 0

– Math :

```
math_random_int: 2
math_add: 1
math_compare: 1
math_number: 6
```

Tip: Try to include more categories to improve your level

#### GENERALIZATION

You have a medium level in Generalization due to the use of these blocks :

– Procedures :

- procedures\_callnoreturn: 2
- procedures\_defnoreturn: 1

– Generalization :

- generic: 0

Tip: Try to include both categories to improve your level

#### DATA MANAGEMENT

You have a medium level in Data management due to the use of these blocks :

– Component Set :

- component\_set\_get: 8

– Lists : 0

Tip: Try to include both categories to improve your level

#### FLOW CONTROL

You have a high level in Flow Control due to the use of these blocks :

– Control :

- controls\_if: 1

– Component\_Events :

- component\_event: 4

Tip: You can add more of them in order to improve your skills

■ Nivel Bajo en Usabilidad: por el uso de una sola pantalla.

You have included 1 screen in your app

Tip: With this schedule you have a simple app. Why not improve its design by adding one or two screens more?

### 5.3. Nivel Bajo: YOUB Analogic Clock

*YOUB Analogic Clock* se trata de una aplicación sencilla donde se muestra un reloj analógico. Internamente está compuesto por componentes de tipo imagen que se relacionan a través de sencillos bloques de modificación de variables.

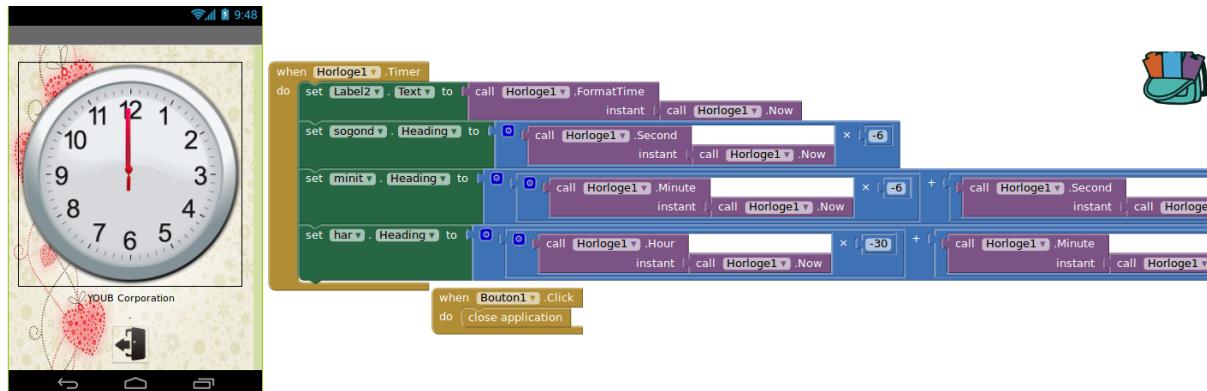


Figura 5.5: Analogic Clock: diseño

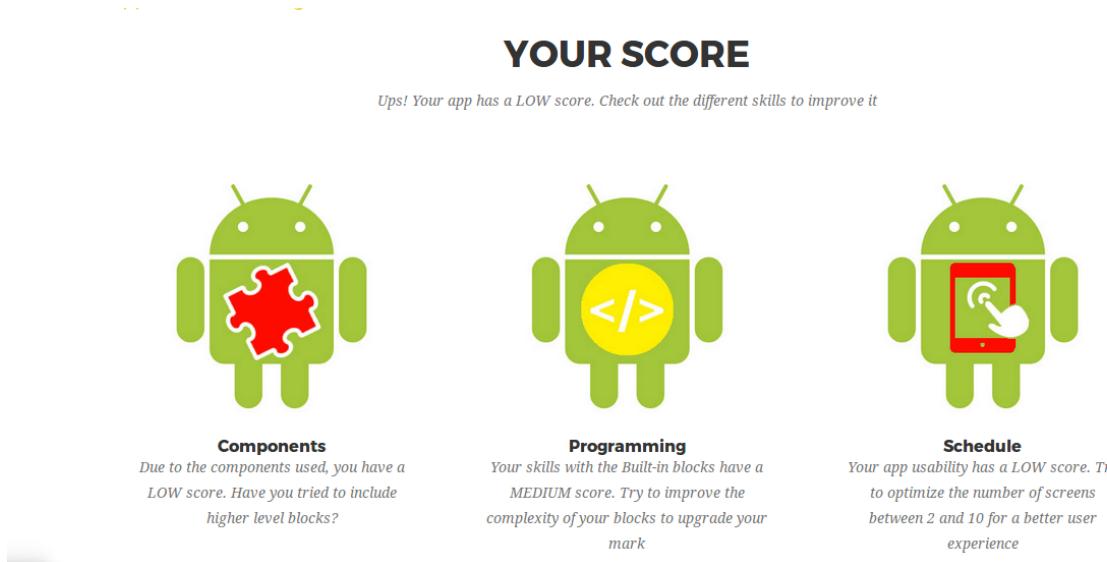


Figura 5.6: Analogic Clock: clasificación general

Tras analizar el proyecto, My App Inventor le da una clasificación general Baja con el siguiente desglose de criterios:

- Nivel Bajo en Componentes: utiliza elementos sencillos como imágenes, un reloj y botones.

To reach this score you have used these low level components:

- UserInterface: Button , Label
- Drawing: ImageSprite
- Sensors: Clock

Tip: Try to include high and medium level blocks to improve your mark.

- Nivel Medio en Programación: control de variables y gestión de datos tienen un nivel medio, generalización, un nivel bajo por la ausencia de procedimientos y control de flujo, alto por el uso de controles y eventos.

#### VARIABLE CONTROL

You have a medium level in Variable Control due to the use of these blocks :

- Variable :0
- Text: 0
- Math:
  - math\_division: 2
  - math\_add: 2
  - math\_multiply: 3
  - math\_number: 5

Tip: Try to include more categories to improve your level

#### GENERALIZATION

You have a low level in Generalization due to the use of these blocks :

- Procedures: 0
- Generalization: 0

Tip: Try to include at least one of the categories to improve your level

#### DATA MANAGEMENT

You have a medium level in Data management due to the use of these blocks :

- Component Set:
  - component\_set\_get: 4
- Lists: 0

Tip: Try to include both categories to improve your level

#### FLOW CONTROL

You have a high level in Flow Control due to the use of these blocks:

- Control:
  - controls\_closeApplication: 1
- Component\_Events:
  - component\_event: 2

Tip: You can add more of them in order to improve your skills

- Nivel Bajo en Usabilidad: sólo se incluye una pantalla.

You have included 1 screen in your app

Tip: With this schedule you have a simple app. Why not improve its design by adding one or two screens more?



# **Capítulo 6**

## **Conclusiones**

En este capítulo se analizarán los objetivos planteados al inicio del proyecto, viendo cuáles se han cumplido y cuáles no. También haremos un repaso de los conocimientos adquiridos durante la carrera que me han sido de utilidad para realizar este proyecto y qué he aprendido durante su creación. Por último, plantearemos las posibilidades de desarrollo de My App Inventor de cara a futuros proyectos.

### **6.1. Consecución de objetivos**

El objetivo general del proyecto fue crear una plataforma donde poder evaluar los puntos fuertes y débiles de una aplicación creada en App Inventor. Este punto se ha cumplido con la creación de una web en la que el usuario, tras subir su código comprimido, puede ver el resultado del análisis y guardar su programa para futuras referencias. Además se han unido distintas tecnologías (Bootstrap, Ajax) para que la página sea más dinámica y siga las tendencias actuales.

A continuación, veremos algunos obstáculos que fui encontrándome durante el desarrollo del proyecto y cómo los fui solventando:

- La web de App Inventor requiere un registro para entrar y poder utilizar la aplicación. Dentro de la vista de usuario, tiene almacenados los proyectos guardados y en caso de querer descargar aplicaciones de otras personas, debemos antes duplicarlas en nuestro directorio. Al no existir una galería pública a la que acceder por una url simple, tuve que

modificar el enfoque de Dr. Scratch y hacer que los usuarios sólo dispongan de la opción de subir un fichero comprimido desde su disco local.

- Otro de los puntos que frenó el avance en la definición del algoritmo de análisis es el amplio abanico de opciones que podemos incluir en App Inventor y su escasa documentación. Al no encontrar una lista de posibilidades, opté por incluir todos los tipos de bloque en un programa y analizar el fichero resultante, estudiando distintas combinaciones y modificaciones.
- Al no existir un *plugin* como Hairball para el análisis del código de cada programa, al igual que en el punto anterior, desglosé los ficheros de código, creé un árbol con la información del XML y saqué patrones para posteriormente crear funciones como *countBlocks* (contabiliza las etiquetas solicitadas), *lookForBlocks* (busca bloques específicos) o *listOfTypes* (crea una lista con los atributos buscados)

## 6.2. Aplicación de lo aprendido

Si tuviera que destacar algún punto de la carrera de Telecomunicaciones sería la capacidad de razonar que me han aportado todas las asignaturas de programación. A parte de aprender a programar en distintos lenguajes, lo que te da una visión global, valoro el desarrollo de mi habilidad lógica y de razonamiento. Ambas han sido de gran ayuda en el ámbito laboral y por eso me gustaría poner mi granito de arena en dar a conocer iniciativas que fomentan aficiones positivas con alto valor añadido entre los jóvenes.

Asignaturas como **Fundamentos de la Programación**, donde se asentaron las bases de todo lo que sé actualmente sobre programación y aprendí los conceptos básicos (condiciones, variables, bucles, funciones...), o **Sistemas Telemáticos** en el que nos enseñaron la arquitectura Cliente-Servidor con sus comunicaciones, han sido importantes para llegar a comprender la estructura de un programa informático. Aunque si hay una asignatura clave que me haya empujado a realizar este proyecto ha sido **Servicios y Aplicaciones Telemáticas**, donde conocí el lenguaje de programación Python y obtuve los conocimientos necesarios para crear una página web más elaborada con tecnologías como CSS o Ajax.

## 6.3. Lecciones aprendidas

Las lecciones aprendidas durante este proyecto han sido muchísimas, pero si tuviera que destacar alguna sería:

- Hace unos meses aplicaciones como Scratch o App Inventor eran desconocidas para mí. Ahora que las conozco un poco más las recomiendo encarecidamente no sólo como plataformas para que los niños aprendan a programar, sino también los adultos, nunca es tarde para empezar nuevos retos.
- He podido profundizar y afianzar mis conocimientos sobre Python, Django y CSS.
- Al buscar la mejora de la presentación web, he aprendido a utilizar tecnologías como Bootstrap o AJAX.
- Dentro de la recopilación de información para incluir componentes a la página, he descubierto webs muy interesantes. Por ejemplo *Font Awesome*<sup>1</sup>, un servidor gratuito de iconos en formato vector muy útil a la hora de decorar mensajes informativos en la página web ya que se formatean dentro del CSS.

## 6.4. Trabajos futuros

My App Inventor abre una nueva línea de desarrollo para futuros proyectos. Algunas ideas para hacer crecer la aplicación serían:

- Añadir un componente social en el que poder comparar mi puntuación con la de otros usuarios.
- Incluir gamificación con un sistema de recompensas si el usuario consigue ciertos logros en sus programas.
- Creación de cuentas para profesores de cara a la utilización en clases de informática y como método de apoyo en la valoración
- Implementación de mecanismos de seguridad adicionales a los proporcionados por Django

---

<sup>1</sup><http://fontawesome.io>



# Bibliografía

[1] App Inventor.

[http://appinventor.mit.edu/explore/.](http://appinventor.mit.edu/explore/)

[2] Bootstrap.

[http://getbootstrap.com/.](http://getbootstrap.com/)

[3] Documentación Django.

[https://www.djangoproject.com/.](https://www.djangoproject.com/)

[4] Documentación Python.

[https://www.python.org/.](https://www.python.org/)

[5] Plantilla Bootstrap.

[https://startbootstrap.com/.](https://startbootstrap.com/)

[6] Sqlite.

[http://www.sqlite.org/.](http://www.sqlite.org/)

[7] W3schools.

[https://www.w3schools.com/sql/.](https://www.w3schools.com/sql/)

[8] World Wide Web Consortium (W3C).

<https://www.w3.org>.

[9] M. S. Derek Walter. *Learning MIT App Inventor: A Hands-On Guide to Building Your Own Android Apps*. Addison-Wesley Professional, 2014.