



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



TFG del Grado en Ingeniería Informática
GoBees
Monitorización del estado de una
colmena mediante la cámara de un
smartphone.



Presentado por David Miguel Lozano
en la Universidad de Burgos — 7 de febrero de 2017
Tutores: Dr. José Francisco Díez Pastor
y Dr. Raúl Marticorena Sánchez

Índice general

Índice general	I
Índice de figuras	III
Apéndice A Manuales	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	1
Apéndice B Especificación de Requisitos	2
B.1. Introducción	2
B.2. Objetivos generales	2
B.3. Catalogo de requisitos	2
B.4. Especificación de requisitos	2
Apéndice C Especificación de diseño	3
C.1. Introducción	3
C.2. Diseño de datos	3
C.3. Diseño arquitectónico	5
C.4. Diseño procedimental	13
C.5. Diseño de interfaces	15
Apéndice D Documentación técnica de programación	18
D.1. Introducción	18
D.2. Estructura de directorios	18
D.3. Manual del programador	19
D.4. Pruebas del sistema	33
Apéndice E Documentación de usuario	39
E.1. Introducción	39

ÍNDICE GENERAL

II

E.2. Requisitos de usuarios	39
E.3. Instalación	39
E.4. Manual de usuario	41
Bibliografía	53

Índice de figuras

C.1. Diagrama E/R.	4
C.2. Diagrama relacional.	4
C.3. Patrón MVP.	5
C.4. Patrón repositorio.	6
C.5. Arquitectura de la aplicación.	7
C.6. Diagrama de paquetes simplificado.	8
C.7. Paquetes de las diferentes características.	8
C.8. Paquete tipo de una característica.	9
C.9. Diagrama de clases general.	10
C.10. Paquete <i>monitoring</i>	10
C.11. Diagrama de clases del paquete <i>camera</i>	11
C.12. Diagrama de clases del paquete <i>algorithm</i>	11
C.13. Paquete <i>data</i>	12
C.14. Paquete <i>model</i>	12
C.15. Diagrama de clases del paquete <i>source</i>	13
C.16. Diagrama de secuencia del algoritmo.	14
C.17. Prototipos iniciales.	15
C.18. Diseños finales de las interfaces.	16
C.19. Diagrama de navegabilidad.	17
C.20. Paleta de colores.	17
D.1. Android Studio.	20
D.2. Terminal Git Bash.	20
D.3. Variable del sistema de OpenCV.	21
D.4. Clonar repositorio de GitHub.	22
D.5. Importar proyecto en Android Studio.	22
D.6. Dependencias del proyecto.	24
D.7. Compilar proyecto.	25
D.8. TravisCI.	28

D.9. Codecov	29
D.10. CodeClimate.	30
D.11. Configuración de SonarQube en Gradle.	30
D.12. SonarQube.	31
D.13. VersionEye.	32
D.14. Página web generada con ReadTheDocs.	33
D.15. Test unitarios.	34
D.16. Ejecución de los test unitarios.	34
D.17. Ejecución del test de integración del algoritmo.	35
D.18. Aplicación de etiquetado de fotogramas.	36
D.19. Plataforma de desarrollo del algoritmo.	37
E.1. GoBees en Google Play.	40
E.2. Instalación desde Google Play	41
E.3. Colmenar de muestra.	42
E.4. Añadir colmenar.	43
E.5. Información meteorológica.	45
E.6. Colocación del <i>smartphone</i> en la colmena.	47
E.7. Ajustes de monitorización.	49
E.8. Detalle de la grabación.	50
E.9. Sobre GoBees.	52

Apéndice A

Manuales

A.1. Introducción

A.2. Planificación temporal

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

- B.1. Introducción
- B.2. Objetivos generales
- B.3. Catalogo de requisitos
- B.4. Especificación de requisitos

Apéndice C

Especificación de diseño

C.1. Introducción

En este anexo se define cómo se han resuelto los objetivos y especificaciones expuestos con anterioridad. Define los datos que va a manejar la aplicación, su arquitectura, el diseño de sus interfaces, sus detalles procedimentales, etc.

C.2. Diseño de datos

La aplicación cuenta con las siguientes entidades:

- **Colmenar (Apiary)**: tiene un nombre, una imagen, una localización y unas notas. A su vez, guarda un registro del tiempo meteorológico actual y varios registros del tiempo que hacia cuando se realizaron las grabaciones de sus colmenas.
- **Colmena (Hive)**: tiene un nombre, una imagen y unas notas. A su vez, posee varias grabaciones de distintas monitorizaciones de la colmena.
- **Registro (Record)**: se corresponde a la salida del algoritmo de conteo al analizar un fotograma. Tiene un *timestamp* y el número de abejas que había en el fotograma.
- **Registro meteorológico (MeteoRecord)**: guarda información sobre el estado meteorológico en una localización y un momento dado. Tiene un *timestamp*, la localidad, el código correspondiente a la condición meteorológica, el icono correspondiente, temperatura, presión, humedad, velocidad y dirección del viento, porcentaje de nubes, precipitaciones, y nieve.

Diagrama E/R

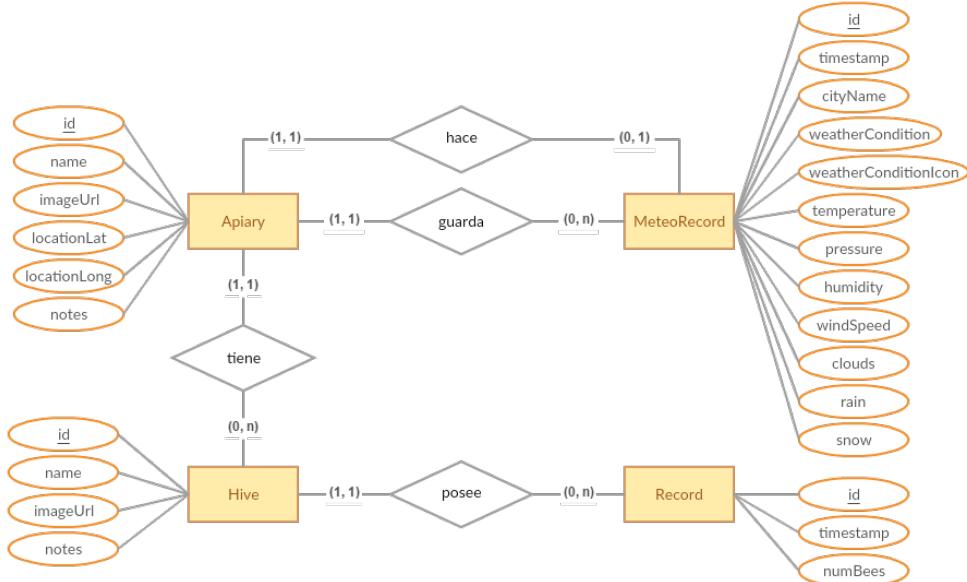


Figura C.1: Diagrama E/R.

Diagrama Relacional

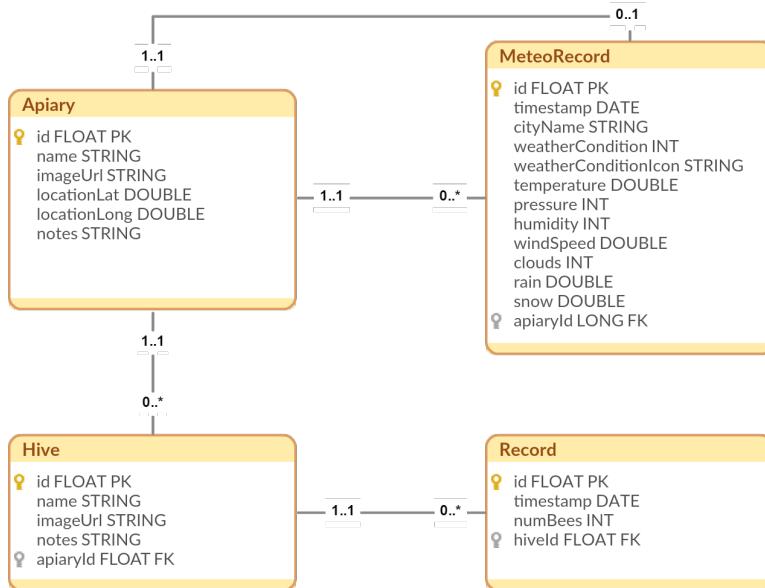


Figura C.2: Diagrama relacional.

C.3. Diseño arquitectónico

El hecho de que el proyecto se haya realizado para la plataforma Android ha condicionado muchas de las decisiones de diseño. Aun así, se han aplicado una serie de patrones para intentar desacoplar el código lo máximo posible y así mejorar su testabilidad y mantenibilidad.

Model-View-Presenter (MVP)

Uno de los patrones arquitectónicos que más relevancia está ganando para el desarrollo de aplicaciones es MVP (*Model-View-Presenter*). Se trata de un patrón derivado del MVC (*Model-View-Controller*) cuyo objetivo es separar la vista del modelo de datos subyacente. MVP introduce la figura del *presenter* que actúa de mediador entre estas dos capas. Su segundo objetivo es maximizar la cantidad de código que se puede testear de forma automática.

MVP divide la aplicación en las siguientes capas:[6]

- *Model*: se corresponde únicamente con el acceso a datos. Se encarga de almacenar y proporcionar los diferentes datos que maneja la aplicación. En nuestra aplicación se corresponde con el Repositorio.
- *View*: se encarga de la visualización de los datos (del modelo). Propaga todas las acciones de usuario al *presenter*. En nuestra aplicación se corresponde con los *Frgments*.
- *Presenter*: enlaza las dos capas anteriores. Sincroniza los datos mostrados en la vista con los almacenados en el modelo y actúa ante los eventos de usuario propagados por la vista. En nuestra aplicación se corresponde con los *Presenters*.



Figura C.3: Patrón MVP.

Existen varias variantes sobre cómo implementar MVP en Android. En nuestro caso, se ha seguido la expuesta Google en Android Architecture Blueprints [13]. En ella se realizan las siguientes consideraciones:

- Se utilizan las *Activity* como controladores globales que se encargan de crear y conectar las vistas con los *presenters*.
- Se utilizan los *Fragment* como vistas ya que proporcionan numerosas ventajas cuando se trabaja con múltiples vistas.

Patrón repositorio

Para la capa del modelo, se ha utilizado el patrón repositorio que proporciona una abstracción de la implementación del acceso a datos con el objetivo de que este sea transparente a la lógica de negocio [16].

En nuestra aplicación existen dos fuentes de datos: por una parte, está la base de datos local implementada con Realm, y por otra, tenemos la API remota que nos da acceso a la información meteorológica. Ambas fuentes son transparentes para los *presenters*.

El repositorio media entre la capa de acceso a datos y la lógica de negocio de tal forma que no existe ninguna dependencia entre ellas. Consiguiendo desacoplar, mantener y testear más fácilmente el código y permitiendo la reutilización del acceso a datos desde cualquier cliente.

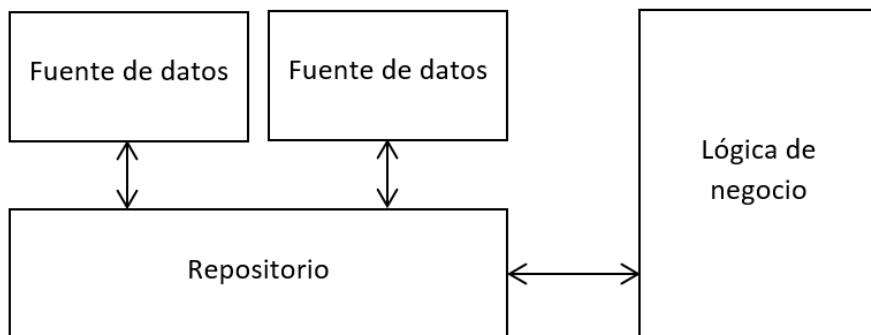


Figura C.4: Patrón repositorio.

Inyección de dependencias

A la hora de testear, es muy frecuente necesitar sustituir la implementación de una clase por otra “falsa” que se comporte de una manera predeterminada para conseguir probar la funcionalidad de manera aislada. En nuestro caso, para facilitar la labor de testeo nos vimos obligados a sustituir la base de datos Realm por una base de datos en memoria. Esta sustitución se realizó mediante la inyección de dependencias.

La inyección de dependencias es un patrón mediante el cual se proporcionan todas las dependencias que una clase necesita para su funcionamiento, en lugar de ser la propia clase quien las cree. Al separar las dependencias de la propia clase, se posibilita la opción de sustituir estas por dobles con un comportamiento definido [25].

Para la implementación de la inyección de dependencias se han utilizado los *build flavors* que proporciona Gradle. Se crearon dos *flavors*:

- **mock:** inyectaba una base de datos en memoria utilizada para el testeo de la aplicación.
- **prod:** inyectaba la base de datos Realm utilizada para producción.

Arquitectura general

El resultado de la arquitectura tras aplicar los patrones explicados es el siguiente:

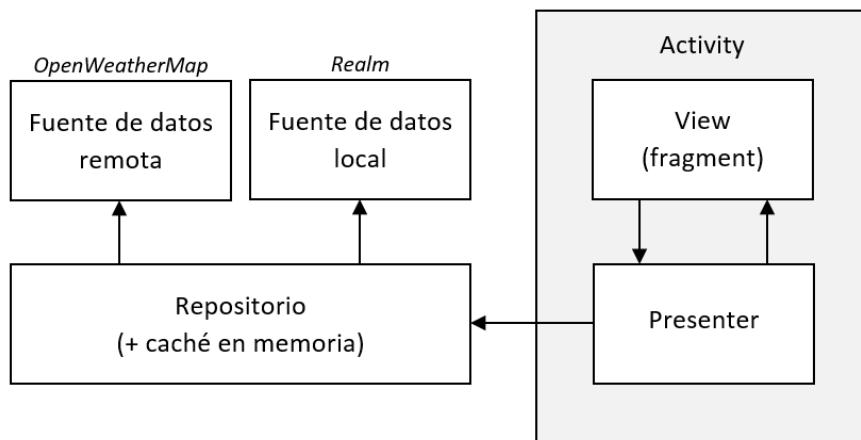


Figura C.5: Arquitectura de la aplicación.

Para agilizar la navegación por la aplicación se implementó una capa de caché en el repositorio.

Diseño de paquetes

Para la organización de los diferentes archivos que componen la aplicación no se utilizó la estrategia convencional de paquete por capa (*package by layer approach*), sino una estrategia de paquete por característica (*package per feature approach*).

Siguiendo esta estrategia se agruparon todos los archivos relacionados cada una de las distintas funcionalidades de la aplicación en un mismo paquete. De esta manera se mejora notablemente la legibilidad y la modularización de la aplicación, ya que se puede modificar cada funcionalidad de forma independiente.

Existen dos paquetes excepcionales que no siguen esta convención:

- Paquete **data**: agrupa toda la capa de modelo.
- Paquete **utils**: reúne un conjunto de clases de utilidad generales que son utilizadas por varias características.

El diagrama de paquetes es el siguiente:

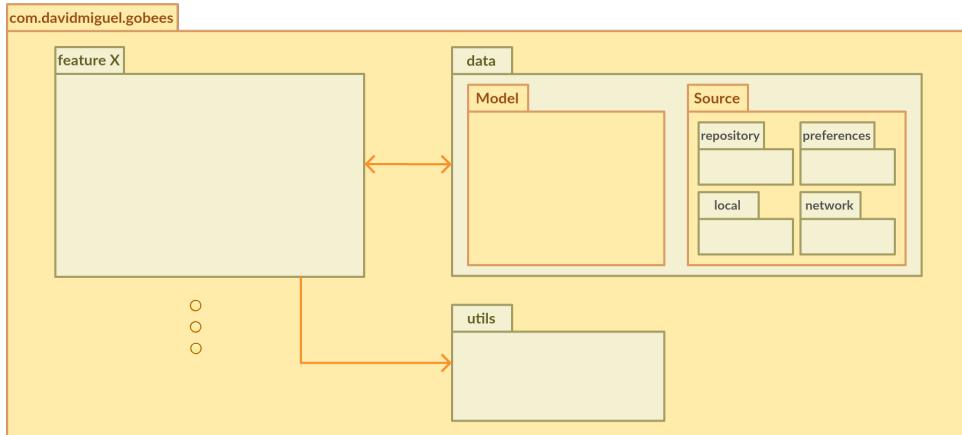


Figura C.6: Diagrama de paquetes simplificado.

El paquete **feature X** se correspondería con cada paquete de cada funcionalidad. Se ha representado de esta manera para simplificar el diagrama.

A continuación, se muestran por separado los paquetes de todas las funcionalidades:

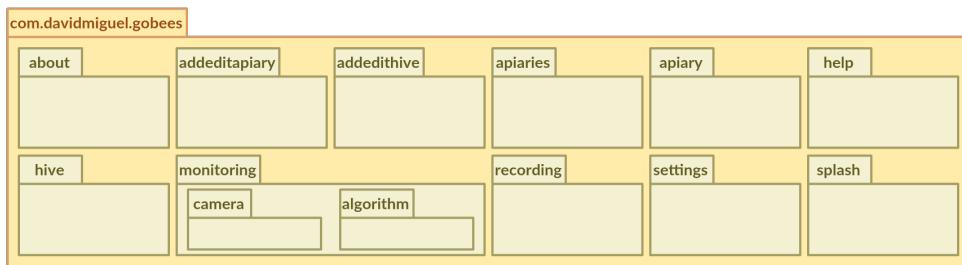


Figura C.7: Paquetes de las diferentes características.

- **about**: contiene la funcionalidad de “Acerca de” de la aplicación. Donde se muestra el autor, licencia, versión de la *app*, sitio web, historial de cambio y todas las dependencias junto con sus licencias.
- **addeditapiary**: permite añadir o editar colmenares.
- **addedithive**: permite añadir o editar colmenas.
- **apiaries**: permite listar los colmenares y gestionarlos.
- **apiary**: permite listar las colmenas de un colmenar, gestionarlas y mostrar la información relativa al colmenar.
- **help**: muestra la ayuda de la aplicación.

- **hive**: permite listar las grabaciones de una colmena, gestionarlas y mostrar la información relativa a la colmena.
- **monitoring**: agrupa toda la funcionalidad de monitorización de la actividad de vuelo de una colmena, desde la configuración hasta la ejecución del algoritmo.
- **recording**: permite visualizar los detalles de una determinada grabación.
- **settings**: permite configurar los distintos parámetros de la aplicación.
- **splash**: muestra una pantalla de inicio mientras la aplicación carga en memoria los recursos necesarios.

Diseño de clases

Aplicando MVP, cada característica clave de la aplicación posee los siguientes componentes:

- **FeatureActivity**: funciona como un controlador global que crea la vista y el *presenter* y los enlaza.
- **FeatureContract**: se trata de una interfaz que establece los siguientes contratos:
 - **FeatureContract.View**: define la capa *view* para esta característica (las únicas funciones que expone a otras capas).
 - **FeatureContract.Presenter**: define la interacción entre las capas *view* y *presenter*. Describe las acciones que pueden ser iniciadas desde la vista.
- **FeatureFragment**: implementación concreta de la capa *view*.
- **FeaturePresenter**: implementación concreta de la capa *presenter*. Escucha las acciones de usuario y actualiza la vista cuando cambia el modelo.

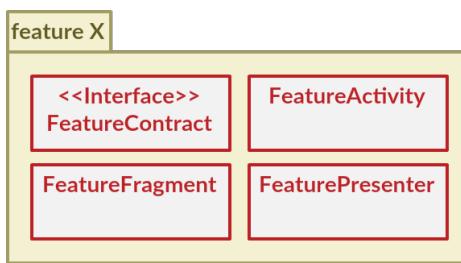


Figura C.8: Paquete tipo de una característica.

El diagrama de clases general que muestra cómo se relacionan todos los componentes de una determinada característica es el siguiente:

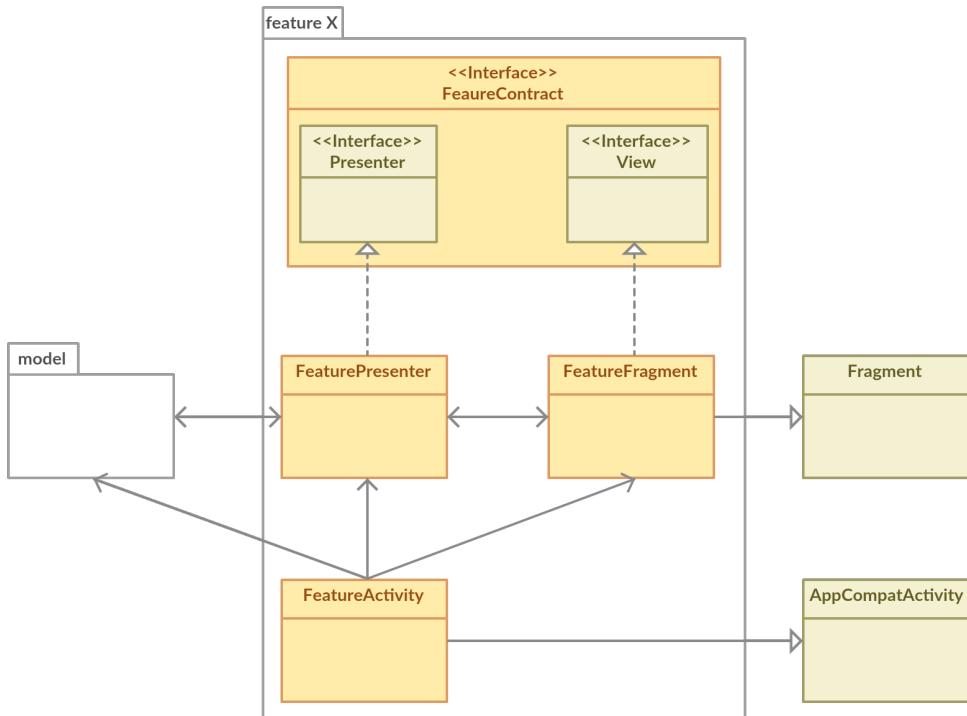
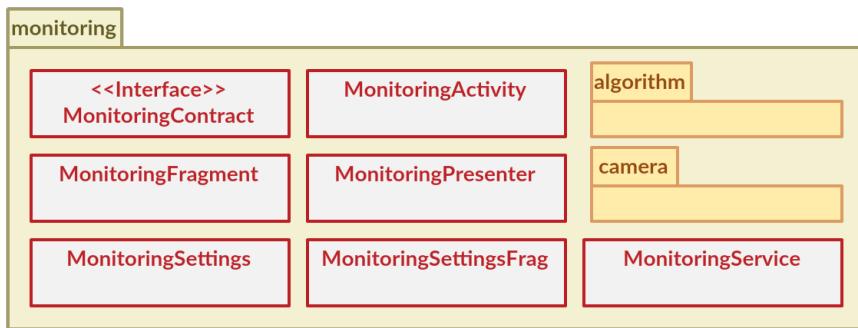
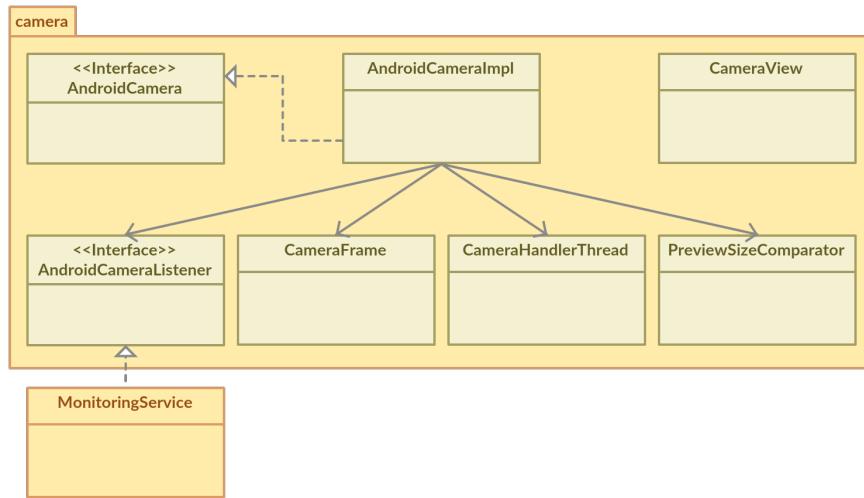


Figura C.9: Diagrama de clases general.

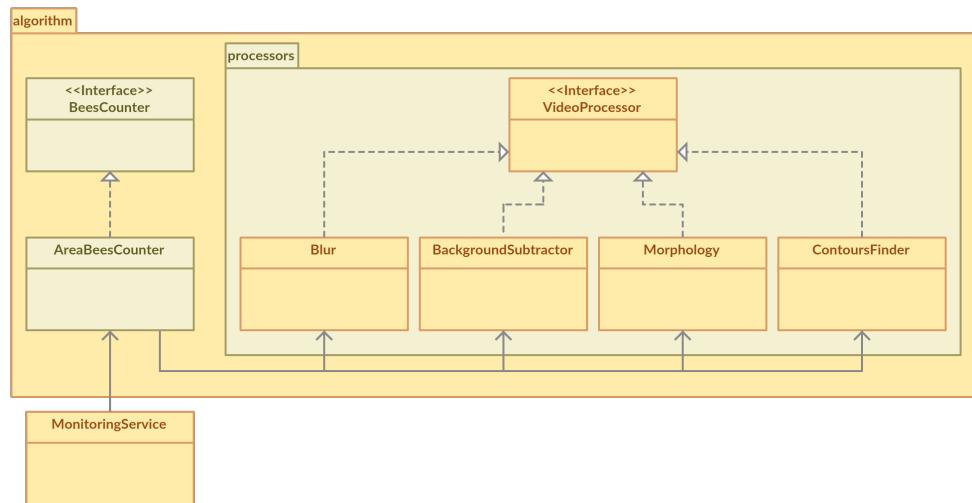
El único paquete que se diferencia de la estructura expuesta es el paquete `monitoring`. Este integra a su vez toda la lógica de acceso a la cámara y todas las clases relacionadas con el algoritmo de conteo.

Figura C.10: Paquete `monitoring`.

El diagrama de clases del paquete `camera` es el siguiente:

Figura C.11: Diagrama de clases del paquete *camera*.

El diagrama de las clases que implementan el algoritmo de conteo es el siguiente:

Figura C.12: Diagrama de clases del paquete *algorithm*.

En la parte del acceso a datos, se poseen dos paquetes como se ha visto en el apartado anterior.

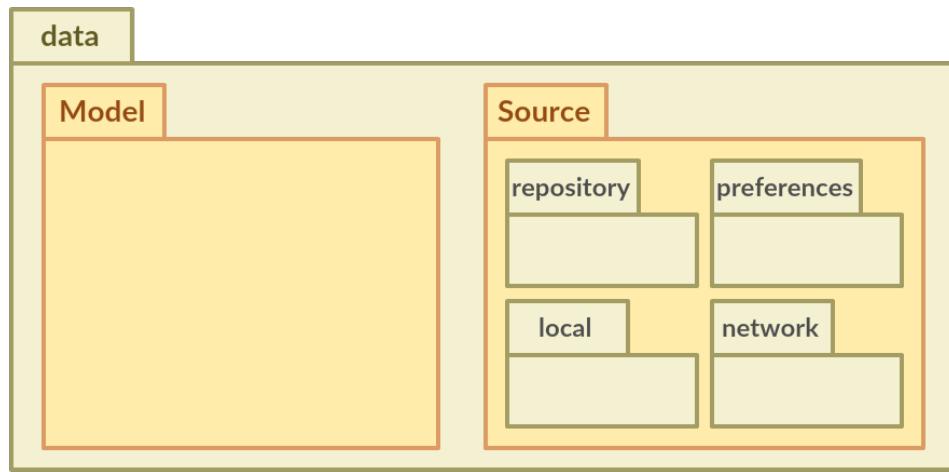


Figura C.13: Paquete *data*.

El paquete `model` contiene todas las clases de modelo que se mapean con la base de datos.

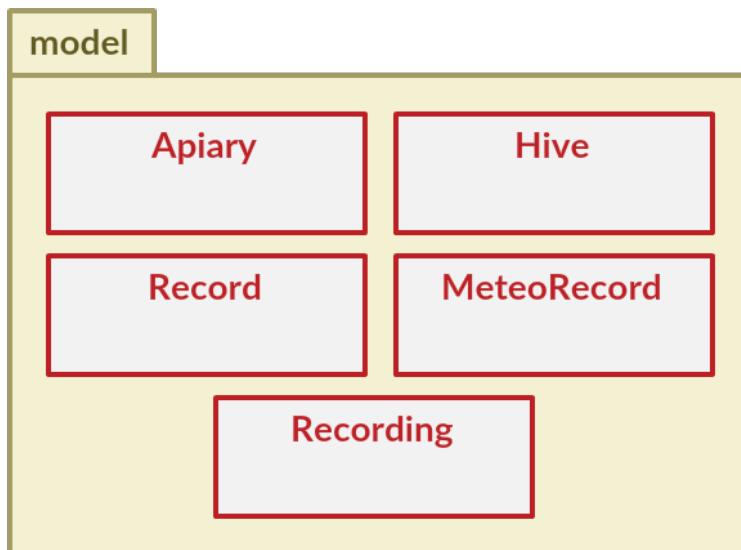


Figura C.14: Paquete *model*.

Nota: la clase `Recording` se utiliza para agrupar a un conjunto de `Records`, pero no se almacena en la base de datos directamente (solo los `Records`).

Por otro lado, el paquete `source` contiene todas las clases correspondientes a los accesos de las diferentes fuentes de datos. Su diagrama de clases es el siguiente:

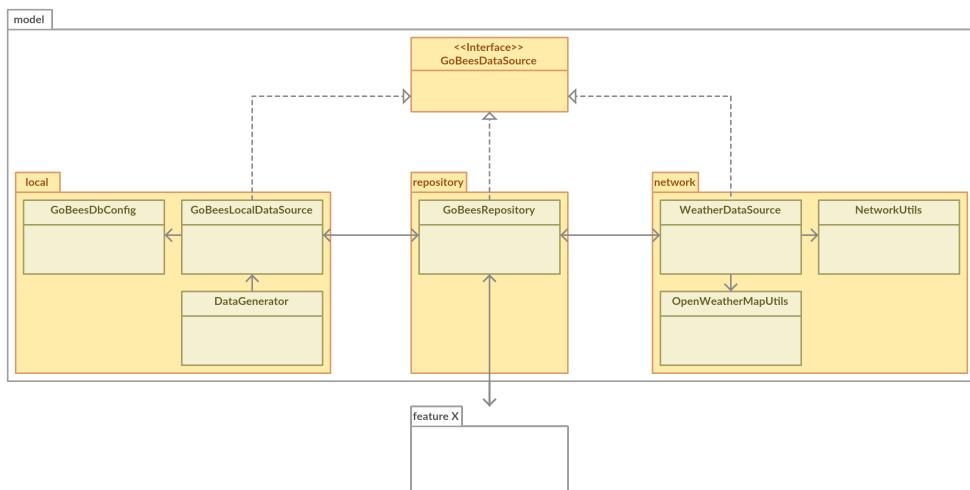


Figura C.15: Diagrama de clases del paquete `source`.

Para conocer a mayor detalle las funciones de cada clase se puede consultar la documentación JavaDoc de la aplicación.

C.4. Diseño procedimental

En este apartado se recogen los detalles más relevantes respecto a la ejecución del algoritmo de monitorización de la actividad de vuelo de una colmena.

En el siguiente diagrama de secuencia se ha representado como es la interacción entre los diferentes objetos que se encargan de la inicialización de la monitorización, la obtención de las imágenes y su posterior procesado por el algoritmo de conteo.

APÉNDICE C. ESPECIFICACIÓN DE DISEÑO

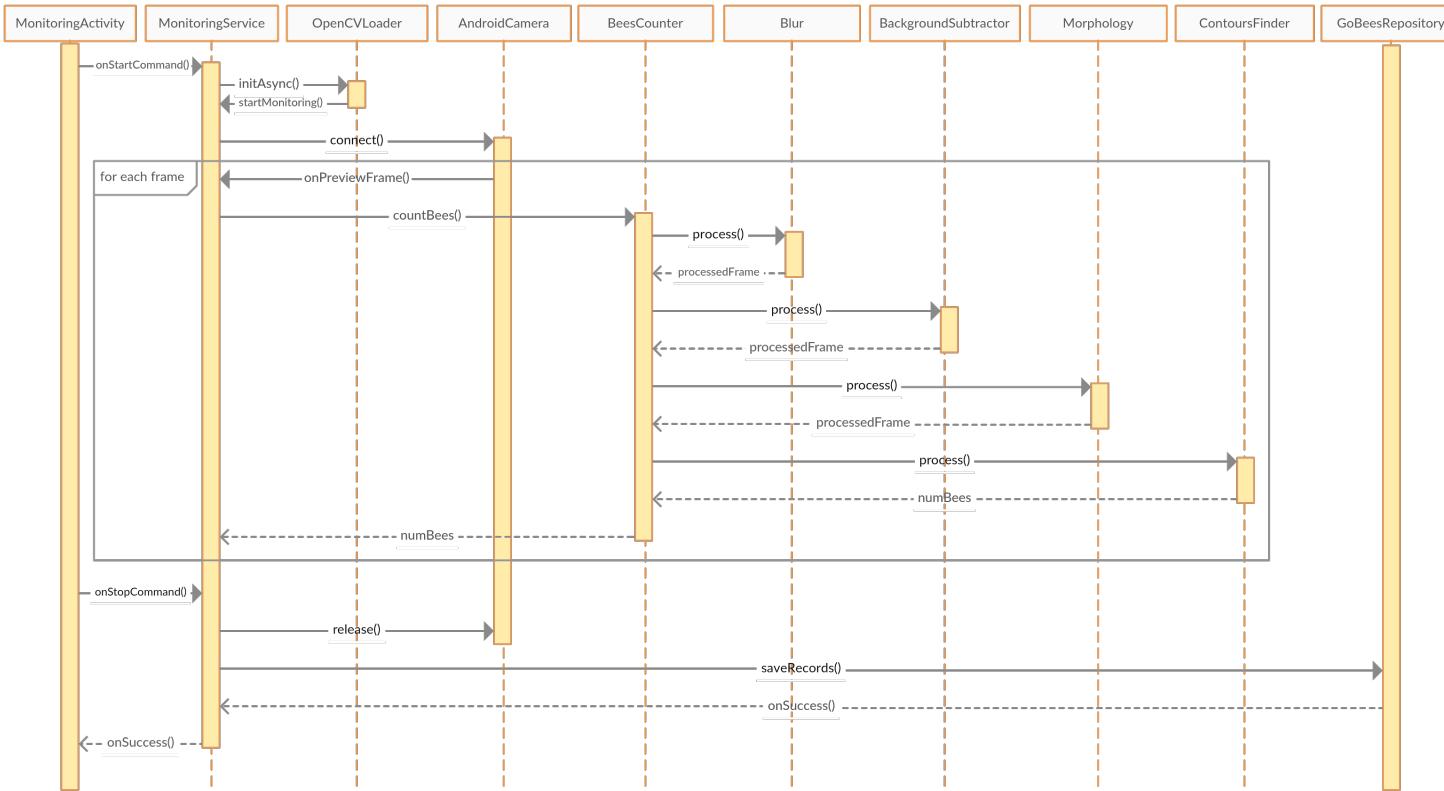


Figura C.16: Diagrama de secuencia del algoritmo.

C.5. Diseño de interfaces

En el diseño de la interfaz se ha seguido la guía de estilos de *Material Design* [9] introducida en el Google I/O 2014 y que se adoptó en Android a partir de la versión 5.0 (*Lollipop*).

En las primeras etapas de proyecto se realizaron una serie de prototipos en los que se plasmaron las principales funcionalidades de la aplicación.

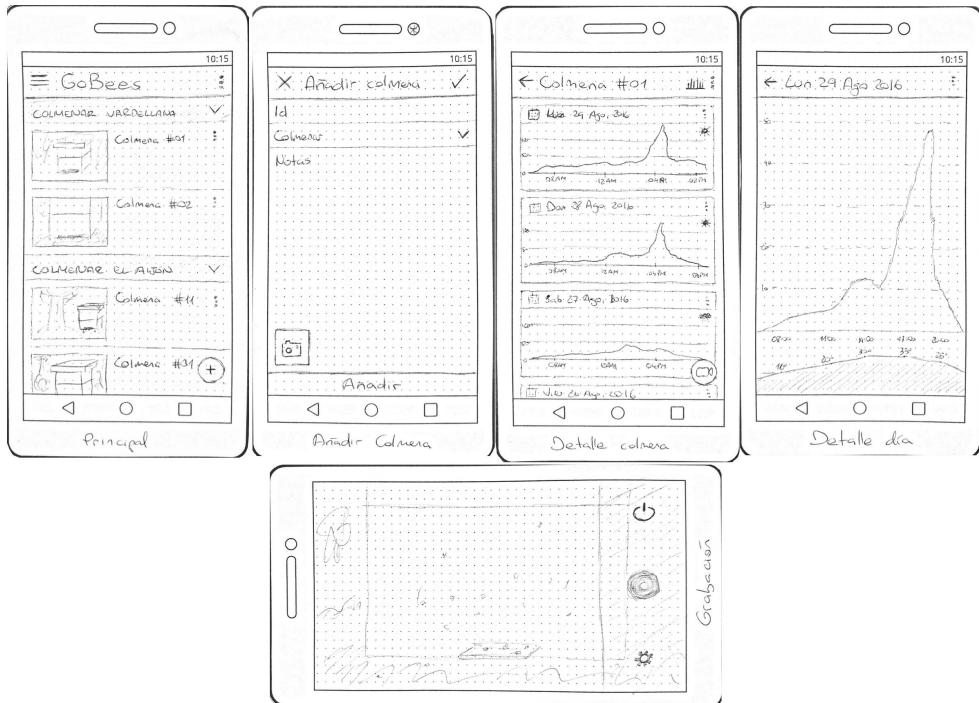


Figura C.17: Prototipos iniciales.

Tras una serie de iteraciones, estos se fueron mejorando hasta obtener las interfaces con las que cuenta hoy en día la *app*.

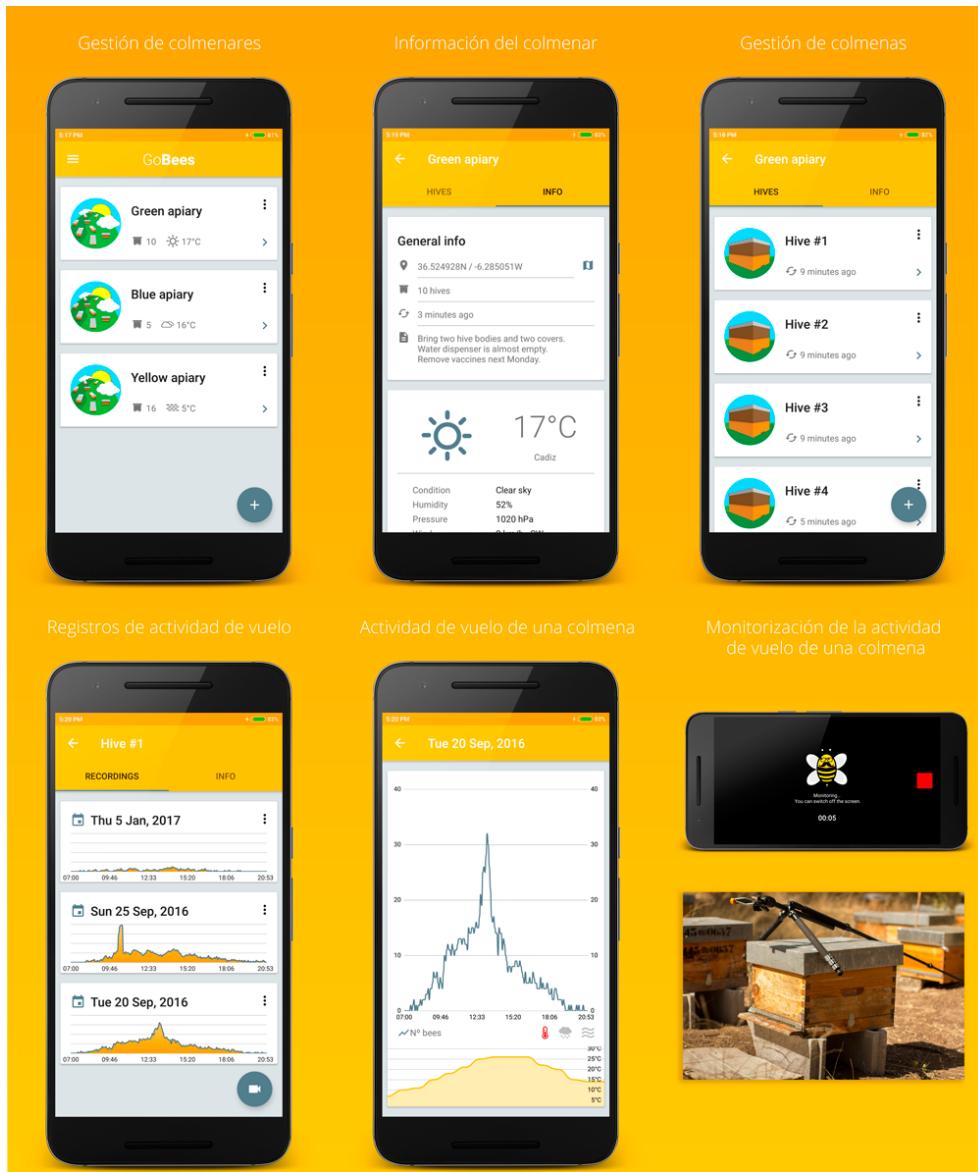


Figura C.18: Diseños finales de las interfaces.

El siguiente diagrama muestra la navegabilidad por la aplicación. Esta ha sido distribuida de acuerdo al tipo de contenido y a las tareas a realizar sobre este.

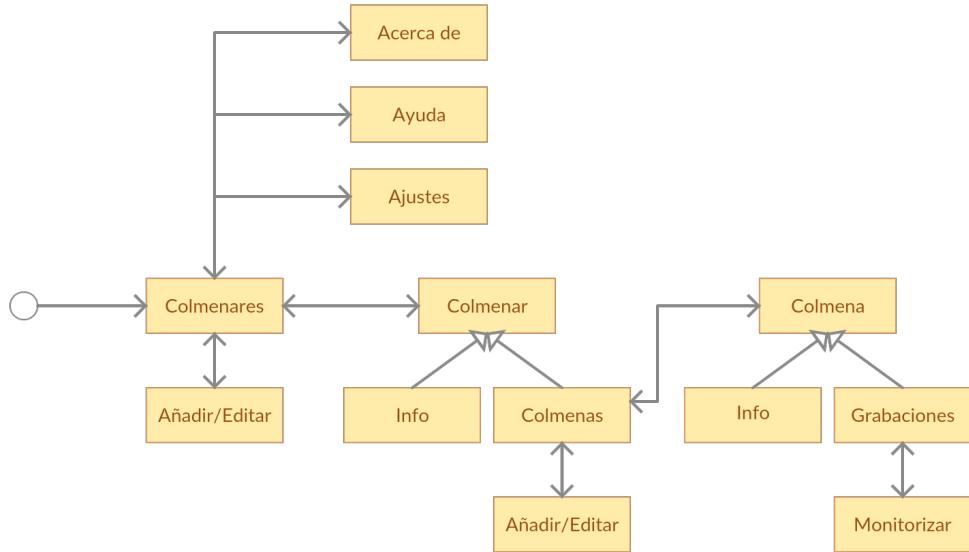


Figura C.19: Diagrama de navegabilidad.

Se ha escogido la paleta de colores entre los recomendados por *Material Design*. Utilizando como principal un color en la gama de los 500, lo que denominan un color *material*, y definiendo otro color que contraste con este para acentuar.

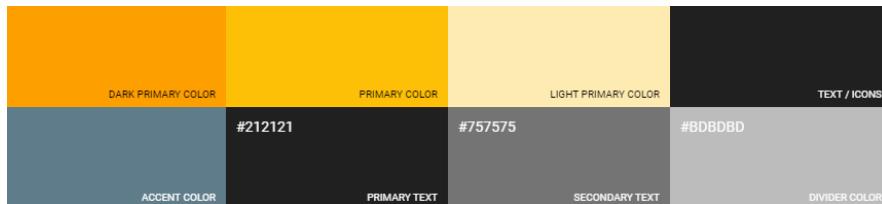


Figura C.20: Paleta de colores.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este anexo se describe la documentación técnica de programación, incluyendo la instalación del entorno de desarrollo, la estructura de la aplicación, su compilación, la configuración de los diferentes servicios de integración utilizados o las baterías de test realizadas.

D.2. Estructura de directorios

El repositorio del proyecto se distribuye de la siguiente manera:

- /: contiene los ficheros de configuración de Gradle, de los servicios de integración continua, el fichero README y la copia de la licencia.
- /app/: módulo correspondiente a la aplicación.
- /app/src/: código fuente de la aplicación.
- /app/src/main/: contiene todas las clases comunes a todos los *flavours*.
- /app/src/main/res/: recursos de la aplicación (*layouts*, menús, imágenes, cadenas de texto, etc.).
- /app/src/mock/: *mock flavour*, utilizado para injectar componentes alternativos durante los test.
- /app/src/prod/: *prod flavour*, utilizado para injectar los componentes que se utilizan en la versión de producción.
- /app/src/test/: test unitarios.
- /app/src/testMock/: test unitarios y de integración que necesitan injectar componentes falsos.
- /app/src/androidTest/: Android UI test.

- `/docs/`: documentación del proyecto.
- `/docs/img/`: imágenes utilizadas en la documentación.
- `/docs/javadoc/`: documentación *javadoc*.
- `/docs/latex/`: documentación en formato *LATEX*.
- `/docs/rst/`: documentación en formato *reStructuredText*.

Para saber más sobre la organización de un proyecto Android consultar la documentación [10].

D.3. Manual del programador

El siguiente manual tiene como objetivo servir de referencia a futuros programadores que trabajen en la aplicación. En él se explica cómo montar el entorno de desarrollo, obtener el código fuente del proyecto, compilarlo, ejecutarlo, testearlo y exportarlo.

Entorno de desarrollo

Para trabajar con el proyecto se necesita tener instalados los siguientes programas y dependencias:

- Java JDK 7.
- Android Studio.
- Git.
- OpenCV.

A continuación, se indica como instalar y configurar correctamente cada uno de ellos.

Java JDK 7

El lenguaje de programación más popular para realizar aplicaciones Android es Java. A día de hoy, Android no soporta la versión 8 de Java, por lo que tenemos que trabajar con la versión 7. Podemos obtener esta versión desde [12]. Se debe elegir correctamente el sistema operativo y la arquitectura del ordenador y, posteriormente, seguir el asistente de instalación.

Android Studio

Android Studio es el IDE oficial para el desarrollo de aplicaciones Android. Está basado en IntelliJ IDEA de JetBrains. Proporciona soporte para Gradle, emulador, editor de *layouts*, refactorizaciones específicas de Android, herramientas Lint para detectar problemas de rendimiento, uso, compatibilidad de versión, etc.

APÉNDICE D. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN20

Se puede obtener desde [11]. Junto con Android Studio se instala también el Android SDK y *Android Virtual Device* (AVD).

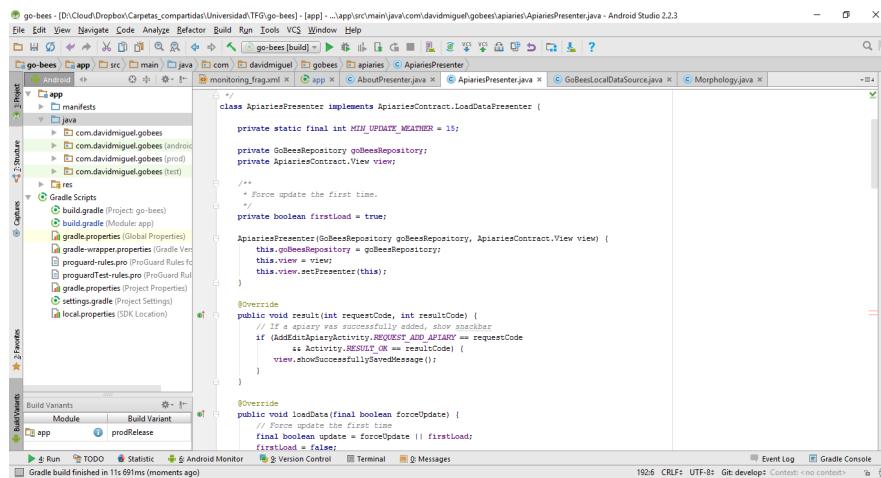


Figura D.1: Android Studio.

Git

Para hacer uso del repositorio se necesita tener instalado el gestor de versiones Git. Este programa nos permitirá clonar el repositorio, movernos por sus diferentes ramas, etiquetas, etc. Se puede obtener desde [7]. Una vez instalado, trabajaremos con Git Bash.

```
MINGW64:/c/Users/ddmll
davidmigloz@ddmlls MINGW64 /
$ cd ~

davidmigloz@ddmlls MINGW64 ~
$ git clone https://github.com/davidmigloz/go-bees.git
Cloning into 'go-bees'...
remote: Counting objects: 7202, done.
remote: Compressing objects: 100% (2284/2284), done.
remote: Total 7202 (delta 3887), reused 7170 (delta 3855), pack-reused 0
Receiving objects: 100% (7202/7202), 68.17 MiB | 1.86 MiB/s, done.
Resolving deltas: 100% (3887/3887), done.
Checking connectivity... done.

davidmigloz@ddmlls MINGW64 ~
$ |
```

Figura D.2: Terminal Git Bash.

OpenCv

OpenCV es un paquete *Open Source* de visión artificial que contiene más de 2500 librerías de procesamiento de imágenes y visión artificial, escritas en C/C++ a bajo/medio nivel.

Para ejecutar OpenCV en un dispositivo Android se necesita tener instalado la aplicación [OpenCV Manager](#). Sin embargo, para el desarrollo de la aplicación también debemos instalar la versión de escritorio de OpenCV para poder ejecutar los test de integración del algoritmo en local.

Podemos obtener OpenCV desde la página oficial [17]. En este proyecto hemos utilizado la versión 3.2.

Una vez instalada, tenemos que añadir al *path* de Windows el directorio donde se encuentran los ejecutables (`opencv/build/java/x64` o `x86`).

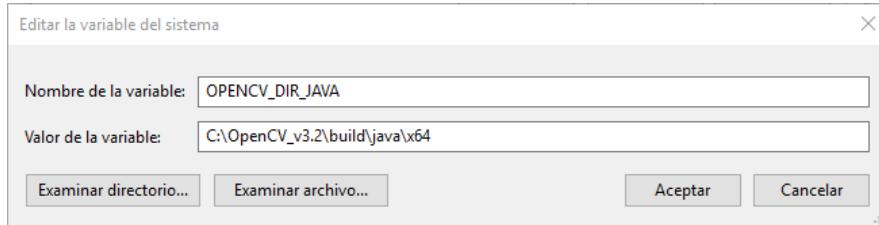


Figura D.3: Variable del sistema de OpenCV.

En la página web oficial se puede obtener información más detallada sobre el proceso de instalación.

Obtención del código fuente

Para el desarrollo de la aplicación se ha utilizado un repositorio Git hospedado en GitHub. Para obtener una copia de este hay que proceder de la siguiente manera:

1. Abrir la terminal Git Bash.
2. Desplazarse al directorio donde se desee copiar el repositorio (utilizando el comando `cd`).
3. Introducir el siguiente comando:
`git clone https://github.com/davidmigloz/go-bees.git`
4. Se iniciará la descarga del repositorio, cuando finalice se dispondrá de una copia completa de este.

```
davidmigloz@ddmlls MINGW64 ~
$ cd ~
davidmigloz@ddmlls MINGW64 ~
$ git clone https://github.com/davidmigloz/go-bees.git
Cloning into 'go-bees'...
remote: Counting objects: 7202, done.
remote: Compressing objects: 100% (2284/2284), done.
remote: Total 7202 (delta 3887), reused 7170 (delta 3855), pack-reused 0
Receiving objects: 100% (7202/7202), 68.17 MiB | 1.86 MiB/s, done.
Resolving deltas: 100% (3887/3887), done.
Checking connectivity... done.

davidmigloz@ddmlls MINGW64 ~
$ |
```

Figura D.4: Clonar repositorio de GitHub.

Para conocer el proceso detalladamente consultar [8].

Importar proyecto en Android Studio

Una vez obtenido el código fuente de la aplicación, tenemos que importarlo como proyecto de Android Studio. Para ello, hay que seguir los siguientes pasos:

1. Abrir Android Studio.
2. Menú **File > Open...**
3. Buscamos el directorio donde hemos clonado el repositorio.
4. Dentro del repositorio, seleccionamos el archivo **build.gradle**.
5. Android Studio detectará que es un proyecto Android y lo importará automáticamente.
6. Si alguna característica de las que hace uso la aplicación no se encuentra instalada, Android Studio mostrará un mensaje de error con un enlace para instalar la característica en cuestión.

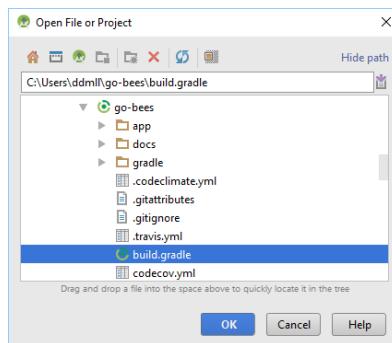


Figura D.5: Importar proyecto en Android Studio.

Para conocer el proceso detalladamente consultar [2].

Añadir nuevas características a la aplicación

Tras importar el proyecto en Android Studio, ya estamos en disposición de realizar modificaciones de la aplicación.

Para añadir una nueva característica siguiendo la arquitectura MVP, la convención de paquete por característica y las metodologías TDD y GitFlow, se deben seguir los siguientes pasos generales.

1. Crear una nueva rama (*feature branch*) desde la rama *develop*:
`git checkout -b export-data develop`
2. Crear un nuevo paquete con el nombre de la característica que se desea añadir (ej. `exportdata`).
3. Crear una interfaz (ej. `ExportDataContract.java`) que contenga a su vez dos interfaces. En una se deben definir las responsabilidades del *presenter* y en la otra las de la vista. Hacer *commit*:
`git add -A`
`git commit -m ".^dd export data contract #x"`
4. Crear una clase para el *presenter* (ej. `ExportDataPresenter.java`) que implemente su correspondiente interfaz anterior (no añadir ninguna lógica todavía). Hacer *commit*.
5. Crear una clase para la vista (ej. `ExportDataFragment`) que descienda de `Fragment` e implemente su correspondiente interfaz anterior (no añadir ninguna lógica todavía). Hacer *commit*.
6. Crear una clase que descienda de `AppCompatActivity` (ej. `ExportDataActivity.java`) y que enlace el modelo, el *presenter* y la vista. Hacer *commit*.
7. Crear un test sobre el *presenter* de acuerdo a los requisitos. Hacer *commit*.
8. Ejecutar el test y comprobar que no pasa.
9. Implementar las clases anteriores hasta conseguir que pasen el test. Hacer *commit*.
10. Refactorizar el código para mejorar su calidad. Hacer *commit*.
11. Añadir un *intent* desde donde se quiera acceder a esa característica. Hacer *commit*.
12. Una vez que se ha implementado correctamente la característica, se debe incorporar a la rama *develop* y sincronizar con GitHub:
`git checkout develop`
`git merge --no-ff export-data`
`git branch -d myfeature`
`git push origin develop`

Actualizar dependencias

Una tarea de mantenimiento común es la actualización de las dependencias de la aplicación. Es importante tenerlas actualizadas para evitar problemas de seguridad o funcionalidad que pudiesen tener en versiones anteriores.

El proyecto utiliza Gradle como sistemas de construcción automática del *software*. Una de sus funcionalidades es la gestión de dependencias. Esta permite al desarrollador definir las dependencias de su aplicación, sus versiones y los repositorios donde se hospedan y Gradle se encarga de descargarlas e importarlas al proyecto automáticamente.

Las dependencias se definen en el fichero `build.gradle` del módulo de la aplicación (`go-bees/app/build.gradle`):

```
dependencies {

    // App's dependencies, including test
    compile 'com.android.support:recyclerview-v7:25.1.1'
    compile 'com.android.support:appcompat-v7:25.1.1'
    compile 'com.android.support:design:25.1.1'
    compile 'com.android.support:cardview-v7:25.1.1'
    compile 'com.android.support:support-v4:25.1.1'
    compile 'com.github.davidmigloz:opencv-android-gradle-repo:3.2.0'
    compile 'com.google.android.gms:play-services-location:10.0.1'
    compile 'com.google.guava:guava:20.0'
    compile 'com.makeramen:roundedimageview:2.3.0'
    compile 'com.github.PhilJay:MPAndroidChart:v3.0.1'
    compile 'com.vanniktech:vnnumberpickerpreference:1.0.0'
    compile 'rebus:permission-utils:1.0.6'
    // Dependencies for local unit tests
    testCompile 'junit:junit:4.12'
    testCompile 'org.mockito:mockito-all:2.0.2-beta'
    testCompile 'org.powermock:powermock-module-junit4:1.6.6'
    testCompile 'org.powermock:powermock-api-mockito:1.6.6'
    testCompile 'org.slf4j:slf4j-api:1.7.22'
    testCompile 'org.slf4j:slf4j-log4j12:1.7.22'
    testCompile 'log4j:log4j:1.2.17'
    testCompile 'org.json:json:20160810'
    // Android Testing Support Library's runner and rules
    androidTestCompile 'com.android.support.test:runner:0.5'
    androidTestCompile 'com.android.support.test:rules:0.5'
    // Dependencies for Android unit tests
    androidTestCompile 'junit:junit:4.12'
    androidTestCompile 'org.mockito:mockito-core:2.6.2'
    // Espresso UI Testing
    androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.2'
    androidTestCompile 'com.android.support.test.espresso:espresso-contrib:2.2.2'
    androidTestCompile 'com.android.support.test.espresso:espresso-intents:2.2.2'
    // Resolve conflicts between main and test APK:
    androidTestCompile 'com.android.support:support-annotations:25.1.1'
    androidTestCompile 'com.android.support:support-v4:25.1.1'
    androidTestCompile 'com.android.support:recyclerview-v7:25.1.1'
    androidTestCompile 'com.android.support:appcompat-v7:25.1.1'
    androidTestCompile 'com.android.support:design:25.1.1'
}
```

Figura D.6: Dependencias del proyecto.

Se puede observar que existen tres formas de importar las dependencias, cada una define con un ámbito de aplicación distinto:

- **Compile**: estará disponible para el código de la aplicación.
- **testCompile**: estará disponible en los test unitarios de la aplicación.
- **androidTestCompile**: estará disponible en los test de instrumentación de la aplicación.

Para actualizar la versión de una dependencia, solamente hay que actualizar el número de la versión que figura en la importación. Posteriormente, se debe sincronizar Gradle (*Sync Project with Gradle Files*).

Compilar código fuente

La compilación del proyecto se realiza mediante la tarea **build** de Gradle. Podemos ejecutarla por línea de comandos (`./gradlew build`) o mediante la interfaz de Android Studio.

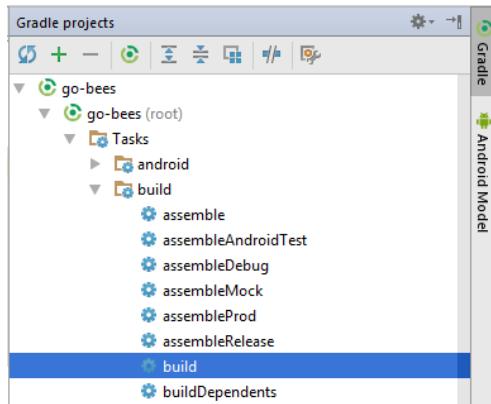


Figura D.7: Compilar proyecto.

Todos los ficheros generados durante la compilación se guardan en la carpeta **build** del proyecto.

Para conocer el proceso detalladamente consultar [1].

Ejecutar aplicación

La aplicación se puede ejecutar en un dispositivo real o en un emulador.

Dispositivo real

Para ejecutar la aplicación en un dispositivo real, se debe conectar este al equipo de desarrollo mediante un cable USB. El equipo debe tener los *drivers* del dispositivo instalado, sino no lo reconocerá.

Una vez conectado el dispositivo:

1. Presionar el botón *Run*.
2. Si el equipo reconoce el dispositivo se mostrará su nombre debajo de “*Connected Devices*”.
3. Seleccionar el dispositivo y pulsa *Ok*.
4. Se transferirá el ejecutable de la aplicación y se instalará.
5. Una vez instalada, se podrá utilizar la aplicación desde el dispositivo.

Emulador

Un emulador (denominados *Android Virtual Device - AVD*) es una aplicación que simula el funcionamiento de un dispositivo real Android. La creación y gestión de los emuladores se hace a través de *AVD Manager*.

Para ejecutar la aplicación en un emulador:

1. Presionar el botón de Run.
2. Si ya se posee algún emulador instalado, se mostrará en la lista de *Android Virtual Devices*.
3. Si no, presionar el botón “*Create New Virtual Device*”.
4. Seleccionar las características que se deseen para el emulador y pulsa finalizar.
5. Seleccionar el emulador creado y pulsar *Ok*.
6. Se iniciará el emulador y se instalará la aplicación en él.
7. Una vez instalada, se podrá utilizar la aplicación desde el emulador.

Para conocer el proceso detalladamente consultar [1].

Exportar aplicación

Para exportar la aplicación como un fichero .apk:

1. Menú *Build > Generate APK*.
2. Se generará un archivo apk y se guardará en `build/output/apk`.

Si el apk que se desea generar es para distribuirlo en Google Play, este debe estar firmado. Para ello:

1. Menú *Build > Generate Signed APK*.
2. Se debe seleccionar el archivo `.jks` con la clave e introducir su contraseña. Si no se dispone de una clave, se puede generar siguiendo el asistente.
3. Se generará un archivo `apk` firmado apto para subir al Google Play.

Para conocer el proceso detalladamente consultar [1].

Servicios de integración continua

En el repositorio se han integrado varios servicios de integración continua para detectar fallos en el software lo antes posible, reduciendo el impacto de estos y aumentando la calidad del código.

A continuación, se describe cada servicio y se indica cómo configurarlo.

TravisCI

TravisCI es una plataforma de integración continua en la nube para proyectos alojados en GitHub. Permite realizar una *build* del proyecto y testearla automáticamente cada vez que se realiza un *commit*, devolviendo un informe con los resultados.

Para integrar Travis en el repositorio hospedado en GitHub se debe crear una cuenta en su página web y dar permisos de acceso al repositorio. Una vez asociado el servicio, este se configura mediante el fichero `travis.yml`.

Las secciones más importantes de este fichero son:

- `sudo`: permite definir si el usuario de la máquina virtual tendrá privilegios o no.
- `language`: permite definir el lenguaje de programación del proyecto.
- `jdk`: permite definir la versión del JDK.
- `compiler`: permite definir el compilador.
- `addons`: permite configurar *plugins* instalados en Travis (como, por ejemplo, el *plugin* de SonarQube).
- `env`: permite definir variables de entorno.
- `android`: permite definir las dependencias Android del proyecto.
- `licenses`: permite aceptar las licencias de las dependencias.
- `before_install`: en esta sección se pueden definir comandos a ejecutar antes de los comandos de la sección `install` (por ejemplo, actualizar la lista de paquetes).
- `install`: en esta sección se deben definir aquellos comandos que instalen alguna dependencia (en nuestro caso `python-numpy`, necesaria para compilar OpenCV).

- **before_script:** en esta sección se pueden definir comandos a ejecutar antes de la sección script. En nuestro caso, nos descargamos el código fuente de OpenCV y lo compilamos.
- **script:** en esta sección se realiza la compilación del proyecto y se ejecutan los diferentes test unitarios y de integración. Además, lanza un emulador y ejecuta los test de interfaz. También ejecuta el motor de chequeo de SonarQube.
- **after_success:** esta sección se utiliza para recolectar datos generados en las secciones anteriores. En nuestro caso, se envían los diferentes informes de ejecución de los test a el servicio Codecov.
- **cache:** permite definir los directorios a cachear entre ejecuciones.

Los *log* de ejecución de Travis son accesibles desde [22].

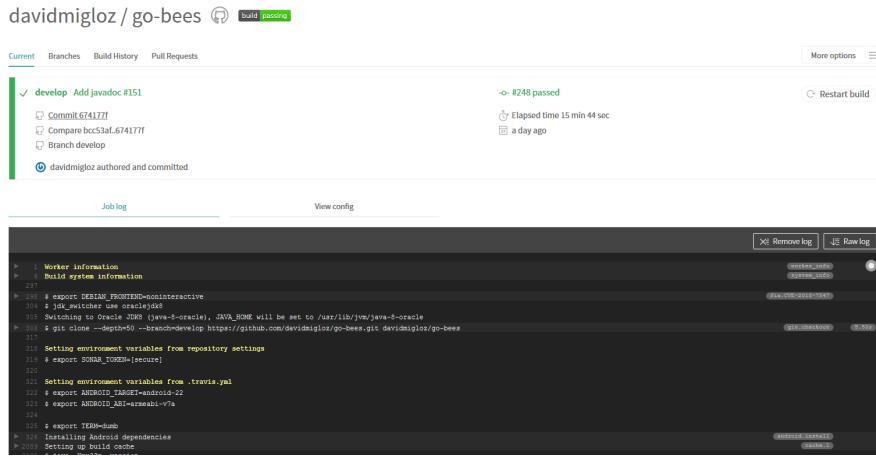


Figura D.8: TravisCI.

Para saber más, acceder a su documentación [21].

Codecov

Codecov es una herramienta que permite medir el porcentaje de código que está cubierto por un test. Además, realiza representaciones visuales de la cobertura y gráficos de su evolución.

La forma de integrarlo en el repositorio es idéntica a cómo se hizo con Travis. Adicionalmente, hay que configurar el *script* que ejecuta Travis para que al finalizar su ejecución envíe los resultados a Codecov.

```
after_success: bash <(curl -s https://codecov.io/bash)
```

La configuración de Codecov se define en el archivo `codecov.yml`.

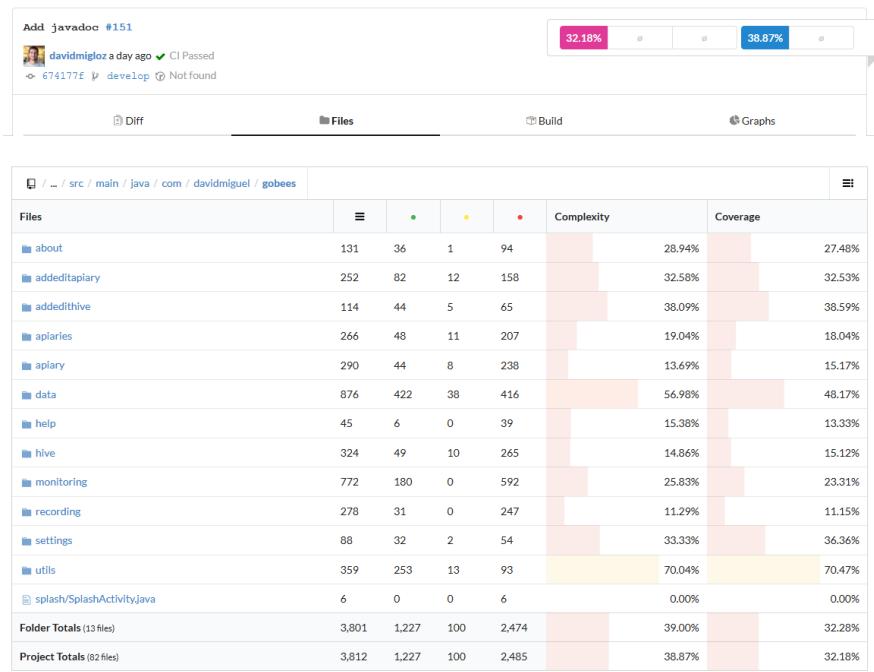


Figura D.9: Codecov.

Para saber más, acceder a su documentación [5].

CodeClimate

Codeclimate es una herramienta que realiza revisiones de código automáticamente.

La integración se realiza de forma similar a Travis. Su fichero de configuración es `.codeclimate.yml`.

En nuestro proyecto hemos activado los siguientes motores de chequeo: `checkstyle`, `fixme`, `markdownlint` y `pmd`.

CodeClimate utiliza el sistema de puntuación GPA (*Grade Point Average*) para indicar el rendimiento general del proyecto. La nota máxima se corresponde con un 4.0.

Los resultados de los chequeos se encuentran disponibles en [4].

APÉNDICE D. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN30

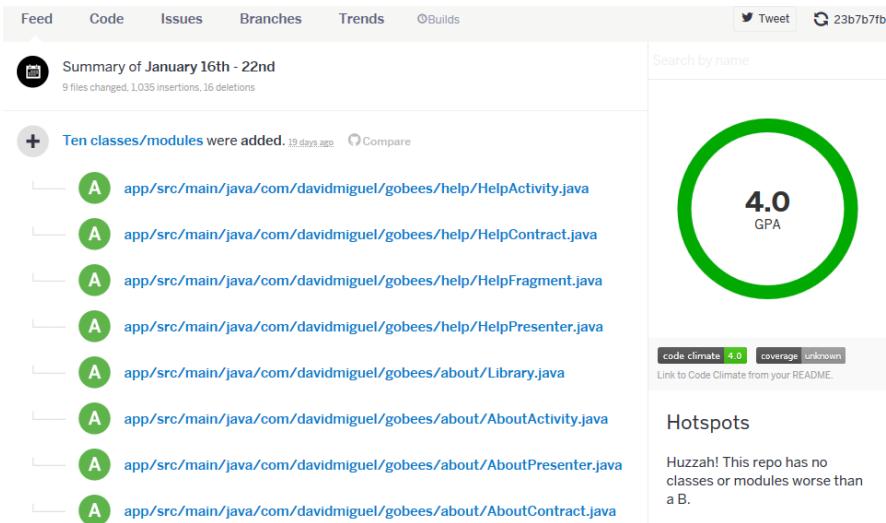


Figura D.10: CodeClimate.

Para saber más, acceder a su documentación [3].

SonarQube es una plataforma de código abierto para la revisión continua de la calidad de código. Permite detectar código duplicado, violaciones de estándares, cobertura de test unitarios, *bugs* potenciales, etc.

Para integrar el servicio hay que seguir los siguientes pasos:

1. Crear una cuenta en www.sonarqube.com.
2. Generar un *token* de autenticación.
3. Instalar el plugin de SonarQube para Gradle (`org.sonarqube`).
4. Configurar SonarQube en el fichero de configuración de Gradle (`build.gradle`).
5. Ejecutar la nueva tarea `sonarqube` de Gradle desde Travis.

```
sonarqube {
    properties {
        property "sonar.projectName", "GoBees"
        property "sonar.projectKey", "com.davidmiguel.gobees"
        property "sonar.language", "java"
        property "sonar.projectVersion", "${android.defaultConfig.versionName}"
        property "sonar.exclusions", "**/*.png,**/*.jpg"
        property "sonar.android.lint.report", "./build/outputs/lint-results-mockDebug.xml"
        property "sonar.junit.reportsPath", "./build/test-results/mockDebug"
        property "sonar.jacoco.reportPath", "./build/jacoco/testMockDebugUnitTest.exec"
        property "sonar.java.coveragePlugin", "jacoco"
        property "sonar.jacoco.reportMissing.force.zero", true
    }
}
```

Figura D.11: Configuración de SonarQube en Gradle.

Los resultados de los análisis son accesibles desde [19].

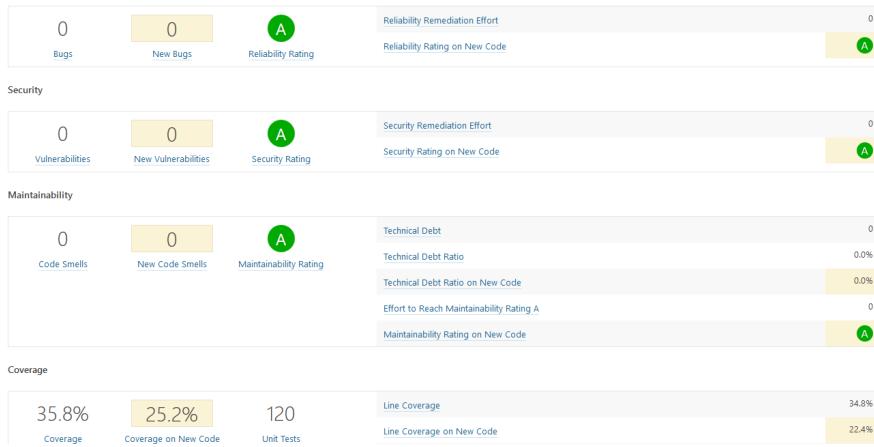


Figura D.12: SonarQube.

Para saber más, acceder a su documentación [18].

VersionEye

VersionEye es una herramienta que monitoriza las dependencias del proyecto y envía notificaciones cuando alguna de estas está desactualizada, es vulnerable o viola la licencia del proyecto.

El servicio se integra de forma similar a Travis. No necesita fichero de configuración.

Cuando se libera una nueva versión de alguna dependencia o se publica alguna vulnerabilidad, VersionEye manda una notificación. Se puede acceder a los informes desde [24].

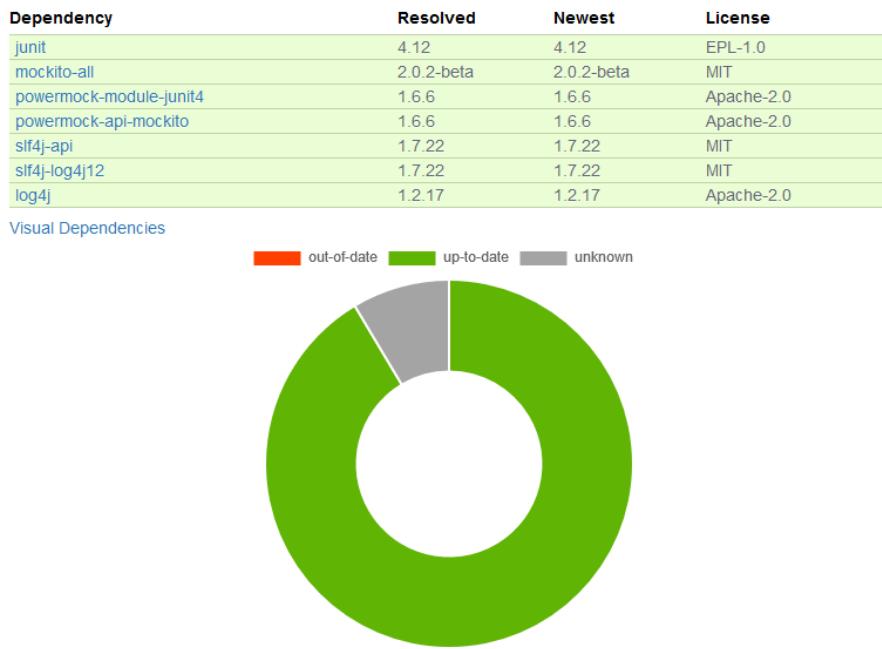


Figura D.13: VersionEye.

Para saber más, acceder a su documentación [\[23\]](#).

Read the Docs

Read the Docs es un servicio de documentación continua que permite crear y hospedar una página web generada a partir de los distintos ficheros Markdown o reStructuredText de la documentación. Cada vez que se realiza un *commit* en el repositorio se actualiza la versión hospedada.

Se integra en el repositorio de la misma manera que Travis. Y se configura mediante el archivo `conf.py` ubicado en `go-bees/docs/rst`.

Actualmente, se encuentra configurado para generar una sección en la página web por cada archivo reStructuredText que encuentre dentro del directorio `rst`.



Figura D.14: Página web generada con ReadTheDocs.

Para saber más, acceder a su documentación [20].

D.4. Pruebas del sistema

Para verificar el funcionamiento de cada uno de los módulos de la aplicación, su integración y la interacción con estos desde la interfaz, se han desarrollado una serie de baterías de test.

Test unitarios

Los test unitarios comprueban la funcionalidad de un único módulo trabajando de forma aislada. Para su escritura se han utilizado las dependencias jUnit y Mockito.

jUnit es un *framework* de Java utilizado para realizar pruebas unitarias. Mockito es un *framework* de *mocking* que permite crear objetos *mock* fácilmente. Estos objetos simulan parte del comportamiento de una clase. De esta manera, podemos aislar el módulo a testear para que los módulos de los que depende no interfieran en los resultados del test.

Se han escrito 120 test unitarios que testean 30 clases distintas. Se han testeado en su mayoría los *presenters* que son los que poseen la lógica de la aplicación y no tienen ninguna dependencia al *framework* de Android. Lo que permite ejecutarlos sin necesidad de lanzar un emulador.

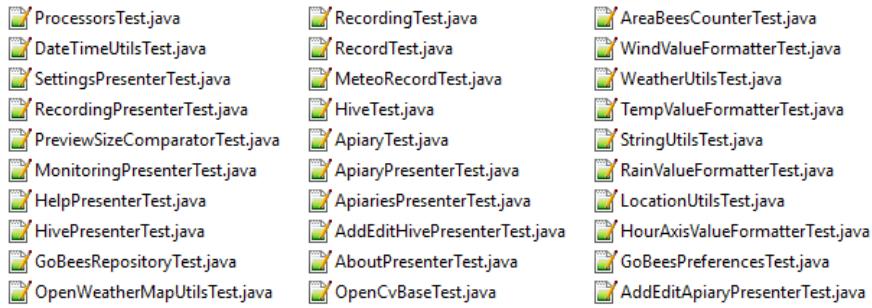


Figura D.15: Test unitarios.

Ejecución de los test unitarios

Los test unitarios se ejecutan automáticamente en Travis cada vez que se realiza un *commit* y se hace un *push* a GitHub. Pero también se pueden ejecutar en local. Para ello:

1. Seleccionar el *Build Variants mockDebug*.
2. Seleccionar como tipo de vista Android.
3. Pulsar botón derecho en el paquete **test > Run test in go-bees**.
4. Se ejecutarán todos los test y se obtendrá un informe de resultados.

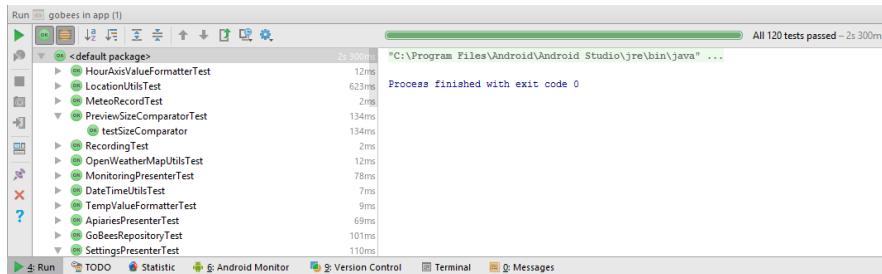


Figura D.16: Ejecución de los test unitarios.

Test del algoritmo

Para testear el algoritmo se han escrito varios test unitarios que prueban cada uno de sus módulos y un test de integración (**AreaBeesCounterTest.java**) que lo testea en su totalidad contra tres conjuntos de fotogramas etiquetados manualmente. De esta manera, se obtiene el error que comete el algoritmo en cada caso y se compara con unos límites prefijados. Si por alguna modificación accidental el error supera el límite, el test falla.

Ejecución del test del algoritmo

El test de integración se ejecuta automáticamente en Travis junto con los test unitarios. También puede ser ejecutado en local, pero es imprescindible tener instalado OpenCV en el equipo. Los pasos a seguir son:

1. Seleccionar el *Build Variants mockDebug*.
2. Seleccionar como tipo de vista Android.
3. Pulsar botón derecho en el paquete `testMock > Run test in go-bees`.
4. Se ejecutará el test y se obtendrá un informe de resultados.

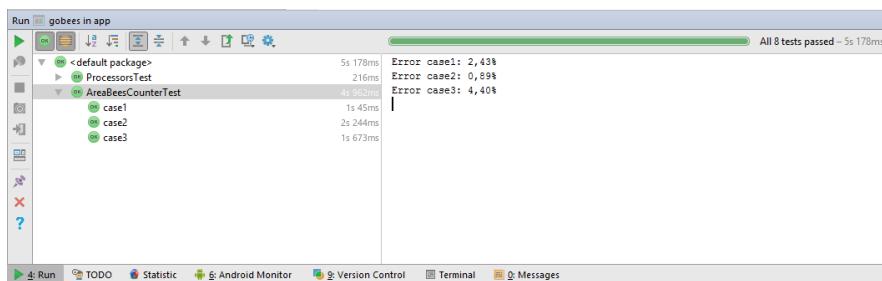


Figura D.17: Ejecución del test de integración del algoritmo.

Etiquetado de nuevos conjuntos de fotogramas

Para etiquetar videos manualmente se ha desarrollado una aplicación en Java que facilita esta labor. La aplicación va mostrando cada fotograma y el usuario solo tiene que pinchar encima de cada abeja existente. Finalmente, la aplicación permite exportar los datos en un archivo CSV con el formato que utiliza el test del algoritmo.

Los pasos a seguir son:

1. Ejecutar la aplicación (Disponible en [14]).
2. Abrir el directorio que posee los fotogramas.
3. Marcar las abejas presentes en cada fotograma con el ratón. La aplicación mostrará el número del fotograma y el número de abejas marcadas.
4. Al finalizar, seleccionar guardar. La aplicación exportará los datos en un archivo CSV.



Figura D.18: Aplicación de etiquetado de fotogramas.

Testeo de la parametrización del algoritmo

Para desarrollar el algoritmo y parametrizarlo de forma óptima, se desarrolló una aplicación Java que permite modificar los diferentes parámetros de cada fase en tiempo real y calcular sus tiempos de cómputo.

Si se desea probar nuevas parametrizaciones:

1. Ejecutar la aplicación (Disponible en [14]. Es necesario tener instalado OpenCV en el equipo).
2. Seleccionar un archivo de vídeo de prueba.
3. En la ventana izquierda se visualiza la entrada del algoritmo y a la derecha existe una pestaña por cada fase de este.
4. En cada pestaña, a parte de la salida del algoritmo para esa fase, se poseen una serie de controles para parametrizar el algoritmo.
5. En la parte inferior izquierda se muestra los fotogramas por segundo que se están procesando. En la parte central, el tiempo total de procesado. Y en la parte derecha, el tiempo parcial de la fase en cuestión.

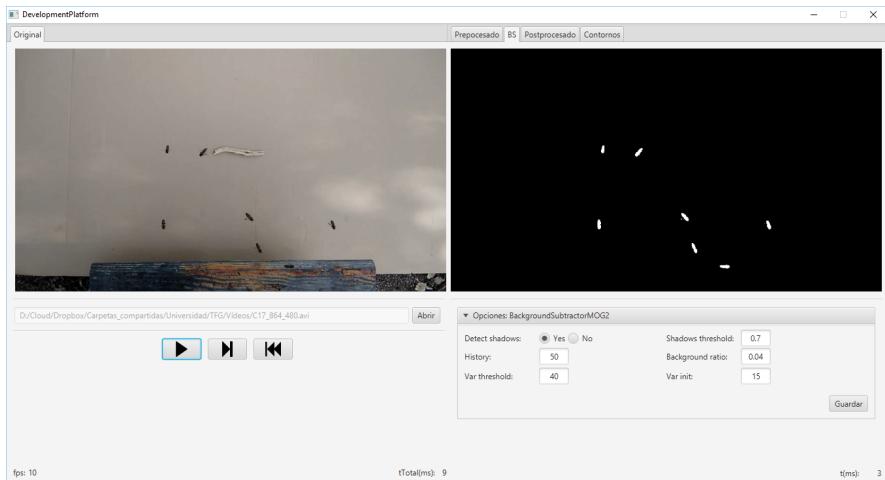


Figura D.19: Plataforma de desarrollo del algoritmo.

Test de interfaz

Por último, se han desarrollado 17 test de interfaz que testean cada uno de los requisitos de la aplicación, a excepción del requisito de monitorización que no fue posible testearlo en un emulador (no se puede utilizar como *feed* de la cámara de un emulador un archivo de vídeo).

Para desarrollar los test se ha utilizado Espresso, un *framework* de *testing* para Android que provee una API para escribir UI test que simulen las interacciones de usuario con la *app*.

En la siguiente tabla se relaciona cada test con el requisito que comprueba.

Test	Requisitos
AddApiaryTest.java	RF-1.1 Añadir colmenar.
EditApiaryTest.java	RF-1.2 Editar colmenar.
DeleteApiaryTest.java	RF-1.3 Eliminar colmenar.
ListApiariesTest.java	RF-1.4 Listar colmenares.
ViewApiaryTest.java	RF-1.5 Ver colmenar.
AddHiveTest.java	RF-2.1 Añadir colmena.
EditHiveTest.java	RF-2.2 Editar colmena.
DeleteHiveTest.java	RF-2.3 Eliminar colmena.
ListHivesTest.java	RF-2.4 Listar colmenas.
ViewHiveTest.java	RF-2.5 Ver colmena.
AddRecordingTest.java	RF-3.1 Añadir grabación.
DeleteRecordingTest.java	RF-3.2 Eliminar grabación.
ListRecordingsTest.java	RF-3.3 Listar grabaciones.
ViewRecordingTest.java	RF-3.4 Ver grabación.
SettingsTest.java	RF-5 Configuración de la aplicación.
HelpTest.java	RF-6 Ayuda de la aplicación.
AboutTest.java	RF-7 Información de la aplicación.

Tabla D.1: Requisitos testeados.

Ejecución de los test de interfaz

Para ejecutar los test de interfaz es imprescindible contar con un dispositivo físico o un emulador. Una vez conectado, se siguen los siguientes pasos:

1. Seleccionar el *Build Variants* `mockDebug`.
2. Seleccionar como tipo de vista Android.
3. Pulsar botón derecho en el paquete `androidTest > Run test in go-bees`.
4. Se ejecutarán cada uno de los test en el dispositivo (Android Studio instala una aplicación adicional que instrumenta a la aplicación a testear).
5. Al finalizar, se obtiene un informe con los resultados.

Apéndice E

Documentación de usuario

E.1. Introducción

En este manual se detallan los requerimientos de la aplicación, cómo instalarla en un dispositivo Android e indicaciones sobre cómo utilizarla correctamente. Todos los procedimientos aquí descritos se encuentran también disponibles en formato video.

E.2. Requisitos de usuarios

Los requisitos mínimos para poder hacer uso de la aplicación son:

- Contar con un dispositivo que posea Android 4.4 (*KitKat* – API 19) o superior.
- Para utilizar la característica de monitorización de la actividad, es necesario tener instalada la aplicación [OpenCV Manager](#).
- También se necesita contar con permiso para acceder a la cámara del dispositivo.
- Si se desea localizar los colmenares mediante GPS, es necesario contar con un dispositivo que lo soporte y conceder el permiso de localización a la aplicación.
- Para acceder a la información meteorológica se requiere conexión a internet.

E.3. Instalación

La instalación se puede realizar de dos maneras: a través de Google Play o instalando directamente el ejecutable de la aplicación en nuestro dispositivo.

Desde Google Play

Google Play es una plataforma de distribución digital de aplicaciones móviles para los dispositivos Android. GoBees se distribuye por esta plataforma desde su versión 1.0.

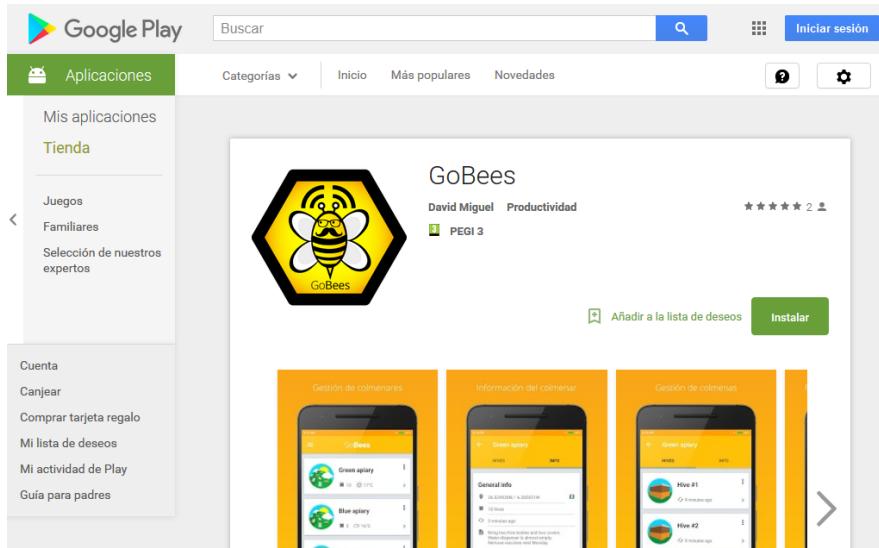


Figura E.1: GoBees en Google Play.

Video-tutorial: <http://gobees.io/help/videos/installacion-google-play>

Para instalar la aplicación debemos realizar los siguientes pasos:

1. Acceder a la aplicación Google Play.
2. Buscar el término “GoBees”.
3. Entrar en la sección correspondiente a la aplicación.
4. Pulsar el botón instalar.
5. Cuando la instalación haya finalizado, pulsar sobre el botón abrir.
6. La instalación habrá finalizado y la aplicación estará lista para su uso.



Figura E.2: Instalación desde Google Play

Desde fichero ejecutable

La otra opción, es realizar la instalación directamente desde el fichero ejecutable de la aplicación. Estos ficheros poseen la extensión .apk. Podemos conseguir la última versión del .apk de GoBees desde [15].

Video-tutorial: <http://gobees.io/help/videos/instalacion-apk>

Una vez descargado, tenemos que seguir los siguientes pasos:

1. En primer lugar, hay que permitir la instalación de “aplicaciones con orígenes desconocidos”. Para ello:
 - a. Ir a ajustes del dispositivo.
 - b. Seguridad (o Privacidad).
 - c. Activar “Orígenes desconocidos”.
2. Ejecutar el fichero descargado.
3. Pulsar el botón instalar.
4. Cuando la instalación haya finalizado, pulsar sobre el botón abrir.
5. La instalación habrá finalizado y la aplicación estará lista para su uso.

E.4. Manual de usuario

En esta sección se describe el uso de las diferentes funcionalidades de la aplicación.

Generar datos de muestra

Una de las mejores maneras de aprender a utilizar una aplicación es indagando en ella. GoBees permite generar un colmenar de prueba, de tal manera, que podemos explorar las diferentes secciones con datos reales.

Video-tutorial: <http://gobees.io/help/videos/generar-colmenar-prueba>

Para generar los datos de prueba:

1. Pulsar el botón menú.
2. Entrar en la sección “Ajustes”.
3. Seleccionar la opción “Generar datos de muestra”.
4. Se generará un colmenar con tres colmenas y tres grabaciones por colmena.

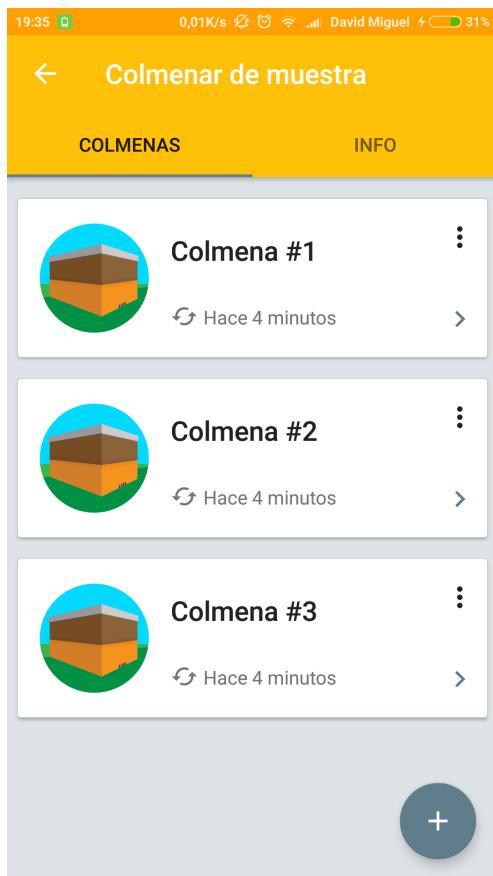


Figura E.3: Colmenar de muestra.

Añadir un colmenar

Un colmenar hace referencia al lugar o recinto donde se poseen un conjunto de colmenas. Un colmenar posee un nombre, una localización y unas notas.

Video-tutorial: <http://gobees.io/help/videos/anadir-colmenar>

Para añadir un nuevo colmenar:

1. Desde la pantalla principal.
2. Pulsar el botón “+”.
3. Definir el nombre del colmenar (obligatorio).
4. Definir la localización del colmenar (opcional).
 - a. Se pueden introducir manualmente las coordenadas, indicando la latitud y la longitud en el sistema de coordenadas geográficas.
 - b. Alternativamente, se puede obtener la localización actual automáticamente pulsando el botón situado en la parte derecha (se necesitan permisos de localización para utilizar esta característica).
5. Definir unas notas sobre el colmenar (opcional). En las notas se puede apuntar cualquier cosa relacionada con el colmenar en general.
6. Pulsar el botón ✓ para guardar el nuevo colmenar.

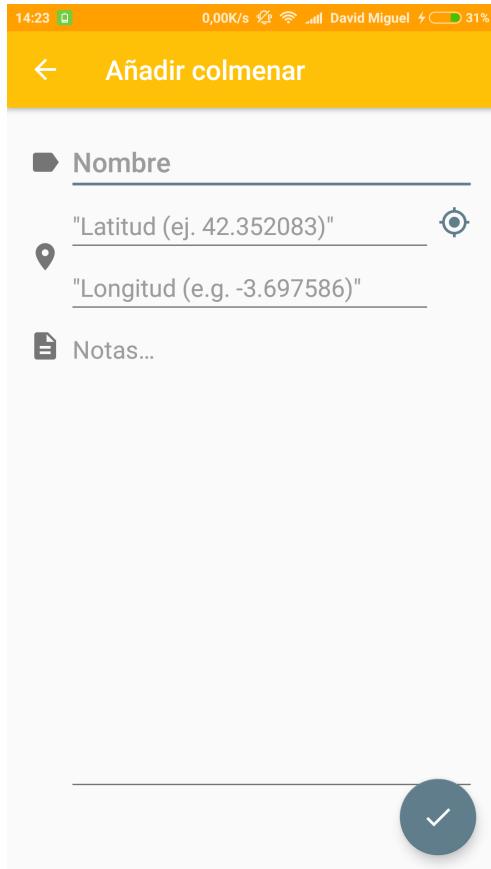


Figura E.4: Añadir colmenar.

Editar un colmenar

Los detalles de un colmenar se pueden editar en cualquier momento.

Video-tutorial: <http://gobees.io/help/videos/editar-colmenar>

Para editar un colmenar existente:

1. Desde la pantalla principal.
2. Pulsar el botón de menú asociado al colmenar a editar (tres puntos verticales situados en la esquina superior derecha).
3. Seleccionar la opción de editar.
4. Se abrirá la pantalla de edición, donde se podrán modificar los datos que se deseen.
5. Pulsar el botón ✓ para actualizar los datos editados.

Eliminar un colmenar

Al eliminar un colmenar, se eliminan también todos los datos asociados a este (información del colmenar, colmenas, grabaciones e información meteorológica).

Video-tutorial: <http://gobees.io/help/videos/eliminar-colmenar>

Para eliminar un colmenar existente:

1. Desde la pantalla principal.
2. Pulsar el botón de menú asociado al colmenar a eliminar (tres puntos verticales situados en la esquina superior derecha).
3. Seleccionar la opción de eliminar.
4. El colmenar se eliminará junto con toda su información.

Consultar la información meteorológica de un colmenar

Para poder consultar la información meteorológica de un colmenar se necesita que este posea una localización y que el dispositivo esté conectado a internet. Si se cumplen estos dos requisitos, la información meteorológica del colmenar se actualizará automáticamente de forma periódica.

Video-tutorial: <http://gobees.io/help/videos/consultar-info-meteo-colmenar>

Para consultar la información meteorológica:

1. Asegurarse de que el colmenar tiene definida una localización y que se posee conexión a internet.
2. En la lista de colmenares, se puede visualizar un resumen con la temperatura y situación meteorológica en cada colmenar.

3. Si se desea consultar la información en detalle, entrar en el colmenar a consultar.
4. Desplazarse a la pestaña “info”.
5. En la parte inferior podremos visualizar todos los detalles de la situación meteorológica actual en ese colmenar.

Se pueden cambiar las unidades meteorológicas, para ello:

1. En la pantalla principal.
2. Pulsar el botón menú.
3. Entrar en la sección “Ajustes” .
4. Seleccionar “Unidades meteorológicas” .
 - a. Sistema métrico: °C y km/h.
 - b. Sistema imperial: °F y mph.

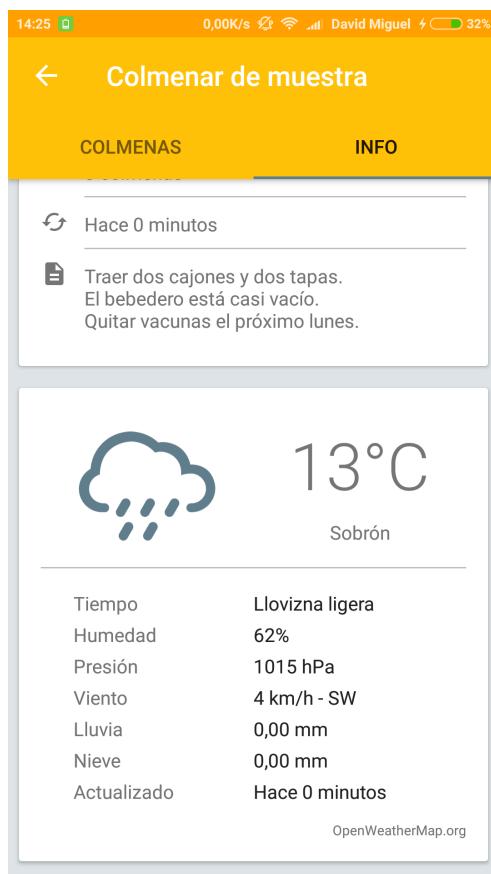


Figura E.5: Información meteorológica.

Visualizar un colmenar en el mapa

GoBees nos permite visualizar fácilmente un determinado colmenar en un mapa utilizando nuestra aplicación de mapas favorita. De esta manera, podemos navegar hacia él o consultar cualquier detalle cartográfico.

Video-tutorial: <http://gobees.io/help/videos/ver-colmenar-mapa>

Para visualizar un colmenar en el mapa:

1. Entrar en el colmenar a visualizar.
2. Desplazarse a la pestaña “info”.
3. Pulsar el botón “mapa” situado a la derecha de la localización del colmenar.
4. Seleccionar la aplicación con la que se desea visualizar el colmenar.

Añadir una colmena

Cada colmena pertenece a un colmenar y tiene un nombre y unas notas. Además, se puede monitorizar su actividad de vuelo, dando lugar a grabaciones.

Video-tutorial: <http://gobees.io/help/videos/anadir-colmena>

Para añadir una colmena en un determinado colmenar:

1. Entrar en el colmenar al que pertenecerá.
2. Definir el nombre de la colmena (obligatorio).
3. Definir unas notas sobre la colmena (opcional). En las notas se puede apuntar cualquier cosa relacionada con la colmena en concreto.
4. Pulsar el botón ✓ para guardar la nueva colmena.

Editar una colmena

Los detalles de una colmena se pueden editar en cualquier momento.

Video-tutorial: <http://gobees.io/help/videos/editar-colmena>

Para editar una colmena existente:

1. Entrar en el colmenar al que pertenece la colmena.
2. Pulsar el botón de menú asociado a la colmena a editar (tres puntos verticales situados en la esquina superior derecha).
3. Seleccionar la opción de editar.
4. Se abrirá la pantalla de edición, donde se podrán modificar los datos que se deseen.
5. Pulsar el botón ✓ para actualizar los datos editados.

Eliminar una colmena

Al eliminar una colmena, se eliminan también todos los datos asociados a esta (información de la colmena y sus grabaciones).

Video-tutorial: <http://gobees.io/help/videos/eliminar-colmena>

Para eliminar una colmena existente:

1. Entrar en el colmenar al que pertenece la colmena.
2. Pulsar el botón de menú asociado a la colmena a editar (tres puntos verticales situados en la esquina superior derecha).
3. Seleccionar la opción de eliminar.
4. La colmena se eliminará junto con toda su información.

Monitorizar la actividad de vuelo de una colmena

La actividad de vuelo, junto con información previa de la colmena y conocimiento de las condiciones locales, permite conocer al apicultor el estado de la colmena con bastante seguridad, pudiendo determinar si esta necesita o no una intervención.

GoBees permite monitorizar este parámetro utilizando la cámara del *smartphone*.

Video-tutorial: <http://gobees.io/help/videos/monitorizacion-act-vuelo>

Para monitorizar la actividad de vuelo es necesario colocar el *smartphone* de forma fija en posición cenital a la colmena. Para esto, se puede utilizar un trípode o un soporte similar. En la siguiente imagen se puede ver un ejemplo de colocación:



Figura E.6: Colocación del *smartphone* en la colmena.

Para mejorar los resultados de la monitorización, es recomendable que el suelo sea de un color claro y uniforme. Si posee maleza, se puede colocar un cartón o similar, como se muestra en la imagen.

Una vez realizado en montaje, hay que seguir los siguientes pasos dentro de la aplicación:

1. Entrar en el colmenar al que pertenece la colmena a monitorizar.
2. Entrar en la colmena.
3. Pulsar en el botón de “monitorización” (situado en la parte inferior derecha con un icono de una cámara).
4. Se abrirá una ventana que permite previsualizar la monitorización.
5. Para configurar los parámetros de la monitorización, pulsar el botón “ajustes” (situado en la parte superior derecha). Se abrirá una pantalla con los siguientes ajustes:
 - **Mostrar salida del algoritmo:** si no se encuentra activado se previsualiza la imagen proveniente de la cámara. Si se activa, se muestran en verde las abejas detectadas y en rojo otros objetos en movimiento que el algoritmo no considera abejas. Además, en la esquina inferior derecha se puede visualizar el número total de abejas contadas en cada fotograma.
 - **Modificar el tamaño de las regiones:** dependiendo de la distancia a la que esté situada la cámara, es posible que las abejas se visualicen demasiado pequeñas o demasiado grandes. Con esta opción, se puede agrandar o disminuir su silueta.
 - **Min. área abeja:** la detección de una abeja se realiza por área. Si el contorno en movimiento detectado posee un área dentro de unos límites se considera una abeja. Este parámetro configura la cota inferior del área. Bien ajustado, permite descartar moscas y mosquitos.
 - **Max. área abeja:** configura la cota superior del área. Permite descartar la mayoría de animales que pueden habitar en el colmenar (avispones, roedores, lagartos o cualquier animal de mayor tamaño).
 - **Zoom:** permite configurar el zoom de la cámara para encuadrar la superficie deseada.
 - **Frecuencia de muestreo:** determina el intervalo de tiempo entre un fotograma analizado y el siguiente a analizar. Es decir, si se establece en 1 segundo, la aplicación captará y analizará un fotograma cada segundo. Cuanto mayor sea el intervalo menor será el consumo de batería.
6. Una vez configurados los parámetros correctamente, se puede iniciar la monitorización pulsado el botón blanco.

7. Se iniciará una cuenta atrás y comenzará la monitorización. Durante esta, la pantalla puede estar apagada para ahorrar batería. Se puede aprovechar la cuenta atrás para apagarla sin influir en la monitorización (al manipular el móvil siempre se producen trepidaciones).
8. Cuando se desee detener la monitorización, se debe pulsar el botón cuadrado rojo. Una vez pulsado, se guardará la grabación y se podrá acceder a los detalles de esta.

Nota: Si se posee alguna aplicación de ahorro de batería es imprescindible añadir una excepción a la aplicación GoBees para que esta se pueda ejecutar en segundo plano sin restricciones. Si no, la aplicación puede ser cerrada durante la monitorización.



Figura E.7: Ajustes de monitorización.

Ver los detalles de una grabación

Al monitorizar una colmena se genera lo que denominamos una grabación. Una grabación contiene los datos de actividad de vuelo de la colmena.

Video-tutorial: <http://gobees.io/help/videos/ver-grabacion>

Para ver los detalles de una grabación:

1. Entrar en el colmenar al que pertenece la colmena monitorizada.
2. Entrar en la colmena.
3. Pulsar en la grabación sobre la que se está interesado.
4. Se mostrará una pantalla con dos gráficos.

- a. El gráfico principal muestra la actividad de vuelo. En el eje de las Y se representa el número de abejas en vuelo y en las X los instantes de tiempo. Si se pulsa sobre un punto del gráfico, se obtiene la medida exacta en ese punto.
 - b. El gráfico inferior muestra la información meteorológica. Existe un selector con tres botones: temperatura, precipitaciones y viento. Según se presione en uno u otro, se muestra su gráfico correspondiente.
5. Con ambos gráficos se puede interpretar la actividad de vuelo de la colmena y determinar si es una actividad normal o la colmena necesita una intervención.

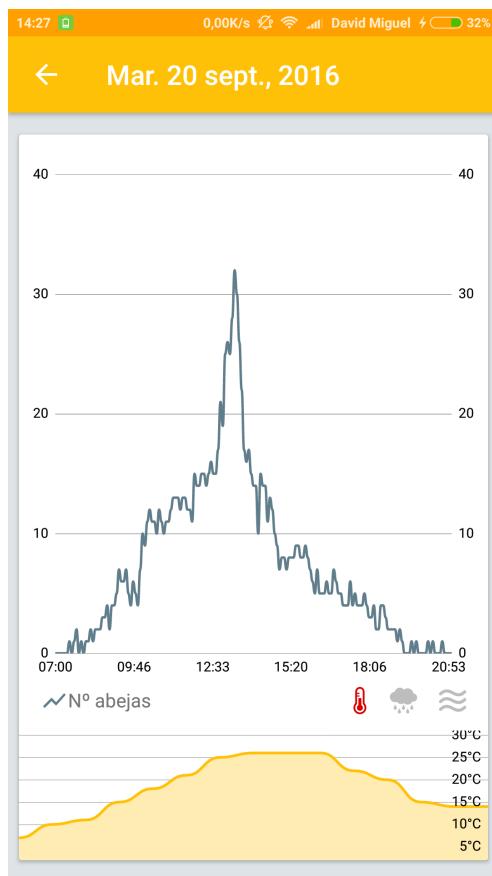


Figura E.8: Detalle de la grabación.

Eliminar una grabación

Al eliminar una grabación, se eliminan también todos los datos asociados a esta.

Video-tutorial: <http://gobees.io/help/videos/eliminar-grabacion>

Para eliminar una grabación existente:

1. Entrar en el colmenar al que pertenece la colmena monitorizada.
2. Entrar en la colmena.
3. Localizar la grabación y pulsar el botón de menú asociado a esta (tres puntos verticales situados en la esquina superior derecha).
4. Seleccionar la opción de eliminar.
5. La grabación se eliminará junto con toda su información.

Eliminar toda la información de la aplicación

Si por algún motivo se desea resetear toda la información almacenada en la aplicación, esta cuenta una opción para ello.

Video-tutorial: <http://gobees.io/help/videos/eliminar-datos>

Para eliminar toda la información de la aplicación:

1. Pulsar el botón menú.
2. Entrar en la sección “Ajustes”.
3. Seleccionar la opción “Borrar todos los datos”.
4. Todos los datos de la aplicación serán borrados. La aplicación volverá al mismo estado que cuando se instaló.

Consultar la información sobre la aplicación

Para conocer la versión instalada de la aplicación, los cambios introducidos en las diferentes versiones, la licencia o el autor de esta hay que acceder a la sección “Acerca de GoBees”.

Video-tutorial: <http://gobees.io/help/videos/acerca-gobees>

Para acceder a la sección “Acerca de GoBees”:

1. Pulsar el botón menú.
2. Entrar en la sección “Acerca de GoBees”.
3. En ella se puede visualizar la versión de la aplicación, el autor y las bibliotecas utilizadas para su desarrollo.
4. Si se presiona el botón “Website” se accede a la página web de GoBees.
5. Si se presiona el botón “Licencia” se visualiza una copia de la licencia de la aplicación.
6. Si se presiona el botón “Changelog” se visualizan los cambios introducidos en cada versión.



Figura E.9: Sobre GoBees.

Bibliografía

- [1] Android. Compilar y ejecutar tu app, 2017. URL <https://developer.android.com/studio/run/index.html?hl=es-419>. [Online; Accedido 24-Enero-2017].
- [2] Android. Importing an existing android project, 2017. URL <https://www.jetbrains.com/help/idea/2016.3/importing-an-existing-android-project.html>. [Online; Accedido 24-Enero-2017].
- [3] CodeClimate. User documentation, 2017. URL <https://docs.codeclimate.com/>. [Online; Accedido 24-Enero-2017].
- [4] CodeClimate. /davidmigloz/go-bees, 2017. URL <https://codeclimate.com/github/davidmigloz/go-bees>. [Online; Accedido 24-Enero-2017].
- [5]Codecov. User documentation, 2017. URL <https://docs codecov io/>. [Online; Accedido 24-Enero-2017].
- [6] Martin Fowler. Gui architectures - mvc and mvp, 2006. URL <https://martinfowler.com/eaaDev/uiArchs.html>. [Online; Accedido 22-Enero-2017].
- [7] Git. Git, 2017. URL <https://git-scm.com/>. [Online; Accedido 24-Enero-2017].
- [8] GitHub. Cloning a repository, 2017. URL <https://help.github.com/articles/cloning-a-repository/>. [Online; Accedido 24-Enero-2017].
- [9] Google. Material design guidelines, 2014. URL <https://material.io/guidelines/>. [Online; Accedido 24-Enero-2017].
- [10] Google. Manage your project, 2016. URL <https://developer.android.com/studio/projects/index.html?hl=es>. [Online; Accedido 24-Enero-2017].

- [11] Google. Android studio, 2017. URL <https://developer.android.com/studio/index.html?hl=es>. [Online; Accedido 24-Enero-2017].
- [12] Java. Java se development kit 7, 2017. URL <http://www.oracle.com/technetwork/es/java/javase/downloads/jdk7-downloads-1880260.html>. [Online; Accedido 24-Enero-2017].
- [13] Stephan Linzner y Mustafa Kurtuldu Jose Alcérreca. Google en android architecture blueprints. URL <https://github.com/googlesamples/android-architecture>. [Online; Accedido 22-Enero-2017].
- [14] David Miguel Lozano. Aplicaciones de etiquetado y desarrollo del algoritmo, 2017. URL <https://github.com/davidmigloz/go-bees-prototypes/releases>. [Online; Accedido 24-Enero-2017].
- [15] David Miguel Lozano. Releases gobees, 2017. URL <https://github.com/davidmigloz/go-bees/releases>. [Online; Accedido 24-Enero-2017].
- [16] Edward Hieatt y Rob Mee Martin Fowler. Repository pattern. URL <https://martinfowler.com/eaaCatalog/repository.html>. [Online; Accedido 22-Enero-2017].
- [17] OpenCV. Opencv official website, 2017. URL <http://opencv.org/>. [Online; Accedido 24-Enero-2017].
- [18] SonarQube. User documentation, 2017. URL <https://docs.sonarqube.org/>. [Online; Accedido 24-Enero-2017].
- [19] SonarQube. /davidmigloz/go-bees, 2017. URL <https://sonarqube.com/dashboard?id=davidmigloz/go-bees>. [Online; Accedido 24-Enero-2017].
- [20] Read the Docs. User documentation, 2017. URL <http://docs.readthedocs.io/en/latest/>. [Online; Accedido 24-Enero-2017].
- [21] Travis. User documentation, 2017. URL <https://docs.travis-ci.com/>. [Online; Accedido 24-Enero-2017].
- [22] Travis. davidmigloz / go-bees, 2017. URL <https://travis-ci.org/davidmigloz/go-bees>. [Online; Accedido 24-Enero-2017].
- [23] VersionEye. User documentation, 2017. URL <https://www.versioneye.com/documentation>. [Online; Accedido 24-Enero-2017].
- [24] VersionEye. /davidmigloz/go-bees, 2017. URL <https://www.versioneye.com/user/projects/57f7b19e823b88004e06ad33>. [Online; Accedido 24-Enero-2017].

- [25] Wikipedia. Dependency injection — wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/w/index.php?title=Dependency_injection&oldid=761366923. [Online; Accedido 07-Enero-2017].



Este obra está bajo una licencia Creative Commons Reconocimiento 4.0 Internacional ([CC-BY-4.0](#)).