Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

Gestión de Proyecto Software,
Departamento de Ingeniería e Informática de Sistemas,
Universidad de Zaragoza

Informe final del proyecto

Equipo 1: GimnasIO
Martínez Menéndez, Alberto
Navarro Castillo, Alejandro
Salueña Sediles, Asier
Sánchez Salvador, Darío

08 de enero, 2018

Resumen

En este documento se detalla el proceso seguido durante el desarrollo de la app para móviles GimnasIO, la aplicación que te permite disponer de todas tus rutinas en tu Smartphone. Cada vez mas gente hace uso de los gimnasios y un alto porcentaje de estas personas desconoce los fundamentos de la práctica deportiva sin tener claro los objetivos de los ejercicios que puede proponer el monitor, o como distribuirlos en su sesión de entrenamiento, GimnasIO pretende suplir este déficit y servir de soporte a todas aquellas personas que quieren iniciarse en el mundo de la práctica deportiva en un gimnasio.

Índice general

Capítulo 1

Introducción

El proyecto que se ha llevado a cabo es una aplicación móvil desarrollada para OS Android cuyo objetivo es servir de soporte en la practica deportiva en gimnasios, esta aplicación denominada **GimnasIO** permite a los usuarios visualizar una colección de ejercicios y consultar distinta información de los mismos (principal musculo trabajado, otros músculos secundarios una descripción de su ejecución y una imagen del desarrollo del mismo), además permite a partir de estos ejercicios crear una rutina de ejercicios con un objetivo y un numero de series y repeticiones así como el descanso que requieren.

Este proyecto ha sido desarrollado por el equipo 1 de la asignatura Gestión de Proyecto Software de la Universidad de Zaragoza, este equipo esta formado por cuatro miembros siendo tres de ellos estudiantes de la rama de *Ingeniería del Software* y el último de la rama de *Tecnologías de la Información*, lo que aporta dos visiones al proyecto, una versión mas precisa del proceso de ingeniería en el desarrollo de un software y la visión del mundo IT.

El documento consta de dos secciones principales además de la sección de introducción y la de conclusiones, en las dos secciones principales se detallan toda la información referente al producto en primer caso y al proceso de desarrollo del proyecto en segundo caso.

En la sección que detalla la información referente al producto, se encuentra la información sobre el plan de producto y la planificación de lanzamientos, el análisis de riesgos llevado a cabo por el equipo en el desarrollo del producto, la pila de producto planteada hasta la finalización del primer sprint detallando a continuación el estado actual de la aplicación (funcionalidad implementada, GUI, aspectos de usabilidad, problemas conocidos...), por último se explica la arquitectura de la aplicación así como detalles sobre implementación y despliegue de la misma.

En la sección que contiene la información del proceso de desarrollo se detallan las estrategias de testing, y de automatización de despliegue, la definición de hecho con las mejoras realizadas a la definición inicial tras la revisión del profesorado, una breve explicación del sistema de control de versiones utilizado, información sobre los esfuerzos realizados así como un diagrama del trabajo completado/-pendiente y un calculo de la velocidad del equipo, un análisis de resultados de la última retrospectiva y otros aspectos referentes a la gestión del proyecto que pudieran ser interesantes.

Capítulo 2

Producto

2.1. Primer Sprint

2.1.1. Plan de producto y planificación de lanzamientos

GimnasIO es una aplicación para Android que permite organizar las rutinas de gimnasio. La aplicación permite mantener una serie de rutinas guardadas, las cuales están compuestas de ejercicios seleccionados previamente durante su creación. Cada ejercicio viene acompañado de una explicación textual del mismo, así como de una imagen para que el usuario pueda observar cómo se realiza correctamente dicho ejercicio. La aplicación cuenta además con un reloj/contador para facilitar el control del tiempo de descanso o de la duración de determinados ejercicios.
Para la monetización de la aplicación se ha pensado la puesta en contacto con gimnasios y centros deportivos para que puedan crear sus rutinas privadas, permitiendo que sólo sus clientes tengan acceso mediante una clave a dichas rutinas más complejas y personalizadas. Los gimnasios y centros deportivos deberán abonar una suscripción para poder utilizar este servicio.
Los requisitos a alto nivel de la aplicación son:

- Se debe poder consultar la lista de ejercicios de la aplicación.
- Se debe poder crear una rutina de ejercicios, la cual consiste en un conjunto de ejercicios.
- Debe existir una forma de seguir una rutina en tiempo real (ejercicio actual, medición de tiempos de descanso, tiempo total, etc.)
- Se debe poder acceder a la aplicación sin conexión a Internet, pero limitando los servicios.
- La conexión a Internet será necesaria cuando se acceda por primera vez a la aplicación, haya alguna actualización de algún ejercicio o se acceda a una rutina privada de gimnasio.
- Debe ser ejecutable en dispositivos Android.

Con respecto a los lanzamientos, se han determinado varias fechas de lanzamiento:

- Una primera fecha de lanzamiento, el 24 de noviembre, que coincide con la finalización del primer sprint, cuya duración fue desde el 20 de octubre al 24 de noviembre de 2017. Para esta fecha de lanzamiento, el objetivo es presentar una aplicación con las funcionalidades básicas de modificación de rutinas (adición, edición, borrado), y la vista y descarga de ejercicios de ejercicios desde un servidor remoto.
- Una segunda fecha de lanzamiento, el 30 de enero de 2018, que coincide con la fecha de finalización del segundo sprint. En esta versión del producto, el objetivo es rehacer la GUI para que siga las guías de estilo y usabilidad de Android, implementar la gestión de rutinas, la gestión de rutinas premium para gimnasios y usuarios premium, e implementar la versión final del back-end.

2.1.2. Análisis de riesgos

Riesgo	Prob.	Daño	Justificación	Estrategia
Caída de la maquina donde se halla el servidor	Medio	Alto	Si el servidor no funciona, los clientes no pueden descargar nuevos ejercicios y deja la aplicación inservible o desactualizada.	Utilización del servicio de bases de datos relacionales (RDS) en AWS. Amazon se encarga de garantizar la fiabilidad de la base de datos.
Alguna de las funcionalidades podría cambiar en futuras versiones	Bajo	Alto	La aplicación podria no funcionar.	Especificar las versiones. No utilizar funciones deprecated. Informarse antes de utilizar una función.
Desconocimiento de las tecnologías	Bajo	Medio	En este proyecto se han empleado tecnologías con las que ningún miembro del equipo tiene experiencia	El equipo puede ver en cualquier momento como se encuentra el proyecto mediante Github. Además, se invirtieron más horas al principio del sprint para asegurar su finalización a tiempo.

Cuadro 2.1: Riesgos detectados

2.1.3. Pila de producto

Crear esqueleto de la app
Servidor Remoto
Base de datos local
Descarga de ejercicios
Ver colección de ejercicios
Ver ejercicio
Ver colección de rutinas
Ver rutina
Modificar rutina
Eliminar rutina
Búsquedas
Ejecutar rutina
Registro básico
Acceso premium
Ver rutinas premium
Crear rutina premium
Ver rutina premium
Modificar rutina premium
Eliminar rutina premium
Registro avanzado
Retroalimentación de la descarga

Cuadro 2.2: Pila de producto inicial.

Crear esqueleto de la app
Servidor Remoto
Base de datos local
Descarga de ejercicios
Ver colección de ejercicios
Ver ejercicio
Ver colección de rutinas
Ver rutina
Modificar rutina
Eliminar rutina
Actualización de la colección de ejercicios
Modificación de la creación de rutina
Modularización del código
Búsqueda e implantación de herramienta de despliegue automático
Búsquedas
Ejecutar rutina
Registro básico
Acceso premium
Ver rutinas premium
Crear rutina premium
Ver rutina premium
Modificar rutina premium
Eliminar rutina premium
Registro avanzado
Retroalimentación de la descarga

Cuadro 2.3: Pila de producto tras retrospectiva.

En azul entradas del primer sprint. En verde entradas del segundo sprint.

PBI: Crear esqueleto de la app

Puntos de historia : 8

Requisitos/Historias de usuario El usuario ha de ser capaz de poder navegar por las principales pantallas de la aplicación, entre ellas todas las relacionadas con la modificación de rutinas, tanto creación, como edición, borrado y ejecución de la mismas, o las destinadas a la visualización de ejercicios. El requisito es tener un conjunto de actividades interrelacionadas entre sí por las que el usuario pueda navegar de una a otra.

Criterios de aceptación:

1. Existe una actividad para cada pantalla de la aplicación, aunque carezcan de información.
2. Se puede navegar entre actividades,aunque sea de forma básica.
3. La actividad principal cuenta con una interfaz básica para navegar por las actividades.

PBI: Servidor Remoto

Puntos de historia : 13

Requisitos/Historias de usuario: El administrador o administradores del sistema tienen que almacenar información sobre los usuarios premium de la aplicación. En concreto, hay que tener almacenados la clave y la contraseña de todos los gimnasios que sean usuarios premium, y también la contraseña y usuario de todos los clientes de ese gimnasio para que puedan tener acceso a las así

rutinas específicas creadas por el gimnasio. Además, también hay que almacenar los ejercicios en el servidor para que los usuarios freemium puedan acceder a ellos, y descargarlos.

Criterios de aceptación

1. Existe una API REST contra la que realizar peticiones para obtener los servicios online requeridos por la aplicación.
2. Existe una base de datos accesible a través de la API con todos los datos de la aplicación.

PBI: Base de datos local

Puntos de historia : 8

Requisitos/Historias de usuario:

- El usuario de la aplicación debe de ser capaz de almacenar toda la información sobre ejercicios y sobre rutinas creadas por él.
- El usuario debe ser capaz de acceder a la información almacenada en la base de datos local, ya sea información sobre rutinas o ejercicios.
- El usuario debe ser capaz de borrar información sobre rutinas de la base de datos local.
- El usuario debe ser capaz de modificar información sobre rutinas de la base de datos.

Criterios de aceptación

1. El usuario puede almacenar rutinas creadas por él.
2. El usuario puede borrar rutinas creadas por él.
3. El usuario puede modificar rutinas creadas por él.
4. El usuario puede extraer información sobre rutinas, tanto creadas por él o por el gimnasio premium al cual está subscrito, y sobre ejercicios.

Información auxiliar (bocetos GUI, documentación adicional...)

Se incluye el diagrama de Entidad-Relación de la base de datos:

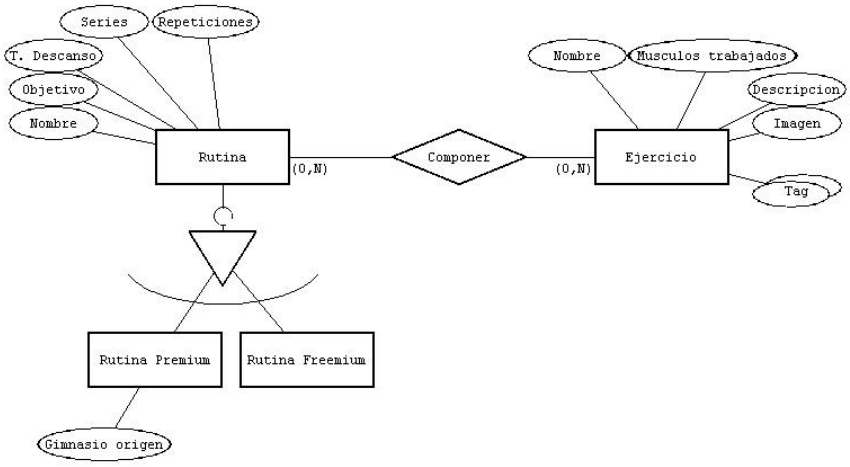

Figura 2.1: Diagrama Entidad-Relación de la BD Local

PBI: Descarga de ejercicios

Puntos de historia : 8

Requisitos/Historias de usuario:

- El usuario debe descargar una colección de ejercicios al iniciar la aplicación, los cuales se almacenarán en la base de datos local para su visualización posterior.
- El usuario podrá descargar una versión actualizada de la colección de ejercicios modificada por los administradores de la aplicación.

Criterios de aceptación:

1. Si no hay una BD local poblada en el dispositivo la app avisará al usuario de la descarga necesaria.
2. Antes de comenzar una descarga se pide permiso al usuario sobre la descarga de archivos al teléfono móvil.
3. Antes de comenzar una descarga se avisa al usuario del tamaño de la descarga.
4. Se le da al usuario la opción de comenzar la descarga o salir.

Información auxiliar (bocetos GUI, documentación adicional...)

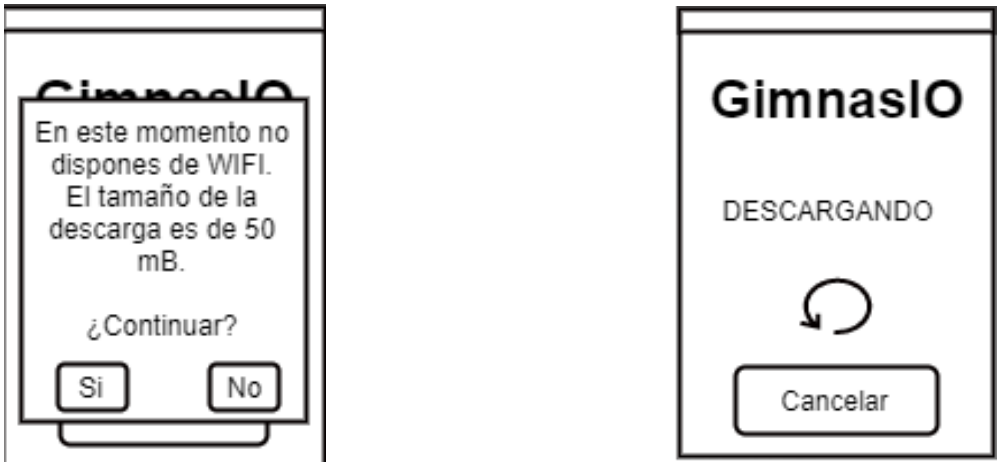

Figura 2.2: Boceto GUI

PBI: Ver colección de ejercicios

Puntos de historia : 13

Requisitos/Historias de usuario

- El usuario de la aplicación debe de ser capaz de poder acceder a una actividad donde se presente una lista de ejercicios almacenados en la base de datos local.
- Si no se ha realizado la descarga de ejercicios y la base de datos local está vacía, se mostrará una lista vacía.
- Si el usuario hace click en algún ejercicio, la aplicación le lleva a otra actividad dónde puede ver el ejercicio en más detalle.

Criterios de aceptación

1. Se puede consultar una lista de ejercicios.
2. Cada elemento de la lista mostrara el nombre del elemento y los tags asociados a él.
3. Se podrá hacer clic una vez en un elemento de la lista para acceder a otra actividad con información del mismo.

PBI: Ver ejercicio

Puntos de historia : 8
Requisitos/Historias de usuario

- El usuario de la aplicación podrá ver la información de los ejercicios en detalle: su nombre, músculo utilizado, tags asociados, imagen que describa cómo hacer el ejercicio, y una descripción que detalle cuál es la forma correcta de realizar el ejercicio.

Criterios de aceptación

1. Al ver un ejercicio aparecerán entre 1 y 5 tags que clasifican el ejercicio.
2. Al acceder al ejercicio se visualizará la imagen, músculos trabajados y descripción del mismo

Información auxiliar (bocetos GUI, documentación adicional...)

Figura 2.3: Boceto GUI

PBI: Ver rutinas

Puntos de historia: 8
Requisitos/Historias de usuario

- El usuario de la aplicación debe de ser capaz de poder acceder a una actividad donde se presente una lista de rutinas almacenadas en la base de datos local.
- Si la base de datos local no almacena ninguna rutina creada por el usuario, se mostrará una lista vacía.
- Si el usuario hace click en una rutina de la lista, aparecerá un menú con opciones para el tratamiento de rutinas, tales como modificar, ver o eliminar.

Criterios de aceptación

1. Se puede consultar una lista de rutinas.
2. Cada elemento de la lista mostrara el nombre del elemento y el objetivo asociado.
3. Se podrá hacer clic una vez en un elemento de la lista para acceder a un menú sobre el tratamiento del elemento pulsado.

Información auxiliar (bocetos GUI, documentación adicional...)

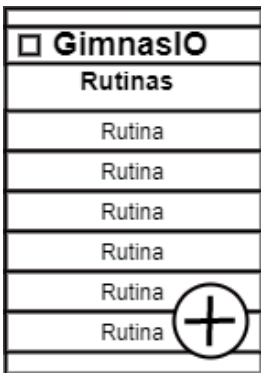

Figura 2.4: Boceto GUI

PBI: Crear rutina

Puntos de historia: 13
Requisitos/Historias de usuario

- El usuario de la aplicación debe de ser capaz de poder crear una rutina, añadiendo los valores deseados a los campos de la rutina, y almacenarse en la base de datos local para su tratamiento posterior.
- El usuario debe ser capaz de añadir todos los ejercicios deseados a la rutina.

Criterios de aceptación

1. Se puede crear una rutina y añadir ejercicios desde la lista.
2. Al crear una rutina se permite darle un nombre, una pequeña descripción y el objetivo de dicha rutina, así como el tiempo de descanso entre ejercicios, el número de repeticiones de los ejercicios, y las series de cada ejercicio.

PBI: Ver rutina

Puntos de historia: 8
Requisitos/Historias de usuario

- El usuario de la aplicación debe de ser capaz de poder ver los campos de cada rutina en más detalle: su nombre, descripción de la misma, objetivo de la rutina, tiempo de descanso entre ejercicios, repeticiones y series de los ejercicios, y una lista de los ejercicios asignados a la rutina.

Criterios de aceptación

1. Se podrá consultar una rutina una vez creada.
2. Una vez que se está consultando la rutina se podrá acceder a cada ejercicio individualmente para consultar su información.

Información auxiliar (bocetos GUI, documentación adicional...)

Figura 2.5: Boceto GUI

PBI: Modificar rutina

Puntos de historia: 5
Requisitos/Historias de usuario

- El usuario de la aplicación puede modificar cualquier campo de la rutina deseado, incluyendo la lista de ejercicios, y guardar sus cambios en la base de datos local.

Criterios de aceptación

1. Se podrá modificar la información y todos los elementos de una rutina una vez creada.

PBI: Eliminar rutina

Puntos de historia: 2
Requisitos/Historias de usuario

- El usuario de la aplicación puede modificar cualquier campo de la rutina deseado, incluyendo la lista de ejercicios, y guardar sus cambios en la base de datos local.

Criterios de aceptación

1. Se podrá modificar la información y todos los elementos de una rutina una vez creada.

2.1.4. Estado actual de la aplicación

Funcionalidad implementada

En el momento actual, con el primer sprint terminado y sin contar posibles bugs encontrados, la aplicación se encuentra finalizada en su versión mas simple, donde el usuario de la aplicación puede crear rutinas personalizadas y consultar los ejercicios almacenados. Sin embargo, se ha contemplado mejoras tanto en el apartado de creación de rutinas como una mejora visual que aporte una UX mejor que la que aporta actualmente.

Boceto inicial GUI

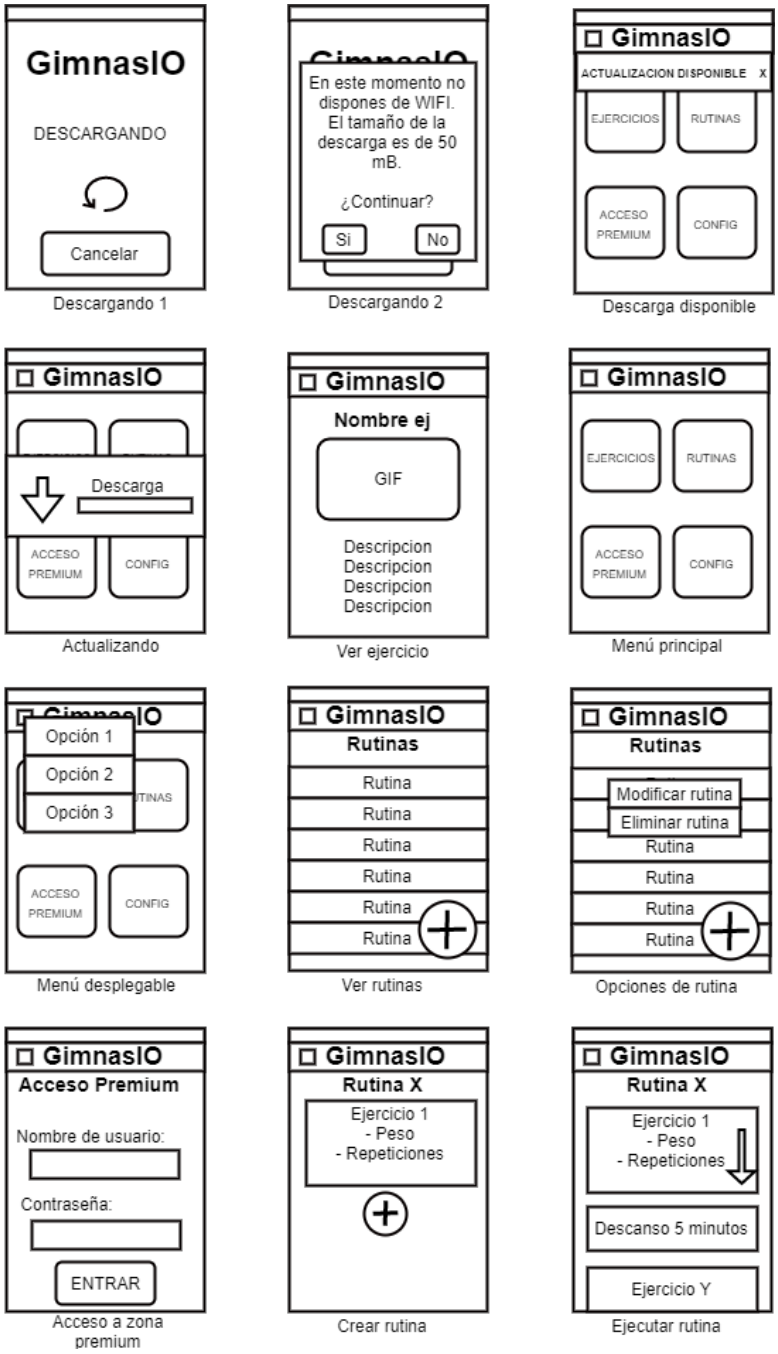

Figura 2.6: Boceto de la GUI

2.1.5. Aspectos de usabilidad

2.1.6. Arquitectura

Arquitectura: Servidor

El servidor está desarrollado en *Node.js* y ofrece una API Rest de la que se sirve la aplicación para las descargas y las funcionalidades online.

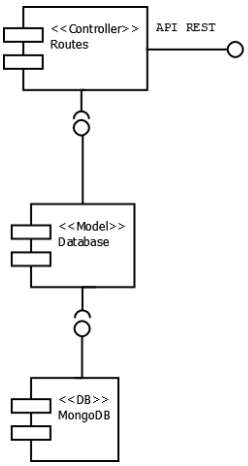

Figura 2.7: Diagrama CvC del Servidor

Este servidor ha sido desarrollado con una arquitectura por capas basada en Model-View-Controller, aunque la vista no tiene importancia en este primer sprint, por lo que no se ha representado en la 2.7. En 2.8 se puede apreciar que el servidor se encuentra desplegado sobre una única máquina EC2 en la nube de Amazon Web Services. En ella se encuentran tanto la base de datos *MongoDB* como la aplicación *Node.js* del servidor. Desde esa máquina se ofrecen los servicios API Rest que utiliza la aplicación, que a su vez almacena los datos en su propia base de datos *SQLite*.

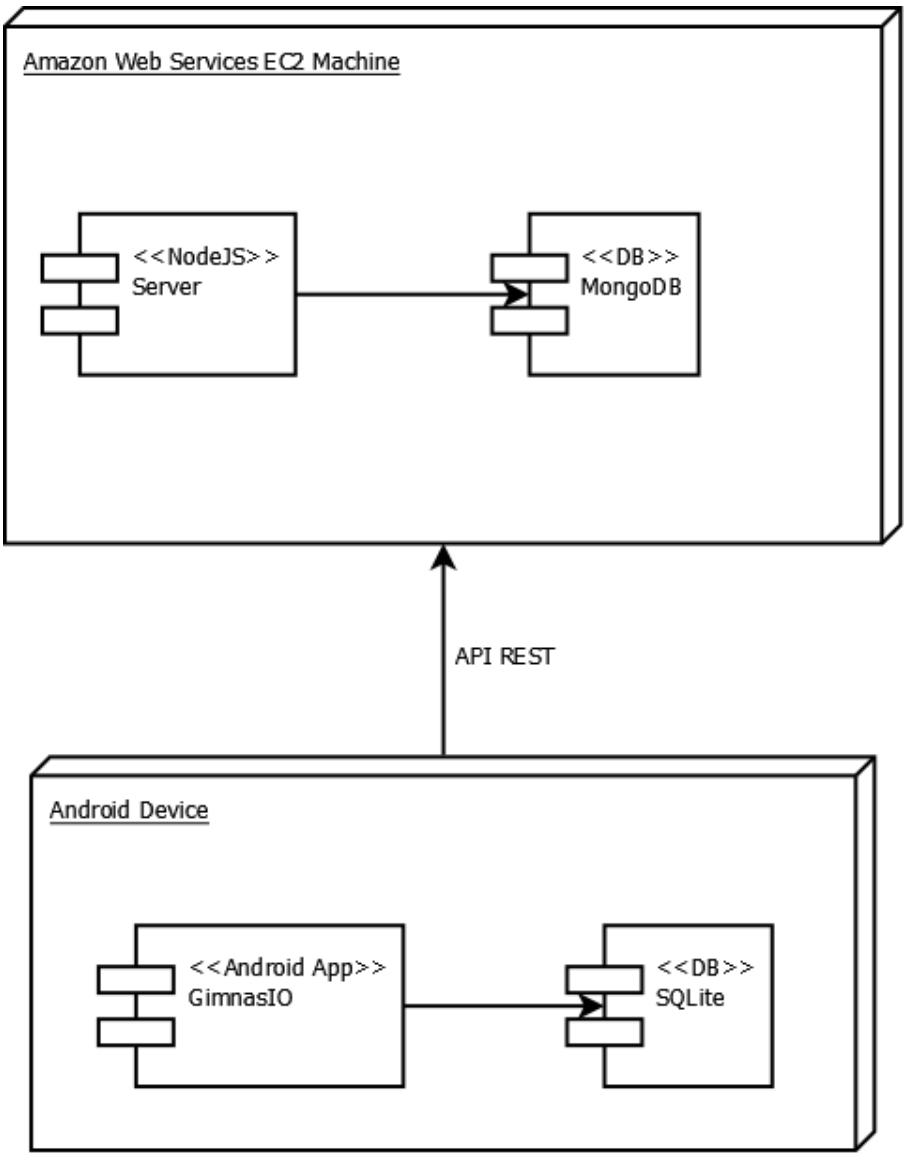

Figura 2.8: Diagrama de despliegue del servidor

La API cuenta con distintos métodos para consultar: ejercicios, rutinas, fechas de actualización, datos de almacenamiento y gimnasios registrados.

Ejercicios

```
/**
 * Type: GET
 * Name: /exercises/
 * Description: Returns a JSON containing every exercise on the
    ↪ Database.
 * Request:
 *     -Headers: Credentials
 *       -user: string
 *       -pwd: string
 * Responses:
 *      200:
 *          -JSON object containing multiple exercise objects:
 *              -id: string
 *              -name: string
 *              -muscle: string
 *              -description: string
 *              -images: [string]
 *              -tag: string
 *      404:
 *          -A feedback object
 *      500:
 *          -A feedback object
 */
```

Figura 2.9: Especificación de la API GET */exercises/*

```
/**
 * Type: GET
 * Name: /exercises/download
 * Description: Returns an image for the exercise given.
 * Request:
 *     -Headers: Credentials
 *       -image: string
 * Responses:
 *      success:
 *          -Sends the image desired
 *      404:
 *          -A feedback object
 */
```

Figura 2.10: Especificación de la API GET */exercises/download*

Datos Almacenamiento

```
/**
 * Type: GET
 * Name: dbdata/
 * Description: Returns a JSON containing the Database info.
 * Request:
 *      -user: string
 *      -pwd: string
 * Responses:
 *      200:
 *          -JSON object containing multiple db information:
 *              -dbSize: double (in MB)
 *              -imageSize: double (in MB)
 *              -totalSize: double (in MB)
 *              -lastUpdate: date
 *      404:
 *          -A feedback message
 *      500:
 *          -A feedback message
 */
```

Figura 2.11: Especificación de la API GET */dbdata/*

Rutinas

```
/**
 * Type: GET
 * Name: routines/
 * Description: Returns a JSON containing every routine on the
    ↪Database
 * for a given Gym access key.
 * Request:
 *      -nameGym: string
 *      -key: string
 *      -user: string
 *      -pwd: string
 * Responses:
 *      200:
 *          -JSON object containing multiple routine objects:
 *              -nameGym: string
 *              -name: string
 *              -objective: string
 *              -series: int
 *              -rep: int
 *              -relaxTime: int
 *      404:
 *          -A feedback object
 *      500:
 *          -A feedback object
 */
```

Figura 2.12: Especificación de la API GET */routines/*

Gimnasios

```
/**
 * Type: POST
 * Name: gym/newGym
 * Description: Inserts a new Gym in the database and returns
   ↪its keys.
 * Request:
 *     -Headers: Credentials
 *       -user: string
 *       -pwd: string
 *      Body:
 *          -user: string
 *          -pwd: string
 *          -nameGym: string
 * Responses:
 *      200:
 *          -JSON object containing access keys:
 *              -userKey: string
 *              -coachKey: string
 *      400:
 *          -A feedback object
 *      500:
 *          -A feedback object
 */
```

Figura 2.13: Especificación de la API POST */gym/newGym*

```
/**
 * Type: POST
 * Name: gym/newRoutine
 * Description: Inserts a new Routine in a specified Gym
   ↪collection.
 * Request:
 *      -Headers: Credentials
 *        -user: string
 *        -pwd: string
 *      -Body: A JSON routine object
 *       -user: string
 *       -pwd: string
 *       -nameGym: string
 *       -name: string
 *       -objective: string
 *       -series: string
 *       -rep: int
 *       -relaxTime: int
 *       -exercises: [string]
 * Responses:
 *       200:
 *           -A feedback object
 *       400:
 *           -A feedback object
 *       500:
 *           -A feedback object
 */
```

Figura 2.14: Especificación de la API POST */gym/newRoutine*

Actualización

```
/**
 * Type: GET
 * Name: /update/
 * Description: Returns the date of the last update.
 * Request:
 *      -Headers: Credentials
 *        -user: string
 *        -pwd: string
 * Responses:
 *       200:
 *           -A feedback message
 *       404:
 *           -A feedback object
 *       500:
 *           -A feedback object
 */
```

Figura 2.15: Especificación de la API GET */update/*

Arquitectura: Aplicación móvil

Para la aplicación se ha seguido la arquitectura descrita en 2.16. El componente *DBAdapter* se encarga de gestionar la base de datos ofreciendo interfaces. Para ello hace uso de los objetos de dominio que representan la información de la base de datos. Existe además un objeto *JSONHandler* que se encarga de ofrecer la transformación de los objetos en formato JSON que devuelve la API a objetos en formato *List*. Finalmente, las actividades se encargan de recoger todas estas funcionalidades y utilizarlas para representar los elementos por la pantalla.

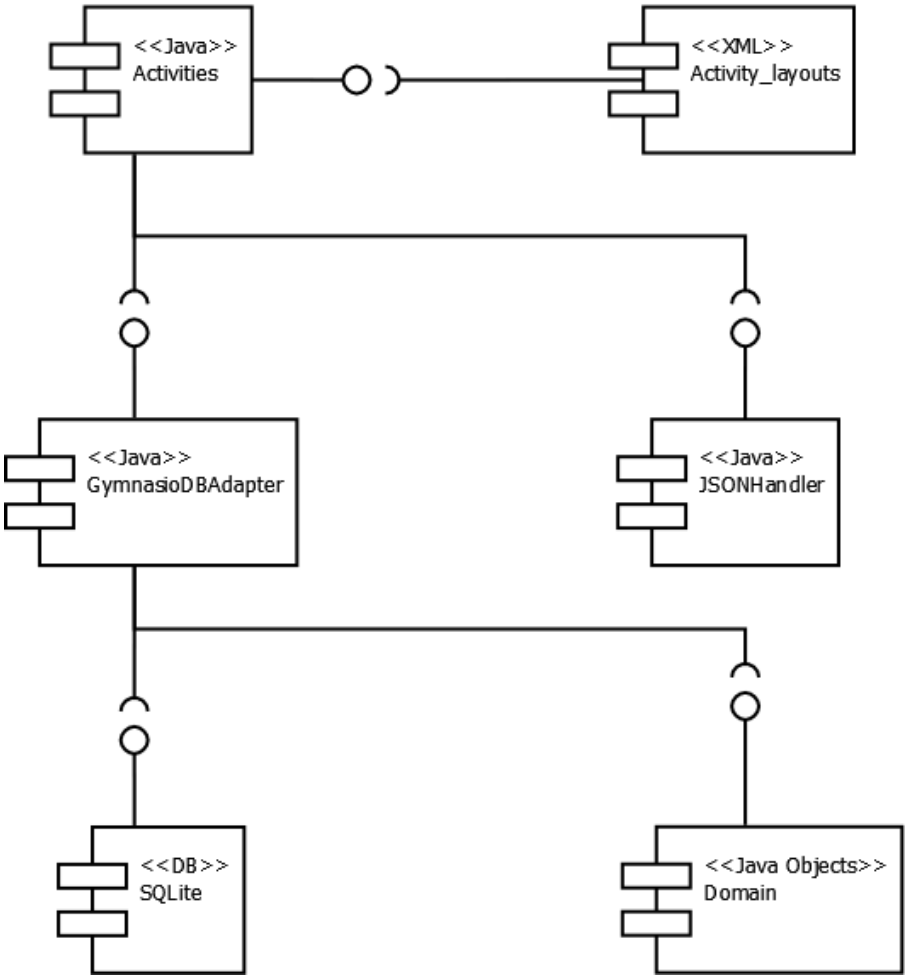

Figura 2.16: Diagrama CvC de la aplicación

2.2. Segundo Sprint

2.2.1. Arquitectura

Servidor

Se mantiene el esqueleto de la arquitectura del primer sprint, a la que se le añaden algunos componentes y se actualizan otros, de acuerdo con las necesidades de este segundo sprint.

En este segundo sprint se ha desarrollado una aplicación web implementada sobre la misma arquitectura, por lo que se ha pasado de un Modelo-Controlador donde la vista era la aplicación Android a un Modelo-Vista-Controlador completo, donde la vista es tanto la aplicación Android como la página web, siendo que los contenidos accesibles son distintos en cada una de ellas.

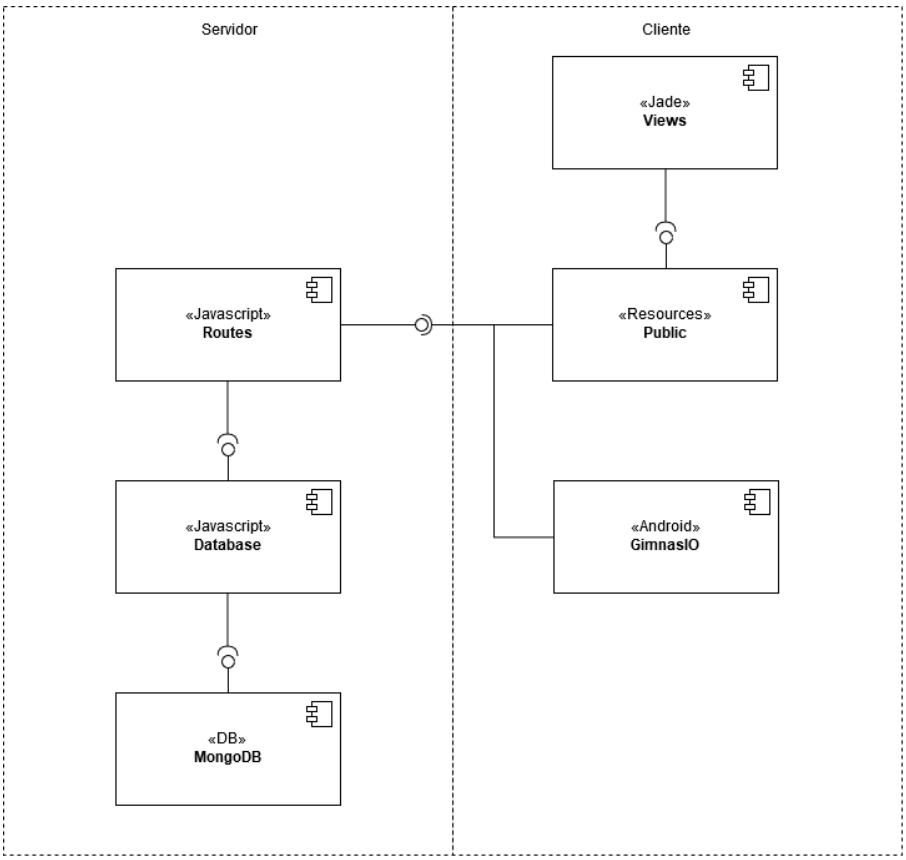

Figura 2.17: Diagrama CyC del servidor

Se mantiene la API Rest como método de comunicación del servidor con el exterior. En la parte de Cliente, la aplicación Android accede a la API Rest de forma nativa con librerías Java, mientras que para la web, se ha utilizado Javascript (jQuery) y Ajax para realizar las peticiones y controlar los contenidos de las páginas estáticas.

Además, se han realizado algunas modificaciones y añadido funcionalidades a la API existente. Estas son:

- Pedir rutinas: Modificada para no devolver repeticiones, series y tiempo de descanso para cada rutina, sino que esa información esté contenida en cada ejercicio.
- Actualizar rutina: Nueva entrada para permitir modificar repeticiones, series y tiempo de descanso para cada ejercicio.
- Nueva rutina: Nueva entrada para permitir establecer repeticiones, series y tiempo de descanso para cada ejercicio.
- Borrar rutina: Nueva entrada utilizada para borrar rutinas en el servidor.
- Nuevo gimnasio: Modificada para enviar las claves por email en vez de en la respuesta.
- Login gimnasio: Nueva entrada utilizada para comprobar las credenciales de entrada a la parte premium. Devuelve si son correctos o no, y en caso afirmativo, el tipo de usuario.
- Pedir ejercicios: Modificada para permitir devolver repeticiones, series y tiempo de descanso para cada ejercicio.

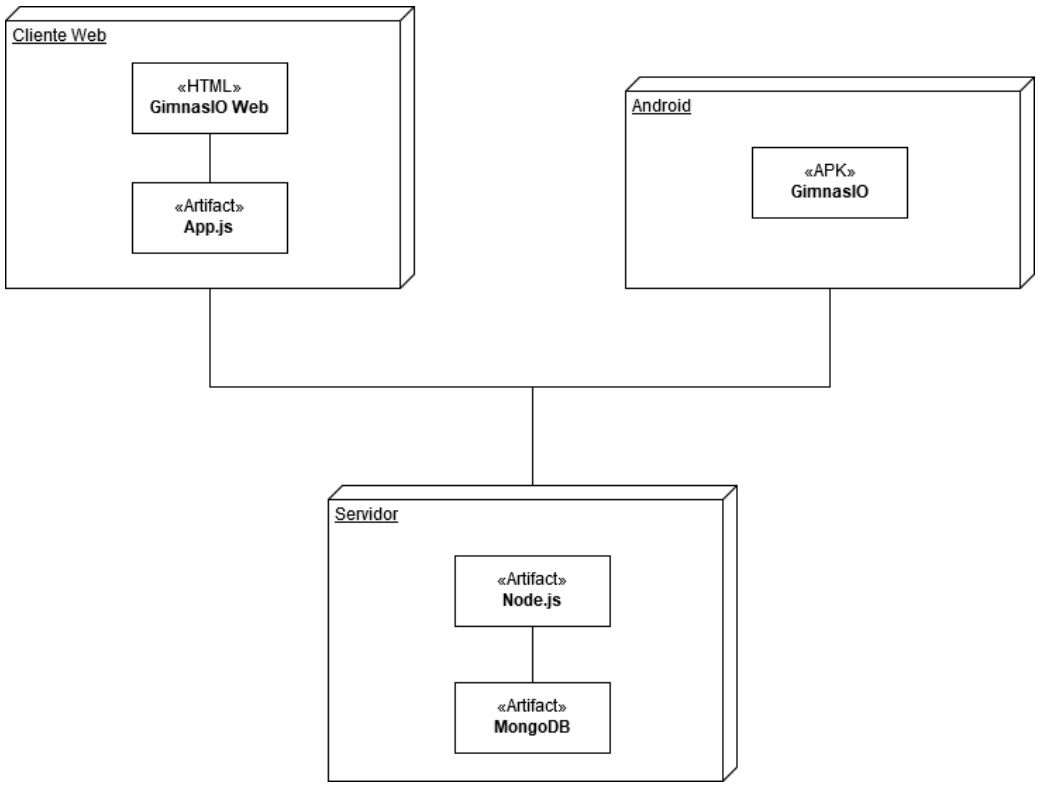

Figura 2.18: Diagrama de Despliegue

```
/**
 * Type: POST
 * Name: gym/newGym
 * Description: Inserts a new Gym in the database and returns
   ↪its keys.
 * Request:
 *      -Headers: Credentials
 *        -user: string
 *        -pwd: string
 *       Body:
 *           -nameGym: string
 *           -email: string
 * Responses:
 *       200:
 *           -A feedback object. Sends email to given address.
 *       400:
 *           -A feedback object
 *       500:
 *           -A feedback object
 */

 /**
 * Type: GET
 * Name: gym/login
 * Description: Returns success if they key given is correct
   ↪for the given gym.
 * Request:
 *      -Headers: Credentials
 *        -user: string
 *        -pwd: string
 *        -namegym: string
 *        -key: string
 * Responses:
 *       200:
 *           -A feedback object:
 *               -success: boolean
 *               -message: string
 *               -type: admin | user
 *       400:
 *           -A feedback object
 *       500:
 *           -A feedback object
 */
```

Figura 2.19: Especificación de la API gym

```
/**
 * Type: GET
 * Name: /exercises/
 * Description: Returns a JSON containing every exercise on the
    ↪ Database.
 * Request:
 *      -Headers: Credentials
 *        -user: string
 *        -pwd: string
 * Responses:
 *       200:
 *           -JSON object containing multiple exercise objects:
 *               -id: string
 *               -name: string
 *               -muscle: string
 *               -description: string
 *               -images: [string]
 *               -tag: string
 *               -series: int
 *               -repetitions: int
 *               -relaxTime: double
 *       404:
 *           -A feedback object
 *       500:
 *           -A feedback object
 */
```

Figura 2.20: Especificación de la API exercises

```
/**
 * Type: GET
 * Name: routines/
 * Description: Returns a JSON containing every routine on the
    ↪Database
 * for a given Gym access key.
 * Request:
 *      -nameGym: string
 *      -key: string
 *      -user: string
 *      -pwd: string
 * Responses:
 *      200:
 *          -JSON object containing multiple routine objects:
 *              -name: string
 *              -objective: string
 *              -exercises: [string]
 *      404:
 *          -A feedback object
 *      500:
 *          -A feedback object
 */

 /**
 * Type: POST
 * Name: routines/newRoutine
 * Description: Inserts a new Routine in a specified Gym
    ↪collection.
 * Request:
 *     -Headers: Credentials
 *       -user: string
 *       -pwd: string
 *       -nameGym: string
 *       -key: string
 *     -Body: A JSON routine object
 *      -name: string
 *      -objective: string
 *      -exercises: [string]
 *      -repetitions: [int]
 *      -series: [int]
 *      -relaxTime: [double]
 * Responses:
 *      200:
 *          -A feedback object:
 *              -success: true | false
 *              -message: A description of how the operation
    ↪went
 *              -id: The id of the routine created
 *      400:
 *          -A feedback object
 *      500:
 *          -A feedback object
 */
```

```
/**
* Type: PUT
* Name: routines/update
* Description: Updates the specified Routine and returns for a
   ↪ given Gym and returns a feedback object.
* Request:
*      -Headers: Credentials
*          -user: string
*          -pwd: string
*          -namegym: string
*          -key: string
*      -Body: A Routine object
*          -id: string
*          -name: string
*          -objective: string
*          -exercises: [string]
*          -repetitions: [int]
*          -series: [int]
*          -relaxTime: [double]
* Responses:
*      200:
*          -A feedback object
*      404:
*          -A feedback object
*      500:
*          -A feedback object
*/
```

Figura 2.21: Especificación de la API routines

Aplicación móvil

Se mantiene una vez más el esqueleto de la arquitectura del primer sprint. Sin embargo, en este sprint se han realizado esfuerzos para modularizar el código y permitir una mayor reusabilidad de métodos y objetos.

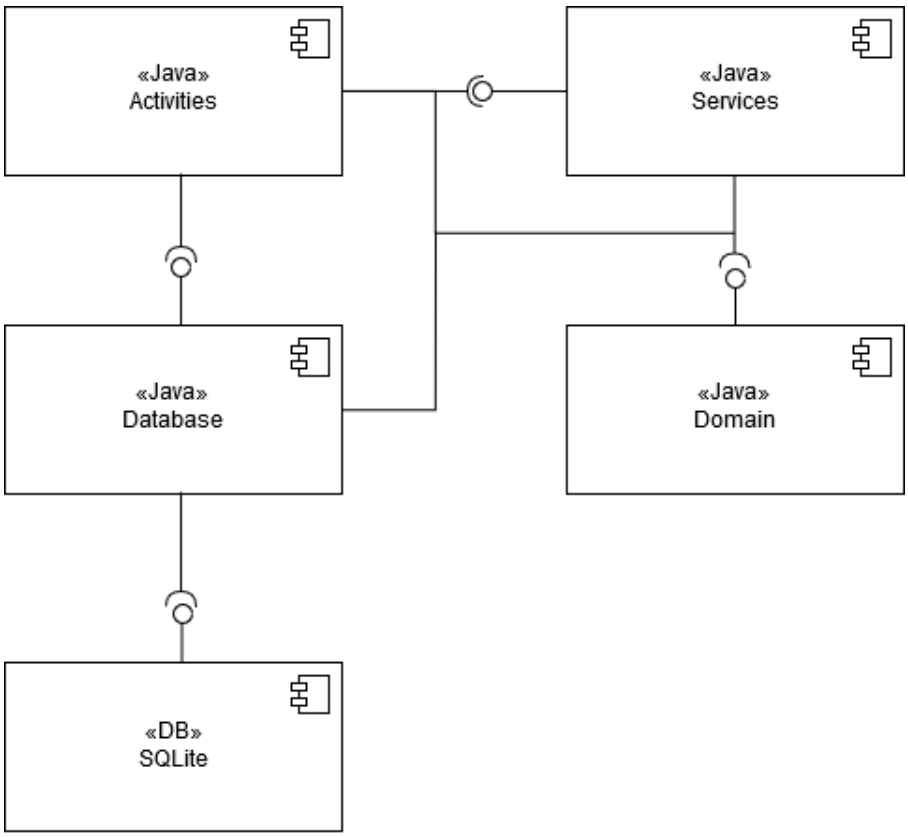

Figura 2.22: Diagrama CyC de la aplicación

Se pueden observar dos nuevos componentes *Services* y *Database*, uno completamente nuevo y el otro toma el lugar de `GymnasioDBAdapter`. Ambos son fruto de dicho proceso de modularización del código. Database contiene todos los objetos y métodos que interactúan con la base de datos SQLite. Por otro lado, Services representa los métodos encargados de interactuar con APIs externas, que en este caso es el servidor de GimnasIO.

Capítulo 3

Proceso

3.1. Primer Sprint

3.1.1. Estrategia y herramientas en test y cobertura de código

Se ha seguido la estrategia de crear como mínimo un test unitario por función de la aplicación que no se base en la interacción con el usuario. Por lo tanto se han testeado acciones como operaciones de acceso a la base de datos local y peticiones a la base de datos remota, pero no se han testeado el correcto funcionamiento de los botones o demás elementos interactuables de la aplicación. Los test que incluyan interacción con el usuario serán realizados en una etapa posterior del proyecto, cuando el diseño de las GUI este más cerca de ser el definitivo.

Para testear la aplicación se ha utilizado la librería JUnit, la cual permite la creación de test de una manera rápida y sencilla. Además es la librería con la que más miembros del equipo habían trabajado, por lo que se consideró la mejor opción.

Por tanto en este sprint se han testeado las operaciones en la base de datos local y las peticiones al servidor remoto, consiguiendo un cobertura del 82 % para el servidor remoto. La cobertura de la aplicación no se puede calcular debido a los problemas de integración con *Travis CI*.

3.1.2. Estrategia para la construcción automática de software

Debido a que el equipo utiliza Github como herramienta de control de versiones, se ha decidido aprovechar algunas de las herramientas de integración continua que son aplicables a Github. Se planteó desde el principio añadir *Travis CI* a los dos repositorios Github del equipo (el de la aplicación y el del servidor remoto) para que cada vez que hubiera una actualización de código en el repositorio, *Travis CI* pasara todos los tests para comprobar si el nuevo codigo era correcto.

Se ha conseguido añadir *Travis CI* al repositorio del servidor remoto de forma satisfactoria, pero no se ha conseguido añadir al repositorio de la aplicación. El equipo ha empleado muchas horas en este intento sin obtener ningún resultado correcto, por tanto se esta planteando la posibilidad de utilizar en el repositorio de la aplicación una herramienta distinta a Travis. Actualmente el equipo esta explorando alternativas a dicha herramienta.

3.1.3. Definición de hecho

Se ha tomado la decisión de que una tarea estará completa cuando cumpla la definición de hecho, que está conformada por los siguientes puntos:

- Todas las tareas de una entrada de la pila están completadas.
- Se han escrito y ejecutado test unitarios con al menos un 50 % de cobertura como resultado.
- Se ha comprobado que, al terminar las tareas de una entrada, el resto de test siguen dando un resultado satisfactorio.

- Se ha realizado una documentación de mínimos del diseño e implementación del código de cara a entender el funcionamiento de este, mediante especificaciones en el código y diagramas de diseño e implementación.
- Se ha probado su correcto funcionamiento en dos dispositivos Android diferentes: OnePlus One y Nexus 4 emulado. Se prueba en el emulador durante el desarrollo y en el dispositivo físico una vez finalizadas las tareas. Se probará tanto de manera automática corriendo los test, como de manera manual, utilizando las funcionalidades recién implementadas.

3.1.4. Estrategia de control de versiones

Para el control de versiones se ha creado una organización como se indicaba desde el profesorado de la asignatura, ademas dado que el proyecto se divide en dos entornos, el primero de ellos la propia aplicación Android y el segundo el servidor que provee los datos de ejercicios se decidió que se dividiría en dos repositorios independientes para evitar problemas entre ellos, sobre esos repositorios se estableció una política basada en forks donde cada miembro del equipo realizaba un fork del repositorio principal y actualiza el repositorio principal tras la finalización de un *issue* mediante un pull request que previa a su aceptación debe ser revisado por otro miembro, estableciendo un *pseudo-responsable* de cada subrepositorio, que en caso de duda es el responsable último de la decisión de aceptación.

3.1.5. Esfuerzos por persona

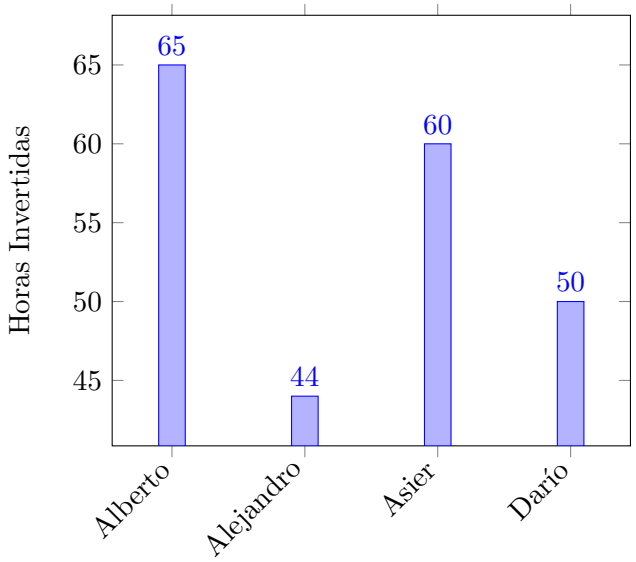

Figura 3.1: Distribución de horas invertidas en el primer Sprint.

3.1.6. Diagrama de trabajo completado/pendiente

3.1.7. Velocidad del equipo

A continuación en 3.2 y 3.3 se muestran la gráfica de seguimiento del sprint:

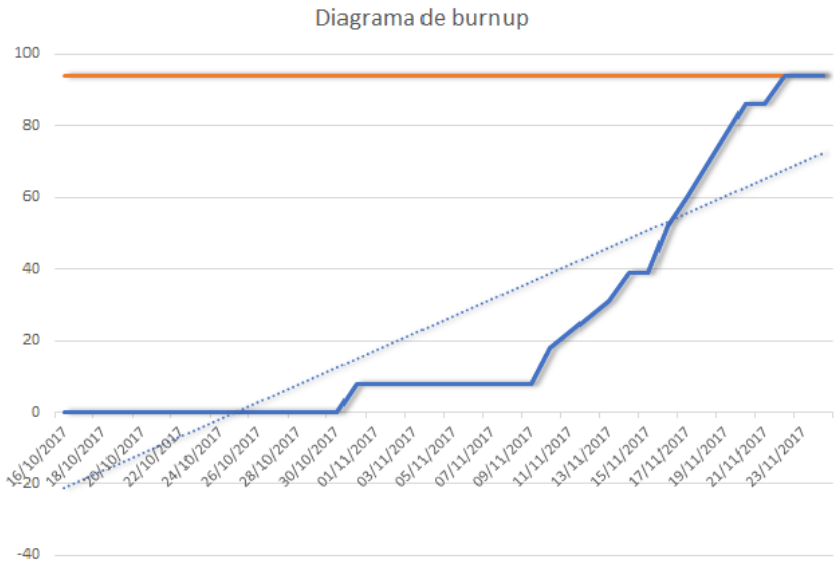

Figura 3.2: Diagrama Burnup

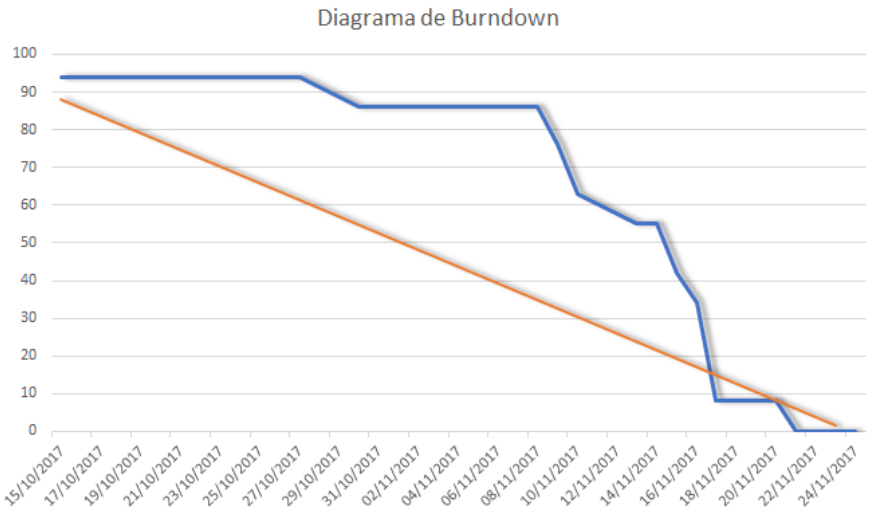

Figura 3.3: Diagrama Burndown

La velocidad del equipo ha sido de 94 P.H en este sprint.

3.1.8. Resultados de la última retrospectiva

En la reunión de retrospectiva con el profesorado se han extraído varias conclusiones:

- La escasa modularización del código ha dificultado el testeo.
- No se ha podido determinar la cobertura de código de la aplicación.
- Se necesitó reescribir el código de la base de datos local.
- Documentación de la API poco definida provoco confusiones en la implementación de la misma.
- Algunos miembros del equipo fueron reacios a modificar código ajeno a sus tareas.
- Una mala estimación de las PBI provocó que una entrada de la pila llegara al sprint demasiado grande y que fuera necesario fraccionarla y reestimarla.

- La falta de formación en el entorno Android afecto negativamente al tiempo y a las estimaciones.
- En algunos momentos del sprint el *Work in Progress* era demasiado alto.
- Desuso del tablero Scrum.
- No se consiguió integrar *Travis CI* con Android.

Tras una puesta en común entre los miembros del equipo de los problemas encontrados se ha determinado los tres problemas mas prioritarios a resolver en el siguiente sprint:

- **Modularización del código:**
  Se van a aplicar patrones Vista-Controlador para el desarrollo de las funcionalidades.
  Se ha decidido crear una PBI para solucionar este problema.
- **Desuso del tablero Scrum:**
  El ScrumMaster revisara el tablero al menos una vez a la semana supervisando que el tablero refleja la situación actual del equipo.
- **Integración de *Travis CI* con Android:**
  Se ha decidido buscar una herramienta alternativa a *Travis CI*.
  Se creará una PBI para solucionar este problema.

3.1.9. Otros aspectos de la gestión de proyecto y/o proceso de trabajo

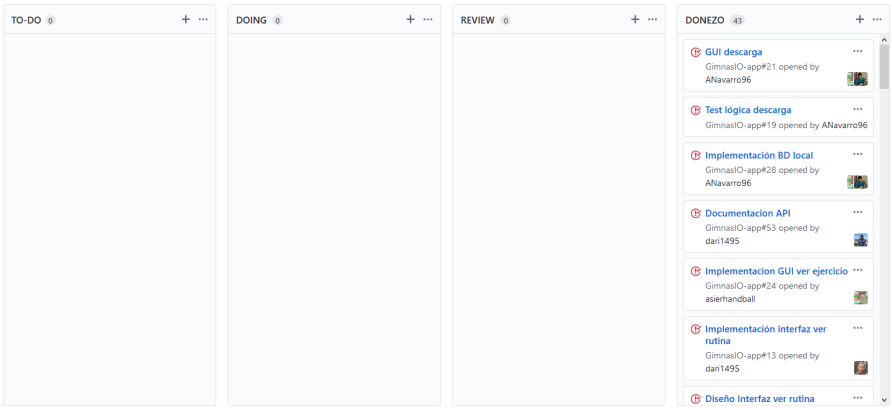

Figura 3.4: Tablero

Para la gestión del tablero del sprint se ha utilizado la herramienta de Github Projects. Para ello, se ha creado un proyecto en el perfil de la organización en el que se manejarán las tareas en un tablero formado por cuatro columnas:

- **TO-DO:** Es la columna de tareas que no se están realizando en este momento
- **DOING:** Tareas que actualmente se están desarrollando.
- **REVIEW:** Tareas que se han enviado en un PR para la aprobación del equipo. Una vez aprobadas, esas tareas se considerarán hechas. Este tablero, junto a DOING, representa el WiP del equipo.
- **DONE:** Tareas terminadas.

En el tablero se representan las tareas como tarjetas interactivas. Cada una de estas tarjetas es, en realidad, un Issue creado en uno de los repositorios de la organización. Esto nos permite sacar el máximo partido a la herramienta de Github projects, y es uno de los motivos por los que se eligió frente a las otras posibilidades. Esta herramienta nos permite además crear Milestones en la misma

pestaña Issues. Una Milestone representa un objetivo, y por tanto se le puede añadir una fecha de finalización y asignarle Issues previamente creados. Resulta entonces una herramienta perfecta para representar las entradas de la pila, con la fecha de finalización del sprint como objetivo y una serie de Issues que representan cada tarea de la entrada de la pila asignadas a ese Milestone. Además, conforme se van cerrando los Issues asignados a una Milestone, se puede ver el porcentaje de tareas completadas, aportando de esta manera un feedback visual que permite ver de un vistazo cuánto falta para finalizar cada entrada de la pila. El workflow deseado con esta implementación es tal que:

- Una o más personas se asignan una tarea de la columna TO-DO utilizando la herramienta Issues de Github.
- El o los encargados de esa tarea la mueven a la columna DOING cuando empiecen a trabajar en ella.
- Cuando consideran que está hecha, mandan un Pull Request al repositorio principal y la mueven a la columna REVIEW. El Pull Request cuenta en la descripción con una referencia a la Issue de la tarea completada. (P.Ej.: Cerrado issue #21)

Aquí se producen dos escenarios posibles:

- El Pull Request es aceptado, de forma que la tarea se cierra automáticamente gracias a la referencia. La tarjeta se mueve definitivamente a la columna DONE.
- El Pull Request es incorrecto, incompleto o incompatible con el contenido del repositorio. En ese caso, la tarjeta se mueve de vuelta a la columna DOING hasta que se esté listo para volverlo a enviar.

De esta forma, un vistazo al tablero permite ver el WiP actual, las tareas que quedan por realizar e incluso si hay algún Pull Request pendiente de aprobación. En este primer sprint este workflow ha sido ligeramente deficiente en la práctica. El alto requerimiento de atención que exige para mantenerse actualizado provoca que por un despiste de algún miembro del equipo el tablero se desactualice y pierda precisión, pudiendo provocar duplicidades de trabajo o confusión generalizada. El problema se produce cuando estos despistes se convierten en algo habitual, cosa que, por suerte, no ha llegado a ser realmente grave. De cara al segundo sprint se pretende realizar un énfasis en esta implementación. Los miembros del equipo serán recordados al menos una vez a la semana de mantener actualizadas sus tareas en el tablero, y, a su vez, se les exigirá una mayor atención hacia el tablero.

3.2. Segundo Sprint

Capítulo 4

Conclusiones

Se espera obtener un grado de al menos notable en este primer sprint. Para ello se han seguido las recomendaciones establecidas en la presentación de Evaluación del proyecto. Las conseguidas satisfactoriamente son:

- **El código del proyecto se aloja en Github y se trabaja de forma habitual contra él.** El equipo ha trabajado continuamente contra dos repositorios de Github.
- **El software funciona correctamente e implementa suficiente funcionalidad.** La app al final del primer sprint implementa correctamente las funcionalidades propuestas para este sprint.
- **Informe de proyecto suficiente y adecuado.** En este mismo documento se encuentran todas las secciones demandadas, así como la documentación del diseño y la implementación del sistema.
- **Cobertura de tests automáticos de al menos el 30 % del código.** El servidor tiene una cobertura del 82 % del código, esto se puede consultar en el repositorio del servidor en la medalla de Codecov presente en el readme.Pese a que no conocemos aún la cobertura del código Android, con la cantidad de test que ahora mismo se encuentran programados, es seguro decir que esa cobertura es perfectamente alcanzable. Se espera mejorar en el segundo sprint hacia más de un 50
- **Al menos el 50 % de las historias de usuario / requisitos tienen buenos criterios de aceptación.** Se ha trabajado de forma que todos los requisitos tengan buenos criterios de aceptación.
- **Definición de hecho clara.** La definición de hecho existente al final del primer sprint se considera suficientemente clara, pese a que de cara al segundo sprint se van a plantear algunas mejoras.
- **Compilación y gestión de dependencias.** A nivel de servidor, se ha utilizado npm para la gestión de dependencias y lanzamiento. A nivel de aplicación, se ha utilizado Gradle.
- **Se lleva un control de esfuerzos con horas dedicadas por persona a cada tarea.** Se han entregado ya distintas hojas de esfuerzos con el desglose por persona de las horas dedicadas y a que tareas.
- **Toda la documentación del proyecto está bajo control de versiones.** La documentación del proyecto se encuentra en el Dropbox del equipo, por lo que se considera que la gestión de versiones de este sistema es suficiente para esta tarea.

Además, se han realizado algunas de las tareas del sobresaliente:

- **Todas las historias de usuario / requisitos implementados tienen buenos criterios de aceptación.** Como se ha mencionado anteriormente, todos los requisitos del proyecto están dotados de buenos criterios de aceptación.

- **La construcción se ha automatizado más allá de la compilación y gestión de dependencias.** Este requisito se ha completado a medias, en la parte de servidor, la integración con TravisCI y Codecov permite no solo construir el código y sus dependencias, sino comprobar el funcionamiento de los test y realizar un informe incluyendo la cobertura de código. En el segundo sprint se trabajará para implementar esta funcionalidad también en el proyecto de la aplicación y ampliarla hacia la generación del APK automáticamente y el despliegue automático del servidor cuando haya cambios en el repositorio.