UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

PROJECT

CLEVERFIT

Grado en **Ingeniería Informática**

05/2017 (MAYO DE 2017)

Jose Luis Molina Vega - **autor**
Beatriz Correas Suárez - **tutora**

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

ÍNDICE

ESTADO ACTUAL **6**
Definición del proyecto 6
Motivación 6
Competidores 7
FITSTAR 7
FREELETICS 8
¿Por qué CLEVERFIT? 9
OBJETIVOS INICIALES **9**
JUSTIFICACIÓN DE COMPETENCIAS ESPECÍFICAS **10**
APORTACIONES **13**
Generación inteligente de rutinas deportivas de alta intensidad 13
DESARROLLO **16**
Planificación inicial y ajuste a la misma 16
Metodología aplicada 19
Fases de desarrollo 20
Fase de análisis 20
Fase de especificación 21
Fase de diseño 21
Fase del desarrollo del software 22
Fase de verificación del producto 22
REQUISITOS **23**
Análisis del producto 23
Descripción del producto 23
Par Producto/Mercado 24
Variables de mercado y análisis DAFO 25
Variables de mercado 25

Análisis DAFO 27
Business Model Canvas 28
Requisitos funcionales 29
Requisitos no funcionales 30
Tiempos de respuesta 30
Usabilidad 31
Desarrollo 31
Compatibilidad 31

DISEÑO **32**
Diseño del sistema 32
Arquitectura del sistema 32
Arquitectura de la base de datos (Entidad-Relación) 34
Historias de usuario 35
Casos de uso 36
Crear una cuenta 36
Modificar datos personales 37
Ver historial de progreso 38
Ver historial de entrenamiento 39
Ver plan de entrenamiento 39
Ver ejercicio 40
Solicitar nuevo entrenamiento 41
Comenzar entrenamiento 41
Cancelar entrenamiento 42
Pasar al siguiente ejercicio durante el entrenamiento 42
Pausar entrenamiento 43
Reanudar entrenamiento 43
Diseño de interfaz de usuario 44
Estructura de la página inicial 45
Estructura de la vista de plan de entrenamiento 46
Estructura de la vista de entrenamiento 47
Estructura de la vista de historial de rutinas 48
Estructura de la vista de progreso 49

Estructura de la vista de crear cuenta 50
Estructura de la vista de editar datos personales 51

MANUAL DE USUARIO 53
Registro 53
Generar entrenamiento 54
Comenzar entrenamiento 55
Asistencia durante el entrenamiento 56
Comprobar historial previo 57
Comprobar progreso actual 57
Ver rutina de entrenamiento actual 58
Ver ejercicio 59
Editar mis datos 60

DOCUMENTACIÓN DE DESARROLLO 61
Instalación, requisitos y dependencias utilizadas 61
El código tras Cleverfit 64
Arquitectura de la aplicación 64
El módulo Common 65
Los controladores base 65
Comandos de la aplicación: Patrón Command 67
Gestores de la base de datos: Realm 72
Gestores de la base de datos: CSV 77
Tratamiento de cadenas de texto 77
Cronómetro de entrenamiento 80
El núcleo: La Inteligencia Artificial tras Cleverfit 83

CONCLUSIONES 91
Resultados y grado de consecución de los objetivos 91

TRABAJOS FUTUROS, PLAN DE EXPANSIÓN 95
Estrategias de expansión 95
Software 95
Hardware 96

FUENTES DE INFORMACIÓN 98

Conocimientos propios 98
Fuentes web externas 98
Libros 98

AGRADECIMIENTOS 99

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

PROJECT CLEVERFIT

ESTADO ACTUAL

Definición del proyecto

Muchas personas desean iniciar un plan deportivo para bajar de peso pero no disponen del suficiente tiempo, y las aplicaciones móviles que prometen ser entrenadores personales carecen de una inteligencia propia lo suficientemente potente como para proporcionarla, prometiendo evoluciones prácticamente imposibles.

La solución es simple: desarrollar una aplicación con un núcleo de inteligencia artificial que genere entrenamientos cortos y de alta intensidad desde cero, al momento, considerando determinadas características del usuario, de forma que cada persona tenga un entrenamiento propio. Por estos motivos presentamos este proyecto para el desarrollo de una aplicación para iOS que genera entrenamientos de alta intensidad (HIT) en función de las necesidades del usuario.

Motivación

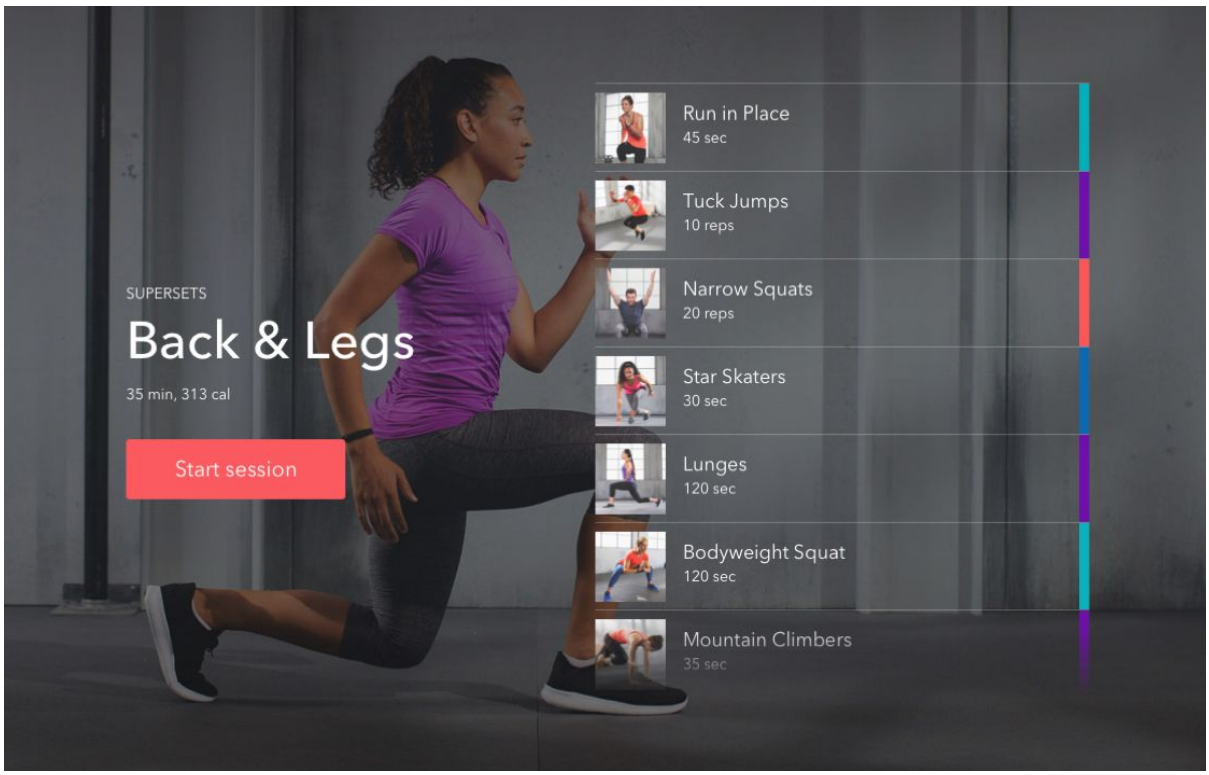
Hoy día a muchas personas que quieren mejorar su estado de salud, y una de las formas más asequibles es el ejercicio físico. Desde nuestro punto de vista, el ejercicio físico es un derecho universal, que puede mejorar la vida de las personas de forma inimaginable y de manera incuestionable. Realizar una aplicación que ayude a las personas a mejorar su salud es algo que no sólo resulta estimulante, sino que también resulta útil. Es por ello que consideramos

que esta aplicación podría ayudar a muchas personas a cumplir sus objetivos deportivos y, con ello, a incrementar su crecimiento personal.

Competidores

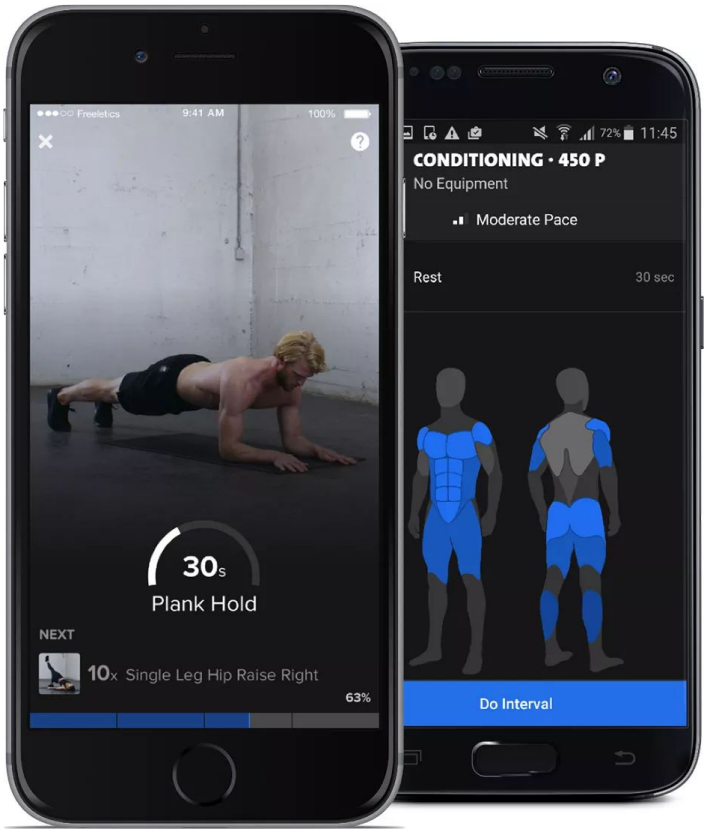
FITSTAR

Fitstar promete entrenamientos personalizados y adaptados. En cierto modo, esto es falso. Como el resto de aplicaciones, Fitstar depende de rutinas predefinidas, no generadas al momento. No puede existir un entrenamiento personalizado si la rutina no es adaptada al usuario.



¿Lo mejor de Fitstar? Es de Fitbit, y por tanto dispone de muchísimos más recursos que nosotros. La pulsera de Fitbit monitoriza la actividad diaria y lo envía a la aplicación. Por suerte, no será un problema a la larga debido a los Kits de desarrollo de salud de Apple, que permiten monitorizar las calorías quemadas y consumidas.

FREELETICS



A Freeletics le sucede lo mismo que a Fitstar, y es que sus rutinas están predefinidas. No hay una IA que las genere, y por tanto no están realmente adaptadas. ¿Lo mejor? Su interfaz. Freeletics muestra el índice de efecto de los ejercicios en los músculos del usuario. Aunque, en cierto modo, puede resultar incómodo. Freeletics no se enfoca en el tiempo, sino en la motivación. Por ello, su interfaz es más compleja que la de Cleverfit, y no ayuda al ahorro de tiempo.

¿Por qué CLEVERFIT?

Porque Cleverfit genera rutinas inteligentemente. En Cleverfit no existen rutinas predefinidas o preestablecidas que te son asignadas. En Cleverfit, se genera una rutina deportiva para ti, con tu experiencia y tu historial de entrenamiento previo. En Cleverfit sólo hemos necesitado entrenadores personales para definir la inteligencia artificial inicial, que podrá ser mejorada con el tiempo mediante nuevos módulos. Cleverfit no sólo genera rutinas contigo, sino que evoluciona contigo gracias a la arquitectura modular que se encuentra tras los cimientos de la aplicación.

OBJETIVOS INICIALES

Los objetivos inicialmente establecidos fueron los siguientes:

- Desarrollar una aplicación para iOS que permita a los usuarios mejorar su calidad de vida a través del ejercicio físico.
- Facilitar el mantenimiento de peso (o bajada del mismo) mediante entrenamientos de corta duración
- Desarrollar un núcleo inteligente básico que permita que todos los entrenamientos deportivos sean diferentes.

Además de estos objetivos, se han añadido el siguiente, que de hecho **se ha convertido en el objetivo principal** durante el desarrollo del proyecto:

- Desarrollar un código limpio, con el mínimo acoplamiento entre módulos y la máxima cohesión entre los mismos, aplicando los principios SOLID del desarrollo de Software.

JUSTIFICACIÓN DE COMPETENCIAS ESPECÍFICAS

Además de las competencias generales, se han adquirido las siguientes competencias específicas:

CÓDIGO	NOMBRE	JUSTIFICACIÓN
ISO1	Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.	Se ha realizado un estudio de requisitos previo al desarrollo, siendo estos asequibles de desarrollar y mantener, y respetando los principios SOLID de la ingeniería del software, explicados más adelante.
ISO2	Capacidad para valorar las necesidades del cliente y especificar los requisitos software para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia	Se ha realizado un estudio de mercado con el fin de comprender si la necesidad estaba actualmente satisfecha, y si la solución de adecuaba a unos límites de tiempo razonables. Así mismo, se ha comprobado previamente si el coste de desarrollo superaría el mínimo establecido. Al complir todos los

	de sistemas ya desarrollados y de las propias organizaciones.	objetivos, se pudo continuar con el proyecto.
ISO3	Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.	Se han estudiado los problemas de los clientes potenciales y se ha adecuado la solución a los mismos, estableciendo un proceso de diseño, desarrollo, implementación, verificación y, por supuesto, documentación. Todo este proceso se refleja en este documento.
ISO5	Capacidad de identificar, evaluar y gestionar los riesgos potenciales asociados que pudieran presentarse.	Se ha realizado un análisis de riesgos y amenazas prvio al desarrollo del proyecto. Este análisis se encuentra en el apartado de Análisis del Producto.
ISO6	Capacidad para diseñar soluciones apropiadas en uno o más dominios de aplicación utilizando métodos de la ingeniería del software que integren aspectos éticos, sociales, legales y económicos.	La solución implementada ayudará a las personas a bajar de peso, lo que inevitablemente mejorará su estado de salud. Esto incrementa el crecimiento personal del usuario. Realmente estamos comprometidos éticamente con cualquier usuario que use la aplicación, generando rutinas adecuadas a su experiencia y su historial de entrenamiento.
CII08	Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.	Se ha estudiado el lenguajes a utilizar, debido a que iOS dispone actualmente de dos lenguajes fuertemente divididos y con una sintaxis nada similar. Se ha elegido Swift como lenguaje principal debido a que se trata del lenguaje por el que actualmente apuesta Apple. Es muy robusto y ofrece muchas posibilidades, permitiendo realizar un código más limpio a niveles generales. Así mismo, se han aprovechado estas características y los conocimientos

		adquiridos durante la carrera para seleccionar una arquitectura que permitiera mejorar los tiempos de desarrollo y del futuro mantenimiento. Este análisis y diseño se encuentra detallado en el apartado de Análisis del producto
CII017	Capacidad para diseñar y evaluar interfaces persona computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas.	Se ha realizado un diseño básico de a aplicación antes de desarrollarla. Considerando que el usuario meido de la aplicación no quiere perder tiempo, se ha realizado una interfaz limpia y muy sencilla de manejar. Este apartado está explicado en la sección de diseño del presente documento.

APORTACIONES

Generación inteligente de rutinas deportivas de alta intensidad

Desde un punto de vista tecnológico, Cleverfit sienta una base de generación de rutinas deportivas mediante Inteligencia Artificial.

El mercado del fitness se encuentra en auge, eso no es ningún secreto. Además, la tecnología incrementa su potencia cada año que pasa. Existen muchas aplicaciones de fitness, sobretodo de ejercicio aeróbico. Estas aplicaciones permiten al usuario mejorar su condición física.

El único problema es que desde un punto de vista anaeróbico hay menos alternativas, y ninguna de ellas es capaz de generar rutinas conforme al historial de entrenamiento del usuario.

Cleverfit aporta una aplicación inteligente capaz de generar rutinas en función del historial de usuario registrado en la aplicación, así como de su experiencia. Además, utiliza el modelo de entrenamiento de alta intensidad (HIT) con el fin de conseguir mejores resultados generales a la hora de bajar de peso debido a los beneficios de este método de entrenamiento, que explicamos a continuación.

Este método consigue reunir lo mejor de dos mundos que, tradicionalmente, se consideran incompatibles: retener/ganar masa muscular y, al mismo tiempo, perder tejido adiposo. Este tipo de entrenamiento también consigue otro gran efecto: quemar más calorías y en menos tiempo que a través de entrenamiento de resistencia y cardiovascular convencional. De acuerdo con investigaciones recientes, que midieron y analizaron varios índices corporales relevantes, esta afirmación se fundamenta en la respuesta metabólica y hormonal del organismo ante este tipo de esfuerzo de alta intensidad.

El recurso al tradicional ejercicio cardiovascular de baja intensidad para quemar grasa se basa en el supuesto de que este tipo de rutina provoca una utilización muy significativa de la grasa corporal como combustible para el ejercicio. Sin embargo, los datos registrados en equipamientos de medición de rendimiento cardiovascular revelan que el ejercicio de baja

intensidad, ejecutado a cerca del 60% de la frecuencia máxima cardiaca, no es, en realidad, la mejor estrategia para quemar grasas.

De hecho, un paseo rápido o un trote ligero consiguen una quema de grasa corporal relativamente superior al consumo de glucógeno. No obstante, debemos distinguir la quema relativa y la quema total de grasa corporal, considerando que el índice fundamental para aquellos que se entrenan con el objetivo de perder grasa es el de la quema total de grasa corporal. A intensidades cardiacas más elevadas, la quema total de grasa corporal es ampliamente superior al resultado obtenido mediante ejercicios prolongados de baja intensidad.

Además, el Entrenamiento en Intervalos de Alta Intensidad aumenta la resistencia de los deportistas, permitiéndoles ejercitarse a intensidades muy elevadas durante periodos de tiempo más prolongados en comparación con el entrenamiento de baja intensidad.

Por último, el Entrenamiento en Intervalos de Alta Intensidad produce un efecto colateral extremadamente interesante, conocido como Consumo de Oxígeno Postejercicio (EPOC por sus siglas en inglés). El efecto del EPOC, generado por la aceleración del metabolismo creada por el HIT, permite que el cuerpo continúe quemando calorías extra por un período de hasta 24 horas tras el entrenamiento. En cambio, el efecto del EPOC producido por el ejercicio cardiovascular de baja intensidad es mínimo, quemando poquísimas calorías tras el entrenamiento.

Con respecto al efecto anabólico del HIT, también hay datos sólidos que demuestran su eficacia. En primer lugar, el HIT ha demostrado que aumenta el umbral de lactato, esto es, el nivel de tolerancia de los deportistas ante la acumulación de ácido láctico en el tejido muscular. Al aumentar el umbral de lactato, los deportistas pueden entrenar con mayor intensidad y durante más tiempo, lo que promueve el crecimiento muscular.

El HIT también mejora la sensibilidad a la insulina. Este efecto de optimización hormonal permite que la glucosa disponible en el organismo sea empleada rápidamente como combustible, en lugar de almacenarse en los tejidos adiposos. La optimización de la sensibilidad a la insulina también mejora la recuperación y la construcción muscular.

Por último, los datos obtenidos en estudios han permitido comprobar que el HIIT, en combinación con una dieta ligeramente más calórica que lo habitual, consigue activar los procesos anabólicos del organismo, originando un efecto de construcción muscular. Asimismo, los datos sugieren el efecto contrario en el ejercicio cardiovascular de baja intensidad, que acciona el efecto catabólico del cuerpo, promoviendo la degradación de tejido muscular para ser empleado como energía.

DESARROLLO

Planificación inicial y ajuste a la misma

La planificación inicial establecida fue adoptada debido a que aplicaba una metodología de desarrollo en cascada:

FASES	DURACIÓN ESTIMADA (HORAS)	ACTIVIDADES
FASE DE ANÁLISIS	50	Análisis del producto
		Análisis de requisitos
FASE DE ESPECIFICACIÓN	20	Especificación completa del producto
		Comportamiento esperado
FASE DE DISEÑO	50	Diagramas que reflejen el diseño del sistema
		Diagramas que reflejen la interacción con el usuario
FASE DE DESARROLLO DEL SOFTWARE	145	Desarrollo del núcleo
		Desarrollo del software (traducción de diseño a código)
FASE DE VERIFICACIÓN DEL PRODUCTO.	20	
ELABORACIÓN MEMORIA	15	
TOTAL HORAS	300	

La adaptación a la misma fue casi total, pero el número de horas varió considerablemente entre las diferentes etapas, que ahora puede comprobarse que se encuentran más detalladas debido a que se añadieron nuevas subetapas:

FASES	DURACIÓN ESTIMADA (HORAS)	ACTIVIDADES
FASE DE ANÁLISIS	70	Análisis del producto
		Análisis del mercado
		Análisis de requisitos
		Análisis de casos de uso
FASE DE ESPECIFICACIÓN	20	Especificación completa del producto
		Especificación de requisitos
		Especificación de casos de uso
		Especificación de historias de usuario
FASE DE DISEÑO	60	Diagramas que reflejen el diseño del sistema
		Diagramas que reflejen la interacción con el usuario
FASE DE DESARROLLO DEL SOFTWARE	100	Desarrollo del software
FASE DE VERIFICACIÓN DEL PRODUCTO.	10	Testeo del producto
ELABORACIÓN MEMORIA	50	
TOTAL HORAS	300	

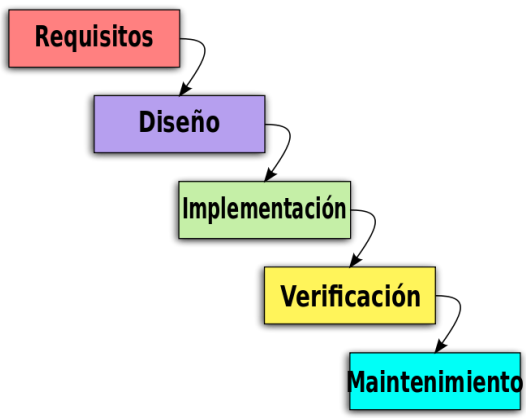
La etapa de desarrollo disminuyó muy considerablemente gracias a las etapas de análisis y especificación, que facilitaron enormemene el desarrollo del producto. La fase de análisis, inicialmente planeada como análisis de producto y de requisitos, incluyó un pequeño análisis de mercado y de casos de uso. Éste último disminuyó consiguió mantener la estimación de la etapa de especificación.

La etapa de especificación, inicialmente planeada como especificación del producto y de los requisitos, se extendió por tanto con la inclusión de la especificación de los casos de uso y de las historias de usuario. Estos dos últimos facilitaron la etapa de desarrollo, que disminuyó cuantiosamente las horas estimadas.

Metodología aplicada

Se ha utilizado una metodología de desarrollo en cascada en este proyecto. Si bien es cierto que no es la preferida de muchos, sí lo es el hecho de que es la más usada. Al tener conocimiento sobre el dominio, esta metodología resulta adecuada para este proyecto.

En Ingeniería de software el desarrollo en cascada, también llamado modelo en cascada (denominado así por la posición de las fases en el desarrollo de esta, que parecen caer en cascada "por gravedad" hacia las siguientes fases), es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la



finalización de la etapa anterior. Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de todos los demás modelos de ciclo de vida.

Un ejemplo de una metodología de desarrollo en cascada es:

1. Análisis de requisitos.
2. Diseño del sistema.
3. Diseño del programa.
4. Codificación.
5. Pruebas.
6. Implementación del programa.
7. Mantenimiento.

Esta metodología, por otro lado dispone de las siguientes ventajas:

- Realiza un buen funcionamiento en equipos débiles y productos maduros, por lo que se requiere de menos capital y herramientas para hacerlo funcionar de manera óptima.
- Es un modelo fácil de implementar y entender.
- Está orientado a documentos.
- Es un modelo conocido y utilizado con frecuencia.
- Promueve una metodología de trabajo efectiva: Definir antes que diseñar, diseñar antes que codificar

Fases de desarrollo

Cabe destacar que todas las fases se encuentran detalladas en el documento. Cada una de las fases tiene su propio apartado.

Fase de análisis

En esta fase se analizan las necesidades de los usuarios finales del software para determinar qué objetivos debe cubrir. De esta fase surge una memoria llamada SRD (documento de especificación de requisitos), que contiene la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos.

Es importante señalar que en esta etapa se debe consensuar todo lo que se requiere del sistema y será aquello lo que seguirá en las siguientes etapas, no pudiéndose requerir nuevos resultados a mitad del proceso de elaboración del software de una manera.

Esta fase se encuentra detallada en el apartado de requisitos del presente documento.

Fase de especificación

Es la tarea de describir detalladamente el software a ser escrito, de una forma rigurosa. Se describe el comportamiento esperado del software y su interacción con los usuarios y/o otros sistemas.

Esta fase se encuentra detallada en el apartado de requisitos del presente documento.

Fase de diseño

Se realiza la descripción de la estructura relacional global del sistema y la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras.

Es conveniente distinguir entre diseño de alto nivel o arquitectónico y diseño detallado. El primero de ellos tiene como objetivo definir la estructura de la solución (una vez que la fase de análisis ha descrito el problema) identificando grandes módulos (conjuntos de funciones que van a estar asociadas) y sus relaciones. Con ello se define la arquitectura de la solución elegida. El segundo define los algoritmos empleados y la organización del código para comenzar la implementación.

También se realiza el diseño de la interfaz de usuario.

Esta fase se encuentra detallada en el apartado de diseño del presente documento

Fase del desarrollo del software

Es la fase en donde se implementa el código fuente, haciendo uso de prototipos así como de pruebas y ensayos para corregir errores.

Dependiendo del lenguaje de programación y su versión se crean las bibliotecas y componentes reutilizables dentro del mismo proyecto para hacer que la programación sea un proceso mucho más rápido.

Esta fase se encuentra detallada en el apartado de manual de usuario y software del presente documento.

Fase de verificación del producto

En esta fase, se realizan diversas pruebas para comprobar que todo funciona correctamente.

REQUISITOS

Análisis del producto

Descripción del producto

Antes de definir los requisitos, debemos analizar las necesidades del productos y de los clientes potenciales. Es por ellos que realizaremos un pequeño análisis del mismo, así como del mercado en el que nos sumergimos.

A continuación se describirá ligeramente el producto y el mercado objetivo, así como las posibles ventajas competitivas.

DESCRIPCIÓN DEL PRODUCTO EN UNA LÍNEA
App multiplataforma de entrenamiento personalizado para personas con poco tiempo disponible.

Mercado objetivo - actividad del negocio

Consumidores de aplicaciones de ejercicio físico para iOS, que dispongan de poco tiempo tiempo para realizar ejercicio físico.

Tecnología - ventajas del producto/servicio

Nuestra aplicación carece de rutinas predefinidas. Es nuestra Inteligencia Artificial la que se encargará de **generar planes deportivos de alta intensidad conforme a las necesidades del usuario**. Las rutinas dependerán de su historial de entrenamiento previo, su experiencia deportiva, su objetivo (que en esta iteración del proyecto se centrará exclusivamente en la pérdida de peso) y el tiempo que disponga para entrenar, desde 5 hasta 30 minutos.

Clientes clave

Personas de clase media-alta, residentes en ciudades, de entre 27-49 años, preferentemente trabajadores, que no puedan realizar actividad física de forma habitual (menos de 3 veces por semana) por falta de tiempo y que deseen mejorar su salud a través del ejercicio físico.

Alianzas/acuerdos posibles de cooperación clave

Federaciones deportivas, grandes superficies y deportistas de élite.

Nivel de competencia esperado

Competidores clave: Freeletics, Fitstar, 8fit.

Esperamos un incremento considerable de nuestra competencia en los meses siguientes al lanzamiento. Ésto se debe a que el mercado de Fitness se encuentra totalmente en auge.

Par Producto/Mercado

Cabe destacar que todo análisis de producto, por mínimo que sea, debería tener un ligero análisis del par producto/mercado. Antes de poder desarrollar este proyecto, se debía comprobar su viabilidad. Es por ello que se decidió realizar este análisis. No sólo nos permitía poder ir comprobando si el producto era viable o no, sino también si tenía un hueco en el mercado. Gracias a este análisis logramos describir a nuestro público objetivo presente y futuro:

PAR PRODUCTO / MERCADO	POSICIONAMIENTO COMPETITIVO	
	PRESENTE	FUTURO
ÁMBITO DEL PRODUCTO	Aplicaciones del cuidado de la salud y la forma física	Aplicaciones del cuidado de la salud y la forma física **y redes sociales**
SEGMENTACIÓN DEL MERCADO POR CLIENTES	Personas de clase media-alta, residentes en ciudades, de entre 27-49 años, trabajadores, que no realicen actividad física de forma habitual (menos de 3 veces por semana) por falta de tiempo y que deseen mejorar su salud a través del ejercicio físico.	Personas **de clase media**, residentes en ciudades, de entre **21-49 años**, trabajadores **o estudiantes**, que no realicen actividad física de forma habitual (menos de 3 veces por semana) por falta de tiempo y que deseen mejorar su salud a través del ejercicio físico.

Variables de mercado y análisis DAFO

Variables de mercado

¿Qué pasaría si fuéramos a publicar la aplicación en el mercado? Con el siguiente cuadro se demuestra que disponemos de suficientes oportunidades para hacerlo, con muy pocas amenazas en relación.

Variables de mercado	Breve descripción	Oportunidad o amenaza
Tamaño del mercado	Grande	**O**
Estructura del mercado, cuota de mercado	Hay un líder (Runtastic), el resto está muy repartido.	**A**
Evolución del mercado	Creciente	**O**
Segmentación del mercado	Mercado muy segmentado. Nuestro segmento es aún muy joven.	**O**
Grado de internacionalización	Alto	**O/A**
Proveedores	Número: Creciente	**O**
Canales de distribución	Venta indirecta	**O**
Clientes	Racional	**O**

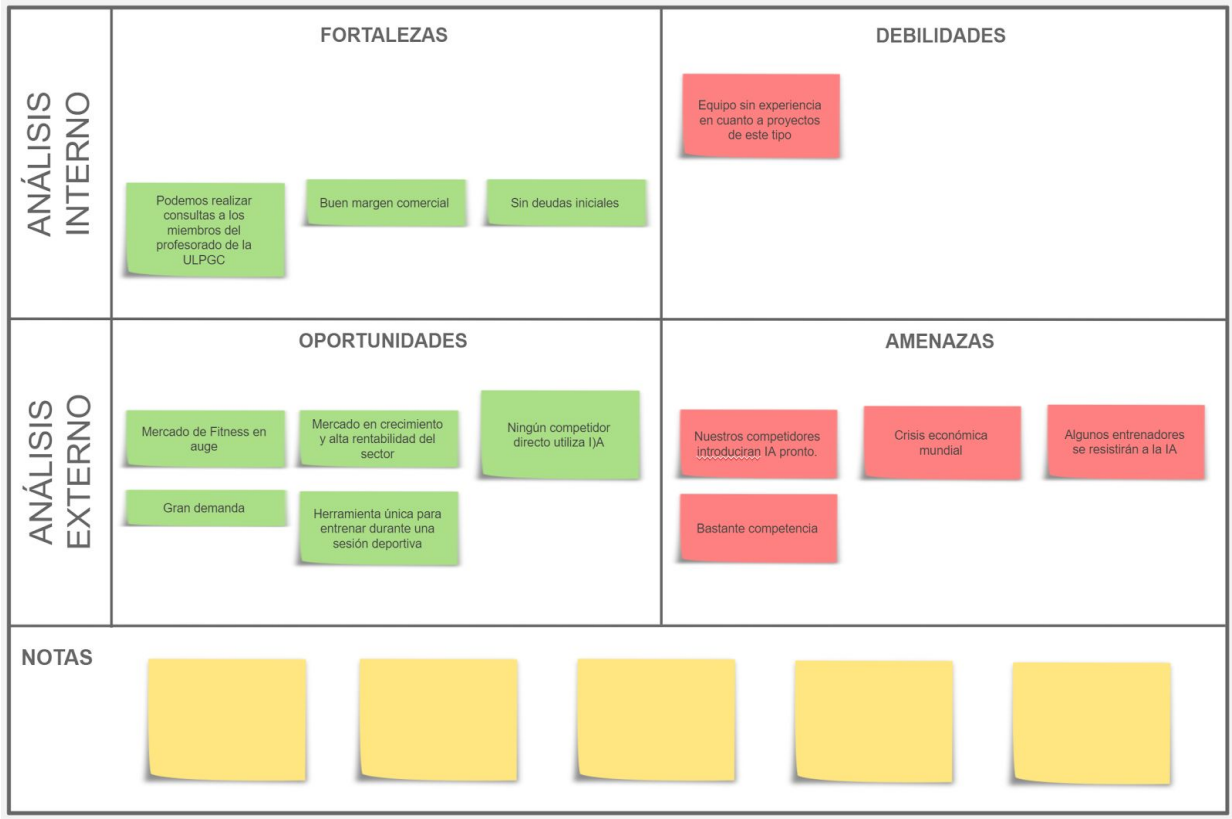
Factores y motivaciones de compra	Calidad, precio, servicio, garantía.	**O**

La viabilidad del proyecto, desde un punto de mercado, parece correcta. En futuros apartados estudiaremos más profundamente la viabilidad general del mismo, pero por ahora podemos ver que nuestro producto iría por buen camino.

Análisis DAFO

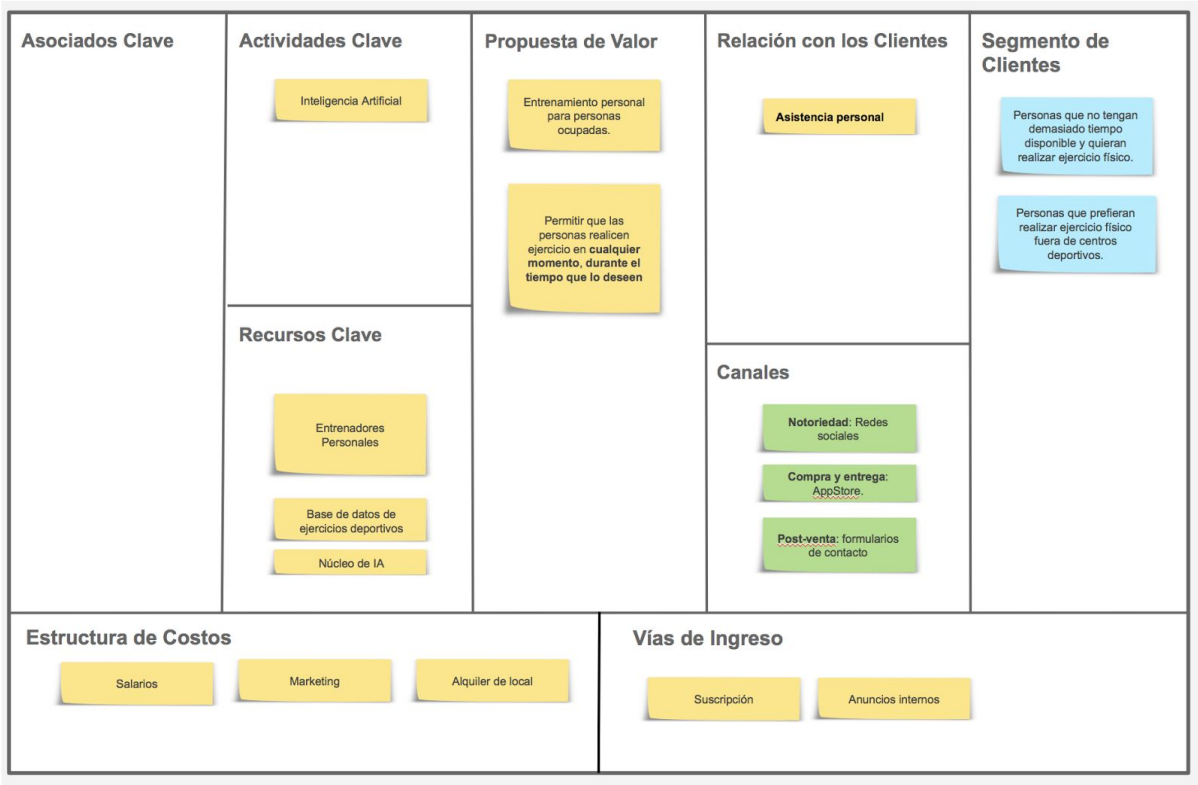
Si obtenemos todas las oportunidades y amenazas previamente encontradas, y las resumimos y abstraemos, podemos crear el análisis DAFO. Sólo faltarían las debilidades y las fortalezas a la hora de desarrollar el proyecto.

Tras meditar profundamente, se ha obtenido el siguiente resultado:



Con todo lo anterior, podemos definir al fin nuestro Bussiness Model Canvas, el modo más sencillo de comprender qué es lo que ofrece Cleverfit

Business Model Canvas



En cuanto al apartado de vías de ingreso, cabe destacar que las suscripciones no se encontrarán en la primera versión del producto, que es la que concierne a este documento. Las vías de ingreso serán detalladas en el apartado de aspectos económicos de la presente memoria.

Requisitos funcionales

Los requisitos funcionales del proyecto se detallan a continuación:

CÓDIGO	REQUISITO
CFR01	El sistema permitirá al usuario crear una cuenta.
CFR02	El sistema permitirá al usuario editar sus datos.
CFE01	El sistema permitirá al usuario realizar el entrenamiento.
CFE02	El sistema permitirá al usuario tener un control de los tiempos durante la realización de su entrenamiento.
CFE03	El sistema permitirá al usuario cancelar el entrenamiento
CFE04	El sistema permitirá al usuario pasar al siguiente ejercicio durante el entrenamiento
CFE05	El sistema permitirá al usuario pausar/reanudar el entrenamiento.
CFE06	El sistema indicará al usuario cuándo ha terminado el entrenamiento.
CFE07	El sistema mostrará una imagen del ejercicio que se está realizando durante el entrenamiento
CFE08	El sistema indicará al usuario cuánto falta para que termine el ejercicio que se encuentra realizando
CFH01	El sistema permitirá al usuario revisar su progreso mediante un historial, mostrando su peso y su Índice de Masa Corporal.
CFH02	El sistema generará una nueva entrada en el historial de progreso cada vez que se actualicen los datos del usuario.
CFH03	El sistema calculará el Índice de Masa Corporal para cada entrada del registro.
CFH04	El sistema permitirá al usuario ver todos los entrenamientos anteriormente realizados o generados.

CFIA01	El sistema generará un entrenamiento para el usuario cuando éste último lo desee.
CFIA02	El sistema generará entrenamientos no superiores al tiempo máximo de entrenamiento definido por el usuario.
CFIA03	El sistema generará entrenamientos considerando el objetivo del usuario.
CFIA04	El sistema generará entrenamientos considerando la experiencia del usuario.
CFIA05	El sistema generará entrenamientos considerando el historial de entrenamiento del usuario.
CFIA06	El sistema generará entrenamientos dinámicos, intentando, en la medida de lo posible, repetir ejercicios en la misma sesión.
CFIA07	El sistema generará entrenamientos en los que se abarquen todos los grupos musculares siempre que se pueda.

Requisitos no funcionales

A continuación se detallan los requisitos no funcionales:

Tiempos de respuesta

CÓDIGO	REQUISITO
CFNFT01	El sistema deberá generar las rutinas deportivas en menos de 5 segundos.

Usabilidad

CÓDIGO	REQUISITO
CFNFU01	El usuario debería tardar, como máximo, 30 minutos en comprender la interfaz del sistema
CFNFU02	El sistema debe tener una interfaz simple y minimalista

Desarrollo

CÓDIGO	REQUISITO
CFNFD01	La aplicación debe cumplir los principios SOLID
CFNFD02	La aplicación debe disponer del mínimo acoplamiento entre módulos
CFNFD03	El código de la aplicación debe respetar los estándares de Clean Code.

Compatibilidad

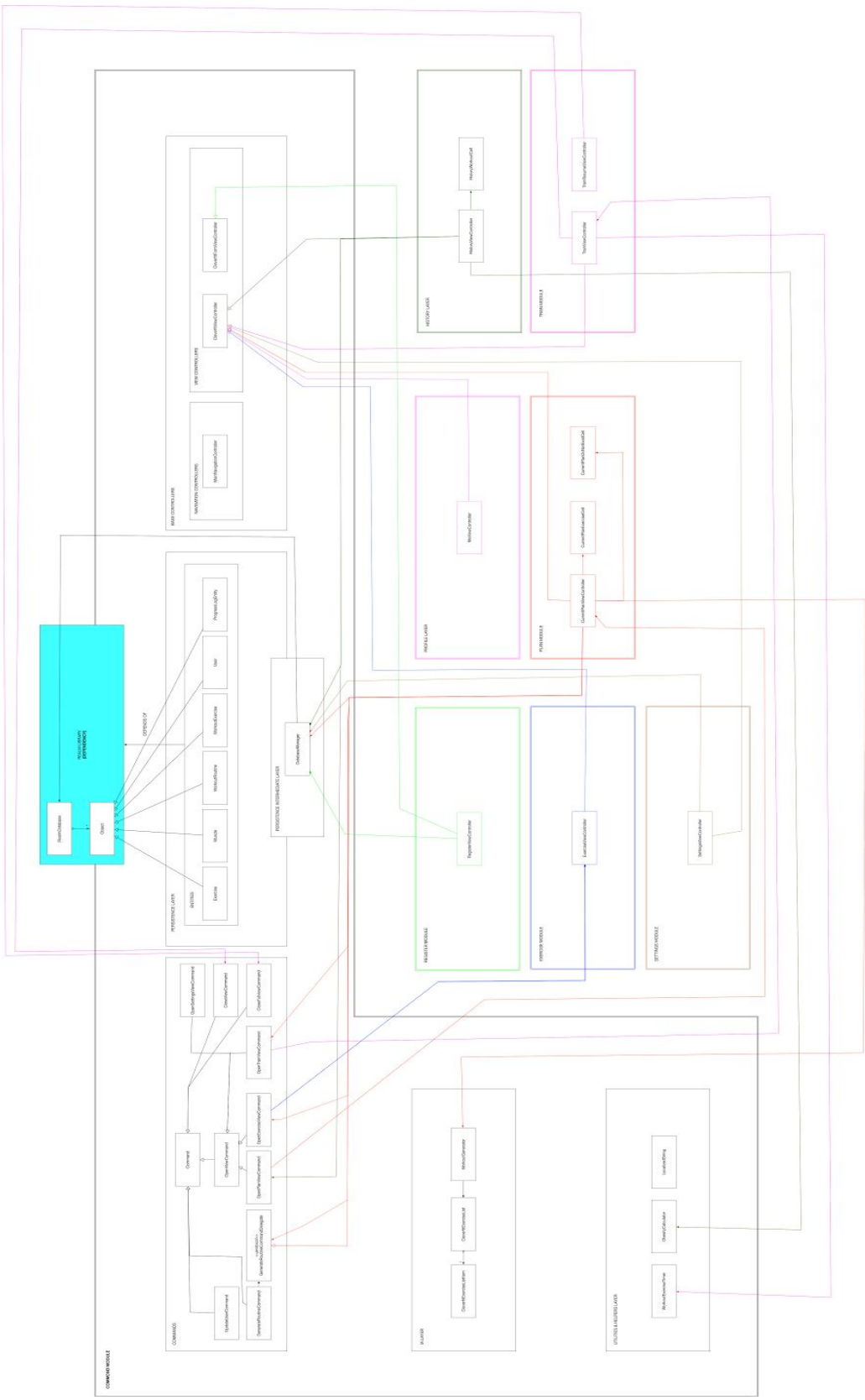
CÓDIGO	REQUISITO
CFNFC01	El sistema deberá ser compatible con iOS 9.0 o mayor
CFNCF02	La aplicación no puede ocupar más de 10 megabytes de espacio en disco
CFNCF03	La aplicación puede traducirse fácilmente a cualquier idioma

DISEÑO

Diseño del sistema

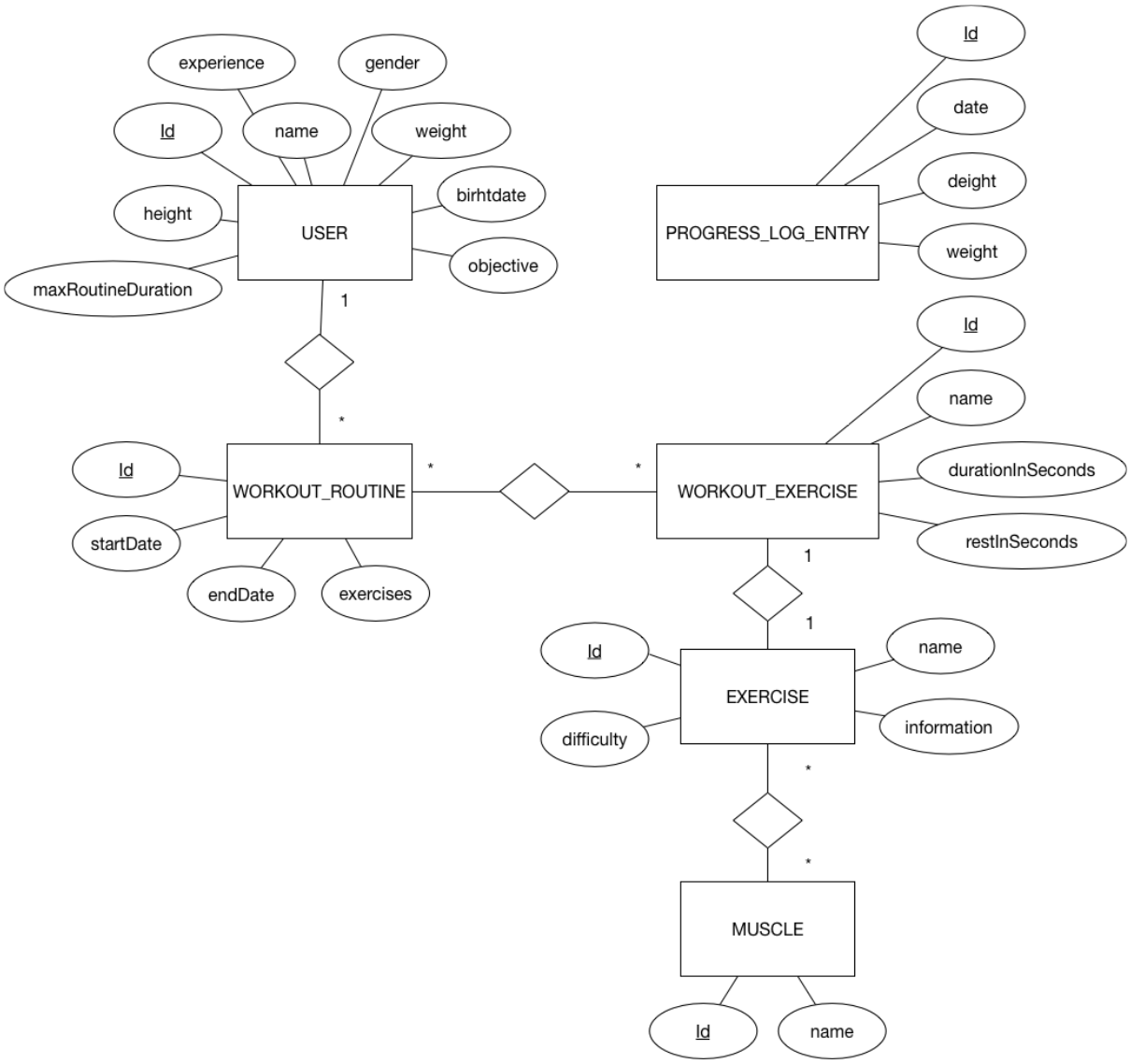
Arquitectura del sistema

(Siguiente página)



Arquitectura de la base de datos (Entidad-Relación)

A continuación se encuentra el modelo Entidad Relación correspondiente a la base de datos:E



Historias de usuario

Las historias de usuario se muestra en la siguiente tabla:

ID	NOMBRE	ROL	DESCRIPCIÓN	VALIDACIÓN
HU01	Crear usuario	Como **usuario**	quiero poder crear una cuenta con mis datos	Se completa el registro
HU02	Editar datos	Como **usuario**	quiero poder editar mis datos personales o de entrenamiento	Se editan los datos
HU03	Ver progreso	Como **usuario**	quiero poder ver mi progreso actual, con la evolución de mi peso y mi Índice de Masa Corporal	se muestra correctamente el progreso.
HU04	Ver historial	Como **usuario**	quiero poder ver mi historial de entrenamiento, con todas mis rutinas pasadas	se muestra correctamente el historial.
HU05	Ver entrenamiento actual	Como **usuario**	quiero poder ver mi entrenamiento actual	Se muestra el entrenamiento actual
HU07	Ver plan de entrenamiento	Como **usuario**	quiero poder ver cualquier plan de entrenamiento de mi historial	Se muestra correctamente el plan de entrenamiento
HU08	Ver ejercicio	Como **usuario**	quiero poder la información de un ejercicio de mi rutina	El ejercicio se muestra correctamente.
HU09	Solicitar nuevo entrenamiento	Como **usuario**	quiero poder solicitar un nuevo entrenamiento para poder continuar con mi plan general	Se genera un nuevo entrenamiento y se muestra.

HU10	Comenzar entrenamiento	Como **usuario**	quiero poder comenzar el entrenamiento actual	Comienza el entrenamiento asistido
HU11	Tener un entrenamiento asistido	Como **usuario**	quiero que mi entrenamiento sea asistido	El entrenamiento que se está realizando indica al usuario qué ejercicio debe hacer en cada momento
HU12	Pausar entrenamiento	Como **usuario**	quiero poder pausar el entrenamiento en cualquier momento	El entrenamiento de pausa
HU13	Reanudar entrenamiento	Como **usuario**	quiero poder reanudar mi entrenamiento siempre que esté pausado	El entrenamiento asistido se reanuda
HU14	Cancelar entrenamiento	Como **usuario**	quiero poder cancelar fácilmente el entrenamiento que estoy realizando	Se cancela el entrenamiento y se vuelve a la vista donde se muestra el plan actual.

Casos de uso

Los casos de uso aplicables a la aplicación se encuentran especificados a continuación:

Crear una cuenta

CASO DE USO	R001	CREAR CUENTA
Descripción		El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **crear una cuenta**.
Actores		Usuario
Precondiciones		No haber ejecutado antes el sistema

Flujo normal	Paso	Acción	
	1	El usuario especifica su nombre	
	2	El usuario especifica su edad	
	3	El usuario especifica su género	
	4	El usuario especifica su peso	
	5	El usuario especifica su altura	
	6	El usuario indica su experiencia	
	7	El usuario indica su objetivo	
	8	El usuario indica el máximo de tiempo disponible para entrenar	
	9	El usuario pulsa sobre el botón hecho	
Postcondiciones	El sistema crea un usuario local con los datos introducidos		
Excepciones	**Paso**	**Condición**	**Acción**
	9	Los datos introducidos no son correctos	Se muestra un área roja en los campos del formulario con los datos incorrectos
Observaciones			

Modificar datos personales

CASO DE USO	R002	MODIFICAR DATOS PERSONALES	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **modificar sus datos personales**.		
Actores	Usuario		
Precondiciones	Haber creado previamente una cuenta local		
Flujo normal	**Paso**	**Acción**	
	1	El usuario especifica su nombre	
	2	El usuario especifica su edad	
	3	El usuario especifica su género	

	4	El usuario especifica su peso	
	5	El usuario especifica su altura	
	6	El usuario indica su experiencia	
	7	El usuario indica su objetivo	
	8	El usuario indica el máximo de tiempo disponible para entrenar	
	9	El usuario pulsa sobre el botón hecho	
Postcondiciones	El sistema crea un usuario local con los datos introducidos		
	El sistema añade un registro en el historial de progreso con los nuevos datos introducidos		
Variaciones	**Paso**	**Acción**	
Extensiones	**Paso**	**Condición**	**Caso de Uso**
	9		
Excepciones	**Paso**	**Condición**	**Acción**
	9	Los datos introducidos no son correctos	Se muestra un área roja en los campos del formulario con los datos incorrectos
Observaciones			

Ver historial de progreso

CASO DE USO	**R003**	**VER HISTORIAL DE PROGRESO**
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **ver su historial de progreso**.	
Actores	Usuario	
Precondiciones	Haber creado previamente una cuenta local	
	Encontrarse en el menú inicial	
Flujo normal	**Paso**	**Acción**

	1	El usuario pulsa sobre la imagen que representa el historial de progreso
Postcondiciones	El sistema debe mostrar la vista de progreso, con el progreso actual del usuario	
Observaciones		

Ver historial de entrenamiento

CASO DE USO	**R004**	**VER HISTORIAL DE ENTRENAMIENTO**
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **ver su historial de entrenamiento**.	
Actores	Usuario	
Precondiciones	Haber creado previamente una cuenta local	
	Haberse generado un entrenamiento previamente	
	Encontrarse en la vista del menú inicial	
Flujo normal	**Paso**	**Acción**
	1	El usuario pulsa sobre la imagen que representa el historial de entrenamiento del usuario
Postcondiciones	El sistema debe mostrar la vista de entrenamiento, con todos los entrenamientos realizados por el usuario hasta día de hoy	
Observaciones		

Ver plan de entrenamiento

CASO DE USO	**R006**	**VER PLAN DE ENTRENAMIENTO**
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **ver su plan de entrenamiento**.	
Actores	Usuario	
Precondiciones	Haber creado previamente una cuenta local	

	Haberse generado previamente una rutina de ejercicios	
	Encontrarse en la vista inicial o en el historial de entrenamiento	
Flujo normal	**Paso**	**Acción**
	1	El usuario selecciona un plan de entrenamiento desde el historial de entrenamiento, o pulsa sobre el botón “empezar entrenamiento” de la vista principal
Postcondiciones	El sistema debe mostrar el plan de entrenamiento actual del usuario	
Observaciones		

Ver ejercicio

CASO DE USO	**R007**	**VER EJERCICIO**
Descripción		El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **ver un ejercicio de su rutina**.
Actores		Usuario
Precondiciones		Haber creado previamente una cuenta local
		Haberse generado previamente una rutina de ejercicios
		Encontrase en la vista de visualización de rutina
Flujo normal	**Paso**	**Acción**
	1	El usuario pulsa sobre el ejercicio del cuál desea obtener información
Postcondiciones		El sistema debe mostrar el ejercicio deseado
Observaciones		

Solicitar nuevo entrenamiento

CASO DE USO	R008	SOLICITAR NUEVO ENTRENAMIENTO	
Descripción		El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **solicitar un nuevo entrenameinto**.	
Actores		Usuario	
Precondiciones		Haber creado previamente una cuenta local	
		Encontrarse en la vista del plan actual	
Flujo normal		**Paso**	**Acción**
		1	El usuario pulsa sobre el botón “generar entrenamiento”
Postcondiciones		El sistema debe generar una nueva rutina	
		El sistema debe guardar la nueva rutina en la base de datos	
		El sistema debe actualizar la vista y mostrar la nueva rutina	
Observaciones			

Comenzar entrenamiento

CASO DE USO	R009	COMENZAR ENTRENAMIENTO	
Descripción		El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **comenzar a entrenar**.	
Actores		Usuario	
Precondiciones		Haber creado previamente una cuenta local	
		Haber generado un entrenamiento previamente	
		Encontrarse en la vista del plan actual	
Flujo normal		**Paso**	**Acción**
		1	El usuario pulsa sobre el botón “empezar entrenamiento”
Postcondiciones		El sistema debe inicializar la viste de entrenamiento	

	El sistema debe iniciar el temporizador interno
Observaciones	

Cancelar entrenamiento

CASO DE USO	**R010**	**CANCELAR ENTRENAMIENTO**	
Descripción		El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **parar su entrenamiento actual**.	
Actores		Usuario	
Precondiciones		Haber creado previamente una cuenta local	
		Haber generado un entrenamiento previamente	
		Encontrarse realizando un entrenamiento	
Flujo normal		**Paso**	**Acción**
		1	El usuario pulsa sobre el botón correspondiente de la vista
Postcondiciones		El sistema debe finalizar la ejecución del temporizador interno	
		El sistema debe abandonar la vista	
Observaciones			

Pasar al siguiente ejercicio durante el entrenamiento

CASO DE USO	**R011**	**PASAR AL SIGUIENTE EJERCICIO DURANTE EL ENTRENAMIENTO**
Descripción		El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **saltarse el ejercicio actual durante el entrenamiento**.
Actores		Usuario
Precondiciones		Haber creado previamente una cuenta local
		Haber generado un entrenamiento previamente
		Encontrarse realizando un entrenamiento

Flujo normal	Paso	Acción	
	1	El usuario pulsa sobre el botón correspondiente de la vista	
Postcondiciones	El sistema debe cambiar el ejercicio actual		
	El sistema debe actualizar el temporizador interno		
Excepciones	**Paso**	**Condición**	**Acción**
	1	No quedan ejercicios	Se cancela la acción
Observaciones			

Pausar entrenamiento

CASO DE USO	R012	PAUSAR ENTRENAMIENTO	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **pausar el entrenamiento**.		
Actores	Usuario		
Precondiciones	Haber creado previamente una cuenta local		
	Haber generado un entrenamiento previamente		
	Encontrarse realizando un entrenamiento		
	Que el entrenamiento no se encuentre pausado		
Flujo normal	**Paso**	**Acción**	
	1	El usuario pulsa en la zona central de la vista	
Postcondiciones	El sistema debe pausar el temporizador interno		
Observaciones			

Reanudar entrenamiento

CASO DE USO	R012	REANUDAR ENTRENAMIENTO

Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso concreto cuando alguna persona pretenda **reanudar el entrenamiento**.	
Actores	Usuario	
Precondiciones	Haber creado previamente una cuenta local	
	Haber generado un entrenamiento previamente	
	Encontrarse realizando un entrenamiento	
	Que el entrenamiento no se encuentre pausado	
Flujo normal	**Paso**	**Acción**
	1	El usuario pulsa en la zona central de la vista
Postcondiciones	El sistema debe reanudar el temporizador interno	
Observaciones		

Diseño de interfaz de usuario

Bajo la premisa principal: ahorra tiempo, se pensó en crear una interfaz simple y minimalista, que pudiera ser comprendida por el usuario casi al instante, y que requiriera poco tiempo de uso para acostumbrarse al 100%. Es por eso que se decidió prescindir de menús complejos y funciones adicionales, centrándonos en lo imprescindible para el usuario. Recordamos que nuestro usuario medio no tiene demasiado tiempo libre, luego no quiere perder más tiempo. Si incluimos muchas características, o sencillamente una interfaz un poco más compleja de lo que requiere, es posible que el usuario desinstale la aplicación de inmediato.

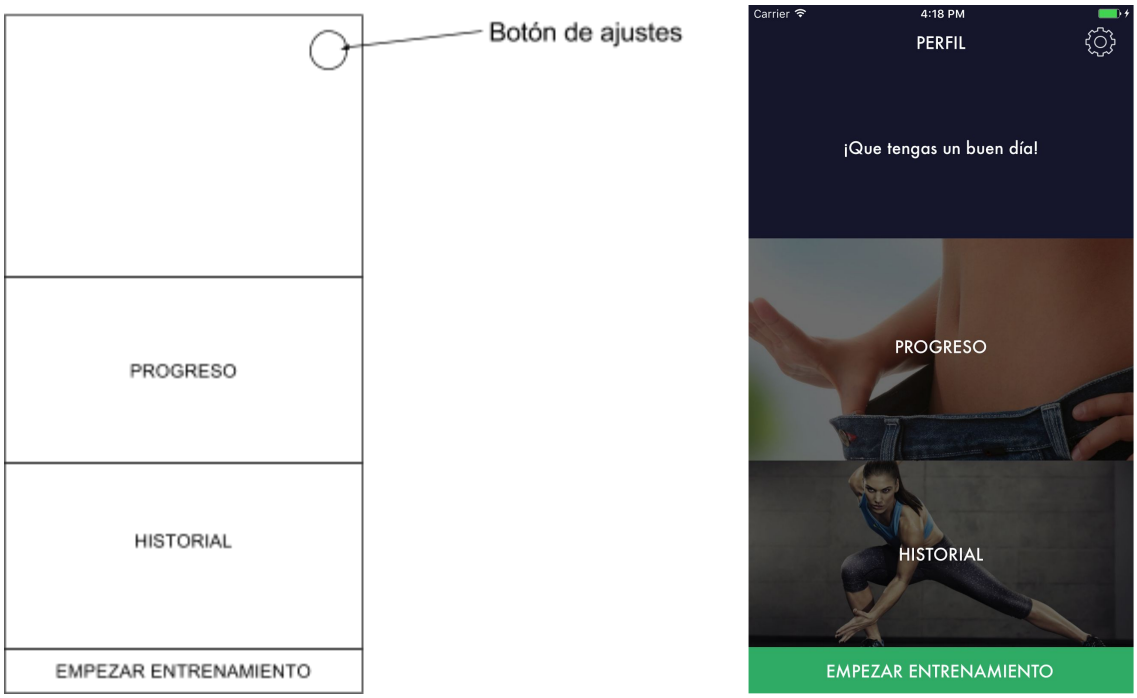
En la primera fase de diseño de la interfaz de usuario se idearon los Wireframes, esqueletos básicos del diseño de la aplicación, y se escogieron los colores principales (**verde** y **violeta**

oscuro). Luego se incluyeron los wireframes en la aplicación, y posteriormente se fue jugando con las combinaciones de los colores principales, quedando las vistas como se muestra a continuación. A la derecha, los wireframes. A la izquierda, el diseño final.

Estructura de la página inicial

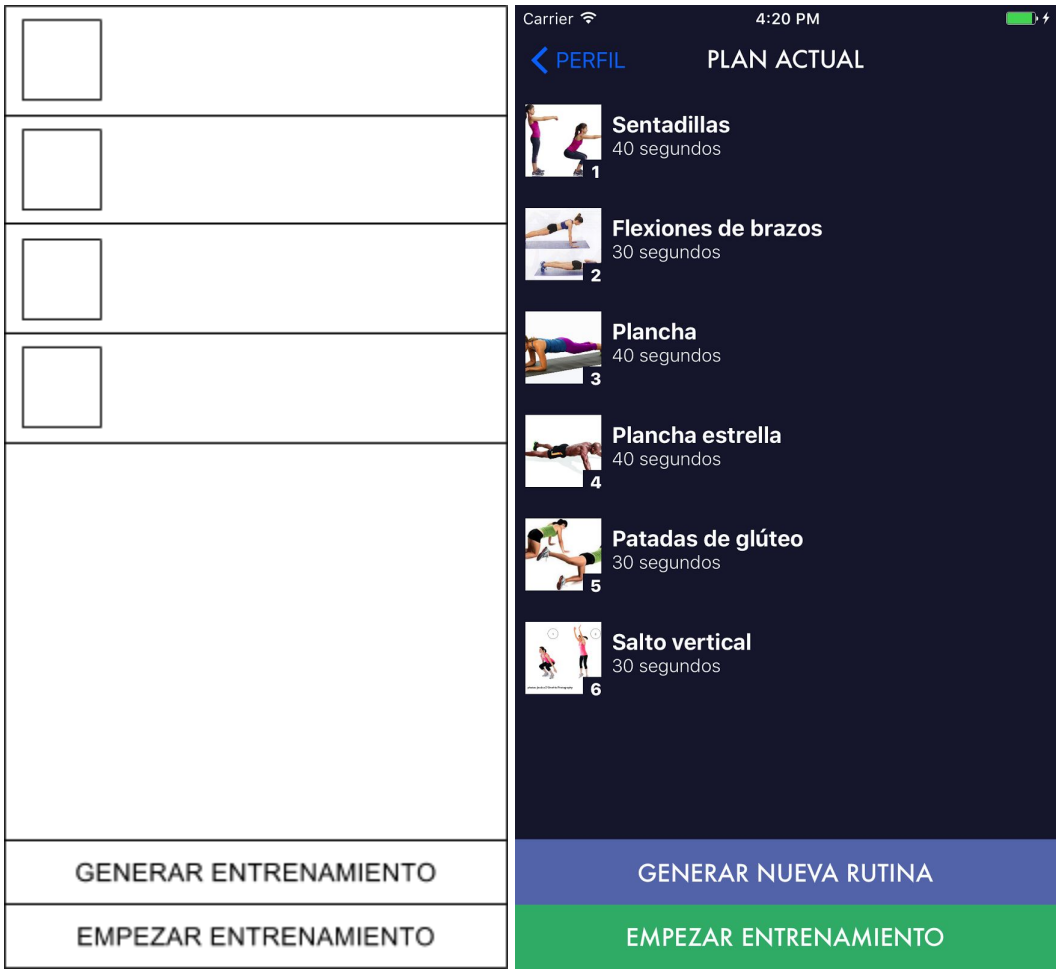
La página inicial representa el punto de llegada de los usuarios. Es lo primero que ven los usuarios registrados al abrir la aplicación, y lo segundo que ven aquellos usuarios que no tienen una cuenta.

Como uno de los principales objetivos de la aplicación es el ahorro de tiempo, esta pantalla debe representar simpleza, sencillez. Pero, a su vez, no debe dejar de lado la elegancia y la calidad. Es por ello que se ha decidido crear una interfaz simple, con accesos a todas las características de la aplicación con un sólo toque.



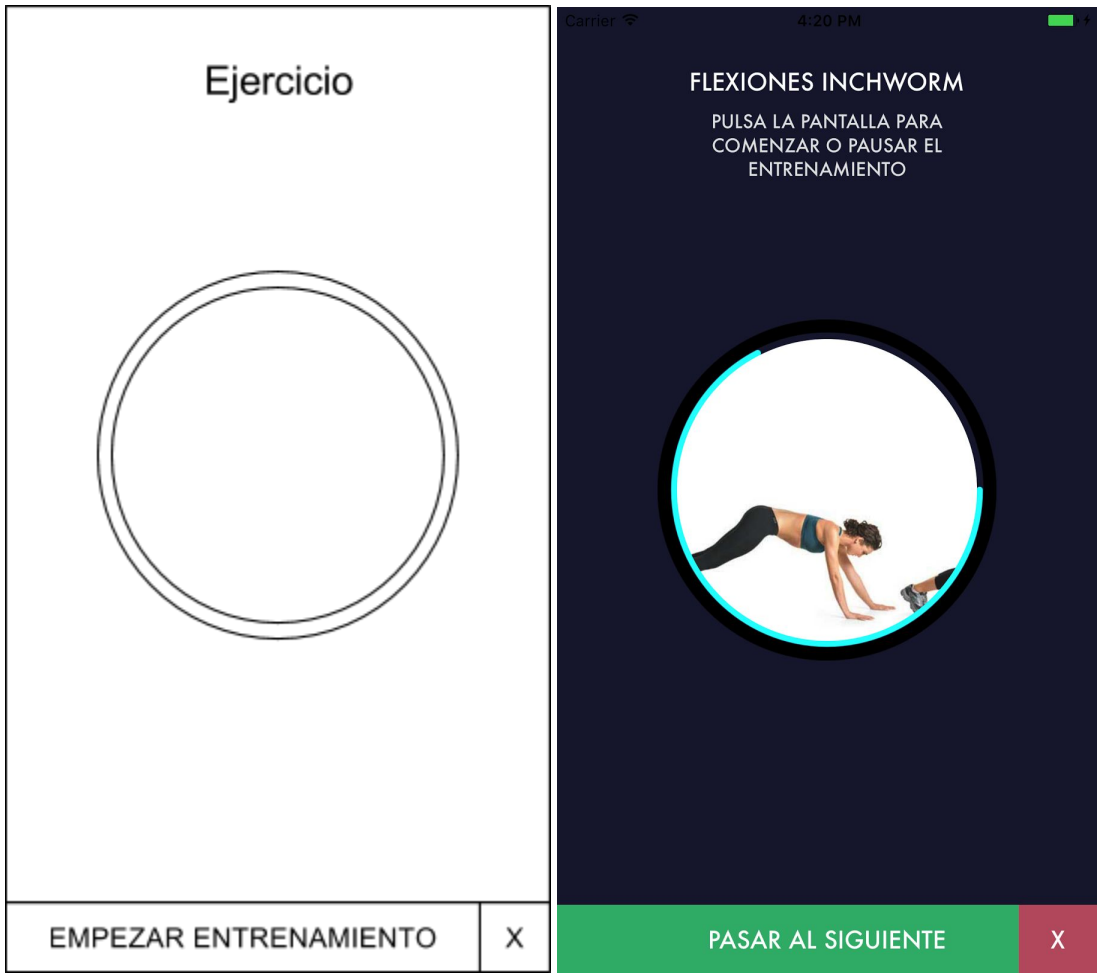
Estructura de la vista de plan de entrenamiento

Otra vez más, esta vista debe evitar perder tiempo al usuario. Por ello, se muestra un resumen del entrenamiento a realizar, y dos botones inferiores que permiten generar una nueva rutina y realizar la rutina actual. Se puede hacer click en cada ejercicio para saber la información.



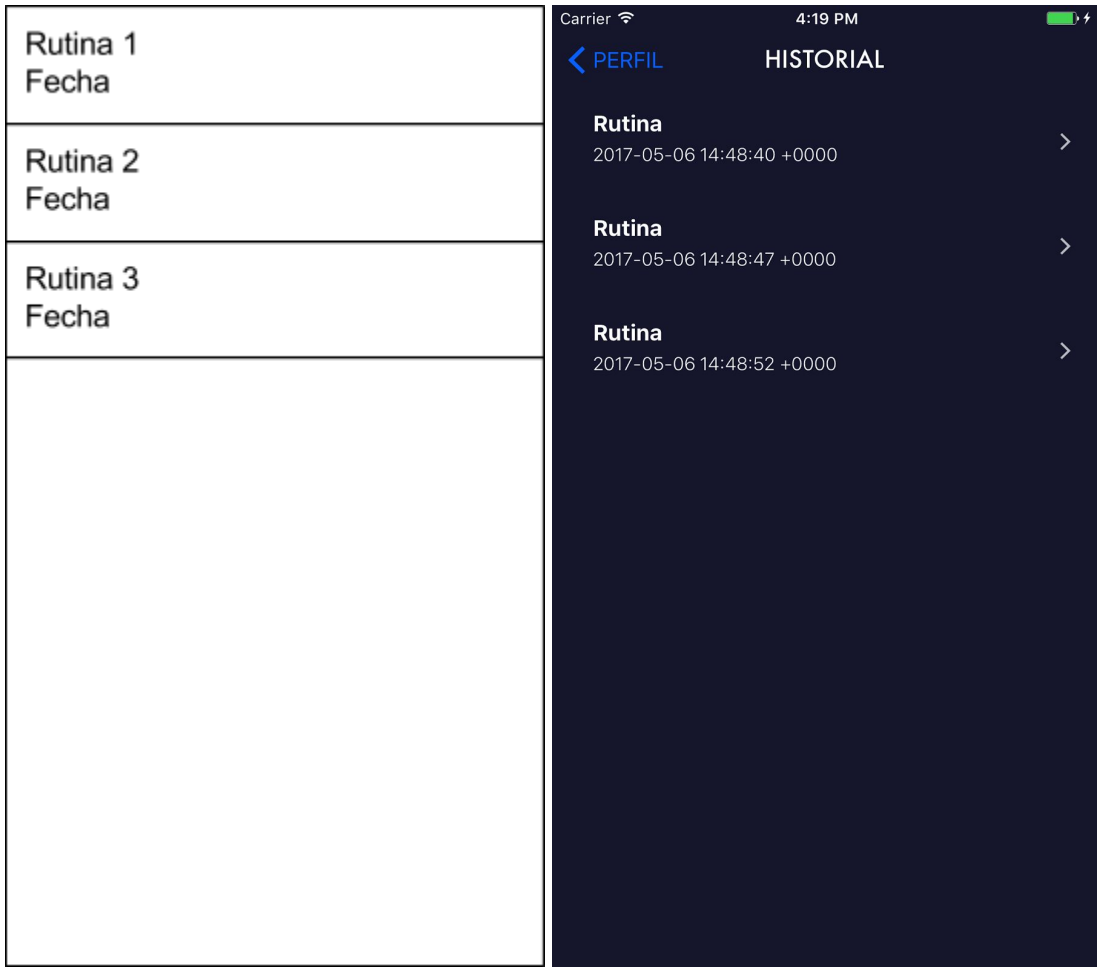
Estructura de la vista de entrenamiento

En esta vista se destaca la sencillez también. Está formada por un indicador de progreso del ejercicio actual, un botón para saltarse el ejercicio, y otro para cancelar el entrenamiento.,



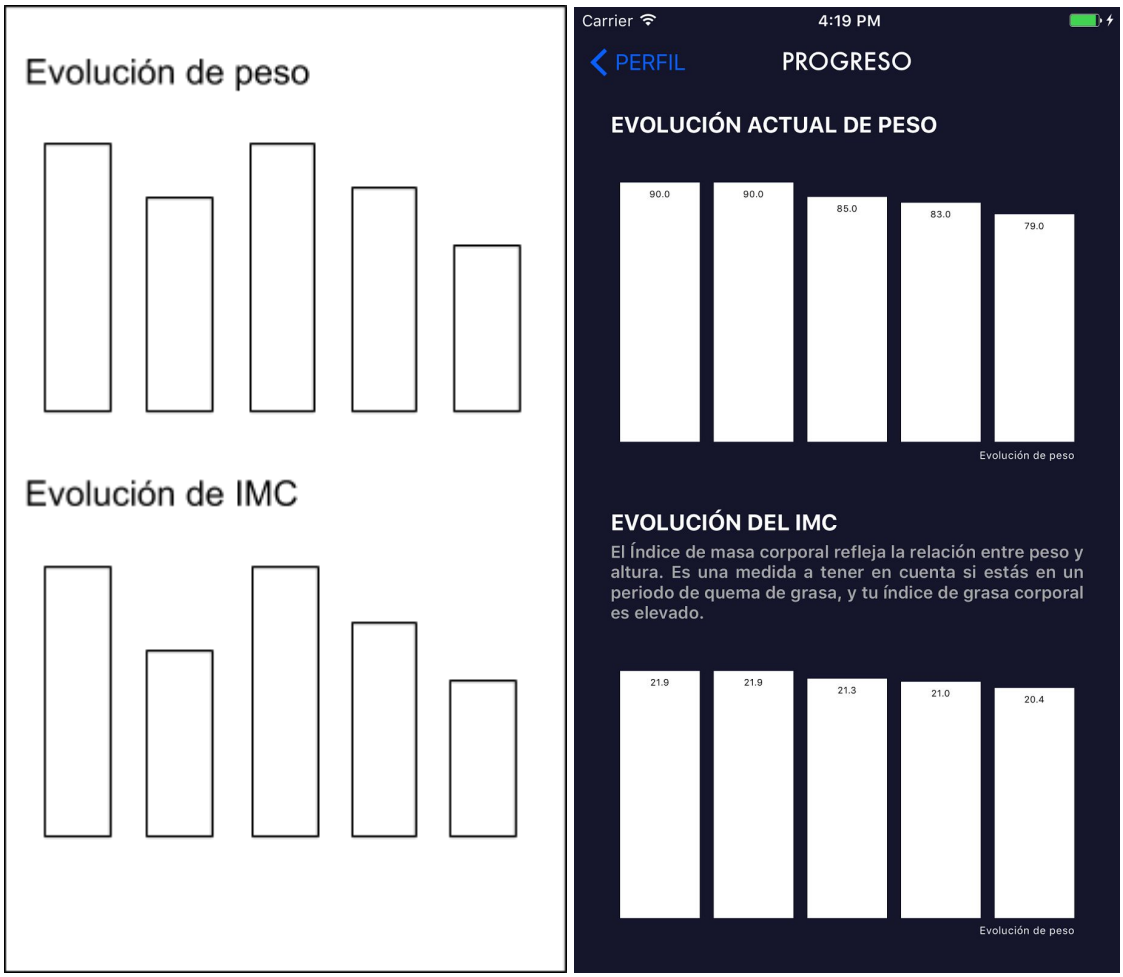
Estructura de la vista de historial de rutinas

Esta vista es mucho más simple que las anteriores. No es la más relevante, y es por ello por lo que no se ha hecho tanto énfasis.



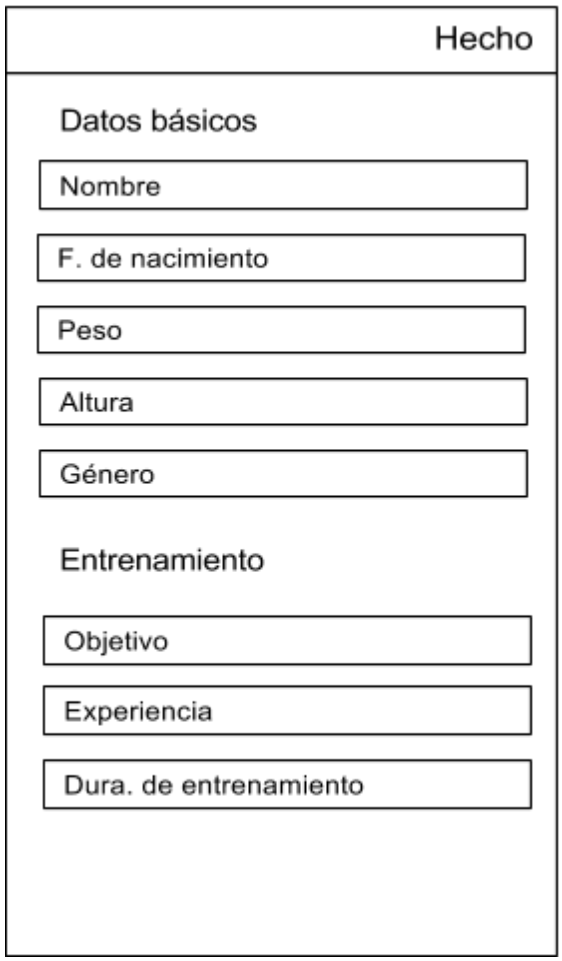
Estructura de la vista de progreso

Esta vista refleja el proceso del usuario. Cada vez que el usuario modifique sus ajustes, se generará una nueva entrada en el historial de progreso personal, y se representará aquí. Mediante una calculadora interna, por cada registro se calcula el IMC a partir del peso y la altura del usuario, y se representa en una segunda tanda de estadísticas. Esta vista es modular, en el sentido que se pueden añadir gráficas muy fácilmente y mejorar la calculadora interna, con el fin de calcular más parámetros.



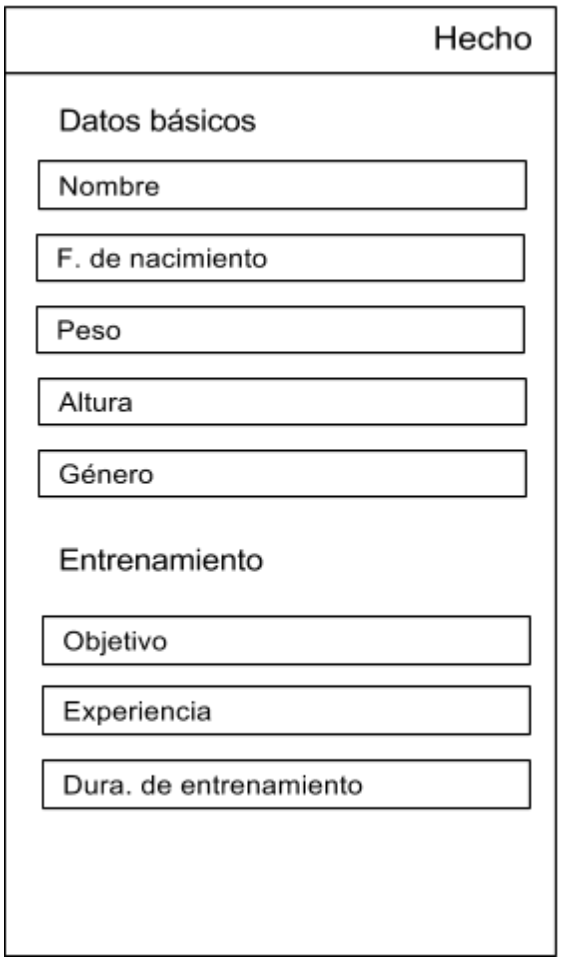
Estructura de la vista de crear cuenta

Crear una cuenta debe ser un proceso sencillo. En este caso, no pedimos datos que no necesitamos, y establecemos un formulario simple, dividido en dos intuitivas secciones para el usuario. La primera equivale a los datos personales. La segunda, a la personalización del entrenamiento. Esta última sección contiene los datos que usará la IA para generar entrenamientos.



Estructura de la vista de editar datos personales

La explicación de la vista anterior se aplica a esta, y es que se decidió realizar una vista muy similar con el fin de aprovechar el aprendizaje de uso del usuario durante el registro.


Hecho
Datos básicos
Nombre
F. de nacimiento
Peso
Altura
Género
Entrenamiento
Objetivo
Experiencia
Dura. de entrenamiento


Carrier
4:19 PM
PERFIL
AJUSTES
Done
INFORMACIÓN BÁSICA
NOMBRE Jose
FECHA DE NACIMIENTO May 6, 2008
PESO 90
ALTURA 190
GÉNERO Hombre
CONFIGURACIÓN DE ENTRENAMIENTO
OBJETIVO Perder peso
EXPERIENCIA Nuevo
DURACIÓN DE ENTRENAMIENTO 7 minutos (p...

MANUAL DE USUARIO

Registro

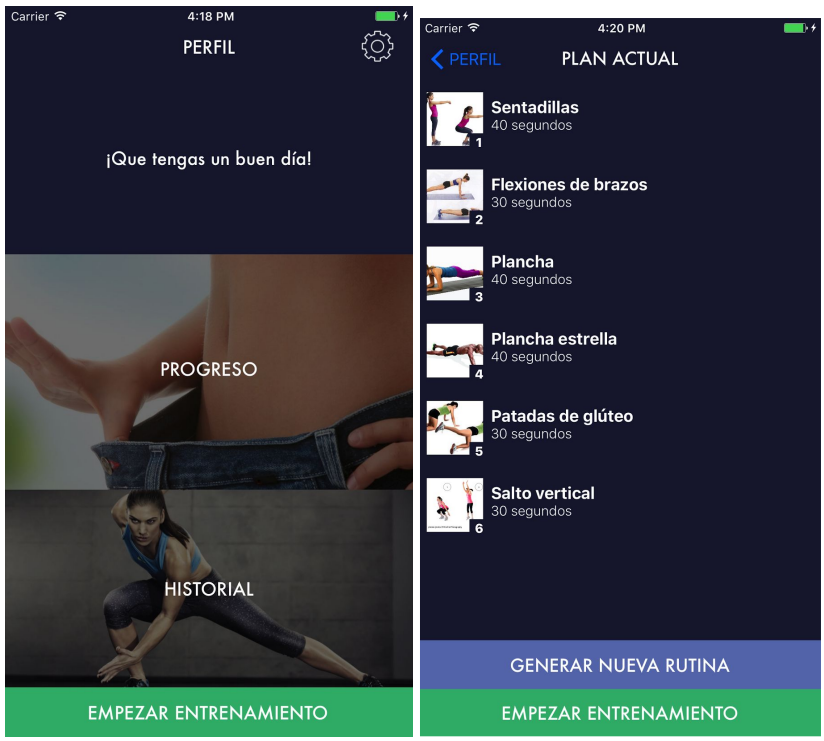
Para registrarte, simplemente abre la aplicación por primera vez, y aparecerá la pantalla de registro. Deberás introducir los siguientes campos:

- Nombre: Indica tu nombre de usuario o nombre real
- Fecha de nacimiento: Indica cuándo naciste. Esto nos ayudará a calcular tu Índice de Masa corporal
- Peso: Indica tu peso para que puedas conocer el progreso del mismo.
- Altura: Indica tu altura

- Género: ¿Eres hombre o mujer?
- Objetivo: Por ahora, el único objetivo disponible es la pérdida de peso
- Experiencia: Tu experiencia influye en la selección de ejercicios, cuya dificultad variará en función de la misma.
- Duración de entrenamiento: Indica la duración máxima aproximada de tu entrenamiento

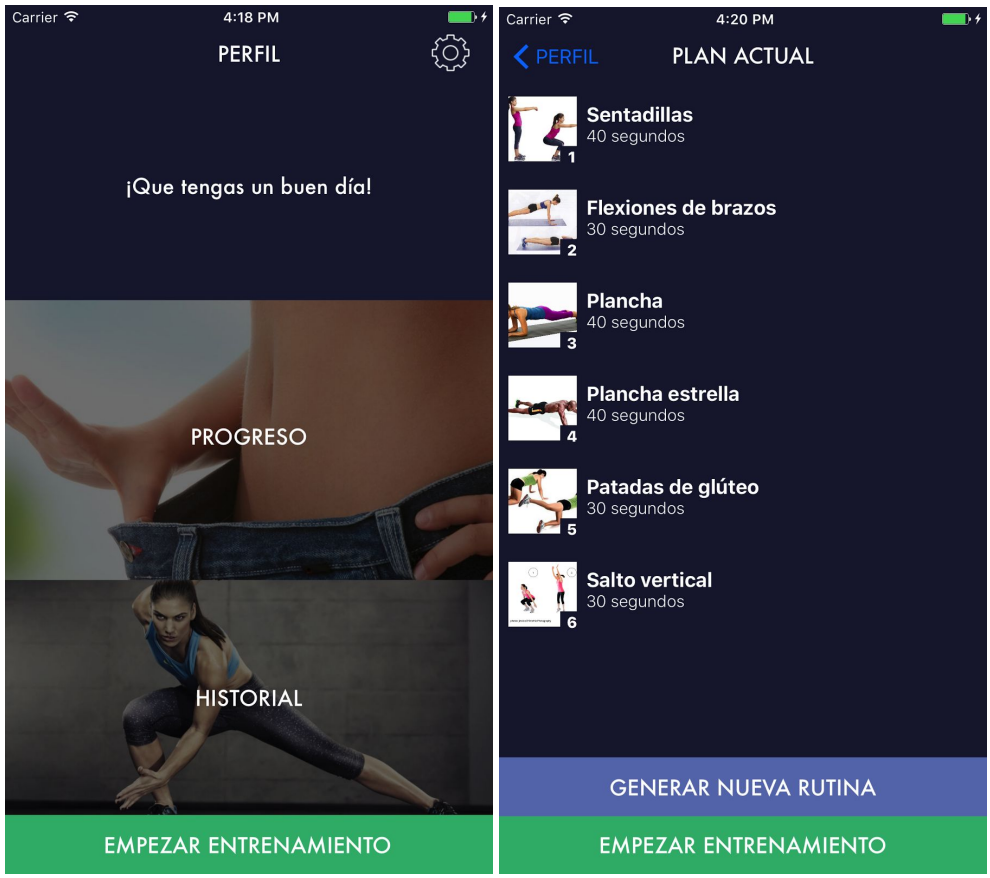
Generar entrenamiento

Para generar un entrenamiento pulsa sobre el botón “Empezar entrenamiento” de la imagen de la izquierda. Después, pulsa sobre generar entrenamiento. Se generará una rutina para ti en función de tu experiencia, historial previo de entrenamiento y, por supuesto, objetivo.



Comenzar entrenamiento

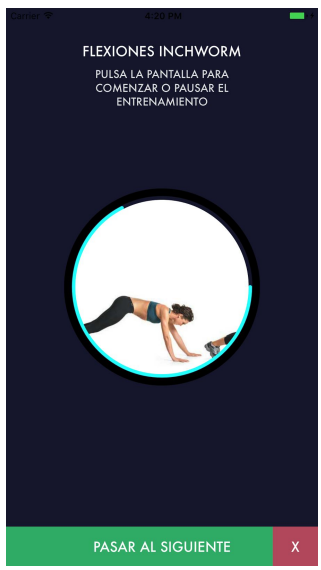
Para comenzar un entrenamiento, simplemente pulsa sobre el botón "Empezar entrenamiento" de la pantalla principal, tal como se muestra en la imagen izquierda. Accederás a la vista de resumen de entrenamiento, donde podrás ver la lista de ejercicios a realizar. Posteriormente, pulsa nuevamente sobre comenzar entrenamiento (imagen derecha) para empezar con el mismo.



Asistencia durante el entrenamiento

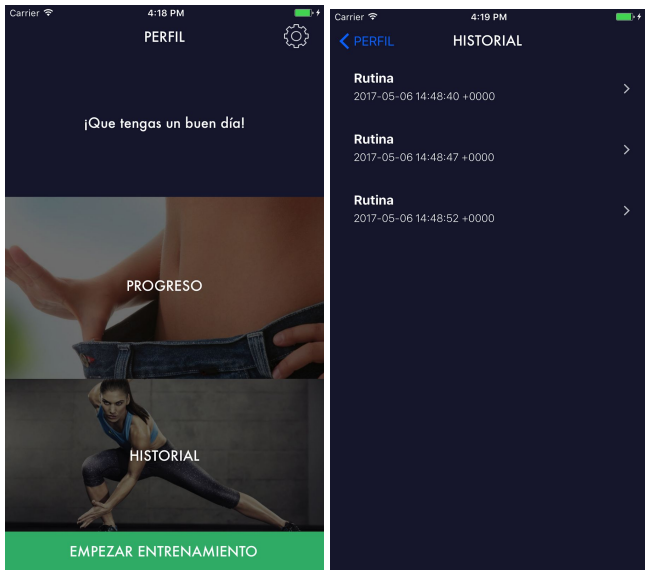
El asistente o ayudante de entrenamiento te echara una mano durante el mismo. Te indicará, en todo momento cuánto te falta para finalizar el ejercicio. Al finalizar un ejercicio, el asistente también te indicará cuánto tiempo de descanso tienes. El asistente de entrenamiento estará contigo durante todas tus rutinas, ayudándote a realizar todo de forma más amena, mostrándote en todo momento el ejercicio actual, indicando su nombre en la zona superior de la pantalla, y mostrando una imagen en la zona media.

Si quieres saltar un ejercicio, puedes hacerlo pulsando el botón “pasar al siguiente”. Si deseas cancelar el entrenamiento actual, simplemente pulsa sobre el botón rojo ubicado en la zona inferior derecha de la pantalla.



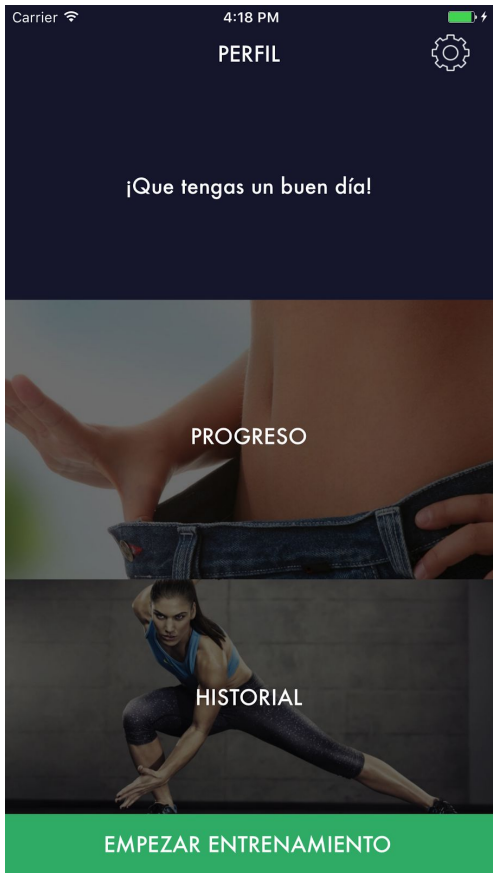
Comprobar historial previo

Puedes comprobar en cualquier momento tus entrenamiento previos. Simplemente ve a la sección "historial" de la vista principal (imagen izquierda), y podrás ver todas las rutinas previamente generadas (imagen derecha)



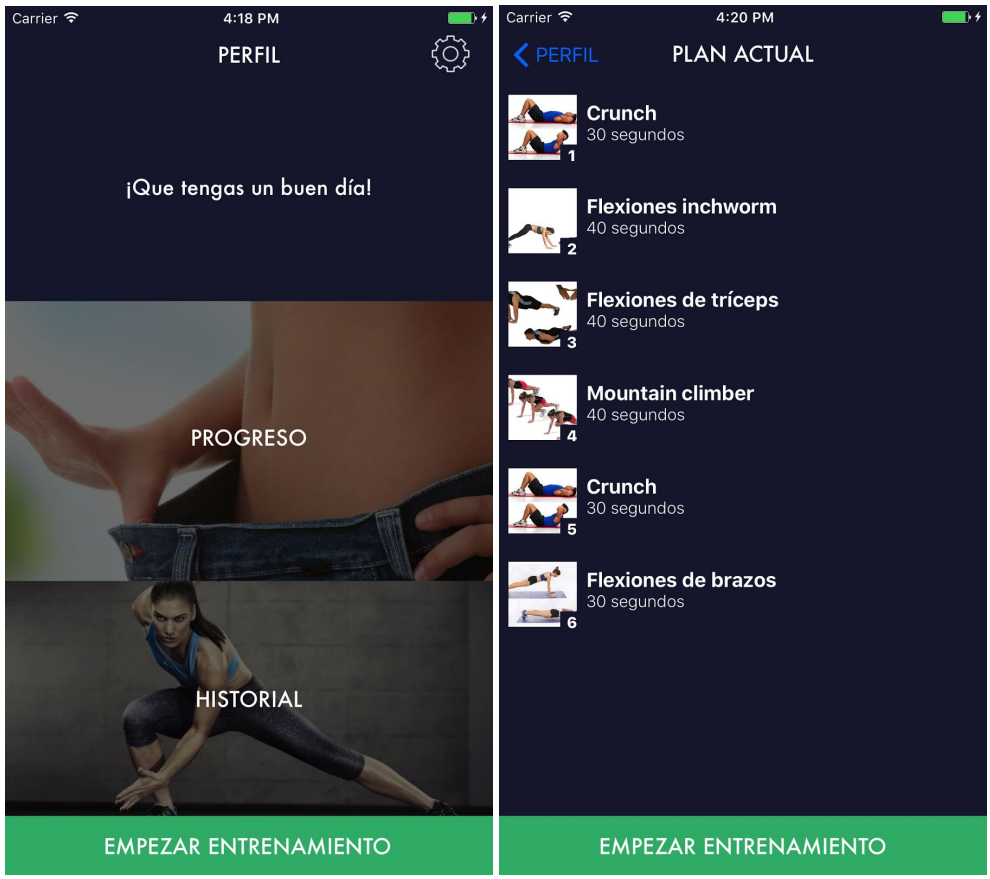
Comprobar progreso actual

Cada vez que modifiques tus datos desde ajustes, se añadirá un nuevo registro a tu historial de progreso. Puedes ver tu progreso pulsando sobre la pantalla "progreso" de la vista principal. Verás la evolución de tu peso, así como tu Índice de Masa Corporal.



Ver rutina de entrenamiento actual

Después de haber generado una rutina por primera vez, podrás ver tu rutina de entrenamiento actual haciendo click en el botón “empezar entrenamiento” de la vista principa (imagen izquierda). Automáticamente accederás a la pantalla de rutina de entrenamiento, donde podrás ver tu plan de entrenamiento actual (imagen derecha)

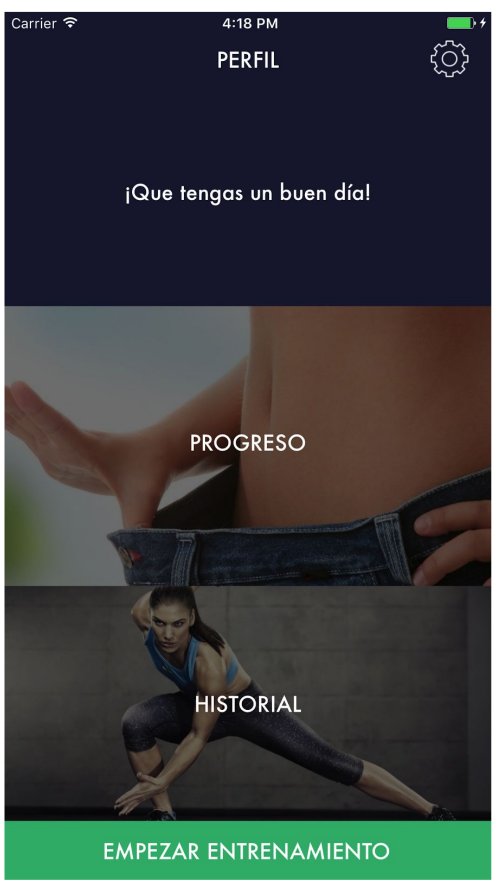


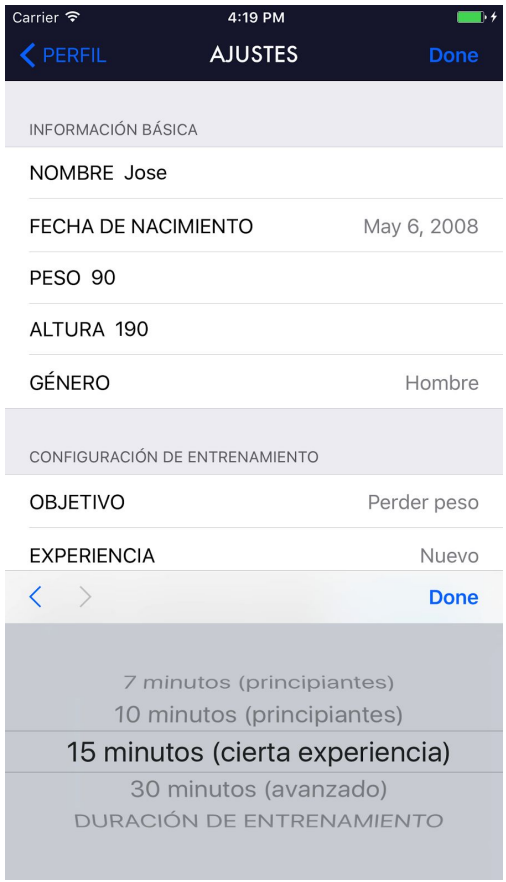
Ver ejercicio

Pulsa sobre cualquier ejercicio del plan actual o de cualquier plan dentro del historial de entrenamiento para ver información sobre el mismo.

Editar mis datos

Para editar tus datos, simplement pulsa sobre el botón de ajustes situado en la esquina superior derecha de la ventana principal. Automáticamente accederás a la pantalla de ajustes, donde podrás modificar tus datos. Cada vez que modifiques tus datos, se actualizará tu historial de progreso.





DOCUMENTACIÓN DE DESARROLLO

Instalación, requisitos y dependencias utilizadas

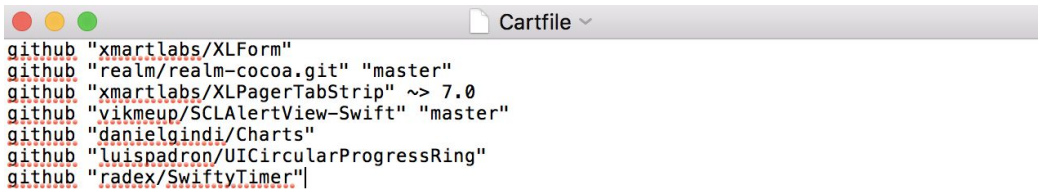
Este proyecto ha sido desarrollado en Swift 3.0, por lo que requiere de **Xcode 8.0** (o una versión mayor) para ser compilado y ejecutado.

Así mismo, y al tratarse de iOS,. se ha utilizado **Carthage** como gestor de dependencias, de modo que instalarlas sea tan sencillo como ejecutar el comando **carthage -build** en el directorio principal del proyecto. De este modo, se descargarán todas las dependencias, y el proyecto podrá ser compilado correctamente.

Por tanto, y sabiendo esto, podríamos concretar que:

- **Requisitos para la ejecución del código**
    - Xcode 8.0 o mayor
    - Carthage instalado en el equipo (https://github.com/Carthage/Carthage)
- **Pasos para ejecutar el proyecto**
    - Ejecutar el comando **carthage -build** mediante el terminal, directamente en el directorio principal del proyecto.
    - Ejecutar el proyecto normalmente en Xcode
- **Dependencias utilizadas durante el desarrollo del proyecto**

El principal motivo por el que se eligió Carthage ha sido por la facilidad a la hora de manejar las dependencias. Es tan sencillo como declararlas en un fichero, y serán descargadas adecuadamente mediante el comando anteriormente indicado, pudiendo dejar el proyecto bastante más ligero. Las librerías se indican en un fichero, quedando de este modo:



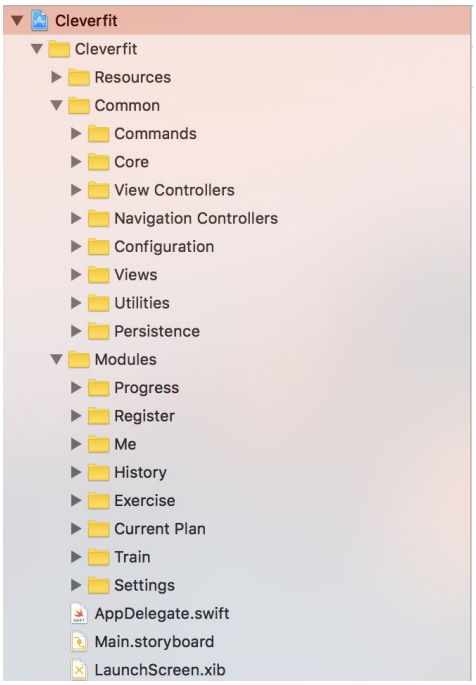
Las librerías utilizadas, por tanto, son las siguientes

- **XLForm**: Permite un mejor manejo de formularios que la librería nativa de iOS
- **Realm**: El gestor de base de datos que hemos utilizado por defecto. Es bastante sencillo de usar, permite tener una base local controlada, y se trata de una de las librerías más importantes en el mundo del desarrollo. La explicaremos más adelante.
- **SCAlertViewSwift**: Permite mostrar alertas de una forma muy, muy sencilla en la aplicación, respetando los estándares establecidos por Clean Code.
- **UICircularProgressRing**: Permite crear una barra de progreso circular. ¿Lo mejor? Es invisible al código. Su declaración se efectúa directamente en la vista de la aplicación. Lo explicaré más adelante.
- **SwiftyTimer**: Como su nombre indica, se trata de un Timer, y realmente se ha usado como una simple base para un cronómetro propio.

El código tras Cleverfit

Con el fin de incrementar el conocimiento sobre la aplicación, así como explicar lo que realmente es más relevante para este proyecto, su arquitectura, se ha decidido incluir extractos del código con el fin de explicar el funcionamiento tras Cleverfit.

Arquitectura de la aplicación



Al desarrollar Cleverfit, quería hacer relucir su arquitectura, creando módulos independientes entre sí, aumentando así la coherencia del proyecto y disminuyendo drásticamente su acoplamiento. Podemos apreciar en la imagen adjunta, que Cleverfit dispone de varios módulos independientes, correspondientes a cada una de las vistas del proyecto. Cada uno de los módulos no conoce la existencia de los otros, excepto de uno: **El módulo Common**.

El módulo Common se trata, sencillamente, de un módulo común a la mayoría de módulos. Es un recurso a usar por cada uno de ellos cuando se necesite. Eso sí, si bien es cierto que el módulo Common tiene conocimiento sobre la existencia de los módulos independientes, bajo ningún concepto puede utilizar recursos de los mismos y generar una referencia circular. En este proyecto, las referencias circulares están estrictamente prohibidas. De hecho, y debido a la facilidad de la arquitectura, se ha decidido

establecer una arquitectura **MVC**. En un comienzo se pensó en incluir una arquitectura MVVM, pero no fue necesario debido a las facilidades que otorga iOS a la hora de gestionar vistas y definir las clases.

Procedo a detallar los módulos, de modo que se puedan apreciar algunos de los patrones de diseño utilizados. Eso sí, también se detallarán más abajo

El módulo Common

Los controladores base

Una de las partes más relevantes del módulo común son los controladores base. No porque sea el código más importante, ni mucho menos. De estas clases heredan todos los controladores de la aplicación, y disponen de métodos útiles que posibilitan la escritura de código limpio. Hay que ver estas clases como una capa de nivel medio, ya que no llega a hacer gestiones a bajo nivel, pero tampoco a alto. Simplemente definen métodos para que sus clases hijas no tengan que realizar tanto esfuerzo a la hora de realizar operaciones básicas de gestión de vista

```
class CleverFitFormViewController: XLFormViewController {

    override open func viewDidLoad() {
        super.viewDidLoad()
    }

    func showNavigationBar() {
        navigationController?.setNavigationBarHidden(false, animated: false)
    }

    func hideNavigationBar() {
        navigationController?.setNavigationBarHidden(true, animated: false)
    }


    func addDescriptor(to section: XLFormSectionDescriptor,
                       with tag: FormTag,
                       title: String,
                       descriptorType: String,
                       required: Bool) {
        addDescriptor(to: section, with: tag, title: title, descriptorType: descriptorType, options: [XLFormOptionObject]() as! [XLFormOptionsObject],
            required: required)
    }

    func addDescriptor(to section: XLFormSectionDescriptor,
                       with tag: FormTag,
                       title: String,
                       descriptorType: String,
                       options: [XLFormOptionsObject],
                       required: Bool) {
        var row: XLFormRowDescriptor
        row = XLFormRowDescriptor(tag: tag.rawValue, rowType: descriptorType, title: title)
        row.isRequired = required
        if (!options.isEmpty) {
            row.selectorOptions = options
            row.value = options[0]
        }
        section.addFormRow(row)
    }
```

Si observamos, por ejemplo, la clase **CleverFitFormViewController**, podemos ver que dispone de dos métodos básicos, aparte de los métodos de esconder o mostrar la barra de navegación. Estos dos métodos permiten añadir los descriptores de los formularios de forma simple, de tal forma que sus hijos sólamente tengan que llamar al método, sin necesidad de realizar comprobaciones cada vez que desean añadir un descriptor. Cabe destacar que un descriptores se trata de un campo del formulario en la librería XLForm. De hecho, se puede apreciar cómo esta clase base precisamente se ha realizado para facilitar la realización de formularios.

El primer método **addDescriptor**, como se puede observar, se diferencia con respecto al segundso en el hecho de que no dispone del parámetro options. ¿Por qué? Sencillo, el método superior permite crear campos sin opciones, mientras que el inferior necesita de opciones

para poder trabajar correctamente, creando un campo que despliega un selector de opciones de cara al usuario. Simple, ¿verdad? ¡Clean Code!

Por otro lado, también se ha definido una base para los controladores que no desean implementar formularios en las vistas. Realmente dispone de métodos simples, pero considero que es una API más agradable que la nativa de Swift.

```swift
import UIKit

class CleverFitViewController: UIViewController {

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
    }

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    func showNavigationBar() {
        navigationController?.setNavigationBarHidden(false, animated: false)
    }

    func hideNavigationBar() {
        navigationController?.setNavigationBarHidden(true, animated: false)
    }

    func hideBackButton(animated: Bool) {
        navigationItem.setHidesBackButton(false, animated: animated)
    }

    func showBackButton(animated: Bool) {
        navigationItem.setHidesBackButton(true, animated: animated)
    }

}
```

Comandos de la aplicación: Patrón Command

¿Cómo hacemos que los controladores no sean responsables de decidir el comportamiento de las interacciones del usuario? Utilizando el patrón Command, que nos ayudará a aislar este tipo de decisiones, permitiendo evitar interacciones entre distintos módulos, y evitando dotar a controladores y vistas de decisiones que carecen de relevancia para ellos.

Se han definido dos tipos de protocolos (equivalente a una interfaz en Java) principales en la App:

- FeedbackCommand: Permite definir comandos que indiquen si su ejecución ha sido correcta o no. Genial cuando se trata de ejecutar operaciones sobre la base de datos o cuando se intenta acceder a otra vista.
- NoFeedbackCommand: Comandos sin retroalimentación. No es necesario que indiquen nada, aunque se podría incluir un delegado para operaciones asíncronas.

```
import Foundation
import UIKit

protocol FeedbackCommand {

    func execute()-> Bool?

}

protocol NoFeedbackCommand {

    func execute()

}
```

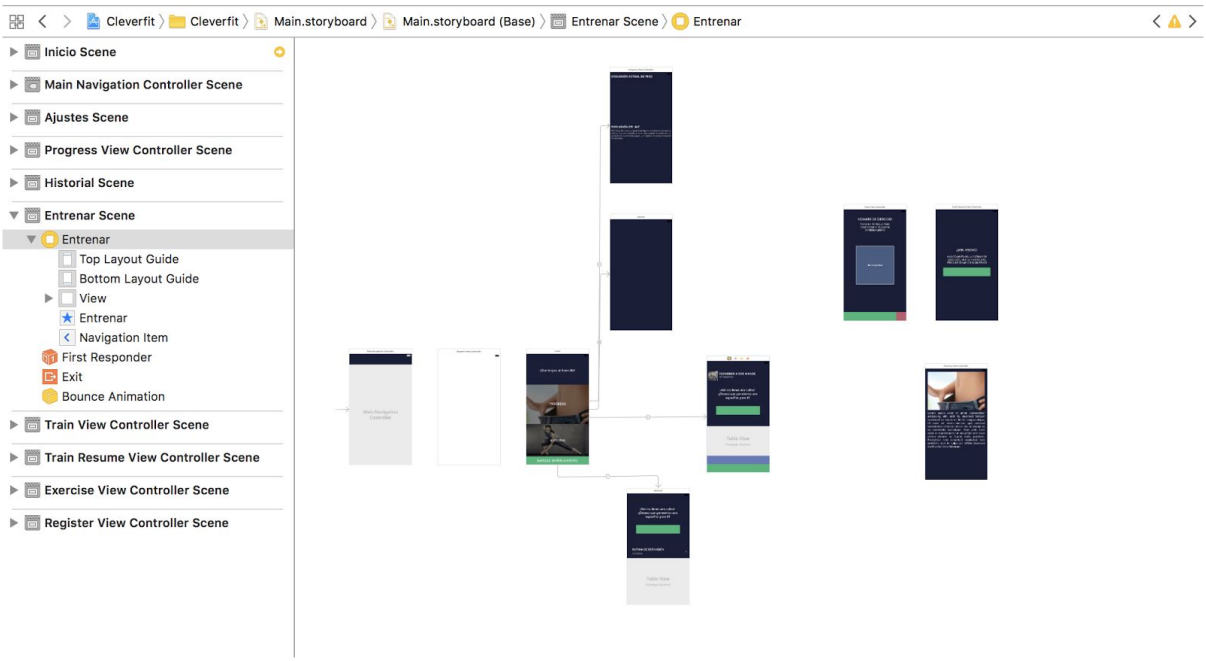
Con esta base de definición, sae pueden crear comandos que permitan, como indicamos anteriormente, abrir/cerrar vistas, ejecutar operaciones sobre la base de datos, etc. Aislando, de este modo, la responsabilidad que inicialmente cae, en este caso sin sentido, sobre los controladores de la aplicación. Pongamos algún ejemplo:

- **Actualizar la información de un usuario registrado:** Si un usuario desea modificar sus datos, pulsando sobre el botón actualizar, automáticamente el módulo que controle esa vista llamará al comando UpdateUserCommand, inyectándole el

controlador de navegación y los datos del usuario a actualizar. Al ejecutar el comando mediante el método execute(), se ejecutará el comando- El método execute() intentará actualizar el usuario en la base de datos, y si lo consigue, intentará volver atrás y cerrar la vista de ajustes. Simple, ¿verdad? Imagináos todo esto realizado por el controlador, no quedaría nada limpio.

```
import Foundation
import UIKit

final class UpdateUserCommand: FeedbackCommand {

    let currentNavigationController: UINavigationController
    let user: User

    init(currentNavigationController: UINavigationController, user: User) {
        self.currentNavigationController = currentNavigationController
        self.user = user
    }

    func execute()-> Bool? {
        if (updateUserInDatabase()) {
            return openViewController()
        }
        return false
    }

    private func updateUserInDatabase()-> Bool {
        if DatabaseManager.sharedInstance.update(user: user) {
            return DatabaseManager.sharedInstance.addProgressEntry(height: Double(user.height), weight: Double(user.weight))
        }
        return false;
    }

    private func openViewController()-> Bool {
        currentNavigationController.popViewController(animated: true)
        return true

    }
}
```

- **Abrir/cerrar vistas:** Se estuvo pensando en incluir el patrón de diseño FlowController para controlar el flujko de las vistas, aislándolo así del controlador principal. Pero lo cierto es que iOS facilita mucho la navegación entre las mismas, definiendo la navegación automáticamente desde la gestión del diseño de la app



Como se puede observar, Xcode permite conectar los botones directamente con la siguiente vista a ejecutar, por lo que los FlowControllers fueron descartados. Sin embargo, el patrón Command podría ayudar en esos aislados casos en los que se necesite ejecutar una vista de forma manual. De este modo, y haciendo uso del FeedbackCommand, se creó la siguiente clase:

```
class OpenViewCommand: FeedbackCommand {

    private let currentNavigationController: UINavigationController
    private let viewControllerToOpen: UIViewController

    func execute()-> Bool? {
        return openViewController()
    }

    init(currentNavigationController: UINavigationController, viewControllerToOpen: UIViewController) {
        self.currentNavigationController = currentNavigationController
        self.viewControllerToOpen = viewControllerToOpen
    }

    private func openViewController()-> Bool {
        currentNavigationController.pushViewController(viewControllerToOpen, animated: true)
        return true
    }
}
```

A este comando se le debe inyectar tanto el controlador de navegación como el controlador a ejecutar. Pero, espera… si tenemos que ejecutar un comando para abrir una desde un módulo, de forma que se acceda a otro… ¡ese módulo conocería de la existencia del otro! Toda la independencia entre módulos se iría al traste. ¿Cómo solucionarlo? Creando extensiones propias del módulo que permitan ejecutar cada vista. Por ejemplo:

```
final class OpenPlanViewCommand: OpenViewCommand {

    required init(currentNavigationController: UINavigationController, workoutToOpen: WorkoutRoutine) {
        let viewControllerToOpen : CurrentPlanViewController = UIStoryboard(name: CleverFitParams.storyboardName, bundle: nil).instantiateViewController
            (withIdentifier: CleverFitParams.ViewController.currentPlanViewController.rawValue) as! CurrentPlanViewController
        viewControllerToOpen.workoutRoutine = workoutToOpen

        super.init(currentNavigationController: currentNavigationController, viewControllerToOpen: viewControllerToOpen)
    }

}
```

Esa comando, que hereda de OpenViewCommand, y que por cierto, ha sido declarado como clase final (no admite hijos), será encargado de abrir la vista del plan de entrenamiento actual. ¿Lo mejor? Puede llamarla cualquier módulo, y para nada tienen que conocer el controlador de un módulo no común.

Por cierto, como curiosidad, y con el fin de facilitar la realización del código, se ha creado una clase Config que contiene todos los nombres de las vistas:

```swift
//

import Foundation

final class CleverFitParams {

    static let storyboardName = "Main"

    enum ViewController: String {
        case historyViewController
        case exerciseViewController
        case meViewController
        case currentPlanViewController
        case trainViewController
        case trainResumeViewController
        case generateWorkoutViewController
        case registerUserViewController
        case settingsViewController
        case mainTabbarController
        case mainNavigationController
    }
```

De este modo, para ejecutar cualquier vista, sólo habría que escoger su nombre desde esta clase. Swift requiere que, para abrir una vista declarada en el panel de vistas, se utilice su identificador único. Sería muy engorroso escribir la cadena de texto constantemente. Por eso, cada vez que se añada una vista nueva, se debe registrar en este fichero su identificador único, facilitando luego su llamada:

Algo que facilita bastante la legibilidad del código.

```swift
let viewControllerToOpen : CurrentPlanViewController = UIStoryboard(name: CleverFitParams.storyboardName, bundle: nil).instantiateViewController
    (withIdentifier: CleverFitParams.ViewController.currentPlanViewController.rawValue) as! CurrentPlanViewController
```

Gestores de la base de datos: Realm

Se ha incluido un gestor de la base de datos sobre el módulo común, de modo que cualquier módulo puede hacer uso del mismo para realizar operaciones de carga, actualización eliminación y adición. Este gestor funciona sobre Realm.

Realm básicamente es una base de datos bastante completa, y de fácil uso, que se encuentra orientada para desarrolladores, y que funciona para la construcción de aplicaciones para móvil. Usándolo puedes manejar datos complejos, realizar consultas avanzadas o manejar objetos de vínculo dentro de un gráfico. Trabaja con objetos nativos que son asignados de forma dinámica, bajo el uso de un motor personalizado de base de datos. Este ofrece la adquisición de una API simple, mientras se mejora el rendimiento, el cual no se sacrifica por otras herramientas o acciones que trabaja el sistema. Su rendimiento se considera óptimo gracias a la asignación de memoria, a el motor de almacenamiento y a la carga lenta que hacen del trabajo algo fluido y rápido. Se le considera más rápido que un ORM, más fluido y veloz que SQLite, la base de datos móvil más famosa.

Si hablamos de compatibilidad Realm puede trabajar con distintos lenguajes; Java, Swift y Objective-C, React Native y la plataforma Xamarin. En cuanto a la depuración, los archivos Realm se pueden abrir con el Navegador Realm. En el caso de que desees compartir archivos, es posible hacerlo en otras plataformas Realm y usar los mismo modelos de datos, así el modo o la estructura de trabajo se vuelve familiar y compatible al ejecutar esta acción.

De este modo, un objeto de Realm sería definido como una entidad con propiedades dinámicas específicas. Por ejemplo:

```
import RealmSwift
import Foundation

class WorkoutRoutine: Object {
    dynamic var id = ""
    dynamic var startDate = NSDate()
    dynamic var endDate = NSDate().add(days: 30)
    let workoutExercises = List<WorkoutExercise>()
}
```

En esta clase, en la que definimos la entidad "Rutina de entrenamiento", tenemos varias propiedades. Estas propiedades nunca pueden ser nulas, es por ello que están inicializadas con valores por defecto. Sin embargo, al crear un nuevo objeto, sus propiedades pueden ser modificadas antes de añadir al mismo a la base de datos.

Como hemos indicado antes, para modificar el estado de la base de datos, basta con utilizar la clase del módulo Common llamada DatabaseManager, que se encarga de realizar, de forma transparente, todas las operaciones, e indicar, para cada una de ellas, si se ha realizado correctamente o no. De este modoi, tenemos operaciones de…

- **Carga de datos:**

```
func load() -> User? {
    return realm.objects(User.self).first as User?
}

func load() -> [WorkoutRoutine]? {
    let storedRoutines = realm.objects(WorkoutRoutine.self)
    return Array(storedRoutines)
}
```

- **Modificación de datos:**

```
func update(user: User) -> Bool {
    var updated = false
    let storedUser = realm.objects(User.self).first
    try! realm.write {
        storedUser!.birthDate = user.birthDate
        storedUser!.height = user.height
        storedUser!.name = user.name
        storedUser!.objectiveFeedback = user.objectiveFeedback
        storedUser!.userAlertsPreference = user.userAlertsPreference
        storedUser!.userExperience = user.userExperience
        storedUser!.weight = user.weight
        storedUser!.maxRoutineDurationInSeconds = user.maxRoutineDurationInSeconds
        updated = true
    }
    return updated
}

func update(routine: WorkoutRoutine) -> Bool {
    var updated = false
    let storedRoutine = realm.objects(WorkoutRoutine.self).filter("id = \(routine.id)").first
    try! realm.write {
        storedRoutine!.startDate = routine.startDate
        storedRoutine!.endDate = routine.endDate
        storedRoutine!.workoutExercises.removeAll()
        storedRoutine!.workoutExercises.append(objectsIn: routine.workoutExercises)
        updated = true
    }
    return updated
}
```

- **Adición de datos:**

```
func add(user: User) -> Bool {
    var added = false
    try! realm.write {
        realm.add(user)
        added = true
    }
    return added
}

func add(routine: WorkoutRoutine) -> Bool {
    var added = false
    try! realm.write {
        realm.add(routine)
        added = true
    }
    return added
}

func add(progressEntry: ProgressLogEntry) -> Bool {
    var added = false
    try! realm.write {
        realm.add(progressEntry)
        added = true
    }
    return added
}
```

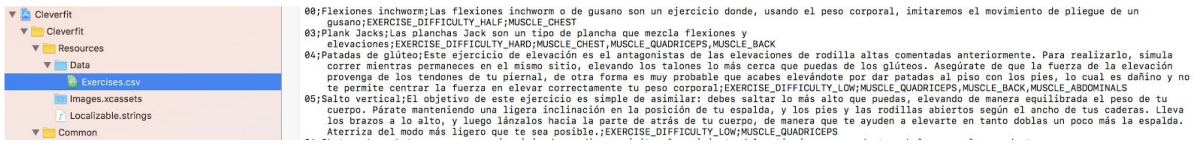
Esta clase ha sido definida en el proyecto como una clase Singleton. Si bien es cierto que muchos desarrolladores y arquitectos de software están en contra de este sistema de compartición de instancia, yo considero que todo es absolutamente relativo, y que los patrones, las arquitecturas, y los fundamentos varían en función de las necesidades del proyecto. En este caso, y al ser una base de datos local, lo veo bastante útil, y considero que su uso está totalmente justificado.

El uso de Realm ha acortado enormemente el tiempo de desarrollo. La gestión nativa de una base de datos hubiera acabado en la creación de varias clases independientes, controlado varios de los aspectos de la misma. Temporalmente, en este caso, no hubiera sido rentable.

Además, algo que queremos ahorrar de cara al usuario, es tiempo. Ahora mismo, pocas bases de datos locales superan la gran velocidad que aporta Realm.

Gestores de la base de datos: CSV

No es tan relevante como Realm, debido a que esta base de datos está oculta de cara a la API. Se trata de un gestor de CSV que nos permite extraer los ejercicios incluidos en la aplicación, y modificarlos de forma externa, sin necesidad de tocar el código. Los CSV tienen lectores y editores propios, por lo que resulta muy sencillo modificarlos como si de una hoja de excel se tratara, y luego exportarlos nuevamente. En formato plano, la base de datos de ejercicios quedaría así:



Esa información se extrae fácilmente desde el código, separando cada campo mediante los punto y coma.

Tratamiento de cadenas de texto

Seamos sinceros, una cadena de texto escrita directamente en el código no es demasiado razonable. No sólo porque no sea limpio, sino también porque dificulta los cambios de las mismas, así como las características multilenguaje de las plataformas.

Pensando en cómo solucionar ese problema, deduje que pueden utilizarse las características de Swift para idear una solución no sólo limpia, sino también acorde a las necesaidades del proyecto. De este modo, se ha hecho uso de las **extensiones** del lenguaje,

Las **extensiones** son definiciones específicas de Swift que permiten extender la funcionalidad de una clase. de tal modo que podríamos definir la clase base, con su responsabilidad única, y extenderla con diferentes responsabilidades. Esto se explicará más adelante, pero para lo que ahora nos conviene, cabe destacar que todas las Strings se han declarado en un fichero, y que este fichero es accedido mediante una extensión de la clase nativa String. De este modo, la clase accedería al fichero con una propiedad (que no función), de la siguiente manera:

```
var localized: String {
    return NSLocalizedString(self, comment: "")
}
```

Pero, espera, ¿una propiedad? ¿esto no debería hacerlo una función? Bueno, como Swift es un lenguaje reciente, explicaré esto ligeramente:

Swift 3.0 permite definir propiedades con funciones internas (esto es, con comportamiento propio). De este modo, el código queda mucho más legible. Dificulta la diferenciación entre la definición apropiada de funciones y parámetros (¿cuándo definir una función? ¿cuándo establecer un parámetro?) pero esta discusión daría para demasiadas páginas. En este caso, se ha decidido utilizar un parámetro porque la clase nativa String los utiliza muchas veces para comportamientos similares.

Bien, conocido esto, debemos saber que todas las cadenas de texto están definidas en un fichero de localización que puede tener diferentes lenguajes, y cada cadena de texto dispone de un par Código-Valor. De este modo:

```
"30_MINUTES_WORKOUT" = "30 minutos (avanzado)";

// USER EXPERIENCE
"USER_EXPERIENCE_NEW" = "Nuevo";
"USER_EXPERIENCE_HALF" = "Experimentado";
"USER_EXPERIENCE_HIGH" = "Avanzado";

// USER ALERTS CONFIGURATION
"USER_ALERT_NOTIFY" = "Notifícame";
"USER_ALERT_NOT_NOTIFY" = "No me notifiques";

// USER OBJECTIVE
"USER_OBJECTIVE_LOSE_WEIGHT" = "Perder peso";
"USER_OBJECTIVE_MAINTENANCE" = "Mantener peso";

// USER GENDER
"WOMAN" = "Mujer";
"MAN" = "Hombre";

// EXERCISE DIFFICULTY
"EXERCISE_DIFFICULTY_LOW" = "Bajo";
"EXERCISE_DIFFICULTY_HALF" = "Medio";
"EXERCISE_DIFFICULTY_HIGH" = "Difícil";

// EXERCISE FEEDBACK
"EXERCISE_FEEDBACK_BAD" = "No me gusta";
"EXERCISE_FEEDBACK_WELL" = "Me gusta";

// MUSCLES
"MUSCLE_BICEPS" = "Bíceps";
"MUSCLE_TRICEPS" = "Tríceps";
"MUSCLE_FOREARMS" = "Antebrazos";
"MUSCLE_BACK" = "Espalda";
"MUSCLE_CHEST" = "Pectorales";
"MUSCLE_ABDOMINALS" = "Abdominales";
"MUSCLE_QUADRICEPS" = "Cuádriceps";

// PLAN VIEW
"GENERATE_ROUTINE_DIALOG_TITLE" = "¡Ey!";
"GENERATE_ROUTINE_DIALOG_BODY" = "Has terminado esta rutina de ejercicios, ¿quieres que generemos una nueva?";
"GENERATE_ROUTINE_DIALOG_BUTTON" = "¡Vamos allá!";
"UPDATE_DATA_DIALOG_BUTTON" = "Quiero actualizar mis datos";
"CLOSE_DIALOG_BUTTON" = "Mejor otro día";
```

Cada vez que queramos utilizar una cadena de texto para representarla en la vista, deberíamos realizar la operación *NombreDeString.localized.* La extensión que hemos definido se encargará de buscar la cadena de texto en el proyecto. Por ejemplo:

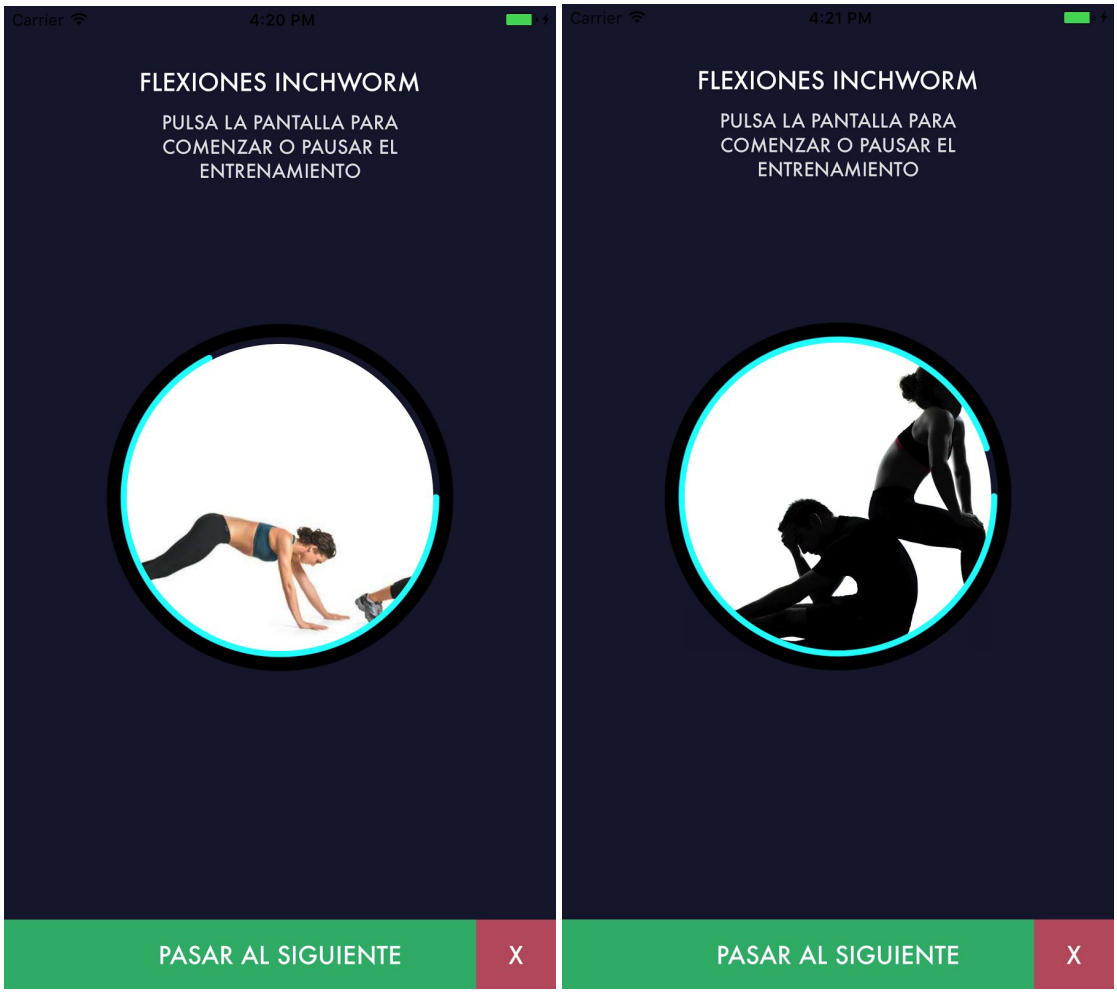
```
// USER OBJECTIVE
"USER_OBJECTIVE_LOSE_WEIGHT" = "Perder peso";
```

“USER_OBJECTIVE_LOSE_WEIGHT”.localized devolvería la cadena de texto **“Perder peso”.**

Simple, ¿verdad? Conseguimos tratar las cadenas de texto inteligentemente y, además, de forma intuitiva, de forma que admiten multilenguaje sin ningún tipo de problema.

Cronómetro de entrenamiento

Puede que a nivel de código, esto no parezca tan relevante. Pero creedme, lo es. La definición de esta clase y su arquitectura ha sido más compleja de definir. El proceso de abstracción ha tenido que ser más preciso, ya que para conseguir que el usuario pueda realizar un entrenamiento mediante la aplicación, y que la aplicación pueda cronometrar los tiempos:



(Ambas imágenes corresponden al ayudante de entrenamiento, que ayuda a realizar la rutina generada, cronometrando cada ejercicio y descanso de la rutina de ejercicios, de principio a fin. La primera imagen corresponde a un ejercicio. La segunda, al periodo de descanso tras realizarlo)

El responsable principal del ayudante de entrenamiento es el controlador de la vista se trata de un módulo independiente. Sinceramente, no creía que ese módulo tenga que encargarse de cronometrar tiempos, así que se ha creado una clase específica que permite controlarlo, y que pueda usarse fácilmente por otro módulo en caso de necesidad. Esta clase, **WorkoutExerciseTimer** de forma transparente, realiza todas las operaciones, e informa a sus delegados (en este caso, el controlador principal de la vista), sobre el estado del entrenamiento. Las operaciones que sus delegados deben implementar son:

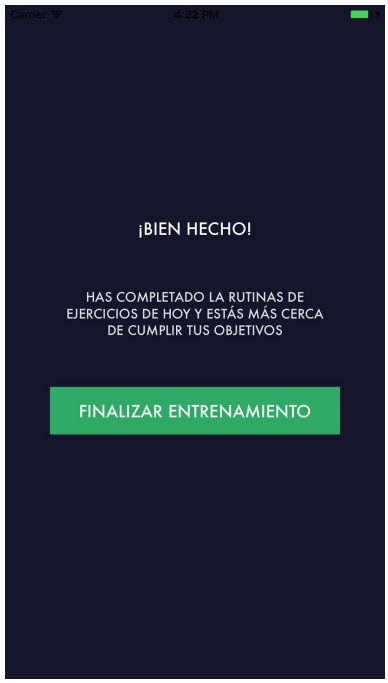
```
protocol WorkoutExerciseTimerDelegate {
    func workoutStarted()
    func workoutFinished()
    func timeUpdated(currentTimeValue: Int)
    func restTimeUpdated(currentTimeValue: Int)
    func exerciseChanged(workoutExercise: WorkoutExercise)
    func restTimeStarted(workoutExercise: WorkoutExercise)
}
```

De esta forma, aquellos que quieran hacer uso del ayudante de entrenamiento, deberán implementar su delegado, y serán informados de los cambios de estado en todo momento.

Por ejemplo, el módulo Train, que se encarga de gestionar la vista de entrenamiento, implementará el delegado de la siguiente manera que su comportamiento sea:

```
extension TrainViewController: WorkoutExerciseTimerDelegate {
    func timeUpdated(currentTimeValue: Int) {
        circularProgressView.value = CGFloat(currentTimeValue)
    }

    func restTimeUpdated(currentTimeValue: Int) {
        circularProgressView.value = CGFloat(currentTimeValue)
    }

    func workoutStarted() {
    }

    func restTimeStarted(workoutExercise: WorkoutExercise) {
        circularProgressView.maxValue = CGFloat(workoutExercise.durationInSeconds)
        circularProgressView.value = CGFloat(workoutExercise.durationInSeconds)
        if let exercise = workoutExercise.exercise {
            exerciseNameLabel.text = exercise.name.uppercased()
            ExerciseImageView.image = UIImage(named: "rest")
        }
    }

    func workoutFinished() {
        OpenTrainResumeCommand(currentNavigationController: navigationController!).execute()
    }

    func exerciseChanged(workoutExercise: WorkoutExercise) {
        circularProgressView.maxValue = CGFloat(workoutExercise.durationInSeconds)
        circularProgressView.value = CGFloat(workoutExercise.durationInSeconds)
        if let exercise = workoutExercise.exercise {
            exerciseNameLabel.text = exercise.name.uppercased()
            ExerciseImageView.image = UIImage(named: exercise.id)
        }
    }
```

- **Al actualizarse el timer** (**timeUpdated** o **restTimeUpdated**): El controlador de entrenamiento actualizará el indicador circular
- **Al cambiar de ejercicio o comenzar un tiempo de descanso** (**restTimeStarted** o **exerciseChanged**): El controlador de entrenamiento reiniciará el indicador circular
- **Al finalizar el entrenamiento** (**workoutFionished**): Se abrirá la vista de "Entrenamiento completado"



El núcleo: La Inteligencia Artificial tras Cleverfit

A pesar de que generamos rutinas deportivas en función del usuario, este proyecto se centra más en la arquitectura de la aplicación que en la propia IA debido a la especialización de Ingeniería del Software que he seleccionado. Por supuesto, eso no le resta importancia a la Inteligencia Artificial.

Disponemos de una clase **WorkoutGenerator**, que se encarga de todas las operaciones relacionadas con la generación inteligente de ejercicios. A esta clase se le deben inyectar, por defecto, el usuario actual y la lista completa de ejercicios. Además, también debe disponer del historial de ejercicios del usuario.

```
    init(forUser: User, andExercises: [Exercise]) {
        user = forUser
        exercises = andExercises
    }

    func generateRoutine(history: [WorkoutRoutine]) -> WorkoutRoutine {
        let workoutRoutine = WorkoutRoutine()

        let cleverExerciseList = CleverExerciseList(exercises: exercises, for: user, previousWorkouts: history)

        let generatedExercises = cleverExerciseList.generateRecomendedExerciseList()

        for exercise in generatedExercises {
            workoutRoutine.workoutExercises.append(exercise)
        }

        return workoutRoutine
    }
```

La función **GenerateRoutine** se encarga de generar una lista mediante la clase CleverFitExerciseList, cuyo inicializador es:

```
init(value: Exercise, for user: User, previousWorkouts: [WorkoutRoutine]) {
    self.value = value
    self.user = user
    self.previousWorkouts = previousWorkouts
}
```

Esta lista dispondrá de todos los ejercicios contenidos en una serie de nodos con valor heurístico. Realmente no es nada complejo, una lista simple sobre la que se aplica un cálculo heurístico. De hecho, el valor heurístico es una propiedad de Swift sobre la que se aplica una función matemática que considera varios parámetros:

```
var heuristicValue: Int {
    get {
        return calculateHeuristicValueForNode()
    }
}
```

La función previa quedaría definida de la siguiente manera:

```
private func calculateHeuristicValueForNode()-> Int {
    return calculateExperienceValue() + calculateExperienceValue() * 2 - numberOfTimesTrainedInParent()  - previousMuscularGroupTrainerIntensity() +
        calculateExistInPreviousRoutinesValue()
}
```

- **Experiencia del usuario**: Se incrementará el valor heurístico de un nodo en función de si la experiencia del usuario encaja con la dificultad del ejercicio. Si una persona tiene una experiencia mínima, se premiarán los ejercicios básicos, teniendo mayor valor heurístico que los ejercicios medios, y más que los ejercicios complejos.

```
private func calculateExperienceValue()-> Int {
    if user.isNotExperimented() && value.isEasy() {
        return 3
    } else if user.isNotExperimented() && value.isNormal() {
        return 2
    }

    if user.isModeratelyExperimented() && value.isNormal() {
        return 3
    } else if user.isNotExperimented() && value.isEasy() {
        return 2
    }

    if (user.isExperimented() && value.isHard()) {
        return 3
    } else if user.isNotExperimented() && value.isNormal() {
        return 2
    }

    return 0
}
```

- **Ejercicio previamente entrenado**: Debido a que nuestra IA genera rutinas de alta intensidad, y en estas destaca sobretodo el dinamismo, se intentará, en la medida de lo posible, que no se repitan los ejercicios previos. De este modo, tal vez en algún momento logre repetirse alguno debido al peso de las otras variantes del valor

heurístico final, pero si es así, será el menor número de veces posible.

```
private func numberOfTimesTrainedInParent()-> Int {
    if previousExercises == nil {
        return 0
    }

    var repetitions = 0

    for previousExercise in previousExercises! {
        if previousExercise.name.lowercased() == value.name.lowercased() {
            repetitions += 1
        }
    }

    return repetitions
}
```

- **Ejercicio entrenado en rutinas anteriores**: Si el ejercicio ha sido entrenado en la rutina anterior a la actual, entonces se disminuirá su valor heurístico tantas veces como la posición de la rutina se encuentre en la base de datos, por orden de fecha. Esto quiere decir que, si el usuario ha realizado varias rutinas, los ejercicios de la rutina más antigua tienen mayor posibilidad de aparecer que los ejercicios de la rutina anterior. De este modo se evita la repetición de ejercicios de las rutinas más recientes, lo cual evitaría que los músculos se adapten a los ejericios y que, por tanto, el

progreso no se adecuado.

```
private func calculateExistInPreviousRoutinesValue()-> Int {
    guard !previousWorkouts.isEmpty else { return 0 }

    var repetitions = 0

    for index in 0..<previousWorkouts.count {
        for workoutExercise in previousWorkouts[index].workoutExercises {
            if let exercise = workoutExercise.exercise {
                if exercise.name.lowercased() == value.name.lowercased() {
                    repetitions -= 1
                    if index == previousWorkouts.count - 1 {
                        repetitions -= index
                    }
                }
            }
        }
    }

    return repetitions
}
```

- **Si el ejercicio ya existe en la rutina actual**: Por defecto, si el ejercicio existe en la rutina actual, y con el fin de incentivar que no se repita, su valor heurístico disminuirá en 3.

```
private func calculateValueForRepetitiveExercise()-> Int {
    if exerciseExistsInParent() {
        return 0
    }
    return 3
}
```

- **Grupos musculares previamente entrenados**: Si el ejercicio actual afecta a grupos musculares previamente entrenados, tendrá mejor valor heurístico.

  Cabe destacar que en la base de datos de ejercicios se encuentra, por cada ejercicio, los grupos musculares afectados. Es más, están ordenador de forma que los primeros son los más afectados por el mismo. Y los últimos, los menos. Es por ello que el valor

heurístico disminuirá en función de esto, de tal modo que si un ejercicio que se encuentra en la rutina influye en un grupo muscular mínimamente, otro ejercicio que afecte a ese mismo grupo muscular tendrá más posibilidades, que si se tratara de un ejercicio que influyera de forma drástica.

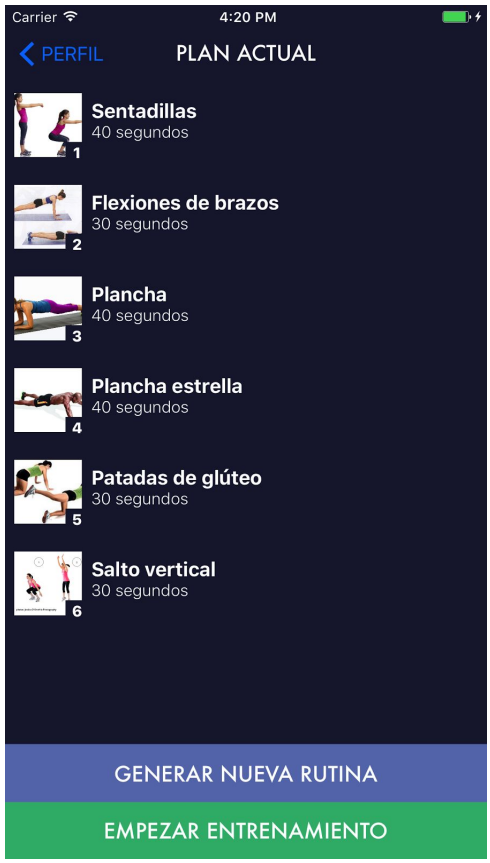
```
private func previousMuscularGroupTrainerIntensity()-> Int {
    var repetitions = 0

    if previousExercises == nil {
        return repetitions
    }

    for previousExercise in previousExercises! {
        for index in 0..<value.affectedMuscles.count {
            if previousExercise.affectedMuscles.contains(value.affectedMuscles[index]) {
                repetitions += 1 * value.affectedMuscles.count - index
            }
        }
    }

    return repetitions
}
```

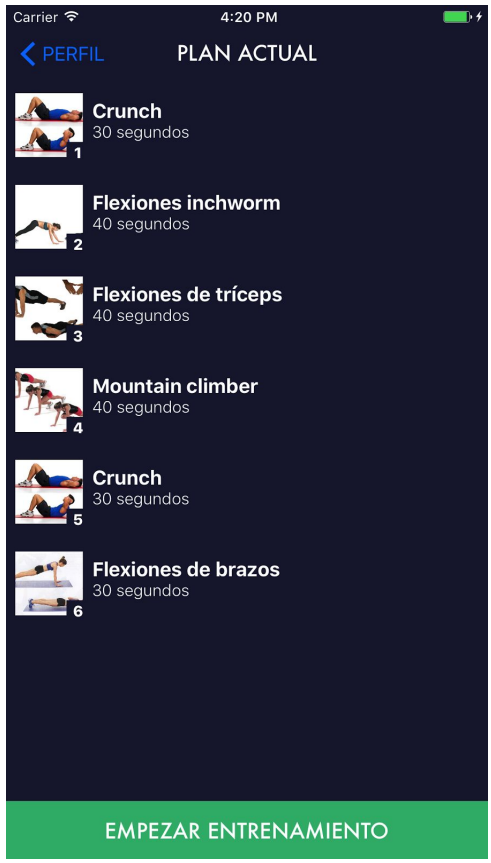
De esta forma, se garantizan rutinas dinámicas, con mínima repetición, adaptadas a la experiencia del usuario.

Aquellas clases que quieran usar el generador deberán implementar su delegado. Es conveniente saber cuándo la rutina ha comenzado, cuándo está siendo procesada, y cuándo ha terminado. Es por ello que el módulo encargado de hacer uso de esta misma clase, el **CurrentPlanViewController**, implementa el delegado de la siguiente forma:

```
extension CurrentPlanViewController: GenerateRoutineCommandDelegate {

    func generationStarted() {
        print(LocalizedString.CurrentPlanView.generationStarted)
        DispatchQueue.main.async() {
            self.trainButton.isHidden = true
            self.generateButton.isHidden = true
        }
    }

    func generationFinished(workoutRoutine: WorkoutRoutine) {
        print(LocalizedString.CurrentPlanView.generationFinished)

        if saveExercises(workoutRoutine: workoutRoutine) {
            self.workoutRoutine = workoutRoutine
            self.tableView.reloadData()
        }

        DispatchQueue.main.async() {
            self.trainButton.isHidden = false
        }
    }

    private func saveExercises(workoutRoutine: WorkoutRoutine)-> Bool {
        return DatabaseManager.sharedInstance.add(routine: workoutRoutine)
    }

}
```

Afectando a la siguiente vista adjunta, los botones de entrenar y generar se ocultarán durante la generación del ejercicio en caso de que sea necesario, y el botón de **Comenzar Entrenamiento** volverá a mostrarse cuando la rutina se haya generado correctamente:


Carrier
4:20 PM
PERFIL
PLAN ACTUAL
Sentadillas
40 segundos
1
Flexiones de brazos
30 segundos
2
Plancha
40 segundos
3
Plancha estrella
40 segundos
4
Patadas de glúteo
30 segundos
5
Salto vertical
30 segundos
6
GENERAR NUEVA RUTINA
EMPEZAR ENTRENAMIENTO


Carrier
4:20 PM
PERFIL
PLAN ACTUAL
Crunch
30 segundos
1
Flexiones inchworm
40 segundos
2
Flexiones de tríceps
40 segundos
3
Mountain climber
40 segundos
4
Crunch
30 segundos
5
Flexiones de brazos
30 segundos
6
EMPEZAR ENTRENAMIENTO

CONCLUSIONES

Resultados y grado de consecución de los objetivos

Los objetivos inicialmente establecidos fueron los siguientes:

- Desarrollar una aplicación para iOS que permita a los usuarios mejorar su calidad de vida a través del ejercicio físico.
- Facilitar el mantenimiento de peso (o bajada del mismo) mediante entrenamientos de corta duración
- Desarrollar un núcleo inteligente básico que permita que todos los entrenamientos deportivos sean diferentes.
- Desarrollar un código limpio, con el mínimo acoplamiento entre módulos y la máxima cohesión entre los mismos, aplicando los principios SOLID del desarrollo de Software.

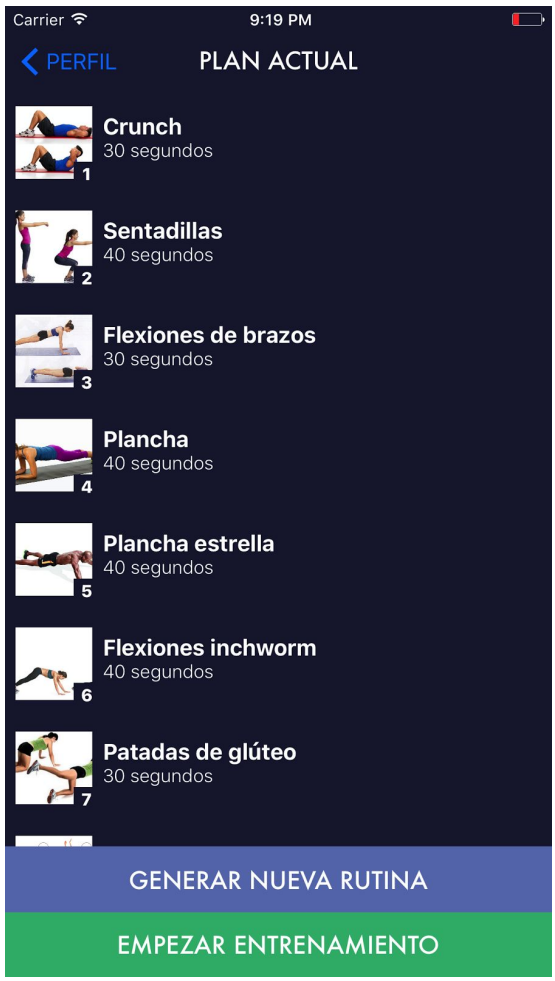
Se puede concluir que todos los objetivos han sido cumplidos correctamente. Se ha conseguido un código limpio y con una arquitectura modular, con una inteligencia capaz de generar rutinas diferentes en función de la experiencia del usuario, así como del límite de tiempo que desea usar para entrenar, y de su historial. Cabe destacar que la Inteligencia Artificial considerará el historial registrado. Si alguien tiene experiencia previa en entrenamiento, podrá indicarlo en la aplicación, pero evidentemente no podrá registrar sus

rutinas previas. Esto no es algo negativo, porque esta aplicación no está enfocada a personas extremadamente experimentadas.

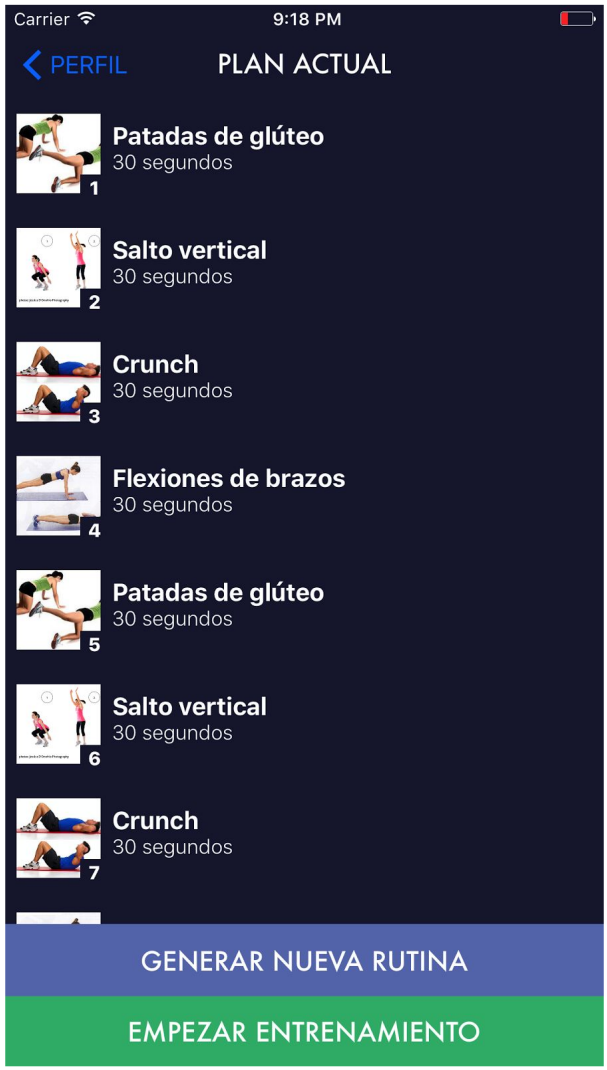
Omitiendo esto, por tanto, sí que tenemos una aplicación que cumple las promesas iniciales, así como sus objetivos. Es por ellos que consideramos que el desarrollo ha sido un éxito.

Como ejemplo, y para finalizar, se mostrará a continuación un ejemplo de diferentes rutinas para 3 individuos diferentes:

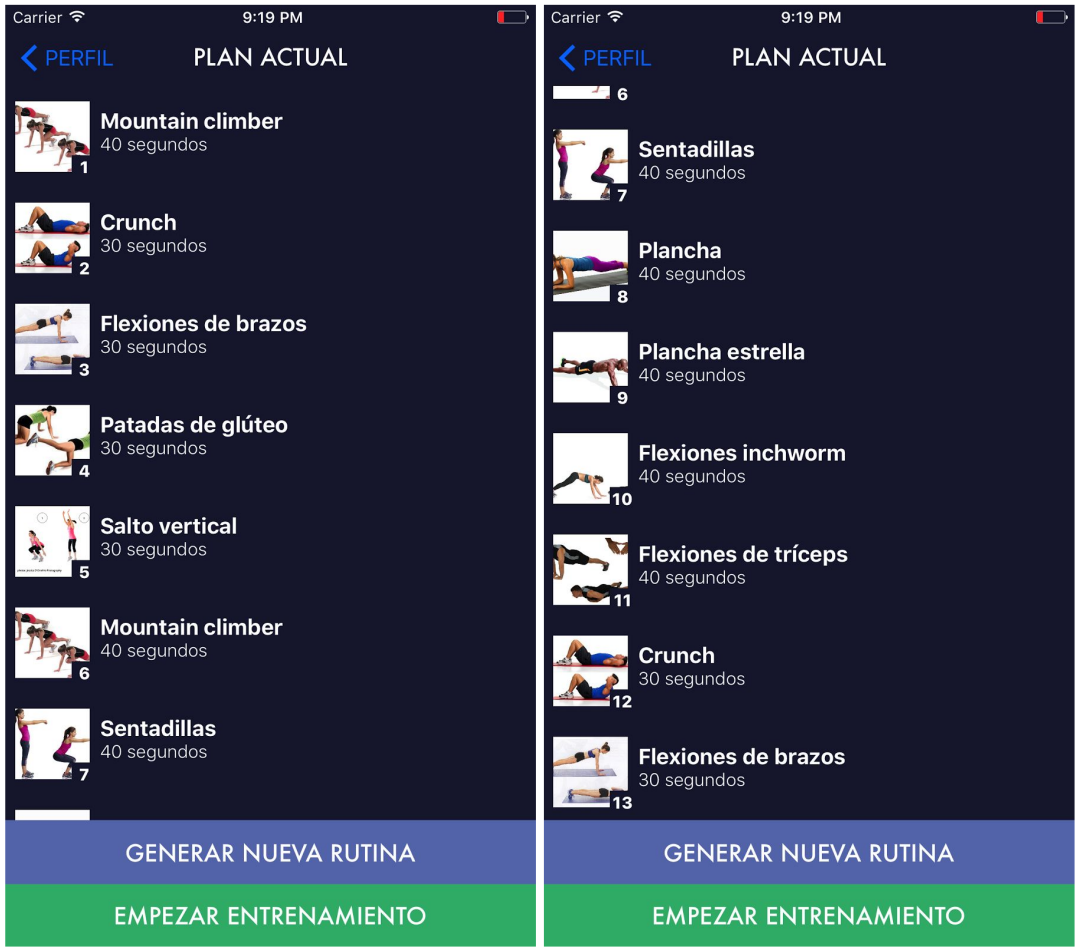
- Susana, 30 años, experiencia media, sin historial previo, 10 minutos de rutina:



- Pepe, 34 años, experiencia mínima, con historial previo, 7 minutos de rutina:



- Víctor, 22 años, experiencia avanzada, sin historial previo, 20 minutos de rutina:



TRABAJOS FUTUROS, PLAN DE EXPANSIÓN

Estrategias de expansión

Software

- **Mejora de la Inteligencia Artificial**: Para la aplicación actual, pretendemos incluir en un futuro técnicas de aprendizaje automático y mejoras en la inteligencia artificial. De este modo, podremos recopilar datos de los mejores entrenamientos para cada tipo de persona, y mediante análisis Big Data, podríamos conseguir los mejores resultados posibles en cuanto a generación de rutinas deportivas. Todos los usuarios se verían altamente beneficiados.
- **Aplicación para Android**: Incluir la versión de Android incrementaría el Target del producto considerablemente.
- **Red social interna**: Es inevitable pensar que la unión hace la fuerza. Cuando los usuarios podemos compartir nuestros progresos y compararlos con otros, logramos vernos más motivados. Realizar una red social deportiva interna sería extremadamente beneficiosos para ellos, tanto a nivel físico como a nivel mental.
- **Servidores dedicados**: Por ahora, realizar toda la base de datos en local no ha sido un problema. De hecho, es extremadamente posible que alojarla en un servidor tenga un considerable coste si se lanza al mercado. sin embargo, para cumplir este plan de expansión, es inevitable un servidor dedicado. No sólo para hacer que la Inteligencia artificial pùeda

tomar muchos más datos a la hora de generar rutinas, sino también para facilitar al usuario poder cambiar de dispositivo fácilmente.

- **Sistema de recompensas:** Añadir un sistema de logros le aportará al usuario el beneficio de cumplir objetivos acordes a su estado. Y si lo sincronizamos con las redes sociales, podría ser un gran aliciente para acercarse a ese estado saludable que tanto desea.

El software que desarrollaremos en un futuro estará enfocada a las siguientes áreas:

- Cleverfit como **sistema de administración y gestión de centros deportivos.**
- Cleverfit como **herramienta de inclusión del deporte para los más pequeños.** De este modo, se comenzaría a conocer nuestro sistema desde edades jóvenes, y estos jóvenes influyen en sus padres.
- Cleverfit para personas con **Diabetes** y **Enfermedades Crónicas del Sistema Esquelético.**

Hardware

Pretendemos que nuestro sistema inicial de Inteligencia Artificial no se quede simplemente en una herramienta deportiva vía móvil/Web, sino que avance hasta convertirse en un **completo sistema global interconectado**. Para ello, crearemos Hardware específico que nos permita una mayor adaptación de nuestros sistemas a los entornos de entrenamiento que utilice el usuario, así como a su forma de entrenar.

- **Pulseras de corrección postural** que faciliten al usuario el desarrollo funcional de los ejercicios y de su vida cotidiana.

- **Dispositivos electrónicos conectados a las diversas máquinas de los centros deportivos**, para conocer su usabilidad y disponibilidad en cada momento. De este modo, los usuarios que usen nuestra aplicación recibirán sugerencias de ejercicios alternativos si el lugar de realización del mismo está ocupado, y nuestro sistema podrá registrar directamente qué ejercicio ha hecho y está realizando qué usuario, sin necesidad de que toque la pantalla de su dispositivo móvil.

FUENTES DE INFORMACIÓN

Conocimientos propios

- Apuntes y conocimientos adquiridos a lo largo de la carrera
- Apuntes de cursos de entrenamiento personal

Fuentes web externas

- **HIT, entrenamiento de alta intensidad**:

  https://www.vitonica.com/entrenamiento/entrenamiento-de-alta-intensidad-o-hit-i
- **Los principios SOLID**: https://es.wikipedia.org/wiki/SOLID

Libros

- **Clean Code**; Robert Cecil Martel
- **The Pargmatic Programmer**; Handy Hunt & David Thomas

AGRADECIMIENTOS

La Inteligencia Artificial no fue mi única compañera durante este proceso. Por suerte, mucha gente ha aportado su grano de arena en este proyecto.

Cuando comenzamos un proyecto profesional, laboral o académico, también iniciamos un proceso de crecimiento personal bastante intenso. Al final de un proyecto, no somos iguales que al comienzo, y eso es fascinante.

Todo proyecto tiene una cara que lo representa, pero detrás de ello, detrás de todo el proceso de desarrollo siempre ha varias personas que apoyan de forma indiscutible cada pequeño paso durante el trayecto.

Por ello, quiero agradecer a **Margarita de la Iglesi**a, mi terapeuta, por asesorarme, y ayudar a que mi vida mejore hasta límites que jamás vi posibles en el pasado. Sin ella, es muy posible que nunca hubiera podido saber que tenía Asperger, y jamás hubiera evolucionado como persona. Cada día, en silencio, me recuerdo que es muy importante para mi, y todo, absolutamente todo, hubiera sido extremadamente más complejo sin ella. Margarita, muchas gracias, porque jamás hubiera conseguido presentar este proyecto sin tu ayuda. Jamás hubiera conseguido mirar a los ojos, ni tan siquiera hubiera logrado verme como alguien capaz de conseguir lo que me propongo.

A su vez, quiero agradecer a **Beatriz**, mi tutora, toda la ayuda durante estos años. He evolucionado muchísimo durante estos últimos años. Yo no soy la misma persona que

comenzó, hace 5, el GII. He crecido y evolucionado muchísimo, y debo agradecerle la confianza depositada en mí, y la ayuda que me ha otorgado. Muchas gracias, Bea, porque jamás me olvidaré de ti.

Gracias a **mis abuelos** por parte de madre, personas absolutamente asombrosas. No sé ni cuantas veces me han ayudado a lo largo de mi vida, pero estoy seguro que jamás podré agradecerlo lo suficiente. No es que tengan un hueco en mi corazón, es que son parte de él.

A mi **pareja**, por soportar, sin tener que hacerlo, todos los fines de semana que no pude dedicarle tiempo por el proyecto, y pòr comprenderme de forma tan clara y sincera. Un proyecto no es sencillo si no tienes a alguien a tu lado que te apoya tanto.

Y, por supuesto, y **para nada** menos importante, quiero agradecer **especialmente** a **mis padres**, que son el pilar fundamental de mi vida. He tenido mucha suerte con ellos, y su apoyo es incuestionablemente abismal. Sin ellos, no sólo éste proyecto no sería posible, sino que jamás hubiera conseguido aspirar siquiera a esta carrera. Me han extendido su mano cuando he caído, y me han sostenido antes de derrumbarme. Son una enorme parte de la esencia de mi existencia. Por ello, muchas gracias. Gracias por ser como sois, por aguantar mi mal humor constante, mi extraña forma de ser… gracias por hacer de mi vida algo maravilloso. Este proyecto no es mío, también es vuestro. También habéis sufrido la carrera conmigo, y jamás podré tener las palabras adecuada para deciros cuánto me habéis apoyado.