PROYECTO DE FIN DE CICLO DAM-2

Ignasio

Descripción breve

Desarrollo de aplicaciones multiplataforma para gestion de tus rutinas de gimnasio

Curso 2024 – 2025
9 de junio del 2025

Marcos Remón Blesa – mremonb@iesch.org

Contenido

1 Documento Descripción del Proyecto

1.1 Contexto del proyecto

1.1.1 Ámbitos y entorno

Hoy en día, mucha gente utiliza el móvil para casi todo, pero en los gimnasios todavía es muy habitual llevar las rutinas en papel o intentar recordarlas de memoria. Con esta app, quiero facilitar a los usuarios tener toda esa información bien organizada y disponible en cualquier momento.

El proyecto se desarrolla con la herramienta **Flutter**, que me permite crear una app que funcione tanto en Android como en iPhone. Además, uso una API hecha en **.NET y C#**, que se encarga de guardar los datos en un servidor, para que el usuario no pierda su información si cambia de móvil.

1.1.2 Análisis de la realidad

Muchas personas van al gimnasio, pero no siempre llevan un control de su rutina. Algunas usan papel o intentan recordar de memoria.

Existen aplicaciones de fitness, pero muchas de ellas están enfocadas a entrenamientos genéricos, vídeos de ejercicios o planes automáticos, pero no permiten personalizar completamente la rutina ni hacer un seguimiento detallado del progreso personal (por ejemplo: cuánto peso levantaba hace un mes y cuánto ahora).

Detecté que hay una necesidad real de contar con una app sencilla y específica para este tipo de gestión, donde el usuario pueda personalizar sus rutinas, registrar su progreso y ver su evolución de manera cómoda desde el móvil.

1.1.3 Solución y justificación de la solución propuesta

La solución que propongo es crear una aplicación móvil que permita a los usuarios gestionar de manera fácil y rápida sus rutinas de entrenamiento. La idea es que puedan crear sus propias rutinas personalizadas, registrar los ejercicios que hacen cada día y anotar los pesos, repeticiones y observaciones.

Además, la app guardará el historial de los entrenamientos, de forma que el usuario pueda ver su progreso a lo largo del tiempo. Por ejemplo, podrá comprobar si ha aumentado el peso en determinados ejercicios o si ha mejorado en la cantidad de repeticiones.

La aplicación también permitirá modificar las rutinas en cualquier momento y adaptarlas según las necesidades del usuario, ya que en muchos casos las rutinas cambian a medida que se avanza en el entrenamiento.

1.1.4 Destinatarios

- Personas que entrenan en gimnasios
- Usuarios que entrenan en casa

1.2 Objetivos del proyecto

El objetivo principal del proyecto es desarrollar una aplicación móvil multiplataforma que permita a los usuarios gestionar de forma sencilla y personalizada sus rutinas de gimnasio.

Con esta aplicación, se busca que cada usuario pueda:

- Crear y modificar sus rutinas de entrenamiento.
- Registrar los ejercicios realizados, repeticiones, series y pesos.
- Consultar su progreso a lo largo del tiempo.
- Organizar sus entrenamientos de manera más eficiente

1.3 Objetivos del proyecto en lengua extranjera

Es el mismo objetivo ya que es una app que puede usar un inglés, un español o un indio.

2 Documentación de acuerdo del Proyecto

2.1 Requisitos funcionales y no funcionales.

Requisitos funcionales:

RF1: El usuario podrá registrarse en la aplicación y autenticarse (iniciar sesión) con su cuenta o con google.

RF2: El usuario podrá agregar, o eliminar amigos, además de ver la información de su amigo.

RF3: El usuario podrá añadir rutinas a su entrenamiento.

RF4: El usuario podrá ver sus rutinas.

RF5: El usuario podrá añadir días o eliminar días de una rutina, además de añadir o eliminar ejercicios y añadir el peso, repeticiones y series.

RF6: El usuario podrá cambiar la contraseña tanto sin conocerla en la login_screen.dart como conociéndola dentro de la app.

RF7: El usuario podrá editar su información personal.

RF8: El usuario podrá eliminar su cuenta.

R9: El usuario puede desloguearse.

R10: El usuario puede consultar su friendCode.

R11: Si el usuario es admin puede acceder a una web para gestión total.

Requisitos no funcionales:

RNF 1: La aplicación debe ser multiplataforma, es decir, funcionar tanto en Android y lo haré en iOS.

RNF2: La interfaz debe ser intuitiva y fácil de usar, incluso para usuarios sin conocimientos técnicos.

RNF 3: La aplicación debe sincronizar correctamente los datos entre el dispositivo del usuario y el servidor.

RNF4: La aplicación debe responder de manera rápida, sin tiempos de espera largos al hacer consultas a la api.

RNF5: La aplicación debe garantizar la seguridad y privacidad de los datos del usuario, especialmente los datos de autenticación.

2.2 Tareas

Hice lo primero la API junto a la creación de la BBDD lo cual me di cuenta al hacer la app que fue un error y habría sido mejor hacer primero la app y hacer los endpoints a medida de necesitarlos, luego hice la app con jwt, luego hice la web con jwt y por último subí todo a un servidor temporal Debian hasta que me compre una raspi, fui probando todo a medida que lo hacía, pero al subirlo al servidor probé la app ya terminada.

2.3 Metodología a seguir para la realización del proyecto.

Para el desarrollo de este proyecto, para avanzar por fases, en vez de ir haciendo todo a la vez probando y ajustando a medida que me iban según las necesidades que iban surgiendo.

Primero desarrollé la API y la base de datos para tener el backend listo.

1- Después trabajé en la aplicación móvil, implementando funciones y añadiendo los endpoints de la API según me iban haciendo falta.
2- Luego desarrollé la aplicación web utilizando la API creada.
3- Fui probando constantemente cada parte y haciendo ajustes para arreglar errores
4- Finalmente, subí la API y la web en un servidor temporal Debian para poder hacer pruebas de acceso remoto y funcionamiento en un entorno real.

2.4 Planificación temporal de tareas.

- 04/04/2025 - 17/04/2025 -> Full Api
- 02/05/2025 - 17/05/2025 -> Full Api
- 18/05/2025 - 04/06/2025 -> Full App y modificación API
- 05/06/2025 -> Full Web
- 06/06/2025 -> Instalar y configurar servidor vpn, apache y subir el servicio

2.5 Presupuesto (gastos, ingresos, beneficio)

- Horas de Desarrollo ni las sé, demasiadas… Muchas más de 40
- Futura compra de una raspi
- Gastos: raspi, tiempo.
- Beneficios: el uso de la app, monetario -> un amigo me ofrece un kebab
- Ingresos: un posible kebab

2.6 Análisis de riesgos

Riesgo	Probabilidad	Impacto	Medidas de mitigación
Cambios en los requisitos del proyecto	Media	Medio	Mantener comunicación constante, definir bien el alcance.
Problemas técnicos con la API	Baja	Alto	Realizar pruebas frecuentes, diseñar la API de forma modular y escalable.
Fallos en la autenticación JWT	Baja	Alto	Implementar pruebas de seguridad y manejo de errores.
Retrasos en el desarrollo	Media	Medio	Planificar con margen, dividir el proyecto en tareas pequeñas.
Problemas con el servidor o despliegue	Baja	Medio	Usar un entorno de pruebas, backups regulares.
Falta de experiencia en algunas tecnologías	Media	Medio	Investigar y aprender durante el desarrollo, pedir ayuda si es necesario.
Pérdida de datos	Baja	Alto	Copias de seguridad periódicas, control de versiones.

3 Documento de análisis y diseño Modelado de datos.

3.1 Análisis y diseño del sistema funcional

El Sistema functional es un servicio API REST desarrollado en c# bajo una arquitectura en capas tiene una capa se controllers, una capa de acceso a datos, una capa de control entre otras capas de utils…

3.2 Análisis y diseño de la interfaz de usuario.

- Simplicidad: Pantallas con navegación clara para evitar confusión.
- Consistencia: Uso de colores, tipografías y elementos visuales en toda la app.
- Accesibilidad: Tamaños adecuados de botones y textos para facilitar la interacción táctil.
- Feedback visual: Mensajes claros para acciones realizadas (guardado, errores, etc.).
- Pantalla de inicio y autenticación: Formulario de registro y login con validación de datos.
- Pantalla principal: Vista general de las rutinas activas y acceso a crear nuevas.
- Pantalla de gestión de rutinas: Lista de rutinas con opciones para editar, borrar o añadir ejercicios.
- Pantalla de progreso: Gráficos y listados del historial de entrenamientos y resultados.

3.3 Diseño de la arquitectura de la aplicación

3.3.1 Tecnologías/Herramientas usadas y descripción de las mismas

- .net8 c# -> para la realización de la API REST
- PostgreSQL -> gestión de bbdd
- Dart -> para la app flutter
- HTML, CSS, JavaScript -> para la web y llamada a la api
- Clean Arquitecture -> Sistema de organización por capas
- Comandos bash -> los necesarios para instalar un servidor apache, vpn
- Docker -> para subir mi servicio a el servidor

3.3.2 Arquitectura de componentes de la aplicación

La aplicación tiene una arquitectura organizada por capas para que sea fácil de mantener y de ampliar en el futuro. todos mis endpoints reciben un post ya que manejo todo con dtos para mejor mantenimiento ya que si quiero añadir 1, 10 o 100 nuevos atributos los añado al dto (Request) y ya llegan desde la capa del controller a la de bbdd sin poner todos los atributos en 3 clases

El sistema completo está compuesto por tres partes: **API REST** para el backend, **app móvil** como interfaz de usuario, y **web** para los admins, todas ellas comunicándose entre sí.

Componentes:

1. API REST (Backend)

Hecha en .NET 8 con C#.

Tiene varias capas:

- **Service:** aquí está el servicio de la api es decir los controllers appsettings…
- **Application:** aquí tengo 3 proyectos 1 para los dtos otro para todas las interfaces y otro para el application que es donde llama el controller aquí se hacen las comprobaciones de nulls emptys ….
- **Domain:** aquí tengo las entities y los enums
- **Transversal:** mi api al tener clean architecture como arquitectura hay capas que no pueden conectarse entre sí, pero transversal se conecta con todas, aquí hay utils y clases genéricas
- **Infraestrucutre:** aquí tengo el applicationdbcontext para conectarse a bbdd y el repository que es donde hago la lógica de negocio y las llamadas a bbdd

2. App móvil (Cliente Flutter)

- **UI (pantallas):** la parte visible que usa el usuario (pantalla de login, rutinas, progreso, etc.).
- **Controlles:** aquí hago la lógica de negocio necesaria para las screens
- **Providers:** recopilan la información de las screens y al igual que en la api las mando al repositroy, guardo datos en los SharedPreferences como el token email…
- **Repository:** aqui haria lógica de negocio en caso de ser necesario y llamaría al datasource
- **Datasource:** aqui haria las llamadas HTTP a la api comprobando si la api devuelve IsSuccess como true o false y devolviendo en forma de toast el mensaje que me da la api
- **Otros:** tengo una clase constantes para la api y url, tengo unas clases utils como una para hacer toast a mano, un por así decirlo errorDTO entre otros

3. Web (Cliente web)

- Hecha con HTML + CSS + JS (bastante simple).
- Tiene un login que solo acepta usuarios con role admin, y tiene implementado jwt para que no puedas acceder a las otras screens si no te has logueado
- También consume los endpoints de la API con JavaScript.

Flujo general:

1. La app/web hace una petición a la API (por ejemplo: "quiero las rutinas del usuario X con email x").
2. La API procesa la petición, consulta PostgreSQL, y devuelve el resultado en JSON.
3. La app hace algo con los datos en el caso de get-users mostrar una tabla con todos los users en la web.

Seguridad:

- Toda la autenticación se hace con JWT.
- Si no tienes un token válido no puedes llamar a los endpoints con role admin(create-admin y get-users).
- El token se guarda localmente en el móvil o navegador mientras la sesión está activa.

Infraestructura:

- Todo esto ahora mismo lo tengo subido a un servidor temporal Debian (cuando compre la Raspi lo moveré ahí).

- Uso Docker para que el servicio de la API se despliegue de forma más controlada.

4 Documento de implementación e implantación del sistema

4.1 Pruebas

No he realizado pruebas unitarias, pero todos los endpoints están probados tanto con bruno, swagger y en producción. y también en la app y la web todas las casuísticas que se me ocurrieron las probe

5 Documento de cierre

5.1 Documento de instalación y configuración.

Para usar la app correctamente debes instalarte postgreSQL crea una server llamado localhost con connection localhost y con contraseña 1234 luego en databases click derecho créate database y la nombras ignasioo con dos letras ‘o’ tras crearla clikas en la bbdd con click derecho y le das a query tool (de las ultimas opciones) y pegas el contenido PostgresDatabaseSchema.sql con eso la bbdd estará creada. Para ejecutar la api en local debes tener instalado visual studio, dentro del proyecto en la api, debes poner el proyecto WebApi como arranque default, esto en visual studio se hace yendo a la carpeta service y con click derecho sobre el proyecto seleccionas ‘Set as startup proyect’ luego vas dentro de la carpeta service -> TFC.Service.WebAPI y ahí en el appsettins.json modificas la cadena de conexión “PostgreSQLConnection” con la ip de su ordenador pero si vas a usar un emulador en vez de un dispositivo real puedes usar “localhost”, tras esos pasos pulsas f5 o le das al botón verde que pondrá http en la parte superior y se encenderá la api abriendo el swagger en un navegador, en la app de flutter hay que irse a core -> constants -> api_constants.dart y modificar la ip por la de tu pc si vas a debuguear en un dispositivo físico o 10.0.2.2 si usaras el emulador virtual, por último para que funcione la web hay que irse a presentation -> screens -> other -> admin_web_view_screen.dart y modificar la ip al final del documento además de irse a assets -> web -> js -> constants.js y modificar nuevamente la ip

5.2 Manual de usuario

No es necesario un manual de usuario para el uso de la aplicación ya que dentro de la propia aplicación hay un botón que al pulsarlo está detallado como usar todos campos de la aplicación.

5.3 Resultados obtenidos y conclusiones.

Los resultados obtenidos de la aplicación son exactamente los que quería, tener una aplicación de control del progreso en el gimnasio en vez de usar un excel en el móvil o usar una libreta, me ha llevado la realización de la aplicación aproximadamente más de 70 horas seguro no tengo un número concreto incluso no se si llegará a las 100 horas

5.4 Diario de bitácora

Empecé haciendo la API REST que no me dio excesivos problemas, ya que en las prácticas hemos rehecho un servicio en c# infinitamente más complejo que mi API así que no hubieron muchos problemas, los problemas que me tocaron resolver en la API fueron con respecto a la bbdd ya que tuve que hacer en el ApplicationDbContext de mi API todas las relaciones de las tablas en ese fichero ya que por alguna razón daban errores, y el otro principal error fue que si quería insertar datos a un usuario no era tan simple como saco al usuario del usuario saco sus splits los meto en una lista luego los ejercicios y lo añado, no tenía usar un .implement de las tablas y ahí ya me dejaba insertar o sacar los datos del usuario o sacarlos en caso de que quisiera mostrar al usuario

En la APP es donde tuve todos los problemas, para empezar el emulador no reconoce la palabra localhost tuve que poner 10.0.2.2 luego tuve que arreglar el datasource ya que si devolvió un badrequest daba excepción instantáneo aunque yo quiera manejarlo empecé haciendo el login y el register que ahí no tuve problemas, el problema ahí fue el login con google, no puedo hacer un login como tal con google ya que dentro de la app se sacan los datos desde bbdd y si un usuario se loguea con google un día pero otro día le apetece loguearse con email y password también puede hacerlo, a si tome la decisión de poner el login con google de firebase solo hasta donde se seleccione el usuario y

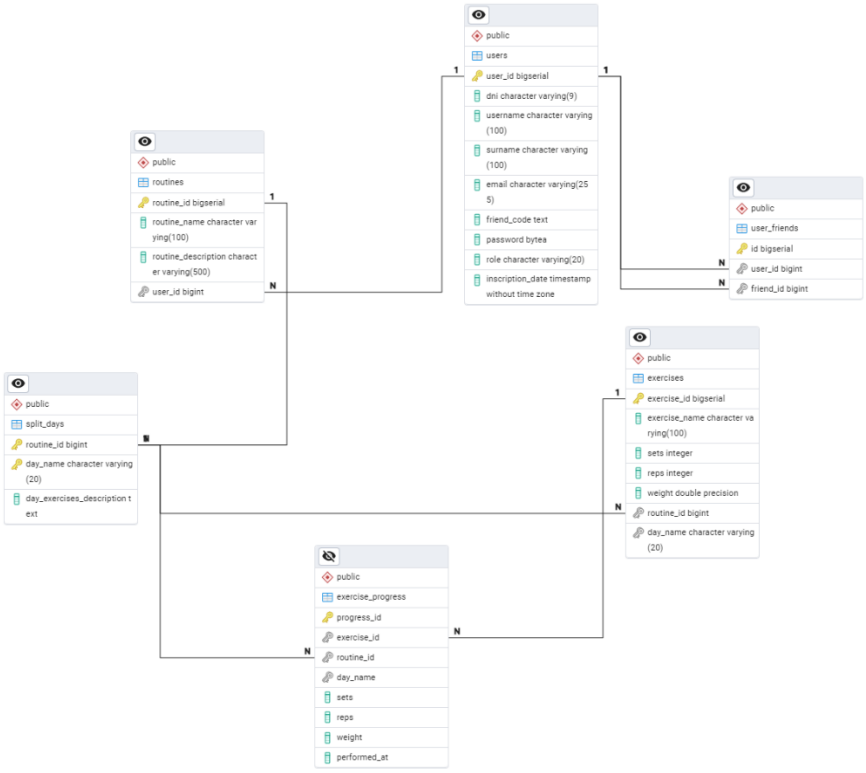
luego si no existe el usuario en bbdd hago un insert y si existe simplemente se loguea, de sus datos, si se hace un insert no se añade ni el DNI, ni su password real ya que la password la tuve que setear igual que su email porque no se puede sacar la password real, pero se le envia un correo electronico al usuario para decirle que cambie la contraseña por seguridad, dentro de la app por así decirlo, hice la screen de amigo que puedes buscar con un textfield el friendCode de un amigo y agregarlo para poder ver sus rutinas y datos, además puedes borrar a ese amigo si quieres en esta screen me tope con el problema de que se cargaban mis rutinas en la screen de mi amigo ya que el get-user-by-email lo estaba haciendo con el email del SharedPreferences, luego hice la screen de perfil ya que la de training me resulto la más complicada en profile no tuve problemas ya que es muy sencilla, training screen fue la mas complicada, al principio me lie más de lo necesario ya que puse que pudieras poner aparte del nombre del ejercicio un numero de series, repeticiones y peso que más adelante quite ya que no tiene sentido ponerlo ahí, el mayor problema aquí es que al insertar los datos en bbdd estaba mandando un enum con formato string pero me daba error de que era un int a sí que tras mucho rato decidí cambiar la bbdd porque me harte y problema arreglado

la web también me resultó muy sencilla ya que html y css lo llevamos haciendo desde smr pero js sí que me costó más.

6 Bibliografía.

Documentación de microsoft, stackoverflow, el primer enlace de búsqueda en google y por último ia

7 Anexos


public
users
user_id bigserial
dni character varying(9)
username character varying (100)
surname character varying (100)
email character varying(255)
friend_code text
password bytea
role character varying(20)
inscription_date timestamp without time zone
public
routines
routine_id bigserial
routine_name character varying(100)
routine_description character varying(500)
user_id bigint
public
user_friends
id bigserial
user_id bigint
friend_id bigint
public
split_days
routine_id bigint
day_name character varying (20)
day_exercises_description text
public
exercises
exercise_id bigserial
exercise_name character varying(100)
sets integer
reps integer
weight double precision
routine_id bigint
day_name character varying (20)
public
exercise_progress
progress_id
exercise_id
routine_id
day_name
sets
reps
weight
performed_at
1
N