ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

EPSC

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Informática

Diseño, desarrollo e implementación Front-end y Back-end de una aplicación web para gestión deportiva y de nutrición en el ámbito de un gimnasio

Manual de código

Autor: Miguel Ángel Chacón Pérez
DNI: 26829434A
Correo: i92chpem@uco.es

Director: Dr. Juan Alfonso Lara Torralbo
Correo: in2latoj@uco.es

Septiembre, 2025

UNIVERSIDAD D CÓRDOBA

Índice

1. Introducción

Este documento corresponde al manual de código de la aplicación web, y tiene como finalidad servir como guía técnica para comprender la arquitectura, configuración y despliegue de la misma.

La aplicación ha sido diseñada siguiendo buenas prácticas de seguridad, integridad de la información y control de accesos según los distintos roles definidos en el sistema. Además, implementa un diseño responsivo, garantizando que la interfaz se adapte correctamente a cualquier dispositivo desde el que se acceda.

Dentro de este manual se describen los entornos, tecnologías y configuraciones necesarias para ejecutar la aplicación, así como la estructura del código que compone el sistema.

1.1. Lenguaje de programación PHP

PHP es un lenguaje de programación ampliamente utilizado en el desarrollo de aplicaciones web. Su popularidad radica en su eficiencia, facilidad de uso y compatibilidad con múltiples plataformas.

En este proyecto, PHP se emplea principalmente en el backend, donde se encarga de procesar la lógica de negocio, gestionar la comunicación con la base de datos y exponer los servicios necesarios para el frontend.

1.2. Framework Symfony

Symfony es un framework de PHP diseñado para el desarrollo de aplicaciones web de forma rápida y estructurada. Proporciona un conjunto de componentes reutilizables que facilitan la implementación de funcionalidades comunes como:

- Autenticación y control de accesos.
- Gestión de base de datos mediante Doctrine ORM.
- Manejo de formularios y validaciones.
- Administración de cookies y sesiones.

Symfony organiza el código siguiendo el patrón Modelo-Vista-Controlador (MVC), lo que permite mantener una arquitectura clara, modular y escalable.

En esta aplicación, Symfony gestiona el backend, recibiendo las solicitudes enviadas desde el cliente, procesando los datos y respondiendo con la información necesaria a través de API REST.

1.3. Biblioteca React

React es una biblioteca de JavaScript utilizada en el frontend para construir interfaces de usuario dinámicas y reactivas.

Su funcionamiento se basa en un modelo de componentes reutilizables que permiten:

- Crear vistas modernas y responsivas.
- Manejar el estado de la aplicación de forma eficiente.

- Integrar fácilmente librerías de diseño, validación de formularios y gráficos interactivos.

En este proyecto, React consume las API expuestas por Symfony, mostrando la información al usuario en tiempo real y garantizando una experiencia fluida tanto en dispositivos móviles como en ordenadores.

1.4. Comunicación Cliente-Servidor

La aplicación se basa en una arquitectura cliente-servidor, donde el frontend (React) y el backend (Symfony) interactúan de manera continua para garantizar el correcto funcionamiento del sistema.

Cliente (React):

- El usuario accede a la aplicación desde un navegador web.
- React se encarga de renderizar la interfaz y de gestionar la interacción con el usuario.
- Cuando el usuario realiza una acción (ejemplo: iniciar sesión, crear una rutina, consultar dietas, etc.), React envía una solicitud HTTP al backend.

Servidor (Symfony):

- Symfony recibe la solicitud y la procesa mediante sus controladores.
- Si es necesario, consulta o actualiza la información en la base de datos MySQL a través de Doctrine ORM.
- Symfony genera una respuesta en formato JSON, que es enviada de vuelta al cliente.

Respuesta al Cliente:

- ❖ React interpreta la respuesta recibida y actualiza la interfaz de usuario en tiempo real.
- ❖ Esto asegura que la aplicación sea interactiva, dinámica y fácil de usar.

De esta forma, el ciclo de solicitud-respuesta permite que los usuarios interactúen con la aplicación de manera fluida, manteniendo la separación de responsabilidades:

- ❖ Symfony gestiona la lógica de negocio y el acceso a datos.
- ❖ React se encarga de la presentación y la experiencia de usuario.

1.3. Modelo-Vista-Controlador (MVC)

La aplicación sigue el patrón de arquitectura Modelo-Vista-Controlador (MVC), ampliamente utilizado en el desarrollo web moderno, lo que permite mantener una separación clara entre la lógica de negocio, la presentación y el control de las solicitudes.

1.3.1. Modelo

En Symfony, los modelos actúan como puente entre la base de datos y la lógica de negocio de la aplicación.

- ❖ Estos modelos están representados por las entidades de Doctrine, que definen la estructura de las tablas y sus relaciones.
- ❖ Mediante el mapeo objeto-relacional (ORM), las tablas de la base de datos se convierten en objetos de PHP, lo que facilita la consulta, creación,

edición y eliminación de registros sin necesidad de escribir sentencias SQL de forma directa.

- De esta manera, los modelos constituyen la fuente única de verdad sobre los datos del sistema.

1.3.2. Vista

En este proyecto, la vista está gestionada por React, que se encarga de renderizar la interfaz gráfica en el navegador.

- React administra todos los componentes visuales (formularios, tablas, gráficos, botones, etc.) y los actualiza en tiempo real cuando cambian los datos.
- En lugar de usar Twig como en un proyecto clásico de Symfony, aquí se utiliza JSX y componentes reutilizables de React, lo que permite un diseño más dinámico y responsivo.
- Estos componentes consumen datos del backend (Symfony) a través de APIs REST, mostrando información de forma clara y organizada al usuario.

1.3.3. Controlador

Los controladores en Symfony son los encargados de procesar las solicitudes que provienen de la aplicación React.

- Cada controlador gestiona una funcionalidad específica, como rutinas, dietas, usuarios, ejercicios o recursos educativos.
- Cuando el cliente (React) envía una solicitud HTTP (por ejemplo, para obtener una lista de rutinas o registrar un usuario), el controlador la recibe, utiliza los modelos para consultar o modificar los datos en la base de datos, y devuelve una respuesta en formato JSON.
- Symfony cuenta con un sistema de enrutamiento que mapea cada URL a un controlador, permitiendo organizar la aplicación de manera clara y modular.

De esta manera, la arquitectura MVC queda adaptada a tu proyecto de la siguiente forma:

- Modelo: Entidades de Doctrine en Symfony.
- Vista: Interfaz gráfica en React.
- Controlador: Clases en Symfony que procesan las solicitudes y devuelven respuestas JSON.

2. Instalación y configuración

Para facilitar el proceso de instalación, el proyecto incorpora un Makefile y archivos de configuración para Docker Compose, lo que permite poner en marcha todos los servicios necesarios de forma rápida y automatizada.

Dentro de la carpeta del proyecto “Infraestructure” se incluye la configuración principal:

- docker-compose.dev.yml: Pensado para el entorno de desarrollo local.

El archivo correspondiente contiene la configuración necesaria para levantar los contenedores y servicios básicos que utiliza la aplicación.

docker-compose.dev.yml:

```
version: '3.3'

services:

 backend:

   container_name: brainygym_backend

   build:

     context: ./services/php/symfony6

     dockerfile: Dockerfile

   ports:

     - "8081:80"

   volumes:

     - ./../backend:/var/www/html

   networks:
```

```
    - brainygym-backend


frontend:
  container_name: brainygym_frontend
  build:
    context: ./services/node/node-front
    dockerfile: Dockerfile
  environment:
    - PORT=8082
  ports:
    - "80:8082"
  tty: true
  volumes:
    - ./../frontend:/usr/src/app
  networks:
    - brainygym-backend
  command: >
   sh -c "chmod 777 -R ./*
     && yarn start || (echo 'No se ha podido arrancar npm. Hay
aplicacion en la carpeta?' && sh)"


mysql:
  container_name: brainygym_mysql
  image: mariadb:11.2.2
  env_file: ./.env
```

```
    command:
      - --max_allowed_packet=500M
      - --sql-mode=
      - --innodb-buffer-pool-size=5G
      - --innodb-flush-log-at-trx-commit=0
      - --innodb-log-file-size=2G
      - --innodb_flush_method=O_DIRECT
    volumes:
      - brainygym-data-mysql:/var/lib/mysql
      - ./sql:/root/
    networks:
      - brainygym-backend


  phpmyadmin:
    container_name: brainygym_phpmyadmin
    image: phpmyadmin/phpmyadmin
    ports:
    - "8084:80"
    environment:
      PMA_HOST: mysql
      UPLOAD_LIMIT: 400M
    networks:
      - brainygym-backend


networks:
```

```
 brainygym-backend:
    driver: bridge


volumes:
 brainygym-data-mysql:
    driver: local
```

Makefile:

```
# Levanta la arquitectura


file_selected := -f infrastructure/docker-compose.$(env).yml
environment := $(env)


up:
    @docker-compose $(file_selected) up -d


ps:
    @docker-compose $(file_selected) ps


down:
    @docker-compose $(file_selected) down
```

```
build:
	@docker-compose $(file_selected) build $(c)

restart:
	@docker-compose $(file_selected) restart $(c)

logs:
	@docker-compose $(file_selected) logs -f $(c)

logs_php:
	@docker-compose $(file_selected) exec -T php tail -f var/logs/$(environment).log

connect:
	@docker-compose $(file_selected) exec $(c) bash

connect_root:
	@docker-compose $(file_selected) exec -u root $(c) bash

copy_env_vars:
	cd infrastructure && cp .env.dist .env
	cd backend && cp .env.dist .env
	cd frontend && cp .env.dist .env
```

```
install:    copy_env_vars    up    install_dependencies    cache_clear
update_database create_admin_user



install_dependencies:

   @docker-compose $(file_selected) exec -T backend composer install

   @docker-compose $(file_selected) exec -T frontend npm install



cache_clear: up

    @docker-compose $(file_selected) exec -T backend php bin/console
cache:clear --env=dev

    @docker-compose $(file_selected) exec -T backend php bin/console
cache:clear --env=prod

      @docker-compose  $(file_selected)  exec  -T  backend  rm  -rf
var/cache/dev

      @docker-compose  $(file_selected)  exec  -T  backend  rm  -rf
var/cache/prod

     @docker-compose  $(file_selected)  exec  -T  backend  chown  -R
www-data:www-data var/

     @docker-compose  $(file_selected)  exec  -T  backend  chown  -R
www-data:www-data public/

    @docker-compose $(file_selected) exec -T backend chmod 755 -R
var/cache



diff_database:

    @docker-compose $(file_selected) exec -T backend php bin/console
doctrine:migrations:diff



update_database: up
```

```
    @docker-compose $(file_selected) exec -T backend php bin/console
doctrine:migrations:migrate --all-or-nothing --no-interaction



create_admin_user:

    @docker-compose $(file_selected) exec -T backend php bin/console
app:user:create-admin-user        --sex=male        --name=SuperAdmin
--email=i92chpem@uco.es     --password=12345678     --target_weight=80
--birthday=2001-07-20 --no-interaction



pull_code:

   git checkout develop

   git pull



deploy:  down  pull_code  up  install_dependencies  update_database
cache_clear
```

El **Makefile** centraliza los comandos necesarios para instalar, configurar y administrar la aplicación dentro de los contenedores Docker. Esto permite automatizar tareas como levantar los servicios, instalar dependencias, actualizar la base de datos o crear un usuario administrador.

A continuación, se resumen los comandos más relevantes para el correcto funcionamiento del sistema:

❖ Instalación completa (solo primera vez):

sudo make env=dev install

Este comando ejecuta todo el proceso inicial:

1. Copia los archivos .env para el backend, frontend e infraestructura.
2. Levanta los contenedores definidos en el docker-compose.dev.yml
3. Instala las dependencias de Symfony y React.
4. Limpia la caché de la aplicación.
5. Aplica las migraciones pendientes a la base de datos.
6. Crea un usuario administrador por defecto con el email i92chpem@uco.es y contraseña 12345678.

❖ Levantar los contenedores

sudo make env=dev up

Inicia los servicios definidos en el entorno de desarrollo. Este comando se utiliza siempre que la aplicación esté detenida, por ejemplo, tras reiniciar el equipo.

❖ Detener los contenedores

sudo make env=dev down

Detiene todos los servicios activos del proyecto. Es recomendable ejecutarlo si no se van a usar, para liberar recursos del sistema.

❖ Actualizar la base de datos

sudo make env=dev update_database

Este comando solo será necesario si se ha creado una nueva migración.

Con estos comandos básicos es posible instalar y poner en marcha la plataforma en cualquier equipo con Linux o macOS, aunque debe alojarse en un servidor linux.

El resto de comandos del Makefile están orientados a tareas de desarrollo y mantenimiento.

3. Arquitectura de la aplicación

La arquitectura del proyecto está compuesta por los siguientes módulos:

- Diet: Encargado de la gestión completa de las dietas. Permite la creación, edición y eliminación de dietas, así como la asociación de alimentos a días y comidas específicas y el registro diario de alimentos.
- Document: Módulo auxiliar para el almacenamiento y gestión de documentos, facilitando la subida, descarga y organización de archivos relacionados con la aplicación.
- EducativeResources: Módulo que gestiona los recursos educativos disponibles, como videos, enlaces y materiales de referencia para los usuarios.
- Exercise: Responsable de la administración de los ejercicios, incluyendo su creación, edición, clasificación en tipos y relación con rutinas.
- Food: Módulo para la gestión de alimentos, que almacena información nutricional y permite relacionarlos con dietas.
- Permission: Administra los permisos y niveles de acceso de los diferentes tipos de usuarios dentro de la aplicación.
- Routine: Responsable de la administración de las rutinas, incluyendo su creación, edición, clasificación en tipos y relación con rutinas. Además, se encarga del registro de realización de rutinas con sus ejercicios.
- User: Módulo central para la gestión de usuarios, incluyendo su registro, edición de perfiles, roles asignados y administración general.

Se mostrará de la siguiente forma para no tener que repetir a que se refiere en cada epígrafe:

- Entidad: Objeto que representa una tabla en la base de datos. Define los atributos (campos) y relaciones con otras entidades. Es la base del modelo de datos que será manipulado. **Se encuentran en backend/src/Entity**.
- Request: Objeto intermedio que encapsula y valida la información proveniente de una petición (ejemplo: datos enviados por un formulario o una API). Permite asegurar que los datos lleguen con el formato correcto antes de pasarlos a la capa de servicio. **Se encuentran en backend/src/Request**.
- Controller: Componente encargado de recibir la petición del cliente (frontend o API externa). Se limita a orquestar el flujo. **Se encuentran en backend/src/Controller.**
- Services: Contiene la lógica de negocio de la aplicación. Aquí se implementan las funcionalidades principales que manipulan entidades, validan reglas de negocio y coordinan acciones entre repositorios y controladores. **Se encuentran en backend/src/Services.**
- Repository: Encargado de la comunicación con la base de datos. Se define como una capa de acceso a datos, donde se implementan funciones para obtener, crear, actualizar o eliminar registros. Se suele utilizar en conjunto con las entidades. **Se encuentran en backend/src/Repository.**
- Vista: Interfaz presentada al usuario. En aplicaciones web, corresponde a las pantallas del frontend (formularios, listados, gráficos, etc.), que

interactúan con los controladores para mostrar datos o recoger información. **Se encuentran en frontend/src/pages.**

3.1 Diet

Funcionalidades y vistas de dietas.

3.1.1 Modelo

3.1.1.1 Entidades

Las siguientes entidades son creadas en base de datos, manteniendo la información de las dietas, su relación con los alimentos y el registro diario.

DailyIntake

Entidad para registros de ingesta diaria.

Diet

Entidad que representa una dieta con sus variables.

DietHasFood

Entidad intermedia que permite gestionar la relación muchos a muchos (N:N) entre Diet y Food, almacenando además información adicional de la relación, como la cantidad de alimento asignada por día o por comida.

3.1.1.2 Repository

Se detallan las funciones con relación directa que controlan los datos de la entidad.

DailyIntakeRepository

Gestiona las operaciones de la entidad DailyIntake, incluyendo creación, edición, eliminación y consultas. Permite filtrar y ordenar ingestas diarias, obtener registros por usuario y fecha, y calcular calorías, proteínas, carbohidratos y grasas en un rango de fechas. Utiliza Doctrine ORM y FilterService para paginación y filtrado avanzado.

DietHasFoodRepository

Gestiona la entidad intermedia DietHasFood, que representa la relación N:N entre dietas y alimentos. Permite crear, editar, eliminar y consultar relaciones por dieta, alimento o identificador, con soporte para filtros y ordenamiento. Facilita la obtención de listas paginadas y filtradas de alimentos asociados a dietas específicas.

DietRepository

Gestiona la entidad Diet, permitiendo crear, editar, eliminar y consultar dietas. Incluye métodos para obtener dietas por ID, nombre o listas completas, así como acceder a los alimentos asociados mediante DietHasFood. Soporta filtros avanzados, ordenamientos, paginación y relaciones con usuarios, roles y preferencias de objetivo.

3.1.2 Vista

AssignUserDietModal

Componente React que asigna usuarios a una dieta usando Formik y SearchableSelect. Filtra usuarios, combina con los ya asignados y envía cambios con DietService. Muestra spinner al cargar y notificaciones con toast. Incluye modal con botones de cancelar y asignar.

DietCreate

Componente React para crear dietas usando DietForm. Gestiona el envío con DietService, muestra toast según el resultado y redirige según el rol del usuario. Incluye navegación, iconos y botón de regreso.

DietEdit

Componente React para editar dietas usando DietForm. Carga los datos con DietService, permite actualizar con manejo de errores y toast, y redirige según el rol del usuario. Incluye navegación y botones de regreso.

DietView

Componente React que muestra una dieta con días y comidas en acordeón, calcula nutrición diaria, resalta calorías fuera de rango y permite edición o eliminación según privilegios. Muestra resumen nutricional y estadísticas como objetivo diario, total de alimentos y ayunos. Usa DietService, filtros, Redux y toast para notificaciones.

DietFilters

Componente React que renderiza un input de búsqueda para filtrar dietas, actualiza filtros en tiempo real con updateFilters y permite reiniciarlos con resetFilters.

DietList

Componente React que lista dietas con filtros y paginación usando CustomTable, permite crear, editar, eliminar y asignar usuarios a dietas con privilegios, maneja estado y errores, y abre un modal AssignUserDietModal para asignaciones.

DietForm

DietForm es un formulario de React con Formik y Yup para crear o editar dietas. Permite definir nombre, descripción, objetivo calórico, objetivos de dieta y alimentos por día y comida. Calcula automáticamente calorías y macronutrientes por día. Gestiona permisos de usuario, mostrando campos adicionales si es administrador. Soporta añadir, actualizar y eliminar alimentos en cada comida.

DietListWrapper

Define un componente wrapper que se encarga de envolver la lista de dietas dentro de un contexto de filtros.

index

Este archivo define un wrapper general para rutas anidadas de React Router.

UserHasDietsFilters

Vista de usuario. UserHasDietsFilters es un componente React que muestra un campo de búsqueda para filtrar usuarios con dietas. Actualiza los filtros a medida que se escribe y permite reiniciarlos. Incluye un ícono de búsqueda y un input personalizado (CustomSearchInput).

DailyIntake

Muestra la vista del registro diario de alimentos. A un lado formulario de relleno, a otro dieta asignada como referencia.

UserHasDietsList

Vista de Usuario. UserHasDietsList es un componente que lista las dietas asignadas a usuarios, aplicando filtros por usuario y mostrando información como nombre, descripción, creador, fecha y estado. Permite crear, editar, eliminar dietas y cambiar su estado "en curso". Utiliza CustomTable para la tabla con paginación y ordenación, y DietsFilters para filtrar. Incluye manejo de privilegios según rol y muestra loaders o errores según corresponda.

3.1.3 Controladores

3.1.3.1 Controller

DailyIntakeController

DailyIntakeController gestiona los endpoints relacionados con la ingesta diaria de dietas. Permite obtener, listar, crear, editar y eliminar registros mediante peticiones POST. Cada acción delega en DailyIntakeRequestService para la lógica de negocio. Está protegido con permisos de grupo diets.

DietController

DietController gestiona endpoints relacionados con las dietas. Permite obtener, listar, crear, editar y eliminar dietas, además de listar alimentos de una dieta y obtener dietas organizadas por días. Incluye un endpoint para obtener una dieta lista para edición y otro para asignar una dieta a un usuario. Toda la lógica se delega en DietRequestService y está protegida por permisos del grupo diets.

3.1.3.2 Request

CreateDailyIntakeRequest

CreateDailyIntakeRequest valida la creación de una ingesta diaria. Requiere un userId existente en la base de datos y un array de comidas (meals). Extiende BaseRequest e integra servicios de usuario, permisos y

JWT. Usa validaciones como NotBlank, NotNull y ConstraintExistsInDatabase.

CreateDietRequest

CreateDietRequest gestiona la validación al crear una dieta. Requiere nombre y objetivo, admite descripción opcional y lista de alimentos (dietFood). Incluye flags para ganar músculo, perder o mantener peso, y un userId opcional. Valida unicidad del nombre usando DietService y extiende BaseRequest con control de permisos y JWT.

DeleteDailyIntakeRequest

DeleteDailyIntakeRequest valida la eliminación de una ingesta diaria. Requiere un dailyIntakeId no nulo, en formato UUID y existente en la base de datos. Extiende BaseRequest, integrando validación con JWT y permisos de usuario mediante UserService.

DeleteDietRequest

DeleteDietRequest valida la eliminación de una dieta. Exige un dietId válido y existente en la base de datos. Usa DietService para verificar la existencia y propietario de la dieta, asegurando que solo el creador, admin o superadmin pueda eliminarla. Integra validación de permisos mediante JWT y hereda de BaseRequest.

EditDailyIntakeRequest

EditDailyIntakeRequest valida la edición de una toma diaria. Requiere dailyIntakeId, userId y dietId, asegurando que existan en la base de datos. Usa UserService y UserPermissionService para comprobar permisos, integra validación con JWT y hereda de BaseRequest para gestionar errores y consistencia.

EditDietRequest

EditDietRequest valida la edición de una dieta. Requiere dietId, name y goal, además de campos opcionales como descripción y alimentos. Comprueba que la dieta exista, que no haya duplicados en el nombre y que el usuario tenga permisos (admin, superadmin o dueño). Usa DietService y UserPermissionService junto con validaciones de BaseRequest.

GetDailyIntakeRequest

GetDailyIntakeRequest valida la consulta de una toma diaria. Requiere userId (UUID válido existente en la BD) y date. Extiende BaseRequest y usa UserService para comprobaciones adicionales. Se apoya en validaciones automáticas y en validate() para asegurar consistencia.

GetDietRequest

GetDietRequest valida la obtención de una dieta por su ID. Exige dietId no nulo y existente en la BD. Extiende BaseRequest y usa DietService para

verificar que la dieta existe. Si no se encuentra, añade error y resuelve la petición.

GetDietWithDaysRequest

GetDietWithDaysRequest valida que se pueda obtener una dieta con sus días asociados. Requiere un dietId existente en la base de datos. Usa DietService para confirmar que la dieta existe y lanza un error si no se encuentra.

ListDailyIntakeRequest

ListDailyIntakeRequest hereda de FilterRequest y se usa para listar tomas diarias aplicando filtros predefinidos, sin campos adicionales propios.

ListDietFoodRequest

ListDietFoodRequest hereda de FilterRequest y se utiliza para listar los alimentos de una dieta aplicando filtros predefinidos, sin campos adicionales propios.

ListDietRequest

ListDietRequest hereda de FilterRequest y se usa para listar dietas aplicando filtros y paginación, sin campos adicionales propios.

3.1.3.3 Services

DailyIntakeRequestService

El DailyIntakeRequestService gestiona las operaciones de ingestas diarias: crear (createDailyIntake), listar (list), obtener por usuario y fecha (getDailyIntake) y eliminar (delete). Valida usuarios, controla alimentos y cantidades, organiza los datos por tipo de comida y devuelve respuestas estandarizadas con APIJsonResponse.

DailyIntakeService

El DailyIntakeService es la capa de negocio para la entidad DailyIntake. Gestiona creación (create), edición (edit), eliminación (remove), listado (list) y obtención por usuario y fecha (getByUserAndDate), además de la eliminación masiva por usuario y fecha (deleteByUserAndDate). Todas las operaciones delegan la persistencia al DailyIntakeRepository.

DietRequestService

DietRequestService gestiona las peticiones de dietas, validando requests y llamando a los servicios. Permite crear, editar, eliminar y obtener dietas, listarlas con filtros y asignarlas a usuarios. También maneja alimentos y la estructura de dietas por días.

DietService

DietService gestiona la lógica de negocio de las dietas: crea, edita, elimina y obtiene dietas y sus relaciones con alimentos. También formatea dietas

para estructuras diarias y edición, y permite listarlas con filtros o por nombre. Además, maneja las relaciones DietHasFood entre dietas y alimentos.

3.2 Document

Funcionalidades y vistas de documentos.

3.2.1 Modelo

3.2.1.1 Entidades

Las siguientes entidades son creadas en base de datos, manteniendo la información de los documentos y tipos de documentos.

Document

La clase Document representa un documento en la base de datos, con atributos como nombre original, extensión, archivo, subdirectorio, fechas de creación y actualización, estado y tipo de documento. Incluye métodos getter y setter para todos los campos, genera automáticamente UUID como ID y marca por defecto los documentos como habilitados (status = true). Además mantiene la relación ManyToOne con DocumentType.

DocumentType

La clase DocumentType representa los tipos de documentos en el sistema, con atributos como nombre, si son obligatorios y el tipo de entidad asociado (user por defecto). Mantiene una relación OneToMany con Document y

proporciona getters y setters para todos sus campos. Inicializa por defecto requiredDocument en false y entityType en user.

3.2.1.2 Repository

Se detallan las funciones con relación directa que controlan los datos de la entidad.

DocumentRepository

El DocumentRepository gestiona la persistencia de la entidad Document. Permite crear documentos (createDocument), buscar documentos por ID (findDocument, findDocumentById) y eliminarlos (deleteDocument). Utiliza DoctrineStorableObject para operaciones de guardado y borrado, y admite devolver los resultados como objeto o array según se indique.

DocumentTypeRepository

El DocumentTypeRepository gestiona la persistencia de la entidad DocumentType. Permite buscar un tipo de documento por su ID mediante el método findDocumentType y hereda métodos de EntityRepository. También utiliza DoctrineStorableObject para operaciones de almacenamiento si fuera necesario.

3.2.2 Vista

Este módulo no dispone de ninguna vista como tal, ya que es el encargado de manejar archivos en la aplicación.

3.2.3 Controlador

3.2.3.1 Controller

Este módulo no dispone de un controlador propio, usa funciones de PublicController.

3.2.3.2 Request

GetDocumentRequest

GetDocumentRequest válida solicitudes de documentos, el campo document debe ser no nulo, no vacío y un UUID válido, verifica que el ID exista en la base de datos (Document), extiende BaseRequest para integrar validaciones automáticas y se usa para asegurar datos correctos antes de procesar la solicitud.

3.2.3.3 Services

DocumentRequestService

DocumentRequestService gestiona peticiones relacionadas con documentos, permite obtener datos (get) y renderizar documentos (render y renderDocument) usando DocumentService, integrando validación previa con GetDocumentRequest y devolviendo respuestas como APIJsonResponse o Response.

DocumentService

DocumentService gestiona documentos: permite subir (uploadDocument), renderizar (renderDocument), descargar (downloadDocument), obtener contenido o URL (getContentOfDocumentByUrl, getDocumentUrl), manejar solicitudes de subida (uploadRequest), eliminar (deleteDocument) y buscar por ID (getDocumentById), usando DocumentRepository y Filesystem para almacenamiento y respuestas HTTP con Symfony.

DocumentTypeService

DocumentTypeService gestiona roles de usuario, inicializando RoleRepository y proporcionando un método getRoleById para obtener un rol específico por su ID usando Doctrine.

3.3 EducativeResources

Funcionalidades y vistas de recursos educativos.

3.3.1 Modelo

3.3.1.1 Entidad

La siguiente entidad es creada en base de datos, manteniendo la información de los recursos educativos.

EducativeResources

La entidad EducativeResources representa un recurso educativo con UUID como identificador. Contiene campos como tag, isVideo, title,

youtubeUrl, description y createdAt. Permite diferenciar si el recurso es un video, asociar una URL de YouTube y almacenar una descripción opcional. Incluye constructores, getters y setters para gestionar sus propiedades.

3.3.1.2 Repository

Se detallan las funciones con relación directa que controlan los datos de la entidad.

EducativeResourceRepository

EducativeResourceRepository gestiona la persistencia de la entidad EducativeResources en Doctrine. Incluye métodos para buscar recursos educativos por ID o título, aplicar filtros personalizados con FilterService, y listar resultados con paginación. También permite crear, editar y eliminar registros de forma centralizada, asegurando consistencia en la base de datos. Además, implementa ordenación dinámica según los campos definidos y valores recibidos desde el filtro.

3.3.2 Vista

EducativeResourceCreate

EducativeResourceCreate en React permite crear un recurso educativo. Muestra un formulario (EducativeResourceForm) dentro de una tarjeta y gestiona el proceso de envío. Usa useState para controlar el estado de carga y EducativeResourceService para llamar al backend. Si la creación

es exitosa, muestra un toast de éxito y redirige a la lista de recursos; en caso contrario, maneja los errores con useHandleErrors. También incluye un botón para volver atrás mediante react-router-dom.

EducativeResourceEdit

EducativeResourceEdit en React permite editar un recurso educativo existente. Obtiene el recurso por id usando useFetch y EducativeResourceService, mostrando un loader mientras carga. Usa un formulario (EducativeResourceForm) prellenado con los datos del recurso, gestionando la actualización con handleUpdate. Si la edición es exitosa, muestra un toast y navega atrás; si hay errores, los maneja con useHandleErrors. Incluye un botón de volver y un diseño con tarjeta y cabecera.

EducativeResourceFilters

EducativeResourceFilters define los filtros de búsqueda para recursos educativos. Recibe updateFilters, resetFilters y los filtros actuales como props. Muestra un campo de búsqueda (CustomSearchInput) con un ícono de lupa y actualiza los filtros al escribir. Utiliza search_array como clave para enviar el texto ingresado al sistema de filtros. Es un filtro simple y reutilizable dentro de la interfaz de listado.

EducativeResourceList

Este componente EducativeResourceList muestra un listado de recursos educativos organizados por categorías (tags). Obtiene los datos desde el backend usando useFetch y EducativeResourceService, aplicando filtros gestionados por useFiltersPR. Los recursos se agrupan en un acordeón, permitiendo expandir o colapsar cada categoría. Si son videos de YouTube, se muestran con miniatura y posibilidad de reproducción dentro de la tarjeta; si no lo son, se ofrece un enlace externo. Además, permite crear, editar o eliminar recursos según permisos del usuario, mostrando confirmación al borrar y mensajes de éxito/error con toast.

EducativeResourceForm

Este formulario usa Formik y Yup para crear o editar recursos educativos. Inicializa valores desde entityData o en blanco, valida campos como título, URL, descripción, tipo y si es video. Muestra errores con invalid-feedback y aplica estilos dinámicos. Incluye inputs, un selector, textarea y un switch para indicar si es video. Al enviar, ejecuta submit y muestra un spinner si está cargando.

EducativeResourceListIndex

Define un componente wrapper que se encarga de envolver la lista de recursos educativos dentro de un contexto de filtros.

index

Este archivo define un wrapper general para rutas anidadas de React Router.

3.3.3 Controlador

3.3.3.1 Controller

EducativeResourcesController

Controlador de recursos educativos que maneja endpoints para listar, obtener, crear, editar y eliminar recursos mediante EducativeResourcesRequestService. Todas las rutas usan método POST bajo /educative-resources. Cada endpoint tiene permisos definidos con #[Permission] según acción y grupo. Se inyecta el servicio en el constructor y se manejan excepciones como NonUniqueResultException y APIException. Facilita la gestión completa de recursos educativos desde el backend de manera controlada y segura.

3.3.3.2 Request

CreateEducativeResourceRequest

Clase de request para crear un recurso educativo que valida campos obligatorios como title, youtubeUrl y tag. Incluye validación personalizada para evitar títulos duplicados usando EducativeResourcesService. Se inyectan servicios de seguridad, JWT y permisos de usuario en el constructor. Extiende BaseRequest y sobrescribe validate() para añadir

lógica específica. Permite manejar la creación de recursos con control de errores y validaciones en el backend.

DeleteEducativeResourceRequest

Clase de request para eliminar un recurso educativo que valida el campo obligatorio educativeResourceId. Utiliza EducativeResourcesService para verificar si el recurso existe antes de la eliminación. Inyecta servicios de seguridad, JWT y permisos de usuario en el constructor. Extiende BaseRequest y sobrescribe validate() para añadir la validación específica de existencia. Maneja errores si el recurso no se encuentra en la base de datos.

EditEducativeResourceRequest

Clase de request para editar un recurso educativo que valida campos obligatorios como educativeResourceId, title, youtubeUrl y tag. Extiende BaseRequest e inyecta servicios de seguridad, JWT, permisos y EducativeResourcesService. Incluye propiedades opcionales description e isVideo. Sobrescribe validate() pero actualmente solo llama a la validación base sin lógica adicional. Se utiliza para asegurar que los datos enviados al editar un recurso cumplan las restricciones básicas.

GetEducativeResourceRequest

Request para obtener un recurso educativo que valida educativeResourceId obligatorio. Extiende BaseRequest e inyecta servicios

de seguridad, JWT, permisos y EducativeResourcesService. En validate() comprueba si el recurso existe mediante el servicio. Si no existe, agrega un error y detiene la solicitud. Se usa para asegurar que se consulte un recurso válido antes de procesar la petición.

ListEducativeResourceRequest

Request para listar recursos educativos que extiende FilterRequest. No tiene lógica adicional, por lo que hereda filtros y paginación de FilterRequest. Se usa para manejar parámetros de filtrado al obtener múltiples recursos desde el backend.

3.3.3.3 Services

EducativeResourcesRequestService

Servicio que gestiona las peticiones de recursos educativos, delegando la lógica al EducativeResourcesService. Permite obtener, listar, crear, editar y eliminar recursos mediante request específicos. Cada método recibe un request, valida y llama al servicio correspondiente, devolviendo APIJsonResponse. Maneja errores como recurso inexistente y asegura mensajes de éxito estandarizados. Utiliza FilterService para aplicar filtros en listados y JWTHandlerService para la autenticación.

EducativeResourcesService

Servicio que gestiona la lógica de recursos educativos, incluyendo creación, edición, eliminación y obtención por ID o nombre. Utiliza el repositorio

EducativeResourceRepository para interactuar con la base de datos. Permite listar recursos aplicando filtros con FilterService. Cada método devuelve el recurso modificado o listado según corresponda. Incluye manejo de errores y soporte para conversiones a arrays.

3.4 Exercise

Funcionalidades y vistas de ejercicios.

3.4.1 Modelo

3.4.1.1 Entidades

Las siguientes entidades son creadas en base de datos, manteniendo la información de los ejercicios y sus tipos.

Exercise

Entidad Exercise que representa un ejercicio en la base de datos con ID UUID, nombre, descripción, estado activo y fechas de creación/actualización. Tiene relaciones con User (creador), ExerciseCategory y colecciones de RoutineHasExercise y RoutineRegisterExercises. Incluye getters y setters para todos los campos y relaciones. Inicializa createdAt y active en el constructor, y la colección de rutinas como ArrayCollection. Implementa __toString() para devolver el ID del ejercicio.

ExerciseCategory

Entidad ExerciseCategory representa categorías de ejercicios con ID UUID, nombre, descripción y fechas de creación/actualización. Tiene relación ManyToOne con User (creador) y OneToMany con Exercise. Inicializa createdAt y la colección de ejercicios en el constructor. Incluye getters y setters para todos los campos y relaciones. Sirve para organizar ejercicios dentro de categorías y permite persistencia y eliminación en cascada.

3.4.1.2 Repository

Se detallan las funciones con relación directa que controlan los datos de la entidad.

ExerciseCategoryRepository

Repositorio ExerciseCategoryRepository gestiona persistencia de categorías de ejercicios con funciones para buscar por ID o nombre, listar con filtros y paginación, y crear, editar o eliminar categorías. Implementa filtros de búsqueda por nombre y descripción y permite ordenar resultados. Utiliza Doctrine Paginator para manejar la paginación. Las funciones create, edit y remove interactúan con la entidad ExerciseCategory y el EntityManager. Sirve como capa de acceso a datos centralizada para la entidad de categorías de ejercicios.

ExerciseRepository

El ExerciseRepository gestiona la persistencia de la entidad Exercise. Permite buscar ejercicios por ID o nombre, listarlos con filtros y ordenes, y manejar relaciones con categorías, usuarios y rutinas. También incluye métodos para crear, editar y eliminar ejercicios, usando Doctrine y un sistema de paginación. Es equivalente al repositorio de categorías pero adaptado a ejercicios.

3.4.2 Vista

ExerciseCategoryCreate

Este componente ExerciseCategoryCreate renderiza un modal para crear una nueva categoría de ejercicio. Usa ExerciseService para enviar los datos al backend y muestra notificaciones con react-toastify. Maneja estado de carga (loading) y navegación (useNavigate) tras la creación exitosa. Contiene un formulario CategoryForm y un botón para cerrar el modal.

ExerciseCategoryEdit

El componente ExerciseCategoryEdit muestra un modal para editar una categoría de ejercicio existente. Obtiene los datos de la categoría mediante ExerciseService y el hook useFetch. Maneja estado de carga (loading) y errores con useHandleErrors. Usa CategoryForm para editar los campos y

react-toastify para notificaciones. Al guardar, navega hacia atrás si la edición es exitosa.

CategoriesFilters

El componente CategoryFilters es un filtro de búsqueda para categorías de ejercicios. Muestra un icono de búsqueda y un input personalizado (CustomSearchInput). Cada vez que el usuario escribe, se llama a updateFilters con el valor ingresado (search_array). También permite mostrar un valor por defecto basado en los filtros actuales (filters.filter_filters?.search_text).

ExerciseCategoryList

El componente ExerciseCategoriesList muestra un listado de categorías de ejercicios. Incluye filtros de búsqueda (CategoryFilters), paginación y ordenación mediante useFiltersPR. Permite crear, editar y eliminar categorías según privilegios del usuario (PrivilegeContext). Los datos se obtienen con useFetch desde ExerciseService. Se renderiza con CustomTable, mostrando nombre, descripción, creador y acciones, y maneja errores y estados de carga con Loader y ErrorMessage.

ExerciseCategoryListWrapper

Define un componente wrapper que se encarga de envolver la lista de tipos de ejercicios dentro de un contexto de filtros.

ExerciseCategoryForm

El componente ExerciseCategoryForm es un formulario reutilizable para crear o editar categorías de ejercicios. Usa Formik para manejar el estado del formulario y Yup para validación (nombre requerido y máximo 100 caracteres). Recibe props isLoading para mostrar un spinner, submit para enviar los datos y data para editar. Contiene inputs para nombre y descripción, muestra errores de validación y un botón de envío que cambia según el modo ("Crear" o "Editar").

index

Este archivo define un wrapper general para rutas anidadas de React Router.

ExerciseCreate

El componente ExerciseCreate permite crear un nuevo ejercicio. Contiene un formulario ExerciseForm que envía los datos mediante ExerciseService. Maneja el estado de carga (loading) y muestra notificaciones con react-toastify según el resultado. Incluye navegación para volver atrás y un diseño con SubHeader y Card para la interfaz.

ExerciseEdit

Muestra la vista del El componente ExerciseEdit permite editar un ejercicio existente. Usa useFetch para cargar los datos del ejercicio por su id y los muestra en un formulario ExerciseForm. Gestiona el estado de

carga (loading), maneja errores con useHandleErrors y muestra notificaciones con react-toastify. Incluye navegación para volver atrás y un diseño con SubHeader y Card. de edición de un ejercicio.

ExercisesFilters

ExerciseFilters es un panel lateral para filtrar ejercicios. Permite búsqueda por texto y selección de categoría. Usa updateFilters para aplicar filtros y resetFilters para reiniciarlos. Se abre/cierra con un botón y mantiene su estado con useState.

ExerciseList

ExercisesList muestra un listado de ejercicios con filtros y paginación. Permite crear, editar o eliminar ejercicios según permisos del usuario. Muestra columnas como nombre, categoría, creador y fecha, con enlaces para ver detalles. Usa useFetch para cargar datos y CustomTable para renderizar la tabla.

ExerciseView

ExerciseView muestra los detalles de un ejercicio, incluyendo imagen, video, categoría, dificultad, grupo muscular y creador. Permite editar o eliminar según permisos del usuario. Usa useFetch para obtener datos y handleConfirmationAlert para confirmaciones. Renderiza un diseño responsive con badges y sección de descripción. Navegación con botones de volver, editar y eliminar.

ExerciseForm

ExerciseForm es un formulario para crear o editar ejercicios con campos de nombre, categoría y descripción. Usa Formik y Yup para manejo de estado y validación. Renderiza errores debajo de los inputs y permite enviar datos al submit. Muestra un spinner mientras isLoading es true. Se inicializa con datos existentes si se está editando.

ExerciseListIndex

Define un componente wrapper que se encarga de envolver la lista de ejercicios dentro de un contexto de filtros.

index

Este archivo define un wrapper general para rutas anidadas de React Router.

3.4.3 Controlador

3.4.3.1 Controller

ExerciseController

ExerciseController maneja endpoints POST para ejercicios y categorías, delegando la lógica al ExerciseRequestService. Cada acción (get, list, create, edit, delete) tiene permisos específicos con #[Permission]. Permite CRUD completo de categorías y ejercicios, centralizando validaciones y

autorización. Todas las rutas son /exercises/... y usan Requests específicos para cada operación.

3.4.3.2 Request

CreateExerciseCategoryRequest

CreateExerciseCategoryRequest valida los datos para crear una categoría de ejercicio. Requiere name no vacío y opcional description. Usa ExerciseService para verificar que no exista otra categoría con el mismo nombre y agrega un error si ya existe. Extiende BaseRequest para integrar validaciones y manejo de errores de forma centralizada.

CreateExerciseRequest

CreateExerciseRequest valida los datos para crear un ejercicio. Requiere exerciseCategoryId existente, name no vacío y opcional description. Usa ExerciseService para verificar que no exista otro ejercicio con el mismo nombre y agrega un error si ya existe. Extiende BaseRequest para integrar validaciones y manejo de errores centralizado.

DeleteExerciseCategoryRequest

DeleteExerciseCategoryRequest valida que una categoría de ejercicios exista antes de eliminarla. Requiere exerciseCategoryId no vacío y que exista en la base de datos (ConstraintExistsInDatabase). Usa ExerciseService para verificar la existencia real de la categoría y agrega

un error si no se encuentra. Extiende BaseRequest para manejar validaciones y errores de forma centralizada.

DeleteExerciseRequest

DeleteExerciseRequest valida que un ejercicio exista antes de eliminarlo y controla permisos. Requiere exerciseId no vacío y que exista en la base de datos (ConstraintExistsInDatabase). Verifica que el usuario sea SuperAdmin, Admin o el creador del ejercicio; si no, lanza un error de permisos. Extiende BaseRequest para centralizar la validación y manejo de errores.

EditExerciseCategoryRequest

EditExerciseCategoryRequest valida la edición de una categoría de ejercicio. Comprueba que exerciseCategoryId exista en la base de datos y que el nuevo name no esté duplicado en otra categoría. Permite opcionalmente actualizar description. Controla errores mediante BaseRequest y lanza excepciones si las validaciones fallan.

EditExerciseRequest

EditExerciseRequest valida la edición de un ejercicio. Verifica que el exerciseId y exerciseCategoryId existan en la base de datos y que el name no esté duplicado en otro ejercicio. Además, asegura que solo el creador, un administrador o superadministrador puedan editarlo. Controla errores mediante BaseRequest y lanza excepciones si alguna validación falla.

GetExerciseCategoryRequest

GetExerciseCategoryRequest valida la obtención de una categoría de ejercicio verificando que exerciseCategoryId exista en la base de datos; si no existe, genera un error.

GetExerciseRequest

GetExerciseRequest valida la obtención de un ejercicio verificando que exerciseId exista en la base de datos; si no existe, genera un error y detiene la solicitud.

ListExerciseCategoriesRequest

ListExerciseCategoriesRequest hereda de FilterRequest y se utiliza para listar categorías de ejercicios aplicando filtros y paginación si es necesario.

ListExercisesRequest

ListExercisesRequest hereda de FilterRequest y se utiliza para listar ejercicios aplicando filtros, orden y paginación.

3.4.3.3 Services

ExerciseRequestService

El ExerciseRequestService actúa como capa intermedia entre los Request de ejercicios y categorías de ejercicios y los métodos del ExerciseService. Gestiona la lógica de negocio para crear, editar, listar, obtener y eliminar tanto ejercicios como categorías, controlando permisos, validaciones y

respuestas en formato APIJsonResponse. También aplica filtros según el usuario y maneja excepciones específicas, devolviendo mensajes claros para cada operación exitosa o fallida.

ExerciseService

El ExerciseService gestiona la lógica de negocio relacionada con ejercicios y categorías de ejercicios. Proporciona métodos para obtener, listar, crear, editar y eliminar tanto ejercicios como categorías, interactuando directamente con sus respectivos repositorios. También incluye versiones "simples" de obtención por ID y búsquedas por nombre, y utiliza filtros para listados, manejando la persistencia de datos mediante Doctrine. En resumen, es la capa que centraliza todas las operaciones CRUD de ejercicios y categorías, delegando la persistencia en los repositorios correspondientes.

3.5 Food

Funcionalidades y vistas de alimentos.

3.5.1 Modelo

3.5.1.1 Entidad

La siguiente entidad es creada en base de datos, manteniendo la información de los alimentos.

Food

La entidad Food representa un alimento con propiedades como nombre, descripción, calorías y macronutrientes. Tiene un ID UUID y está relacionada con usuarios, dietas (DietHasFood) y registros diarios de ingesta (DailyIntake). Incluye getters y setters para todas sus propiedades. Se gestiona mediante Doctrine y permite persistencia y manipulación de datos nutricionales.

3.5.1.2 Repository

Se detallan las funciones con relación directa que controlan los datos de la entidad.

FoodRepository

El FoodRepository gestiona la persistencia de alimentos en la base de datos. Permite buscar por ID o nombre, listar con filtros y paginación, crear, editar y eliminar alimentos. Usa FilterService para filtros dinámicos y Doctrine para consultas y operaciones CRUD. Incluye métodos simples y completos para obtener datos con relaciones.

3.5.2 Vista

FoodCreate

Componente React que muestra un formulario para crear alimentos, maneja el estado de carga con useState, envía los datos usando FoodService, muestra notificaciones con toast según el resultado, maneja

errores con useHandleErrors y redirige al listado de alimentos tras la creación exitosa.

FoodEdit

Componente React para editar un alimento que obtiene los datos por ID usando FoodService y useFetch, muestra un formulario FoodForm con los valores existentes, gestiona el estado de carga y errores, actualiza el alimento con la API, muestra notificaciones con toast y permite volver atrás con un botón.

FoodsFilters

Componente React FoodsFilters que muestra un buscador y un panel lateral (OffCanvas) con filtros de calorías, proteínas, grasas y carbohidratos para alimentos, actualiza los filtros mediante updateFilters, permite resetearlos con resetFilters y gestiona la apertura/cierre del menú de filtros con estado local.

FoodList

Componente `FoodList` que muestra un listado de alimentos con filtros, paginación y acciones de editar/eliminar según privilegios, usa `useFetch` para obtener datos desde `FoodService`, muestra mensajes con `toast`, y renderiza la tabla con `CustomTable` incluyendo columnas de nombre, macros, creador y fecha de creación.

FoodForm

Componente FoodForm que gestiona la creación y edición de alimentos usando Formik y Yup, incluye validación de campos como nombre, calorías, proteínas, carbohidratos, grasas y descripción, muestra errores en línea, y envía los datos al submit recibido; el botón de envío muestra un spinner mientras isLoading es true.

FoodListIndex

Define un componente wrapper que se encarga de envolver la lista de recursos educativos dentro de un contexto de filtros.

index

Este archivo define un wrapper general para rutas anidadas de React Router.

3.5.3 Controlador

3.5.3.1 Controller

FoodController

Controlador FoodController que gestiona los endpoints privados de alimentos, incluyendo obtener (getFood), listar (listFood), crear (createFood), editar (editFood) y eliminar (deleteFood), usando FoodRequestService y permisos específicos para cada acción mediante atributos #[Permission].

3.5.3.2 Request

CreateFoodRequest

Clase CreateFoodRequest que valida los datos para crear un alimento, asegurando que name, calories, proteins, carbs y fats no estén vacíos, y verifica con FoodService que no exista un alimento con el mismo nombre antes de procesar la petición.

DeleteFoodRequest

Clase DeleteFoodRequest que valida la eliminación de un alimento verificando que foodId exista en la base de datos y que el usuario tenga permisos adecuados (superadmin, admin o creador del alimento) antes de procesar la petición.

EditFoodRequest

Clase EditFoodRequest que valida la edición de un alimento, asegurando que foodId exista, que el nombre no esté duplicado y que el usuario tenga permisos (superadmin, admin o creador del alimento) antes de procesar la actualización.

GetFoodRequest

Clase GetFoodRequest que valida la obtención de un alimento asegurando que foodId exista en la base de datos antes de procesar la solicitud.

ListFoodRequest

Clase ListFoodRequest que extiende FilterRequest para manejar solicitudes de listado de alimentos con soporte de filtros y paginación.

3.5.3.3 Services

FoodRequestService

Servicio FoodRequestService que centraliza las operaciones de CRUD sobre alimentos: obtiene (getFoodRequestService), lista (listFoodRequestService), crea (createFoodRequestService), edita (editFoodRequestService) y elimina (deleteFoodRequestService) alimentos, validando permisos y usando FoodService y UserService para manejar datos y relaciones con usuarios, devolviendo siempre respuestas estándar APIJsonResponse.

FoodService

El FoodService gestiona la lógica de negocio para la entidad Food, conectando controladores con el repositorio. Permite obtener alimentos por ID o nombre, listarlos con filtros y paginación, crear nuevos alimentos asociados a un usuario, editarlos y eliminarlos. Controla validaciones básicas, como duplicados o existencia de registros. Trabaja con EntityManager y DocumentService para persistir y gestionar datos. Actúa como intermediario asegurando consistencia y permisos antes de modificar la base de datos.

3.6 Permission

Funcionalidades y vistas de recursos educativos.

3.6.1 Modelo

3.6.1.1 Entidades

Las siguientes entidades son creadas en base de datos, manteniendo la información de permisos y grupos de permisos.

Permission

La clase Permission representa un permiso del sistema asociado a un grupo, usuarios y roles. Define acción, etiqueta y descripción opcional, con indicadores de gestión por admins y dependencia de módulo. Mantiene relaciones ManyToOne y OneToMany con grupos, usuarios y roles. Incluye métodos getters y setters para acceder y modificar sus propiedades.

PermissionGroup

La clase PermissionGroup representa un grupo de permisos, manteniendo una colección de permisos asociados. Define nombre y etiqueta para identificación y organiza los permisos por acción. Incluye métodos getters y setters, además de funciones para agregar o eliminar permisos del grupo. Se relaciona con Permission mediante una relación OneToMany.

3.6.1.2 Repository

Se detallan las funciones con relación directa que controlan los datos de la entidad.

PermissionGroupRepository

El PermissionGroupRepository gestiona consultas sobre grupos de permisos. Proporciona métodos para obtener todos los permisos disponibles o solo aquellos que no son administrados (adminManaged = 0). Utiliza QueryBuilder de Doctrine y permite devolver los resultados como objetos o arrays. Integra el trait DoctrineStorableObject para operaciones de almacenamiento genéricas.

PermissionRepository

El PermissionRepository permite consultar permisos en la base de datos. Incluye métodos para obtener un permiso por su ID (findById) o varios permisos por un array de IDs (findByIds). Ambos métodos cargan también el grupo de permisos relacionado (permissionGroup) y permiten devolver los resultados como objetos o arrays. Utiliza QueryBuilder de Doctrine para construir las consultas y maneja resultados únicos o múltiples según corresponda.

3.6.2 Vista

No tiene una vista propia como tal, sino que está ligada a la sección de User.

3.6.3 Controlador

3.6.3.1 Controller

PermissionController

El PermissionController expone un endpoint /permissions/get-all que permite obtener todos los permisos. Depende de PermissionRequestService para la lógica de negocio y devuelve la respuesta como un Response. El endpoint usa el método POST y lanza excepciones si ocurre algún error durante la ejecución.

3.6.3.2 Request

Las request de permission están asociados a User, serán detalladas en su sección.

3.6.3.3 Services

PermissionRequestService

El PermissionRequestService hereda de JWTHandlerService y proporciona la lógica para manejar solicitudes relacionadas con permisos. Su método getAll() obtiene todos los permisos disponibles, diferenciando entre superadministradores y usuarios normales, usando PermissionService. Devuelve los datos dentro de un APIJsonResponse con un mensaje de éxito.

PermissionService

El PermissionService gestiona la obtención de permisos y grupos de permisos. Permite obtener un permiso por su ID (getPermissionById), todos los permisos de la plataforma filtrando según si el usuario es superadministrador o no (getAvailablePermissions), y todos los permisos sin filtro (getAll). Utiliza los repositorios PermissionRepository y PermissionGroupRepository para interactuar con la base de datos.

3.7 Routine

3.7.1 Modelo

3.7.1.1 Entidades

Las siguientes entidades son creadas en base de datos, manteniendo la información de las rutinas, tipos de rutinas, y registro de rutinas.

Routine

La entidad Routine representa una rutina de ejercicios en la aplicación. Incluye relaciones con User, RoutineCategory, ejercicios (RoutineHasExercise) y usuarios asociados (UserHasRoutine). Contiene campos de información general como nombre, descripción, fechas, estado activo y objetivos físicos o de salud. Además, gestiona indicadores de rehabilitación o corrección de articulaciones. La clase proporciona getters y setters completos para todas sus propiedades y relaciones.

RoutineCategory

La entidad RoutineCategory representa categorías de rutinas de ejercicios, vinculadas opcionalmente a un usuario creador. Mantiene una colección de rutinas (Routine) asociadas y almacena información básica como nombre, descripción, fechas de creación/actualización y estado activo. Proporciona métodos completos de getters y setters para gestionar sus propiedades y relaciones. Se utiliza UUID como identificador único.

RoutineHasExercise

La entidad RoutineHasExercise vincula ejercicios (Exercise) con rutinas (Routine) específicas, indicando sets, repeticiones, tiempo de descanso y día de la rutina. Usa UUID como identificador y guarda fechas de creación y actualización. Permite gestionar cada relación rutina-ejercicio con getters y setters completos. Se asegura integridad con CASCADE al borrar rutinas o ejercicios. Facilita la estructura de rutinas divididas por días.

RoutineRegister

La entidad RoutineRegister registra la ejecución de una rutina por un usuario, vinculando la rutina (Routine) y los ejercicios realizados (RoutineRegisterExercises). Almacena día, hora de inicio y fin, y fechas de creación y actualización. Usa UUID como identificador y asegura integridad con CASCADE al borrar usuario o rutina. Facilita seguimiento detallado de rutinas por usuario y por día.

RoutineRegisterExercises

La entidad RoutineRegisterExercises vincula un ejercicio específico (Exercise) con un registro de rutina (RoutineRegister). Guarda sets, repeticiones, peso usado y fecha de creación. Usa UUID como ID y asegura integridad con CASCADE al borrar rutina o ejercicio. Permite un seguimiento detallado del desempeño de cada ejercicio en cada sesión.

3.7.1.2 Repository

Se detallan las funciones con relación directa que controlan los datos de la entidad.

RoutineCategoryRepository

El RoutineCategoryRepository permite buscar categorías por ID o nombre, listar con filtros y paginación, crear, editar, eliminar y cambiar su estado activo. Utiliza DoctrineStorableObject para persistencia y admite resultados como objetos o arrays. Incluye métodos para aplicar filtros dinámicos y orden según campos. Se integra con las entidades Routine y User mediante joins para consultas completas.

RoutineHasExerciseRepository

El RoutineHasExerciseRepository gestiona las relaciones entre rutinas y ejercicios, permitiendo buscarlas por ID, rutina, ejercicio o combinaciones, listar con filtros y orden, y crear, editar o eliminar relaciones. Utiliza DoctrineStorableObject para la persistencia y admite resultados como

objetos o arrays. Incluye métodos para aplicar filtros por rutinas o rango de fechas y establecer orden dinámico según campos.

RoutineRegisterExercisesRepository

El RoutineRegisterExercisesRepository gestiona la relación entre registros de rutinas y ejercicios, permitiendo buscar por ID, listar con filtros y orden, crear, editar o eliminar relaciones, y obtener detalles de ejercicios por rango de fechas para un usuario específico, usando DoctrineStorableObject para la persistencia y soportando resultados como objetos o arrays.

RoutineRegisterRepository

El RoutineRegisterRepository gestiona los registros de rutinas de usuarios, permitiendo buscar por ID, listar con filtros y orden, crear, editar, eliminar y finalizar registros, obtener la rutina activa de un usuario y consultar los días de ejercicio dentro de un rango de fechas, incluyendo información de duración, ejercicios y rutinas asociadas.

RoutineRepository

El RoutineRepository gestiona la persistencia y consulta de rutinas (Routine). Permite buscar rutinas por ID, nombre o ejercicios asociados, y obtener todas o los ejercicios de una rutina específica. Soporta listados paginados con filtros avanzados (nombre, categoría, estado, usuario, objetivos, roles) y ordenamiento por múltiples campos. Facilita la creación,

edición y eliminación de rutinas, así como activar o desactivar su estado. Integra filtros complejos combinando condiciones AND y OR para atributos booleanos y relaciones con usuarios y ejercicios. Esencialmente, centraliza toda la lógica de acceso a datos de rutinas en la aplicación.

3.7.2 Vista

RoutineCategoryCreate

RoutineCategoryCreate es un componente que muestra un modal para crear una categoría de rutina. Maneja estados de carga y navegación con useState y useNavigate. Al enviar CategoryForm, llama a RoutineService y muestra un toast según el resultado. Incluye un encabezado con ícono, título y botón de cierre.

RoutineCategoryEdit

RoutineCategoryEdit es un componente que muestra un modal para editar una categoría de rutina. Obtiene los datos de la categoría usando useFetch y RoutineService. Al enviar el formulario CategoryForm, actualiza la categoría y muestra notificaciones con toast. Incluye un loader mientras carga y un encabezado con ícono, título y botón de cierre.

CategoriesFilters

CategoryFilters es un componente que muestra un input de búsqueda para filtrar categorías. Obtiene la lista de usuarios con UserService y permite actualizar los filtros mediante updateFilters. Usa useFetch para

cargar datos y CustomSearchInput para capturar la búsqueda del usuario. También integra control de privilegios con PrivilegeContext.

RoutineCategoriesList

RoutineCategoriesList es un componente que muestra un listado paginado de categorías de rutinas. Permite crear, editar y eliminar categorías según los permisos del usuario (PrivilegeContext). Integra filtros con CategoryFilters y gestiona la paginación, orden y búsqueda mediante useFiltersPR. Los datos se obtienen de RoutineService usando useFetch. El listado se renderiza con CustomTable y se muestra un loader o mensaje de error según el estado de la petición.

index

Este archivo define un wrapper general para rutas anidadas de React Router.

RotuineCategoriesListWrapper

Define un componente wrapper que se encarga de envolver la lista de categorías de rutinas dentro de un contexto de filtros.

RoutineCategoryForm

RoutineCategoryForm es un formulario reutilizable para crear o editar categorías de rutinas. Usa Formik para la gestión de estados y validación con Yup (name requerido, description opcional). Muestra campos de nombre y descripción con validación visual de errores. Incluye un botón de

envío que muestra un spinner mientras se procesa la petición. El modo del formulario se adapta según si recibe datos (Crear o Editar).

AssignUserRoutineModal

AssignUserRoutineModal es un modal que permite asignar usuarios a una rutina. Carga todos los usuarios (excluyendo roles administrativos) y combina los ya asignados aunque no estén en la lista filtrada. Usa Formik para manejar los IDs seleccionados y envía la asignación al backend con RoutineService. Incluye carga, manejo de errores y feedback con toast. Permite selección múltiple con SearchableSelect y muestra un spinner mientras se procesa la acción.

RoutineCreate

CreateRoutine es un componente que permite crear nuevas rutinas. Muestra un formulario (RoutineForm) para ingresar los datos de la rutina y maneja la creación vía RoutineService. Usa toast para feedback, gestiona la navegación según el rol del usuario (admin o usuario normal), y muestra un estado de carga mientras se procesa la acción. Incluye botones de navegación y encabezado visual con íconos y título.

RoutineEdit

EditRoutine es un componente que permite editar una rutina existente. Obtiene los datos de la rutina mediante RoutineService y los muestra en un formulario (RoutineForm) para su edición. Gestiona la actualización de

la rutina con manejo de errores y feedback con toast, y redirige según el rol del usuario (admin o usuario normal). Incluye carga (Loader) mientras se obtienen los datos y un botón de navegación para volver atrás.

RoutineView

RoutineView es un componente que muestra los detalles de una rutina y los ejercicios asignados a un día específico. Obtiene los datos de la rutina mediante RoutineService y filtra los ejercicios según el dayNumber. Permite la edición de la rutina si el usuario es admin o propietario y tiene permisos, y muestra un loader mientras se cargan los datos. Cada ejercicio se muestra con nombre, series, repeticiones y tiempo de descanso, enlazando a la vista individual del ejercicio.

RoutineFilters

RoutinesFilters es un componente que permite buscar y filtrar rutinas. Incluye un campo de búsqueda, filtros por categoría y ejercicios mediante selects y componentes de búsqueda personalizados. Los filtros se aplican en tiempo real usando updateFilters, y se pueden resetear con un botón. También utiliza un OffCanvas para mostrar los filtros de manera lateral, ofreciendo una interfaz compacta y accesible.

RoutineDaysView

RoutinesDaysView es un componente que muestra los días de una rutina y sus ejercicios asociados. Permite ver, agregar o editar días según los

permisos del usuario, usando un CustomTable para listar los días y un Popover para mostrar los ejercicios de cada día. Además, presenta detalles de la rutina, incluyendo nombre, categoría, descripción y notas, con manejo de carga y errores mediante Loader y mensajes adecuados. La navegación y acciones se controlan mediante react-router y los permisos de PrivilegeContext.

RoutinesList

RoutinesList muestra un listado de rutinas con filtros, búsqueda y paginación usando CustomTable y useFiltersPR. Permite asignar usuarios, editar o eliminar rutinas según permisos y roles. Integra filtros por categoría y ejercicios mediante RoutinesFilters. Obtiene datos de servicios (RoutineService, UserHasRoutineService) y maneja estados de carga y error. Usa Redux y PrivilegeContext para controlar permisos y visibilidad de acciones.

index

Este archivo define un wrapper general para rutas anidadas de React Router.

RoutineForm

RoutineForm es un formulario dinámico para crear o editar rutinas que gestiona información general, objetivos, días y ejercicios usando Formik y Yup. Permite seleccionar categoría de rutina, usuario (si es admin) y

añadir múltiples días y ejercicios con sets, reps y tiempo de descanso. Valida que cada día tenga al menos un ejercicio y que sets/reps no sean cero, mostrando errores correspondientes. Integra datos de categorías de rutinas, ejercicios y usuarios mediante hooks y servicios (RoutineService, ExerciseService, UserService). Ofrece controles visuales interactivos con botones para añadir/quitar días o ejercicios y envía los datos finales al submit proporcionado.

RoutinesListWrapper

Define un componente wrapper que se encarga de envolver la lista de rutinas dentro de un contexto de filtros.

ActiveRoutineModal

ActiveRoutineModal es un modal que alerta al usuario sobre una rutina activa, mostrando nombre, día e inicio si existen datos válidos, y permite cancelar, continuar o finalizar y empezar una nueva rutina, deshabilitando acciones mientras los datos no estén disponibles.

RoutineDayPlayModal

RoutineDayPlayModal es un modal que permite al usuario seleccionar un día de la rutina para iniciar, mostrando los ejercicios de cada día y ofreciendo botones para reproducir el día o cancelar la acción.

RoutineRegister

El componente RoutineRegister muestra los ejercicios de un día de rutina con sus sets, permite ingresar peso, repeticiones y descanso, valida los datos y registra los sets completados usando servicios del backend, calcula y muestra el tiempo transcurrido de la rutina, y permite finalizar la rutina asegurando que todos los ejercicios estén completos con notificaciones de éxito o error.

UserHasRoutinesFilters

El componente UserHasRoutinesFilters permite buscar y filtrar rutinas por texto, categoría y ejercicios, abre un menú lateral (OffCanvas) con opciones de filtrado avanzadas, aplica cambios usando updateFilters y permite reiniciar los filtros con resetFilters.

UserHasRoutineDaysView

Este componente UserHasRoutinesDaysView muestra los días de una rutina con sus ejercicios, permite ver, añadir o editar días según permisos del usuario, despliega detalles y descripción de la rutina, gestiona popovers con la lista de ejercicios de cada día, y maneja carga, errores y notificaciones al eliminar o interactuar con la rutina.

UserHasRoutinesList

El componente UserHasRoutinesList muestra y gestiona las rutinas asignadas a un usuario, permite filtrar, crear, editar, eliminar o iniciar

rutinas según permisos, maneja modales para seleccionar días o continuar rutinas activas, y muestra tablas con detalles como nombre, categoría, duración, creador y fecha de creación.

index

Este archivo define un wrapper general para rutas anidadas de React Router.

UserHasRoutinesListWrapper

Define un componente wrapper que se encarga de envolver la lista de recursos educativos dentro de un contexto de filtros.

3.7.3 Controlador

3.7.3.1 Controller

RoutineController

Controlador de rutinas y categorías con endpoints para crear, editar, eliminar, listar y togglear rutinas y categorías. Incluye endpoints externos para listar ejercicios y obtener rutinas formateadas por días. Gestiona permisos con #[Permission] y delega la lógica a RoutineRequestService.

RoutineRegisterController

Controlador de registros de rutinas con endpoints para crear, editar, eliminar, listar y obtener registros. Permite finalizar registros y consultar

la rutina activa de un usuario. Todos los métodos usan permisos #[Permission] y delegan la lógica a RoutineRegisterRequestService.

RoutineRegisterExercisesController

Controlador de registros de ejercicios de rutinas con endpoints para crear, editar, eliminar, listar y obtener registros de ejercicios. Cada acción requiere permisos específicos y delega la lógica a RoutineRegisterExercisesRequestService.

3.7.3.2 Request

CreateRoutineCategoryRequest

Request para crear categoría de rutina. Valida que name no esté vacío y no exista ya en la base. Tiene description opcional.

CreateRoutineRegisterExercisesRequest

Request para crear ejercicios en un registro de rutina. Valida que routineRegisterId y exerciseId existan, y que sets y reps sean obligatorios. weight es opcional.

CreateRoutineRegisterRequest

Request para crear un registro de rutina. Valida que routineId exista y que day sea obligatorio.

CreateRoutineRequest

Request para crear una rutina. Valida que routineCategoryId exista en la base de datos y que el name sea único. Permite definir ejercicios, objetivos y usuario opcional.

DeleteRoutineCategoryRequest

Request para eliminar una categoría de rutina. Valida que routineCategoryId exista en la base de datos y que sea accesible antes de proceder con la eliminación.

DeleteRoutineRegisterExercisesRequest

Request para eliminar un registro de ejercicio de rutina. Verifica que routineRegisterExerciseId exista antes de permitir la eliminación.

DeleteRoutineRegisterRequest

Request para eliminar un registro de rutina. Verifica que routineRegisterId exista antes de proceder con la eliminación.

DeleteRoutineRequest

Request para eliminar una rutina. Verifica que routineId exista y que el usuario tenga permisos (superadmin, admin o dueño de la rutina) antes de permitir la eliminación.

EditRoutineCategoryRequest

Request para editar una categoría de rutina. Verifica que routineCategoryId exista y que el nuevo name no esté duplicado en otra categoría.

EditRoutineRegisterExercisesRequest

Request para editar un registro de ejercicio de rutina. Verifica que routineRegisterExerciseId exista y permite actualizar reps y weight.

EditRoutineRegisterRequest

Request para editar un registro de rutina. Valida que routineRegisterId y routineId existan y permite actualizar startTime y endTime.

EditRoutineRequest

Request para editar una rutina existente. Valida que routineId y routineCategoryId existan, que el nombre sea único y que el usuario tenga permisos para editar la rutina.

FinishRoutineRegisterRequest

Request para finalizar un registro de rutina. Valida que routineRegisterId exista antes de permitir la finalización.

GetActiveRoutineRegisterByUserAndRoutineRequest

Request para obtener la rutina activa de un usuario específico. Valida que routineId y userId existan en la base de datos.

GetRoutineCategoryRequest

Request para obtener una categoría de rutina por su ID. Valida que routineCategoryId exista en la base de datos.

GetRoutineForEditRequest

Request para obtener una rutina específica para edición. Valida que routineId exista en la base de datos.

GetRoutineRegisterExercisesRequest

Request para obtener un ejercicio específico de un registro de rutina. Valida que routineRegisterExerciseId exista.

GetRoutineRegisterRequest

Request para obtener un registro de rutina específico. Valida que routineRegisterId exista en la base de datos.

GetRoutineRequest

Request para obtener una rutina específica. Valida que routineId exista en la base de datos y devuelve un error si no se encuentra.

GetRoutineWithDaysRequest

Request para obtener una rutina con sus días asociados. Valida que routineId exista en la base de datos y lanza error si no se encuentra.

ListRoutineCategoriesRequest

Request para listar categorías de rutina. Extiende FilterRequest, por lo que hereda filtros, paginación y ordenamiento.

ListRoutineExercisesRequest

Request para listar ejercicios de rutina. Hereda de FilterRequest, permitiendo filtros, paginación y ordenamiento.

ListRoutineRegisterExercisesRequest

Request para listar ejercicios de registros de rutina. Hereda de FilterRequest para soportar filtros, paginación y ordenamiento.

ListRoutineRegisterRequest

Request para listar registros de rutina. Hereda de FilterRequest para permitir filtros, paginación y ordenamiento.

ListRoutinesRequest

Request para listar rutinas. Extiende FilterRequest y permite filtros, paginación y ordenamiento.

ToggleRoutineCategoryRequest

Request para activar o desactivar una categoría de rutina. Valida que la categoría exista en la base de datos.

ToggleRoutineRequest

Request para activar o desactivar una rutina. Valida que la rutina exista en la base de datos.

3.7.3.3 Services

RoutineRegisterExercisesRequestService

Gestiona las peticiones relacionadas con los ejercicios del registro de una rutina, validando datos, organizando y asegurando la consistencia de los registros.

RoutineRegisterExercisesService

Se encarga de la lógica de negocio para los ejercicios del registro de una rutina.

RoutineRegisterRequestService

Gestiona las peticiones relacionadas con los registros de rutina, validando datos, organizando y asegurando la consistencia de los registros.

RoutineRegisterService

Se encarga de la lógica de negocio para los ejercicios del registro de una rutina.

RoutineRequestService

Gestiona las peticiones relacionadas con las rutinas, validando datos, organizando y asegurando la consistencia de los registros.

RoutineService

Se encarga de la lógica de negocio para los ejercicios del registro de una rutina.

3.8 User

3.8.1 Modelo

3.8.1.1 Entidades

Las siguientes entidades son creadas en base de datos, manteniendo la información de los usuarios, sus datos y roles.

Role

Entidad Role que representa un rol de usuario, con campos como nombre, descripción, activo e inmutable, relaciones con usuarios y permisos, constantes de roles principales, métodos para gestionar permisos y clonar roles, y getters/setters para todos los campos.

RoleHasPermission

Entidad intermedia que permite gestionar la relación muchos a muchos (N:N) entre Role y Permission, almacenando además información adicional de la relación.

User

Entidad User que representa un usuario con ID UUID, datos personales, credenciales, metas y flags de salud, y relaciones con roles, permisos, documentos, ejercicios, rutinas, dietas y estadísticas. Implementa JWTUserInterface y PasswordAuthenticatedUserInterface, incluye getters/setters, métodos de autenticación, gestión de roles y permisos, y funciones auxiliares como toArray y getPermissionsSchema. Contiene métodos para verificar si es superadmin o admin y manejar colecciones de relaciones.

UserHasDiet

Entidad intermedia que permite gestionar la relación muchos a muchos (N:N) entre User y Diet, almacenando además información adicional de la relación.

UserHasDocument

Entidad intermedia que permite gestionar la relación muchos a muchos (N:N) entre User y Document, almacenando además información adicional de la relación.

UserHasMentalStats

Entidad UserHasMentalStats que registra estadísticas mentales de un usuario, incluyendo mood, sleepQuality y fecha de registro recordedAt. Tiene relación ManyToOne con User, ID UUID como clave primaria, getters y setters para todos los campos, y maneja la persistencia con Doctrine.

UserHasPermission

Entidad intermedia que permite gestionar la relación muchos a muchos (N:N) entre User y Permission, almacenando además información adicional de la relación.

UserHasPhysicalStats

Entidad UserHasPhysicalStats que registra estadísticas físicas de un usuario, incluyendo weight, height, bodyFat, bmi y fecha recordedAt, con relación ManyToOne a User, ID UUID como clave primaria y métodos getters y setters para todos los campos.

UserHasRole

Entidad intermedia que permite gestionar la relación muchos a muchos (N:N) entre User y Role, almacenando además información adicional de la relación.

UserHasRoutine

Entidad intermedia que permite gestionar la relación muchos a muchos (N:N) entre User y Routine, almacenando además información adicional de la relación.

3.8.1.2 Repository

Se detallan las funciones con relación directa que controlan los datos de la entidad.

RoleRepository

El RoleRepository gestiona la entidad Role con métodos para buscar por ID o nombre, listar roles con filtros y paginación, crear, editar, eliminar y activar/desactivar roles, y agregar o remover permisos usando RoleHasPermission, soportando resultados como objetos o arrays y persistencia mediante Doctrine.

UserHasDietRepository

El UserHasDietRepository gestiona la relación entre usuarios y dietas, permitiendo crear, editar, eliminar y listar relaciones con filtros, búsquedas y paginación, además de activar/desactivar dietas, desactivar todas las dietas de un usuario, buscar por nombre, ID o usuario seleccionado, y eliminar relaciones por ID de dieta, usando Doctrine para persistencia y consultas complejas.

UserHasDocumentRepository

El UserHasDocumentRepository permite eliminar la relación de un documento con un usuario, desactivando el documento y eliminando la entidad UserHasDocument en la base de datos usando Doctrine.

UserHasRoleRepository

El UserHasRoleRepository permite obtener el rol de un usuario específico mediante su ID, incluyendo los datos del rol asociado, retornando un objeto o arreglo según se indique.

UserHasRoutineRepository

El UserHasRoutineRepository gestiona la relación entre usuarios y rutinas, permitiendo crear, editar, eliminar y listar rutinas de un usuario, aplicar filtros y órdenes personalizados, y buscar rutinas por nombre o ID, incluyendo información de ejercicios, categorías y roles del creador.

UserRepository

El UserRepository gestiona usuarios con funciones para crear, editar, eliminar, cambiar contraseña y actualizar último login; maneja roles y permisos asociados, incluyendo agregar y removerlos. Incluye filtrado y orden dinámico de usuarios mediante FilterService, con soporte para búsquedas por flags, fechas y roles. Permite manejar imágenes de perfil, estadísticas físicas y mentales, y obtener datos agregados por períodos.

Contiene métodos para calcular ingesta calórica y ejercicios registrados por fechas.

3.8.2 Vista

UserCreate

UserCreate es un componente de React que permite crear usuarios usando UserForm y UserService. Maneja el estado de carga y la navegación, mostrando notificaciones con react-toastify. Al crear el usuario exitosamente, redirige a la lista de usuarios. Incluye un botón de retroceso, un ícono de usuario y un título dentro de una tarjeta centrada. Todo se renderiza en un contenedor Page responsivo.

UserEdit

UserEdit es un componente de React que permite editar usuarios y sus permisos. Obtiene los datos del usuario por id, muestra un formulario UserForm y permite cambiar contraseña y rol. Valida cambios críticos y, si el usuario actual modifica su información importante, lo desconecta automáticamente. Incluye control de permisos para mostrar el botón de edición de permisos solo a admins o superadmins. Usa react-toastify para notificaciones, maneja estado de carga y navegación, y abre un modal UserEditPermissions para gestionar permisos.

UserEditPermissions

UserEditPermissions permite editar y restaurar permisos de un usuario, mostrando grupos en un Accordion con checkboxes y gestionando la selección, actualizando con UserService y mostrando notificaciones y spinners.

PersonalInfoCard

PersonalInfoCard muestra información personal y física del usuario, incluyendo peso, altura, IMC, estado físico y mental, calorías diarias, objetivos y necesidades, con íconos y badges visuales, calculando IMC y metas calóricas, y adaptándose a mostrar u ocultar información avanzada según props.

ReportsModal

ReportsModal es un modal que permite generar informes PDF personalizados para un usuario según un período (semanal, mensual o anual), mostrando opciones de selección, información sobre los datos incluidos en el informe y gestionando la descarga del PDF con feedback visual de carga y notificaciones de éxito o error.

UserProfile

UserProfile carga los datos del usuario y sus roles, mostrando información básica y estadísticas físicas, mentales y de calorías. Usa useFetch para obtener datos de estado físico, mental y consumo calórico, evitando

mostrarlos si el usuario es admin o superadmin viendo su propio perfil. Permite agregar estados físicos y mentales diarios mediante formularios con validación Yup. Calcula estadísticas como peso, IMC, ánimo, calidad de sueño y calorías consumidas, y genera gráficas con Chart.js. Incluye botones para generar informes PDF, editar y eliminar usuarios según permisos. Maneja modales para agregar datos diarios y mostrar informes personalizados. Controla la visualización de datos avanzados según privilegios. Utiliza Redux para obtener el usuario autenticado y moment para fechas. Implementa feedback con react-toastify y controla la recarga de datos con refreshKey.

UsersFilters

UsersFilters permite buscar y filtrar usuarios, incluye un input de búsqueda y un botón que abre un OffCanvas con filtros de fechas, actualiza los filtros al cambiar valores y permite resetearlos con un botón.

UsersList

UsersList muestra un listado de usuarios con filtros, búsqueda y paginación, permite crear, editar o eliminar usuarios según privilegios, muestra información como imagen, nombre, email, fechas, roles y permisos, utiliza CustomTable para renderizar los datos, y maneja confirmaciones y notificaciones al eliminar usuarios.

index

Este archivo define un wrapper general para rutas anidadas de React Router.

UserForm

UserForm es un formulario para crear o editar usuarios que maneja validaciones con Yup y Formik, permite ingresar datos generales como nombre, email, rol, contraseña, fecha de nacimiento, sexo y peso objetivo, incluye objetivos físicos y alimenticios con checkboxes mutuamente excluyentes, muestra select de roles solo si el usuario es admin o superadmin, y gestiona errores de validación, envío y estado de carga.

UserListIndex

Define un componente wrapper que se encarga de envolver la lista de recursos educativos dentro de un contexto de filtros.

RoleCreateModal

RoleCreateModal es un modal para crear roles que usa Formik para manejar un formulario con nombre, descripción y permisos. Carga permisos desde el backend, organizados por grupos, y permite seleccionar permisos individuales o todos de un grupo usando selectedPermissions y selectAll. Muestra un spinner mientras carga y maneja errores con useHandleErrors. Al enviar, llama a RoleService.createRole, cierra el modal y ejecuta handleSuccessCreation si la creación es exitosa.

RoleEditPermissions

RoleEditPermissions es una página que permite editar un rol específico. Obtiene el id del rol desde la URL y usa useFetch para cargar los datos del rol con RoleService.getRoleById. Muestra un spinner mientras carga y un mensaje de error si falla. Una vez obtenidos los datos, renderiza RoleForm con el nombre, descripción y permisos del rol. Incluye un botón de navegación para volver a la lista de roles.

RoleList

RoleList muestra roles en una tabla con paginación y acciones según permisos. Permite crear, editar y eliminar roles usando RoleService. Al crear o eliminar, actualiza la lista y muestra notificaciones.

index

Este archivo define un wrapper general para rutas anidadas de React Router.

RoleForm

RoleForm muestra un formulario para crear o editar roles con nombre, descripción y permisos. Usa useFormik para manejar el estado y useFetch para cargar permisos, permitiendo seleccionar todos o individualmente. Al enviar, actualiza el rol vía RoleService y muestra notificaciones.

RoleListIndex

Define un componente wrapper que se encarga de envolver la lista de roles dentro de un contexto de filtros.

3.8.3 Controlador

3.8.3.1 Controller

RoleController

Este RoleController define endpoints para manejar roles: obtener uno (get), listar todos (list), crear (create), editar (edit), editar permisos (editPermissions), activar/desactivar (toggle) y eliminar (delete). Cada método llama al RoleRequestService y está protegido con atributos Permission para controlar acceso según acción y grupo.

UserController

Este UserController expone endpoints para manejar usuarios: obtener datos (get), listar (list), crear (create), editar (edit), eliminar (delete), cambiar contraseña (changePassword), gestionar permisos (editPermissions, resetPermissions), activar/desactivar (toggle), listar roles (listRoles), agregar estadísticas físicas o mentales (createPhysicalActivity, createMentalStats), obtener estadísticas (getMentalStats, getPhysicalStats, getCalorieIntake) y generar informes (generateReport). Cada método delega la lógica al UserRequestService y usa atributos Permission para controlar accesos según acción y grupo.

UserHasDietController

Este UserHasDietController expone endpoints para manejar la relación usuario-dieta: obtener (get), listar (list), crear (create), editar (edit), eliminar (delete) y activar/desactivar (toggle) asignaciones de dietas a usuarios, delegando toda la lógica en UserHasDietRequestService y controlando accesos con atributos Permission.

UserHasRoutinesController

Este UserHasRoutinesController maneja la relación usuario-rutina con endpoints para obtener (get), listar (list), crear (create), editar (edit) y eliminar (delete) rutinas asignadas a usuarios, delegando la lógica en UserHasRoutineRequestService y controlando permisos mediante atributos Permission.

3.8.3.2 Request

AddRoleUserRequest

AssignDietToUserRequest

AssignDietToUserRequest es una clase de request que valida la asignación de una dieta a uno o varios usuarios. Contiene un array opcional userIds y un campo obligatorio dietId con validaciones (NotBlank, NotNull, existencia en base de datos). Extiende BaseRequest y recibe servicios como

UserService y UserPermissionService. La validación se ejecuta en el método validate(), que hereda la lógica de la clase base.

AssignRoutineToUserRequest

AssignRoutineToUserRequest es una clase de request que gestiona la asignación de rutinas a usuarios. Define un array opcional userIds y un campo obligatorio routineId validado con NotBlank, NotNull y una restricción que asegura su existencia en la base de datos. Extiende BaseRequest y utiliza servicios como UserService y UserPermissionService. El método validate() hereda la lógica de validación de la clase base.

ChangePasswordUserRequest

ChangePasswordUserRequest valida la petición de cambio de contraseña de un usuario. Contiene los campos userId (obligatorio y debe existir en la base de datos), password (obligatoria, mínimo 8 caracteres, con número, mayúscula y minúscula) y passwordConfirm. Extiende BaseRequest y usa UserService y UserPermissionService. En validate() comprueba si el usuario existe y si las contraseñas coinciden; en caso contrario agrega errores y detiene la petición.

CreateUserRequest

CreateUserRequest gestiona la validación al crear un usuario. Requiere campos obligatorios como name (solo letras y espacios), email, password,

birthdate, sex, targetWeight y roleId (el rol debe existir en BD). También admite opcionalmente height, weight y varias metas/condiciones booleanas. Extiende FilterRequest e inyecta UserService y UserPermissionService. En validate() comprueba que el email no esté ya registrado; si existe, agrega un error y detiene la petición.

DeleteImageUserRequest

DeleteImageUserRequest valida la petición para eliminar la imagen de un usuario. Requiere el campo obligatorio userId, que debe ser un UUID válido y existir en la base de datos (User). Extiende BaseRequest e inyecta UserService para las comprobaciones. En validate() verifica si el usuario existe; si no, añade un error y detiene la petición.

DeleteUserRequest

DeleteUserRequest valida la petición para eliminar un usuario. Exige el campo userId, obligatorio, tipo UUID y existente en la base de datos (User). Hereda de BaseRequest e inyecta UserService para las comprobaciones. En validate() comprueba si el usuario existe y, si no, agrega un error y detiene la petición.

EditPermissionsUserRequest

EditPermissionsUserRequest maneja la petición para editar permisos de un usuario. Requiere userId como UUID válido y existente en la base de

datos, además de un array de permissions. Extiende BaseRequest, recibe UserService en el constructor y en validate() comprueba que el usuario exista; si no, agrega error y corta la petición.

EditUserRequest

EditUserRequest gestiona la validación para editar usuarios. Exige un userId válido y existente, además de nombre, email, peso objetivo, sexo y fecha de nacimiento. Permite campos opcionales como roleId y varios flags de objetivos de salud. En validate() comprueba que el usuario exista y que el email no esté duplicado en otro usuario, lanzando errores en caso contrario.

GetUserRequest

GetUserRequest valida la obtención de un usuario por su userId. Requiere que el ID no sea nulo, cumpla formato UUID y exista en la base de datos. En validate() comprueba que el usuario exista realmente y, de no ser así, agrega un error y corta la ejecución.

InsertImageUserRequest

InsertImageUserRequest maneja la subida de imagen de perfil de un usuario. Recibe userId (UUID válido que debe existir en la BD) y profileImg (archivo opcional, restringido a formatos de imagen comunes). Sobrescribe includeFiles() para aceptar ficheros en la petición. En

validate() hereda las validaciones básicas y comprueba que el usuario exista, de lo contrario lanza error y detiene la ejecución.

ListUsersRequest

ListUsersRequest extiende de FilterRequest y no añade ninguna propiedad ni método adicional. Esto significa que hereda toda la funcionalidad de filtrado, ordenamiento y paginación que FilterRequest proporciona para listar usuarios, sin personalizaciones específicas.

RemoveRoleUserRequest

RemoveRoleUserRequest valida la solicitud para eliminar un rol de un usuario, asegurando que userId y userRoleId existan, sean UUID válidos y estén en la base de datos, y que el usuario exista antes de continuar.

ResetPermissionsUserRequest

ResetPermissionsUserRequest valida que userId exista, sea un UUID válido y corresponda a un usuario en la base de datos antes de permitir restablecer sus permisos.

ToggleUserRequest

ToggleUserRequest valida que userId exista, sea un UUID válido y corresponda a un usuario registrado antes de permitir cambiar su estado activo/inactivo.

CreateUserHasDietRequest

CreateUserHasDietRequest extiende BaseRequest, asegura que los IDs del usuario y de la dieta existan en la base de datos usando ConstraintExistsInDatabase, y utiliza UserService y UserPermissionService para la validación y posibles permisos adicionales antes de crear la relación usuario-dieta.

DeleteUserHasDietRequest

DeleteUserHasDietRequest extiende BaseRequest y valida que el ID de la relación usuario-dieta (userHasDietId) exista en la base de datos usando ConstraintExistsInDatabase, sin lógica adicional aparte de la validación heredada de BaseRequest.

EditUserHasDietRequest

EditUserHasDietRequest extiende BaseRequest y valida que existan el ID de la relación usuario-dieta (userHasDietId), el usuario (userId) y la dieta (dietId) en la base de datos usando ConstraintExistsInDatabase, sin agregar lógica extra más allá de la validación heredada.

GetUserHasDietRequest

GetUserHasDietRequest extiende BaseRequest y valida que el userHasDietId exista en la base de datos usando ConstraintExistsInDatabase, sin agregar lógica adicional más allá de la validación heredada.

ListUserHasDietRequest

ListUserHasDietRequest extiende FilterRequest y se utiliza para listar registros de dietas asignadas a usuarios, sin agregar validaciones adicionales.

ToggleUserHasDietRequest

ToggleUserHasDietRequest valida que la dieta asignada al usuario exista en la base de datos mediante UserHasDietService antes de permitir alternar su estado activo/inactivo.

CreateUserHasRoutineRequest

CreateUserHasRoutineRequest valida que tanto el usuario como la rutina existan en la base de datos antes de asignar la rutina al usuario, usando los servicios UserService y UserPermissionService.

DeleteUserHasRoutineRequest

DeleteUserHasRoutineRequest valida que la relación específica entre usuario y rutina exista en la base de datos antes de permitir su eliminación, usando el servicio UserService.

EditUserHasRoutineRequest

EditUserHasRoutineRequest valida que la relación entre un usuario y una rutina exista, así como que el usuario y la rutina seleccionados también existan en la base de datos, usando UserService y UserPermissionService.

GetUserHasRoutineRequest

GetUserHasRoutineRequest valida que la relación entre un usuario y una rutina exista en la base de datos mediante su ID, usando UserService para asegurar que el registro solicitado es válido antes de procesarlo.

ListUserHasRoutineRequest

ListUserHasRoutineRequest es una clase que extiende FilterRequest y se utiliza para manejar solicitudes de listado de relaciones entre usuarios y rutinas, soportando filtrado y paginación de forma genérica.

CreateMentalStatsRequest

CreateMentalStatsRequest es una clase que extiende BaseRequest y se utiliza para validar y manejar solicitudes de creación de estadísticas mentales de un usuario, incluyendo userId, mood y sleepQuality.

CreatePhysicalStatsRequest

CreatePhysicalStatsRequest es una clase que extiende BaseRequest y se utiliza para validar y manejar solicitudes de creación de estadísticas físicas de un usuario, incluyendo userId, weight y height.

CreateReportRequest

CreateReportRequest es una clase que extiende BaseRequest y valida solicitudes para generar reportes de un usuario, asegurando que userId sea un string no vacío y que period sea uno de los valores permitidos: weekly, monthly o yearly.

GetCalorieIntakeRequest

GetCalorieIntakeRequest es una clase que extiende BaseRequest y valida solicitudes para obtener la ingesta calórica de un usuario, asegurando que userId sea un UUID válido existente en la base de datos y comprobando que el usuario realmente exista antes de procesar la petición.

GetMentalStatsRequest

GetMentalStatsRequest es una clase que extiende BaseRequest y valida solicitudes para obtener estadísticas mentales de un usuario, asegurando que userId sea un UUID válido existente en la base de datos y verificando que el usuario realmente exista antes de procesar la petición.

GetPhysicalStatsRequest

GetPhysicalStatsRequest es una clase que extiende BaseRequest y valida solicitudes para obtener estadísticas físicas de un usuario, asegurando que userId sea un UUID válido existente en la base de datos y confirmando que el usuario exista antes de procesar la petición.

3.8.3.3 Services

RoleRequestService

RoleRequestService gestiona las operaciones de rol a través de requests tipadas, interactuando con RoleService para crear, editar, eliminar, listar, activar/desactivar y modificar permisos de roles, validando su existencia y si son mutables, y devolviendo siempre respuestas estandarizadas en APIJsonResponse.

RoleService

RoleService maneja la lógica de negocio de los roles, interactuando con los repositorios RoleRepository y PermissionRepository para obtener, listar,

crear, editar, eliminar, activar/desactivar roles y gestionar sus permisos, incluyendo validaciones como excluir roles inmutables o superadministradores y soportando filtros y conversiones a arrays.

UserHasDietRequestService

UserHasDietRequestService gestiona las solicitudes relacionadas con las dietas asignadas a usuarios, permitiendo crear, editar, eliminar, listar, obtener por ID y activar/desactivar dietas. Utiliza UserService y DietService para validar entidades y UserHasDietService para ejecutar la lógica de negocio, devolviendo siempre respuestas estandarizadas mediante APIJsonResponse.

UserHasDietService

UserHasDietService gestiona la lógica de negocio relacionada con las dietas asignadas a los usuarios. Permite crear, editar, eliminar, listar y obtener relaciones usuario–dieta por ID o nombre, además de activar/desactivar dietas seleccionadas y desactivar todas las dietas de un usuario. Se comunica con el repositorio UserHasDietRepository para realizar todas las operaciones sobre la base de datos.

UserHasRoutineRequestService

UserHasRoutineRequestService gestiona las solicitudes relacionadas con las rutinas asignadas a usuarios, permitiendo crear, editar, eliminar, listar y obtener por ID las rutinas de un usuario. Utiliza UserService y

RoutineService para validar entidades y UserHasRoutineService para ejecutar la lógica de negocio, devolviendo siempre respuestas estandarizadas mediante APIJsonResponse.

UserHasRoutineService

UserHasRoutineService gestiona la lógica de negocio relacionada con la relación entre usuarios y rutinas, permitiendo crear, editar, eliminar, listar y obtener por ID las asignaciones de rutinas a usuarios. Utiliza UserHasRoutineRepository para interactuar con la base de datos y abstrae operaciones comunes como creación, edición, eliminación y consulta de entidades UserHasRoutine.

UserPermissionService

UserPermissionService gestiona la verificación de permisos del usuario actualmente autenticado, utilizando JWT para decodificar el token y obtener sus permisos. Proporciona la función can para determinar si el usuario tiene acceso a una acción específica dentro de un grupo de permisos, devolviendo un valor booleano.

UserRequestService

UserRequestService gestiona todas las solicitudes de usuarios, incluyendo creación, edición, eliminación, roles, permisos, imágenes, estadísticas y reportes. Asigna automáticamente dietas y rutinas según los objetivos del usuario y calcula estadísticas físicas como BMI y bodyFat. Utiliza

UserService, RoleService, DietService, RoutineService y MailService para la lógica de negocio y notificaciones. Devuelve siempre respuestas estandarizadas mediante APIJsonResponse con datos, estado y mensajes.

UserService

El UserService gestiona usuarios en la aplicación, permitiendo crear, editar y eliminar usuarios, administrar roles y permisos, actualizar contraseñas, imágenes y estados físicos y mentales. Proporciona métodos para obtener usuarios por ID, email o flags específicos, listar usuarios filtrados, y registrar estadísticas físicas y mentales. También genera reportes de actividad, ejercicio, calorías e ingestas para períodos semanales, mensuales o anuales, incluyendo cálculo de IMC y tendencias. Además, permite crear usuarios administradores, asignar o remover roles, resetear permisos y generar informes en PDF con gráficos y tablas resumidas de rutinas, ejercicios y alimentación. Se integra con repositorios y servicios de Doctrine, utilizando UserPasswordHasherInterface para manejo de contraseñas y Dompdf para generación de PDFs.

3.9 Public

3.9.1 Controller

PublicController

Gestiona solicitudes públicas, permitiendo enviar correos para recuperar contraseñas, restablecer contraseñas, subir archivos al servidor y registrar nuevos usuarios, delegando toda la lógica al servicio PublicRequestService y utilizando Requests personalizados para validar los datos.

3.9.2 Request

RecoverPasswordRequest

Esta clase RecoverPasswordRequest define y valida los datos necesarios para restablecer la contraseña de un usuario, asegurando que query_token, password y password_confirmation no estén vacíos, y que la nueva contraseña cumpla con los requisitos de seguridad (mínimo 8 caracteres, al menos un número, una mayúscula y una minúscula).

RegisterUserRequest

La clase RegisterUserRequest valida los datos necesarios para registrar un nuevo usuario, incluyendo información personal (nombre, email, sexo, fecha de nacimiento, peso, altura, objetivo de peso), contraseña y objetivos de salud o rehabilitación. También verifica que el email no esté ya registrado y aplica restricciones de formato para campos como el nombre.

SendEmailToRecoverPasswordRequest

La clase SendEmailToRecoverPasswordRequest valida que se proporcione un correo electrónico no vacío para poder enviar un email de recuperación de contraseña.

UploadFileToServerRequest

La clase UploadFileToServerRequest valida que se envíe un archivo y que exista una apiKey antes de permitir subirlo al servidor, extendiendo la funcionalidad de BaseRequest para manejar archivos.

3.9.3 Services

PublicRequestService

La clase PublicRequestService maneja las solicitudes públicas del sistema: envía correos para recuperar contraseñas, cambia contraseñas de usuarios, permite subir archivos al servidor con validación de API key, y registra nuevos usuarios calculando estadísticas físicas básicas y enviando credenciales por correo.

PublicService

La clase PublicService gestiona operaciones públicas de usuarios: envía correos para recuperación de contraseña generando un token temporal, cambia contraseñas usando ese token, y registra nuevos usuarios asignándoles rol y permisos adecuados.

3.9.4 Vistas

ForgotPassword

Este componente LoginForgotPassword en React renderiza un formulario para que el usuario solicite restablecer su contraseña. Usa Redux para obtener el estado de autenticación y errores, y LoginService para enviar la solicitud de correo de recuperación. Al enviar el formulario, si hay éxito, redirige al login y muestra un toast de confirmación; si falla, muestra un toast de error. Además, muestra un Alert si existe un error global en el estado de auth. Todo se encapsula dentro del contenedor LoginForgotPasswordContainer.

ForgotPasswordContainer

El componente LoginForgotPasswordContainer es un contenedor de presentación para la página de "Olvidé mi contraseña". Renderiza un Card centrado con un enlace para volver al login, un mensaje explicativo y cualquier contenido hijo (children), como el formulario de recuperación.

ForgotPasswordForm

El componente LoginForgotPasswordForm es un formulario controlado por Formik que permite al usuario introducir su correo electrónico para recuperar la contraseña. Valida que el campo no esté vacío y muestra mensajes de error si es necesario. Al enviarse, ejecuta la función submit recibida por props con el correo ingresado y deshabilita el botón mientras isLoading es true, mostrando un spinner durante el proceso.

Register

El componente Register muestra un formulario de registro de usuario dentro de RegisterContainer. Usa Redux para obtener el estado de autenticación y errores, y llama a LoginService.registerUser al enviar el formulario. Muestra notificaciones con toast según el éxito o fallo del registro y redirige al login si el registro es exitoso.

RegisterContainer

El componente RegisterContainer es un contenedor visual para la página de registro. Muestra un Card centrado con el título "Crear usuario", renderiza cualquier contenido hijo (como el formulario de registro), e incluye un enlace con botón para ir al login. Adapta estilos según el modo oscuro y usa PageWrapper y Page para la estructura de la página.

RegisterForm

Este código define un formulario de registro en React usando Formik y Yup para manejar validaciones, incluyendo nombre, email, contraseña, fecha de nacimiento, sexo, peso, altura y objetivos físicos, permite mostrar u ocultar contraseñas, controla que ciertos campos numéricos solo acepten números, gestiona opciones mutuamente excluyentes para objetivos de peso y envía los datos al hacer submit a una función submit proporcionada desde el componente padre.

ResetPassword

Este código define un componente de React para restablecer la contraseña, que muestra un formulario dentro de un contenedor, valida y envía la nueva contraseña junto con el token de usuario usando LoginService, maneja errores mostrando alertas o notificaciones con toast, y redirige al usuario al login tras un restablecimiento exitoso.

ResetPasswordContainer

Este componente define un contenedor de página para resetear la contraseña, mostrando un logo, un título y un botón de regreso al login, aplica estilos según el modo oscuro, y renderiza cualquier contenido hijo (como el formulario de restablecimiento) dentro de una tarjeta centrada en la página.

ResetPasswordForm

Este componente renderiza un formulario de restablecimiento de contraseña que valida los campos de nueva contraseña y confirmación usando Yup, permite alternar la visibilidad de ambas contraseñas con botones, extrae un token de la URL para enviar junto con la contraseña al backend al enviar el formulario, y muestra un spinner mientras se procesa la petición.

Login

Este componente maneja la pantalla de login, mostrando un formulario para introducir usuario y contraseña, valida las credenciales mediante LoginService, actualiza el estado de autenticación en Redux, muestra mensajes de error o éxito con toast, y redirige al usuario a la primera ruta disponible según sus permisos usando PrivilegeContext.

LoginContainer

Este componente define un contenedor para la pantalla de login, mostrando un Card centrado con el logo de la aplicación y renderizando cualquier contenido hijo pasado.

LoginForm

Este componente renderiza un formulario de login con validación usando Formik, permite mostrar u ocultar la contraseña, incluye enlaces para recuperar la contraseña o registrarse, y muestra un spinner mientras se procesa el envío.

3.10 Servicios API

Los archivos que se encargan de establecer la conexión del frontend con el backend son los servicios. Estos, tienen como base el siguiente archivo **restServiceConnection.**

restServiceConnection

RestServiceConnection facilita hacer peticiones HTTP con Axios, configurando automáticamente la URL base (pública o privada), los encabezados y la autenticación con token JWT, maneja errores de autorización 401 cerrando sesión si el token expira, y proporciona métodos para acceder a la respuesta completa o solo a los datos.

3.10.1 Auth

loginService

Este código define la clase LoginService que extiende RestServiceConnection y proporciona métodos para autenticar usuarios (authUserCredentials), enviar correos de recuperación de contraseña (sendEmailForgotPassword), resetear contraseñas usando un token (resetForgotPassword) y registrar nuevos usuarios (registerUser), encapsulando las peticiones HTTP y almacenando la respuesta para acceder a ella fácilmente.

permissionService

Este código define la clase PermissionService que extiende RestServiceConnection y proporciona un método getPermissions para obtener todos los permisos del sistema mediante una petición POST autenticada al endpoint /permissions/get-all, almacenando la respuesta para poder acceder a ella posteriormente.

roleService

Este código define la clase RoleService que extiende RestServiceConnection y proporciona métodos para gestionar roles: obtener todos los roles (getRoles o listRoles), obtener un rol por ID (getRoleById), crear (createRole), editar (editRole), eliminar (deleteRole) y activar/desactivar un rol (toggleRoleStatus) mediante peticiones POST autenticadas al backend, almacenando la respuesta para su posterior uso.

3.10.2 Dietas

dietService

`DietService` gestiona dietas mediante peticiones POST autenticadas, incluyendo crear (`createDiet`), editar (`editDiet`), listar (`getDiets`), obtener por ID (`getDietById`), eliminar (`deleteDiet`) y obtener para edición o con días (`getDietForEdit`, `getDietWithDays`). También permite asignar usuarios a una dieta (`assignUserToDiet`). Cada método almacena la respuesta para su uso posterior.

3.10.3 Documents

documentService

`DocumentService` gestiona documentos mediante peticiones POST autenticadas, permitiendo obtener (`getDocument`), renderizar o descargar como blob (`renderDocument`, `downloadDocument`), eliminar uno o varios

documentos (`deleteDocument`, `deleteDocuments`) y generar la URL de renderizado directo (`renderDocumentURL`). Cada método guarda la respuesta para su uso posterior.

documentTypeService

`DocumentTypeService` gestiona tipos de documentos mediante peticiones autenticadas, permitiendo crear (`createDocumentType`), listar con filtros (`getDocumentTypes`), obtener por ID (`getDocumentTypeById`), eliminar (`deleteDocumentType`) y obtener tipos de entidad (`getEntityTypes`). Cada método guarda la respuesta para uso posterior.

3.10.4 EducativeResources

educativeResourcesService

`EducativeResourceService` gestiona recursos educativos mediante peticiones autenticadas, permitiendo crear (`createEducativeResource`), editar (`editEducativeResource`), listar con filtros (`getEducativeResource`), obtener por ID (`getEducativeResourceById`) y eliminar (`deleteEducativeResource`). Cada método almacena la respuesta para uso posterior.

3.10.5 Exercises

exerciseService

`ExerciseService` gestiona ejercicios y sus categorías mediante peticiones autenticadas. Permite crear, editar, listar (`getExercises`), obtener por ID (`getExerciseById`) y eliminar (`deleteExercise`) ejercicios, así como manejar categorías con métodos para listar, obtener, crear, editar y eliminar. Todas las respuestas se almacenan para uso posterior.

3.10.6 Foods

foodService

`FoodService` gestiona alimentos mediante peticiones autenticadas. Permite crear (`createFood`), editar (`editFood`), listar (`getFood`), obtener por ID (`getFoodById`) y eliminar (`deleteFood`) alimentos, almacenando las respuestas para uso posterior.

3.10.7 Reports

reportsService

`ReportsService` permite generar reportes de usuario autenticados según un periodo específico (`weekly`, `monthly` o `yearly`) mediante la función `generateUserReport`.

3.10.8 Routine-register

routineRegisterService

`RoutineRegisterService` gestiona registros de rutinas, permitiendo crear, editar, listar, eliminar y finalizar registros, así como obtener registros activos de un usuario específico mediante llamadas autenticadas al backend.

3.10.9 Routine-register-exercises

routineRegisterExercisesService

`RoutineRegisterExercisesService` gestiona los ejercicios asociados a registros de rutinas, permitiendo crear, editar, listar, obtener por ID y eliminar ejercicios mediante llamadas autenticadas al backend

3.10.10 Routines

routineService

`RoutineService` gestiona rutinas y sus categorías, permitiendo crear, editar, listar, eliminar, exportar, asignar usuarios y obtener detalles con días o ejercicios, todo mediante llamadas autenticadas al backend.

3.10.11 User-has-diet

userHasDietService

`UserHasDietService` gestiona la relación de usuarios con dietas, permitiendo crear, editar, listar, eliminar, activar/desactivar dietas asignadas y manejar la ingesta diaria mediante llamadas autenticadas al backend.

3.10.12 User-has-routine

userHasRoutineService

`UserHasRoutineService` gestiona la asignación de rutinas a usuarios, permitiendo crear, editar, listar y eliminar rutinas asignadas mediante llamadas autenticadas al backend.

3.10.13 Users

userService

`UserService` gestiona usuarios, permitiendo crear, editar, eliminar y listar usuarios, manejar roles, permisos, documentos e imágenes, actualizar contraseñas y estadísticas físicas/mentales, así como obtener información del perfil propio mediante llamadas autenticadas al backend.

4. Anexo

El código fuente completo del sistema desarrollado, está disponible en el siguiente repositorio: https://github.com/mchacon2001/TFG_Miguel-Angel-Chacon-Perez.