Proyecto - ResponsiveMeals

David Tejera González
Cristian Marrero Marrero

Índice

Introducción

La aplicación de ResponsiveMeals está diseñada para realizar pedidos de envíos de comida a domicilio de forma planificada, de tal manera que desde nuestra aplicación puedas planificar el menú de cada semana y se te enviarán los platos correspondientes a esa semana al principio de cada semana.

La idea nace de la necesidad de tener un servicio de comidas fiable para cuando el tiempo escasea y los usuarios no disponen de tiempo suficiente para preparar sus comidas, además que cuente con un servicio de suscripción que facilita métodos de pago mensuales y cubre gastos de envíos para cubrir las necesidades de la población canaria dado el aislamiento y los elevados costes de envío de otros territorios a las islas.

En la aplicación aparte de la propia funcionalidad se encontrarán manuales de ayuda, informes con datos relevantes de la aplicación, información de contacto y gestión de dudas; además de contar con una interfaz amigable y el uso de tonos naranjas que incitan a pensar en la comida.

Modelo de Datos

Diagrama E/R

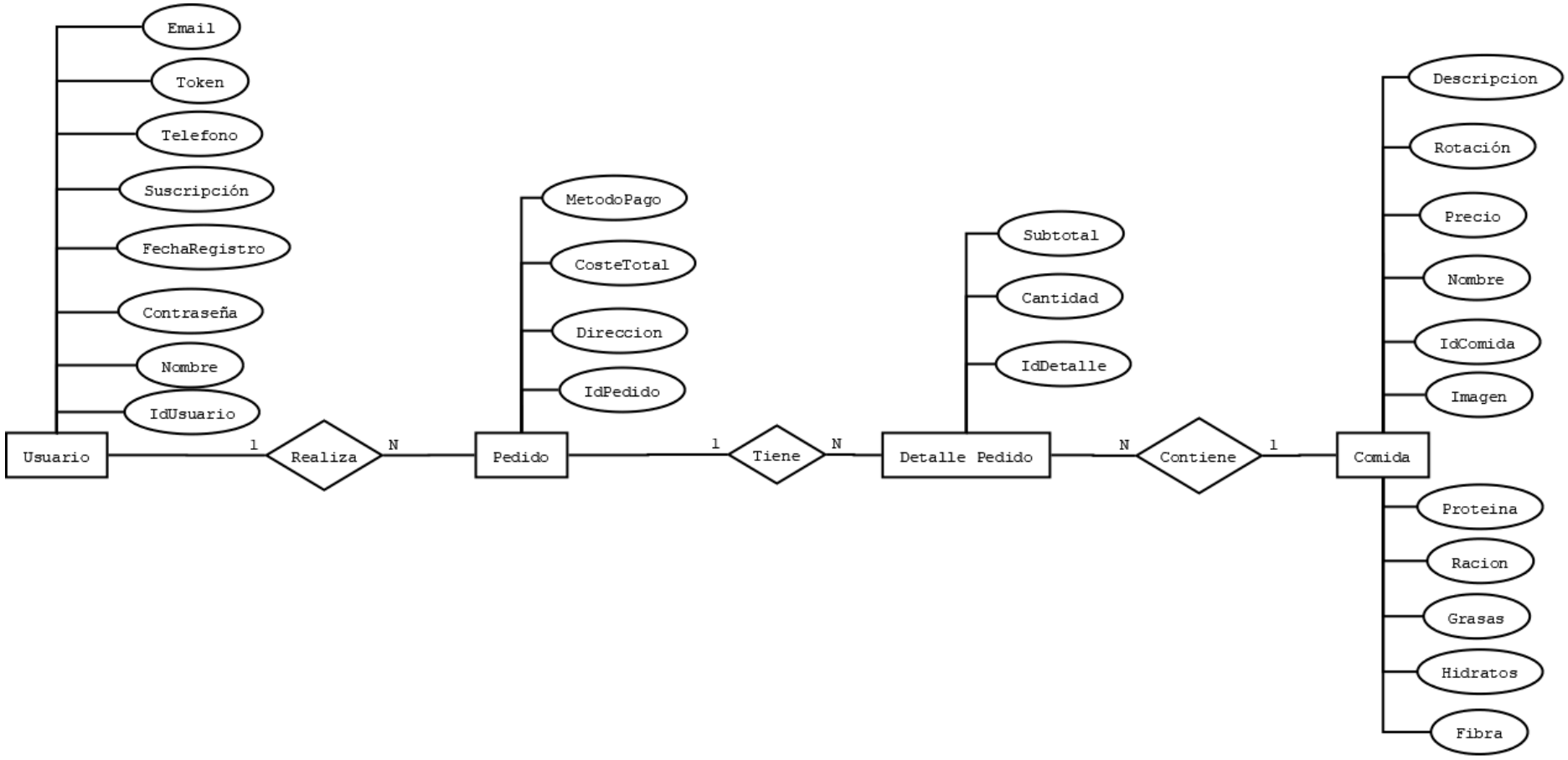


Diagrama UML

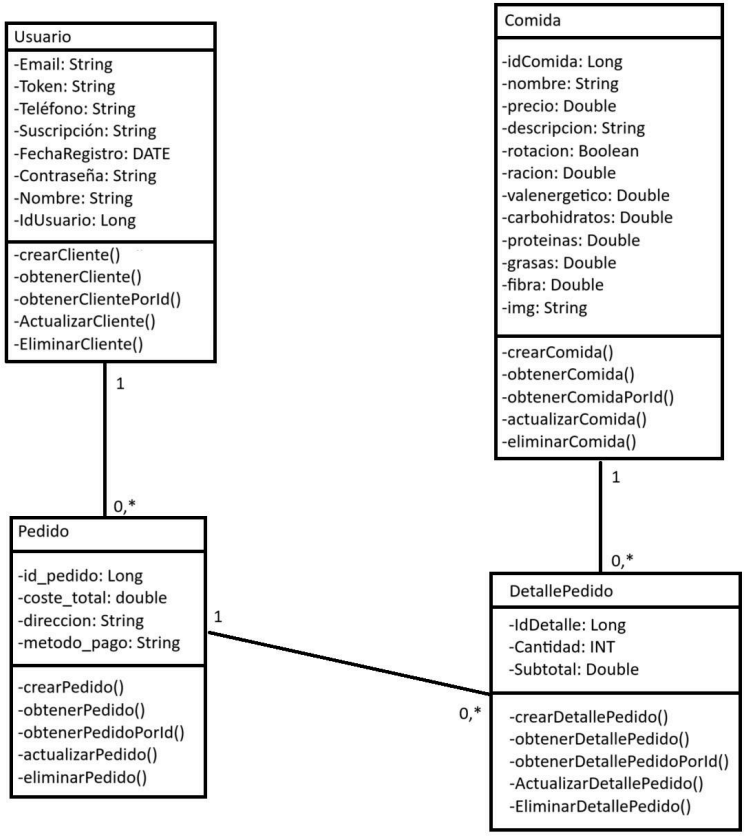
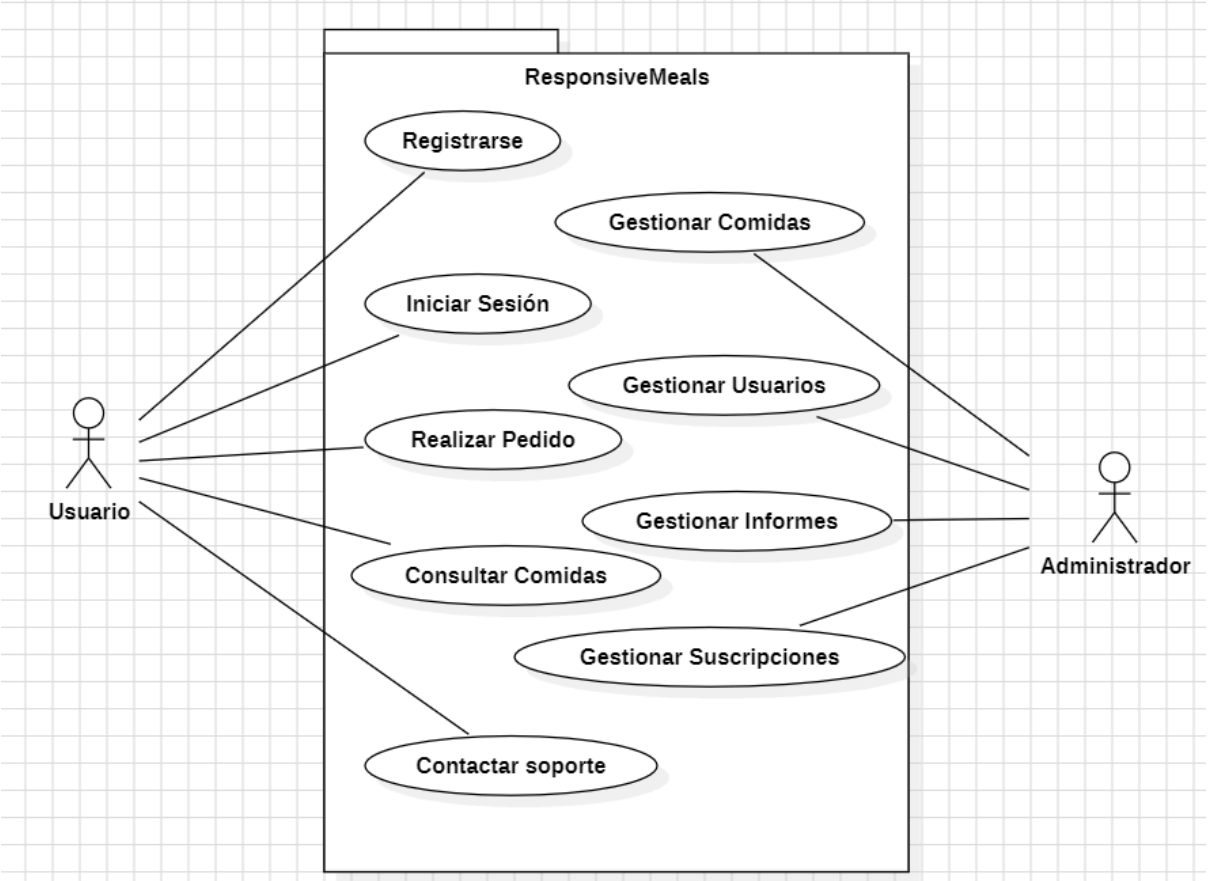


Diagrama Casos de Uso



Explicación Relaciones y Campos

- Usuario

Un usuario posee una relación de uno a muchos con la entidad Pedido porque un cliente puede realizar varios pedidos pero un único pedido solo ha podido ser realizado por un usuario en concreto.

- **IdUsuario (int):** Cada Usuario tiene un Id único que le identifica en la base de datos.
- **Nombre(string):** Nombre con el que se registra cada usuario.
- **Contraseña(string):** Cadena de caracteres que guarda la contraseña de cada usuario.
- **FechaRegistro(timestamp):** Fecha y hora con la que se registra cada usuario.
- **Suscripción(string):** Tipo de suscripción que cada usuario tiene en la aplicación. Puede ser estándar, premium, o no tener ninguna.
- **Teléfono(string):** Campo que almacena el teléfono con el que se registra cada usuario.
- **Email(string):** Campo que almacena el email con el que se registra cada usuario.
- **Token(string):** Campo que almacena el token generado al loguearse cada usuario que sirve para identificación dentro de la aplicación.

- Pedido

Un pedido está en la parte de muchos a uno de la relación con Usuarios, y a su vez tiene una relación de uno a muchos con Detalle Pedido ya que diversos Detalles de pedido componen un pedido.

- **IdPedido(int):** Id único a cada pedido que lo identificará en la base de datos.
- **Dirección(string):** Dirección a la que se envía un pedido.
- **Coste total(double):** Coste total del pedido sumando sus detalles.
- **Método de pago(string):** Método de pago elegido por el usuario para pagar el pedido.

- Detalle Pedido

Los detalles de pedido están en el lado de muchos a uno tanto con pedidos como con comidas ya que un detalle está presente varias veces en un pedido, y una comida puede estar presente en diversos detalles de pedido.

- **IdDetalle(int):** Id único a cada detalle que lo identifica en la base de datos.
- **Cantidad(int):** Cantidad de platos de una comida en concreto que componen el detalle.
- **Subtotal(double):** Precio total del detalle de pedido calculado con la cantidad y el precio de cada comida.

- Comida

Cada comida tiene relación de uno a muchos con detalle pedido ya que una comida puede estar presente en muchos detalles distintos.

- **IdComida(int):** Id con el que se identificará a cada comida en la base de datos.
- **Nombre(string):** Nombre identificativo de cada comida.
- **Precio(double):** Precio asignado a cada comida.
- **Rotación(boolean):** Campo booleano que indica si una comida está presente en la rotación mensual de platos incluidos en la suscripción estándar.
- **Descripción(string):** Descripción de cada comida indicando sus ingredientes principales, método de preparación, etc.
- **Campos de información nutricional(Proteínas,Ración,Grasas,HIdratos, Fibra)(double):** Todo son campos relacionados con la información nutricional de cada plato de comida.

Modelo Relacional

- Usuario: IdUsuario, Nombre, Contraseña, FechaRegistro, Suscripción, Teléfono, Email, Token.
- **Pedido:** IdPedido, Dirección, CosteTotal, MétodoPago, IdUsuario*.
- **Detalle Pedido:** IdDetalle, Cantidad, Subtotal, IdComida*, IdPedido*.
- **Comida:** IdComida, Imagen, Nombre, Precio, Rotación, Descripción, Proteínas, Ración, Grasas, Hidratos, Fibra.

Arquitectura

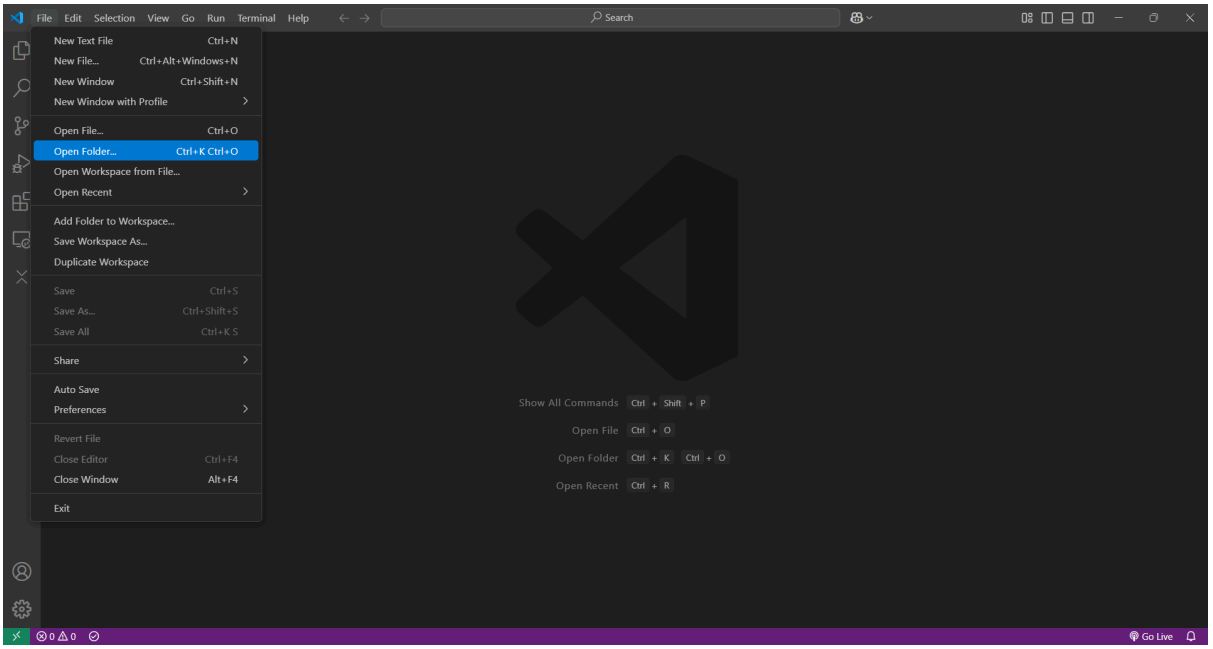
Para la realización de esta aplicación hemos usado diversos lenguajes, frameworks e IDEs, los cuales serán nombrados a continuación:

- **Java con Springboot**: Para la realización de la API y toda la parte de back hemos usado Java con el framework de Spingboot el cual incluye muchas anotaciones y funciones que facilitan la codificación del proyecto.
- **React con Typescript**: Toda la parte de front y la interfaz de usuario han sido desarrolladas en React con el lenguaje de TypeScript que permite crear componentes y usar librerías externas para facilitar el trabajo y cumplir con buenos criterios de usabilidad y accesibilidad en la aplicación.
- **Visual Studio Code**: Principal IDE donde hemos desarrollado todo el código del proyecto. No consta nativamente como un IDE pero es un editor de texto avanzado al cual se le pueden añadir extensiones para que interprete los diferentes lenguajes que usamos.
- **Postman**: Herramienta útil para comprobar la funcionalidad de la API ya que permite acceder a los diferentes endpoints y activando los respectivos métodos HTTP para hacer comprobaciones básicas y simples de validaciones, inserciones, eliminaciones, modificaciones, etc.

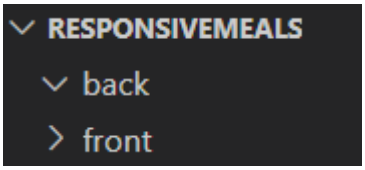
Manual de instalación

Para desarrolladores

Para poder instalar esta aplicación usaremos Visual Studio Code. Puedes instalar esta aplicación desde este enlace. Una vez instalado, entramos, le damos a file y luego a Open Folder (o pulsamos ctrl+k seguido de ctrl+o).



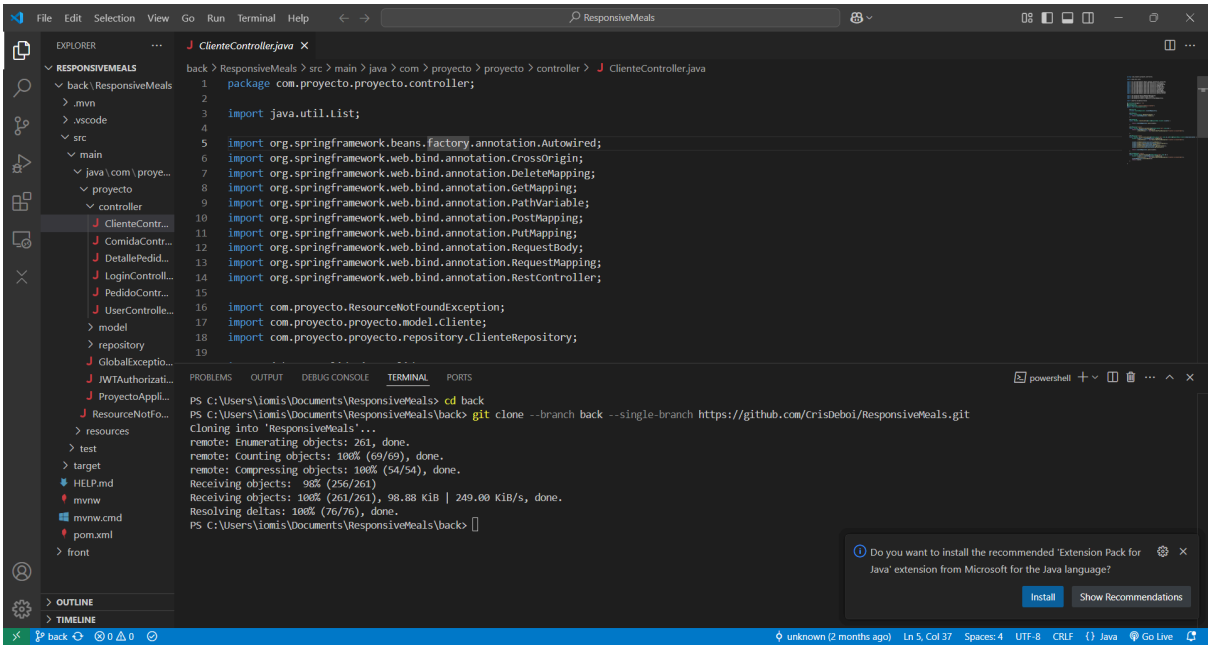
Creamos una nueva carpeta y la seleccionamos. Dentro de esa carpeta creamos dos nuevas, back y front. Para crear una carpeta hay que pasar el ratón por encima y darle al segundo icono, el que pone New Folder.



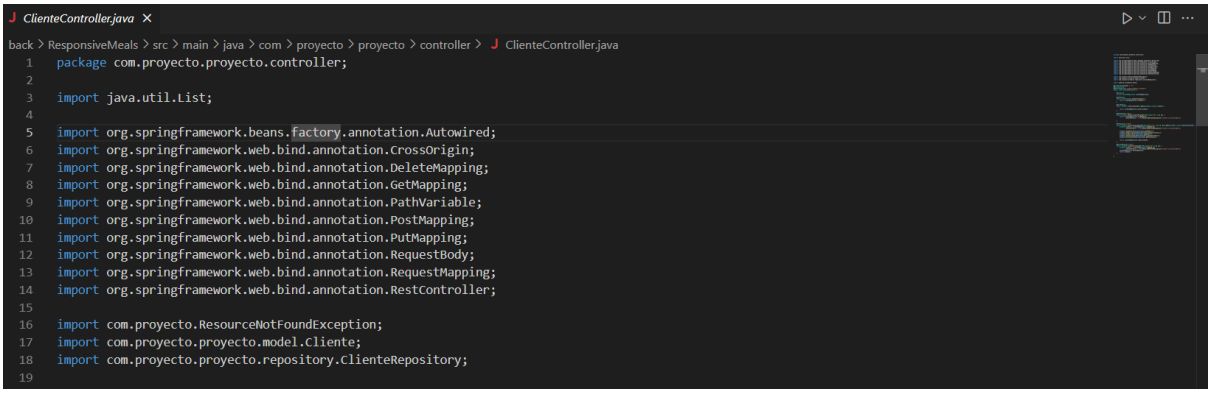
Una vez creadas, pulsamos Ctrl+ñ para abrir una terminal. En la terminal ponemos “cd back”, que nos posicionará en la carpeta “back” que acabamos de crear. El segundo comando, “git clone --branch back --single-branch https://github.com/CrisDeboi/ResponsiveMeals.git”, nos copia la aplicación en la carpeta back.

```
PS C:\Users\iomis\Documents\ResponsiveMeals> cd back
PS C:\Users\iomis\Documents\ResponsiveMeals\back> git clone --branch back --single-branch https://github.com/CrisDeboi/ResponsiveMeals.git
```

Al instalar la aplicación, si entras a cualquier archivo java saldrá la ventana de abajo a la derecha. Dale a instalar.



Tras instalar la extensión de java, podrás ver que salió un botón con forma de triángulo arriba.



Pulsalo desde cualquier página de java de la aplicación y se encenderá la API. Debería salir este mensaje en la consola.

```
2025-02-18T20:30:22.458Z  INFO 21236 --- [responsivemeals] [           main] r$InitializeUserDetailsManagerConfigurer : Glo
bal AuthenticationManager configured with UserDetailsService bean with name inMemoryUserDetailsManager
2025-02-18T20:30:23.665Z  INFO 21236 --- [responsivemeals] [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exp
osing 1 endpoint beneath base path '/actuator'
2025-02-18T20:30:24.683Z  INFO 21236 --- [responsivemeals] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tom
cat started on port 8080 (http) with context path '/'
2025-02-18T20:30:24.735Z  INFO 21236 --- [responsivemeals] [           main] c.proyecto.proyecto.ProyectoApplication  : Sta
rted ProyectoApplication in 15.804 seconds (process running for 16.63)
Funciona socio
```

Sin cerrar la terminal, abrimos otra y ponemos los comandos “cd front” y “git clone --branch front --single-branch https://github.com/CrisDeboi/ResponsiveMeals.git”.

```
PS C:\Users\iomis\Documents\DAD\ResponsiveMeals> cd front
PS C:\Users\iomis\Documents\DAD\ResponsiveMeals\front> git clone --branch front --single-branch https://github.com/CrisDebo
i/ResponsiveMeals.git
```

Cuando se termine de clonar el front de la aplicación pon los comandos “cd responsivemeals” y “npm install”

```
PS C:\Users\iomis\Documents\DAD\ResponsiveMeals\front> cd responsivemeals
PS C:\Users\iomis\Documents\DAD\ResponsiveMeals\front\responsivemeals> npm install
```

Cuando se termine de instalar, finalmente pon el comando “npm run dev”

```
PS C:\Users\iomis\Documents\DAD\ResponsiveMeals\front\responsivemeals> npm run dev
```

Ve a aqui y finalmente habrás iniciado la aplicación.

Documentación de la API

Cliente

El controlador de cliente lo hemos definido con el endpoint de /responsivemeals/clientes, donde tiene sus diferentes método de get, get by id, post, put y delete:

```
@GetMapping
public List<Cliente> obtenerClientes() {
    return clienteRepository.findAll();
}

@PostMapping
public Cliente crearCliente(@Valid@RequestBody Cliente cliente) {

    return clienteRepository.save(cliente);
}

@GetMapping("/{id}")
public Cliente obtenerClientePorId(@PathVariable("id") Long id) {
    return clienteRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Cliente no encontrada"));

}

@PutMapping("/{id}")
public Cliente actualizarCliente(@PathVariable("id") Long id, @Valid@RequestBody Cliente detallesCliente) {
    Cliente cliente = clienteRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"cliente no encontrada"));

    cliente.setNombre(detallesCliente.getNombre());
    cliente.setSuscripcion(detallesCliente.getSuscripcion());
    cliente.setEmail(detallesCliente.getEmail());
    cliente.setContrasena(detallesCliente.getContrasena());
    cliente.setTelefono(detallesCliente.getTelefono());
```

```
    return clienteRepository.save(cliente);
}

@DeleteMapping("/{id}")
public Cliente eliminarCliente(@PathVariable("id") Long id) {
    Cliente cliente = clienteRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Cliente no encontrado"));
    clienteRepository.deleteById(id);
    return cliente;
}
```

Pedido

El endpoint definido para la entidad proyecto ha sido el de /responsivemeals/pedidos, al cual en el controlador hemos asignado los siguientes métodos:

```java
@GetMapping
public List<Pedido> obtenerPedidos() {
    return pedidoRepository.findAll();
}

@PostMapping
public Pedido crearPedido(@Valid@RequestBody Pedido pedido) {
    System.out.println("Datos recibidos: " + pedido);
    // Validar si el cliente viene en el pedido
    if (pedido.getCliente() == null || pedido.getCliente().getIdCliente() == null) {
        throw new IllegalArgumentException(s:"El cliente es requerido para crear un pedido");
    }

    // Verificar si el cliente existe
    Cliente cliente = clienteRepository.findById(pedido.getCliente().getIdCliente())
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Cliente no encontrado"));

    // Asociar el cliente al pedido
    pedido.setCliente(cliente);

    // Guardar el pedido
    return pedidoRepository.save(pedido);
}

@GetMapping("/{id}")
public Pedido obtenerPedidoPorId(@PathVariable("id") Long id) {
    return pedidoRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Pedido no encontrado"));
```

```java
@GetMapping("/cliente/{id}")
public List<Pedido> obtenerPedidosPorCliente(@PathVariable("id") Long id) {
    // Buscar cliente para validar existencia (opcional)
    Cliente cliente = clienteRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Cliente no encontrado"));

    // Obtener pedidos del cliente
    return pedidoRepository.findByCliente(cliente);
}

@PutMapping("/{id}")
public Pedido actualizarPedido(@PathVariable("id") Long id, @Valid@RequestBody Pedido detallesPedido) {
    Pedido pedido = pedidoRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Pedido no encontrado"));

    pedido.setCoste_total(detallesPedido.getCoste_total());
    pedido.setDireccion(detallesPedido.getDireccion());
    pedido.setMetodo_pago(detallesPedido.getMetodo_pago());

    return pedidoRepository.save(pedido);
}

@DeleteMapping("/{id}")
public Pedido eliminarPedido(@PathVariable("id") Long id) {
    Pedido pedido = pedidoRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Pedido no encontrado"));
    pedidoRepository.deleteById(id);
    return pedido;
}
```

Detalle Pedido

Para la entidad Detalle Pedido hemos creado el endpoint /responsivemeals/detallepedido, en el cual hemos configurado los siguientes métodos:

```java
@GetMapping
public List<DetallePedido> obtenerDetallePedidos() {
    List<DetallePedido> detalles = detallePedidoRepository.findAll();
    System.out.println("DetallePedidos recuperados: " + detalles); // Depuración
    return detalles;
}

@PostMapping
public ResponseEntity<DetallePedido> crearDetallePedido(@RequestBody DetallePedidoDTO detalleDTO) {
    // Buscar las entidades relacionadas
    Pedido pedido = pedidoRepository.findById(detalleDTO.getIdPedido())
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Pedido no encontrado"));
    Comida comida = comidaRepository.findById(detalleDTO.getIdComida())
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Comida no encontrada"));

    // Crear la entidad DetallePedido
    DetallePedido detallePedido = new DetallePedido();
    detallePedido.setPedido(pedido);
    detallePedido.setComida(comida);
    detallePedido.setCantidad(detalleDTO.getCantidad());
    detallePedido.setSubtotal(detalleDTO.getCantidad() * comida.getPrecio());

    // Actualizar coste total en el pedido
    pedido.setCoste_total(pedido.getCoste_total() + detallePedido.getSubtotal());

    // Guardar el nuevo detallePedido en la base de datos
    detallePedidoRepository.save(detallePedido);

    return ResponseEntity.ok(detallePedido);
}
```

```java
@PutMapping("/{id}")
public ResponseEntity<DetallePedido> actualizarDetallePedido(
        @PathVariable("id") Long id,
        @RequestBody DetallePedidoDTO detalleDTO) {
    DetallePedido detallePedido = detallePedidoRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Detalle no encontrado"));
    Comida comida = comidaRepository.findById(detalleDTO.getIdComida())
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Comida no encontrada"));
    Pedido pedido = pedidoRepository.findById(detalleDTO.getIdPedido())
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Pedido no encontrado"));
    detallePedido.setComida(comida);
    detallePedido.setPedido(pedido);
    detallePedido.setCantidad(detalleDTO.getCantidad());
    detallePedido.setSubtotal(detalleDTO.getCantidad() * comida.getPrecio());
    DetallePedido detalleActualizado = detallePedidoRepository.save(detallePedido);
    return ResponseEntity.ok(detalleActualizado);
}

@DeleteMapping("/{id}")
public DetallePedido eliminarDetallePedido(@PathVariable("id") Long id) {
    DetallePedido detallePedido = detallePedidoRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Detalle no encontrado"));
    detallePedidoRepository.deleteById(id);
    return detallePedido;
}
```

Comida

La entidad de Comida la hemos definido en el endpoint de /responsivemeals/comida en el que hemos definido los siguientes métodos:

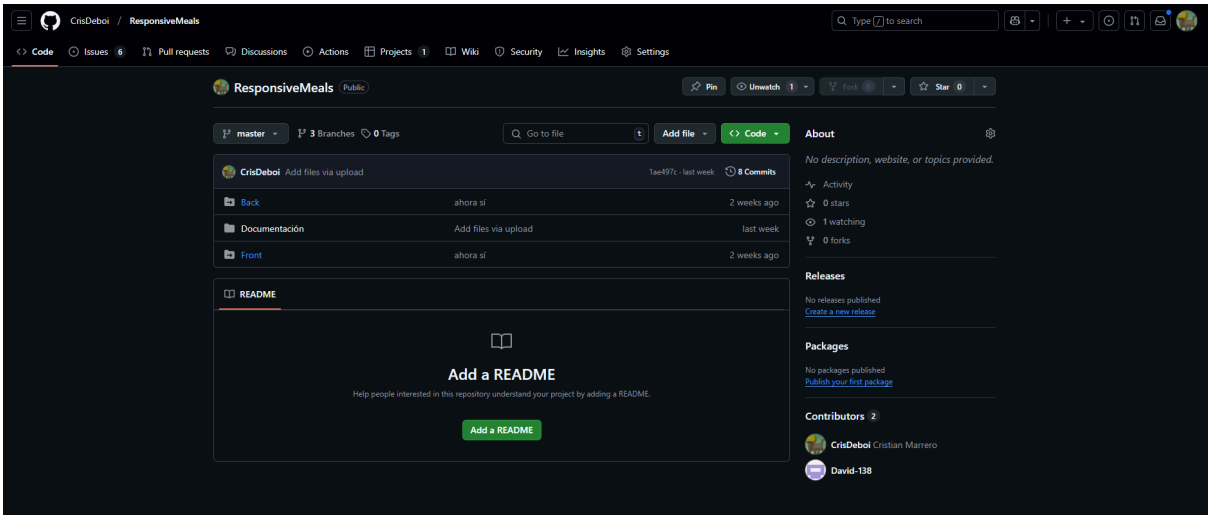
```
@GetMapping
public List<Comida> obtenerComidas() {
    return comidaRepository.findAll();
}

@PostMapping
public Comida crearComida(@Valid@RequestBody Comida comida) {
    comida.setImg(comida.getNombre()+".jpg");
    return comidaRepository.save(comida);
}

@GetMapping("/{id}")
public Comida obtenerComidaPorId(@PathVariable("id") Long id) {
    return comidaRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Comida no encontrada"));

}

@PutMapping("/{id}")
public Comida actualizarComida(@PathVariable ("id") Long id, @Valid@RequestBody Comida detallesComida) {
    Comida comida = comidaRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Comida no encontrada"));

    comida.setNombre(detallesComida.getNombre());
    comida.setPrecio(detallesComida.getPrecio());
    comida.setRotacion(detallesComida.getRotacion());
    comida.setDescripcion(detallesComida.getDescripcion());
    comida.setCarbohidratos(detallesComida.getCarbohidratos());
    comida.setFibra(detallesComida.getFibra());
```

```
    comida.setFibra(detallesComida.getFibra());
    comida.setGrasas(detallesComida.getGrasas());
    comida.setRacion(detallesComida.getRacion());
    comida.setImg(detallesComida.getImg());

    return comidaRepository.save(comida);
}

@DeleteMapping("/{id}")
public Comida eliminarComida(@PathVariable("id") Long id){
    Comida comida = comidaRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException(mensaje:"Comida no encontrada"));
    comidaRepository.deleteById(id);
    return comida;
}
```

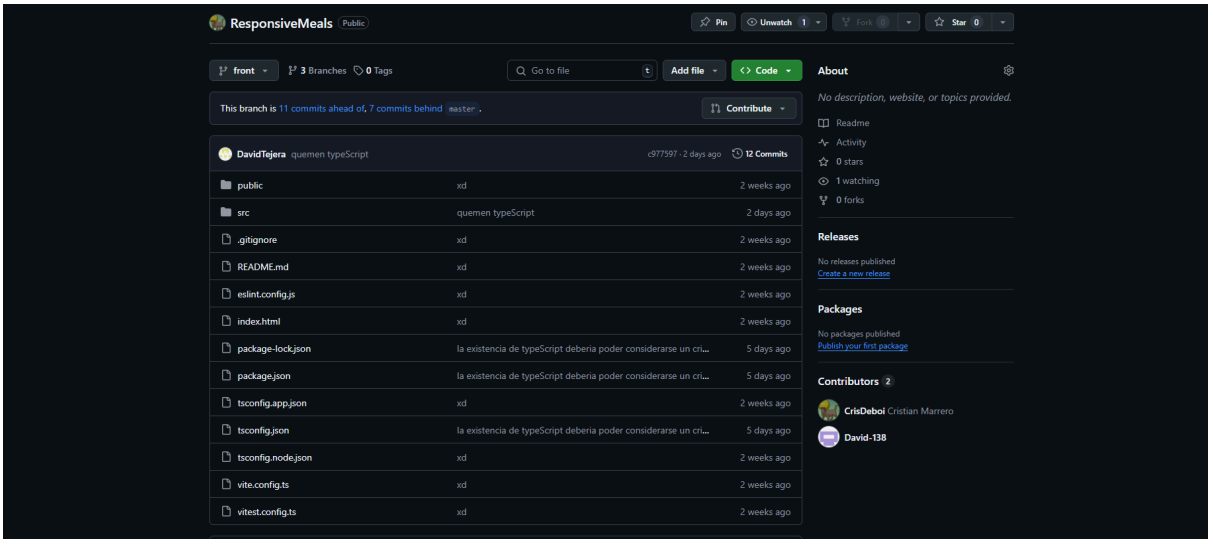
Metodología y Control de versiones

Control de versiones

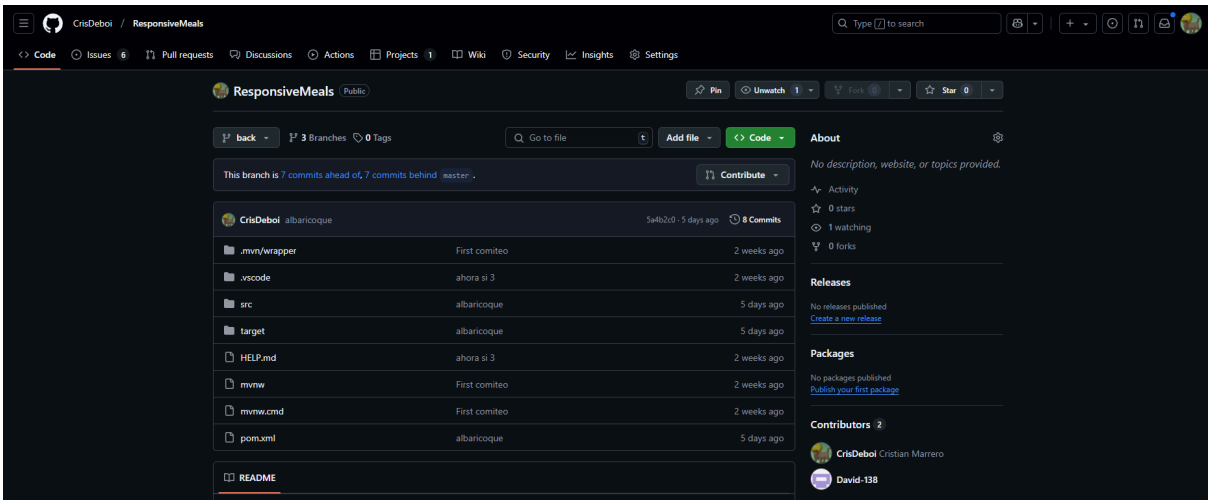
Para el control de versiones, en este proyecto hemos hecho uso de GitHub, creando un repositorio en común en el que ambos figuramos como colaboradores. Para gestionar bien el avance que ha hecho cada uno, hemos creado una rama dentro del repositorio para cada uno de los aspectos a trabajar en el proyecto, dejando la rama master para el producto final.



De las ramas que hemos creado, una de ellas es la rama de Front, donde volcamos todos los cambios relativos al Frontend de nuestra aplicación, que incluye todo lo trabajado en React con TypeScript:



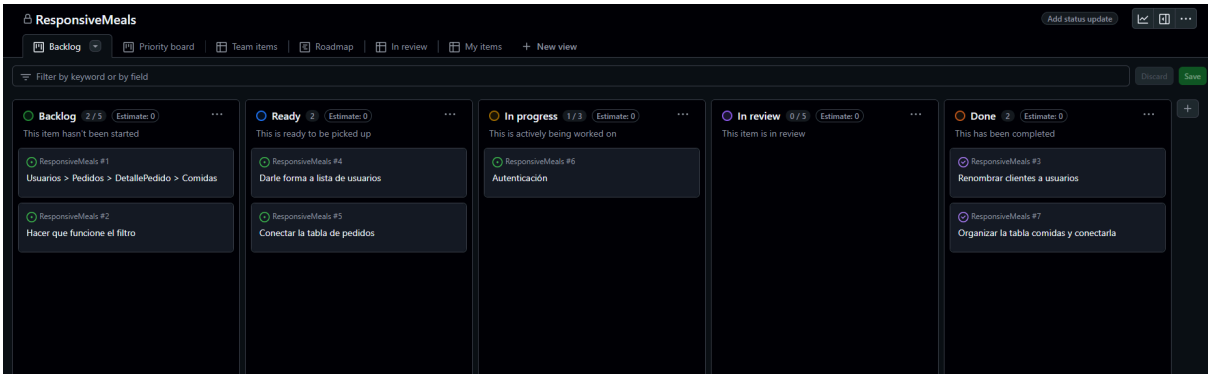
Por otro lado, hemos creado la rama de Back, que recoge todos los cambios en el apartado de backend que incluyen los trabajos en la API y usando Java con SpringBoot:



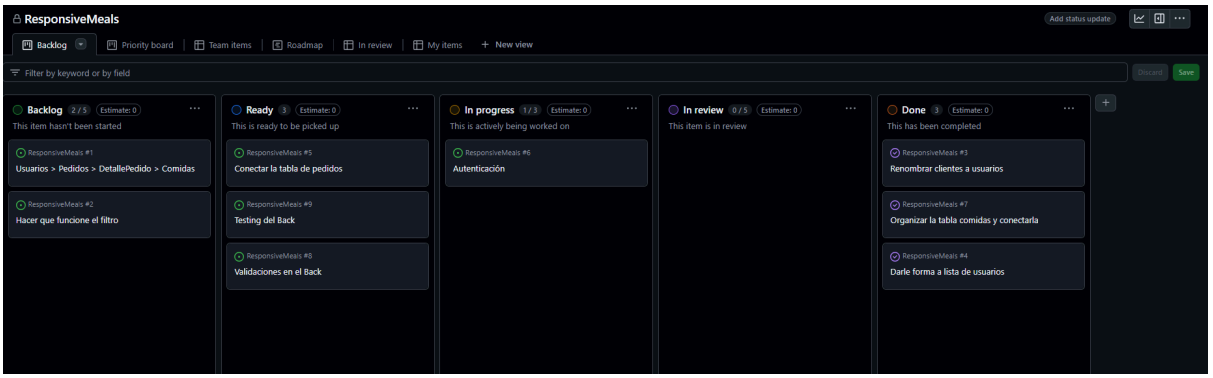
Metodología

En cuanto a la metodología de trabajo, para organizar nuestras tareas en equipo hemos optado por usar el gestor de tareas de proyectos de Git, en el cual hemos ido añadiendo las tareas y mejoras a realizar donde cada uno elige una con la que ponerse a trabajar y las va clasificando en hechas, pendientes o en progreso según se dé la situación.

- Para el comienzo del proyecto, en la semana del **27 al 31 de enero,** nuestro tablón de tareas estaba así al día 31:



- Al término de la semana del **3 al 7 de febrero**, el tablón tenía este estado, con nuevas tareas en el backlog y otras terminadas:



Enlaces

- Enlace al repositorio de GitHub:
https://github.com/CrisDeboi/ResponsiveMeals/tree/back

- Enlace al proyecto con las tareas de GitHub:
https://github.com/users/CrisDeboi/projects/3