

MÓDULO PROYECTO

Ciclo Superior Desarrollo de Aplicaciones Web

Departamento: Informática y Comunicaciones

IES "María Moliner"

Curso: 2023/2024

Grupo: S2I



Proyecto: Comida En Marcha

Fernando Juan Estrada Gallardo

Email: fern.j.e.g@gmail.com

Tutor individual: María José González García

Tutor colectivo: Enrique Carballo Albarrán

Fecha de presentación: 27/05/2024

Contenido

DESCRIPCIÓN GENERAL DEL PROYECTO	6
Idea	6
Objetivos	7
Cuestiones metodológicas	8
Entorno de trabajo	8
Extensiones	11
DESCRIPCIÓN GENERAL DEL PRODUCTO	12
Límites del sistema	12
Funcionalidades básicas	13
Usuarios	14
Hosting	15
PLANIFICACIÓN Y PRESUPUESTO	18
Planificación	18
Metodología	19
Bocetos	20
Interfaces finales	22
Admin	22
Página de error	22
Pantalla de Login	23
Navegación	25
Editor de comidas	26
Menú edición	27
Editor del restaurante	28
Pantalla de tickets	29
Cliente	30
Página para iniciar	30
Interfaz principal	31
Historial de pedido	32
Otros diseños	33
Icono	35
Presupuesto	36

Modelo de negocio	37
DOCUMENTACIÓN TÉCNICA	38
Base de datos	38
Entidades	38
User	39
Restaurante	40
Ticket	41
Rutas API	42
### Main Router	42
### Auth Router	42
### Restaurante Router	42
### User Router	43
### Ticket Router	43
### Plato Router	44
### Mesas Router	44
Estructura del código	45
Librerías	45
Códigos de interés	46
Subida de imágenes	46
Errores customizados para respuestas	48
Enrutamiento con React Router	49
Tokens para autenticación	50
Punto de entrada React.	51
Previsualización de imágenes	52
Pruebas	53
MANUALES DE USUARIO	54
Instalación en local	54
Instalación en la nube (versión recomendada)	56
General	56
Base de datos	57
Frontends	59
Servidor	62
Servidor	64
Cargar datos de ejemplo	67
Uso	68
CONCLUSIONES, CORRECCIONES Y AMPLIACIONES	69

Conclusiones	69
Ampliaciones	70
Correcciones	71
Relación de ficheros en formato digital	72
General	72
>> Backend	73
>> Frontends	75
BIBLIOGRAFÍA	77
GLOSARIO	78

Descripción general del proyecto

Idea

La idea detrás de la aplicación fue un directo en twitch dónde un español pedía comida en un restaurante en Japón.



Objetivos

Se pretende la realización de una aplicación web que sirva para varios establecimientos y consiga reemplazar a parte del capital humano que se dedica a tomar nota. Para visualizarlo mejor un ejemplo parecido son las típicas pantallas de sitios que sirven comida rápida que hay en todas las ciudades.

Como objetivos secundarios se pretende permitir ver todos los pedidos, configurar el sitio con datos del restaurante y otras personalizaciones.

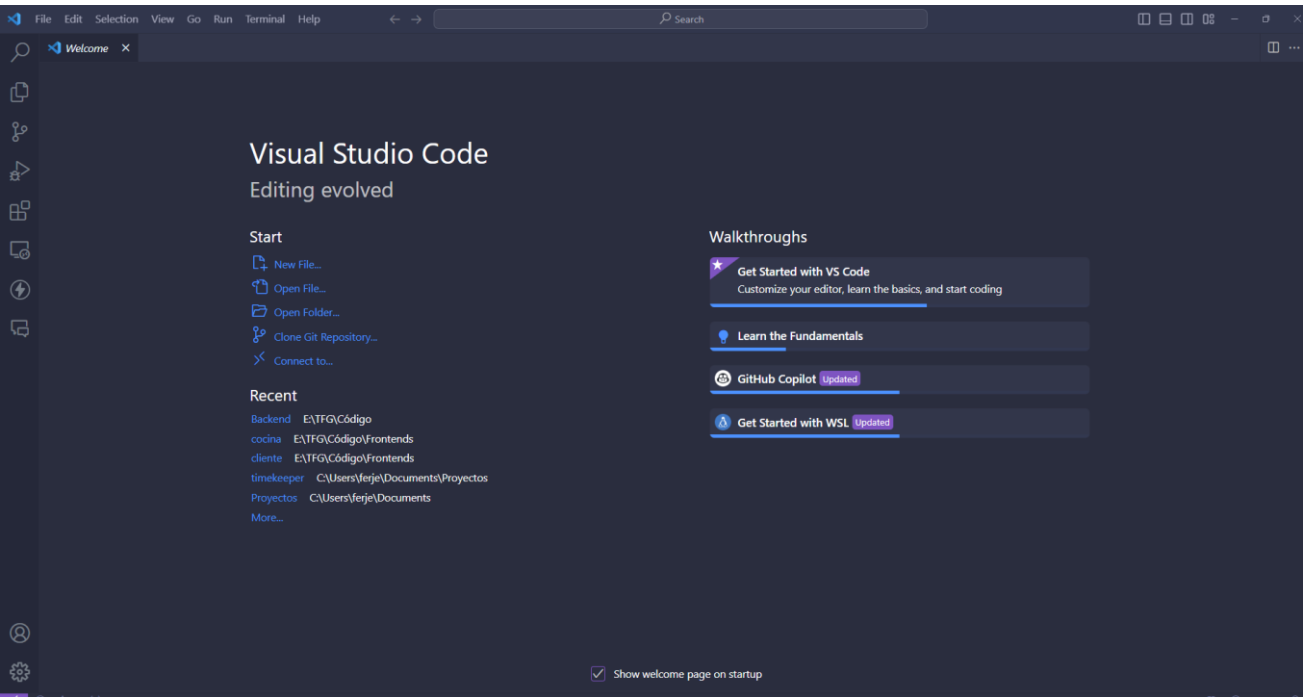
La interfaz debe ser intuitiva y fácil de usar por cualquier tipo de usuario.

En el apartado de conclusiones y ampliaciones habrá más información sobre objetivos futuros.

Cuestiones metodológicas

Entorno de trabajo

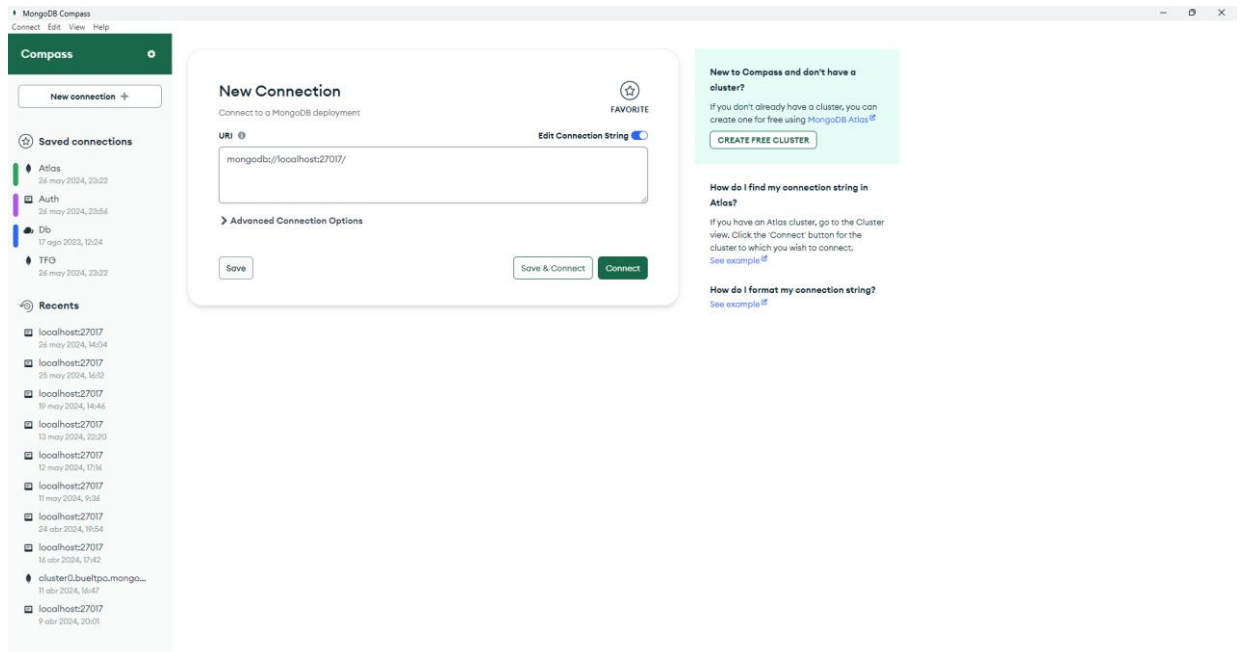
Como herramienta principal del desarrollo de la aplicación se usará Visual Studio Code, este es el editor de texto más usado en la actualidad. Es propiedad de Microsoft pero es libre y multiplataforma.



La herramienta de control de versiones usada es GitHub. El enlace al repositorio es el siguiente: <https://github.com/FerZeg/TFG>

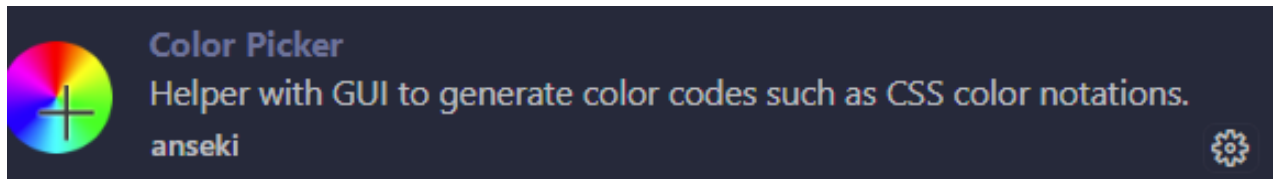
 FerZeg Merge branch 'development' ✓	7633859 · 16 hours ago	 167 Commits
 Código	Arreglado bug carrito	16 hours ago
 Documentos	Muchos cambios	18 hours ago
 .gitignore	Primer commit, creado repositorio	2 months ago
 Dockerfile	Editado el comando	last month
 README.md	Update README.md	last month

Para manejar la base de datos uso MongoDB Compass, una interfaz super potente que permite tanto ejecutar comandos como modificar configuraciones del sistema.

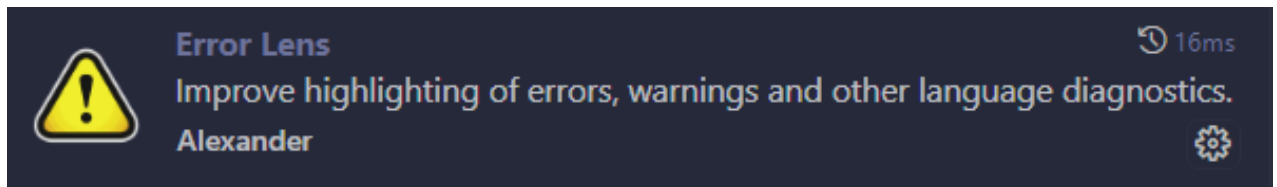


Para planificar la aplicación se ha usado Notion, un software de gestión de proyectos y para tomar notas. Con esta aplicación se ha organizado tanto la temporalidad del proyecto como las diferentes tareas a realizar.

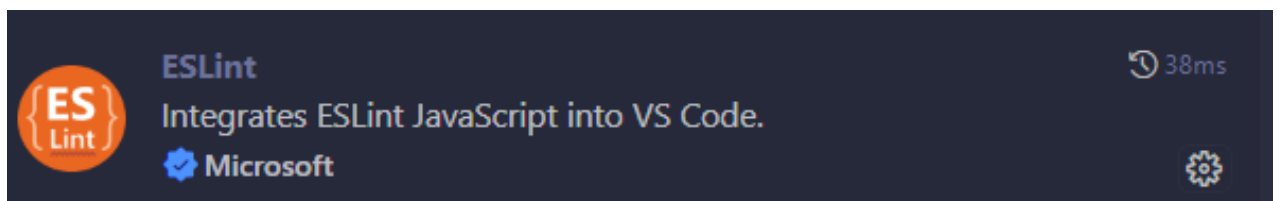
Extensiones



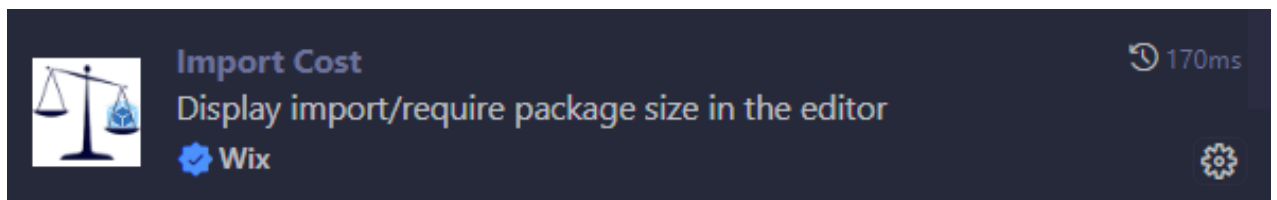
Ayuda a escoger colores.



Ver los errores en el editor directamente.



Integra la herramienta ESLint en el editor, muy útil para detectar errores o posibles malfuncionamientos.



Te dice cuanto ocupará cada extensión.

Descripción general del producto

Límites del sistema

La aplicación es totalmente autónoma, es decir, no depende de otro proyecto web. Al ser una aplicación web puede ser accesible desde cualquier lado en cualquier momento.

Se puede instalar en cualquier sistema operativo al ser NodeJS un intérprete instalable en cualquier entorno.

Es recomendable usar un entorno que tenga por lo menos 1 núcleo para poder ejecutarse. NodeJS se ejecuta en un solo núcleo por lo que no es necesario más. Se puede hacer crecer a la aplicación de manera horizontal haciendo nuevas instancias, en vez de asignar mayores recursos.

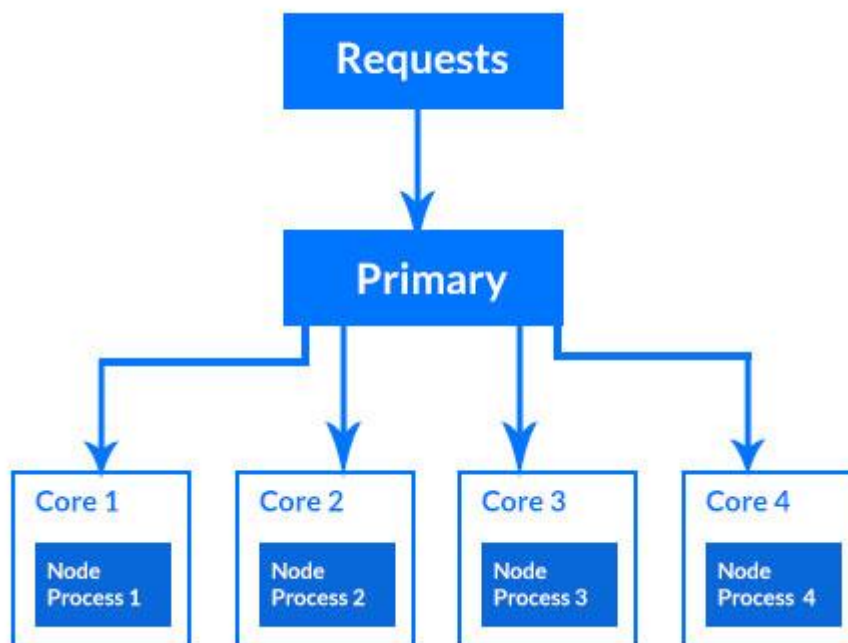


Imagen de <https://medium.com/>

Funcionalidades básicas

La aplicación se define por las siguientes funcionalidades básicas:

- Gestión de tickets. (Ver, borrar)
- Gestión de usuarios y roles.
- Gestor de pedidos de comida.
- Gestión de datos internos.
- Filtrado de datos.
- Autenticación (2 diferentes, usuarios y mesas)
- Datos en tiempo real.

Usuarios

La aplicación para una posible futura ampliación se ha desarrollado de tal manera que pueda haber usuarios con más privilegios que otros. En el contexto de la aplicación hay 2 tipos de usuarios; los superadmin y los usuarios normales. Para la presentación de esta aplicación un usuario superadmin solo podría interactuar con la API de la plataforma, no se ha diseñado el frontend para soportarlo.

Dentro de los usuarios normales existen otros dos tipos de usuarios hasta la fecha. Los cocineros tienen acceso únicamente a las funcionalidades básicas del restaurante, como ver los pedidos o editar los platos. Por otro lado, los administradores tienen acceso completo al entorno de su restaurante, lo que les permite modificar los datos del establecimiento, gestionar los usuarios y mesas, y acceder a los tickets.

Así se ve un usuario en la base de datos

```
_id: ObjectId('6654bb04a98531a3ddc13d78')
nombre: "admin"
contraseña: "$2b$10$cfd21S5f9Y0Xq8o0ve4Fnus5./xwZys922GUKbIiAfN5rqnD4Nmb0"
email: "admin@admin.com"
type: "normal"
createdAt: 2024-05-27T16:55:32.322+00:00
lastModifiedDate: 2024-05-27T16:55:32.322+00:00
__v: 0
```

Y así los usuarios designados en restaurante con su rol.

```
▼ users: Array (2)
  ▼ 0: Object
    user: ObjectId('6654bb04a98531a3ddc13d78')
    role: "admin"
  ▼ 1: Object
    user: ObjectId('6654bb04a98531a3ddc13d79')
    role: "cocinero"
```

Hosting

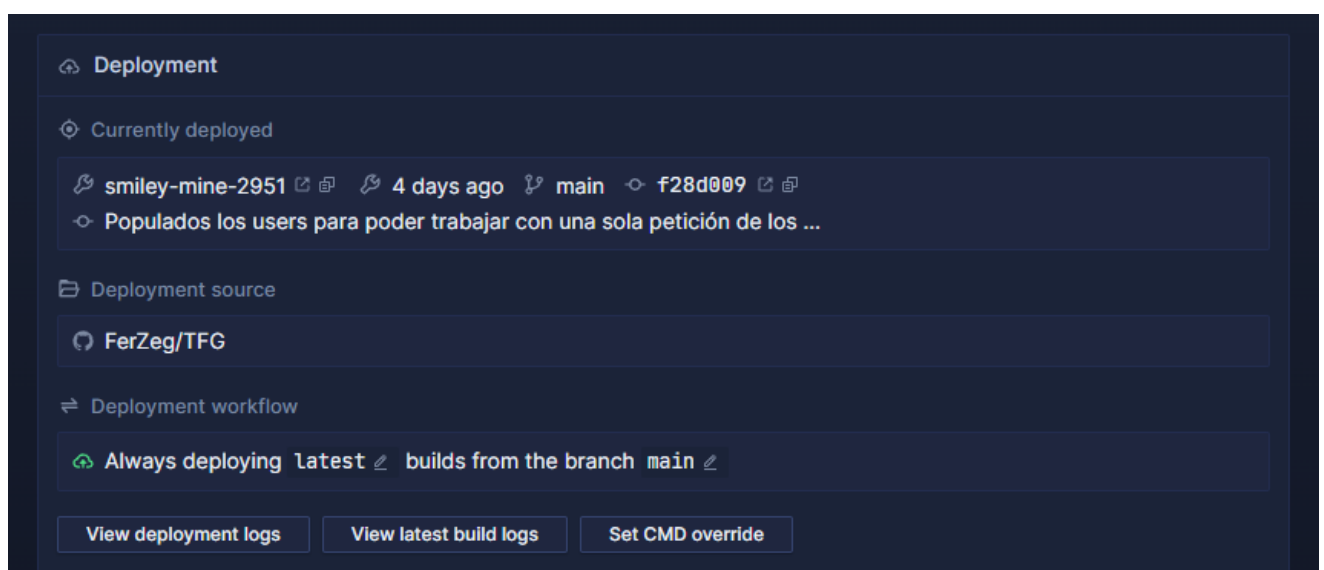
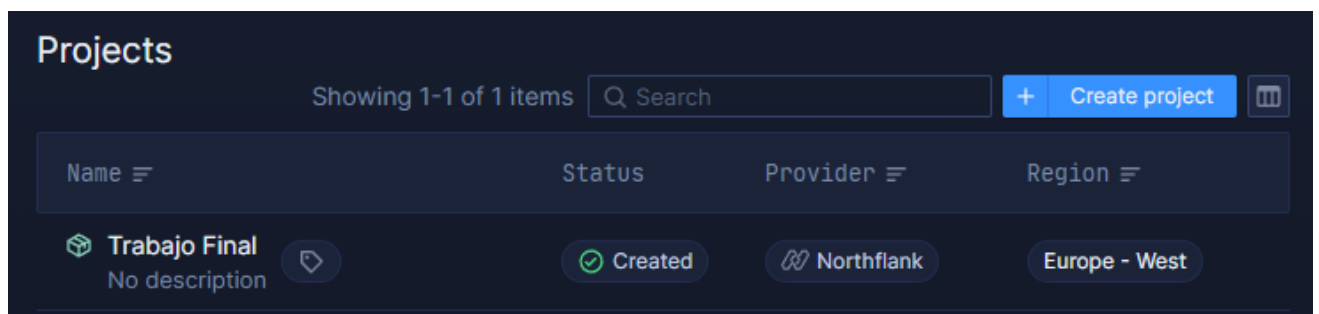
Hay 3 hostings utilizados, diferenciando la parte del front(interfaces de usuario), la del servidor y la de la base de datos.

- Para las diferentes interfaces se usa Cloudflare.
- Para el servidor se usará Northflank.
- Para la base de datos se usará Mongo Atlas.

Cloudflare y Northflank son de uso gratuito, siendo Cloudflare un *CDN (Content Delivery Network)* bastante popular y ampliamente usado por muchas aplicaciones, en contraposición a Northflank que es un servicio bastante nuevo y poco conocido.

El frontend al ser una **SPA (Single Page Application)** termina siendo una compilación de archivos estáticos que se pueden distribuir a través de la red mundial de Cloudflare. Esto reduce de forma considerable el tiempo de carga.

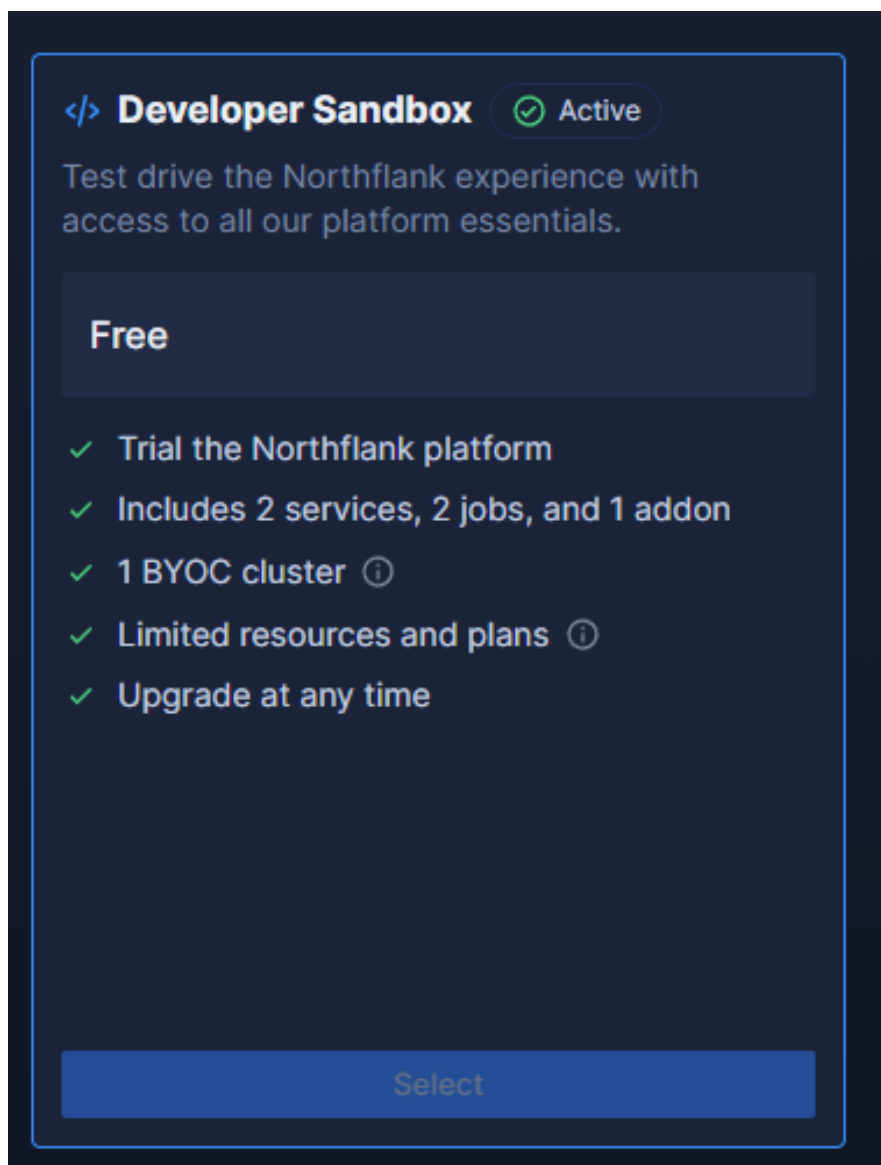
Northflank es una plataforma Cloud que trae muchas funcionalidades para facilitar el despliegue de la aplicación. Con un dockerfile sencillo defino el entorno de ejecución de mi servidor. (Referencia en el apartado de instalación).



Entre otras funcionalidades este servicio ofrece:

- Panel de control donde ver los logs de la aplicación
- Diferentes tipos de métricas
- Una consola de la instancia que se está ejecutando
- Sistema para exponer puertos diferentes según las necesidades
- Herramienta para configurar dominios customizados, certificados de SSL automáticos al conectar el domino.
- Ejecutar código cada cierto tiempo (Cron Jobs) o manualmente.
- Bases de datos.
- Usuarios y equipos.
- Notificaciones.

Todo esto con una capa gratuita bastante generosa para probar proyectos sin un tráfico excesivo.



</> Developer Sandbox ✓ Active

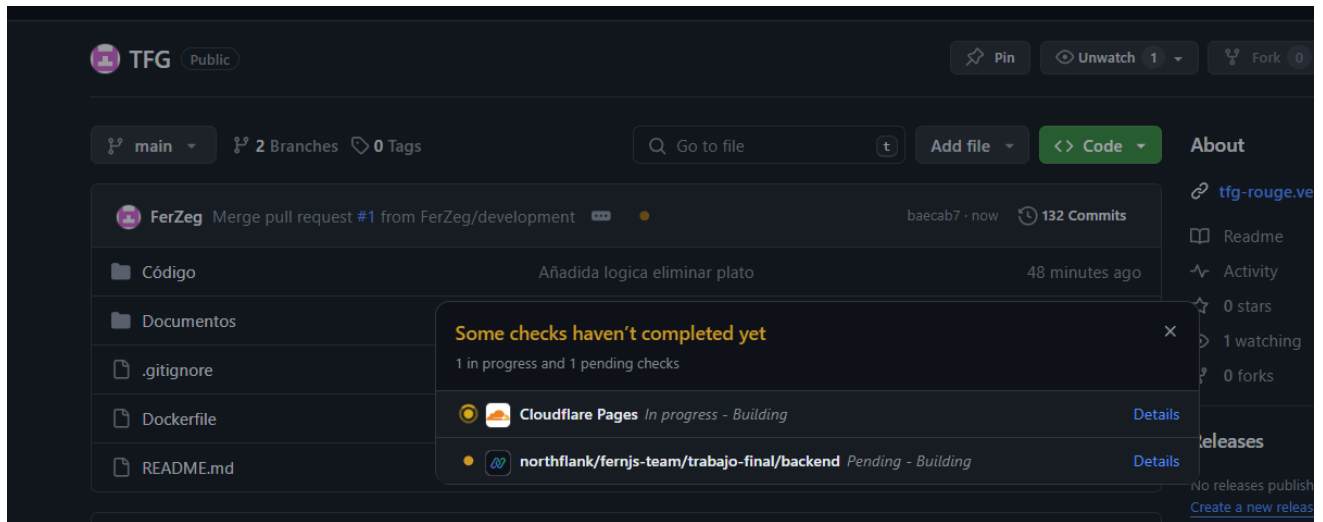
Test drive the Northflank experience with access to all our platform essentials.

Free

- ✓ Trial the Northflank platform
- ✓ Includes 2 services, 2 jobs, and 1 addon
- ✓ 1 BYOC cluster ⓘ
- ✓ Limited resources and plans ⓘ
- ✓ Upgrade at any time

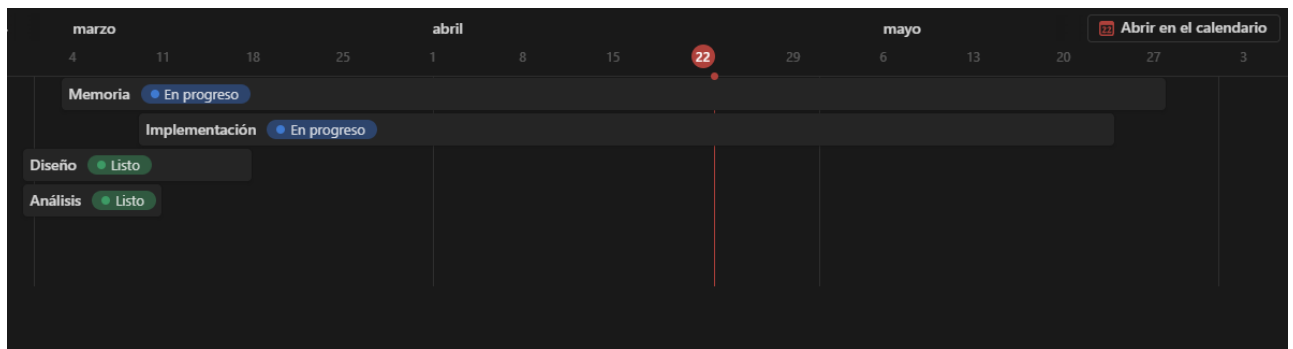
Select

Ambas plataformas ofrecen un servicio de *CD (Continuous Deployment)*, lo que facilita muchísimo el despliegue de la aplicación al ser automatizado al hacer un push a la rama Main del repositorio.



Planificación y presupuesto

Planificación



Esta es la planificación que he propuesto para la consecución de los objetivos de este trabajo. Es un resumen grosso modo de las diferentes fases troncales del desarrollo.

La planificación en un proyecto de una sola persona es muy subjetiva y depende mucho de las circunstancias personales de cada uno en cada momento. En mi caso, aunque no sea muy evidente ya que los tiempos se han cumplido, he tenido que meter un ligero sprint final al casi no llegar a la fecha de entrega propuesta.



Este es el mapa de calor de contribuciones diarias, esto cuenta los commits que se han hecho al día. Como se ve, ha habido una contribución constante durante los meses de abril y Mayo sobre todo.

Metodología

La metodología usada es la metodología en cascada retroactiva, es simplemente una modificación a la tan conocida metodología en cascada paso por paso pero con comprobaciones y pudiendo volver a una fase anterior en caso de ser necesario.

En realidad, ninguna metodología es usada al 100% ya que es inviable por todas las variables diferentes que ocurren en un desarrollo, pero esta es la que mejor se ajusta a la utilizada en este caso.



Sobre todo hay que destacar el proceso que se ha tenido que hacer con el modelado de la base de datos al mismo tiempo que se desarrollaban funcionalidades que quizás no estaban previstas desde el inicio. Se ha tenido que volver varias veces a la etapa de diseño para hacer esos cambios y que encajaran correctamente sobre el código ya hecho.

Bocetos

PLATOS

BEBIDAS

EXTRAS

EJEMPLO_COMIDAS

TOTAL 10.90€

PEDIR

FINALIZAR

COCINA

x3

x1

Más antiguos

COCINA

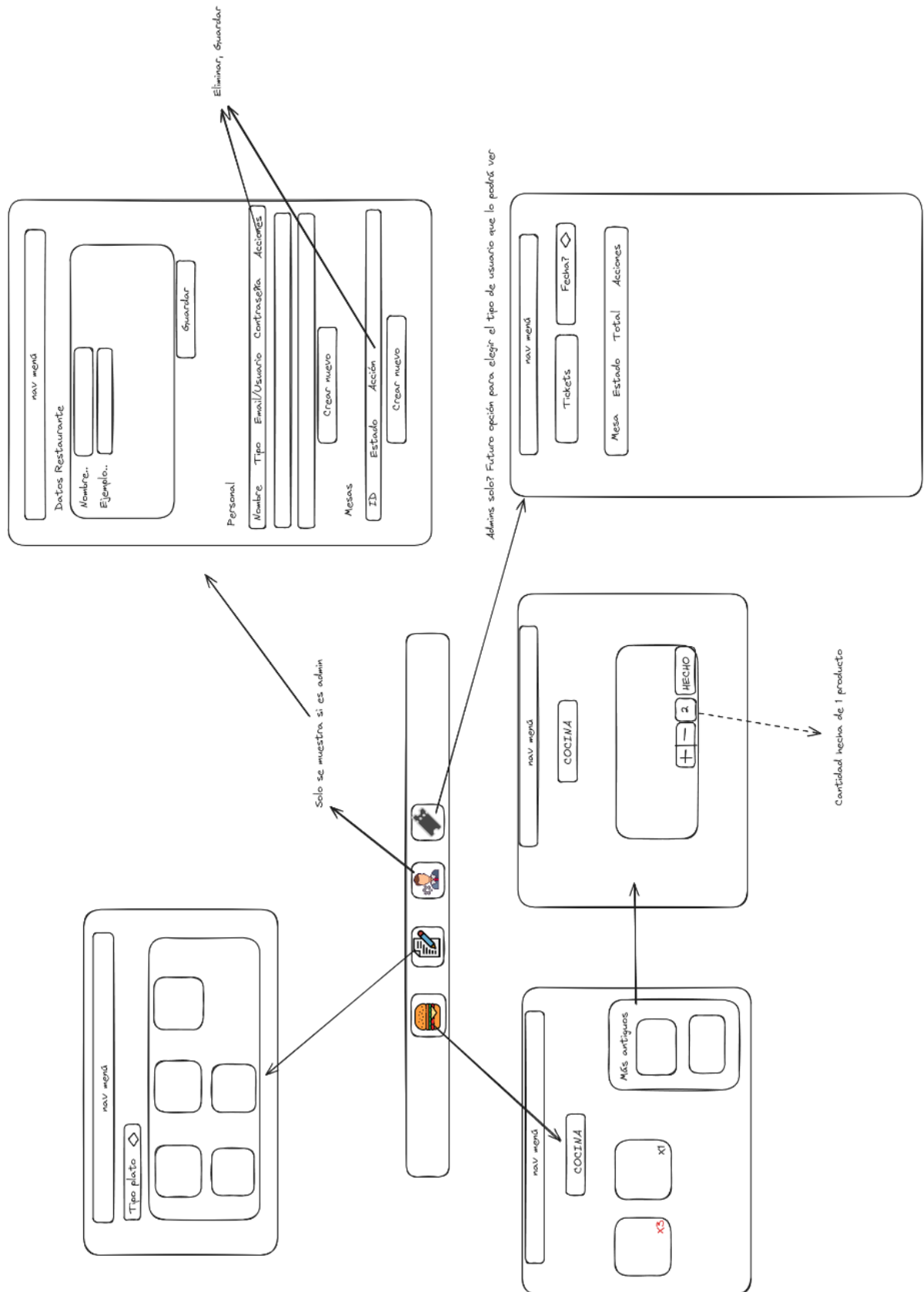
+

-

2

HECHO

Cantidad hecha de 1 producto



Interfaces finales

Admin

Página de error

CÓDIGO ERROR: 404

OOOPS!!

¡La pagina que estabas buscando no existe!

Volver



ComidaEnMarcha

Entrar

En la pantalla de login del panel de trabajos solo se necesita un correo y una contraseña. Los datos por defecto cuando se cargan de ejemplo son:

- Correo: admin@admin.com
- Contraseña: *admin*

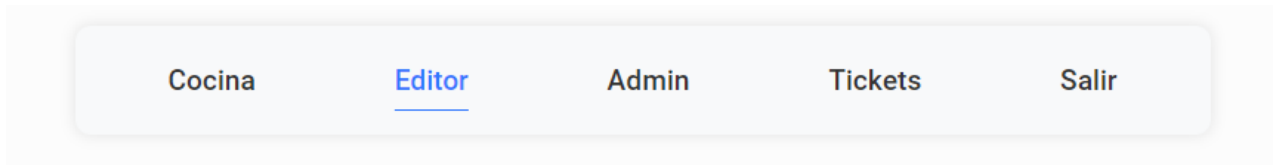
ComidaEnMarcha

Mesas

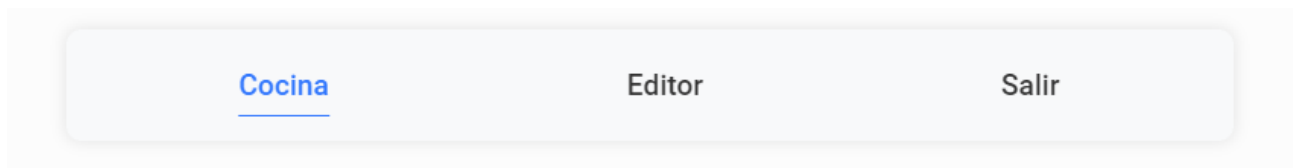
En la pantalla de login del cliente solo se necesita un correo, una mesa y una contraseña. Los datos por defecto cuando se cargan de ejemplo son:

- Correo: admin@admin.com
- Usuario: Mesa 1
- Contraseña: 1234

Navegación

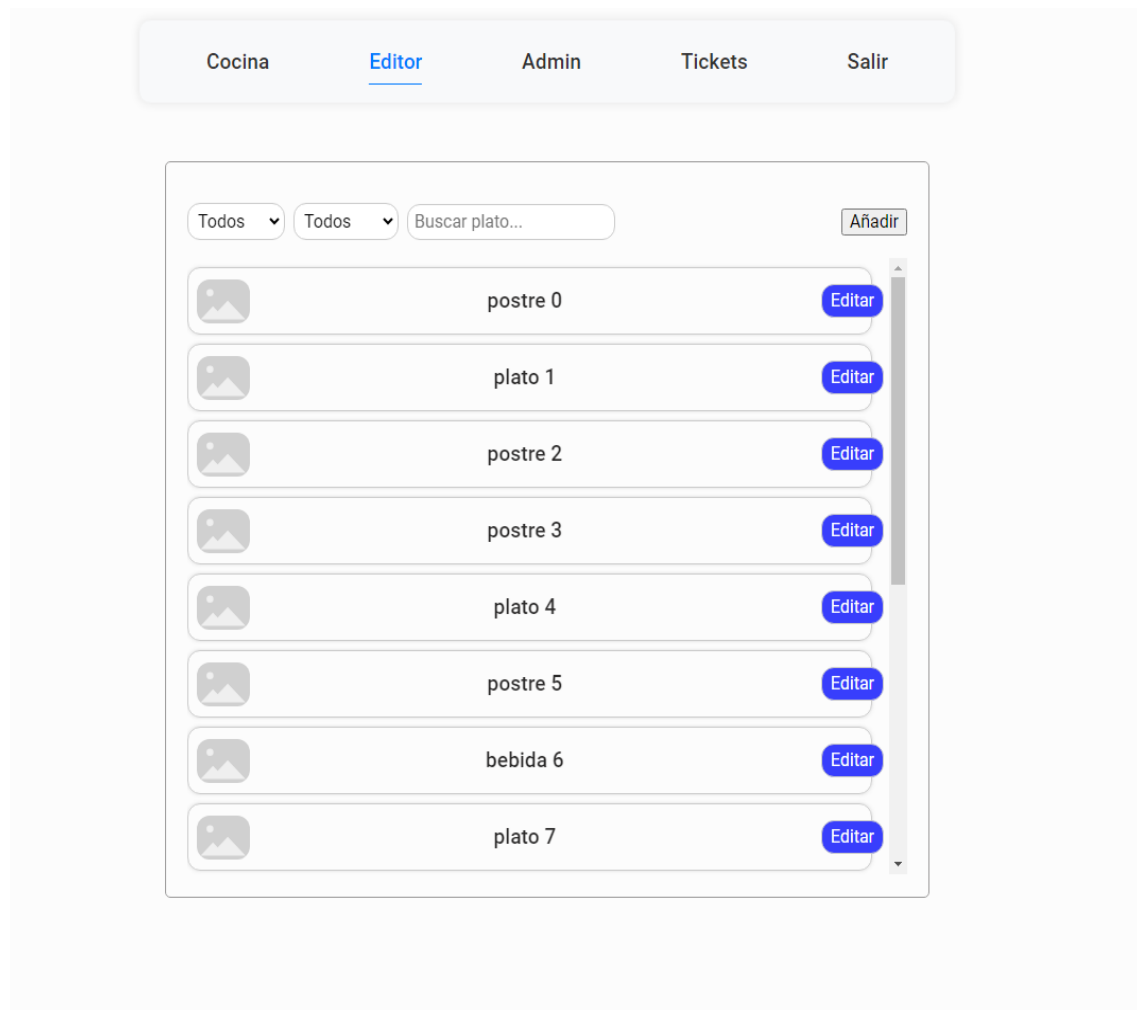


El menú de navegación de un administrador se ve así. Tiene acceso a todas las secciones.



Como se puede ver la navegación de un usuario en este caso, de rol cocinero, no tiene la parte del Admin y Tickets. Esto se consigue mediante el renderizado condicional en React.

Editor de comidas




En la siguiente interfaz se pueden tanto visualizar con diferentes filtros (tipo, estado y búsqueda) como editar los productos existentes e incluso añadir nuevos.

Menú edición

Cocina Editor Salir

Editar plato



Nombre:

Precio:

Tipo:

Activo:

Imagen: Ningún archi... seleccionado

Cuando quieres editar o añadir un plato se abre un menú con una especie de formulario. Aquí puedes subir una imagen, o configurar el producto a tu gusto. Cuando subes una imagen puedes ver una preview antes de guardarlo.

Editor del restaurante

[Cocina](#) [Editor](#) [Admin](#) [Tickets](#) [Salir](#)

Datos Restaurante
Nombre

Dirección

Teléfono

Contraseña Mesas

Personal

Nombre	Rol	Contraseña	Email	Acciones
admin	Admin ▼	admin@admin.com	<input type="button" value="Guardar"/> <input type="button" value="Eliminar"/>
cocinero	Cociner ▼	cocinero@cocinero.co	<input type="button" value="Guardar"/> <input type="button" value="Eliminar"/>

[Añadir](#)

Esta interfaz es exclusiva para los administradores del establecimiento, se pueden editar diversos datos del propio negocio como las cuentas de usuario relacionadas con este y las mesas.

Pantalla de tickets

Cocina

Editor

Admin

Tickets

Salir

Mesa	Fecha	Estado	Total	Acciones
Mesa 1	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 1	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 1	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 1	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 2	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 2	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 2	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 2	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 2	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 3	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 3	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 3	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 4	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 4	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 4	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>
Mesa 4	8/5/2024 20:17:37	ABIERTO	0.00 €	<div>Eliminar</div>

En esta interfaz se pueden visualizar los diferentes tickets que han sido creados en la aplicación. De momento la única funcionalidad disponible es borrar el ticket.

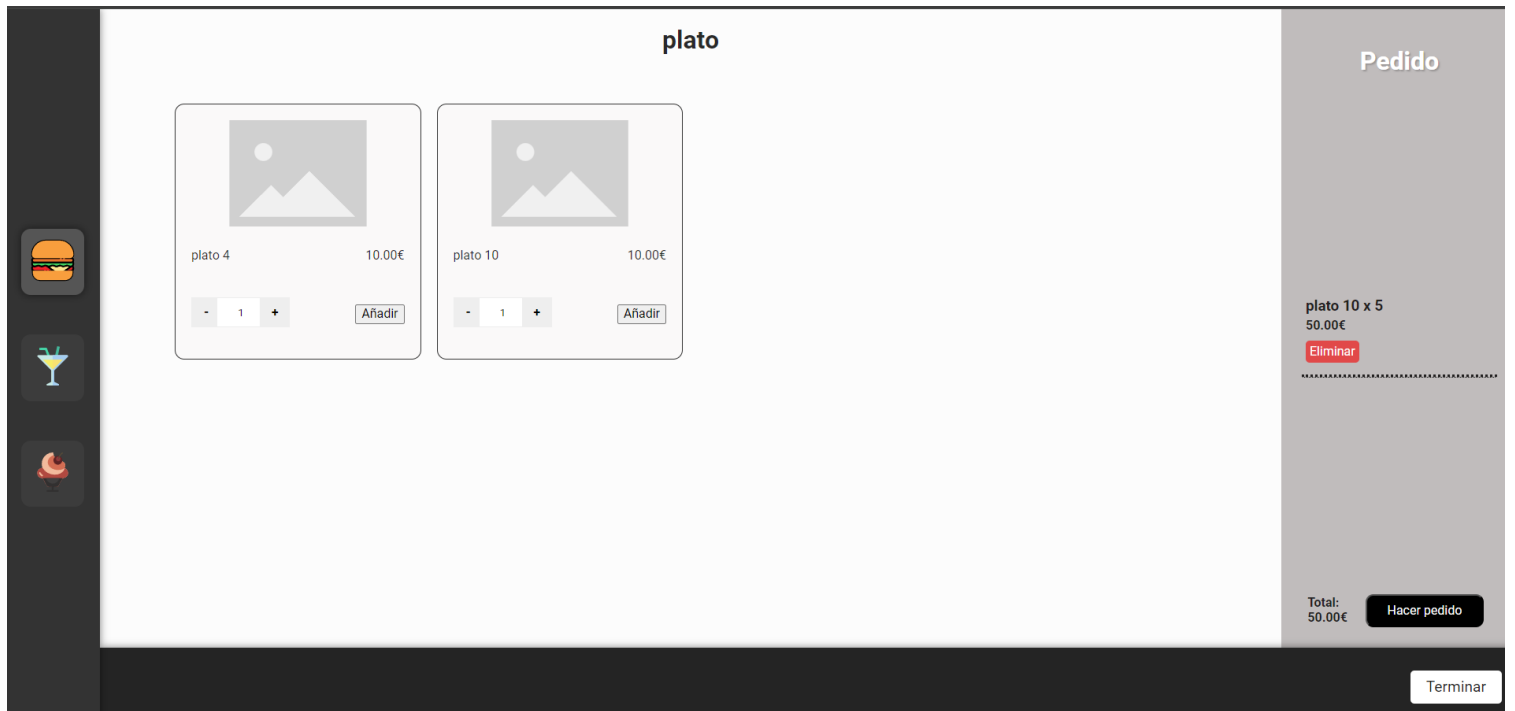
Cliente

Página para iniciar

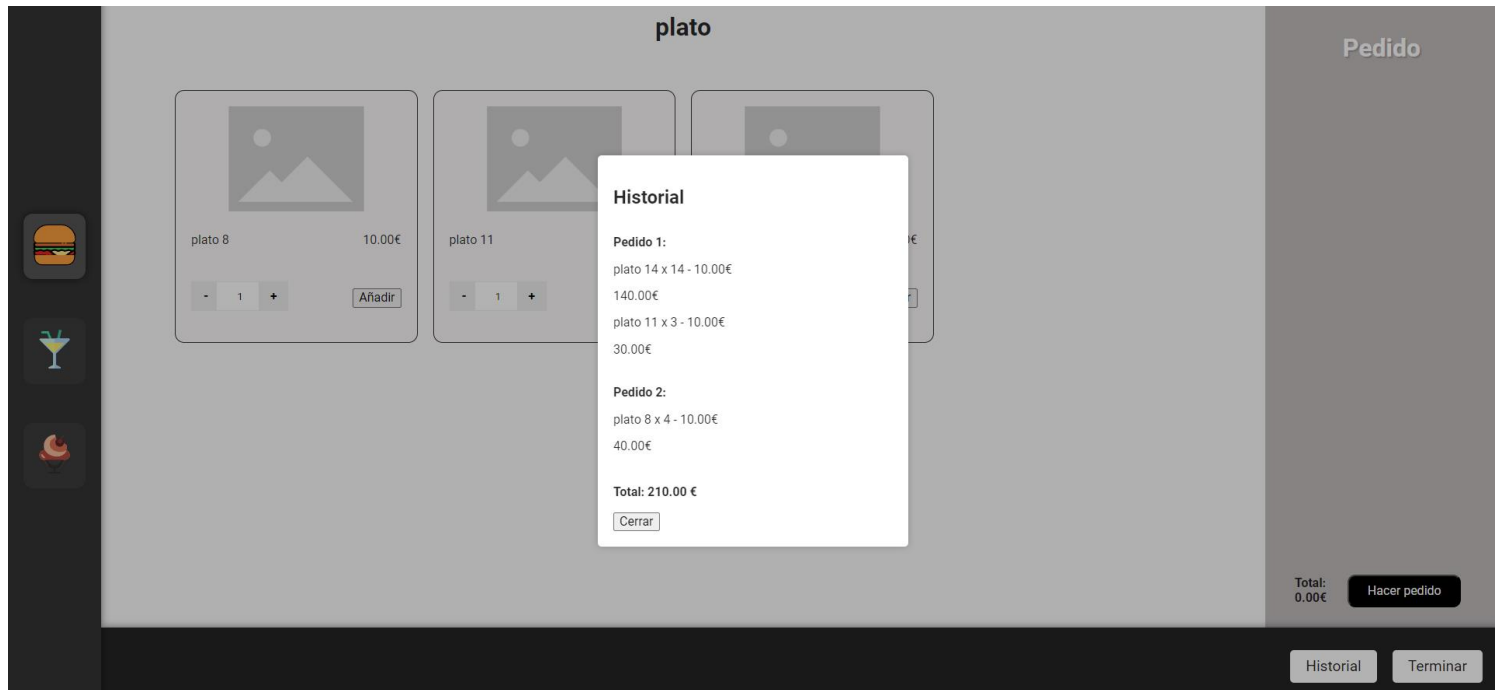
Gaeguri

COMENZAR

Interfaz principal



Historial de pedido



Responsive cliente

La aplicación no ha sido diseñada principalmente en vertical, es una de las posibles futuras mejoras. Es usable pero no es recomendable.



Responsive panel administrador

El panel de control si es más usable desde un dispositivo con una ratio de aspecto más estrecho.



Otros diseños

Icono



Este es el icono principal de la aplicación. Se ha desarrollado con ayuda de una inteligencia artificial y representa un huevo frito feliz.

Presupuesto

El grosor del coste del desarrollo de la aplicación es el capital humano ya que las herramientas utilizadas son todas software libre, así como los alojamientos están usando capas gratuitas de diferentes proveedores.

El único coste extra a parte del propio trabajo ha sido el dominio `cocinaenmarcha.com` que tiene un coste de 10€/año.

El coste por horas de un programador Junior oscila entre 15€ - 20€ la hora en España. Al ser el primer proyecto el código base se computará a 15€ la hora.

El coste total por lo tanto será, $140 \text{ horas} * 15 \text{ €} + 10 \text{ €} = 2110 \text{ €}$.

Modelo de negocio

Esta aplicación no está diseñada para venderse a un cliente particular, si no para ser un modelo de suscripción a contratar por diferentes establecimientos.

El coste sería de alrededor de 150€/mes por establecimiento, y debido a la necesidad de ampliar los recursos de servidor se tendría un coste variable de 10€ por establecimiento.

Se tendrán en cuenta proposiciones de cambios o si hubiese una necesidad de una funcionalidad personalizada se presupuestaría.

Los mínimos como objetivo serían tener 5 clientes al mes y como cifra de éxito serían alrededor de 10. Estos darían un ingreso de 700€ o 1400€ respectivamente siendo el único trabajo un mantenimiento pasivo y ocasionalmente trabajos presupuestados.

Documentación técnica

Base de datos

La base de datos que he elegido para almacenar la información es MongoDB, una base de datos no relacional muy flexible. He utilizado este sistema para aprender más sobre él. En este tipo de aplicaciones pequeñas es un poco indiferente qué tipo de sistema elegir, si relacional o no relacional, debido a que el coste computacional es muy bajo. En una aplicación más grande yo habría optado por un sistema mixto o por el uso de PostgreSQL, que tiene muchas más funcionalidades que MySQL u otros derivados.

Entidades

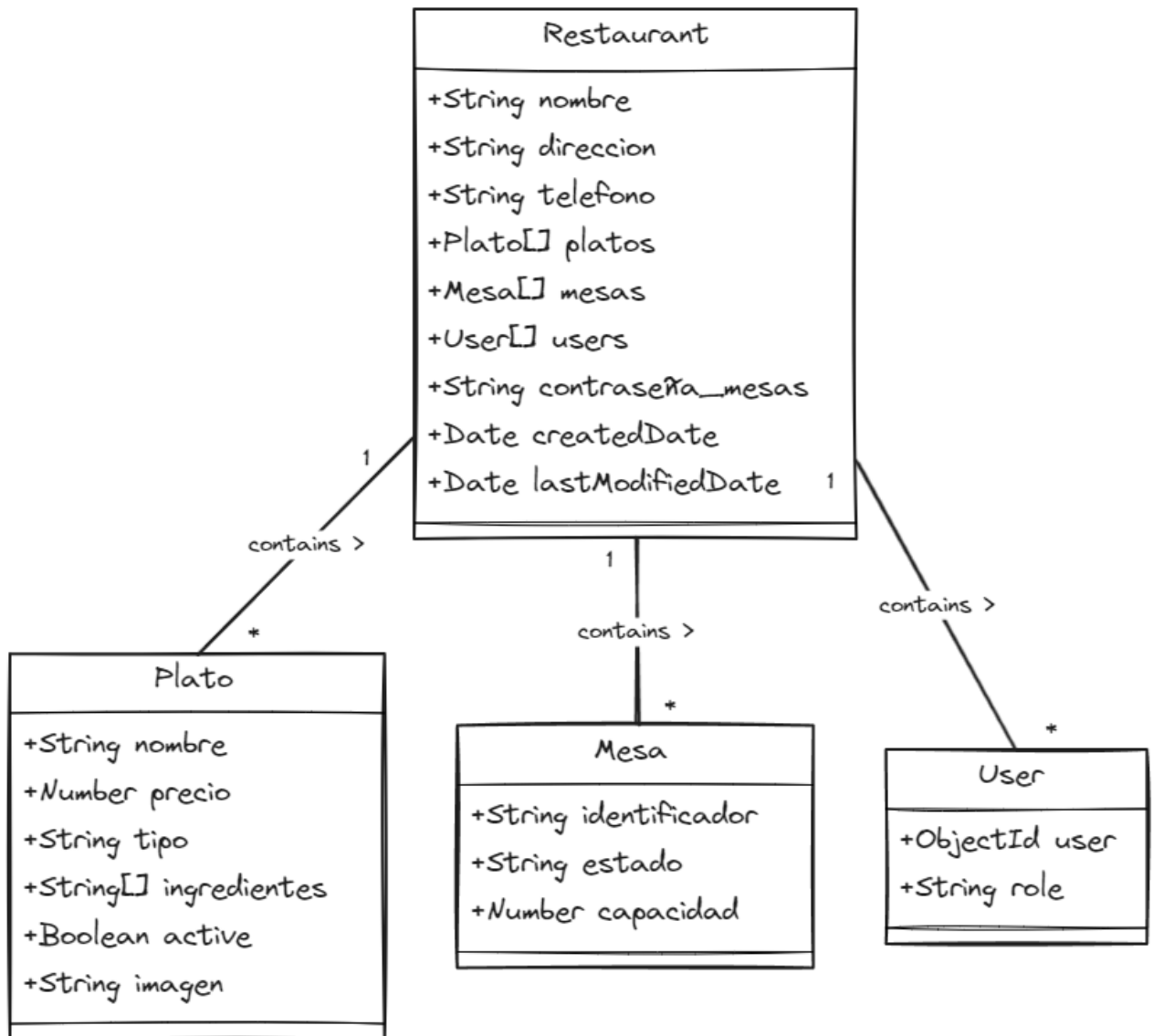
He definido las diferentes entidades existentes de la aplicación mediante una definición en sus tipos en formato JSON. He generado un diagrama mediante la herramienta excalidraw para poder visualizarlo de la mejor manera,

User

```
nombre: "String"  
contraseña: "String"  
email: "String"  
createdDate: "Date"  
lastModifiedDate: "Date"  
type: "String"
```

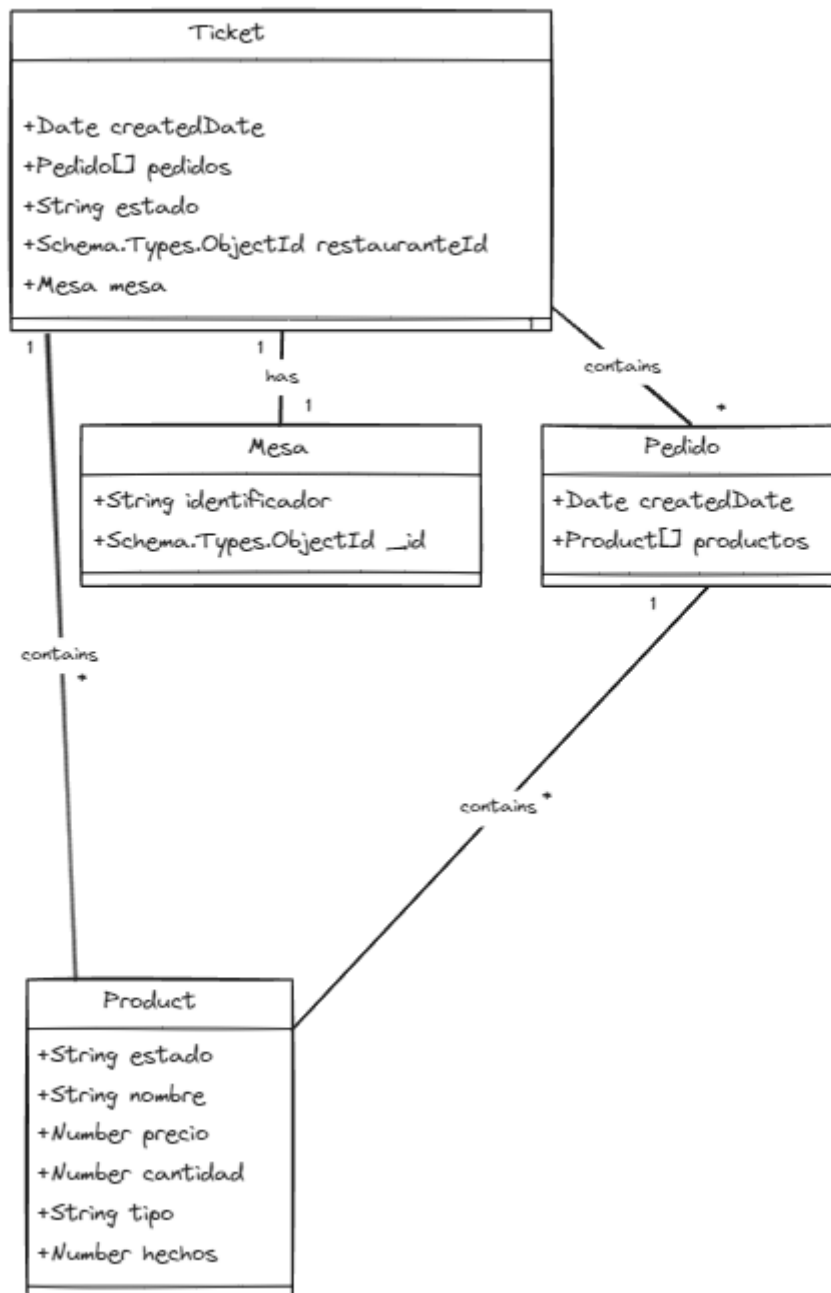
Usuario es la entidad más sencilla, ya que solo contiene estos datos. Un usuario no pertenece a un restaurante, si no que el restaurante define la relación con el usuario.

Restaurante



Restaurante es la entidad principal, contiene los datos del restaurante, roles de usuario, mesas y productos. Los usuarios definen los usuarios que pertenecen y sus roles.

Ticket



restaurantId es una referencia a un documento de la colección Restaurante.

Rutas API

Las rutas se han dividido en diferentes instancias del enrutamiento de express.js para facilitar su gestión.

Main Router

Rutas principales del sistema, agrupando las funcionalidades de autenticación y gestión de restaurantes.

****GET**** /auth

****GET**** /restaurantes

Auth Router

Rutas relacionadas con la autenticación y la gestión de datos de mesas.

****POST**** /auth/login - Iniciar sesión.

****GET**** /auth/data - Obtener datos de usuario autenticado.

****POST**** /auth/mesa - Autenticar una mesa.

****GET**** /auth/mesa - Obtener datos de una mesa autenticada.

Restaurante Router

Rutas para la gestión de restaurantes, incluyendo creación, actualización, eliminación y obtención de datos de restaurantes. También incluye subrutas para la gestión de usuarios, mesas, tickets y platos dentro de un restaurante específico.

****GET**** /restaurantes/:restauranteld - Obtener datos de un restaurante por ID.

****GET**** /restaurantes/ - Obtener datos de todos los restaurantes.

****POST**** /restaurantes/ - Crear un nuevo restaurante.

****PUT**** /restaurantes/:restauranteld - Actualizar un restaurante por ID.

****DELETE**** /restaurantes/:restauranteld - Eliminar un restaurante por ID.

Subrutas de Restaurante:

****GET**** /restaurantes/:restauranteld/users - Gestión de usuarios.

- ****GET**** /restaurantes/:restaurantId/mesas - Gestión de mesas.
- ****GET**** /restaurantes/:restaurantId/tickets - Gestión de tickets.
- ****GET**** /restaurantes/:restaurantId/platos - Gestión de platos.

User Router

Rutas para la gestión de usuarios dentro de un restaurante específico.

- **GET**** /restaurantes/:restaurantId/users - Obtener usuarios.
- **POST**** /restaurantes/:restaurantId/users - Crear un nuevo usuario.
- **DELETE**** /restaurantes/:restaurantId/users/:userId - Eliminar un usuario por ID.
- **PUT**** /restaurantes/:restaurantId/users/:userId - Actualizar un usuario por ID.

Ticket Router

Rutas para la gestión de tickets dentro de un restaurante específico, incluyendo operaciones sobre productos y pedidos.

- **GET**** /restaurantes/:restaurantId/tickets - Obtener todos los tickets.
- **GET**** /restaurantes/:restaurantId/tickets/pendiente - Obtener tickets pendientes.
- **GET**** /restaurantes/:restaurantId/tickets/:ticketId - Obtener un ticket por ID.
- **DELETE**** /restaurantes/:restaurantId/tickets/:ticketId - Eliminar un ticket por ID.
- **POST**** /restaurantes/:restaurantId/tickets/:ticketId/producto/status - Actualizar el estado de un producto en un ticket.
- **POST**** /restaurantes/:restaurantId/tickets/:ticketId/producto/quantity - Actualizar la cantidad de un producto en un ticket.
- **POST**** /restaurantes/:restaurantId/tickets/:ticketId/pedido - Crear un nuevo pedido en un ticket.
- **POST**** /restaurantes/:restaurantId/tickets/:ticketId/finish - Finalizar un ticket.

Plato Router

Rutas para la gestión de platos dentro de un restaurante específico.

****PUT**** /restaurantes/:restaurantId/platos/:platId - Actualizar un plato por ID.

****PUT**** /restaurantes/:restaurantId/platos - Actualizar platos.

****DELETE**** /restaurantes/:restaurantId/platos/:platId - Eliminar un plato por ID.

****GET**** /restaurantes/:restaurantId/platos - Obtener todos los platos.

Mesas Router

Rutas para la gestión de mesas dentro de un restaurante específico.

****DELETE**** /restaurantes/:restaurantId/mesas/:mesaId - Eliminar una mesa por ID.

****PUT**** /restaurantes/:restaurantId/mesas/:mesaId - Actualizar una mesa por ID.

****POST**** /restaurantes/:restaurantId/mesas/:mesaId/ticket - Crear un ticket para una mesa específica.

Cada una de estas rutas está protegida por controladores de permisos que aseguran que solo los usuarios con los roles adecuados pueden acceder y modificar los datos correspondientes.

Estructura del código

Librerías

Librerías principales del proyecto en el lado del servidor:

- Express: Framework de desarrollo para facilitar la elaboración del servidor, consta de un sistema avanzado de routing y librerías para parsear las cookies u otros.
- Mongoose: Interfaz para la base de datos Mongo, permite crear Schemas (Estructuras de objetos) y muchas funciones añadidas.
- Bcrypt: Librería para encriptar las contraseñas.
- Jsonwebtoken: Librería para generar tokens con una carga que se pueden codificar, decodificar y verificar mediante una contraseña. Es muy útil su uso para el sistema de autenticación y persistencia de la sesión.
- Formidable: Módulo utilizado para manejar formularios y archivos enviados mediante peticiones HTTP. Es especialmente útil para procesar formularios multipart/form-data, que son comunes en la carga de archivos a través de la web.

Librerías principales del proyecto en el lado del cliente:

- React: Framework de javascript creado por Facebook (actualmente META) que facilita mucho la creación de software mediante la encapsulación de código, tanto el lenguaje de marcado como las funcionalidades, para una mayor reutilización de este. Es actualmente el framework de javascript más usado en la actualidad.
- Vite: Herramienta de compilación de código ampliamente usada en la actualidad por su rapidez en tiempo de compilación o actuando como servidor de desarrollo para aplicar los cambios en el código en tiempo real cuando guardas un archivo.
- Sonner: Librería que se encarga de las notificaciones mediante cuadros de diferentes colores.
- React-router-dom: Librería hecha para react pero que no es parte de ella directamente, la uso para manejar las rutas de la aplicación y crear una SPA (Single Page Application).

Códigos de interés

Subida de imágenes

Con este código se maneja el acceso al contenedor que contiene los archivos estáticos. Usa el mismo api que AWS y son casi compatibles al 100% por lo que hay bastante documentación.

```
import { S3Client } from "@aws-sdk/client-s3"

export const S3ClientAccess = () => {
  const { ACCOUNT_ID, ACCESS_KEY_ID, SECRET_ACCESS_KEY } = process.env
  const S3 = new S3Client({
    region: "auto",
    endpoint: `https://${ACCOUNT_ID}.r2.cloudflarestorage.com`,
    credentials: {
      accessKeyId: ACCESS_KEY_ID,
      secretAccessKey: SECRET_ACCESS_KEY,
    },
  })
  return S3
}
```

Este trozo de código ha sido un verdadero reto para realizar. Es el controlador que maneja la edición de los platos, pero en específico la parte para subir archivos a “R2 de Cloudflare”.

```
export const putPlato = async(req, res) => {
  const uploads = []

  const form = formidable({
    maxFiles: 1,
    maxFileSize: 5 * 1024 * 1024,
    filter: function ({mimetype}) {
      return mimetype && mimetype.includes("image")
    },
    fileWriteStreamHandler: function(file) {
      const passThroughStream = new PassThrough()
      const params = {
        Bucket: process.env.BUCKET_NAME,
        Key: `platos/${file.newFilename}`,
        Body: passThroughStream,
        ContentType: file.mimetype,
      }
      const upload = new Upload({
        client: S3ClientAccess(),
        params
      })
      const uploadPromise = upload.done()
        .then((data) => {
          file.location = process.env.MEDIA_URL + "/" + data.Key
        })
      uploads.push(uploadPromise)

      return passThroughStream
    }
  })
}
```

Usando la librería formidable, recojo los datos que llegan al servidor y se procesan para poder crear una tubería entre mi servidor y Cloudflare. Esto se llama comúnmente “stream de datos”, mi servidor no está guardando los archivos directamente, si no que redirige el flujo de los datos hacia el contenedor de Cloudflare.

```
try {
  const [fields, file] = await form.parse(req)
  await Promise.all(uploads)
  if(file.file && file.file[0]) {
    req.body = { ...JSON.parse(fields.datos), imagen: file.file[0].location}
  } else {
    req.body = { ...JSON.parse(fields.datos)}
  }
  const result = await PlatoService.putPlato(req)
  return res.status(201).send(result)
} catch (err) {
  console.log(err)
  if (err.code === formidableErrors.biggerThanMaxFileSize) {
    throw new BadRequestError("El archivo es demasiado grande")
  } else {
    throw new BadRequestError("Error al subir la imagen")
  }
}
```

También verifica que la imagen sea de un formato predeterminado y un tamaño máximo. Después de subir la imagen se actualiza el plato en cuestión en la base de datos con la información recibida junto con la dirección de archivo generada por Cloudflare. En este caso se usan diferentes rutas para la versión de desarrollo y para la de producción.

Errores customizados para respuestas

Es una práctica muy buena crear errores propios para poder estandarizar las respuestas del servidor y poder manejarlas correctamente.

```
class NotFoundError extends Error {
  constructor(message) {
    super(message)
    this.name = "NotFoundError"
  }
}

class ValidationError extends Error {
  constructor(message, errors) {
    super(message)
    this.name = "ValidationError"
    this.errors = errors
  }

  addError(key, error) {
    this.errors[key] = error
  }
}

class UnauthorizedError extends Error {
  constructor(message) {
    super(message)
    this.name = "UnauthorizedError"
  }
}

class BadRequestError extends Error {
  constructor(message) {
    super(message)
    this.name = "BadRequestError"
  }
}

export { NotFoundError, ValidationError, UnauthorizedError, BadRequestError }
```

```
// eslint-disable-next-line no-unused-vars
app.use((err, req, res, next) => {
  if(err instanceof UnauthorizedError) {
    return res.status(401).send({ message: err.message })
  }
  if(err instanceof ValidationError) {
    return res.status(400).send({ message: "Hay errores en los datos enviados", errors: err.errors })
  }
  if(err instanceof NotFoundError) {
    return res.status(404).send({ message: err.message })
  }
  if(err instanceof BadRequestError) {
    return res.status(400).send({ message: err.message })
  }
  res.status(500).send({ message: "Error en el servidor" })
  console.log(err)
})
```

Este es un “**middleware**” que captura los errores por defecto en express. Toda ruta que lance un error pasará por aquí automáticamente y responderá al cliente.

Enrutamiento con React Router

```
const router = createBrowserRouter([
  {
    path: "",
    errorElement: <Errors/>,
    children: [
      { path: "logout", element: <Logout/> },
      { path: "login", element: <Login/> },
      {
        path: "/",
        element: <Layout/>,
        children: [
          { path: "", element: <Cocina/> },
          { path: "edit", element: <EditorPage/> },
          { path: "admin", element: <Admin/> },
          { path: "tickets", element: <TicketsPage/> },
        ]
      }
    ]
  }
])

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <RouterProvider router={router}/>
  </React.StrictMode>
)
```

Usando una jerarquía con formato JSON y con los diferentes componentes creados en la aplicación se crea un enrutamiento en la parte del cliente. Por cómo funciona react, todas las peticiones van dirigidas al index.html que carga el javascript. La librería se encarga de cargar las páginas o componentes necesarios para cada ruta, por lo que /edit y /admin serán totalmente diferentes.

Cuando hay zonas que se repiten o se necesitan implementar funcionalidades comunes, véase autenticación o elementos como la barra de navegación se usan layouts.

Tokens para autenticación

```
function sign(payload) {
  if(!payload.type) {
    throw new Error("Type is required to generate token")
  }
  if(!payload.id) {
    throw new Error("Id is required to generate token")
  }
  if(payload.type === "superadmin") {
    return jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: "1d" })
  }
  if(!payload.restauranteId) {
    throw new Error("Restaurant ID is required to generate token")
  }
  if(!payload.role) {
    throw new Error("Role is required to generate token")
  }
  return jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: "1d" })
}
```

Este código se dedica a firmar tokens después de un inicio sesión satisfactorio. Verifica algunos datos necesarios para poder firmarlo.

La función “sign” de la librería jsonwebtoken toma como parámetro un conjunto de datos, una clave secreta y opcionalmente opciones. En este caso está hecho para que cada 24 horas el token caduque y tenga que volver a iniciar sesión.

```
function verify(token) {
  try {
    return jwt.verify(token, process.env.JWT_SECRET)
  } catch {
    throw new UnauthorizedError("The token is invalid or has expired")
  }
}
```

Para verificar el token y sustraer los datos que almacena se usa la función “verify”.

Punto de entrada React.

Todas las aplicaciones tienen un punto de entrada, en react el punto de entrada está en el index.html.

```
<!doctype html>
<html lang="es">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="icon" href="icono.webp">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
      rel="stylesheet"
      href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;
    >
    <link href="https://fonts.googleapis.com/css2?family=Dancing+Script:wght@400..700&family=Roboto&disf
    <title>Restaurante</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

El punto donde se renderiza la aplicación es ese div con id root.

El módulo main.jsx es donde comienza toda la magia del javascript.

```
ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <RouterProvider router={router}/>
  </React.StrictMode>
)
```

Previsualización de imágenes

Este código es parte de un componente en React del panel de administrador.

```
const [previewImage, setPreviewImage] = useState(null)

const handleImageChange = (e) => {
  const file = e.target.files[0]
  if (file) {
    const reader = new FileReader()
    reader.onloadend = () => {
      setPreviewImage(reader.result)
    }
    reader.readAsDataURL(file)
  }
}
```

```
<img
  src={previewImage || state.imagen || "Placeholder.svg"}
  alt="" className="dialog-img"
/>
```

```
</div>
<div className="input-wrapper">
  <label htmlFor="imagen">Imagen:</label>
  <input type="file" name="imagen" id="imagen" accept="image/*" onChange={handleImageChange}/>
</div>
```

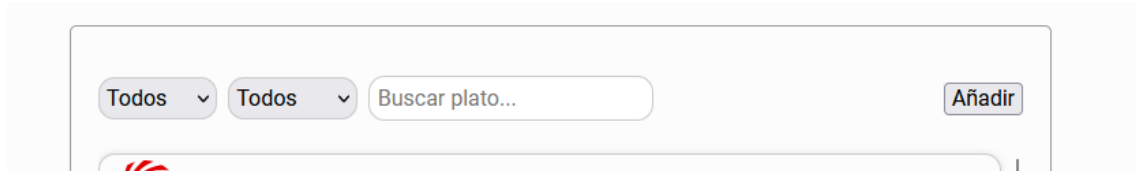
Carga el archivo que se introduce al input y establece el estado previewImage a esa imagen. Ese previewImage antes era null por lo que se mostraba un "Placeholder.svg" como se puede ver arriba.

Esta etiqueta imagen carga el previewImage si has subido un archivo, si esto no es así carga la imagen que tuviese antes ese producto y en caso de no tenerla carga Placeholder.svg.

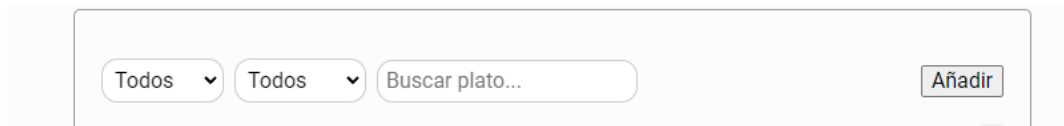
Pruebas

Se ha probado la aplicación en diferentes navegadores (Chrome, Firefox y Opera), las diferencias apercibidas son tan solo de estilos, en botones, formularios y otros ya que no han sido editados a fondo.

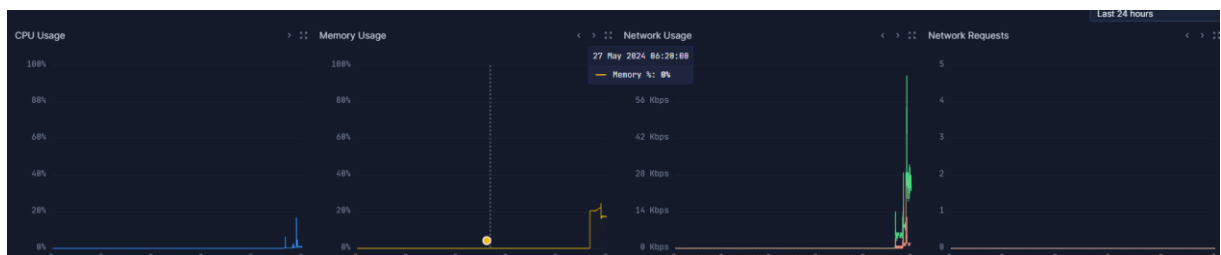
Firefox:



Chrome:



También he estado probando para ver el uso de recursos y las métricas que ofrece Northflank y se comporta bastante bien la aplicación.

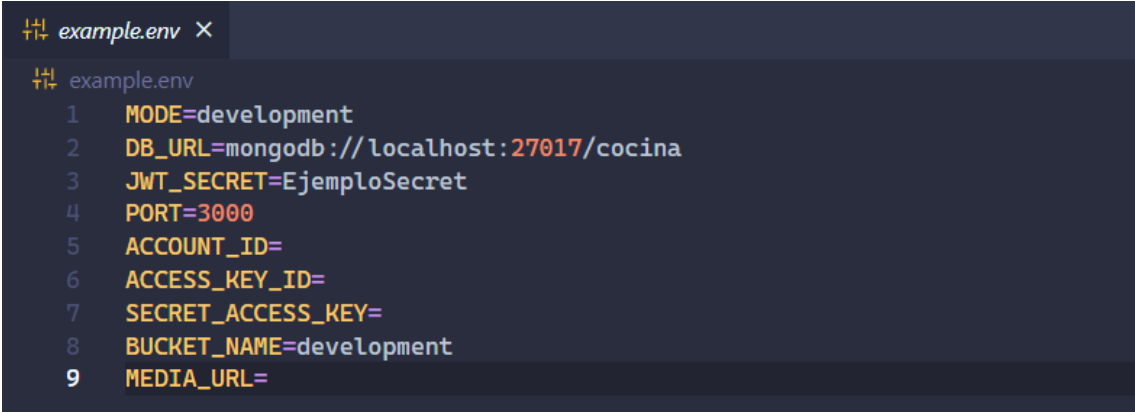


Solo he llegado a usar el 20% de CPU y Memoria.

Manuales de usuario

Instalación en local

1. Descargar los siguientes softwares para poder empezar:
 - Node.js (versión 20 o superior, 22 recomendada):
 - MongoDB: <https://www.mongodb.com/try/download/community>
 - Git: <https://git-scm.com/>
2. Descargar una copia del proyecto con el comando “git clone <https://github.com/FerZeg/TFG.git>”
3. Como recomendación abrir 3 pestañas diferentes en el Visual Studio Code ya que la parte del servidor y los 2 frontends están separados. Las carpetas son las siguientes en las que debes situarte:
 - a. \TFG\Código\Backend
 - b. \TFG\Código\Frontends\cocina
 - c. \TFG\Código\Frontends\cliente
4. Instalar las dependencias con el manejador de paquetes que viene con Node.js por defecto NPM (Node Package Manager), “npm install”. Se pueden usar otros más eficientes como pnpm que debes instalar antes usando este mismo. Esto debe hacerse en cada una de las carpetas al ser proyectos diferentes.
5. Configurar las variables de entorno necesarias. Hay un ejemplo de archivo .env en la carpeta del backend, duplicar el archivo y editar con los valores necesarios.



```
example.env
example.env
1  MODE=development
2  DB_URL=mongodb://localhost:27017/cocina
3  JWT_SECRET=EjemploSecret
4  PORT=3000
5  ACCOUNT_ID=
6  ACCESS_KEY_ID=
7  SECRET_ACCESS_KEY=
8  BUCKET_NAME=development
9  MEDIA_URL=
```

MODE: es el tipo de entorno en el que se está ejecutando, en este caso local.

DB_URL: es la dirección de acceso a la base de datos en mongodb, para facilitar la instalación usar una instancia en MongoDB Atlas, para facilitar la instalación y no tener que descargar e instalar MongoDB.

PORT: Puerto que escuchará el servidor, recomendable no cambiarlo para desarrollar en local.

ACCOUNT_ID, ACCESS_KEY_ID, SECRET_ACCESS_KEY y BUCKET_NAME, MEDIA_URL son claves relativas a la configuración para poder subir archivos. No son estrictamente necesarias para instalar en local ya que solo se usa en la parte de subida de imágenes. En la parte de instalación en la nube se desarrollan más.

En caso de querer instalar MongoDB en local descargar el software en <https://www.mongodb.com/try/download/community> e importante descargar también la Shell de MongoDB. <https://www.mongodb.com/try/download/shell>

El siguiente manual le ayudará a iniciar una instancia en modo replica <https://www.mongodb.com/docs/manual/tutorial/deploy-replica-set/>. Es totalmente obligatorio este paso si quieres instalar la base de datos en local para su correcto funcionamiento.

6. Iniciar los procesos que se vayan a utilizar. En la carpeta del proceso que quieras iniciar, por ejemplo, /Código/Backend, lanzar el comando “node –run dev” si estás en la versión 22 o “npm run dev”.

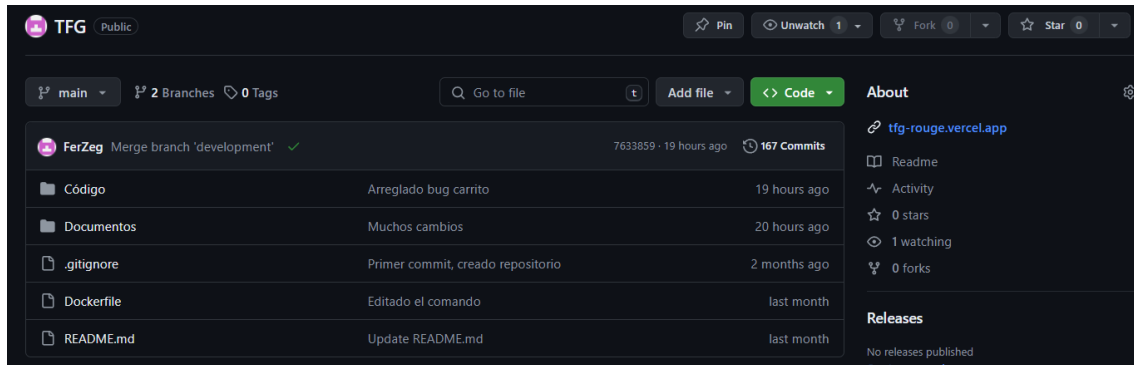
Si inicias los frontends, vite creará un servidor que servirá los archivos estáticos y refrescará cada vez que se guarde un cambio. El backend también escucha a todos los cambios por lo que no deberás refrescar la página, los cambios son instantáneos.

Instalación en la nube (versión recomendada)

General

Para instalarlo en la nube se recomiendan los siguientes servicios que han sido usados en el desarrollo de esta; Cloudflare, Northflank y MongoDB Atlas.

Como primer paso casi indispensable, hacer una copia (**fork**) en git para poder aprovechar al máximo el “Continuos Deployment” y todas las facilidades que se desarrollan en torno al repositorio.



El botón para hacer fork está en la página principal del repositorio.

Base de datos

MongoDB Atlas ofrece una capa gratuita, bastante conveniente para aplicaciones pequeñas o de pruebas.

1. Crear cuenta en <https://www.mongodb.com/cloud/atlas/register>. Puedes usar Google como vía rápida y fácil para el registro.
2. Directamente nos redirigirá a la página de creación de Clusters. Seleccionaremos la capa gratuita y daremos un nombre a nuestra BD.

Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

☒ M10

\$0.09/hour

For production applications with sophisticated workload requirements.

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

☐ Serverless

For application development and testing, or workloads with variable traffic.

STORAGE	RAM	vCPU
Up to 1 TB	Auto-scale	Auto-scale

☐ M0

Free

For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared

Pay-as-you-go! You will be billed hourly and can terminate your cluster anytime. Excludes variable data transfer, backup, and taxes.

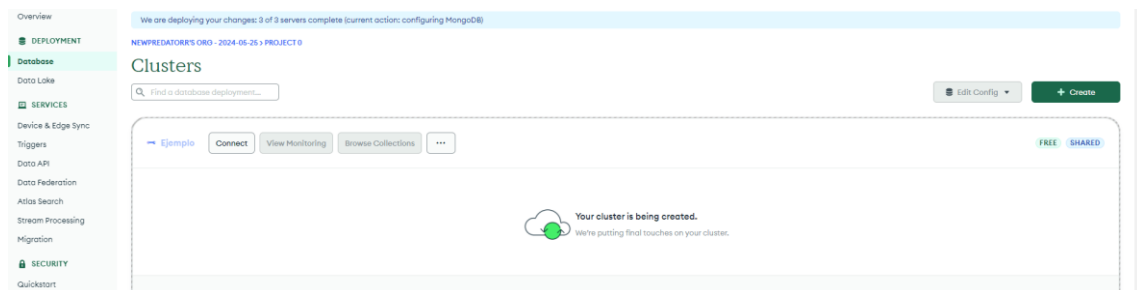
Name

You cannot change the name once the cluster is created.

☐ Preload sample dataset

Provider

3. Esperar unos segundos a la creación de la BD.



4. Una vez creada, te pedirá generar unas credenciales para poder administrar el sitio.

Connect to Ejemplo



You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

1. Add a connection IP address

✓ Your current IP address (84.123.163.60) has been added to enable local connectivity. Add another later in [Network Access](#).

2. Create a database user

This first user will have [atlasAdmin](#) permissions for this project.

ⓘ You'll need your database user's credentials in the next step. Copy the database user password.

Username	Password
<input type="text" value="ex. dbUser"/>	<input type="password" value="ex. dbUserPassword"/> HIDE Copy
<input type="button" value="Create Database User"/>	

5. Una vez generadas las credenciales, importante apuntarlas, dar a conectar aplicación. Realmente el modo de conexión es muy parecido en las diferentes opciones pero usaremos la opción de driver de NodeJS para conseguir la cadena de conexión.

3. Add your connection string into your application code

☐ View full code sample

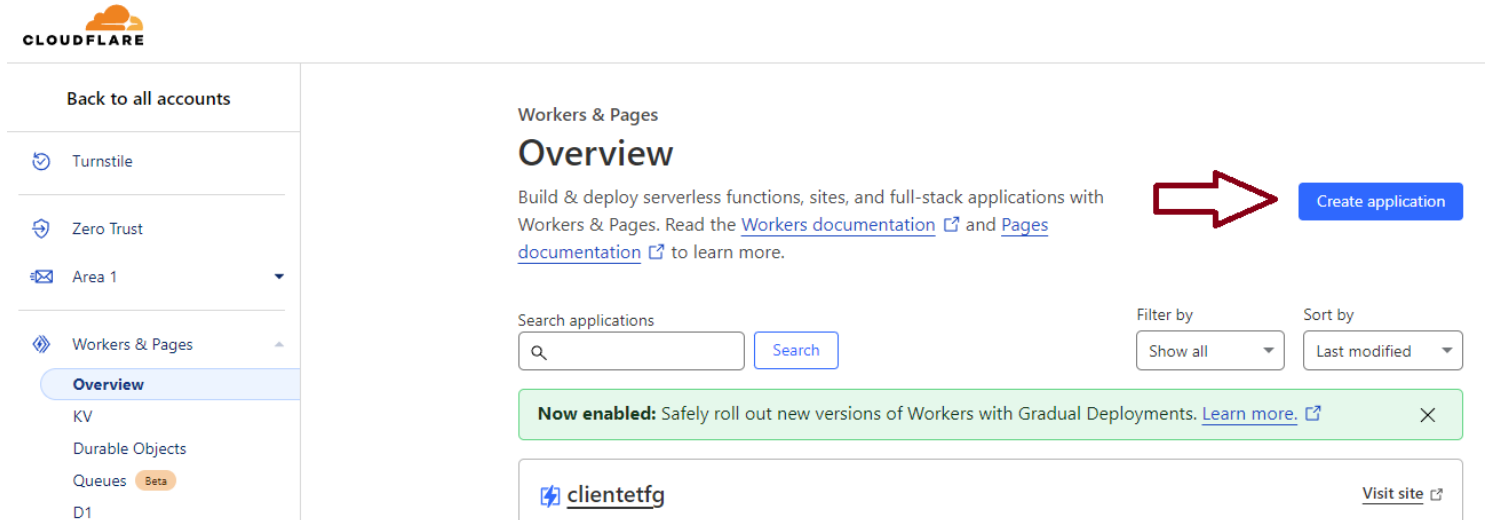
```
mongodb+srv://admin:<password>@ejemplo.wdbtuhu.mongodb.net/?
retryWrites=true&w=majority&appName=Ejemplo
```

6. Remplazar <password> por tu contraseña y añadirlo a las variables de entorno.

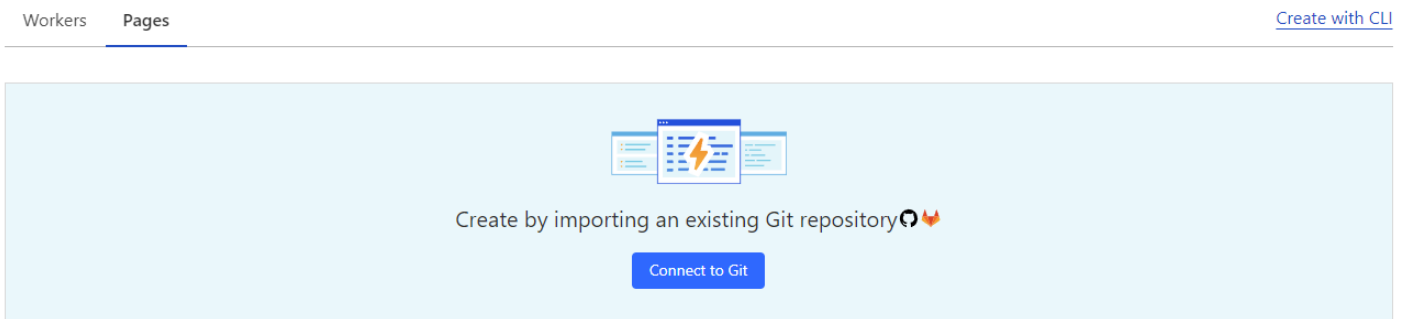
Puedes crear varias instancias de MongoDB para usarlo también en local y no tener que instalar la base de datos en tu ordenador. Esta opción es más fácil ya que no tendrás que iniciarla cada vez que quieras usarla y además es mucho más conveniente ya que se necesita que la BD esté en modo réplica para el uso de transacciones. La instancia local no está en modo réplica por defecto por lo que es un poco más complicado configurarla.

Frontends

1. Crear una cuenta en cloudflare <https://www.cloudflare.com/es-es/>
2. En la sección de workers y pages accionar el botón llamado “Create application”



3. En los apartados de creación habrá dos pestañas, Workers y Pages; seleccionar Pages.
4. Conectar con el repositorio de git donde almacenaste el fork.



Se pueden subir los archivos estáticos del frontend manualmente, pero es recomendable hacerlo de esta forma para que automáticamente actualice los cambios cuando haces un push a la rama principal.

5. Configurar el despliegue.

[← Create an application](#)

✓ Select repository — 2 Set up builds and deployments — 3 Deploy site

Set up builds and deployments

Configure automatic builds and deployments for FerZeg/TFG

Project name

Your project will be deployed to **tfg-2zt.pages.dev**.

Production branch

Pushes to this branch automatically trigger deployments to the Production environment. Pushes to all other branches will trigger deployments within the Preview environment.

Build settings

If your project uses a static site generator or build tool, set the build instructions for Cloudflare.

Framework preset

Select a framework to prefill recommended settings.

Build command ^①

e.g. npm run build

Build output directory ^①

/

e.g. dist

✓ Root directory (advanced)

Path

/

Your project's root directory, where Cloudflare runs the **build command**. If your site is not in a subdirectory, leave this path value empty.

Esta sería la configuración para el panel de administrador de la aplicación. Vite construye la aplicación en la carpeta /dist y la ruta donde esta esta parte es la siguiente /Código/Frontends/cocina.

6. Repetir los mismos pasos con el cliente. La ruta es /Código/Frontends/cocina
7. Configurar el dominio en ambas partes desde el apartado de Custom domains.

Workers & Pages / clientetfg

Deployments Functions metrics Custom domains Integrations Beta Settings Manage

Custom domains

Set up custom domains to point to your site. [Custom domains](#)

[Set up a custom domain](#)

cliente.comidaenmarcha.com

● Active

🔒 SSL enabled

⋮

Production

[Visit site](#)

🔄 Automatic deployments enabled

Domains: [clientetfg.pages.dev](#), [cliente.comidaenmarcha.com](#)

Production [main](#) [bd01c2e](#) 14730436.clientetfg.pages.dev

✓ a day ago [View details](#)

All deployments

Environment	Source	Deployment	Status	
Production	main bd01c2e Merge branch 'development'	14730436.clientetfg.pages.dev	✓ a day ago	View details ⋮
Preview	development d57e82a Beta	348b06c9.clientetfg.pages.dev	✓ a day ago	View details ⋮

En el panel de control de la página podremos ver los diferentes entornos, en este caso, producción en main y desarrollo en development

8. Es importante configurar la directiva **CORS** en el servidor dependiendo del dominio que hayas configurado

```
app.use(cors({
  origin: process.env.MODE == "development"
    ? ["http://localhost:5173", "http://localhost:5174"]
    : ["https://cliente.comidaenmarcha.com", "https://comidaenmarcha.com", "https://cocina.comidaenmarcha.com"]
}))
```

Configura los orígenes que tendrán permitidos el acceso. Esto es una restricción de seguridad de los navegadores para acceder a contenido externo a un origen.

Servidor

9. Aprovechando que estamos en Cloudflare, se recomienda a continuación la creación de un bucket en la nube para el almacenamiento de archivos mediante R2 de cloudflare.

← R2

Create a bucket

Get started by creating a new empty bucket. You'll be able to add data to your bucket using the dashboard or [Wrangler CLI](#).

Bucket name

ejemplobucket

✓

Bucket name is permanent

Location:

📁

Documentation

☒ Automatic

R2 automatically places your bucket in the closest available region based on your location

▼ Provide a location hint (optional)

Eastern Europe (EEUR)

▼

☐ Specify jurisdiction

R2 buckets can be restricted to a specific jurisdiction to meet data residency requirements. Locations within the specified jurisdiction will be automatically chosen.

By default buckets are not publicly accessible. You can access objects stored within your bucket by [binding the bucket](#) to a Worker or using the API. Bucket access can be changed to Public at any time.

Cancel

Create bucket

Bucket Details

Name: ejemplobucket
Location: Eastern Europe (EEUR)
Created: May 20, 2024
S3 API: [https://s3.amazonaws.com/ejemplobucket](#)

Public access

Specify how your bucket can be accessed. [Configuring bucket access](#)

Custom Domains

When a custom domain is connected to your bucket, the contents of your bucket will be made publicly accessible through that domain. Websites connected can also benefit from Cloudflare features such as bot management, Access, and Cache. [Learn more](#)

[Connect Domain](#)

There are no Cloudflare domains connected

Conectar un dominio (recomendable un subdominio tipo media.comidanemarcha.com) para poder distribuir los recursos públicamente.

10. Crear API token de R2 en el menú principal. Esta será la llave de acceso para poder subir archivos desde el servidor

[<-R2](#)

API Tokens

Grant access to all your buckets by generating an API token to authenticate your service. All API tokens for Cloudflare can be viewed from your [Profile](#).

Create API token

Token name	Applied to	Permission	Status
R2 Token	development, tfg	Object Read & Write	Active

Create API Token

Token name

R2 Token

Permissions

Specify the R2 Storage permission type for this token.

- ☐ **Admin Read & Write:** Allows the ability to create, list and delete buckets, edit bucket configurations, as well as list, write and read objects
- ☐ **Admin Read only:** Allows the ability to list buckets and view bucket configuration, as well as list and read objects.
- ☒ **Object Read & Write:** Allows the ability to read, write, and list objects in specific buckets.
- ☐ **Object Read only:** Allows the ability to read and list objects in specific buckets.

Permissions can be further customized or created for multiple accounts from your [Profile](#).

Specify bucket(s)

- ☒ Apply to all buckets in this account (including newly created buckets)
- ☐ Apply to specific buckets only

TTL

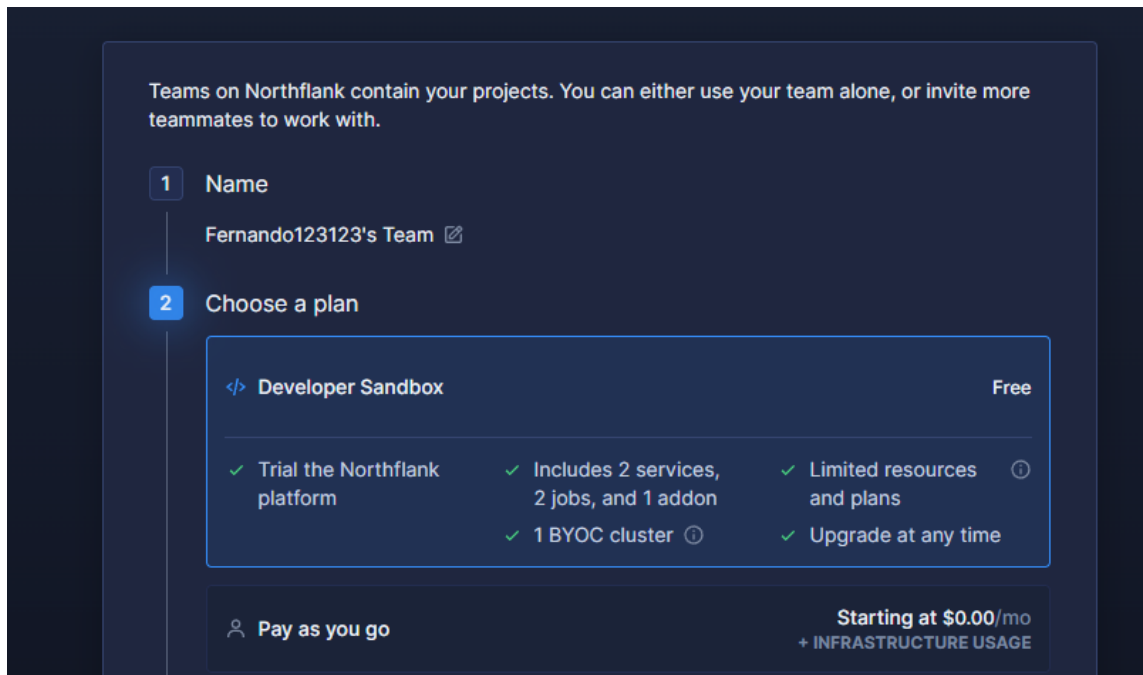
Define how long this token will stay active:

Forever

Puedes configurarlo para que el token tenga acceso a todos los buckets o solo a algunos en específico. Es importante otorgarle el permiso de escribir objetos para subir las imágenes. Estas credenciales, importante guardarlas, serán usadas en el próximo apartado para instalar el servidor.

Servidor

1. Crea una cuenta en <https://app.northflank.com/signup>, es tan fácil como usar tu cuenta de Google o Github directamente.
2. Elegir el plan developer, es totalmente gratis aunque tiene recursos limitados. La única pega de este servicio es que para evitar gente que abuse del plan gratuito te pedirá una tarjeta de crédito, aunque sea totalmente gratis.



3. Tras esta pantalla nos redirigirá a crear un proyecto directamente, y si no es así, deberás navegar hasta los proyectos y crear uno nuevo.
4. Tras crear un proyecto debes añadir un servicio. Configura el repositorio conectando tu cuenta de GitHub y selecciona en "Build options" la pestaña de "Dockerfile". Tras hacer esto se visualizará el archivo que ya está creado en el proyecto.

✓ **Build options**
Select whether your code will build using a Dockerfile or Buildpack

Build type * ⓘ

Dockerfile
Requires your project to have a Dockerfile, gives full control of build process

Buildpack [Heroku](#)
Detects project type and builds automatically, no Dockerfile required

> Advanced build settings

Build context ⓘ * Dockerfile location ⓘ *

Verify ⓘ If you wish to change the dockerfile location, please verify again

```

1 FROM node:21.7-alpine AS builder
2
3 RUN mkdir -p /app
4 WORKDIR /app
5
6 COPY /Código/Backend .
7
8 RUN npm install
9
10 EXPOSE 80
  
```

5. Tendrás que configurar las variables de entorno para tu entorno de producción

MODE	*****
JWT_SECRET	*****
DB_URL	*****
PORT	*****
ACCOUNT_ID	*****
ACCESS_KEY_ID	*****
SECRET_ACCESS_KEY	*****
BUCKET_NAME	*****
MEDIA_URL	*****

6. En el servicio tienes que configurar el dominio personalizado también

Ports & DNS

80 HTTP

p01

api.comidaenmarcha.com



Port *

Protocol *

80

HTTP

☒ Publicly expose this port to the internet

Name *

p01

Port will be accessible at

Public api.comidaenmarcha.com

Private backend:80

Custom domains & security rules (optional)

Custom domains ?

[Manage domains](#)

api.comidaenmarcha.com

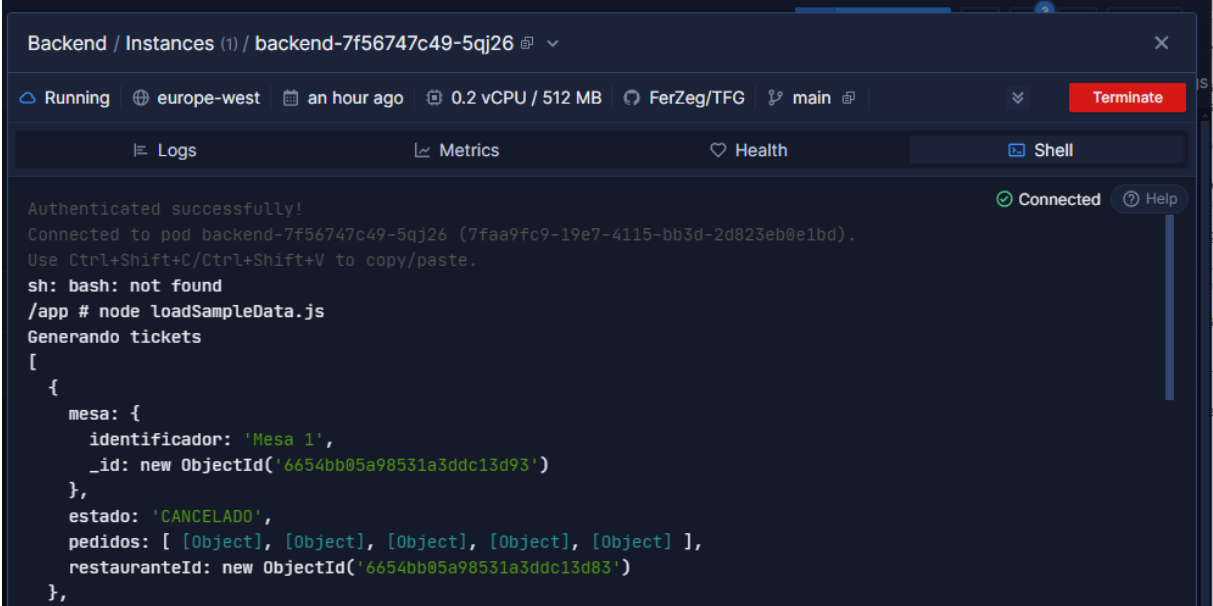
[Add custom domain](#)

☒ Disable default code.run domain

Cargar datos de ejemplo

Para cargar datos de ejemplo debes usar el script “loadSample” mediante el comando “node –run loadSample”. Esto es en el caso del entorno de desarrollo, si necesitas cargar datos de ejemplo en el hosting deberás tan solo ejecutar el archivo mediante “node loadSampleData.js”.

Este script generará un restaurante con registros: mesas; usuarios; pedidos; tickets; etc....



The screenshot shows a terminal window titled "Backend / Instances (1) / backend-7f56747c49-5qj26". The window has tabs for "Running", "europa-west", "an hour ago", "0.2 vCPU / 512 MB", "FerZeg/TFG", "main", and a "Terminate" button. Below the tabs are tabs for "Logs", "Metrics", "Health", and "Shell". The "Shell" tab is active, showing a terminal session. The terminal output is as follows:

```
Authenticated successfully!
Connected to pod backend-7f56747c49-5qj26 (7faa9fc9-19e7-4115-bb3d-2d823eb0e1bd).
Use Ctrl+Shift+C/Ctrl+Shift+V to copy/paste.
sh: bash: not found
/app # node loadSampleData.js
Generando tickets
[
  {
    mesa: {
      identificador: 'Mesa 1',
      _id: new ObjectId('6654bb05a98531a3ddc13d93')
    },
    estado: 'CANCELADO',
    pedidos: [ [Object], [Object], [Object], [Object], [Object] ],
    restauranteId: new ObjectId('6654bb05a98531a3ddc13d83')
  },
```

Northflank incluye una terminal para cada contenedor, puedes ejecutar el comando desde ahí.

Uso

El flujo de trabajo que tiene la aplicación actualmente es el siguiente:

Trabajadores:

- Configurar la aplicación para poder iniciar. Se necesita una cuenta de administrador para configurar todos los apartados.
- Iniciar sesión en las mesas para poder ver el menú de inicio.
- La parte de la cocina permite ver en tiempo real los pedidos entrantes, los cocineros pueden verlos, marcar como hechos un número definido o incluso cancelarlos.

Clientes:

- Un cliente entra en el establecimiento y se sienta en una mesa.
- Habrá una pantalla que te dará la bienvenida y le tendrás que pulsar comenzar.
- El cliente añade los productos que quiere al carrito de la compra y hace un pedido. El mismo cliente puede hacer varios pedidos durante su estancia.
- El cliente puede ver todo lo que ha pedido en el apartado de historial de pedidos.
- Una vez terminada la comida, el cliente pulsará terminar y el ticket se cerrará.

Conclusiones, correcciones y ampliaciones

Conclusiones

El objetivo principal de este trabajo era presentar una aplicación para la consecución del grado superior de desarrollo de aplicaciones. Sin embargo, he intentado igualar en importancia, el máximo aprendizaje posible. Ya tenía ciertos conocimientos dentro del stack de tecnología usado, sin embargo, he afianzado aquellas bases que tenía y siento que he mejorado bastante en cuanto a la calidad del código.

Este proyecto me ha permitido mejorar diversas habilidades, como leer mejor la documentación, planificar de manera más eficaz, y adoptar buenas prácticas, como diseñar previamente para contar con una referencia clara, tanto visual como lógica.

Ampliaciones

Se podrían introducir traducciones mediante la librería i18n para poder cambiar el idioma.

Poder editar los tickets a fondo, pedidos / precio.

Códigos de descuento al de pedir. (Sistema de fidelización).

Informes y reportes del estado del negocio.

- Suma de tickets, agrupación por mes/año/día o incluso horas.
- Reportes estadísticos por producto como el más o menos comprado.

Añadir descripciones a los productos.

Sistema de paginación en lista de registros (tickets).

Apartado para camareros, donde se distingan las mesas que han hecho los pedidos que ya están listos.

Correcciones

Validar mejor los datos, con alguna librería dedicada a ello.

Dar libertad para elegir las categorías de los diferentes artículos.

Mejorar la sintaxis del código unificándolo al inglés todo. (Ahora hay una pequeña mezcla en determinadas secciones).







Mejorar los estilos, no está ni cerca de un resultado final para poder llevarlo a producción.

Mejor reutilización del código.

El responsive debe mejorar y mucho.

Relación de ficheros en formato digital

General

 .git	27/05/2024 0:06	Carpeta de archivos	
 Código	08/04/2024 22:08	Carpeta de archivos	
 Documentos	27/05/2024 15:43	Carpeta de archivos	
 .gitignore	21/03/2024 8:59	Archivo de origen ...	1 KB
 Dockerfile	11/04/2024 16:51	Archivo	1 KB
 README.md	14/04/2024 0:06	Archivo de origen ...	1 KB

Este es el directorio principal de mi repositorio.



Dockerfile es un archivo necesario para desplegar el servidor de la aplicación.

.gitignore es un archivo para manejar los archivos que no se suben al repositorio.

README.md es un archivo para el repositorio de GitHub para dar una descripción.



>> *Documentos*

En esta carpeta hay variedad de archivos, desde la propia memoria hasta otros archivos adjuntos. Modelo datos contienen unos archivos JSON con ejemplos para poder visualizar los datos que contiene la base de datos.

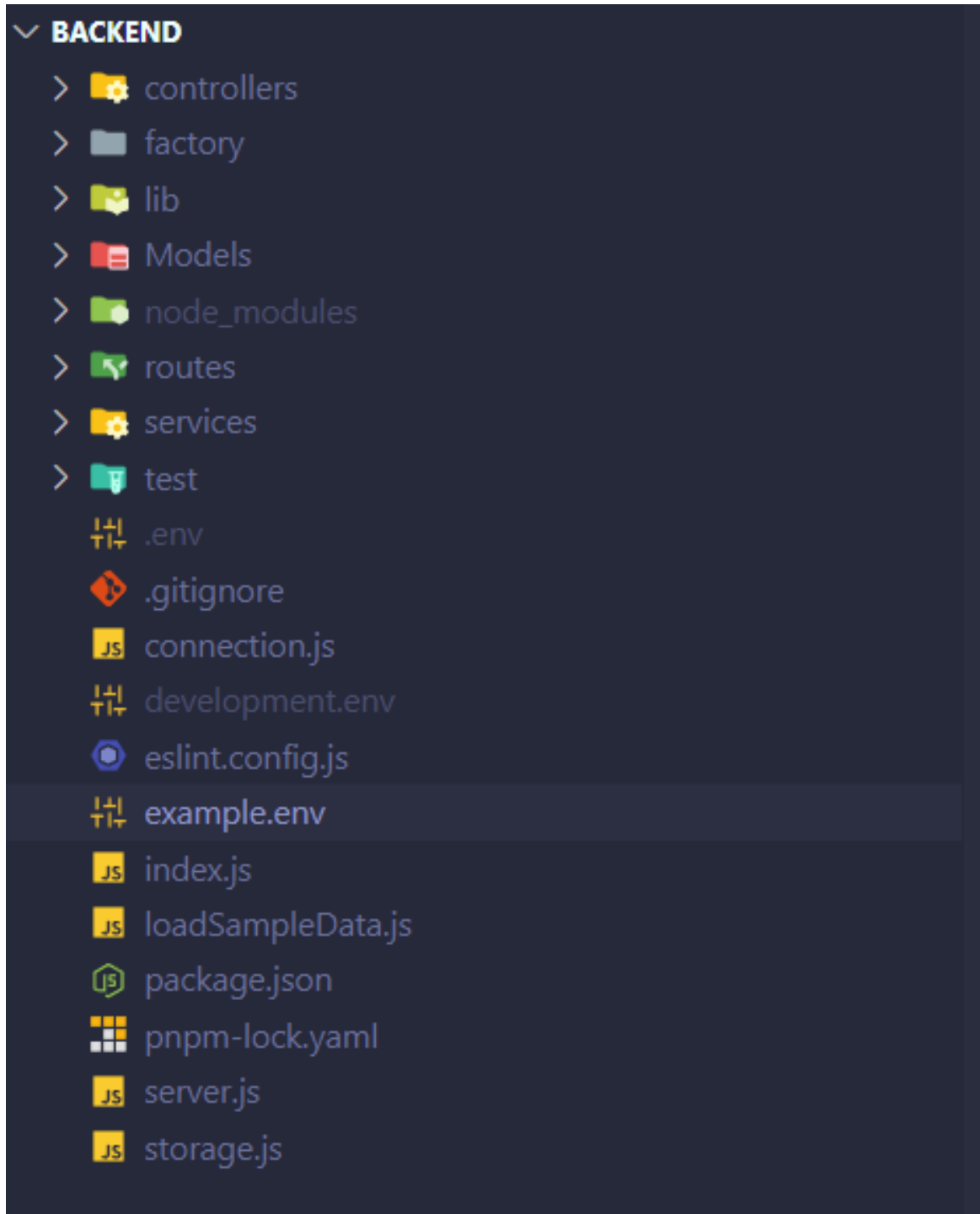
 Modelo datos	26/05/2024 22:49	Carpeta de archivos	
 Memoria.docx	27/05/2024 15:38	Documento de Mi...	4.676 KB

>> *Código*

En la carpeta Código se encuentran las carpetas que contienen el código de la aplicación divididas por backend y frontends.

 Backend	26/05/2024 22:49	Carpeta de archivos	
 Frontends	12/05/2024 20:11	Carpeta de archivos	

>> Backend



El punto de inicio de la aplicación es index.js.

La carpeta controllers contiene los diferentes middlewares que forman la aplicación.

Factory es una carpeta que contiene junto a loadSampleData.js el código para generar datos de ejemplo.

Lib es una carpeta donde hay código de uso general.

Models contiene los **Schemas** de mongoose para estandarizar el modelo de datos de la aplicación.

node_modules es una carpeta común a todos los proyectos con nodejs. Todas las librerías que descargas se guardan ahí.

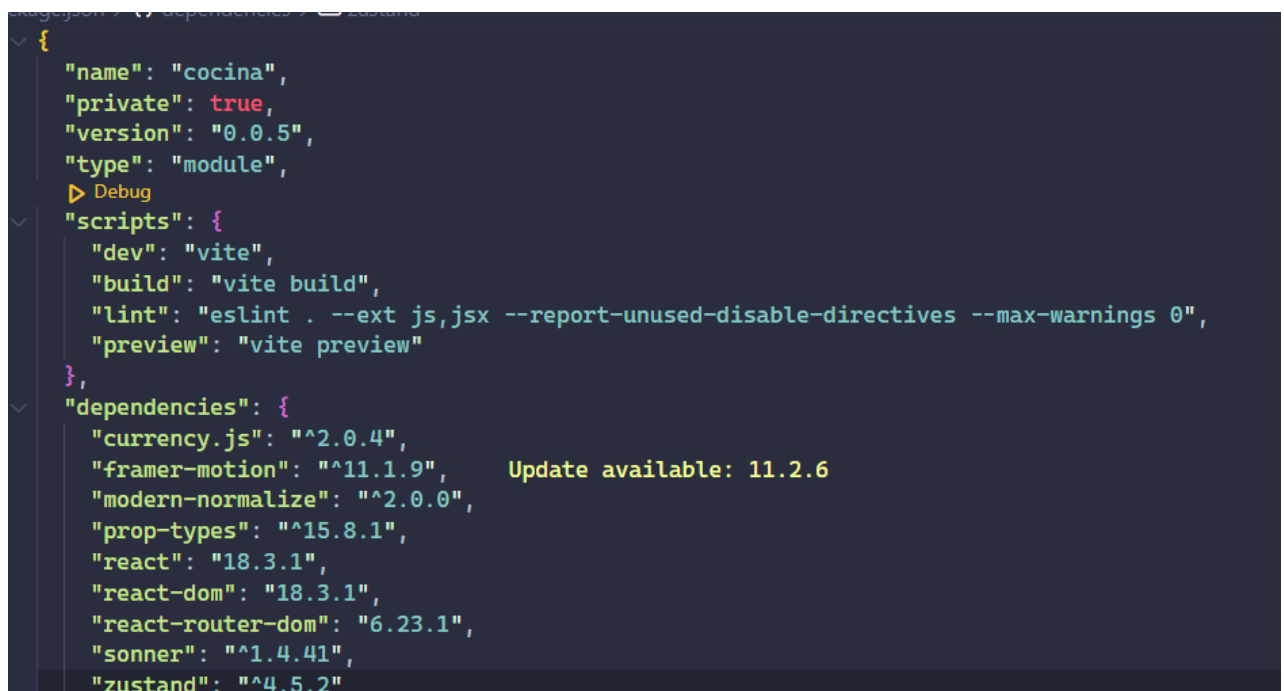
Routes tiene los diferentes routers mencionados anteriormente en la aplicación. Estos definen las rutas de la aplicación.

Services: contienen los códigos encargados de interactuar con la base de datos.

Test: contienen códigos de test. No están actualizados por lo que probablemente fallen.

Los archivos que tienen extensión .env definen las variables de entorno.

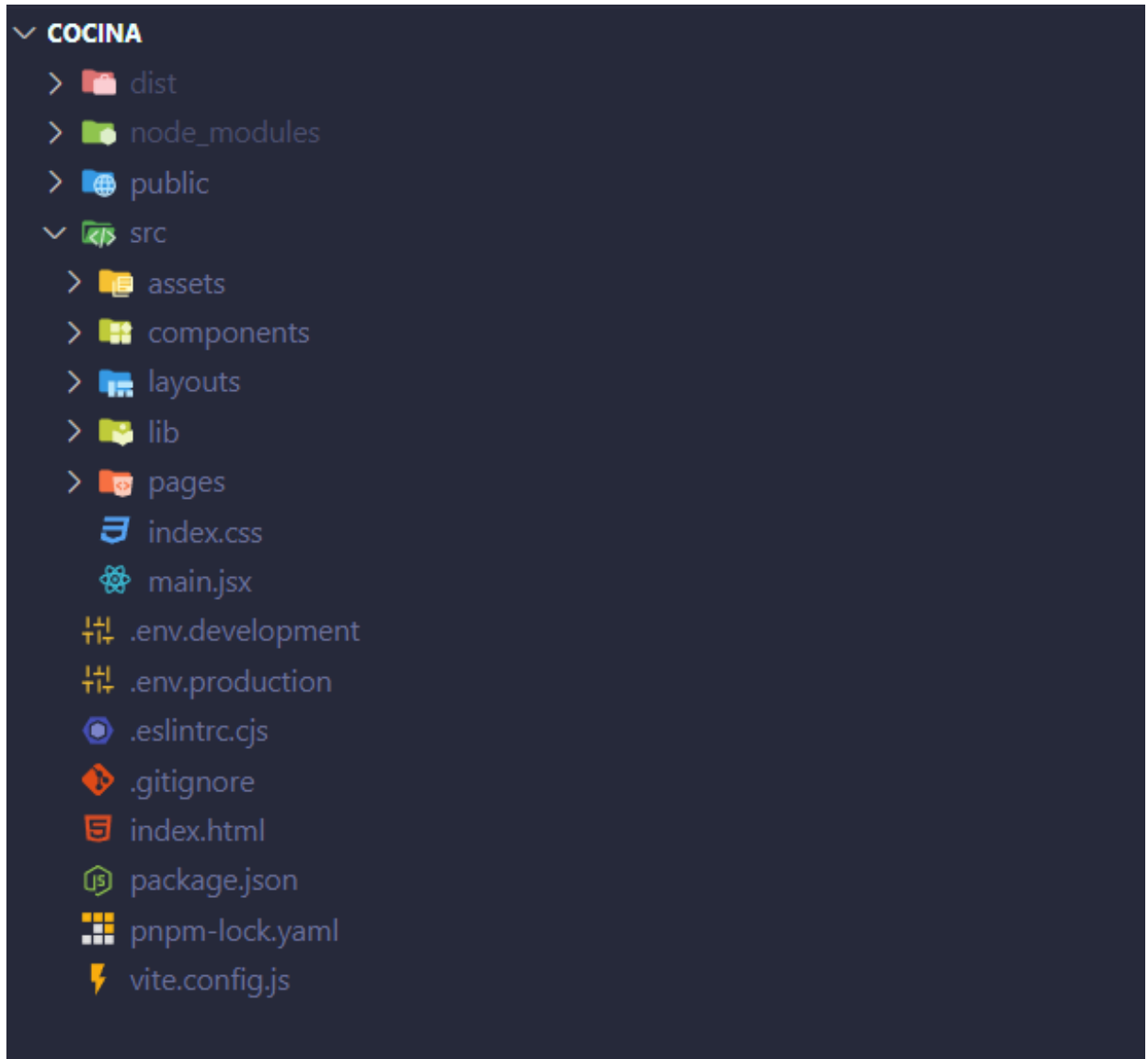
Package.json contiene información sobre las dependencias y la aplicación.



```
{
  "name": "cocina",
  "private": true,
  "version": "0.0.5",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "currency.js": "^2.0.4",
    "framer-motion": "^11.1.9",
    "modern-normalize": "^2.0.0",
    "prop-types": "^15.8.1",
    "react": "18.3.1",
    "react-dom": "18.3.1",
    "react-router-dom": "6.23.1",
    "sonner": "^1.4.41",
    "zustand": "^4.5.2"
  }
}
```

>> Frontends

Como ambos frontends son parecidos solo pongo uno de ejemplo.



dist: es la carpeta que contiene la compilación resultante del proyecto.

public: son archivos que se incluirán en la compilación pero que no necesitan estar dentro del código para ser importados.

layouts: son contenedores de la aplicación que tienen lógica añadida, por ejemplo, rutas con autenticación o código que se repite como el navegador.

lib: contiene funciones y el contexto de la aplicación, (estados globales).

pages: son componentes que actúan como páginas, estas contienen otros muchos componentes.

main.jsx: punto de entrada de la aplicación.

`.env`: archivos de variables de entorno.

`Index.html`: es el archivo que se sirve por defecto a todas las rutas.

`Vite.config.js`: archivo de configuración de la librería de vite.

Bibliografía

Modelar los datos con MongoDB:

- <https://www.youtube.com/watch?v=YsaOcUDUJKY>
- <https://www.youtube.com/watch?v=Hidk36H6hBY>

Información sobre metodología incremental:

- https://www.reddit.com/user/BackNativos22/comments/taiqv2/describir_el_modelo_y_sus_principales/

Algunos iconos (libre uso con atribución):

- <https://www.svgrepo.com/>

Librerías de NodeJS:

- <https://www.npmjs.com/>

Glosario

- **CDN (Content Delivery Network):** Red de servidores interconectados que contienen copias locales de contenidos para distribuirlos rápidamente desde el nodo más cercano al usuario.
- **CD (Continuos Deployment):** Es una estrategia por la cual se publican automáticamente los cambios hechos a producción (producto final).
- **SPA (Single Page Application):** Aplicación web que interactúa con el usuario repintando la aplicación. Esto consigue crear una aplicación que no cambia de página (no se refresca). Se ahorran peticiones en el servidor y se gana velocidad de interacción al utilizarlas.
- **Fork:** Copia completa del repositorio de la que tienes control completo.
- **CORS:** mecanismo de seguridad que permite que los recursos de una página web sean solicitados desde otro dominio diferente al que originó la página.
- **Middleware:** Agente intermedio que se coloca entre diferentes componentes de una aplicación.
- **Schema:** Definición de la estructura de un modelo de datos.