Documentación reto tecnico

Este documento presenta un resumen de las decisiones que se tomaron para realizar el proyecto

Consideraciones a tener en cuenta

De acuerdo a las limitaciones impuestas de utilizar node puro se debe tener en cuenta

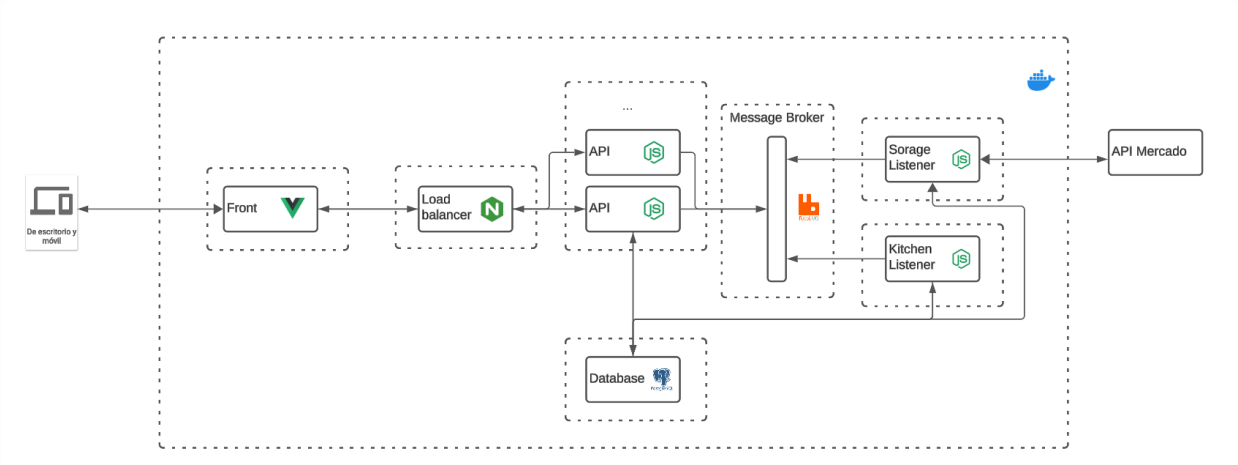
- **No se hizo uso de frameworks/librerias** como express o NestJS para el backend
- **No se hizo uso de ORMs** como Prisma, Squalize etc.
- Las implementaciones se realizaron al menor nivel posible y solo se utilizaron paquetes necesarios como: libreria **manejo de conexion a DB**, **manejo de JWT** y **manejo de conexion a rabbitMQ**

Sin enbargo debido a la **arquitectura hexagonal** del proyecto es posible implementar cualquiera de estas funcionalidades de manera sencilla como se explica mas adelante

Usuario de pruebas frontend

email: jhon@example.com
password: abcd1234
url: https://restaurantapp.juanandresdeveloper.com/login

Arquitectura del proyecto



El proyecto se compone de siete servicios, todos implementados con **Docker** y orquestados mediante **Docker Compose**. Los servicios son:

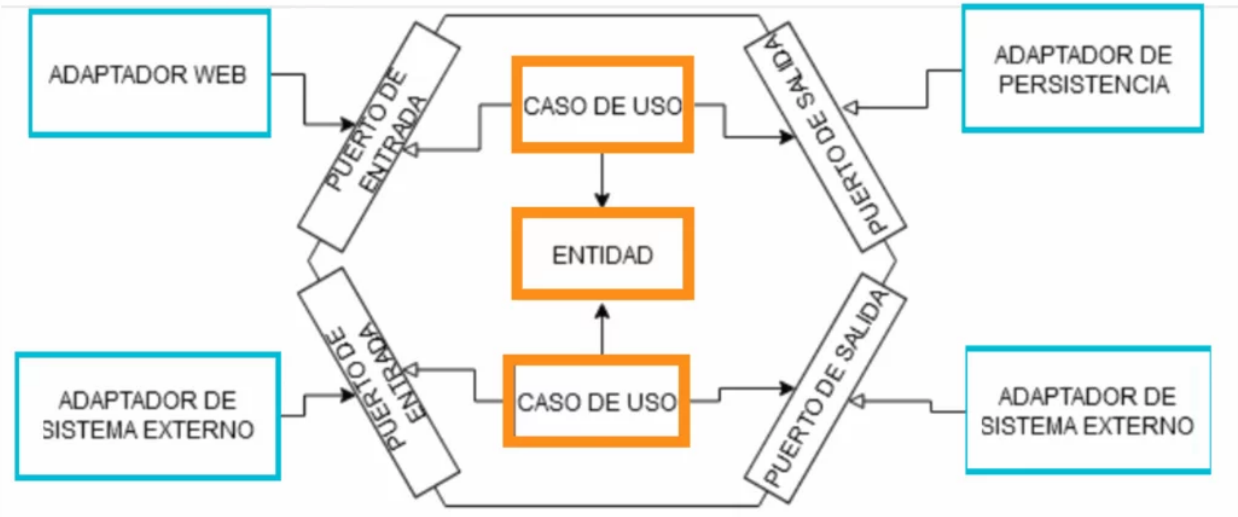
- Frontend
- Balanceador de Carga
- API Backend
- Microservicio de Almacenamiento
- Microservicio de Cocina
- Broker de Mensajería
- Base de Datos

La arquitectura es de microservicios basada en eventos. La API principal funciona como backend for frontend la cual se encarga de gestionar todas las solicitudes del frontend. Esta API genera eventos que se envían al broker de mensajería, el cual utiliza un sistema de **publicación/suscripción (pub/sub)** para notificar a los microservicios pertinentes.

Además, el backend for frontend incorpora un balanceador de carga, lo que permite escalar este servicio según sea necesario, asegurando **alta disponibilidad** y un **alto nivel de procesamiento** en momentos de alta demanda.

Los proyectos de backend estan basados en una **arquitectura hexagonal** la cual permite que el proyecto sea mas escalable a largo plazo permitiendo asi encapsular logica en 3 capas

- infraestructura
- aplicacion
- dominio



En donde toda la logica de negocio esta contenida en la capa de aplicacion y dominio y las configuraciones externas en la capa de infrestructura, esto permite poder realizar cambios o migraciones de tecnologias reduciendo el esfuerzo y la complejidas del cambio.

Por ejemplo si se deseara cambiar la implementacion de la api rest a express, solo es necesario ajustar el archivo de implementacion principal y el adaptador de entrada de la api, lo mismop con la base de datos la cual solo implica el cambio de la conexion y los repositorios.

Base de datos

Despliegue

El proyecto utiliza las siguientes tecnologias

- NodeJS
- Postgres
- VueJS

- RabbitMQ
- Nginx

Presenta la siguiente configuracion

```
.
└── project/
    ├── backend/
    │   ├── api
    │   ├── storages
    │   └── kitchen
    └── frontend/
        └── restaurant
```

En la raiz del project existe un archivo **docker-compose.yml** el cual permite levantar todo el proyecto mediante el comando

```
docker-compose up --build
```

> **En docker no es necesario configurar variables de entorno**
>
> Las variables de entorno estan configuradas para que el proyecto funcione sin realizar configuraciones extras, estas variables son **.env.docker**

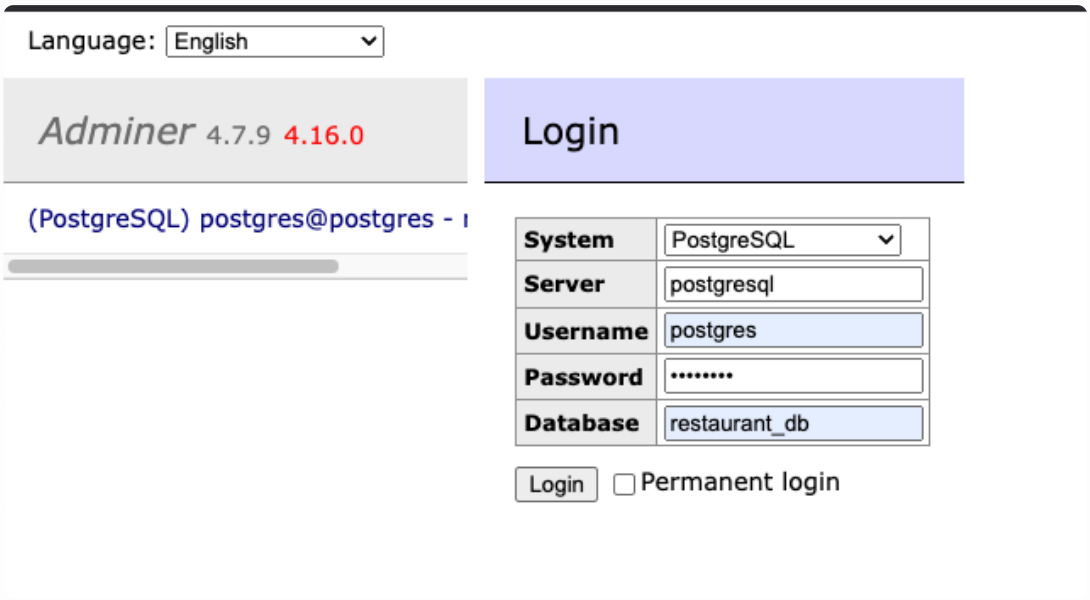
Para uso individual de los proyecto se debe revisar la variable **.env.example**

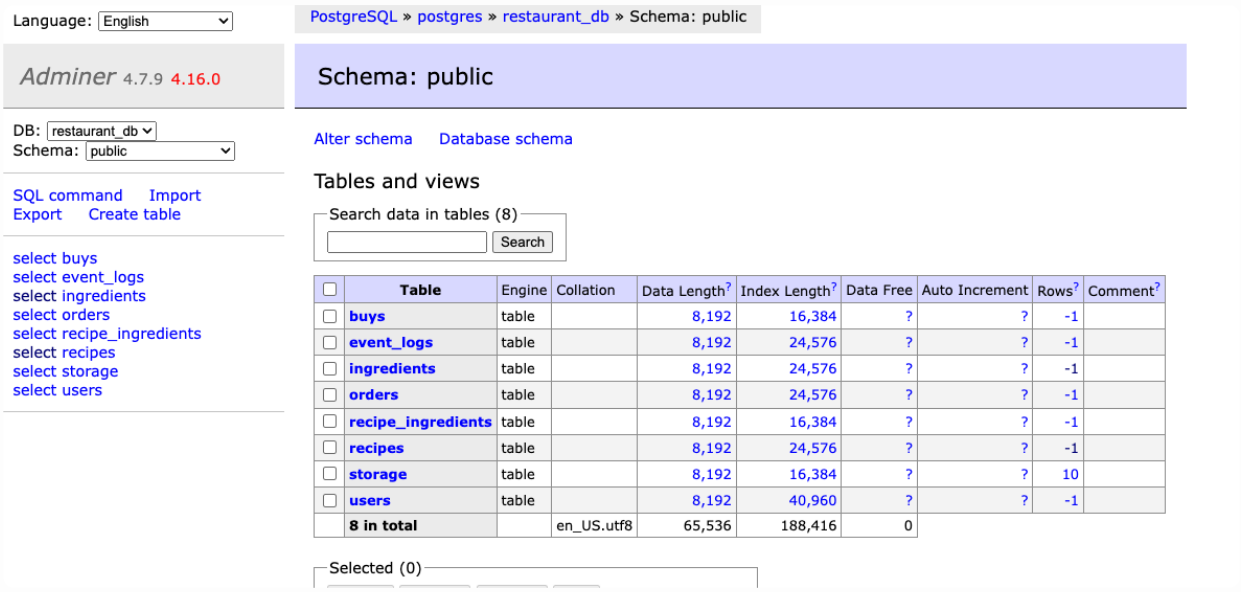
Servicios adicionales

Conexión con base de datos

Para realizar la conexión con la base de datos se dispone del servicio Adminer el cual se puede acceder desde `http://localhost:4001` mediante las siguientes credenciales por defecto

- **server**: postgres
- **user**: postgres
- **password**: postgres
- **database**: restaurant_db

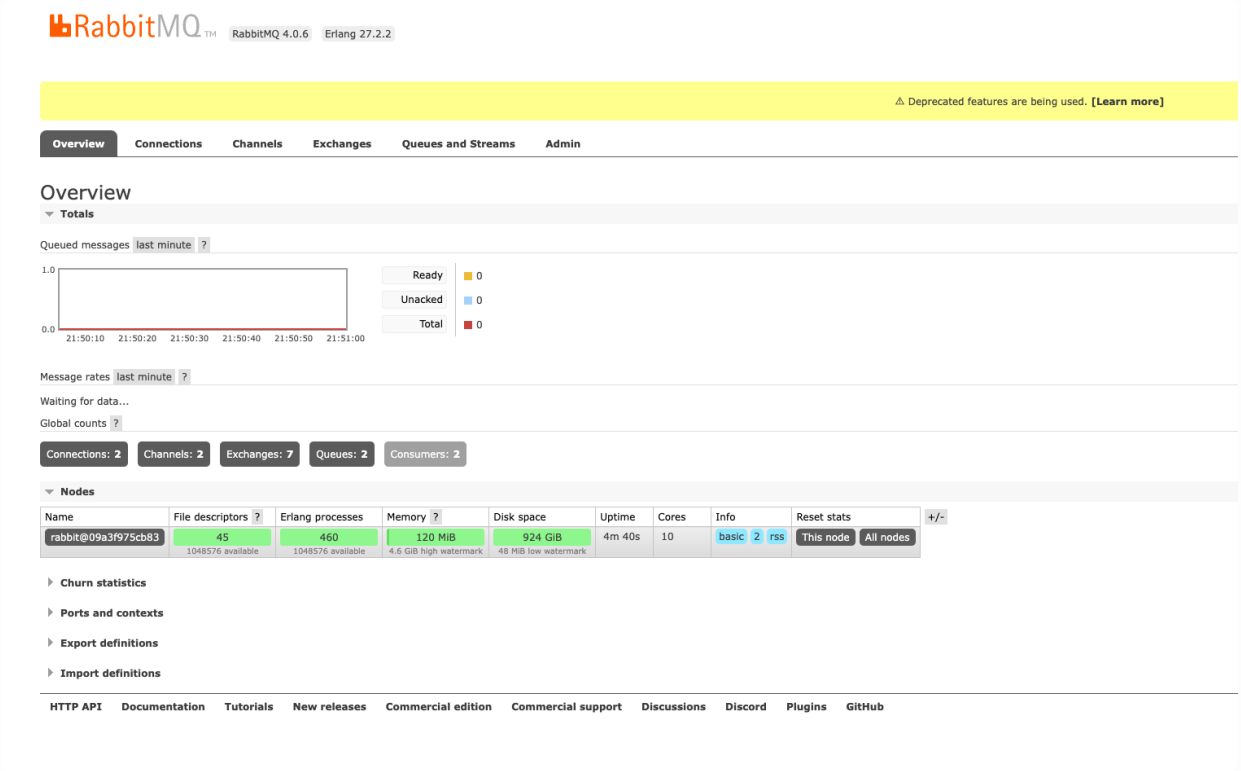




Conexion Admin RabbitMQ

Para entrar al panel de administración de rabbitmq se puede acceder desde `http://localhost:15671` con las credenciales

- **username**: guest
- **password**: guest

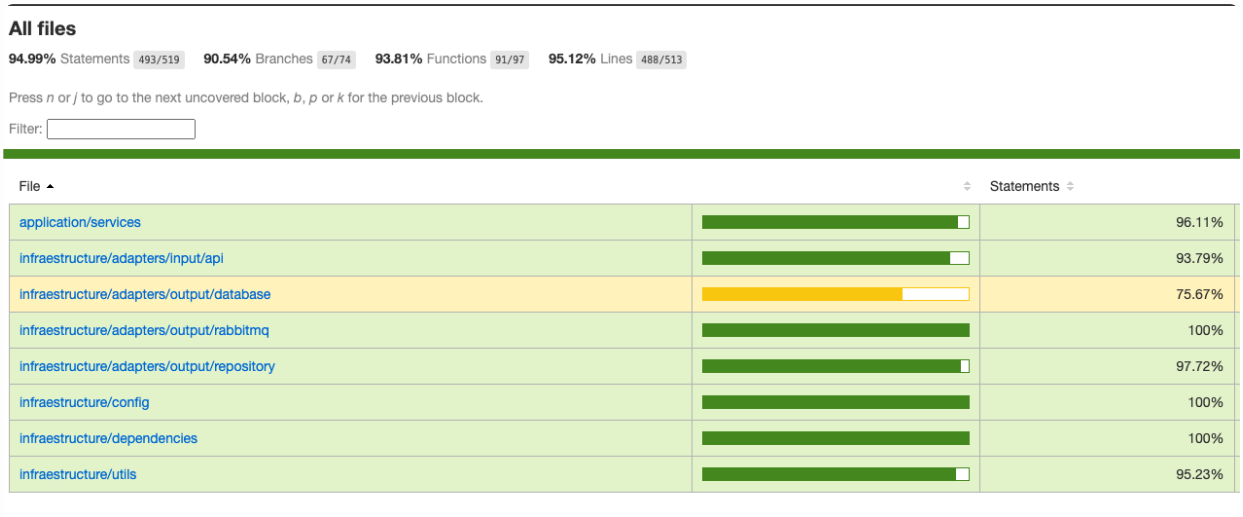


> Estos servicios se pueden editar directamente el el archivo **docker-compose.yml**

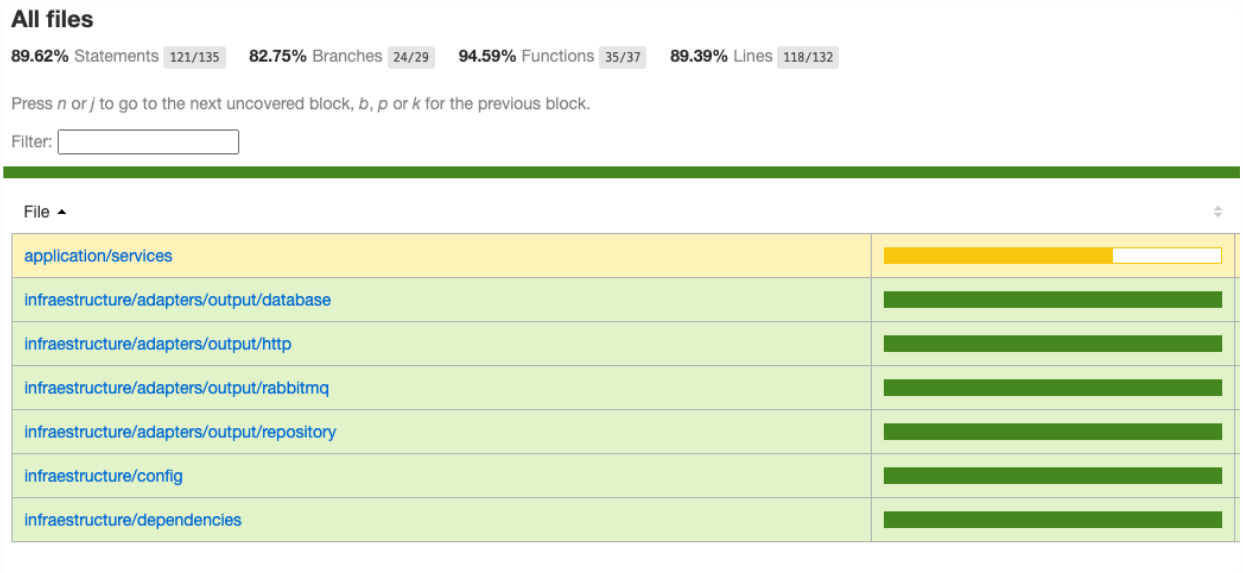
Pruebas unitarias

Las pruebas unitarias se diseñaron para cumplir un requisito de mas del **90%**, el cual se puede lograr gracias a la arquitectura del proyecto

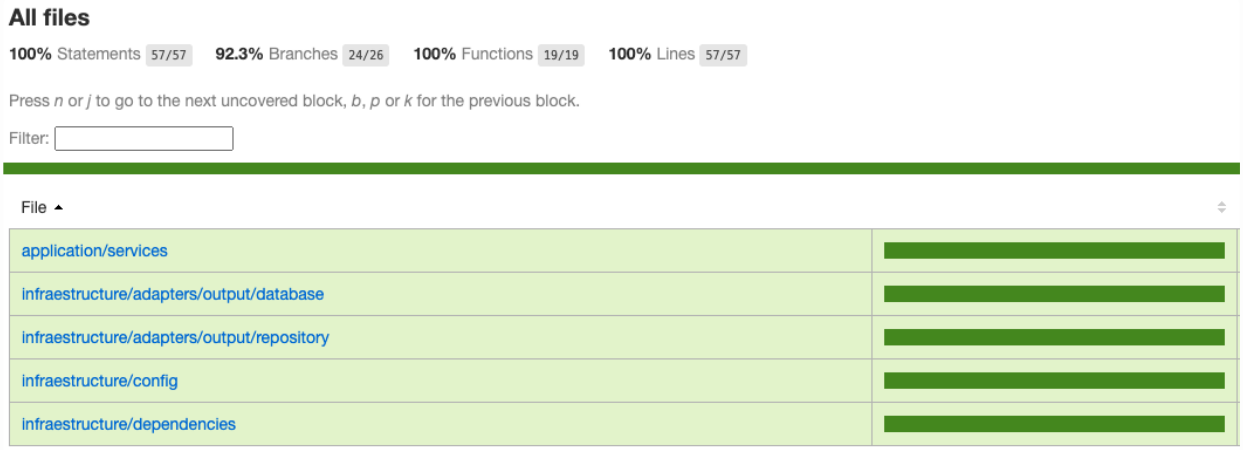
API



Storage Microservice



Kitchen



Documentación

Health Check

Endpoint para verificar el estado del servicio

GET `{{host}}/`

Response

```
{
    "status": "ok"
}
```

Login

Endpoint para el inicio de sesion del usuario, retorna un JWT con un tiempo de expiracion de **1 hora**

POST `{{host}}/api/v1/login`

Body

```
{
    "email": "jhon@example.com",
    "password": "abcd1234"
}
```

Response

```
{
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsInVzZXJFb
}
```

Crear cuenta

Endpoint para la creación de cuentas de usuario

POST `{{host}}/api/v1/signin`

Body

```
{
    "name": "string",
    "email": "string",
    "password": "string",
    "confirmPassword": "string"
}
```

Response

```
{
    "id": int,
    "name": "string",
    "email": "string",
    "password": "string",
    "createdAt": "string"
}
```

Listar todas las ordenes

Endpoint que lista todas las ordenes en base de datos, por defecto esta paginado

GET `{{host}}/api/v1/orders`

Headers

```
Authorizarion: Bearer {{token}}
```

Params

```
page: int
perPage: int
orderId: int
orderName: string
```

Response

```
{
    "data": [
        {
            "id": "int",
            "recipe": {
                "id": "int",
                "name": "string"
            },
            "status": "string",
            "createdAt": "string"
        },
    ],
    "page": "int",
    "perPage": "int",
    "total": "int"
}
```

Crear orden

Endpoint que permite generar una orden

POST {{host}}/api/v1/orders

Headers

```
Authorizarion: Bearer {{token}}
```

Response

```
{
    "id": "int",
    "recipe": {
        "id": "int",
        "name": "string"
    },
    "status": "string",
    "createdAt": "string"
}
}
```

Listar todos los ingredientes

Endpoint que permite listar todos los ingredientes y sus cantidades disponibles

GET {{host}}/api/v1/ingredients

Headers

```
Authorizarion: Bearer {{token}}
```

Params

```
page: int
perPage: int
orderId: int
orderName: string
```

Response

```
{
    "data": [
        {
            "id": 2,
            "name": "lemon",
            "quantity": 1,
            "imageUrl": "/lemon_min.webp"
        }
    ],
    "page": 1,
```

```
    "perPage": 10,
    "total": 10
}
```


Listar todas las compras realizadas en el mercado

Endpoint que permite obtener el historico de todas las compras en la plaza de mercado

GET `{{host}}/api/v1/buys`

Headers

```
Authorizarion: Bearer {{token}}
```

Params

```
page: int
perPage: int
orderId: int
orderName: string
```

Response


```
{
    "data": [
        {
            "id": 1,
            "ingredient": {
                "id": 1,
                "name": "tomato"
            },
            "quantity": 4,
            "createdAt": "2025-02-19T16:17:18.997Z"
        }
    ],
    "page": 1,
    "perPage": 10,
    "total": 1
}
```


Listar todas las recetas

Endpoint que permite ontener el listado de recetas disponibles junto con sus ingredientes y cantidades

GET `{{host}}/api/v1/recipe`

Headers

```
Authorizarion: Bearer {{token}}
```

Params

```
page: int
perPage: int
orderId: int
orderName: string
```

Response


```
{
    "data": [
        {
            "id": 1,
            "name": "Cheesy Chicken and Rice Casserole",
            "imageUrl": "/cheesy_chicken_and_rice_casserole_min.webp",
            "ingredients": [
                {
                    "id": 10,
                    "name": "chicken",
                    "quantity": 1,
                    "image_url": "/chicken_min.webp"
                }
            ]
        }
    ],
    "page": 1,
    "perPage": 10,
    "total": 6
}
```