

# BACKEND ENGINEER TEST

**Luis Alejandro Bravo Ferreira**

RESTful API con autenticación, búsqueda de restaurantes y registro de transacciones.  
Arquitectura Hexagonal con NestJS, PostgreSQL y JWT. Uso de buenas prácticas.





# Arquitectura

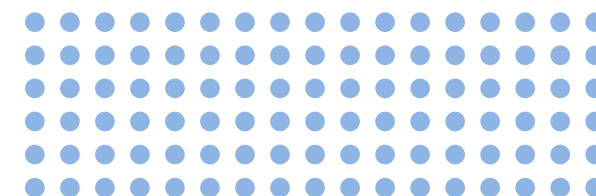
La solución sigue una arquitectura hexagonal que separa el dominio del resto del sistema, mejorando mantenibilidad, testeo y flexibilidad.

## Hexagonal Architecture (Clean Architecture)

- Aisla la lógica de negocio de las dependencias externas.
- Capas: Domain, Application, Infrastructure, Interfaces.
- Patrón: Ports & Adapters.

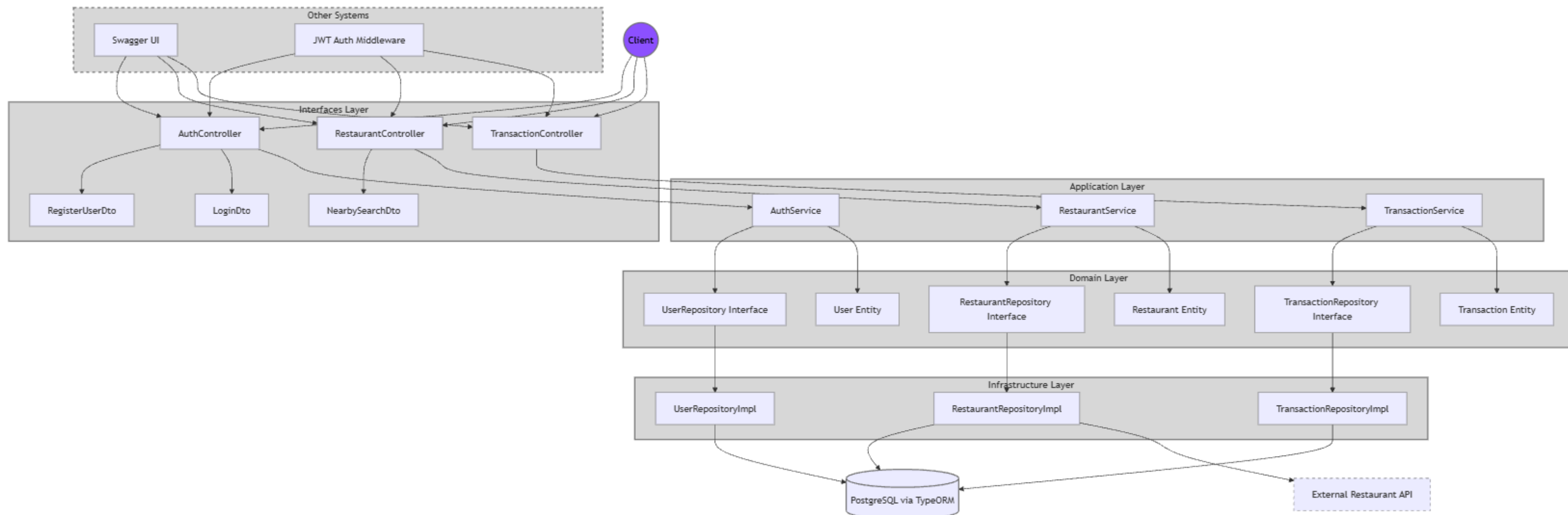
## Ventajas

- Independencia tecnológica
- Alta testabilidad y escalabilidad
- Adaptabilidad al cambio
- Separación de responsabilidades clara



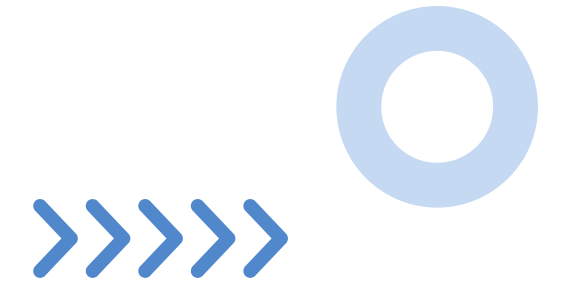
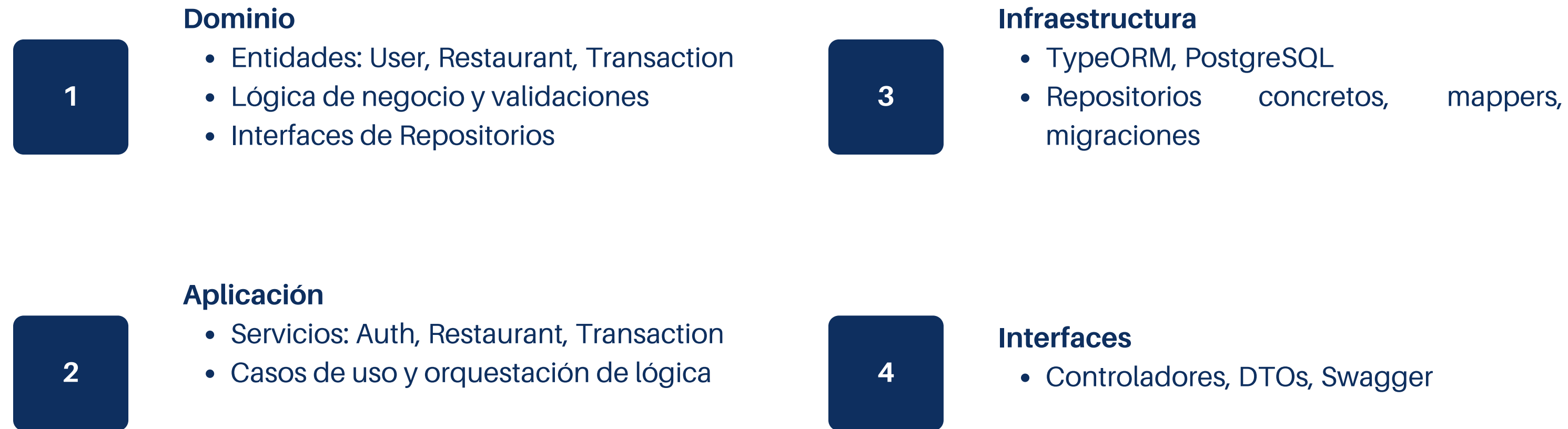
# Diseño de la arquitectura (Vista de desarrollo)

A continuación se presenta un diagrama que describe la arquitectura del proyecto:



# Capas del proyecto

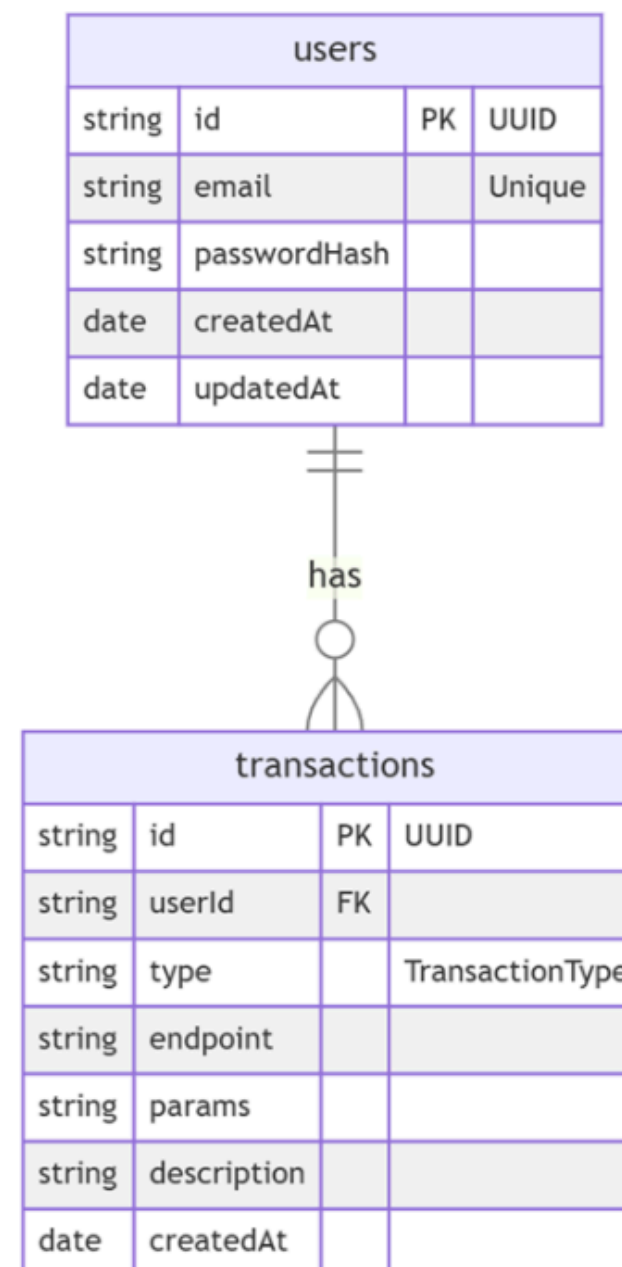
La aplicación está estructurada en capas siguiendo el enfoque de Arquitectura Limpia, lo que permite una separación clara de responsabilidades, facilita las pruebas y mejora la mantenibilidad. Cada capa cumple un rol específico y se comunica solo con las capas adyacentes:





# Diseño de la base de datos (Modelo relacional)

A continuación se presenta un diagrama que describe el modelo relacional creado:

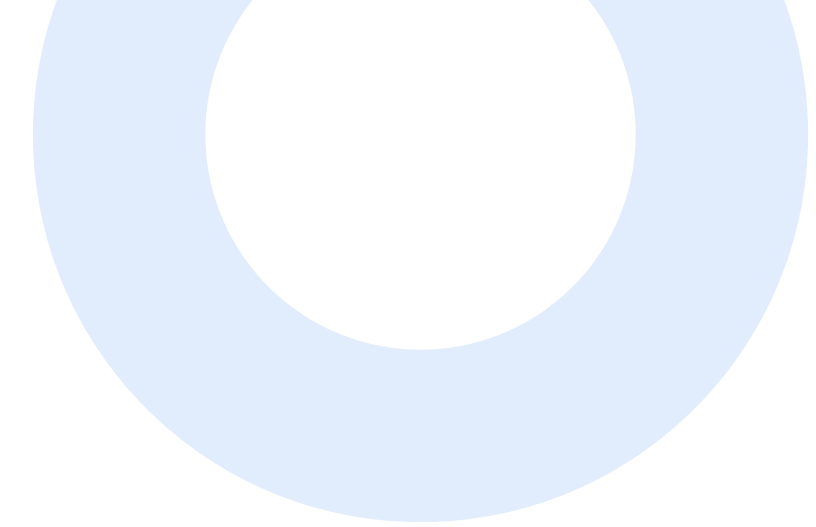


# Entidades principales

Las entidades de base de datos están desacopladas del dominio para preservar la pureza del modelo.

Como ORM, se utilizó TypeORM.

Las entidades creadas son las siguientes:



## UserEntity

UUID, email único, contraseñas  
hasheadas, etc

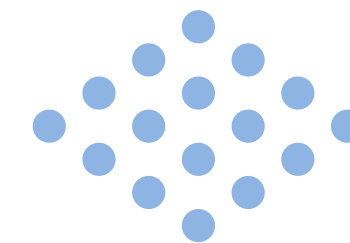
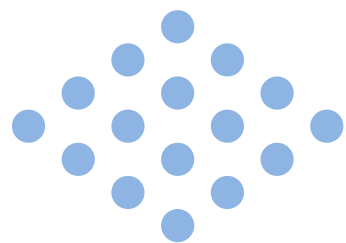


## TransactionEntity

Registro de solicitudes API con detalles

**\*Hay una separación clara  
entre ORM y dominio**





# Autenticación y Seguridad

## JWT Authentication

- Registro, login, logout
- Tokens almacenados en cliente
- Lista negra en memoria (Redis recomendado si se quiere expandir aún más)

## Seguridad

- Contraseñas hasheadas (bcrypt)
- Guardas y estrategias con Passport-JWT



## Configuración - Uso de variables de entorno

Archivo `.env` con la siguiente estructura:

```
# App
PORT=3000
NODE_ENV=development

# Base de datos
DATABASE_HOST=localhost # o "db" en Docker
DATABASE_PORT=5432
DATABASE_USER=your-username
DATABASE_PASSWORD=your-password
DATABASE_NAME=defaultdb
DATABASE_SSL_MODE=prefer

# Autenticación
JWT_SECRET=your-secret-key
JWT_EXPIRATION=1h
```

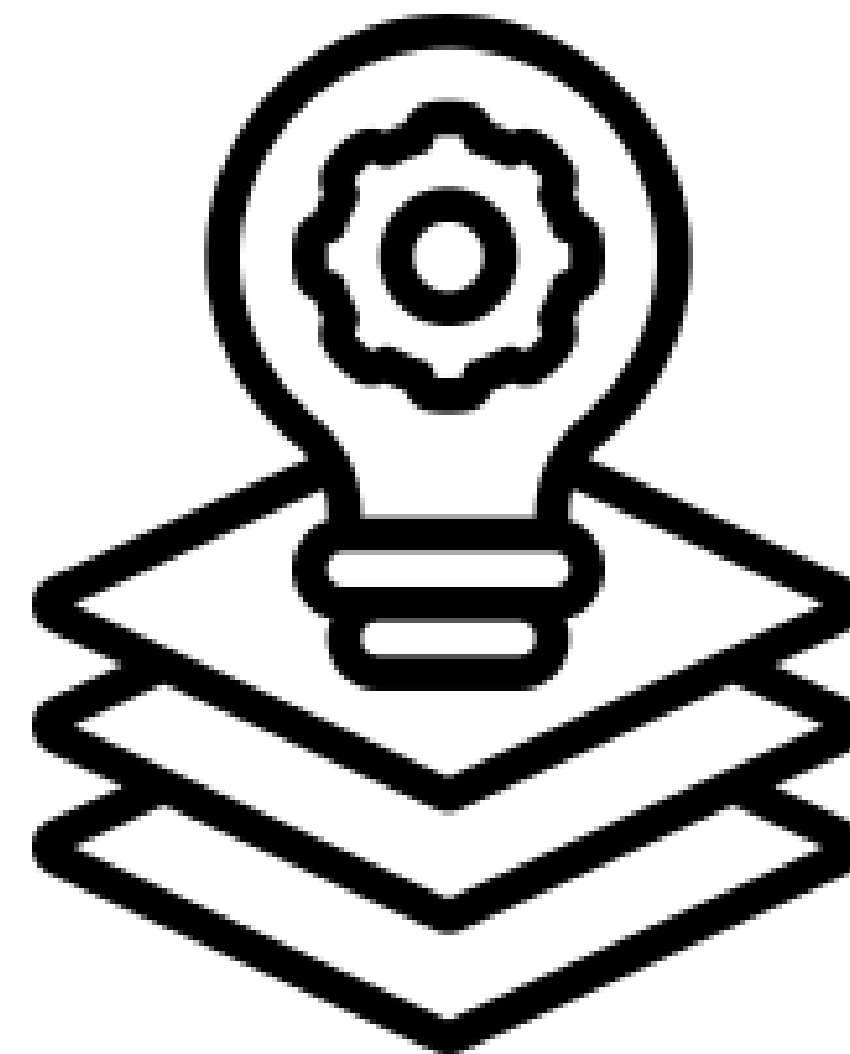


# Stack Tecnológico

Tecnologías modernas que aseguran calidad, mantenibilidad y despliegue eficiente.

- **Framework:** NestJS (Node.js)
- **Lenguaje:** TypeScript
- **Base de datos:** PostgreSQL
- **ORM:** TypeORM
- **Auth:** JWT
- **Docs:** Swagger
- **Contenedores:** Docker + Docker Compose
- **Tests:** Jest

Otros: Bcrypt para hash, class-validator para realizar validaciones, etc. ***Para el servicio externo de Restaurantes, se utilizó Overpass***





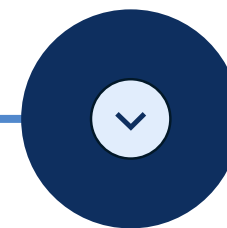
# Funcionalidades (Endpoints)

A continuación se presentan los endpoints REST disponibles en la aplicación, organizados por funcionalidad. Cada grupo de rutas se encuentra protegido por autenticación JWT cuando es necesario y documentado con Swagger para facilitar su uso.



## Auth Endpoints

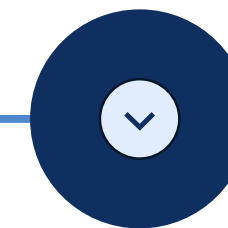
- **POST** /auth/register – Registro de nuevo usuario
- **POST** /auth/login – Inicio de sesión
- **POST** /auth/logout – Cierre de sesión
- **GET** /auth/profile – Perfil del usuario autenticado



## Restaurants Endpoints

- **GET** /restaurants: Buscar restaurantes por latitud/longitud o nombre de ciudad

Usa la API de Overpass Turbo




## Transactions Endpoints

- **GET** /transactions: Obtener el historial de transacciones del usuario actual


# Swagger

Se utilizó Swagger para documentar los endpoints y realizar pruebas fácilmente. A continuación se presentan los endpoints documentados:



 **Swagger**  
Supported by SMARTBEAR

**Tyba Backend API** 1.0 OAS 3.0  
The Tyba Backend API documentation

Authorize 

App

GET /

auth Authentication endpoints

POST /auth/login Login a user and get JWT token

POST /auth/logout Logout a user and invalidate token

GET /auth/profile Get authenticated user profile

POST /auth/register Register a new user

restaurants Restaurant search endpoints

GET /restaurants Find restaurants near a location

transactions Transaction history endpoints

GET /transactions Get transaction history for the authenticated user

Puede encontrar la documentación de endpoints en <http://localhost:3000/api-docs>

# Auth Endpoints: 1

POST /auth/register – Registro de nuevo usuario



Documentación General

POST

/auth/register

Register a new user

Try it out

Parameters

No parameters

Request body required

application/json

Example Value | Schema

```
{  "email": "user@example.com",  "password": "Password123!",  "passwordConfirmation": "Password123!"}
```

Responses

Code	Description	Links
201	User has been successfully registered <div>Media type<div>application/json</div><div>Controls Accept header.</div></div> <div>Example Value   Schema</div> <div><pre>{  "success": true,  "message": "User registered successfully",  "data": {    "id": "550e8400-e29b-41d4-a716-446655440000",    "email": "user@example.com",    "createdAt": "2023-07-24T12:34:56.789Z",    "updatedAt": "2023-07-24T12:34:56.789Z"  } }</pre></div>	No links
400	Validation error or user already exists	No links

# Auth Endpoints: 1

POST /auth/register – Registro de nuevo usuario



*Prueba particular*

POST /auth/register Register a new user

Parameters

Cancel

Reset

No parameters

Request body required

application/json

```
{  "email": "user1@example.com",  "password": "Password123!",  "passwordConfirmation": "Password123!"}
```

Execute

Clear

Responses

Curl

```
curl -X 'POST' \  'http://localhost:3000/auth/register' \  -H 'accept: application/json' \  -H 'Content-Type: application/json' \  -d '{  "email": "user1@example.com",  "password": "Password123!",  "passwordConfirmation": "Password123!"}'
```

Request URL

http://localhost:3000/auth/register

Server response

Code

Details

201

Response body

```
{  "success": true,  "message": "User registered successfully",  "data": {    "id": "44f156a1-f4fb-4131-9266-e17cecfd5a7c",    "email": "user1@example.com",    "createdAt": "2025-05-29T09:02:36.557Z",    "updatedAt": "2025-05-29T09:02:36.557Z"  }}
```

Download

# Auth Endpoints: 2

Documentación General

POST /auth/login - Inicio de sesión



POST /auth/login Login a user and get JWT token

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

```
{  "email": "user@example.com",  "password": "Password123!"}
```

Responses

Code	Description	Links
200	User has been successfully logged in <div><div>Media type<div>application/json</div></div><div>Controls Accept header.</div><div>Example Value   Schema<pre>{  "success": true,  "message": "Login successful",  "data": {    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",    "user": {      "id": "550e8400-e29b-41d4-a716-446655440000",      "email": "user@example.com",      "createdAt": "2023-07-24T12:34:56.789Z",      "updatedAt": "2023-07-24T12:34:56.789Z"    }  }}</pre></div></div>	No links
401	Invalid credentials	No links



### Prueba particular

POST

/auth/login Login a user and get JWT token

Parameters

No parameters

Request body required

application/json

```
{  
  "email": "user1@example.com",  
  "password": "Password123!"  
}
```

Execute

Clear

Responses

Curl

```
curl -X 'POST' \  
  'http://localhost:3000/auth/login' \  
  -H 'accept: application/json' \  
  -H 'Content-Type: application/json' \  
  -d '{  
    "email": "user1@example.com",  
    "password": "Password123!"  
  }'
```

Request URL

```
http://localhost:3000/auth/login
```

Server response

Code

Details

200

Response body

```
{  
  "success": true,  
  "message": "Login successful",  
  "data": {  
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI0NGYxNTZlMmNMGzIltQxdEtoTl2Ni1MTDdjZWVhbnZMcjllbmFpbCI6InVzZXIxQG9VYVwlbWGUyY29tIiwiaWF0IjoxNzQ0NTA5NDYLCjleHAiOjE3NDg1MTAwMDJ9.ijk71Qls5URv0ge8zu1EfIJovh4zc8otba2q2qByajU",  
    "user": {  
      "id": "44f156a1-f4fb-4131-9266-e17cecf5a7c",  
      "email": "user1@example.com",  
      "createdAt": "2025-05-29T09:02:36.557Z",  
      "updatedAt": "2025-05-29T09:02:36.557Z"  
    }  
  }  
}
```

Download

# Auth Endpoints: 3

POST /auth/logout – Cierre de sesión





*Documentación General*

POST

/auth/logout

Logout a user and invalidate token

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	User has been successfully logged out <div>Media type<div>application/json</div><div>Controls Accept header.</div><div>Example Value   Schema<div><pre>{  "success": true,  "message": "Logout successful"}</pre></div></div></div>	No links
401	Invalid or missing token	No links

*\*Se necesita un token JWT*

# Auth Endpoints: 3

POST /auth/logout – Cierre de sesión



*Prueba particular*

POST /auth/logout Logout a user and invalidate token

Parameters

Cancel

No parameters

ExecuteClear

Responses

Curl

```
curl -X 'POST' \
'http://localhost:3000/auth/logout' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ3dWI0I0NGYxNTZhMS1mNGZlTQxMzEtOTI2Ni1lMTdjZWNmZDVhN2MiLCJlbWFPbCI6InVzZXI6QGV4YW1wbGUy29tIiwiaWF0IjoxNzQ4NTA5NDg0LCJleHAiOiJE3NDg1MTMwODR
```

Request URL

http://localhost:3000/auth/logout

Server response

Code	Details
200	<div>Response body<pre>{   "success": true,   "message": "Logout successful" }</pre></div> <div>Download</div>

*\*Se necesita un token JWT*



# Auth Endpoints: 4

GET /auth/profile – Perfil del usuario autenticado




*Documentación General*

GET

/auth/profile

Get authenticated user profile

 ^

Parameters

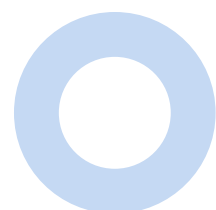
Cancel

No parameters

Execute

Responses

Code	Description	Links
200	<div>User profile has been successfully retrieved</div> <div>Media type<div>application/json</div></div> <div>Controls Accept header.</div> <div>Example Value   Schema<div><pre>{  "success": true,  "message": "Profile retrieved successfully",  "data": {    "id": "550e8400-e29b-41d4-a716-446655440000",    "email": "user@example.com"  }  }</pre></div></div>	No links
401	Invalid or missing token	No links



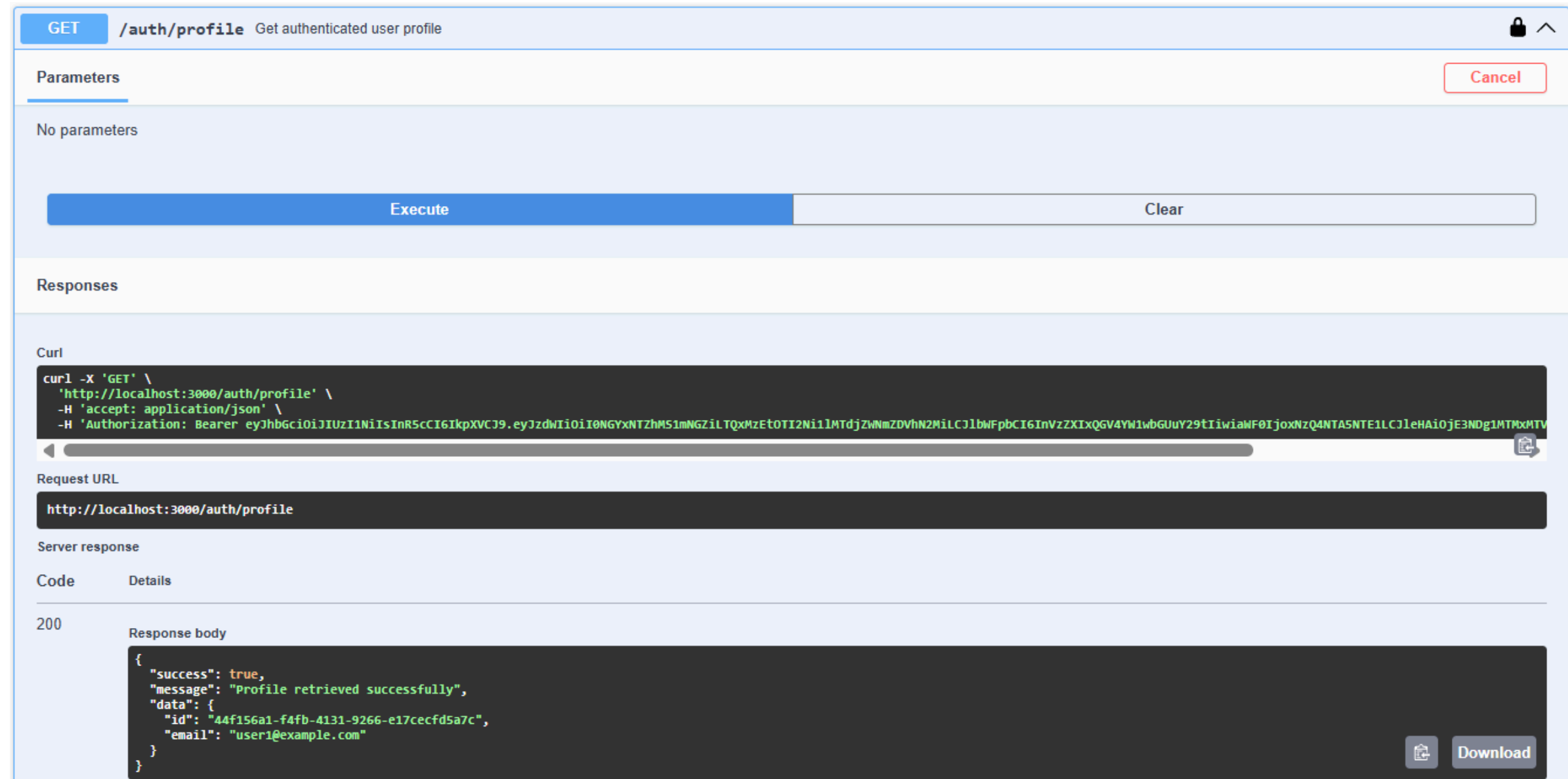
*\*Se necesita un token JWT*

# Auth Endpoints: 4

GET /auth/profile – Perfil del usuario autenticado



*Prueba particular*



The screenshot shows a REST client interface with the following sections:

- Parameters:** A tab labeled "Parameters" with a "Cancel" button. Below it, it says "No parameters". At the bottom of this section are "Execute" and "Clear" buttons.
- Responses:** A section containing a "Curl" block with the following command:

```
curl -X 'GET' \
  'http://localhost:3000/auth/profile' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI0NGYxNTZhMS1mNGZlLTQxMzEtOTI2Ni1MTdjZWNmZDVhbnZMiLCJlbWFPbCI6InVzZXIyQGV4YW1wbGUuY29tIiwiaWF0IjoxNzQ4NTA5NTE1LCJleHAiOiE3NDg1MTMxMTV
```

- Request URL:** A field containing the URL `http://localhost:3000/auth/profile`.
- Server response:** A section with a table showing the response details.

Code	Details
200	<p>Response body</p> <pre>{   "success": true,   "message": "Profile retrieved successfully",   "data": {     "id": "44f156a1-f4fb-4131-9266-e17cecf5a7c",     "email": "user1@example.com"   } }</pre>

*\*Se necesita un token JWT*



# Restaurants Endpoints: 5

*Documentación General*

**GET** `/restaurants`: Buscar restaurantes por latitud/longitud o nombre de ciudad



GET

`/restaurants` Find restaurants near a location

Try it out

Parameters

Name	Description
lat	Latitude coordinate
number (query)	<input type="text" value="40.7128"/>
lon	Longitude coordinate
number (query)	<input type="text" value="-74.006"/>
city	City name for finding restaurants
string (query)	<input type="text" value="New York"/>
radius	Search radius in meters (default: 1000)
number (query)	<input type="text" value="1000"/>

Responses

Code	Description	Links
200	List of restaurants found near the specified location	No links
	<div>Media type</div> <div><div>application/json</div></div> <div>Controls Accept header.</div> <div>Example Value   Schema</div> <div><pre>{  "success": true,  "message": "Operation completed successfully",  "data": {},  "error": null}</pre></div>	
400	Bad request - missing required parameters	No links
401	Unauthorized - invalid or expired token	No links

*\*Se necesita un token JWT*



# Restaurants Endpoints: 5

*Prueba particular*

GET /restaurants: Buscar restaurantes por latitud/longitud o nombre de ciudad



GET /restaurants Find restaurants near a location

Cancel

Name	Description
lat number (query)	Latitude coordinate 40.7128
lon number (query)	Longitude coordinate -74.006
city string (query)	City name for finding restaurants New York
radius number (query)	Search radius in meters (default: 1000) 1000

ExecuteClear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:3000/restaurants?lat=40.7128&lon=-74.006&city=New%20York&radius=1000' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJlbnV4NTZlMTQxMzE1MTdjdWVhbnVzZXI6ImF0eXNzQ4NTA5NTE1L3JleHA1OjE3NDg1MTMwMTV'

```

Request URL

```
http://localhost:3000/restaurants?lat=40.7128&lon=-74.006&city=New%20York&radius=1000

```

Server response

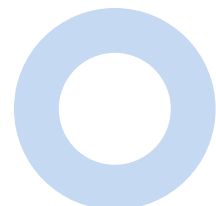
CodeDetails

200

Response body

```
{
  "success": true,
  "message": "Restaurants found successfully",
  "data": [
    {
      "id": "419367357",
      "name": "Nha Trang One",
      "latitude": 40.7167899,
      "longitude": -73.9996711,
      "address": "87 Baxter Street, New York, NY 10013",
      "cuisine": "vietnamese",
      "phone": "+1-212-233-5948",
      "website": "http://nhatrangnyc.com",
      "openingHours": "Mo-Su 10:30-22:00"
    },
    {
      "id": "1396581104",
      "name": "Hoy Mong Restaurant",
      "latitude": 40.7166141,
      "longitude": -73.998003,
      "address": "81 Mott street, New York"
    },
    {
      "id": "1934502370",
      "name": "Blue Smoke",
      "latitude": 40.7146177,
      "longitude": -74.015376,
      "address": "8355 Mott Street, 10003"
    }
  ]
}
```

Download



*\*Se necesita un token JWT*

# Transactions Endpoints: 6

**GET /transactions:** Obtener el historial de transacciones del usuario actual





*Documentación General*

GET

/transactions

Get transaction history for the authenticated user



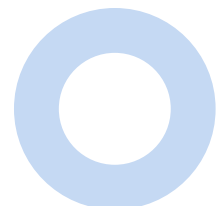
Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	<div>List of transactions for the authenticated user</div> <div>Media type<div>application/json</div></div> <div>Controls Accept header.</div> <div>Example Value   Schema<div><pre>{   "success": true,   "message": "Operation completed successfully",   "data": {},   "error": null }</pre></div></div>	No links
401	Unauthorized - invalid or expired token	No links



*\*Se necesita un token JWT*

## Transactions Endpoints: 6

**GET /transactions:** Obtener el historial de transacciones del usuario actual





### Prueba particular

GET

/transactions

Get transaction history for the authenticated user



Parameters

No parameters

Cancel

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:3000/transactions' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ3ZmI0I0NGYxNTZhMS1mNGZlLTQxMzEtOTI2Ni11MTdjZW5mZDZhN2MlLCJlbWVpbCI6InVzZXIxQG4Yw1wbGUuY29tIiwiaWF0IjoxNzQ0NTA5NTE1LCJleHA1OjE3NDg1MTMxMTV' \
'
```

Request URL

http://localhost:3000/transactions

Server response

Code

Details

200

Response body

```
{
  "success": true,
  "message": "Transactions retrieved successfully",
  "data": [
    {
      "id": "b5c4f1e2-5e57-43f1-a5a6-87a654d170fd",
      "userId": "44f156a1-f4fb-4131-9266-e17cecf5a7c",
      "type": "TRANSACTION",
      "endpoint": "/transactions",
      "params": "{}",
      "description": "Viewed transaction history",
      "createdAt": "2025-05-29T09:07:46.989Z"
    },
    {
      "id": "7abe8974-8c39-4141-be7a-a3e1f988fd16",
      "userId": "44f156a1-f4fb-4131-9266-e17cecf5a7c",
      "type": "SEARCH",
      "endpoint": "/restaurants?lat=40.7128&lon=-74.006&city=New%20York&radius=1000",
      "params": "{\n  \"query\": {\n    \"lat\": \"40.7128\", \"lon\": \"-74.006\", \"city\": \"New York\", \"radius\": \"1000\"\n  }\n}",
      "description": "Searched for restaurants near coordinates (40.7128, -74.006)",
      "createdAt": "2025-05-29T09:07:11.157Z"
    },
    {
      "id": "2f54358c-e20b-47dd-8147-aaadb0878012",
      "userId": "44f156a1-f4fb-4131-9266-e17cecf5a7c",
      "type": "AUTH",
      "endpoint": "/auth/profile",

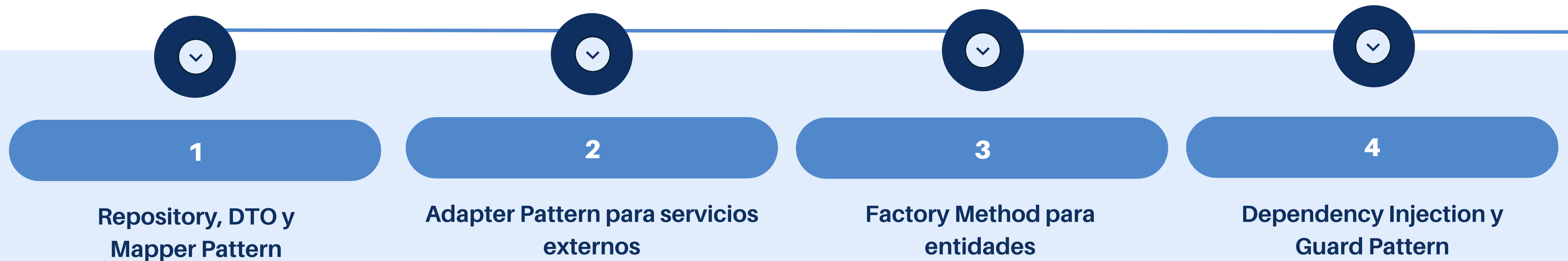
```

Download

**\*Se necesita un token JWT**

# Patrones de Diseño

Se utilizan patrones de diseño para asegurar buenas prácticas, bajo acoplamiento y código reutilizable.



## Buenas prácticas

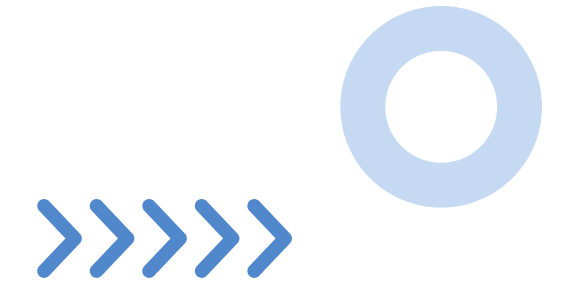
Adicionalmente, se implementan buenas prácticas, como:

- Middlewares
- Manejo de excepciones
- Hashing
- Uso de typescript (fuertemente tipado)
- Contenerización
- Principios SOLID
- Configuración para variables de entorno



# Decisiones de Diseño

Las decisiones tomadas garantizan escalabilidad, simplicidad y adaptabilidad a futuro.



1

- Arquitectura hexagonal

3

- PostgreSQL + TypeORM

2

- JWT como estrategia de autenticación

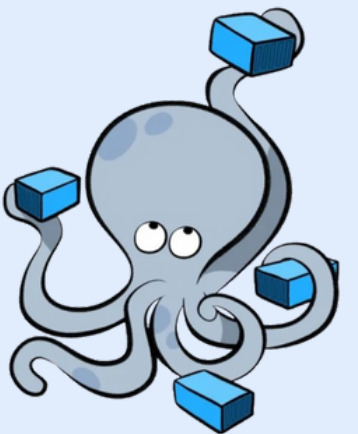
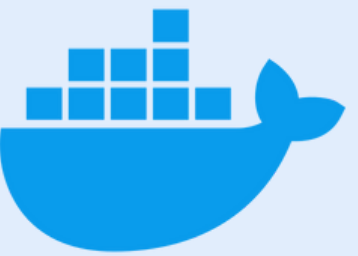
4

- Docker para despliegue y PostgreSQL embebido

## Ejecución

Podrá ejecutar el código en un entorno que tenga instalado Node.js y una base de datos PostgreSQL. Además, si cuenta con docker instalado, podrá contenerizar completamente el sistema ejecutando el comando “Docker compose up” en la raíz del repositorio. Recuerde que debe configurar el archivo .env con sus respectivas variables de entorno.

***Para instrucciones más detalladas, visite el archivo README.md en el repositorio.***





# Pruebas automatizadas

Se realizaron pruebas automatizadas (con Jest) para garantizar la calidad, estabilidad y mantenibilidad del backend. Se encuentran ubicadas en `/api/test`.



**Frameworks:** Jest



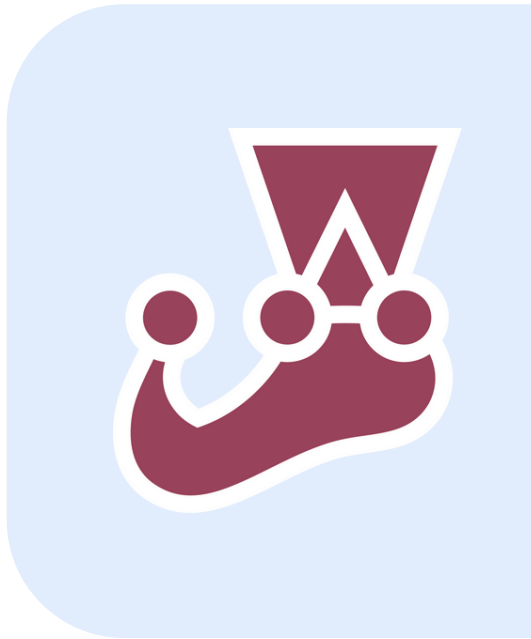
## Unitarias

Verifican la lógica aislada de servicios y entidades del dominio



## Integración

Validan la interacción entre módulos como controladores, servicios y base de datos



Se cubren tanto casos exitosos y fallos esperados (Validaciones, errores de autenticación, etc).  
Se realizaron:

## 23 Pruebas de Integración exitosas

```
PASS test/integration/auth.controller.integration.spec.ts (8.677 s)
PASS test/integration/transaction.controller.integration.spec.ts (9.907 s)
PASS test/integration/restaurant.controller.integration.spec.ts (13.159 s)

Test Suites: 3 passed, 3 total
Tests:      23 passed, 23 total
Snapshots:  0 total
Time:       13.734 s
Ran all test suites.
```

## 15 Pruebas unitarias exitosas

```
PASS test/unit/transaction.service.spec.ts (6.111 s)
PASS test/unit/auth.service.spec.ts (6.604 s)
PASS test/unit/restaurant.service.spec.ts (8.013 s)
A worker process has failed to exit gracefully and has been
on them.

Test Suites: 4 passed, 4 total
Tests:      15 passed, 15 total
Snapshots:  0 total
Time:       9.083 s
Ran all test suites.
```



# Repositorio

Podrá encontrar el código en

<https://github.com/luisalejandrob/tyba-backend-test>

