

PROYECTO COURIERSYNNC – FEATURE 1

ENTREGABLE
SPRINT 2

MATERIA

ARQUITECTURA DE SOFTWARE

INTEGRANTES

MIGUEL ANGEL AGUDELO VERA
DIANA CAROLINA HUERTAS

PROFESOR

DIEGO JOSE LUIS BOTIA VALDERRA



UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
CODE FACTORY

2025-1



Índice

1. Diseño detallado de APIs

- 1.1. Documentación formal de APIs (OpenAPI/Swagger completo) HATEOAS
- 1.2. Definición de contratos (campos obligatorios, respuestas esperadas, códigos de error)
- 1.3. Especificación de mecanismos de seguridad (autenticación/autorización)
- 1.4. Tabla de operaciones de la Api

2. Especificación de Protocolo de Comunicación Interna

- 2.1. Comunicación entre los componentes (HTTP REST, websockets, eventos, mensajería, gRPC, etc.)
- 2.2. Protocolos de comunicación hacia sistemas externos (si existen)

3. Implementación completa del Backend

- 3.1. Repositorio del código del proyecto junto con las pruebas realizadas en POSTMAN
- 3.2. Despliegue de contenedores Docker
- 3.3. Monitoreo de la aplicación con Prometheus / Grafana

1. Diseño detallado de APIs

1.1. Documentación formal de APIs (OpenAPI/Swagger completo)

HATEOAS

Esta sección describe la documentación formal de la API de CourierSync, generada automáticamente utilizando OpenAPI/Swagger (implementado con SpringDoc OpenAPI) y destacando la incorporación de los principios HATEOAS.

Introducción a la Documentación OpenAPI/Swagger

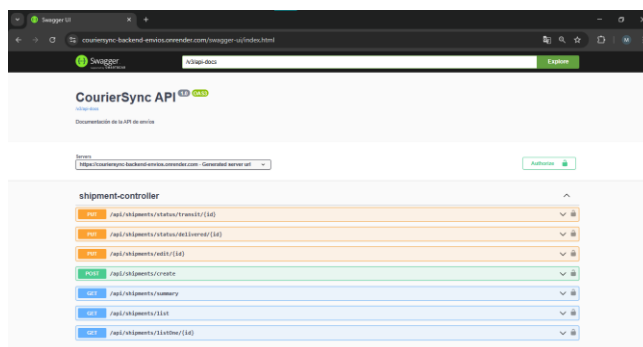
La API de CourierSync se documenta siguiendo la especificación OpenAPI (anteriormente conocida como Swagger), lo que permite una descripción estándar, independiente del lenguaje, de las APIs RESTful. Esto facilita tanto el consumo por parte de los clientes como la comprensión y mantenimiento por parte de los desarrolladores.

Para la implementación, se utiliza SpringDoc OpenAPI, una biblioteca que integra automáticamente la especificación OpenAPI 3 con aplicaciones Spring Boot, generando la documentación a partir de las anotaciones en el código.

Acceso a la Documentación Interfaz de Usuario (Swagger UI)

Una vez que la aplicación backend de CourierSync está en ejecución (típicamente en <https://couriersync-backend-envios.onrender.com>) la documentación interactiva de la API está disponible a través de la interfaz de usuario de Swagger en las siguientes URL:

<https://couriersync-backend-envios.onrender.com/swagger-ui/index.html>



Esta interfaz permite a los usuarios explorar los endpoints, ver los modelos de datos, probar las llamadas a la API directamente desde el navegador y observar las respuestas, incluyendo la implementación HATEOAS.

Incorporación de HATEOAS en la Documentación

La API de CourierSync implementa los principios de HATEOAS (Hypermedia As The Engine Of Application State) para mejorar la descubribilidad, la autonomía del cliente y la capacidad de evolución de la API. Esto significa que las respuestas de la API no solo incluyen los datos del recurso solicitado, sino también enlaces (`_links`) que indican las acciones posibles y las relaciones con otros recursos.

En Swagger UI, la integración HATEOAS se visualiza en los esquemas de respuesta de la siguiente manera:

- Propiedad `_links`: En los DTOs de respuesta (**AddressResponseDTO**, **ClientResponseDTO**, **ShipmentResponseDTO**), así como en las respuestas de colecciones (`_embedded`), se incluye una propiedad `_links` (o `links` dependiendo de la configuración y la versión específica de Spring HATEOAS/SpringDoc). Esta propiedad es un objeto o un array de objetos que contienen los enlaces relevantes.
- Estructura de los Enlaces: Cada enlace dentro de `_links` se compone de:
 - **rel** (Relation): Describe la relación del enlace con el recurso actual (ej., **self** para el propio recurso, **addresses** para la colección de direcciones, **update-shipment** para una acción de actualización).
 - **href** (Hypertext Reference): La URL a la que el cliente puede navegar para acceder a la relación o realizar la acción.

Ejemplos de Respuestas con HATEOAS (tal como se verían en la "Response body" de Swagger UI):

1. GET `/api/addresses/{id}` (o un elemento de `/api/addresses/list`)

Code	Details
200	<div>Response body</div> <pre>{ "id": 1, "city": "Bogotá", "address": "Calle 123 #45-67", "_links": { "self": { "href": "http://localhost:8080/api/addresses/1" }, "addresses": { "href": "http://localhost:8080/api/addresses/list" } } }</pre>

Explicación de Enlaces:

- self: Permite al cliente acceder directamente a los detalles de esta dirección específica.
- addresses: Permite al cliente navegar a la colección completa de direcciones

2. GET /api/shipments/listOne/{id} (o un elemento de /api/shipments/list)

Code	Details
200	<p>Response body</p> <pre>[{ "id": 1, "origin": "Bogotá, calle 123 #45-67", "destination": "Medellín, Carrera 89 #10-12 Sur", "client": "Empresa Alfa S.A.", "weight": 7.5, "priority": "MEDIA", "shippingDate": "2025-05-20T00:00:00.000+00:00", "deliveryDate": "2025-05-25T00:00:00.000+00:00", "registrationDate": "2025-05-17T02:55:22.111+00:00", "status": "entregado", "links": [{ "rel": "self", "href": "http://localhost:8080/api/shipments/listOne/1" }, { "rel": "shipments", "href": "http://localhost:8080/api/shipments/list" }, { "rel": "update-shipment", "href": "http://localhost:8080/api/shipments/edit/1" }, { "rel": "set-in-transit", "href": "http://localhost:8080/api/shipments/status/transit/1" }, { "rel": "set-delivered", "href": "http://localhost:8080/api/shipments/status/delivered/1" }, { "rel": "origin-address", "href": "http://localhost:8080/api/addresses/1" }, { "rel": "destination-address", "href": "http://localhost:8080/api/addresses/2" }, { "rel": "client", "href": "http://localhost:8080/api/clients/1" }] }]</pre>

Explicación de Enlaces:

- self: Acceso directo a los detalles de este envío.
- shipments: Acceso a la colección completa de envíos.
- update-shipment: Permite al cliente saber que este envío puede ser actualizado y proporciona la URL para la operación PUT.
- set-in-transit, set-delivered: Indican las transiciones de estado posibles para el envío y sus respectivas URLs.
- origin-address, destination-address, client: Proporcionan URLs para acceder a los detalles de los recursos relacionados (dirección de origen, dirección de destino y cliente del envío), facilitando la navegación por la API.

Ventajas de HATEOAS en la Documentación

- **Autodescubrimiento:** Los clientes pueden "descubrir" las capacidades de la API navegando por los enlaces en las respuestas, reduciendo la necesidad de una documentación estática y compleja para cada flujo.
- **Reducción de Acoplamiento:** Los clientes no necesitan codificar URL fijas; simplemente siguen los enlaces proporcionados, lo que hace que la API sea más resistente a cambios en la estructura de las URLs (siempre que las relaciones se mantengan).
- **Mejora de la Usabilidad:** Facilita a los desarrolladores entender las posibles interacciones con los recursos.

1.2. Definición de Contratos (campos obligatorios, respuestas esperadas, códigos de error)

Esta sección presenta los contratos para los endpoints más importantes de la API de CourierSync. Se describen los campos requeridos, los posibles códigos de respuesta y se muestran capturas del Swagger UI como evidencia de funcionamiento. Se recomienda consultar el Swagger directamente para ver todos los detalles dinámicos.

POST /api/auth/login


Descripción: Permite a un usuario autenticarse en el sistema. Devuelve un token JWT si las credenciales son válidas.

- **Campos requeridos:**
 - email: correo del usuario (formato válido, obligatorio)
 - password: contraseña del usuario (obligatoria)
- **Códigos de respuesta:**
 - 200 OK: autenticación exitosa
 - 400 Bad Request: campos vacíos o inválidos
 - 401 Unauthorized: credenciales incorrectas

Request URL

http://localhost:8080/api/auth/login

Server response

Code	Details
200	<div>Response body</div> <pre>{ "id": 1, "name": "Juan Pérez", "email": "juan.admin@couriersync.com", "role": "administrador", "message": "Login successful", "success": true, "token": "eyJhbGciOiJIUzI1NiIsInp0eSI6ImlkZW4uQG9vdXQpZXIzeW5jLmNvbSIsImkiOiJ3ZG9yTiwiYWoiOiJ0eXNzLWMTgANTY4L3leHAiOiJ3NTAxOTAzNjh9LmF5bnQ9KXJynQXEmyuuw", "links": { "refresh-token": { "href": "http://localhost:8080/api/auth/refresh-token" } } }</pre> <div> Download</div>

Descripción: Crea un nuevo envío en el sistema. Requiere datos del cliente, direcciones, prioridad, fechas y peso.

- **Campos requeridos:**
 - originAddressInfo y destinationAddressInfo: debe incluir un id de dirección existente o un objeto con city y address.
 - weight: mayor a 0
 - priorityName: ALTA, MEDIA o BAJA
 - clientInfo: cliente existente (id) o cliente nuevo
 - shippingDate y deliveryDate: fechas iguales o posteriores a la actual
- **Códigos de respuesta:**
 - 201 Created: envío creado correctamente
 - 400 Bad Request: validaciones fallidas

- 404 Not Found: datos no existentes (cliente, prioridad, dirección)

Request URL

`http://localhost:8080/api/shipments/create`

Server response

Code	Details
201 <i>Undocumented</i>	<p>Response headers</p> <pre>cache-control: no-cache,no-store,max-age=0,must-revalidate connection: keep-alive content-length: 0 date: Tue, 17 Jun 2025 19:32:18 GMT expires: 0 keep-alive: timeout=60 pragma: no-cache vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers x-content-type-options: nosniff x-frame-options: DENY x-xss-protection: 0</pre>

400
Undocumented

Error: response status is 400

Response body

```
{
  "details": {
    "weight": "Weight must be greater than 0"
  },
  "error": "Bad Request",
  "message": "Validation failed for request body.",
  "timestamp": "2025-06-17T14:45:33.3817063",
  "status": 400
}
```

GET /api/shipments/listOne/{id}

Descripción: Permite consultar los detalles de un envío específico por su ID.

- **Campo obligatorio (en la URL):**
 - id: identificador numérico del envío a consultar. Debe ser un entero positivo y corresponder a un envío existente.
- **Códigos de respuesta:**
 - 200 OK: envío encontrado. Devuelve información completa del envío, incluyendo enlaces HATEOAS.
 - 404 Not Found: no se encontró un envío con el ID proporcionado.



- 403 Forbidden: el usuario autenticado no tiene permisos para ver el envío (según su rol).

Request URL

`http://localhost:8080/api/shipments/listOne/1`

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 1, "origin": "Bogotá, Calle 123 #45-67", "destination": "Medellín, Carrera 89 #10-12 Sur", "client": "Empresa Alfa S.A.", "weight": 7.5, "priority": "MEDIA", "shippingDate": "2025-05-20T00:00:00.000+00:00", "deliveryDate": "2025-05-25T00:00:00.000+00:00", "registrationDate": "2025-05-17T02:55:22.111+00:00", "status": "entregado", "_links": { "self": { "href": "http://localhost:8080/api/shipments/listOne/1" }, "shipments": { "href": "http://localhost:8080/api/shipments/list" }, "update-shipment": { "href": "http://localhost:8080/api/shipments/edit/1" }, "set-in-transit": { "href": "http://localhost:8080/api/shipments/status/transit/1" }, "set-delivered": { "href": "http://localhost:8080/api/shipments/status/delivered/1" } } }</pre>

⚠ Formato estándar de errores

Todas las respuestas de error siguen un formato JSON uniforme, útil para debugging y validación en frontend.

Request URL

`http://localhost:8080/api/addresses/create`

Server response

Code	Details
400	<p><i>Undocumented</i> Error: response status is 400</p> <p>Response body</p> <pre>{ "details": { "city": "City cannot be empty" }, "error": "Bad Request", "message": "Validation failed for request body.", "timestamp": "2025-06-17T14:49:35.710765", "status": 400 }</pre>

1.3 Especificación de Mecanismos de Seguridad (Autenticación/Autorización)

La seguridad de la API de CourierSync se implementa utilizando un enfoque robusto que combina la autenticación basada en JSON Web Tokens (JWT) y la autorización basada en roles proporcionada por Spring Security.

Autenticación (JSON Web Tokens - JWT)

- **Propósito:** Verificar la identidad del usuario que realiza la petición a la API.
- **Mecanismo:** La autenticación se realiza a través de JSON Web Tokens (JWT). Cuando un usuario se autentica exitosamente, el servidor emite un JWT que el cliente debe incluir en las cabeceras de futuras peticiones.
- **Flujo de Autenticación:**
 1. **Petición de Login:** El cliente envía una petición POST al endpoint de autenticación (ej., /api/auth/login) con las credenciales del usuario (email y contraseña).
 2. **Generación del Token:** Si las credenciales son válidas, el servidor genera un JWT firmado digitalmente. Este token contiene información sobre el usuario (Claims), como su ID, roles y tiempo de expiración.
 3. **Respuesta al Cliente:** El servidor devuelve el JWT al cliente.
 4. **Inclusión del Token en Peticiones Posteriores:** Para acceder a los recursos protegidos de la API, el cliente debe incluir este JWT en la cabecera Authorization de cada petición subsiguiente, utilizando el esquema Bearer (ej., Authorization: Bearer <tu_token_jwt>).
 5. **Validación del Token:** En cada petición, el backend intercepta el JWT, lo valida (firma, expiración) y extrae la información del usuario para establecer el contexto de seguridad en Spring Security.
- **Componentes Clave:**
 - **AuthService:** Maneja la lógica de autenticación, verifica credenciales y utiliza JwtService para generar JWTs.
 - **JwtService:** Responsable de la generación, validación y extracción de información de los JWTs (ej., clave secreta, expiración, Claims).
 - **JwtAuthenticationFilter:** Un filtro de Spring Security que intercepta cada petición. Extrae el JWT de la cabecera Authorization, lo valida y, si es válido, autentica al usuario en el contexto de seguridad de Spring.

- **Método:** POST
- **Ruta:** /api/auth/login
- **Request Body (LoginRequest):**

POST	/api/auth/login
Parameters	
No parameters	
Request body <small>required</small>	
<pre>{ "email": "juan.admin@couriersync.com", "password": "admin123" }</pre>	
<pre>{ "email": "usuario@ejemplo.com", "password": "miContraseñaSegura" }</pre>	

- **Respuesta Exitosa (AuthenticationResponse):**

```
Request URL
http://localhost:8080/api/auth/login

Server response

Code    Details

200

Response body
{
  "id": 1,
  "name": "Juan Pérez",
  "email": "juan.admin@couriersync.com",
  "role": "administrador",
  "message": "Login successful",
  "success": true,
  "token": "eyJhcCI6IjUzIiwia3IyZmI0IjogdFwlfkthbWluQGNvdiCpZCJ2ZW5jLmV5S1I0K3V2XTI0JesIjoiJ2V0bWU0IjhzZGpmlzdhZG9yIiwiaWF0IjoxNzUyMTg0NTY4Iiwia10jE3NTAxOTAtNjhh9-faym9Q9KcxjynQXewywnha-6WBG7Zzh2h2atvh0ebcuua",
  "links": {
    "refresh-token": {
      "href": "http://localhost:8080/api/auth/refresh-token"
    }
  }
}
```

- **Propósito:** Determinar si un usuario autenticado tiene los permisos necesarios para acceder a un recurso o realizar una operación específica.
- **Mecanismo:** Spring Security se configura para proteger los endpoints basándose en los roles asignados al usuario.

- **Asignación de Roles:**

- Durante el proceso de autenticación, los roles del usuario se extraen del JWT (o de la base de datos) y se asocian al contexto de seguridad de Spring.
- Los roles definidos en tu sistema incluyen, pero no se limitan a:
ROLE_ADMINISTRADOR, ROLE_OPERADOR, ROLE_CONDUCTOR.

- **Protección de Endpoints:**

- La configuración de seguridad (SecurityConfiguration) define reglas de autorización a nivel de ruta (URL) y/o método.
- Se utilizan anotaciones de Spring Security (@PreAuthorize) o configuraciones de HttpSecurity para especificar qué roles tienen acceso a qué endpoints.

- **Ejemplos de Reglas de Autorización:**

- **Creación de Direcciones (POST /api/addresses/create):** Requiere ROLE_ADMINISTRADOR o ROLE_OPERADOR.

```
@SecurityRequirement(name = "Authorization")  @ Miguel Agudelo
@PreAuthorize("hasAnyAuthority('ROLE_ADMINISTRADOR', 'ROLE_OPERADOR')")
@PostMapping("/create")
public ResponseEntity<AddressResponseDTO> createAddress(@Valid @RequestBody AddressRequestDTO requestDTO) {
    AddressResponseDTO response = addressService.createAddress(requestDTO);
    return ResponseEntity.status(HttpStatus.CREATED).body(response);
}
```

- **Listado de Direcciones/Clientes/Envíos (GET /api/addresses/list, /api/clients/list, /api/shipments/list):** Requiere ROLE_ADMINISTRADOR, ROLE_OPERADOR o ROLE_CONDUCTOR.

```
@SecurityRequirement(name = "Authorization")  @ Miguel Agudelo
@PreAuthorize("hasAnyAuthority('ROLE_ADMINISTRADOR', 'ROLE_OPERADOR', 'ROLE_CONDUCTOR')")
@GetMapping("/list")
public ResponseEntity<List<AddressResponseDTO>> getAllAddresses() {
    List<AddressResponseDTO> addresses = addressService.getAllAddresses();
    return ResponseEntity.ok(addresses);
}
```

- **Resumen de Envíos (GET /api/shipments/summary):** Requiere ROLE_ADMINISTRADOR.

```
@GetMapping("/summary") @Miguel Agudelo
@SecurityRequirement(name = "Authorization")
@PreAuthorize("hasAuthority('ROLE_ADMINISTRADOR')")
public ResponseEntity<ShipmentSummaryResponseDTO> getShipmentSummary() {
    ShipmentSummaryResponseDTO summary = shipmentService.getShipmentSummaryForAdmin();
    return ResponseEntity.ok(summary);
}
```

- **Manejo de Acceso Denegado:**

- Si un usuario autenticado intenta acceder a un recurso para el cual no tiene el rol (ROLE) requerido, Spring Security denegará el acceso.
- **HTTP Status Code:** 403 Forbidden
- **Body de Respuesta (ejemplo):** El formato puede variar, pero generalmente indica que el acceso está prohibido.

Request URL	
http://localhost:8080/api/auth/login	
Server response	
Code	Details
200	<div>Response body</div> <pre>{ "id": null, "name": null, "email": null, "role": null, "message": "Invalid credentials", "success": false, "token": null }</pre>

Configuración de Seguridad General

- **Manejo de Sesiones:** La API está configurada para ser stateless (sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)). Esto significa que no se mantienen sesiones en el servidor entre peticiones, lo que es crucial para la escalabilidad y el funcionamiento de las APIs RESTful basadas en JWT.
- **Deshabilitación de CSRF:** La protección CSRF (Cross-Site Request Forgery) está deshabilitada (csrf().disable()), ya que las APIs RESTful que usan autenticación basada en tokens no suelen ser vulnerables a este tipo de ataque, y su habilitación podría interferir con el funcionamiento de la API.



- **Manejo de Excepciones de Autenticación/Autorización:** Se utiliza un `authenticationEntryPoint` (ej., `JwtAuthenticationEntryPoint`) para manejar las excepciones de autenticación y un `accessDeniedHandler` para las excepciones de autorización, asegurando que se devuelvan respuestas HTTP 401 Unauthorized o 403 Forbidden apropiadas.

1.4. Tabla de operaciones de la Api

Sr. No.	API Name	HTTP Method	Path	Status Code	Description
(1)	Login	POST	/api/auth/login	200 (OK) / 401	Autentica un usuario y genera un token JWT
(2)	Refresh Token	POST	/api/auth/refresh-token	200 (OK) / 403	Genera un nuevo token si el actual está por expirar
(3)	Crear envío	POST	/api/shipments/create	201 (Created) / 400	Registra un nuevo envío
(4)	Editar envío	PUT	/api/shipments/edit/{id}	200 (OK) / 400	Actualiza los datos de un envío existente
(5)	Obtener envío por ID	GET	/api/shipments/listOne/{id}	200 (OK) / 404	Consulta un envío específico
(6)	Listar todos los envíos	GET	/api/shipments/list	200 (OK)	Retorna todos los envíos registrados
(7)	Cambiar a estado En tránsito	PUT	/api/shipments/status/transit/{id}	200 (OK) / 400	Cambia estado de



					envío a "En tránsito"
(8)	Cambiar a estado Entregado	PUT	/api/shipments/status/delivered/{id}	200 (OK) / 400	Cambia estado de envío a "Entregado"
(9)	Resumen de estados de envíos	GET	/api/shipments/summary	200 (OK)	Muestra el resumen general de envíos por estado
(10)	Crear dirección	POST	/api/addresses/create	201 (Created) / 400	Registra una nueva dirección
(11)	Listar direcciones	GET	/api/addresses/list	200 (OK)	Muestra todas las direcciones
(12)	Obtener dirección por ID	GET	/api/addresses/{id}	200 (OK) / 404	Consulta una dirección específica
(13)	Crear cliente	POST	/api/clients/create	201 (Created) / 400	Registra un nuevo cliente
(14)	Listar clientes	GET	/api/clients/list	200 (OK)	Retorna todos los clientes registrados
(15)	Obtener cliente por ID	GET	/api/clients/{id}	200 (OK) / 404	Consulta un cliente específico

2. Especificación de Protocolo de Comunicación Interna

2.1. Comunicación entre los componentes (HTTP REST, websockets, eventos, mensajería, gRPC, etc.)

El sistema CourierSync implementa una arquitectura basada en servicios utilizando el **protocolo HTTP/1.1** bajo el estilo arquitectónico **RESTful API**. La comunicación entre los distintos módulos internos (controladores, servicios, repositorios) se realiza mediante los siguientes mecanismos:

Tipo de Comunicación	Protocolo	Tecnología / Medio	Descripción
Cliente ↔ Backend	HTTP REST	Spring Web + Controllers	Toda la comunicación se basa en endpoints REST con cuerpos en formato JSON
Controladores ↔ Servicios	Interna (Java)	Llamadas directas (inyección de dependencias vía @Autowired)	Se emplea arquitectura en capas, desacoplada mediante interfaces
Servicios ↔ Base de Datos	JDBC (JPA)	Spring Data JPA	El backend se comunica con PostgreSQL usando ORM con Hibernate

- **Formato de datos:** Se utiliza application/json como formato estándar para las peticiones y respuestas.
- **Estilo de comunicación:** Sincrónica en todos los casos.
- **Documentación:** Swagger (OpenAPI v3) expone la estructura y documentación de los endpoints.

Actualmente no se emplean mecanismos asincrónicos como WebSockets, eventos, colas de mensajería (RabbitMQ, Kafka), ni comunicación por gRPC



2.2. Protocolos de comunicación hacia sistemas externos (si existen)

En esta fase del proyecto, el backend no se comunica directamente con sistemas externos. Toda la lógica y persistencia es manejada de forma interna. Sin embargo, la arquitectura está preparada para integrar servicios de terceros en fases futuras, como por ejemplo:

Potencial Integración	Tipo de Protocolo	Ejemplo de uso futuro
Pasarela de pagos	REST / SOAP	Integración con PayU, Stripe, etc.
Envío de notificaciones	REST / SMTP	API de Twilio, Firebase o Email SMTP
Microservicios internos	REST / gRPC	Separar módulos por dominio

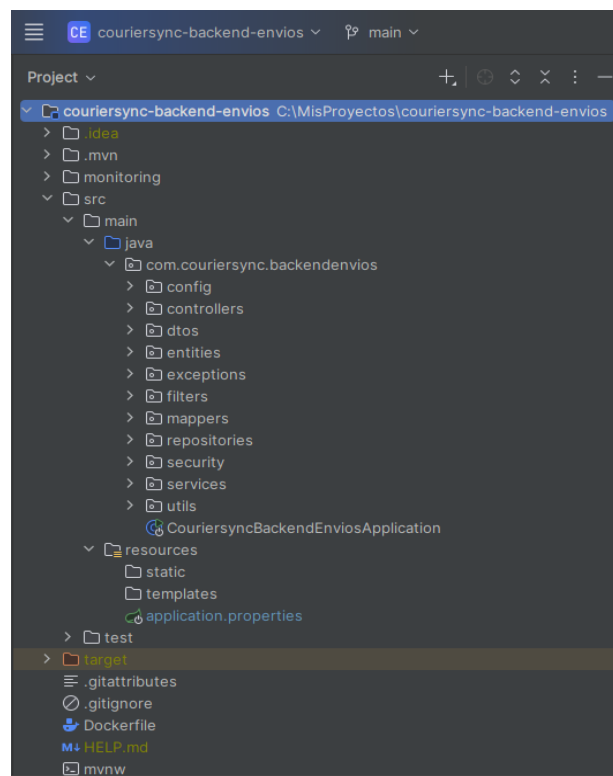
3. Implementación completa del Backend

3.1. Repositorio del código del proyecto junto con las pruebas realizadas en POSTMAN

El código fuente del backend del proyecto **CourierSync** se encuentra implementado en **Java 17** utilizando el framework **Spring Boot 3.3**, con conexión a base de datos **PostgreSQL** y documentación expuesta mediante **Swagger (OpenAPI)**. El backend implementa una arquitectura multicapa con separación de responsabilidades (controladores, servicios, repositorios y DTOs).

Repositorio del proyecto

El proyecto completo se encuentra disponible en un repositorio Git (remoto). El árbol principal de carpetas está estructurado así:

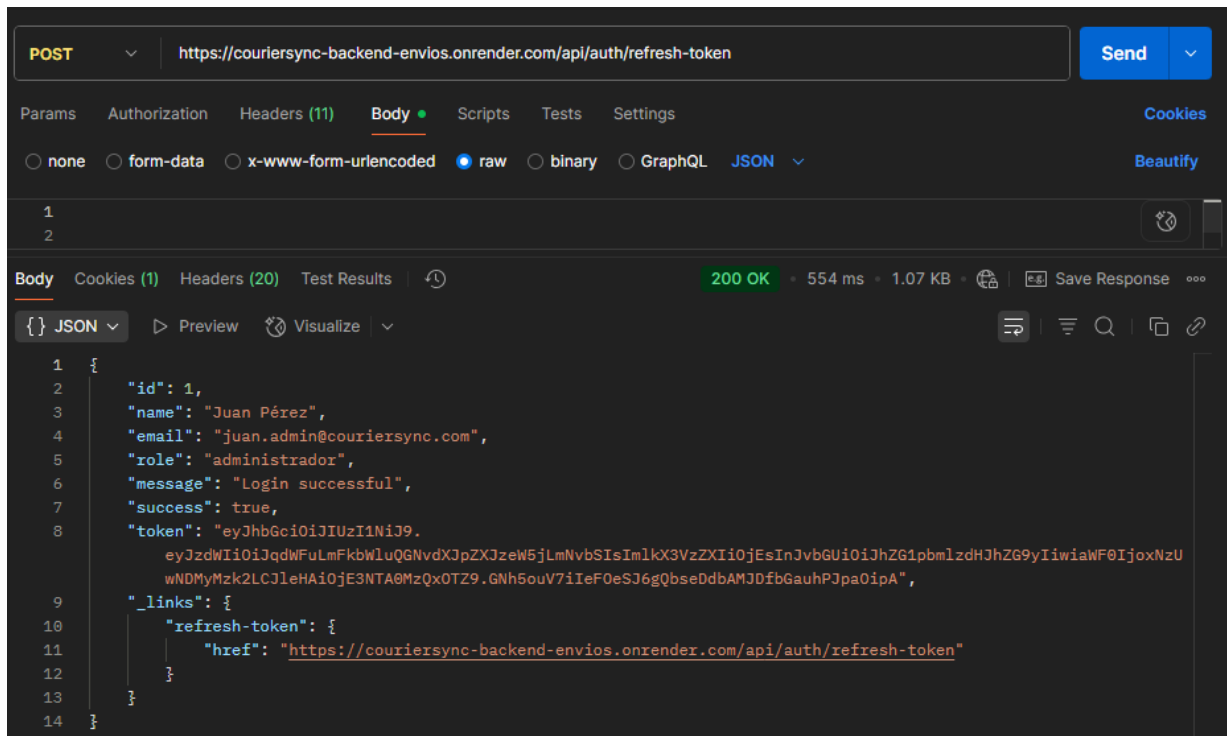


Link Repositorio GitHub: <https://github.com/Miguel-Agudelo/coursiersync-backend-envios>

Pruebas realizadas con Postman

Se realizaron pruebas a los endpoints principales usando **Postman**, incluyendo autenticación, gestión de envíos, clientes y direcciones.

Autenticación:



19



POST

https://coursiersync-backend-envios.onrender.com/api/shipments/create

Send

Params Authorization Headers (11) Body Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "originAddressInfo": {
3     "id": 1
4   },
5   "destinationAddressInfo": {
6     "id": 2
7   },
8   "weight": 5.0,
9   "priorityName": "MEDIA",
10  "clientInfo": {
11    "id": 1
12  },
13  "shippingDate": "2025-08-01",
14  "deliveryDate": "2025-08-25"
15 }
```

Body Cookies (1) Headers (19) Test Results 201 Created 2.49 s 595 B Save Response

Raw Preview Visualize

1

GET

https://coursiersync-backend-envios.onrender.com/api/shipments/list

Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Body Cookies (1) Headers (22) Test Results 200 OK 1.68 s 1.63 KB Save Response

JSON Preview Visualize

```
416 {
417   "id": 10,
418   "origin": "Medellin, Carrera 80 #10-20 Sur",
419   "destination": "Bogotá, Calle 26 #68B-50",
420   "client": "Cliente Nuevo S.A.S.",
421   "weight": 5.0,
422   "priority": "MEDIA",
423   "shippingDate": "2025-08-01T00:00:00.000+00:00",
424   "deliveryDate": "2025-08-25T00:00:00.000+00:00",
425   "registrationDate": "2025-06-20T15:27:46.939+00:00",
426   "status": "pendiente",
427   "links": [
428     {
429       "rel": "self",
430       "href": "https://coursiersync-backend-envios.onrender.com/api/shipments/listOne/10"
431     },
432     {
433       "rel": "shipments",
434       "href": "https://coursiersync-backend-envios.onrender.com/api/shipments/list"
435     },
436     {
437       "rel": "update-shipment",
```

GET

https://coursiersync-backend-envios.onrender.com/api/shipments/listOne/10

Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Body Cookies (1) Headers (20) Test Results 200 OK 525 ms 1.7 KB Save Response

JSON Preview Visualize

```
1 {
2   "id": 10,
3   "origin": "Medellin, Carrera 80 #10-20 Sur",
4   "destination": "Bogotá, Calle 26 #68B-50",
5   "client": "Cliente Nuevo S.A.S.",
6   "weight": 25.0,
7   "priority": "ALTA",
8   "shippingDate": "2025-08-01T00:00:00.000+00:00",
9   "deliveryDate": "2025-08-25T00:00:00.000+00:00",
10  "registrationDate": "2025-06-20T15:27:46.939+00:00",
11  "status": "entregado",
12  "_links": {
13    "self": {
14      "href": "https://coursiersync-backend-envios.onrender.com/api/shipments/listOne/10"
15    },
16    "shipments": {
17      "href": "https://coursiersync-backend-envios.onrender.com/api/shipments/list"
18    },
19    "update-shipment": {
20      "href": "https://coursiersync-backend-envios.onrender.com/api/shipments/edit/10"
21    },
22    "set-in-transit": {
```



PUT

https://couriersync-backend-envios.onrender.com/api/shipments/edit/10

Send

Params Authorization Headers (11) Body Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "originAddressId": 1,
3   "destinationAddressId": 2,
4   "weight": 25,
5   "priorityId": 3,
6   "clientId": 1,
7   "shippingDate": "2025-08-01",
8   "deliveryDate": "2025-08-25"
9 }
10
```

Body Cookies (1) Headers (22) Test Results 200 OK 1.23 s 709 B Save Response

Raw Preview Visualize

1 Shipment updated successfully

GET

https://couriersync-backend-envios.onrender.com/api/shipments/summary

Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Body Cookies (1) Headers (22) Test Results 200 OK 1.19 s 731 B Save Response

{ JSON Preview Visualize

```
1 {
2   "pending": 3,
3   "inTransit": 2,
4   "delivered": 5,
5   "delayed": 0
6 }
```

PUT

https://couriersync-backend-envios.onrender.com/api/shipments/status/transit/10

Send

Params Authorization Headers (10) Body Scripts Tests Settings Cookies

Body Cookies (1) Headers (22) Test Results 200 OK 570 ms 718 B Save Response

Raw Preview Visualize

1 Shipment status updated to 'En transito'

PUT

https://couriersync-backend-envios.onrender.com/api/shipments/status/delivered/10

Send

Params Authorization Headers (10) Body Scripts Tests Settings Cookies

Body Cookies (1) Headers (22) Test Results 200 OK 660 ms 718 B Save Response

Raw Preview Visualize

1 Shipment status updated to 'Entregado'



Gestión de clientes:

```
POST https://courierysync-backend-envios.onrender.com/api/clients/create

{
  "name": "Distribuidora SAS",
  "email": "distribuidora.sas@ejemplo.com",
  "phone": "529343437"
}
```

201 Created • 2.16 s • 912 B

```
{
  "id": 7,
  "name": "Distribuidora SAS",
  "email": "distribuidora.sas@ejemplo.com",
  "phone": "529343437",
  "_links": {
    "self": {
      "href": "https://courierysync-backend-envios.onrender.com/api/clients/7"
    },
    "clients": {
      "href": "https://courierysync-backend-envios.onrender.com/api/clients/list"
    }
  }
}
```

```
GET https://courierysync-backend-envios.onrender.com/api/clients/list

200 OK • 449 ms • 999 B
```

```
[
  {
    "id": 7,
    "name": "Distribuidora SAS",
    "email": "distribuidora.sas@ejemplo.com",
    "phone": "529343437",
    "links": [
      {
        "rel": "self",
        "href": "https://courierysync-backend-envios.onrender.com/api/clients/7"
      },
      {
        "rel": "clients",
        "href": "https://courierysync-backend-envios.onrender.com/api/clients/list"
      }
    ]
  }
]
```

```
GET https://courierysync-backend-envios.onrender.com/api/clients/7

200 OK • 449 ms • 907 B
```

```
{
  "id": 7,
  "name": "Distribuidora SAS",
  "email": "distribuidora.sas@ejemplo.com",
  "phone": "529343437",
  "_links": {
    "self": {
      "href": "https://courierysync-backend-envios.onrender.com/api/clients/7"
    },
    "clients": {
      "href": "https://courierysync-backend-envios.onrender.com/api/clients/list"
    }
  }
}
```



Gestión de direcciones:

The first screenshot shows a POST request to `https://courierysync-backend-envios.onrender.com/api/addresses/create` with a JSON body: `{ "city": "barranquilla", "address": "calle 32 #3234" }`. The response is a 201 Created status with a JSON body: `{ "id": 14, "city": "barranquilla", "address": "calle 32 #3234", "_links": { "self": { "href": "https://courierysync-backend-envios.onrender.com/api/addresses/14" }, "addresses": { "href": "https://courierysync-backend-envios.onrender.com/api/addresses/list" } } }`.

The second screenshot shows a GET request to `https://courierysync-backend-envios.onrender.com/api/addresses/list` with a 200 OK status. The response is a JSON array containing the address object: `[{ "id": 14, "city": "barranquilla", "address": "calle 32 #3234", "links": [{ "rel": "self", "href": "https://courierysync-backend-envios.onrender.com/api/addresses/14" }, { "rel": "addresses", "href": "https://courierysync-backend-envios.onrender.com/api/addresses/list" }] }]`.

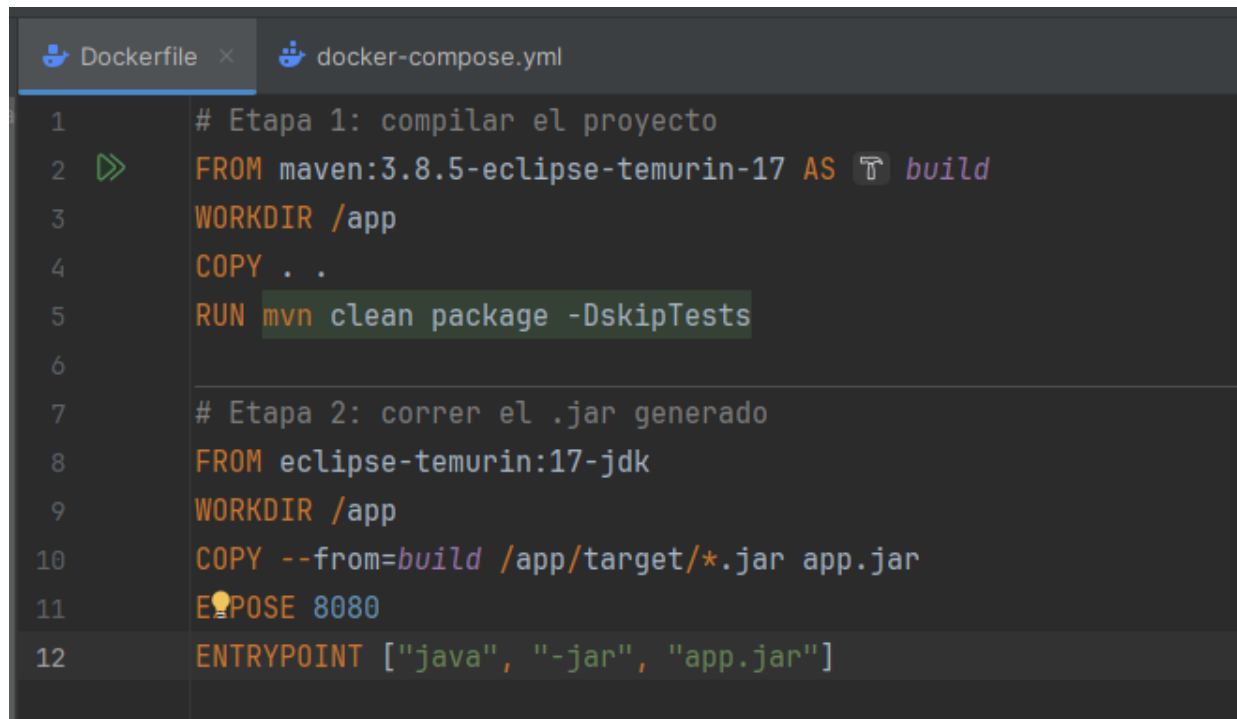
The third screenshot shows a GET request to `https://courierysync-backend-envios.onrender.com/api/addresses/14` with a 200 OK status. The response is a JSON object: `{ "id": 14, "city": "barranquilla", "address": "calle 32 #3234", "_links": { "self": { "href": "https://courierysync-backend-envios.onrender.com/api/addresses/14" }, "addresses": { "href": "https://courierysync-backend-envios.onrender.com/api/addresses/list" } } }`.

3.2. Despliegue de contenedores Docker

Para garantizar la portabilidad y facilidad de despliegue del backend de CourierSync, se utilizó Docker como herramienta de contenedorización. El proceso incluye la creación de una imagen personalizada del backend y su orquestación con Docker Compose.

Dockerfile

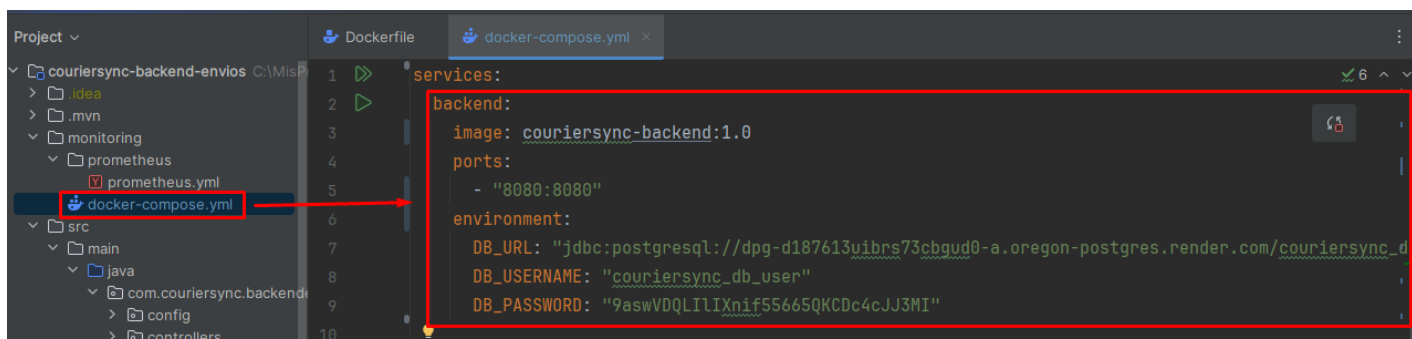
El siguiente Dockerfile fue creado en la raíz del proyecto. Utiliza una arquitectura de dos etapas: compilación y ejecución del .jar.



```
Dockerfile x  docker-compose.yml
1  # Etapa 1: compilar el proyecto
2  FROM maven:3.8.5-eclipse-temurin-17 AS build
3  WORKDIR /app
4  COPY . .
5  RUN mvn clean package -DskipTests
6
7  # Etapa 2: correr el .jar generado
8  FROM eclipse-temurin:17-jdk
9  WORKDIR /app
10 COPY --from=build /app/target/*.jar app.jar
11 EXPOSE 8080
12 ENTRYPOINT ["java", "-jar", "app.jar"]
```

Docker Compose

El archivo docker-compose.yml se colocó en la carpeta monitoring del proyecto. Este archivo define el servicio del backend con conexión a una base de datos PostgreSQL en la nube (Render).



```
Project v
└─ couriersync-backend-envios C:\MisP...
   └─ monitoring
      └─ prometheus
         └─ prometheus.yml
            └─ docker-compose.yml
               └─ src
                  └─ main
                     └─ java
                        └─ com.couriersync.backend
                           └─ config
                              └─ controllers

services:
  backend:
    image: couriersync-backend:1.0
    ports:
      - "8080:8080"
    environment:
      DB_URL: "jdbc:postgresql://dpg-d187613uibrs73cbgud0-a.oregon-postgres.render.com/couriersync-d
      DB_USERNAME: "couriersync_db_user"
      DB_PASSWORD: "9aswVDQLILIXnif55665QK0Dc4cJJ3MI"
```


Proceso de construcción y despliegue

```

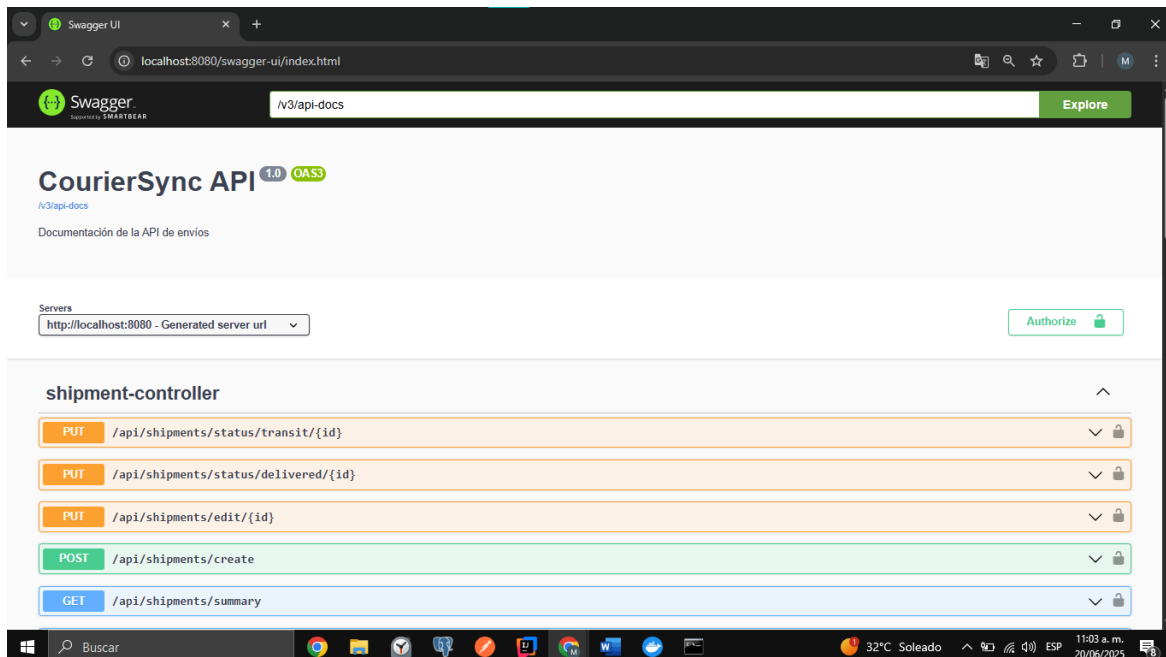
C:\Users\User>cd C:\MisProyectos\couriersync-backend-envios
C:\MisProyectos\couriersync-backend-envios>cd monitoring
C:\MisProyectos\couriersync-backend-envios\monitoring>docker-compose up -d
[+] Running 3/3
  Container monitoring-backend-1      Started      0.4s
  Container monitoring-prometheus-1   Started      0.6s
  Container monitoring-grafana-1      Started      0.9s
C:\MisProyectos\couriersync-backend-envios\monitoring>docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
NAMES
b1dc64df1064   couriersync-backend:1.0             "java -jar app.jar"     19 hours ago  Up 9 seconds  0.0.0.0:8080->8080/tcp
monitoring-backend-1
0364d7528b0c   grafana/grafana:latest              "/run.sh"               20 hours ago  Up 8 seconds  0.0.0.0:3000->3000/tcp
monitoring-grafana-1
14127844440d   prom/prometheus:latest              "/bin/prometheus --c..." 20 hours ago  Up 9 seconds  0.0.0.0:9090->9090/tcp
monitoring-prometheus-1
C:\MisProyectos\couriersync-backend-envios\monitoring>

```

La siguiente imagen muestra el contenedor levantado correctamente, utilizando Docker Compose

Verificación del backend

Una vez iniciado el contenedor, se puede acceder a la aplicación en local host sin necesidad de ejecutar la aplicación desde el IDE.



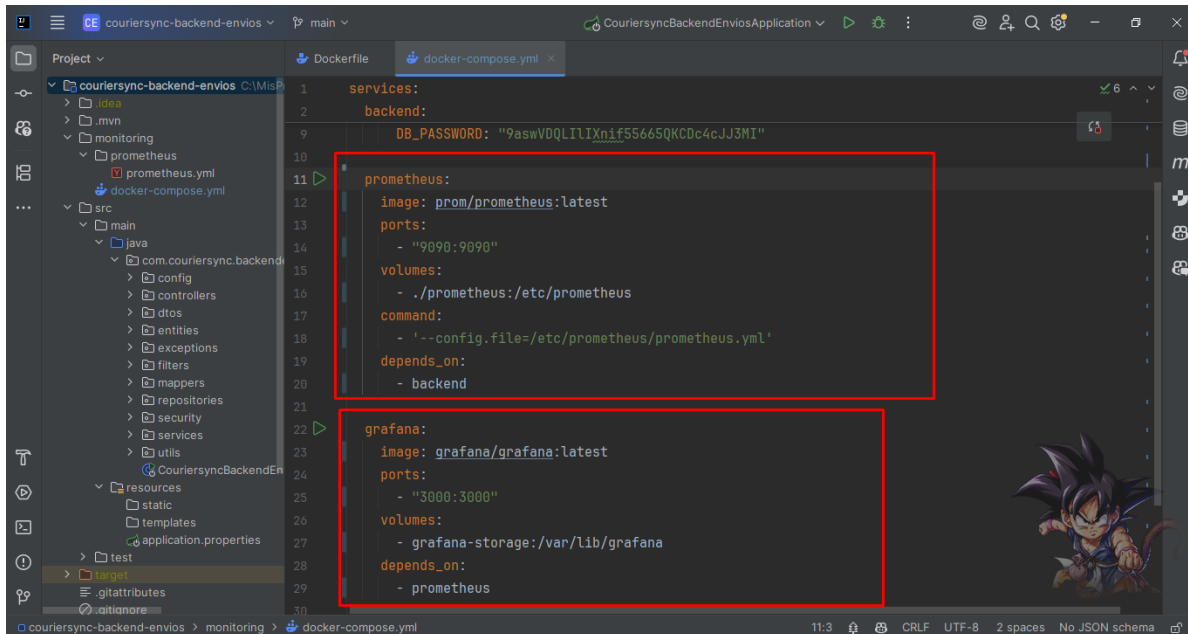
Esto confirma que el backend está desplegado correctamente y que la API está operativa.

3.3. Monitoreo de la aplicación con Prometheus / Grafana

Para implementar un sistema de monitoreo básico del backend de CourierSync, se integraron las herramientas Prometheus (para recolección de métricas) y Grafana (para visualización). Esto permite observar el comportamiento del sistema en tiempo real y facilitar el diagnóstico de problemas en ambientes productivos.

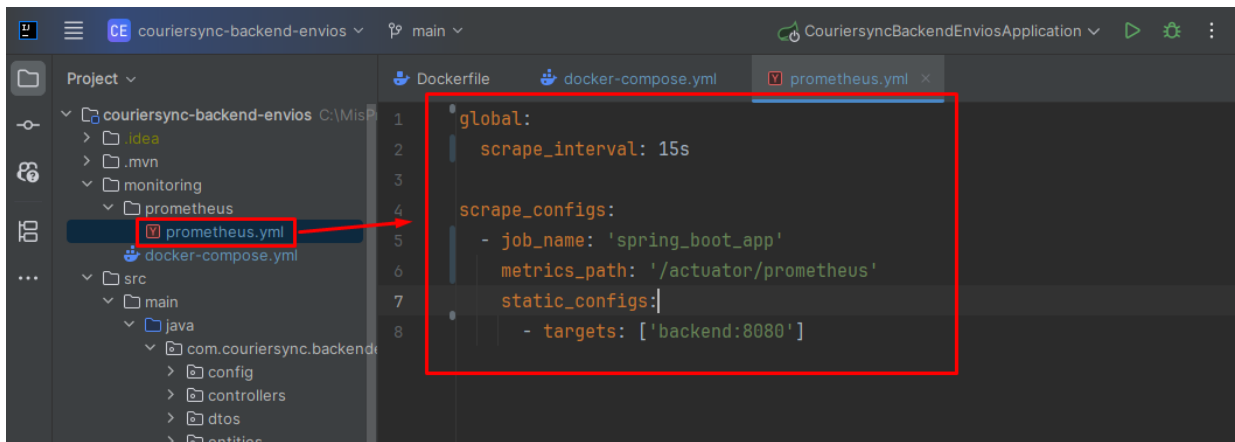
Configuración en docker-compose.yml

Los servicios de Prometheus y Grafana fueron definidos en el archivo docker-compose.yml ubicado en la carpeta monitoring, junto al contenedor del backend:



```
1 services:
2   backend:
3     image: couriersync/backend-envios:latest
4     ports:
5       - "8080:8080"
6     volumes:
7       - ./prometheus:/etc/prometheus
8     command:
9       - '--config.file=/etc/prometheus/prometheus.yml'
10    depends_on:
11      - backend
12
13  prometheus:
14    image: prom/prometheus:latest
15    ports:
16      - "9090:9090"
17    volumes:
18      - ./prometheus:/etc/prometheus
19    command:
20      - '--config.file=/etc/prometheus/prometheus.yml'
21    depends_on:
22      - backend
23
24  grafana:
25    image: grafana/grafana:latest
26    ports:
27      - "3000:3000"
28    volumes:
29      - grafana-storage:/var/lib/grafana
30    depends_on:
31      - prometheus
```

En el archivo prometheus.yml, se define la URL del backend como target para ser monitoreado:



```
1 global:
2   scrape_interval: 15s
3
4  scrape_configs:
5    - job_name: 'spring_boot_app'
6      metrics_path: '/actuator/prometheus'
7      static_configs:
8        - targets: ['backend:8080']
```

▶ Ejecución del entorno completo

El entorno de monitoreo se levanta automáticamente junto con el backend con el siguiente comando:

```

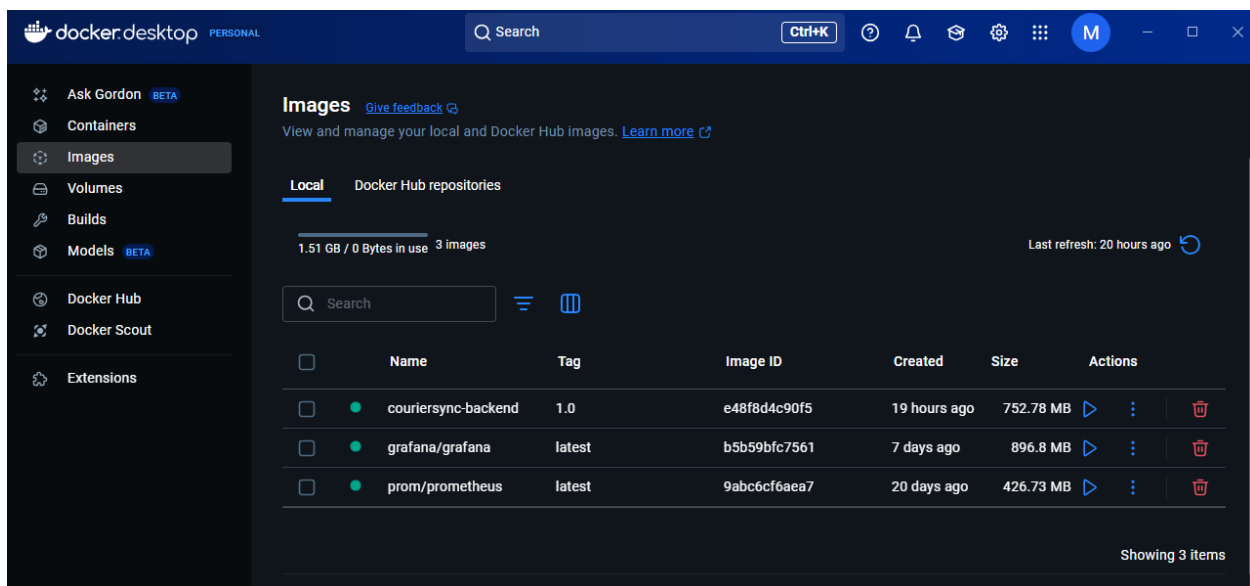
C:\MisProyectos\couriersync-backend-envios\monitoring>cd monitoring
El sistema no puede encontrar la ruta especificada.

C:\MisProyectos\couriersync-backend-envios\monitoring>docker-compose up -d
[+] Running 3/3
  Container monitoring-backend-1      Running
  Container monitoring-prometheus-1   Running
  Container monitoring-grafana-1      Running

C:\MisProyectos\couriersync-backend-envios\monitoring>

```

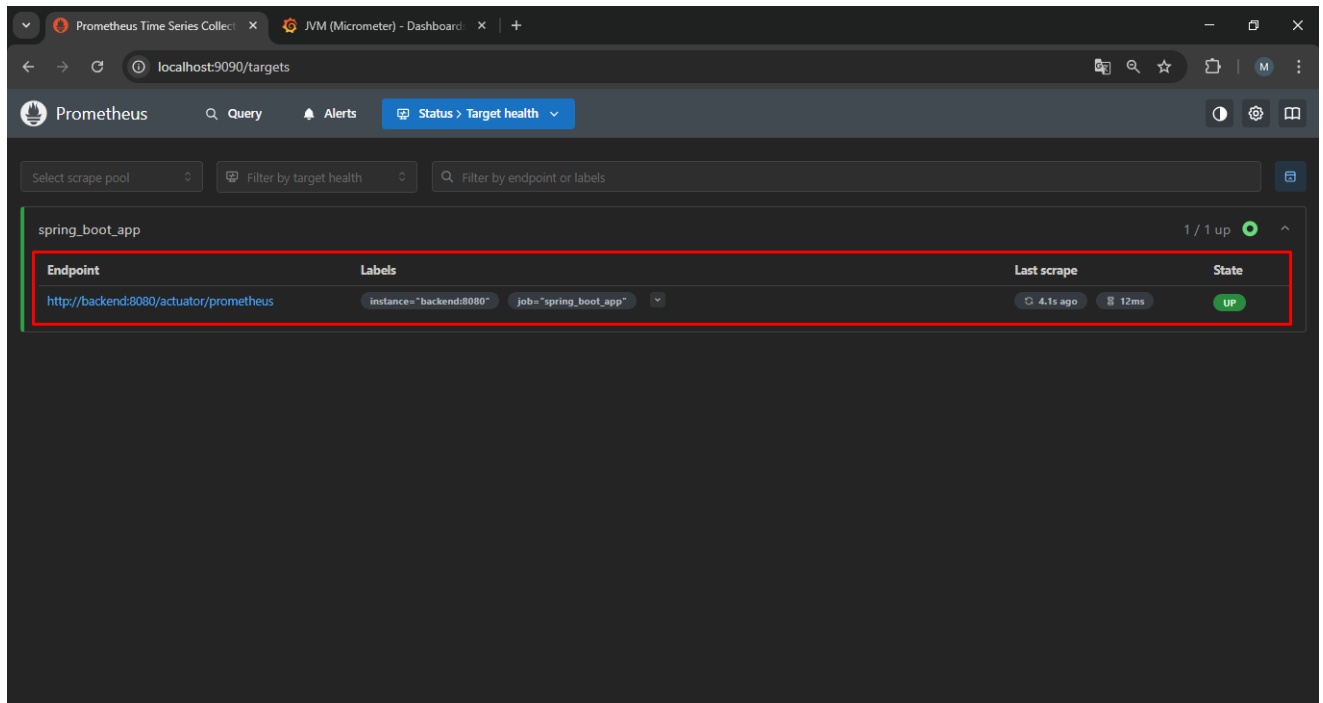
La captura muestra la ejecución correcta de los contenedores



🌐 Verificación del monitoreo

Una vez que los contenedores del backend, Prometheus y Grafana están levantados correctamente con Docker Compose, se procede a verificar el funcionamiento del sistema de monitoreo.

En Prometheus se verificó que el backend está siendo monitoreado correctamente, mostrando el estado UP del target `spring_boot_app` en la pestaña *Targets*.



En Grafana, se visualizó un dashboard que muestra métricas clave como el tiempo de actividad, uso de memoria y duración de peticiones HTTP, lo que confirma que el monitoreo está funcionando correctamente y recolectando datos en tiempo real.

