



**ESPE**  
UNIVERSIDAD DE LAS FUERZAS ARMADAS  
INNOVACIÓN PARA LA EXCELENCIA

**UNIVERSIDAD DE LAS FUERZAS ARMADAS –  
ESPE**

**UNIDAD DE EDUCACIÓN A DISTANCIA**

**ARQUITECTURA DE SOFTWARE**

**PADILLA ENRÍQUEZ DAVID ALEJANDRO**

**NRC: 29692**

**ACTIVIDAD DE APRENDIZAJE N3**

**SEGUNDO PARCIAL**

**PERIODO: OCT 25– FEB 26**

## 1. Introducción

Este documento describe la solución desarrollada para la Actividad 3. La implementación integra dos microservicios (Authors y Publications), un frontend web y el modelado del proceso editorial mediante BPMN en Camunda Modeler. El objetivo es evidenciar: separación por capas (Controller→Service→Repository), dependencia controlada Publications → Authors, aplicación de principios SOLID, uso de patrones de diseño y simulación del flujo editorial con Token Simulation.

### 1.1 Objetivos

- Implementar Authors Service para gestión de autores (PERSON/ORG) con herencia (clase abstracta + derivadas).
- Implementar Publications Service para gestión de publicaciones y estados editoriales.
- Demostrar la interacción Publications → Authors (validación de authorId y enriquecimiento de respuesta).
- Construir un frontend React que consuma ambos servicios y permita operaciones básicas.
- Modelar el proceso editorial en BPMN y definir escenarios de simulación (mínimo 3).

## 2. Tecnologías y herramientas

- Backend: Java 17 + Spring Boot 4.0.2 (REST), Jakarta Validation, Spring Data JPA.
- Persistencia: PostgreSQL.
- Frontend: Vite + React + TypeScript + PrimeReact (UI).
- Orquestación: Docker y Docker Compose.
- Modelado: Camunda Modeler (BPMN 2.0 + Token Simulation).

## 3. Arquitectura de la solución

La arquitectura se compone de un frontend que consume dos APIs REST independientes. Cada microservicio mantiene su propia base de datos (autonomía y desacoplamiento). Publications depende de Authors únicamente mediante HTTP (no existe dependencia circular).

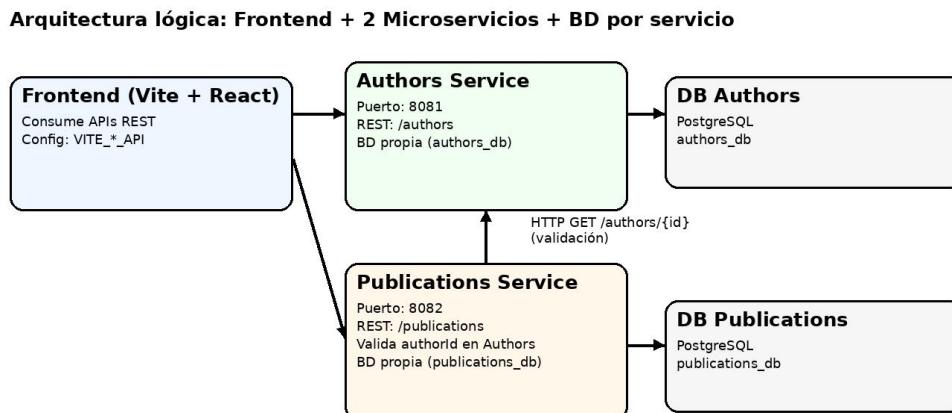


Figura 1. Arquitectura lógica de la solución (frontend + microservicios + BD por servicio).

### 3.1 Evidencia de estructura por capas

La organización de paquetes evidencia separación por capas: controller, service, repository, domain/entity, dto, mapper y exception. Esto facilita mantenibilidad, pruebas y cumplimiento de SOLID.

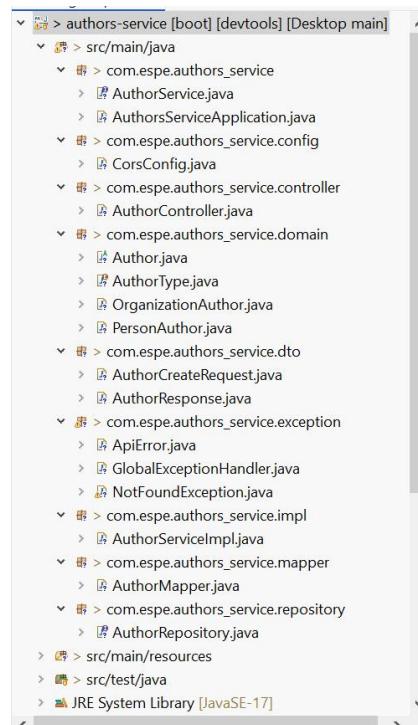


Figura 2. Estructura del microservicio Authors (capas, DTOs, mapper, repository, exceptions).

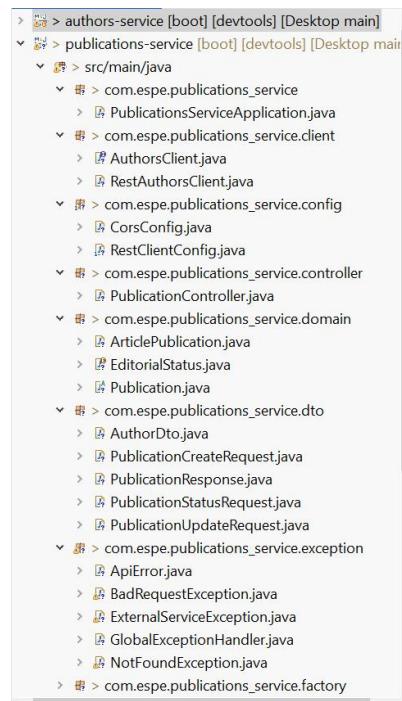


Figura 3. Estructura del microservicio Publications (capas, client, factory, validator, exceptions).

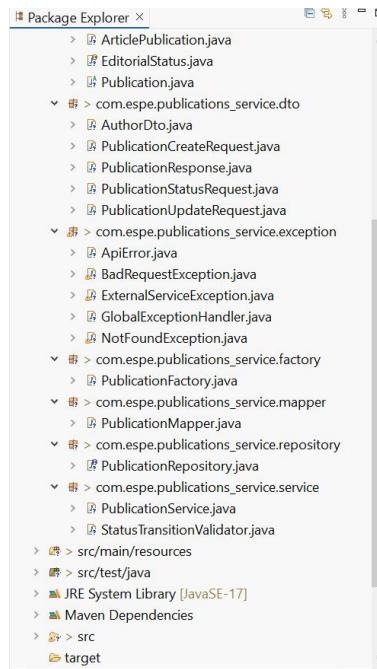


Figura 4. Evidencia adicional en Publications (service, repository, mapper).

## 4. Microservicio Authors

### 4.1 Propósito y modelo de dominio

Authors Service administra el ciclo de vida de autores. El dominio se implementa con una entidad abstracta Author y dos clases derivadas: PersonAuthor y OrganizationAuthor. Se usa herencia SINGLE\_TABLE con discriminador dtype y un enum AuthorType (PERSON/ORG).

### 4.2 Endpoints

- POST /authors – crea un autor (PERSON u ORG).
- GET /authors – lista autores.
- GET /authors/{id} – obtiene un autor por id.
- PUT /authors/{id} – actualiza un autor (no permite cambiar authorType).
- DELETE /authors/{id} – elimina un autor.

### 4.3 Ejemplos (request/response)

Crear autor PERSON (POST /authors):

Request

```
{
  "authorType": "PERSON",
  "fullName": "Juan Perez",
  "email": "juanperez@mail.com"
}
```



### Response (200 OK)

```
{  
  "id": 2,  
  "authorType": "PERSON",  
  "fullName": "Juan Perez",  
  "email": "juanperez@mail.com"  
}
```

The screenshot shows the Postman interface with a successful POST request to `http://localhost:8081/authors`. The request body is a JSON object representing a person author:

```
1  {  
2   "authorType": "PERSON",  
3   "fullName": "Juan Perez",  
4   "email": "juan.perez@mail.com"  
5 }  
6
```

The response is a 200 OK status with a response time of 171 ms and a response size of 337 B.

Figura 5. Evidencia Postman – creación de autor PERSON.

Crear autor ORG (POST /authors):

### Request

```
{  
  "authorType": "ORG",  
  "organizationName": "ESPE Editorial",  
  "contactEmail": "contacto@espeeditorial.com"  
}
```

### Response (200 OK)

```
{  
  "id": 3,  
  "authorType": "ORG",  
  "organizationName": "ESPE Editorial",  
  "contactEmail": "contacto@espeeditorial.com"  
}
```



The screenshot shows a POST request to `http://localhost:8081/authors`. The request body is a JSON object:

```
1 {
2     "authorType": "ORG",
3     "organizationName": "Editorial Andina",
4     "contactEmail": "contacto@editorial.com"
5 }
```

The response status is 200 OK, with a response time of 13 ms and a response size of 358 B. The response body is:

```
1 {
2     "authorType": "ORG",
3     "contactEmail": "contacto@editorial.com",
4     "id": 3,
5     "organizationName": "Editorial Andina"
6 }
```

Figura 6. Evidencia Postman – creación de autor ORG.

Listar autores (GET /authors):

The screenshot shows a GET request to `http://localhost:8081/authors`. The response status is 200 OK, with a response time of 16 ms and a response size of 445 B. The response body is:

```
1 [
2     {
3         "authorType": "PERSON",
4         "email": "juan.perez@mail.com",
5         "fullName": "Juan Perez",
6         "id": 2
7     },
8     {
9         "authorType": "ORG",
10        "contactEmail": "contacto@editorial.com",
11        "id": 3,
12        "organizationName": "Editorial Andina"
13    }
14 ]
```

Figura 7. Evidencia Postman – listado de autores.

Obtener autor por id (GET /authors/{id}):

The screenshot shows a GET request to `http://localhost:8081/authors/2`. The response status is 200 OK, with a response time of 9 ms and a response size of 337 B. The response body is:

```
1 {
2     "authorType": "PERSON",
3     "email": "juan.perez@mail.com",
4     "fullName": "Juan Perez",
5     "id": 2
6 }
```

Figura 8. Evidencia Postman – obtener autor por id.



## 5. Microservicio Publications

### 5.1 Propósito y modelo de dominio

Publications Service gestiona publicaciones editoriales y su ciclo de vida. Implementa una entidad abstracta Publication con herencia SINGLE\_TABLE y una derivada ArticlePublication. El estado editorial se modela con el enum EditorialStatus: DRAFT, IN\_REVIEW, APPROVED, PUBLISHED y REJECTED.

### 5.2 Dependencia Publications → Authors (validación)

Al crear o actualizar una publicación, Publications valida que el authorId exista consultando Authors Service (HTTP GET /authors/{id}). La dependencia se implementa mediante la abstracción AuthorsClient (DIP). Se manejan los casos: autor inexistente (404) y Authors no disponible (503).

### 5.3 Endpoints

- POST /publications – crea una publicación asociada a un autor existente.
- GET /publications/{id} – consulta una publicación.
- GET /publications – lista publicaciones (filtros opcionales: authorId, status).
- PATCH /publications/{id}/status – cambia el estado editorial.
- PUT /publications/{id} – actualiza datos de la publicación (sin tocar status).
- DELETE /publications/{id} – elimina una publicación.

### 5.4 Ejemplos (request/response)

Crear publicación (POST /publications):

```
Request
{
  "publicationType": "ARTICLE",
  "authorId": 2,
  "title": "Tendencias en Arquitectura de Software",
  "content": "Contenido de ejemplo...",
  "category": "Software"
}
```

La respuesta se enriquece con datos del autor (si Authors responde correctamente).



```
POST http://localhost:8082/publications

Body Cookies Headers (8) Test Results (1/1) 201 Created 298 ms 593 B ⏱ JSON Preview Visualize

{
  "author": {
    "authorType": "PERSON",
    "contactEmail": null,
    "email": "juan.perez@mail.com",
    "fullName": "Juan Perez",
    "id": 2,
    "organizationName": null
  },
  "authorId": 2,
  "category": "Tecnologia",
  "content": "Contenido de prueba",
  "createdAt": "2026-02-04T17:50:12.445291183Z",
  "id": 1
}
```

Figura 9. Evidencia Postman – creación de publicación (respuesta enriquecida con autor).

Caso de error: authorId不存在 (POST /publications):

```
POST http://localhost:8082/publications

Body Cookies Headers (8) Test Results (0/1) 404 Not Found 27 ms 400 B ⏱ JSON Preview Debug with AI

{
  "status": 404,
  "error": "Not Found",
  "message": "Autor no existe: id=99999",
  "path": "/publications",
  "timestamp": "2026-02-04T17:50:56.345690949Z"
}
```

Figura 10. Evidencia Postman – validación de authorId (autor不存在).

Listar publicaciones con filtros opcionales (GET /publications?authorId=&status=):

```
GET http://localhost:8082/publications

Body Cookies Headers (8) Test Results (1/1) 200 OK 23 ms 587 B ⏱ JSON Preview Visualize

{
  "author": {
    "authorType": "PERSON",
    "contactEmail": null,
    "email": "juan.perez@mail.com",
    "fullName": "Juan Perez",
    "id": 2,
    "organizationName": null
  },
  "authorId": 2,
  "category": "Tecnologia",
  "content": "Contenido de prueba",
  "createdAt": "2026-02-04T17:50:12.445291183Z",
  "id": 1,
  "status": "PUBLISHED"
}
```

Figura 11. Evidencia Postman – listado de publicaciones.



Consultar detalle (GET /publications/{id}):

```
2 "author": {  
3     "authorType": "PERSON",  
4     "contactEmail": null,  
5     "email": "juan.perez@mail.com",  
6     "fullName": "Juan Perez",  
7     "id": 2,  
8     "organizationName": null  
9 },  
10    "authorId": 2,  
11    "category": "Tecnologia",  
12    "content": "Contenido de prueba",  
13    "createdAt": "2026-02-04T17:50:12.445291Z",  
14    "id": 1,
```

Figura 12. Evidencia Postman – detalle de publicación.

## 5.5 Reglas de transición de estado

Las transiciones permitidas se controlan con un validador (StatusTransitionValidator):

- DRAFT → IN REVIEW
- IN REVIEW → APPROVED o REJECTED
- APPROVED → PUBLISHED
- REJECTED → DRAFT (representa ruta de corrección/retrabajo)
- PUBLISHED – no permite cambios

Evidencia de cambios de estado (PATCH /publications/{id}/status):

```
1 { "status": "IN REVIEW" }  
2  
7     "id": 2,  
8     "organizationName": null  
9 },  
10    "authorId": 2,  
11    "category": "Tecnologia",  
12    "content": "Contenido de prueba",  
13    "createdAt": "2026-02-04T17:50:12.445291Z",  
14    "id": 1,  
15    "publicationType": "ARTICLE",  
16    "status": "IN REVIEW",  
17    "title": "Prueba desde Docker"  
18 }
```

Figura 13. Cambio de estado a IN REVIEW.



The screenshot shows a POSTMAN interface with a PATCH request to `http://localhost:8082/publications/1/status`. The body contains the JSON `{"status": "APPROVED"}`. The response is a `200 OK` with a duration of 21 ms and a size of 588 B. The response body is identical to the request body.

```
1 { "status": "APPROVED" }
2
7   "id": 2,
8   "organizationName": null
9 },
10  "authorId": 2,
11  "category": "Tecnologia",
12  "content": "Contenido de prueba",
13  "createdAt": "2026-02-04T17:50:12.445291Z",
14  "id": 1,
15  "publicationType": "ARTICLE",
16  "status": "APPROVED",
17  "title": "Prueba desde Docker"
18 }
```

Figura 14. Cambio de estado a APPROVED.

The screenshot shows a POSTMAN interface with a PATCH request to `http://localhost:8082/publications/1/status`. The body contains the JSON `{"status": "PUBLISHED"}`. The response is a `200 OK` with a duration of 25 ms and a size of 589 B. The response body is identical to the request body.

```
1 { "status": "PUBLISHED" }
2
7   "id": 2,
8   "organizationName": null
9 },
10  "authorId": 2,
11  "category": "Tecnologia",
12  "content": "Contenido de prueba",
13  "createdAt": "2026-02-04T17:50:12.445291Z",
14  "id": 1,
15  "publicationType": "ARTICLE",
16  "status": "PUBLISHED",
17  "title": "Prueba desde Docker"
18 }
```

Figura 15. Cambio de estado a PUBLISHED.

## 6. Patrones de diseño documentados

Se documentan 4 patrones:

1. Repository: AuthorRepository y PublicationRepository abstraen el acceso a datos mediante Spring Data JPA.
2. Adapter: AuthorsClient (interfaz) + RestAuthorsClient (adaptador) encapsulan el consumo HTTP hacia Authors Service.
3. Factory Method (Factory): PublicationFactory crea instancias del dominio según publicationType (ej. ARTICLE).
4. Strategy/Rules: StatusTransitionValidator concentra y aplica reglas de transición (política de negocio).

Evidencia en código (archivos relevantes):



- authors-service: AuthorRepository.java, AuthorServiceImpl.java, AuthorMapper.java.
- publications-service: PublicationRepository.java, AuthorsClient.java, RestAuthorsClient.java, PublicationFactory.java, StatusTransitionValidator.java.

## 7. Evidencia de principios SOLID

### 7.1 SRP (Single Responsibility)

Cada capa cumple una responsabilidad: controllers exponen endpoints, services concentran reglas, repositories persisten, mappers transforman entidades ↔ DTOs y exceptions estandarizan errores.

### 7.2 DIP (Dependency Inversion)

Publications depende de la abstracción AuthorsClient y no de una implementación concreta. Esto permite cambiar la forma de consumo del servicio (por ejemplo, Feign) sin modificar la lógica de negocio.

### 7.3 OCP (Open/Closed)

El uso de PublicationFactory + herencia permite introducir nuevos tipos de publicación (ej. BOOK) con cambios acotados, manteniendo estable el contrato principal.

### 7.4 LSP e ISP

Las clases derivadas (PersonAuthor, OrganizationAuthor, ArticlePublication) son sustituibles por sus tipos base abstractos. Las interfaces expuestas (AuthorService, AuthorsClient) se mantienen pequeñas y enfocadas.

### 7.5 Observación de mejora

En Publications, EditorialStatus.valueOf(...) puede lanzar IllegalArgumentException si llega un estado inválido. Se recomienda capturarlo y mapearlo a 400 Bad Request en GlobalExceptionHandler para mantener manejo consistente de errores.

## 8. Proceso BPMN en Camunda (Proceso editorial)

Se modeló el flujo editorial con tres roles (Autor, Revisor y Editor). El proceso contempla la creación del borrador, envío a revisión, revisión editorial, decisión (aprobación/rechazo/solicitar cambios) y el ciclo de corrección y reenvío.

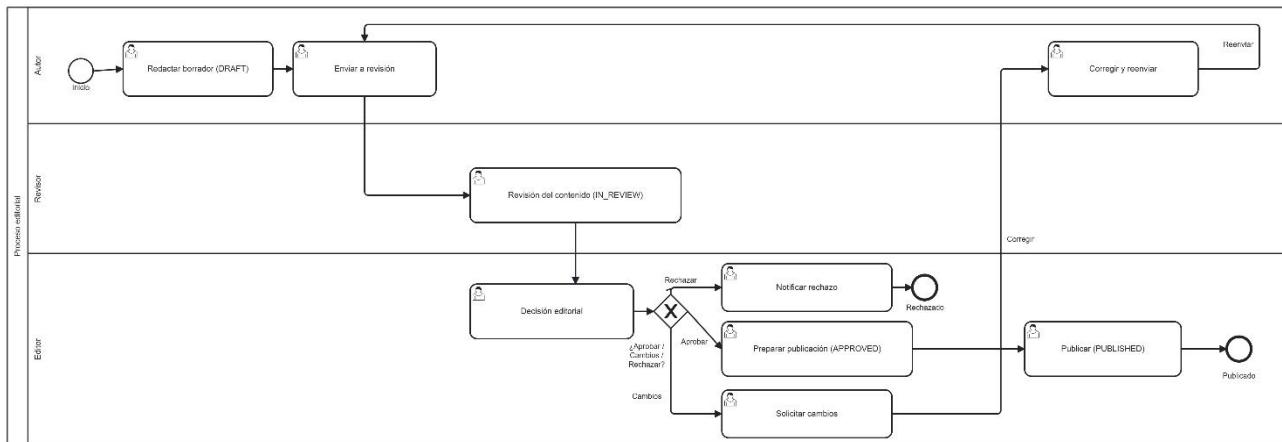


Figura 16. Diagrama BPMN del proceso editorial (Camunda Modeler).

## 9. Frontend (React + Vite)

El frontend se desarrolló con React + TypeScript y PrimeReact. Consuma las APIs mediante variables de entorno y ofrece pantallas para: crear/listar autores, crear/listar publicaciones, cambiar estado editorial y consultar detalle.

### 9.1 Variables de entorno

```
# frontend/.env
VITE_AUTHORS_API=http://localhost:8081
VITE_PUBLICATIONS_API=http://localhost:8082
```

### 9.2 Evidencia de pantallas

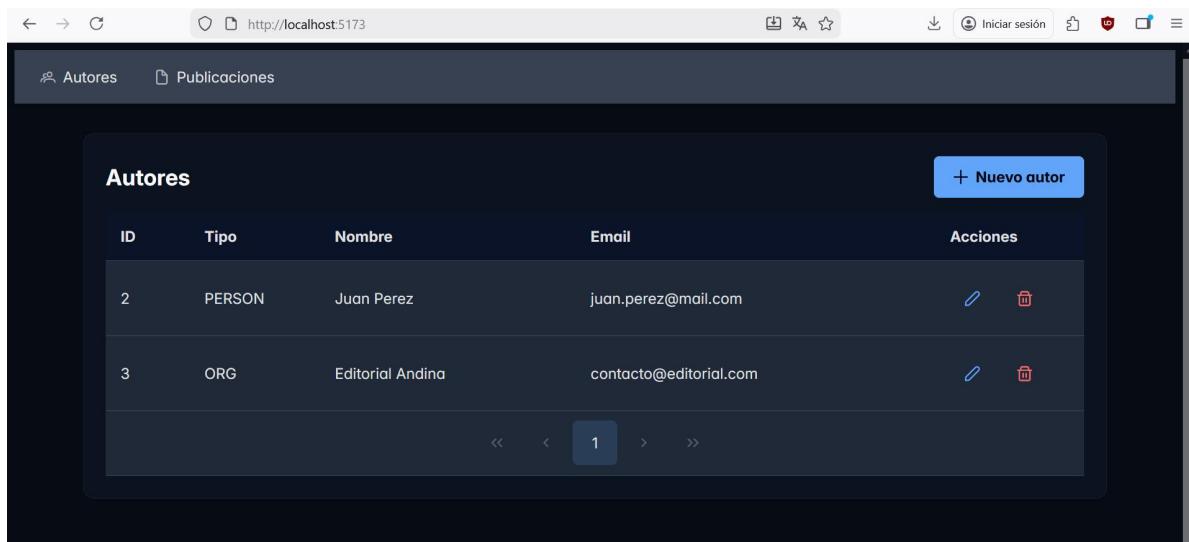


Figura 17. Frontend – pantalla de autores.



The screenshot shows a web application interface for managing publications. At the top, there are two navigation links: 'Autores' and 'Publicaciones'. Below this, a title 'Publicaciones' is displayed, along with two dropdown filters: 'Filtrar por autor' (Todos) and 'Filtrar por status' (Todos). A blue button '+ Nueva publicación' is located in the top right corner. The main area contains a table with columns: ID, Título, AuthorId, Estado, and Acciones. One row is visible, showing ID 1, Title 'Prueba desde Docker', AuthorId 2, Status 'PUBLISHED', and a set of icons for edit and delete. At the bottom, there is a pagination control with buttons for <<, <, 1, >, and >>.

Figura 18. Frontend – pantalla de publicaciones.

## 10. Despliegue con Docker Compose (obligatorio)

La guía solicita ejecución reproducible con Docker Compose (incluyendo authors-service, publications-service, frontend y una base por microservicio). A continuación se incluye un ejemplo de docker-compose.yml (referencial) que cumple el ecosistema.

### 10.1 Ejemplo de docker-compose.yml

```
version: "3.9"
services:
  db-authors:
    image: postgres:16
    container_name: db-authors
    environment:
      POSTGRES_DB: authors_db
      POSTGRES_USER: authors_user
      POSTGRES_PASSWORD: authors_pass
    ports:
      - "5433:5432"
    volumes:
      - authors_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U authors_user -d authors_db"]
      interval: 10s
      timeout: 5s
      retries: 10

  db-publications:
    image: postgres:16
    container_name: db-publications
```



environment:

```
POSTGRES_DB: publications_db
POSTGRES_USER: publications_user
POSTGRES_PASSWORD: publications_pass
```

ports:

```
- "5434:5432"
```

volumes:

```
- publications_data:/var/lib/postgresql/data
```

healthcheck:

```
test: ["CMD-SHELL", "pg_isready -U publications_user -d publications_db"]
```

```
interval: 10s
```

```
timeout: 5s
```

```
retries: 10
```

authors-service:

```
build: ./authors-service
```

```
container_name: authors-service
```

environment:

```
SPRING_DATASOURCE_URL: jdbc:postgresql://db-authors:5432/authors_db
```

```
SPRING_DATASOURCE_USERNAME: authors_user
```

```
SPRING_DATASOURCE_PASSWORD: authors_pass
```

```
SPRING_JPA_HIBERNATE_DDL_AUTO: update
```

ports:

```
- "8081:8081"
```

depends\_on:

```
db-authors:
```

```
condition: service_healthy
```

publications-service:

```
build: ./publications-service
```

```
container_name: publications-service
```

environment:

```
SPRING_DATASOURCE_URL: jdbc:postgresql://db-publications:5432/publications_db
```

```
SPRING_DATASOURCE_USERNAME: publications_user
```

```
SPRING_DATASOURCE_PASSWORD: publications_pass
```

```
SPRING_JPA_HIBERNATE_DDL_AUTO: update
```

```
AUTHORS_BASE_URL: http://authors-service:8081
```

ports:

```
- "8082:8082"
```

depends\_on:

```
db-publications:
```

```
condition: service_healthy
```

```
authors-service:
```

```
condition: service_started
```



```
frontend:  
  build: ./frontend  
  container_name: frontend  
  environment:  
    VITE_AUTHORS_API: http://localhost:8081  
    VITE_PUBLICATIONS_API: http://localhost:8082  
  ports:  
    - "5173:5173"  
  depends_on:  
    - authors-service  
    - publications-service  
  
volumes:  
  authors_data:  
  publications_data:
```

## 10.2 Comandos y URLs

```
# Levantar todo el ecosistema  
docker compose up --build
```

```
# Detener  
docker compose down
```

```
# Detener y borrar volúmenes (limpieza total)  
docker compose down -v
```

- Frontend: <http://localhost:5173>
- Authors API: <http://localhost:8081/authors>
- Publications API: <http://localhost:8082/publications>

## 10.3 Evidencia de contenedores en ejecución

The screenshot shows the Docker Desktop interface with the 'Containers' tab selected. It displays five running containers with their names, IDs, images, ports, and CPU usage. The containers are:

Name	Container ID	Image	Port(s)	CPU (%)
actividadad3	-	-	-	7.11%
frontend	43a0e650ac64	actividadad3-frontend	5173:80	0%
publications-s	947bc62a40b3	actividadad3-publication	8082:8082	0.15%
authors-servi	49f49172ebb5	actividadad3-authors-se	8081:8081	0.14%
db-publicatio	5008a762bc93	postgres:16	5434:5432	3.42%

Container memory usage: 703.6MB / 7.48GB

Figura 19. Evidencia Docker Desktop – contenedores levantados.

The screenshot shows the Docker Desktop interface with the 'Containers' tab selected. It displays five running containers with their names, IDs, images, ports, and CPU usage. The containers are:

Name	Container ID	Image	Port(s)	CPU (%)
publications-s	947bc62a40b3	actividadad3-publication	8082:8082	0.56%
authors-servi	49f49172ebb5	actividadad3-authors-se	8081:8081	0.25%
db-publicatio	5008a762bc93	postgres:16	5434:5432	4.29%
db-authors	fc606344f787	postgres:16	5433:5432	4.25%

Container memory usage: 695.11MB / 7.48GB

Figura 20. Evidencia Docker Desktop – bases de datos y puertos.

## 11. Conclusiones

La solución implementa una arquitectura de microservicios con persistencia independiente por servicio y comunicación REST sin dependencia circular. Se evidencia el uso de herencia (clase abstracta + derivadas) en ambos microservicios, aplicación de SOLID por separación de capas y desacoplamiento mediante interfaces, y patrones de diseño documentados (Repository, Adapter,



Factory, Rules). El proceso editorial fue modelado en BPMN y está listo para simular escenarios con Token Simulation.

## **12. Bibliografía**

- Camunda Services GmbH. (n.d.). Camunda 7 documentation. Camunda Docs. <https://docs.camunda.org/>
- Docker Inc. (n.d.). Docker Compose overview. Docker Documentation. <https://docs.docker.com/compose/>
- PostgreSQL Global Development Group. (n.d.). PostgreSQL 16 documentation. PostgreSQL Documentation. <https://www.postgresql.org/docs/16/>
- VMware, Inc. (n.d.). Spring Boot reference documentation. Spring Documentation. <https://docs.spring.io/spring-boot/docs/current/reference/html/>

### **LINK GITHUB:**

<https://github.com/dpadilla883/actividad3-padilla-david-microservicios-bpmn-frontend.git>