



Nombres y apellidos: Oscar Domínguez

Asignatura: Arquitectura de software

Link del repositorio:

<https://github.com/OscarAdrian2001/microservicios-editorial>

ACTIVIDAD 3

Introducción:

S realizo un sistema editorial basado en una arquitectura de microservicios, cuyo objetivo es gestionar autores y publicaciones de manera independiente pero integrada. La solución está compuesta por dos microservicios principales: Authors Service y Publications Service, los cuales interactúan entre sí mediante comunicación HTTP.

Objetivos

Diseñar e implementar un sistema editorial utilizando una arquitectura basada en microservicios, integrando backend, frontend, contenedores Docker y modelado de procesos BPMN, con el fin de aplicar los conceptos fundamentales de Arquitectura de Software.

Objetivos Específicos

- Implementar microservicios para la gestión de autores y publicaciones, utilizando Node.js, Express y TypeScript.
- Establecer la comunicación entre microservicios mediante llamadas HTTP.
- Utilizar Docker y Docker Compose para la orquestación y despliegue de los servicios del sistema.
- Desarrollar un frontend web que permita consumir los microservicios y realizar operaciones básicas de gestión.
- Modelar el proceso editorial mediante BPMN 2.0, representando los flujos de aprobación y rechazo de publicaciones.
- Realizar la simulación del proceso utilizando Token Simulation para validar el comportamiento del modelo BPMN.
- Documentar la arquitectura, las pruebas realizadas y los resultados obtenidos durante el desarrollo del proyecto.

Descripción general

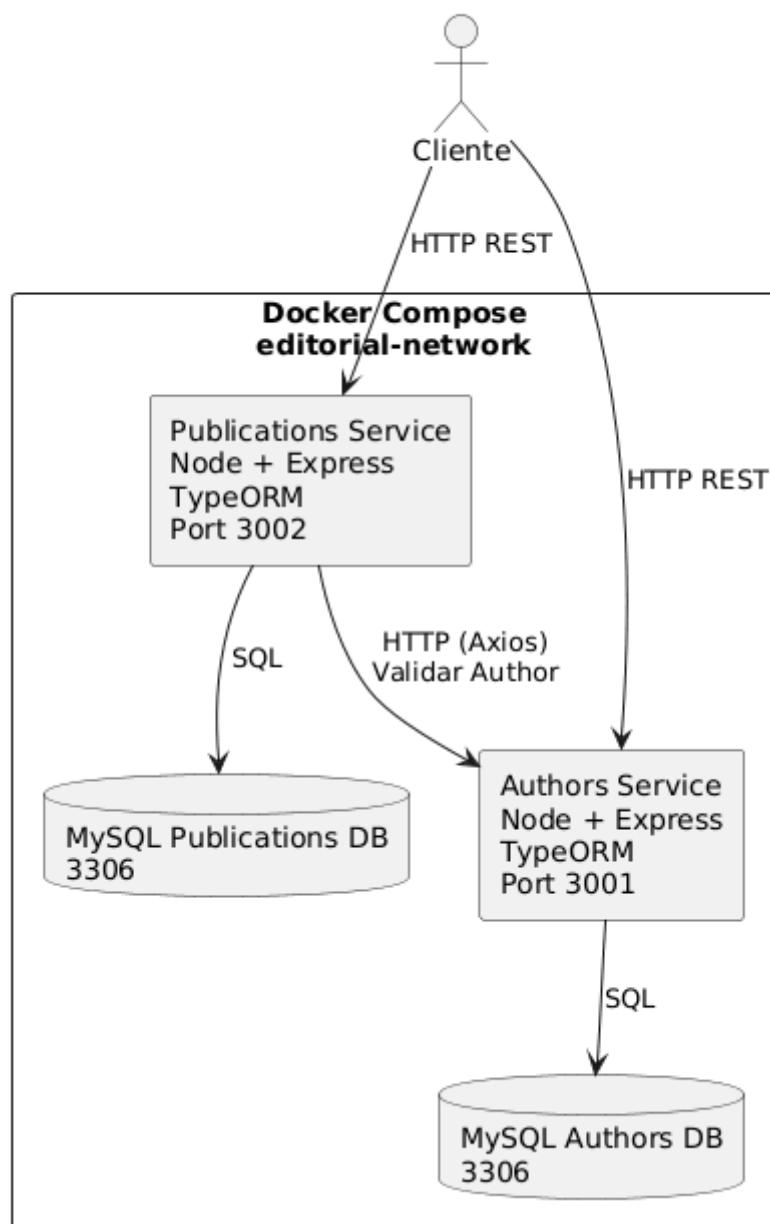
Se implementó una arquitectura de microservicios compuesta por:

- Authors Service
- Publications Service
- Bases de datos independientes
- Comunicación HTTP REST entre servicios
- Contenerización con Docker Compose

Cada microservicio tiene su propia base de datos y se comunica mediante API REST.

Diagrama de Arquitectura

Arquitectura — Sistema de Microservicios Editorial



El sistema se compone de dos microservicios: Authors Service y Publications Service, Cada microservicio cuenta con su propia lógica de negocio y su propio BD, lo que facilita la comunicación entre los microservicios donde las peticiones se realizan a través de HTTP, permitiendo que el microservicio de publicaciones valide la existencia de un autor antes de registrar una publicación.

Por su parte, el acceso al sistema se realiza a través de un frontend, el cual consume los microservicios para realizar operaciones como crear y listar autores y publicaciones entre otros. Por tanto, la ejecución y el despliegue del sistema, se utiliza Docker Compose, que permite levantar todos los servicios y bases de datos de forma conjunta y controlada.

Backend

Authors Service

Creamos el proyecto en nuestra carpeta principal

```
PS C:\espe1\workspace\microservicios-editorial> mkdir authors-service
• PS C:\espe1\workspace\microservicios-editorial> cd authors-service
PS C:\espe1\workspace\microservicios-editorial\authors-service> npm init -y
• Wrote to C:\espe1\workspace\microservicios-editorial\authors-service\package.json:
```

Instalamos dependencias necesarias

```
PS C:\espe1\workspace\microservicios-editorial\authors-service> npm install express typeorm mysql2 reflect-metadata axios
•
found 0 vulnerabilities
• PS C:\espe1\workspace\microservicios-editorial\authors-service> npm install -D typescript ts-node-dev @types/node @types/express
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a goo
```

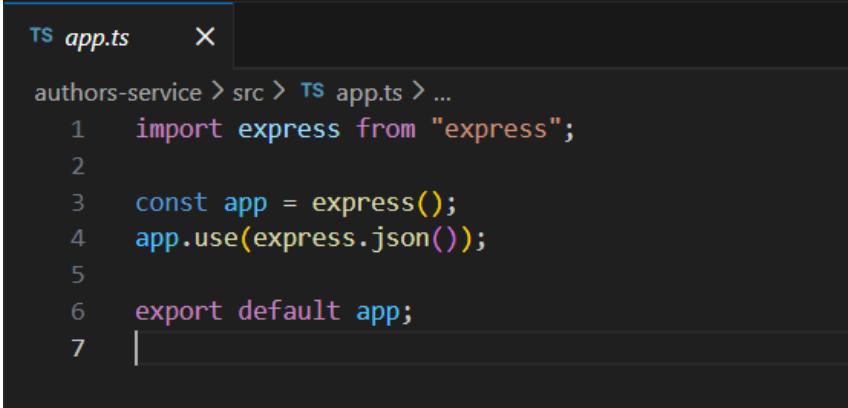


```
tsconfig.json X
authors-service > tsconfig.json > ...
1  {
2    "compilerOptions": {
3      "target": "ES2020",
4      "module": "commonjs",
5      "outDir": "dist",
6      "rootDir": "src",
7      "strict": true,
8      "esModuleInterop": true,
9      "emitDecoratorMetadata": true,
10     "experimentalDecorators": true
11   }
12 }
13 }
```

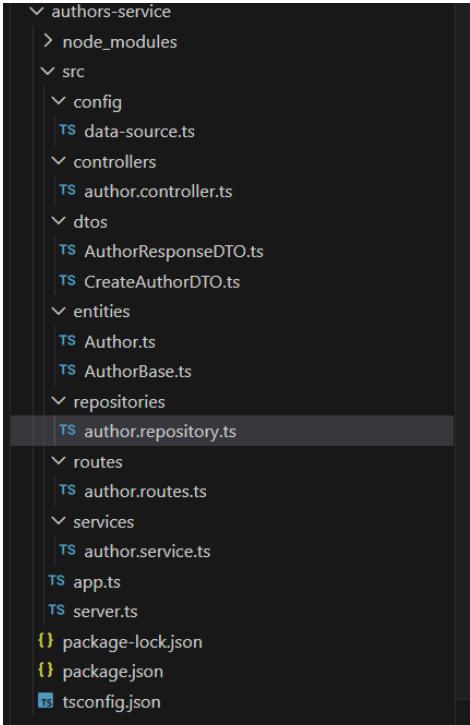
Responsabilidad

Gestiona la información de autores.

Estructura interna del microservicio



```
TS app.ts X
authors-service > src > TS app.ts > ...
1 import express from "express";
2
3 const app = express();
4 app.use(express.json());
5
6 export default app;
7 |
```



```
▽ authors-service
  > node_modules
  ▽ src
    ▽ config
      TS data-source.ts
    ▽ controllers
      TS author.controller.ts
    ▽ dtos
      TS AuthorResponseDTO.ts
      TS CreateAuthorDTO.ts
    ▽ entities
      TS Author.ts
      TS AuthorBase.ts
    ▽ repositories
      TS author.repository.ts
    ▽ routes
      TS author.routes.ts
    ▽ services
      TS author.service.ts
    TS app.ts
    TS server.ts
  {} package-lock.json
  {} package.json
  tsconfig.json
```

Arquitectura aplicada

Se implementó arquitectura por capas:

- Controller: maneja HTTP

```
ts author.controller.ts X ts author.service.ts TS AuthorBase.ts
authors-service > src > controllers > ts author.controller.ts > ...
6  export class AuthorController {
7
8    // Crear autor
9    async create(req: Request, res: Response) {
10      try {
11        const author = await authorService.createAuthor(req.body);
12        return res.status(201).json(author);
13      } catch (error: any) {
14        return res.status(400).json({
15          message: error.message || "Error creating author"
16        });
17      }
18    }
19
20    // Obtener autor por ID
21    async getById(req: Request, res: Response) {
22      const id = Number(req.params.id);
23
24      if (isNaN(id)) {
25        return res.status(400).json({
26          message: "Invalid ID"
27        });
28      }
29
30      try {
31        const author = await authorService.getAuthorById(id);
32
33        if (!author) {
34          return res.status(404).json({
35            message: "Author not found"
36          });
37        }
38      }
39    }
40  }
```

➤ Service:lógica de negocio

```
ts author.service.ts X ts author.controller.ts
authors-service > src > services > ts author.service.ts > ...
5  export class AuthorService {
6    async createAuthor(data: CreateAuthorDTO): Promise<Author> {
7
8      }
9
10     async getAuthorById(id: number): Promise<Author | null> {
11       return await AuthorRepository.findOneBy({ id });
12     }
13
14     async getAllAuthors(): Promise<Author[]> {
15       return await AuthorRepository.find();
16     }
17   }
18 }
```

➤ Repository:acceso a datos

```
TS author.repository.ts X  
authors-service > src > repositories > TS author.repository.ts > ...  
1 import { AppDataSource } from "../config/data-source";  
2 import { Author } from "../entities/Author";  
3  
4 export const AuthorRepository = AppDataSource.getRepository(Author);  
5
```

➤ Entity: modelo ORM

```
TS Author.ts X  
authors-service > src > entities > TS Author.ts > ...  
1 import { Entity, Column } from "typeorm";  
2 import { AuthorBase } from "./AuthorBase";  
3  
4 @Entity()  
5 export class Author extends AuthorBase {  
6  
7   @Column({ nullable: true })  
8   biography?: string;  
9 }  
10
```

```
TS AuthorBase.ts X  
authors-service > src > entities > TS AuthorBase.ts > ...  
1 import { PrimaryGeneratedColumn, Column } from "typeorm";  
2  
3 export abstract class AuthorBase {  
4  
5   @PrimaryGeneratedColumn()  
6   id!: number;  
7  
8   @Column({ nullable: false })  
9   name!: string;  
10  
11  @Column({ unique: true, nullable: false })  
12  email!: string;  
13 }  
14
```

Patrones usados

➤ Repository Pattern

```
TS data-source.ts TS author.repository.ts X TS CreateAuthorDTO.ts TS AuthorResponse  
authors-service > src > repositories > TS author.repository.ts > ...  
1 import { AppDataSource } from "../config/data-source";  
2 import { Author } from "../entities/Author";  
3  
4 export const AuthorRepository = AppDataSource.getRepository(Author);  
5
```

➤ DTO Pattern

```
TS CreateAuthorDTO.ts ×  
authors-service > src > dtos > TS CreateAuthorDTO.ts > ...  
1  export interface CreateAuthorDTO {  
2    name: string;  
3    email: string;  
4    biography?: string;  
5  }  
6
```

```
TS AuthorResponseDTO.ts ×  
authors-service > src > dtos > TS AuthorResponseDTO.ts > ...  
1  export interface AuthorResponseDTO {  
2    id: number;  
3    name: string;  
4    email: string;  
5    biography?: string;  
6  }
```

➤ SRP (SOLID)

```
TS author.service.ts ×  
authors-service > src > services > TS author.service.ts > ...  
1  import { AuthorRepository } from "../repositories/author.repository";  
2  import { CreateAuthorDTO } from "../dtos/CreateAuthorDTO";  
3  import { Author } from "../entities/Author";  
4  
5  export class AuthorService {  
6  
7    async createAuthor(data: CreateAuthorDTO): Promise<Author> {  
8      if (!data.name || data.name.trim() === "") {  
9        throw new Error("Name is required");  
10     }  
11  
12     if (!data.email || data.email.trim() === "") {  
13       throw new Error("Email is required");  
14     }  
15  
16     const existingAuthor = await AuthorRepository.findOne({  
17       where: { email: data.email }  
18     });  
19  
20     if (existingAuthor) {  
21       throw new Error("Author with this email already exists");  
22     }  
23  
24     const author = AuthorRepository.create(data);  
25     return await AuthorRepository.save(author);  
26   }  
27 }
```

```

27
28     async getAuthorById(id: number) {
29         return await AuthorRepository.findOne({
30             where: { id }
31         });
32     }
33
34     async getAllAuthors() {
35         return await AuthorRepository.find();
36     }
37 }

```

Evidencia:

Endpoints

- POST /authors

POST http://localhost:3001/api/authors

Body (raw JSON)

```

1 {
2   "name": "Julio Cortázar",
3   "email": "jc@literatura.com"
4 }
5

```

201 Created

```

1 {
2   "name": "Julio Cortázar",
3   "email": "jc@literatura.com",
4   "biography": null,
5   "id": 1
6 }

```

GET /authors

HTTP http://localhost:3001/api/authors

GET http://localhost:3001/api/authors

Body (JSON)

```

1 [
2   ...
3   {
4     "id": 1,
5     "name": "Julio Cortázar",
6     "email": "jc@literatura.com",
7     "biography": null
8   }
9 ]

```

200 OK

```

1 [
2   ...
3   {
4     "id": 1,
5     "name": "Julio Cortázar",
6     "email": "jc@literatura.com",
7     "biography": null
8   }
9 ]

```

- GET /authors/:id

HTTP <http://localhost:3001/api/authors/1>

GET <http://localhost:3001/api/authors/1>

Send

Docs Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (7) Test Results

200 OK 148 ms 313 B

{ } JSON Preview Visualize

```
1 {  
2   "id": 1,  
3   "name": "Julio Cortázar",  
4   "email": "jc@literatura.com",  
5   "biography": null  
6 }
```

Prueba: ingresar datos incompletos

HTTP <http://localhost:3001/api/authors>

POST <http://localhost:3001/api/authors>

Send

Docs Params Authorization Headers (8) Body Scripts Tests Settings Cookies

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

Beautify

```
1 {  
2   "name": "Jorge Icaza",  
3   "email": "",  
4   "biography": "Escritor ecuatoriano"  
5 }  
6
```

Body Cookies Headers (7) Test Results

400 Bad Request 6 ms 275 B

{ } JSON Preview Debug with AI

```
1 {  
2   "message": "Email is required"  
3 }
```

HTTP <http://localhost:3001/api/authors>

POST <http://localhost:3001/api/authors>

Send

Docs Params Authorization Headers (8) Body Scripts Tests Settings Cookies

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

Beautify

```
1 {  
2   "name": "",  
3   "email": "jorgeo@example.com",  
4   "biography": "Escritor ecuatoriano"  
5 }  
6
```

Body Cookies Headers (7) Test Results

400 Bad Request 6 ms 274 B

{ } JSON Preview Debug with AI

```
1 {  
2   "message": "Name is required"  
3 }
```

HTTP <http://localhost:3001/api/authors>

POST <http://localhost:3001/api/authors>

Body JSON raw binary GraphQL JSON

```

1 {
2   "name": "Jorge Icaza",
3   "email": "jorgeo@example.com",
4   "biography": "Escritor ecuatoriano"
5 }
6

```

400 Bad Request 7 ms 295 B

```

1 {}
2 |   "message": "Author with this email already exists"
3 {}

```

Prueba: Obtener autor por ID

HTTP <http://localhost:3001/api/authors/5>

GET <http://localhost:3001/api/authors/5>

Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

404 Not Found 8 ms 272 B

```

1 {}
2 |   "message": "Author not found"
3 {}

```

Publications Service

Creamos el proyecto en nuestra carpeta principal

```
PS C:\espe1\workspace\microservicios-editorial> mkdir publications-service
Mode          LastWriteTime    Length Name
----          -----        -----
d----
```

Mode	LastWriteTime	Length	Name
d----	26/1/2026 22:12		publications-service

Instalamos dependencias necesarias

```
PS C:\espe1\workspace\microservicios-editorial\publications-service> npm init -y
PS C:\espe1\workspace\microservicios-editorial\publications-service> npm install express typeorm mysql2 reflect-metadata axios
added 173 packages, and audited 174 packages in 24s
55 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\espe1\workspace\microservicios-editorial\publications-service> npm install -D typescript ts-node-dev @types/node @types/express
```

Configuración del TypeScript

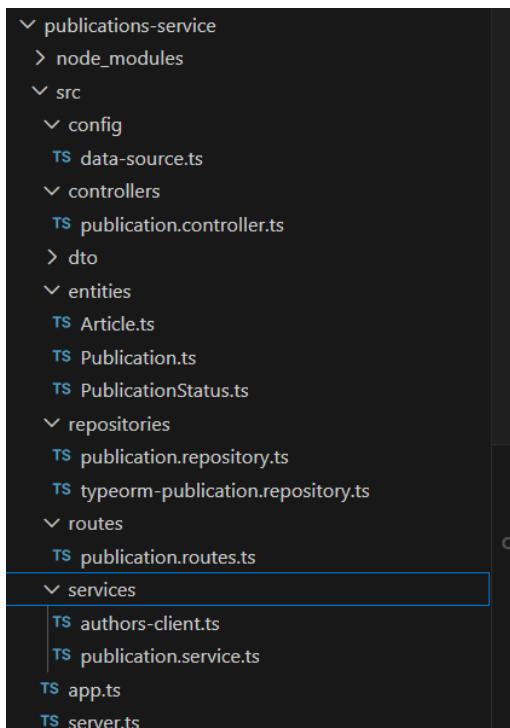


```
tsconfig.json X
publications-service > tsconfig.json > ...
1  {
2    "compilerOptions": {
3      "target": "ES2020",
4      "module": "commonjs",
5      "rootDir": "src",
6      "outDir": "dist",
7      "esModuleInterop": true,
8      "strict": true,
9      "skipLibCheck": true
10     }
11 }
```

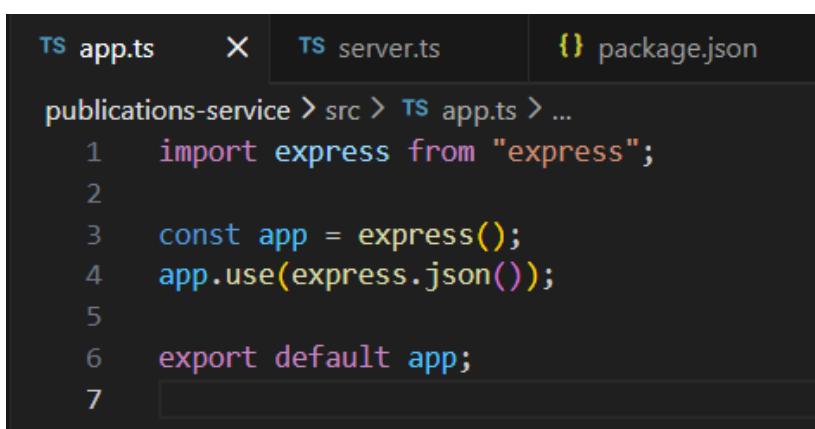
Responsabilidad

Gestiona publicaciones editoriales.

Estructura interna del microservicio



```
publications-service
  node_modules
  src
    config
      data-source.ts
    controllers
      publication.controller.ts
    dto
    entities
      Article.ts
      Publication.ts
      PublicationStatus.ts
    repositories
      publication.repository.ts
      typeorm-publication.repository.ts
    routes
      publication.routes.ts
    services
      authors-client.ts
      publication.service.ts
    app.ts
    server.ts
```



```
app.ts X server.ts package.json
publications-service > src > app.ts > ...
1 import express from "express";
2
3 const app = express();
4 app.use(express.json());
5
6 export default app;
```

Ejecución del servicio

```
TS server.ts X
publications-service > src > TS server.ts > ...
1 import { TypeOrmModule } from '@nestjs/typeorm';
2 import { app } from './app';
3
4 const PORT = 3002;
5
6 app.listen(PORT, () => {
7   console.log(`Publications Service running on port ${PORT}`);
8 })
```

Dominio implementado

Se modeló:

- Clase abstracta Publication

```
TS Publication.ts X | TS Article.ts
publications-service > src > entities > TS Publication.ts > ↗ Publication > ↘ updatedAt
1 export abstract class Publication {
2   @Column()
3   authorId!: number;
4
5   @Column({
6     type: "enum",
7     enum: PublicationStatus,
8     default: PublicationStatus.DRAFT
9   })
10  status!: PublicationStatus;
11
12  @CreateDateColumn()
13  createdAt!: Date;
14
15  @UpdateDateColumn()
16  updatedAt!: Date;
17}
```

- Clase concreta Article

```
TS Article.ts X
publications-service > src > entities > TS Article.ts > ...
1 import { Entity, Column } from "typeorm";
2 import { Publication } from "./Publication";
3
4 @Entity("articles")
5 export class Article extends Publication {
6
7   @Column({ default: "GENERAL" })
8   category!: string;
9 }
10
```

Agregamos esto el archivo: tsconfig.json

```
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true
}
}
```

- Enum PublicationStatus

TS PublicationStatus.ts X

publications-service > src > entities > TS PublicationStatus.ts > ...

```
1  export enum PublicationStatus {
2      DRAFT = "DRAFT",
3      IN_REVIEW = "IN_REVIEW",
4      APPROVED = "APPROVED",
5      PUBLISHED = "PUBLISHED",
6      REJECTED = "REJECTED"
7  }
8
```

Patrones aplicados

- Repository Pattern
- Adapter Pattern (AuthorsClient)
- SOLID (SRP + DIP)

Repository Pattern (TypeORM Repository)

- Aplicar Repository Pattern
- Encapsular el acceso a datos
- Evitar que Services usen TypeORM directamente
- Dejar base sólida para SOLID y pruebas

Creamos la interfaz del repositorio

TS publication.repository.ts X TS typeorm-publication.repository.ts

publications-service > src > repositories > TS publication.repository.ts > ...

```
1  import { Article } from "../entities/Article";
2  import { PublicationStatus } from "../entities/PublicationStatus";
3
4  export interface PublicationRepository {
5      create(publication: Article): Promise<Article>;
6      findById(id: number): Promise<Article | null>;
7      findAll(): Promise<Article[]>;
8      updateStatus(id: number, status: PublicationStatus): Promise<Article | null>;
9  }
10
```

Implementación con TypeORM

```
ts typeorm-publication.repository.ts X
publications-service > src > repositories > ts typeorm-publication.repository.ts > ...
0
7  export class TypeOrmPublicationRepository implements PublicationRepository {
8
9    private repository: Repository<Article>;
10
11   constructor() {
12     this.repository = AppDataSource.getRepository(Article);
13   }
14
15   async create(publication: Article): Promise<Article> {
16     return await this.repository.save(publication);
17   }
18
19   async findById(id: number): Promise<Article | null> {
20     return await this.repository.findOneBy({ id });
21   }
22
23   async findAll(): Promise<Article[]> {
24     return await this.repository.find();
25   }
26
27   async updateStatus(
28     id: number,
29     status: PublicationStatus
30   ): Promise<Article | null> {
31
32     const publication = await this.findById(id);
33     if (!publication) return null;
34
35     publication.status = status;
36     return await this.repository.save(publication);
37   }
38 }
39
```

Se aplicó el Repository Pattern para encapsular el acceso a datos y desacoplar la lógica de negocio del ORM utilizado

Lógica de Negocio más SOLID

Crear el servicio de publicaciones

```
ts publication.service.ts X
publications-service > src > services > ts publication.service.ts > ...
  5
  6  export class PublicationService {
  7
  8    private repository: PublicationRepository;
  9
 10   constructor() {
 11     this.repository = new TypeOrmPublicationRepository();
 12   }
 13
 14   async createPublication(data: {
 15     title: string;
 16     content: string;
 17     authorId: number;
 18     category?: string;
 19   }): Promise<Article> {
 20
 21     const article = new Article();
 22     article.title = data.title;
 23     article.content = data.content;
 24     article.authorId = data.authorId;
 25     article.category = data.category || "GENERAL";
 26     article.status = PublicationStatus.DRAFT;
 27
 28     return await this.repository.create(article);
 29   }
 30   async getPublicationById(id: number): Promise<Article | null> {
 31     return await this.repository.findById(id);
 32   }
 33
 34   async getAllPublications(): Promise<Article[]> {
 35     return await this.repository.findAll();
 36   }
 37
 38   async changeStatus(
 39     id: number,
 40     status: PublicationStatus
 41   ): Promise<Article | null> {
 42     return await this.repository.updateStatus(id, status);
 43   }
 44 }
 45 }
```

Qué SOLID estamos usando:

- SRP: El service solo maneja reglas de negocio
- DIP: Depende de PublicationRepository (interfaz), no de TypeORM
- OCP: Puedes cambiar la implementación del repositorio sin tocar el service

Validación entre servicios

Antes de crear una publicación Publications Service consulta Authors Service vía HTTP usando Axios.

Creamos el Adapter HTTP “cliente de Authors”

```
TS authors-client.ts X
publications-service > src > services > TS authors-client.ts > ...
1 import axios from "axios";
2
3 export class AuthorsClient {
4     private baseUrl = "http://localhost:3001/api/authors";
5
6     async authorExists(authorId: number): Promise<boolean> {
7         try {
8             await axios.get(`${this.baseUrl}/${authorId}`);
9             return true;
10        } catch (error) {
11            return false;
12        }
13    }
14}
15
```

Integramos el Adapter en PublicationService

```
TS publication.service.ts X
publications-service > src > services > TS publication.service.ts > PublicationService > constructor
6
7
8 export class PublicationService {
9
10    private repository: PublicationRepository;
11    private authorsClient: AuthorsClient;
12
13    constructor() {
14        this.repository = new TypeOrmPublicationRepository();
15        this.authorsClient = new AuthorsClient();
16    }
17
18    async createPublication(data: {
19        title: string;
20        content: string;
21        authorId: number;
22        category?: string;
23    }): Promise<Article> {
24
25        const exists = await this.authorsClient.authorExists(data.authorId);
26        if (!exists) {
27            throw new Error("Author does not exist");
28        }
29    }
30
```

```

30   const article = new Article();
31   article.title = data.title;
32   article.content = data.content;
33   article.authorId = data.authorId;
34   article.category = data.category || "GENERAL";
35   article.status = PublicationStatus.DRAFT;
36
37   return await this.repository.create(article);
38 }
39

```

Como se mencionó estamos usando el patrón Adapter Pattern. Donde AuthorsClient adapta HTTP y si cambia Authors, no rompes el service

Controller + Routes

Exponer endpoints REST para:

- Crear publicaciones
- Listar publicaciones
- Obtener una publicación por ID
- Cambiar estado de una publicación

Controller

ts	publication.controller.ts	x	ts	publication.routes.ts	ts	app.ts	ts	server.ts
----	---------------------------	---	----	-----------------------	----	--------	----	-----------

```

publications-service > src > controllers > ts publication.controller.ts > PublicationController
  1 < import { Request, Response } from "express";
  2 < import { PublicationService } from "../services/publication.service";
  3 < import { PublicationStatus } from "../entities/PublicationStatus";
  4
  5 const service = new PublicationService();
  6
  7 < export class PublicationController {
  8
  9   < async create(req: Request, res: Response) {
 10     < try {
 11       < const publication = await service.createPublication(req.body);
 12       < res.status(201).json(publication);
 13     } < catch (error: any) {
 14       < res.status(400).json({ message: error.message });
 15     }
 16   }
 17
 18   < async getAll(req: Request, res: Response) {
 19     < const publications = await service.getAllPublications();
 20     < res.json(publications);
 21   }
 22
 23   < async getById(req: Request, res: Response) {
 24     < const id = Number(req.params.id);
 25
 26     < if (isNaN(id)) {
 27       < return res.status(400).json({ message: "Invalid ID" });
 28     }
 29

```

```

30 |     const publication = await service.getPublicationById(id);
31 |
32 |     if (!publication) {
33 |         return res.status(404).json({ message: "Publication not found" });
34 |     }
35 |
36 |     res.json(publication);
37 |
38 |
39 |     async changeStatus(req: Request, res: Response) {
40 |         const id = Number(req.params.id);
41 |         const { status } = req.body;
42 |
43 |         if (!Object.values(PublicationStatus).includes(status)) {
44 |             return res.status(400).json({ message: "Invalid status" });
45 |         }
46 |
47 |         const updated = await service.changeStatus(id, status);
48 |
49 |         if (!updated) {
50 |             return res.status(404).json({ message: "Publication not found" });
51 |         }
52 |
53 |         res.json(updated);
54 |     }
55 |

```

Routes

publications-service > src > routes > **publication.routes.ts** > ...

```

1  import { Router } from "express";
2  import { PublicationController } from "../controllers/publication.controller";
3
4  const router = Router();
5  const controller = new PublicationController();
6
7  router.post("/", controller.create);
8  router.get("/", controller.getAll);
9  router.get("/:id", controller.getId);
10 router.patch("/:id/status", controller.changeStatus);
11
12 export default router;
13

```

Registrar rutas en app.ts

```

7
8  app.use("/publications", publicationRoutes);
9

```

Evidencia:

Se realizaron pruebas mediante Postman para validar el funcionamiento de los microservicios. El servicio de publicaciones verifica la existencia del autor consultando al Authors Service antes de crear una publicación.

HTTP <http://localhost:3002/publications>

POST <http://localhost:3002/publications>

Body raw binary GraphQL JSON

```

1 {
2   "title": "Rayuela",
3   "content": "Una novela distinta",
4   "authorId": 1,
5   "category": "NOVELA"
6 }

```

Body Cookies Headers (7) Test Results

{ } JSON ▾ ▶ Preview ▶ Visualize ▾

```

1 {
2   ...
3   "title": "Rayuela",
4   "content": "Una novela distinta",
5   "authorId": 1,
6   "category": "NOVELA",
7   "status": "DRAFT",
8   "id": 1,
9   "createdAt": "2026-01-28T03:20:20.574Z",
10  "updatedAt": "2026-01-28T03:20:20.574Z"
}

```

201 Created 120 ms 427 B

A su vez, si se rechaza la operación por que el autor no existe, tenemos que la publicación no se realizó ya que se intenta conectar a la id de un autor inexistente.

HTTP <http://localhost:3002/publications>

POST <http://localhost:3002/publications>

Body raw binary GraphQL JSON

```

1 {
2   "title": "Rayuela",
3   "content": "Una novela distinta",
4   "authorId": 5,
5   "category": "NOVELA"
6 }

```

Body Cookies Headers (7) Test Results

{ } JSON ▾ ▶ Preview ▶ Debug with AI ▾

```

1 {
2   ...
3   "message": "Author does not exist"
4 }

```

400 Bad Request 1.89 s 279 B

Podemos ver posteriormente el “Listar publicaciones”

GET <http://localhost:3002/publications>

Body Cookies Headers (7) Test Results

{ } JSON ▾ ▶ Preview ▶ Visualize ▾

```

1 [
2   ...
3   {
4     "id": 1,
5     "title": "Rayuela",
6     "content": "Una novela distinta",
7     "authorId": 1,
8     "status": "DRAFT",
9     "createdAt": "2026-01-28T03:20:20.574Z",
10    "updatedAt": "2026-01-28T03:20:20.574Z",
11    "category": "NOVELA"
12  }
]

```

200 OK 27 ms 424 B

Patrones de diseño aplicados

Repository Pattern

Encapsula acceso a base de datos.

Adapter Pattern

Implementado en AuthorsClient.

Adapta HTTP: método simple authorExists().

Persistencia

Cada microservicio tiene su propia base de datos MySQL.

- authors_db
- publications_db

Configuración de Base de Datos (TypeORM + MySQL)

- Configuramos TypeORM con DataSource
- Conectamos Authors Service a su propia base de datos

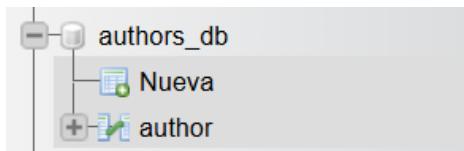
Tenemos el archivo: data-source.ts de “Authors-service” para la configuración de la BD

```
TS data-source.ts ×

authors-service > src > config > TS data-source.ts > [?] AppDataSource
  1 import "reflect-metadata";
  2 import { DataSource } from "typeorm";
  3 import { Author } from "../entities/Author";
  4
  5 export const AppDataSource = new DataSource({
  6   type: "mysql",
  7   host: process.env.DB_HOST || "localhost",
  8   port: Number(process.env.DB_PORT) || 3307,
  9   username: process.env.DB_USER || "root",
 10   password: process.env.DB_PASSWORD || "oscar",
 11   database: process.env.DB_NAME || "authors_db",
 12   synchronize: true,
 13   logging: false,
 14   entities: [Author],
 15 });
 16
```

Inicializamos la conexión al arrancar el servidor

```
ts server.ts X
authors-service > src > ts server.ts > ...
1 import app from "./app";
2 import { AppDataSource } from "./config/data-source";
3
4 const PORT = 3001;
5
6 AppDataSource.initialize()
7   .then(() => {
8     console.log(" Authors DB connected");
9     app.listen(PORT, () => {
10       console.log(` Authors Service running on port ${PORT}`);
11     });
12   })
13   .catch((error) => console.error("DB connection error:", error));
14
```



Configuración de Base de Datos para el servicio de “Publications-Service”

- Configuramos TypeORM con DataSource
- Conectamos Publications Service a su propia base de datos

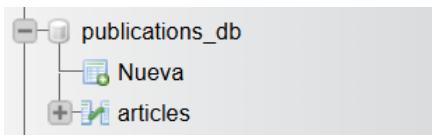
Tenemos el archivo: data-source.ts para la configuración de la BD

```
ts data-source.ts X
publications-service > src > config > ts data-source.ts > AppDataSource > logging
1 import "reflect-metadata";
2 import { DataSource } from "typeorm";
3 import { Article } from "../entities/Article";
4
5 export const AppDataSource = new DataSource({
6   type: "mysql",
7   host: process.env.DB_HOST || "localhost",
8   port: Number(process.env.DB_PORT) || 3307,
9   username: process.env.DB_USER || "root",
10  password: process.env.DB_PASSWORD || "oscar",
11  database: process.env.DB_NAME || "publications_db",
12
13  entities: [Article],
14  synchronize: true,
15  logging: false
16});
```

Inicializamos la conexión al arrancar el servidor

```
ts server.ts X
publications-service > src > ts server.ts > ...
1 import "reflect-metadata";
2 import app from "./app";
3 import { AppDataSource } from "./config/data-source";
4
5 const PORT = 3002;
6
7 AppDataSource.initialize()
8 .then(() => {
9   console.log("Publications DB connected");
10
11   app.listen(PORT, () => {
12     console.log(`Publications Service running on port ${PORT}`);
13   });
14 }
15 .catch((error) => {
16   console.error("DB connection error:", error);
17 });
18
```

- publications_db



Los servicios están configurados con TypeORM + DataSource. Ademas servirán para usar variables de entorno usadas en Docker.

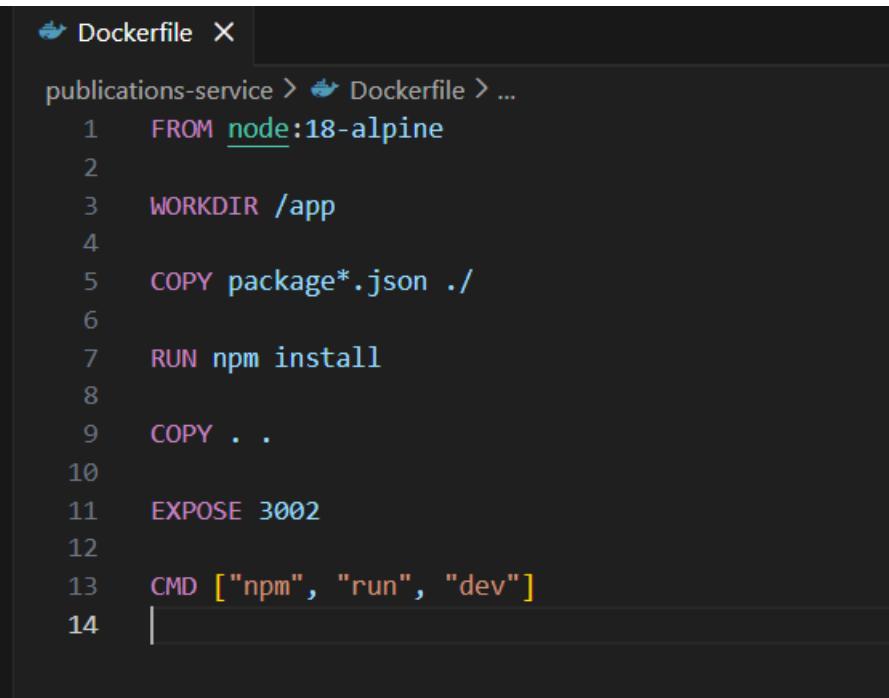
Contenerización con Docker

Dockerfile para cada microservicio

- Authors Service

```
dockerfile Dockerfile X
authors-service > dockerfile Dockerfile > ...
1 FROM node:18-alpine
2
3 # Directorio de trabajo dentro del contenedor
4 WORKDIR /app
5
6 # Copiamos dependencias
7 COPY package*.json ./
8
9 RUN npm install
10
11 # Copiamos el código
12 COPY .
13
14 # Compilamos TypeScript (si no usas build, se queda igual)
15 # RUN npm run build
16
17 EXPOSE 3001
18
19 CMD [ "npm", "run", "dev" ]
20 |
```

➤ Publications Service

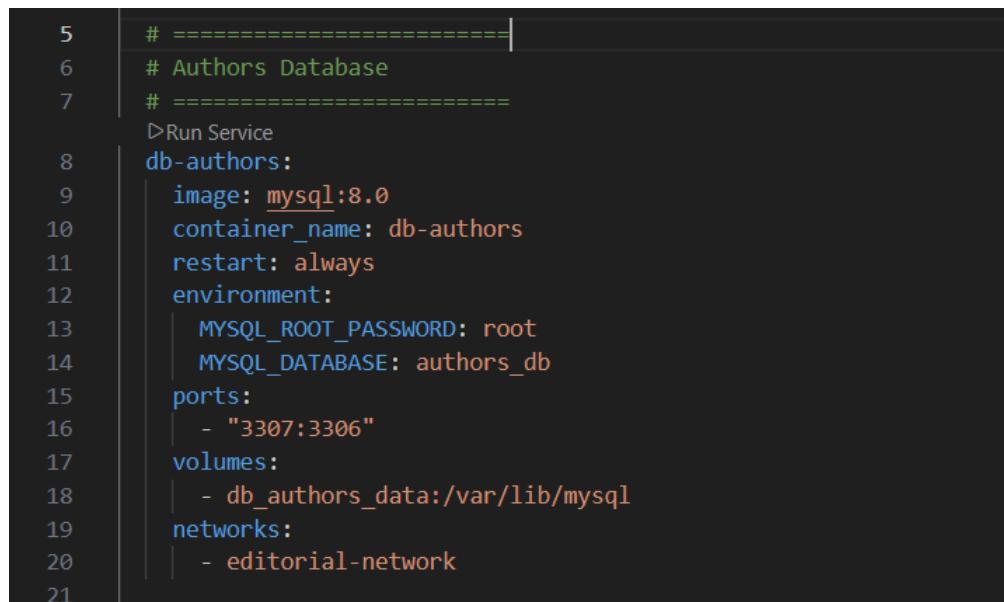


A screenshot of a code editor showing a Dockerfile. The file path is 'publications-service > Dockerfile > ...'. The Dockerfile content is as follows:

```
1 FROM node:18-alpine
2
3 WORKDIR /app
4
5 COPY package*.json ./
6
7 RUN npm install
8
9 COPY . .
10
11 EXPOSE 3002
12
13 CMD ["npm", "run", "dev"]
14
```

Evidencia:

Creamos el archivo docker-compose.yml



A screenshot of a code editor showing a partial docker-compose.yml file. The content is as follows:

```
5 # =====
6 # Authors Database
7 # =====
8 ▷Run Service
9 db-authors:
10   image: mysql:8.0
11   container_name: db-authors
12   restart: always
13   environment:
14     MYSQL_ROOT_PASSWORD: root
15     MYSQL_DATABASE: authors_db
16   ports:
17     - "3307:3306"
18   volumes:
19     - db_authors_data:/var/lib/mysql
20   networks:
21     - editorial-network
```

```

1 docker-compose.yml
2 services:
3   # Publications Database
4   # =====
5   db-publications:
6     image: mysql:8.0
7     container_name: db-publications
8     restart: always
9     environment:
10       MYSQL_ROOT_PASSWORD: root
11       MYSQL_DATABASE: publications_db
12     ports:
13       - "3308:3306"
14     volumes:
15       - db_publications_data:/var/lib/mysql
16     networks:
17       - editorial-network
18
19   # Authors Service
20   # =====
21   authors-service:
22     build: ./authors-service
23     container_name: authors-service
24     restart: always
25     ports:
26       - "3001:3001"
27     environment:
28       DB_HOST: db-authors
29       DB_PORT: 3306
30       DB_USER: root
31       DB_PASSWORD: root
32       DB_NAME: authors_db
33     depends_on:
34       - db-authors
35     networks:
36
37   # Publications Service
38   # =====
39   publications-service:
40     build: ./publications-service
41     container_name: publications-service
42     restart: always
43     ports:
44       - "3002:3002"
45     environment:
46       DB_HOST: db-publications
47       DB_PORT: 3306
48       DB_USER: root
49       DB_PASSWORD: root
50       DB_NAME: publications_db
51     AUTHORS_SERVICE_URL: http://authors-service:3001/api/authors
52     depends_on:
53       - db-publications
54       - authors-service
55     networks:
56       - editorial-network
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80

```

Ajustar data-source.ts para Docker donde tendremos env + host + puerto

Authors Service

```
TS data-source.ts X

authors-service > src > config > TS data-source.ts > [?] AppDataSource
1 import "reflect-metadata";
2 import { DataSource } from "typeorm";
3 import { Author } from "../entities/Author";
4
5 export const AppDataSource = new DataSource({
6   type: "mysql",
7   host: process.env.DB_HOST || "localhost",
8   port: Number(process.env.DB_PORT) || 3307,
9   username: process.env.DB_USER || "root",
10  password: process.env.DB_PASSWORD || "oscar",
11  database: process.env.DB_NAME || "authors_db",
12  synchronize: true,
13  logging: false,
14  entities: [Author],
15});
16
```

Publications Service

```
TS data-source.ts X

publications-service > src > config > TS data-source.ts > [?] AppDataSource
1 import "reflect-metadata";
2 import { DataSource } from "typeorm";
3 import { Article } from "../entities/Article";
4
5 export const AppDataSource = new DataSource({
6   type: "mysql",
7   host: process.env.DB_HOST || "localhost",
8   port: Number(process.env.DB_PORT) || 3307,
9   username: process.env.DB_USER || "root",
10  password: process.env.DB_PASSWORD || "oscar",
11  database: process.env.DB_NAME || "publications_db",
12  synchronize: true,
13  logging: false,
14  entities: [Article],
15});
16
```

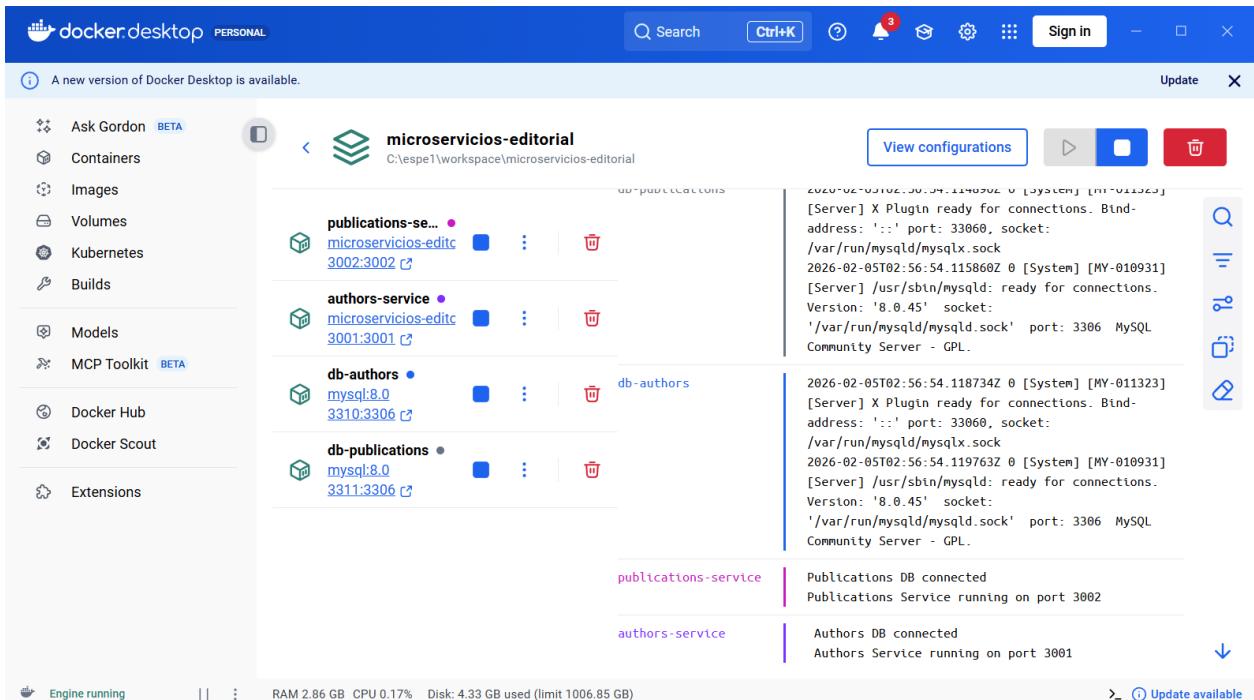
La configuración de la base de datos se realizó mediante variables de entorno, permitiendo que los microservicios funcionen tanto en entorno local como en contenedores Docker.

Levantar todo con Docker Compose: docker compose up --build

```
PS C:\espe1\workspace\microservicios-editorial> docker compose up --build
publications-service | Publications DB connected
publications-service | Publications Service running on port 3002

authors-service      | [INFO] 02:57:55 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.2, typescript ver. 5.9.3)
authors-service      | 📜 Authors DB connected
authors-service      | 🚀 Authors Service running on port 3001
```

Vemos en Docker los servicios conectados



Pruebas finales con Postman

Crear autor

HTTP <http://localhost:3001/api/authors> Save Share </>

POST <http://localhost:3001/api/authors> Send <*>

Docs Params Authorization Headers (8) Body Scripts Tests Settings Cookies Schema Beautify

none form-data x-www-form-urlencoded raw binary GraphQL [JSON](#)

```
1 {  
2   "name": "Julio Cortázar",  
3   "email": "jc@literatura.com"  
4 }  
5
```

Body Cookies Headers (7) Test Results

201 Created 2.18 s 318 B

```
1 {  
2   "name": "Julio Cortázar",  
3   "email": "jc@literatura.com",  
4   "biography": null,  
5   "id": 1  
6 }
```

Crear publicación

HTTP <http://localhost:3002/publications>

POST <http://localhost:3002/publications> Send

Body JSON raw binary GraphQL Beautify

```
1 {
2   "title": "Rayuela",
3   "content": "Una novela distinta",
4   "authorId": 1,
5   "category": "NOVELA"
6 }
```

Body Cookies Headers (7) Test Results

201 Created 120 ms 427 B

```
1 {
2   "id": 1,
3   "title": "Rayuela",
4   "content": "Una novela distinta",
5   "authorId": 1,
6   "category": "NOVELA",
7   "status": "DRAFT",
8   "createdAt": "2026-01-28T03:20:20.574Z",
9   "updatedAt": "2026-01-28T03:20:20.574Z"
10 }
```

Listar publicaciones

HTTP <http://localhost:3002/publications>

GET <http://localhost:3002/publications> Send

Params Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (7) Test Results

200 OK 27 ms 424 B

```
1 [
2   {
3     "id": 1,
4     "title": "Rayuela",
5     "content": "Una novela distinta",
6     "authorId": 1,
7     "category": "NOVELA",
8     "status": "DRAFT",
9     "createdAt": "2026-01-28T03:20:20.574Z",
10    "updatedAt": "2026-01-28T03:20:20.574Z",
11    "category": "NOVELA"
12  }
```

Listar autores

GET Send

Docs Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	***

Body Cookies Headers (7) Test Results | ⏱

200 OK • 51 ms • 315 B • ⌂ • ⚡

{ } JSON ▾ ▶ Preview Visualize | ▾

```
1 [ {  
2   "id": 1,  
3   "name": "Julio Cortázar",  
4   "email": "jc@literatura.com",  
5   "biography": null  
6 } ]  
7 }
```

Buscar autor con authorId correcto

GET Send

Docs Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	***

Body Cookies Headers (7) Test Results | ⏱

200 OK • 148 ms • 313 B • ⌂ • ⚡

{ } JSON ▾ ▶ Preview Visualize | ▾

```
1 {  
2   "id": 1,  
3   "name": "Julio Cortázar",  
4   "email": "jc@literatura.com",  
5   "biography": null  
6 }  
7 }
```

Prueba: ingresar datos incompletos

HTTP http://localhost:3001/api/authors

POST http://localhost:3001/api/authors

Body (8) Body Scripts Tests Settings

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

```
1 {  
2   "name": " Jorge Icaza",  
3   "email": " ",  
4   "biography": "Escritor ecuatoriano"  
5 }  
6
```

Body Cookies Headers (7) Test Results

400 Bad Request 6 ms 275 B

```
1 {  
2   "message": "Email is required"  
3 }
```

Prueba: Cuando falta el nombre del autor

HTTP http://localhost:3001/api/authors

POST http://localhost:3001/api/authors

Body (8) Body Scripts Tests Settings

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

```
1 {  
2   "name": " ",  
3   "email": " jorgeo@example.com ",  
4   "biography": "Escritor ecuatoriano"  
5 }  
6
```

Body Cookies Headers (7) Test Results

400 Bad Request 6 ms 274 B

```
1 {  
2   "message": "Name is required"  
3 }
```

Cuando falta el nombre y el email del autor

HTTP http://localhost:3001/api/authors

POST http://localhost:3001/api/authors

Body (8) Body Scripts Tests Settings

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

```
1 {  
2   "name": " Jorge Icaza",  
3   "email": " jorgeo@example.com ",  
4   "biography": "Escritor ecuatoriano"  
5 }  
6
```

Body Cookies Headers (7) Test Results

400 Bad Request 7 ms 295 B

```
1 {  
2   "message": "Author with this email already exists"  
3 }
```

Prueba: Obtener autor por ID inexistente

HTTP <http://localhost:3001/api/authors/5>

GET <http://localhost:3001/api/authors/5> Send

Docs Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	...

Body Cookies Headers (7) Test Results | ⚡

404 Not Found 8 ms 272 B ⌂ ⚡

{ } JSON ▾ ▶ Preview ⚡ Debug with AI ▾

```

1 {
2   "message": "Author not found"
3 }
```

Crear publicación con la id de un autor inexistente.

HTTP <http://localhost:3002/publications>

POST <http://localhost:3002/publications> Send

Docs Params Authorization Headers (8) Body ● Scripts Settings Cookies

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

Schema Beautify

1 {	2 "title": "Rayuela",
3 "content": "Una novela distinta",	4 "authorId": 5,
5 "category": "NOVELA"	6 }

Body Cookies Headers (7) Test Results | ⚡

400 Bad Request 1.89 s 279 B ⌂ ⚡

{ } JSON ▾ ▶ Preview ⚡ Debug with AI ▾

```

1 {
2   "message": "Author does not exist"
3 }
```

Buscar Endpoint inexistente en publicaciones

HTTP <http://localhost:3002/publications/2>

GET <http://localhost:3002/publications/2> Send

Docs Params Authorization Headers (8) Body ● Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	...

Body Cookies Headers (7) Test Results | ⚡

404 Not Found 80 ms 277 B ⌂ ⚡

{ } JSON ▾ ▶ Preview ⚡ Debug with AI ▾

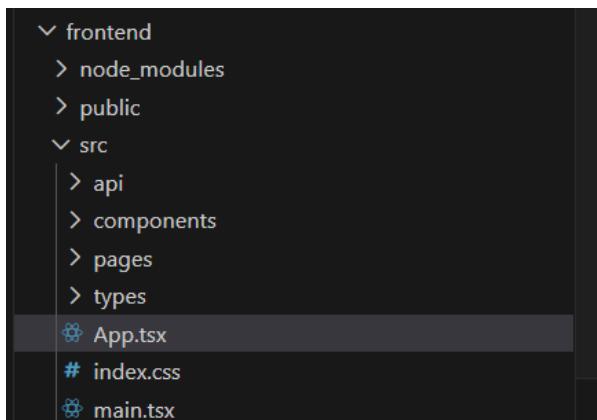
```

1 {
2   "message": "Publication not found"
3 }
```

Frontend

Se desarrolló un frontend en React + TypeScript que consume los microservicios Authors y Publications, donde se puede crear y listar autores y publicaciones, validando dependencias entre microservicios en tiempo real.

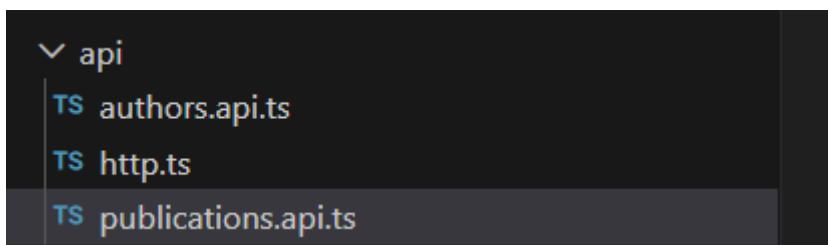
Estructura base



Configuración de conexión con los microservicios

Creamos una capa api/ que tambien centralizamos las llamadas HTTP y separamos el frontend dle backend

Estructura del API layer



Configuración del cliente HTTP

```
ts http.ts  X
frontend > src > api > ts http.ts > ...
1 import axios from "axios";
2
3 export const http = axios.create({
4   timeout: 5000,
5   headers: {
6     "Content-Type": "application/json",
7   },
8});
```

Api de Authors

```
ts authors.api.ts X
frontend > src > api > ts authors.api.ts > ...
5  export const AuthorsAPI = {
6    getAll: async () => {
7      const response = await http.get(AUTHORS_URL);
8      return response.data;
9    },
10
11   getById: async (id: number) => {
12     const response = await http.get(`${AUTHORS_URL}/${id}`);
13     return response.data;
14   },
15
16   create: async (author: {
17     name: string;
18     email: string;
19     biography?: string;
20   }) => {
21     const response = await http.post(AUTHORS_URL, author);
22     return response.data;
23   }

```

Api de Publications

```
ts publications.api.ts X
frontend > src > api > ts publications.api.ts > ...
5  export const PublicationsAPI = {
6    getAll: async () => {
7      return response.data;
8    },
9  },
10
11  getById: async (id: number) => {
12    const response = await http.get(`${PUBLICATIONS_URL}/${id}`);
13    return response.data;
14  },
15
16  create: async (publication: {
17    title: string;
18    content: string;
19    authorId: number;
20    category?: string;
21  }) => {
22    const response = await http.post(PUBLICATIONS_URL, publication);
23    return response.data;
24  }

```

Se implementó una capa de acceso a datos en el frontend, el cual se encargó de la comunicación con los microservicios Authors y Publications mediante Axios

Definición de tipos TypeScript (models)

Aquí definimos las interfaces TypeScript que representen exactamente:

- Lo que devuelve Authors Service
- Lo que envía y devuelve Publications Service

Crear tipos para Authors

The screenshot shows a code editor with two tabs: 'Author.ts' and 'Publication.ts'. The 'Author.ts' tab is active, showing the following TypeScript code:

```
frontend > src > types > TS Author.ts > ...
1  export interface Author {
2      id: number;
3      name: string;
4      email: string;
5      biography?: string | null;
6  }
7  
```

Crear tipos para Publications

The screenshot shows a code editor with one tab: 'publication.ts'. The code defines a type 'Category' and an interface 'Publication'.

```
frontend > src > types > TS publication.ts > ...
1  export type Category = "NOVELA" | "POESIA" | "ENSAYO";
2
3  export interface Publication {
4      id: number;
5      title: string;
6      content: string;
7      authorId: number;
8      category: Category | "";
9  }
10 
```

Aquí definimos los modelos TypeScript en el frontend para representar las entidades Author y Publication.

Vista: listar autores desde el backend

Creamos la página AuthorsPage

```
AuthorsPage.tsx
frontend > src > pages > AuthorsPage.tsx > AuthorsPage > useEffect() callback > loadAuthors
  4
  5   export const AuthorsPage = () => {
  6     const [authors, setAuthors] = useState<Author[]>([[]]);
  7     const [loading, setLoading] = useState(true);
  8     const [error, setError] = useState("");
  9
 10    useEffect(() => {
 11      const loadAuthors = async () => [
 12        try {
 13          const data = await AuthorsAPI.getAll();
 14          setAuthors(data);
 15        } catch (err) {
 16          setError("Error al cargar autores");
 17        } finally {
 18          setLoading(false);
 19        }
 20      ];
 21
 22      loadAuthors();
 23    }, []);
 24
 25    if (loading) return <p>Cargando autores...</p>;
 26    if (error) return <p>{error}</p>;
 27
 28    return (
 29      <div>
 30        <h2>Autores</h2>
 31        <ul>
 32          {authors.map((author) => (
 33            <li key={author.id}>
 34              <strong>{author.name}</strong> - {author.email}
 35            </li>
 36          ))}
 37        </ul>
 38      </div>
 39    );
 40  };

```

Verificamos el tipo Author

```
author.ts
frontend > src > pages > author.ts > ...
  1  export interface Author {
  2    id: number;
  3    name: string;
  4    email: string;
  5    biography?: string | null;
  6  }
  7
```

Conectamos la vista en App.tsx

```
TS author.ts | App.tsx X
frontend > src > App.tsx > ...
1 import { AuthorsPage } from "./pages/AuthorsPage";
2
3 <function App() {
```

Instala CORS

Dentro de authors-service instalamos CORS

```
PS C:\espe1\workspace\microservicios-editorial\authors-service> npm install cors
● added 3 packages, and audited 246 packages in 3s

64 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\espe1\workspace\microservicios-editorial\authors-service>
```

Tambien sus dependencias:

```
PS C:\espe1\workspace\microservicios-editorial\authors-service> npm install --save-dev @types/cors
added 1 package, and audited 247 packages in 2s

64 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\espe1\workspace\microservicios-editorial\authors-service>
```

Dentro de publications-service instalamos CORS y sus dependencias

```
PS C:\espe1\workspace\microservicios-editorial\publications-service> npm install cors
added 2 packages, and audited 246 packages in 2s

64 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\espe1\workspace\microservicios-editorial\publications-service> npm install --save-dev @types/cors
added 1 package, and audited 247 packages in 1s

64 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\espe1\workspace\microservicios-editorial\publications-service>
```

Y agregamos en app.ts

```
2 import cors from "cors";
```

Se instalaron los tipos de TypeScript para el middleware CORS (@types/cors) en cada microservicio, garantizando compatibilidad con TypeScript y correcta compilación del proyecto.

Vista para crear Autor en el formulario

Crear una pantalla en el frontend que permita crear autores usando el authors-service.

Creamos la página CreateAuthorPage.tsx

```
ts app.ts CreateAuthorPage.tsx X
frontend > src > pages > CreateAuthorPage.tsx > CreateAuthorPage > handleSubmit
1 import { useState } from "react";
2 import { AuthorsAPI } from "../api/authors.api";
3
4 export default function CreateAuthorPage() {
5   const [name, setName] = useState("");
6   const [email, setEmail] = useState("");
7   const [biography, setBiography] = useState("");
8   const [message, setMessage] = useState("");
9
10  const handleSubmit = async (e: React.FormEvent) => {
11    e.preventDefault();
12
13    try {
14      await AuthorsAPI.create({
15        name,
16        email,
17        biography,
18      });
19
20      setMessage("Autor creado correctamente ");
21      setName("");
22      setEmail("");
23      setBiography("");
24    } catch (error) {
25      setMessage("Error al crear autor ");
26    }
27  };
28}
29
30  return (
31    <div>
32      <h2>Crear Autor</h2>
33
34      <form onSubmit={handleSubmit}>
35        <div>
36          <input
37            type="text"
38            placeholder="Nombre"
39            value={name}
40            onChange={(e) => setName(e.target.value)}
41            required
42          />
43        </div>
44
45        <div>
46          <input
47            type="email"
48            placeholder="Email"
49            value={email}
50            onChange={(e) => setEmail(e.target.value)}
51            required
52          />
53        </div>
54
55      </form>
56    </div>
57  );
58}
```

Registraremos la ruta en App.tsx

```
App.tsx X
frontend > src > App.tsx > ...
1 import { AuthorsPage } from "./pages/AuthorsPage";
2 import { BrowserRouter, Routes, Route } from "react-router-dom";
3 import CreateAuthorPage from "./pages/CreateAuthorPage";
4
```

Se implementó un formulario en React para la creación de autores, consumiendo el microservicio Authors mediante llamadas HTTP usando Axios. El frontend valida y envía datos al backend, comprobando la integración completa.

EVIDENCIA WEB

Evidenciamos en el navegador nuestro “crear autor”:

The image contains three screenshots of a web browser window titled "frontend".
1. The first screenshot shows the "Crear Autor" form with fields for Nombre (Gabriel Garcia), Email (ggmarquez@example.com), and Biografía (Escritor colombiano). A "Guardar" button is visible below the form.
2. The second screenshot shows the same form after submission, with the message "Autor creado correctamente" displayed below it.
3. The third screenshot shows the "Autores" page, listing two authors: Julio Cortázar (jc@literatura.com) and Gabriel Garcia (ggmarquez@example.com).

Crear Publicaciones desde el Frontend

Definimos el modelo TypeScript de Publication

```
ts publication.ts X
frontend > src > types > ts publication.ts > ...
1  export type Category = "NOVELA" | "POESIA" | "ENSAYO";
2
3  export interface Publication {
4    title: string;
5    content: string;
6    authorId: number;
7    category: Category | "";
8  }
9
```

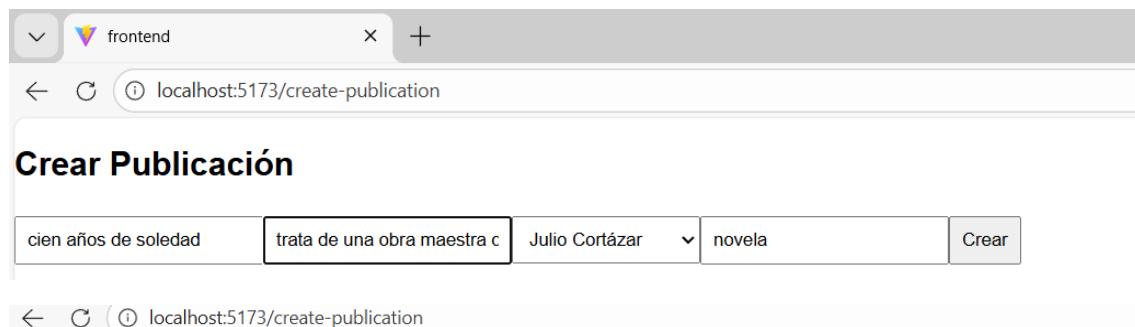
Crear la página CreatePublicationPage.tsx

```
47
48  return (
49    <div>
50      <h2>Crear Publicación</h2>
51
52      <form onSubmit={handleSubmit}>
53        <input
54          name="title"
55          placeholder="Título"
56          value={form.title}
57          onChange={handleChange}
58          required
59        />
60
61        <input
62          name="content"
63          placeholder="Contenido"
64          value={form.content}
65          onChange={handleChange}
66          required
67
68        <select
69          name="authorId"
70          value={form.authorId}
71          onChange={handleChange}
72          required
73        >
74          <option value="">Seleccione autor</option>
75          {authors.map((author) => (
76            <option key={author.id} value={author.id}>
77              {author.name}
78            </option>
79          )))
80        </select>
81
82        <select
83          name="category"
84          value={form.category}
85          onChange={handleChange}
86          required
87        >
88          <option value="">Seleccione categoría</option>
89          <option value="NOVELA">Novela</option>
90          <option value="CUENTO">Cuento</option>
91          <option value="POESIA">Poesía</option>
92        </select>
93
94        <button type="submit">Crear</button>
95
96      </form>
97
```

Verificamos nuestra API de Publications

```
ts publications.api.ts ×
frontend > src > api > ts publications.api.ts > [?] PublicationsAPI
1 import { http } from "./http";
2
3 const PUBLICATIONS_URL = "http://localhost:3002/publications";
4
5 export const PublicationsAPI = {
6   getAll: async () => {
7     const response = await http.get(PUBLICATIONS_URL);
8     return response.data;
9   },
10
11   getById: async (id: number) => {
12     const response = await http.get(`${PUBLICATIONS_URL}/${id}`);
13     return response.data;
14   },
15
16   create: async (publication: {
17     title: string;
18     content: string;
19     authorId: number;
20     category?: string;
21   }) => {
22     const response = await http.post(PUBLICATIONS_URL, publication);
23     return response.data;
24   },
25 };
26
```

Probamos en el navegador: localhost:5173/create-publication



Crear Publicación

cien años de soledad	trata de una obra maestra c	Julio Cortázar	novela	Crear
----------------------	-----------------------------	----------------	--------	-------

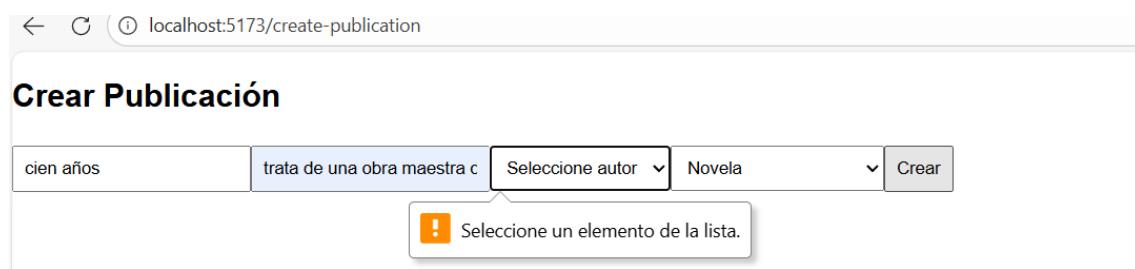


Crear Publicación

cien años de soledad	trata de una obra maestra c	Gabriel Garcia	Novela	Crear
----------------------	-----------------------------	----------------	--------	-------

Publicación creada correctamente

Y que sucede cuando no seleccionó el nombre del autor



Crear Publicación

cien años	trata de una obra maestra c	Seleccione autor	Novela	Crear
-----------	-----------------------------	------------------	--------	-------

! Selecione un elemento de la lista.

Listar publicaciones en el frontend

Crear vista para listar publicaciones

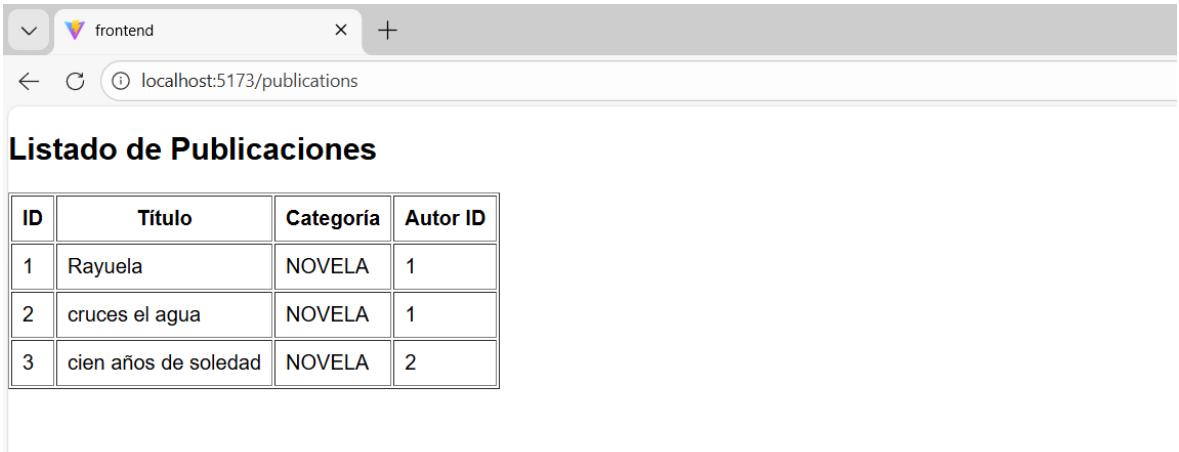
```
15  return (
16    <div>
17      <h2>Listado de Publicaciones</h2>
18
19      {error && <p>{error}</p>}
20
21      <table border={1} cellPadding={8}>
22        <thead>
23          <tr>
24            <th>ID</th>
25            <th>Título</th>
26            <th>Categoría</th>
27            <th>Autor ID</th>
28          </tr>
29        </thead>
30        <tbody>
31          {publications.map((pub) => [
32            <tr key={pub.id}>
33              <td>{pub.id}</td>
34              <td>{pub.title}</td>
35              <td>{pub.category}</td>
36              <td>{pub.authorId}</td>
37            </tr>
38          ])}
39        </tbody>
40      </table>
41    </div>
42  );
43
```

Registrar la ruta en App.ts

```
4  import CreatePublicationPage from './pages/CreatePublicationPage';
5  import PublicationsPage from './pages/PublicationsPage';
6
7
14  <Route path="/publications" element={<PublicationsPage />} />
15  </Routes>
```

Evidenciamos en el navegador

<http://localhost:5173/publications>



The screenshot shows a browser window with the title 'frontend'. The address bar displays 'localhost:5173/publications'. The main content area is titled 'Listado de Publicaciones' and contains a table with three rows of data:

ID	Título	Categoría	Autor ID
1	Rayuela	NOVELA	1
2	cruces el agua	NOVELA	1
3	cien años de soledad	NOVELA	2

Agregamos un Link de react-router-dom

```
    <nav style={{ marginBottom: "20px" }}>
      <Link to="/">Autores</Link> |{" "}
      <Link to="/publications">Publicaciones</Link> |{" "}
      <Link to="/create-author">Crear Autor</Link> |{" "}
      <Link to="/create-publication">Crear Publicación</Link>
    </nav>
```

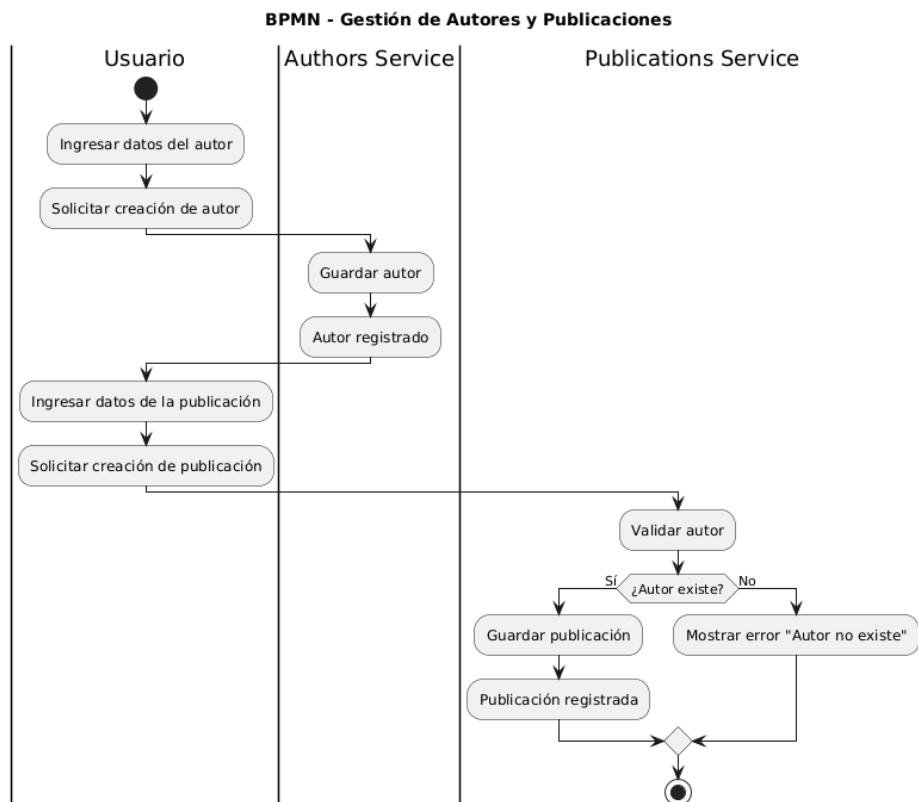
Lo que hicimos fue agregar en la parte de arriba, fuera de <Routes> la navegación hacia las rutas indicadas.

Evidenciamos nuestro listar autores con los links hacia las demás rutas

The screenshot shows a browser window with the title 'frontend'. The address bar displays 'localhost:5173'. Below the address bar, there are four blue underlined links: 'Autores', 'Publicaciones', 'Crear Autor', and 'Crear Publicación'. The main content area has a dark header with the word 'Autores' in white. Below the header, there is a list of two items:

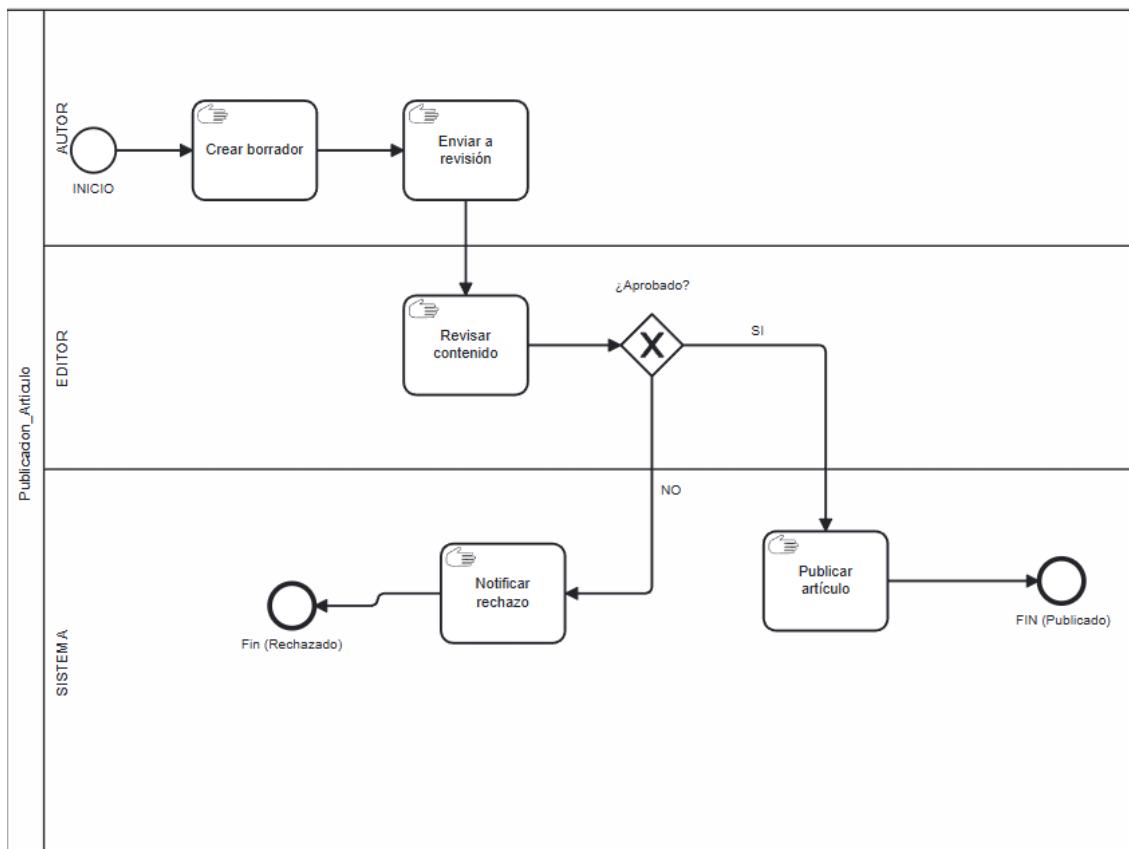
- **Julio Cortázar** — jc@literatura.com
- **Gabriel García** — ggmarquez@example.com

Diagrama BPMN



Modelado BPMN en Camunda

Este modelo permite representar el proceso del microservicio donde tenemos un artículo desde su creación hasta su publicación o rechazo. El proceso inicia cuando se crea un borrador del artículo, el cual es enviado a revisión y luego, un revisor analiza el contenido del artículo y toma una decisión mediante un gateway de decisión, donde se evalúa si el artículo es aprobado o rechazado. Si el artículo es aprobado, se procede a preparar y publicar el contenido, finalizando el proceso con éxito.

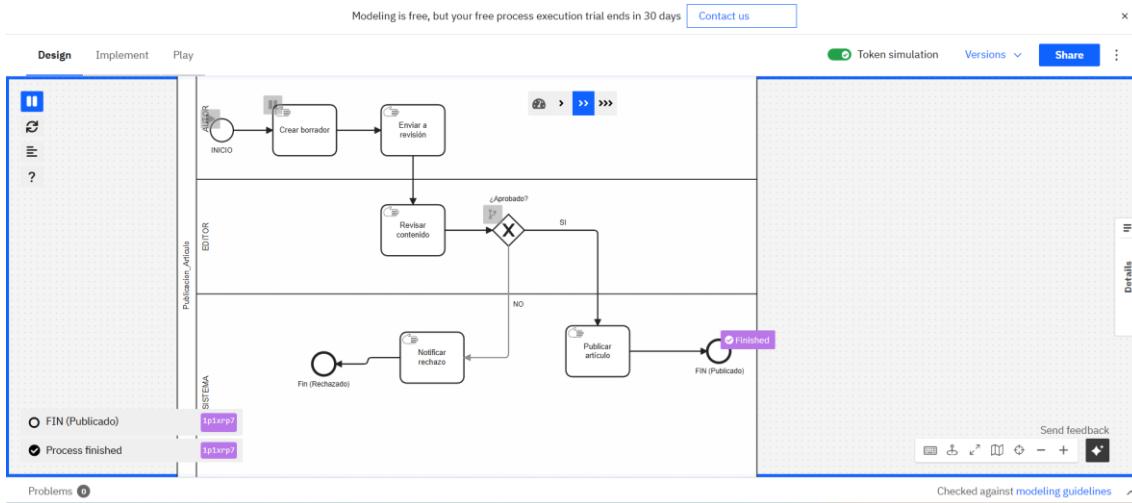


Este modelo nos ayuda a visualizar las actividades, decisiones y resultados del proceso. Además, mediante la **simulación de tokens**, es posible validar el flujo del proceso y comprobar que las decisiones se ejecutan de forma correcta o no, asegurando que el proceso está bien.

Decisión de aprobación (Gateway exclusivo XOR):

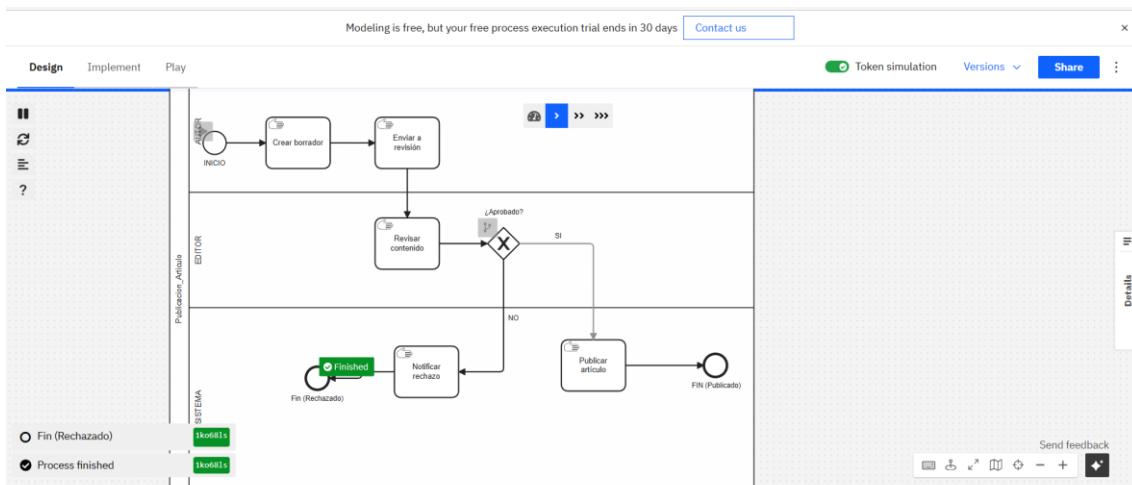
Si aprueba: preparar publicación → publicar → fin (publicado)

Cuando el contenido es correcto (aprobado), el flujo continúa su camino hacia la actividad preparar publicación, seguida de la tarea publicar artículo, y finalmente el proceso en el evento da por finalizado.



Si rechaza: notificar rechazo → fin (rechazado)

A su vez, cuando el contenido es rechazado, el flujo se dirige a la tarea notificar rechazo, concluyendo el proceso en el evento con un mensaje que dice; fin rechazado.



El modelo BPMN representa el proceso de negocio del sistema editorial, desde la creación de un artículo por parte del autor hasta su publicación o rechazo. Aunque la arquitectura del sistema está implementada mediante microservicios (Authors y Publications), el BPMN describe el flujo funcional global del dominio editorial, no un microservicio específico.

CONCLUSIONES

- La arquitectura está basada en microservicios que permitieron desacoplar las funcionalidades del sistema editorial, separando claramente la gestión de autores y publicaciones.
- El uso de Docker y Docker Compose simplificó el despliegue y la ejecución del sistema, permitiendo levantar todos los servicios y bases de datos de forma consistente en distintos entornos.

- La modelación del proceso mediante BPMN y su simulación con Token Simulation permitió visualizar y validar el flujo de negocio de publicación de artículos, asegurando que las decisiones de aprobación y rechazo se ejecuten correctamente antes de una implementación real.

RECOMENDACIONES

- Se recomienda más pruebas automatizadas para cada microservicio, con el fin de detectar errores de forma temprana y asegurar la estabilidad del sistema ante futuros cambios.
- También para una versión productiva del sistema, sería conveniente integrar un API Gateway que centralice el acceso a los microservicios, mejore la seguridad, gestione autenticación y permita un mejor control del tráfico entre cliente y servicios.
- Por último, se recomienda extender el modelo BPMN incorporando métricas de tiempo y carga en la simulación, lo que permitiría analizar cuellos de botella y optimizar el proceso editorial antes de su despliegue en un entorno real.