



Nombres y apellidos: Oscar Domínguez

Asignatura: Arquitectura de software

Link del repositorio:

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

ACTIVIDAD 3

Introducción:

Se realizó 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

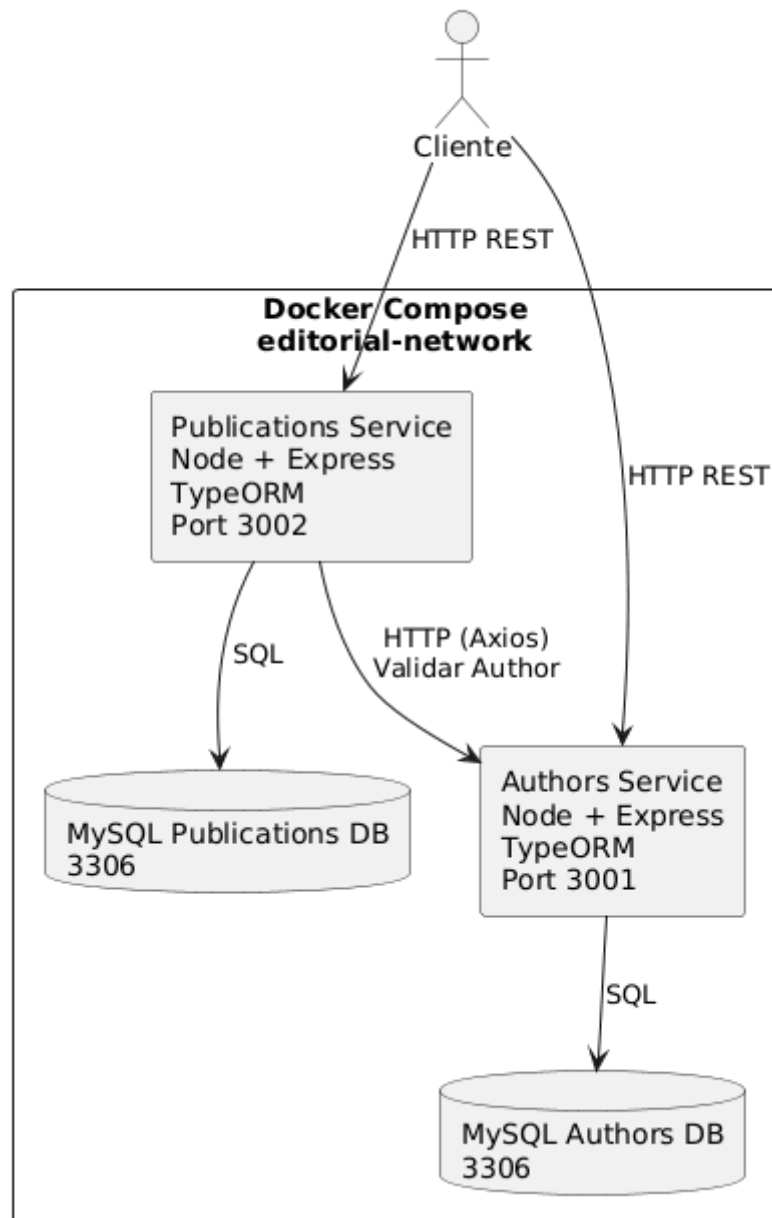
Se implementó una arquitectura de microservicios compuesta por:

- 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 good
```

```
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  |
```

```

v authors-service
  > node_modules
  v src
    v config
      TS data-source.ts
    v controllers
      TS author.controller.ts
    v dtos
      TS AuthorResponseDTO.ts
      TS CreateAuthorDTO.ts
    v entities
      TS Author.ts
      TS AuthorBase.ts
    v repositories
      TS author.repository.ts
    v routes
      TS author.routes.ts
    v 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
7       async createAuthor(data: CreateAuthorDTO): Promise<Author> {
8           // ...
9       }
10
11
12       async getAuthorById(id: number): Promise<Author | null> {
13           return await AuthorRepository.findOneBy({ id });
14       }
15
16       async getAllAuthors(): Promise<Author[]> {
17           return await AuthorRepository.find();
18       }
19   }
20

```

➤ 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 Authors.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 X

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 X

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 X

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

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

- Method:** POST
- URL:** http://localhost:3001/api/authors
- Body:** A JSON object: `{ "name": "Julio Cortázar", "email": "jc@literatura.com" }`
- Response:** 201 Created, 2.18 s, 318 B. The response body is a JSON object: `{ "name": "Julio Cortázar", "email": "jc@literatura.com", "biography": null, "id": 1 }`

GET /authors

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

- Method:** GET
- URL:** http://localhost:3001/api/authors
- Response:** 200 OK, 51 ms, 315 B. The response body is a JSON array: `[{ "id": 1, "name": "Julio Cortázar", "email": "jc@literatura.com", "biography": null }]`

➤ GET /authors/:id

http://localhost:3001/api/authors/1

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

Send

Docs Params Authorization Headers (6) Body Scripts Settings

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 }
```

Prueba: ingresar datos incompletos

http://localhost:3001/api/authors

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

Send

Docs Params Authorization Headers (8) Body Scripts Tests Settings

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

Body Cookies Headers (7) Test Results

400 Bad Request • 6 ms • 275 B

{ } JSON Preview Debug with AI

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

http://localhost:3001/api/authors

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

Send

Docs Params Authorization Headers (8) Body Scripts Tests Settings

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

Body Cookies Headers (7) Test Results

400 Bad Request • 6 ms • 274 B

{ } JSON Preview Debug with AI

```
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> Save Share

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

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

☐ 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

{ JSON Preview Debug with AI

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

Prueba: Obtener autor por ID

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

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 }
```

Publications Service

Creamos el proyecto en nuestra carpeta principal

```
PS C:\espe1\workspace\microservicios-editorial> mkdir publications-service
```

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
```

Configuracion 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
      TS data-source.ts
    ▼ controllers
      TS publication.controller.ts
    > dto
    ▼ entities
      TS Article.ts
      TS Publication.ts
      TS PublicationStatus.ts
    ▼ repositories
      TS publication.repository.ts
      TS typeorm-publication.repository.ts
    ▼ routes
      TS publication.routes.ts
    ▼ services
      TS authors-client.ts
      TS publication.service.ts
  TS app.ts
  TS server.ts
```

```
TS app.ts X TS server.ts {} package.json
publications-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
```

Ejecución del servicio

```
TS server.ts X
publications-service > src > TS server.ts > ...
1 import { reflect-metadata } from "reflect-metadata";
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
9 export abstract class Publication {
20   @Column()
21   authorId!: number;
22
23   @Column({
24     type: "enum",
25     enum: PublicationStatus,
26     default: PublicationStatus.DRAFT
27   })
28   status!: PublicationStatus;
29
30   @CreateDateColumn()
31   createdAt!: Date;
32
33   @UpdateDateColumn()
34   updatedAt!: Date;
35 }
```

- 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 al 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 > ...
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
31  async getPublicationById(id: number): Promise<Article | null> {
32    return await this.repository.findById(id);
33  }
34
35  async getAllPublications(): Promise<Article[]> {
36    return await this.repository.findAll();
37  }
38
39  async changeStatus(
40    id: number,
41    status: PublicationStatus
42  ): Promise<Article | null> {
43    return await this.repository.updateStatus(id, status);
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

```

TS publication.routes.ts X TS app.ts TS server.ts
publications-service > src > routes > TS 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.getById);
10 router.patch("/:id/status", controller.changeStatus);
11
12 export default router;
13

```

Registrar rutas en app.ts

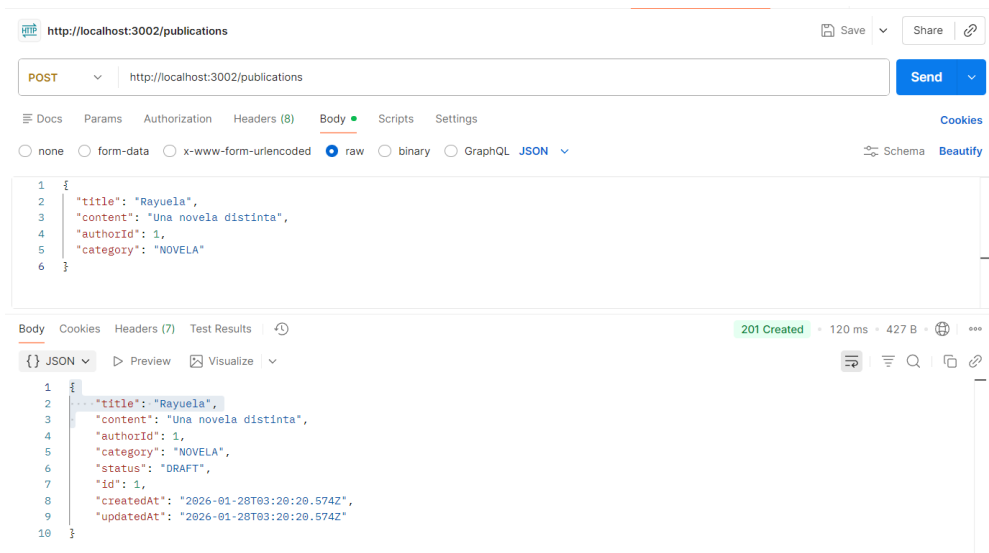
```

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

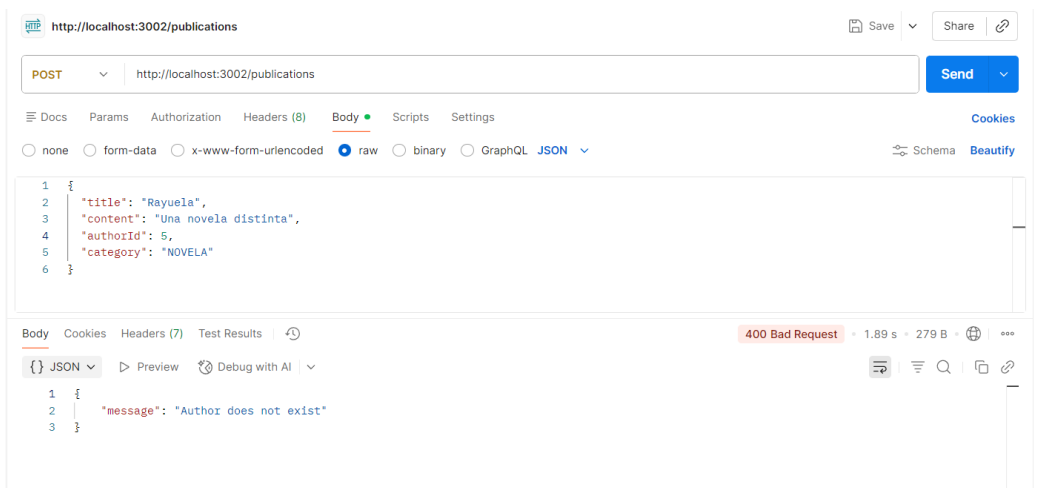
```

Evidencia:

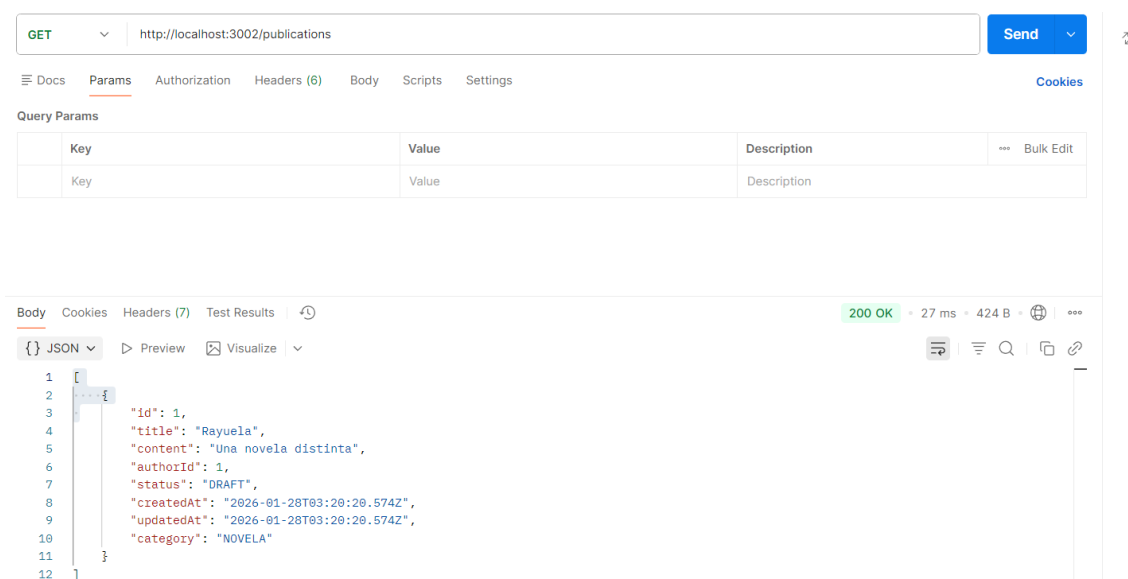
Se realizaron pruebas mediante Postman para validar el funcionamiento de los microservicios. El servicio de publicaciones verifica la existencia del autor consultando al Authors Service antes de crear una publicación.



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.



Podemos ver posteriormente el “Listar publicaciones”



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)

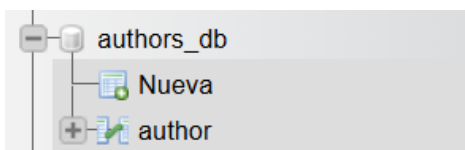
- Configuramos TypeORM con DataSource
- Conectamos Authors Service a su propia base de datos

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

```
TS data-source.ts 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
```

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  });
```

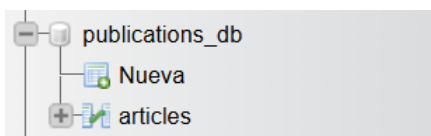
Inicializamos la conexión al arrancar el servidor

```

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. Además servirán para usar variables de entorno usadas en Docker.

Contenerización con Docker

Dockerfile para cada microservicio

- Authors Service

```

Dockerfile  X
authors-service > 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

```
Dockerfile X
publications-service > Dockerfile > ...
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

```
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
```

```

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

```

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
```

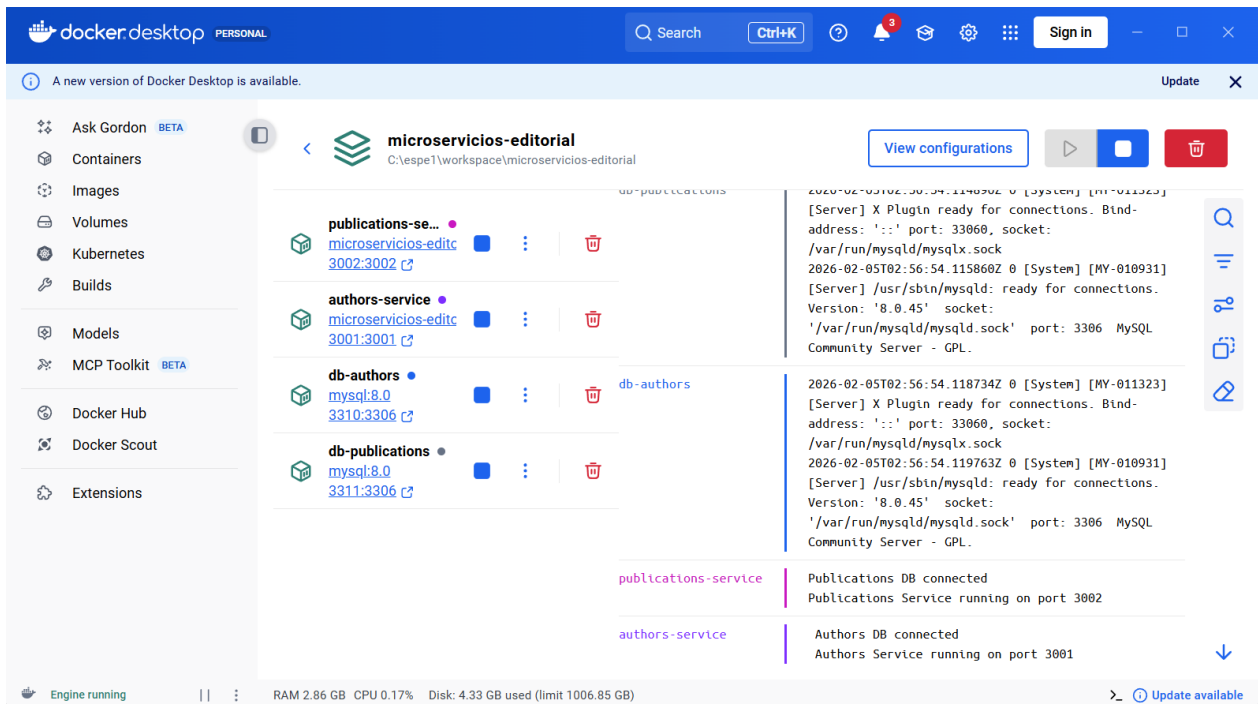
La configuración de la base de datos se realizó mediante variables de entorno, permitiendo que los microservicios funcionen tanto en entorno local como en contenedores Docker.

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

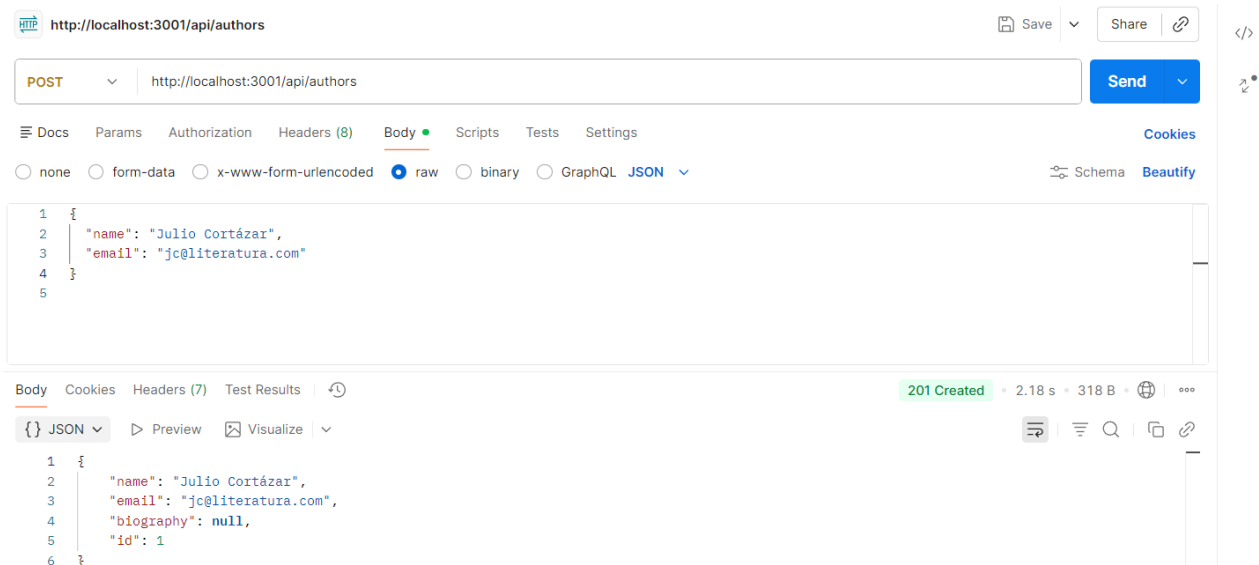
authors-service      | [INFO] 02:57:55 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.2, typescript ver. 5.9.3)
authors-service      | Authors DB connected
authors-service      | Authors Service running on port 3001
```

Vemos en Docker los servicios conectados



Pruebas finales con Postman

Crear autor



Crear publicación

HTTP

http://localhost:3002/publications

Save

Share

</>

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": 1,
5   "category": "NOVELA"
6 }
```

Body

Cookies

Headers (7)

Test Results

201 Created

120 ms

427 B

{}

JSON

Preview

Visualize

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

Listar publicaciones

HTTP

http://localhost:3002/publications

Save

Share

</>

GET

http://localhost:3002/publications

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

27 ms

424 B

{}

JSON

Preview

Visualize

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

Listar autores

GET http://localhost:3001/api/authors 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   {
3     "id": 1,
4     "name": "Julio Cortázar",
5     "email": "jc@literatura.com",
6     "biography": null
7   }
8 ]
```

Buscar autor con authorId correcto

GET http://localhost:3001/api/authors/1 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 }
```

Prueba: ingresar datos incompletos

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 Beautify

☐ 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

{ } JSON Preview Debug with AI

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

Prueba: Cuando falta el nombre del 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 Beautify

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

```
1 {
2   "name": " ",
3   "email": " jorgeo@exmple.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 }
```

Cuando falta el nombre y el email del 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 Beautify

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

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

Body Cookies Headers (7) Test Results 400 Bad Request 7 ms 295 B

{ } JSON Preview Debug with AI

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

Prueba: Obtener autor por ID inexistente

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://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://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

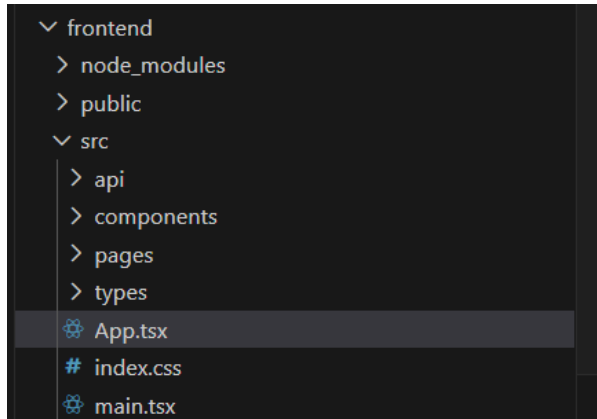
{ } JSON Preview Debug with AI

```
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.

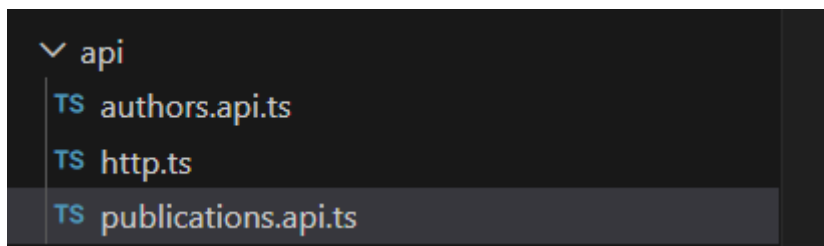
Estructura base



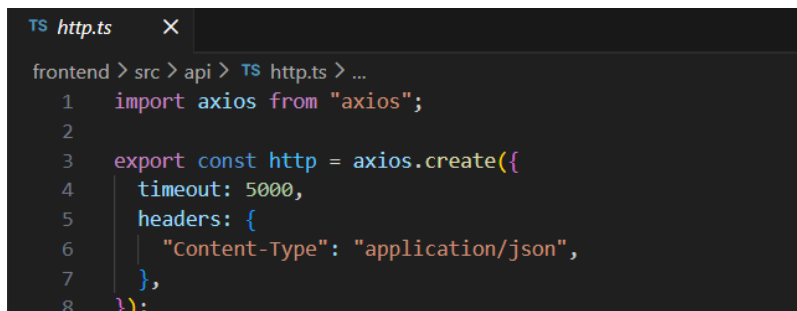
Configuración de conexión con los microservicios

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

Estructura del API layer



Configuración del cliente HTTP



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   }
24 }
```

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


```
TS Author.ts X TS Publication.ts
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

```
TS publication.ts X
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 X
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  };
41  }
```

Verificamos el tipo Author

```
TS author.ts X App.tsx
frontend > src > pages > TS 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
```

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  return (
30    <div>
31      <h2>Crear Autor</h2>
32
33      <form onSubmit={handleSubmit}>
34        <div>
35          <input
36            type="text"
37            placeholder="Nombre"
38            value={name}
39            onChange={(e) => setName(e.target.value)}
40            required
41          />
42        </div>
43
44        <div>
45          <input
46            type="email"
47            placeholder="Email"
48            value={email}
49            onChange={(e) => setEmail(e.target.value)}
50            required
51          />
52        </div>
53      </form>
54    </div>
55  );
56}
```

Registramos la ruta en App.tsx

```
App.tsx
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 first screenshot shows the 'Crear Autor' form at localhost:5173/create-author. It contains three input fields: 'Nombre' (filled with 'Gabriel Garcia'), 'Email' (filled with 'ggmarquez@example.com'), and 'Biografía' (filled with 'Escritor colombiano'). A 'Guardar' button is at the bottom.

The second screenshot shows the same form after submission. The 'Nombre' field is now empty, and a message 'Autor creado correctamente' is displayed below the form.

The third screenshot shows the 'Autores' list at localhost:5173. It displays a list of 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
69        <select
70          name="authorId"
71          value={form.authorId}
72          onChange={handleChange}
73          required
74        >
75          <option value="">Seleccione autor</option>
76          {authors.map((author) => (
77            <option key={author.id} value={author.id}>
78              {author.name}
79            </option>
80          ))}
81        </select>
82
83        <select
84          name="category"
85          value={form.category}
86          onChange={handleChange}
87          required
88        >
89          <option value="">Seleccione categoría</option>
90          <option value="NOVELA">Novela</option>
91          <option value="CUENTO">Cuento</option>
92          <option value="POESIA">Poesía</option>
93        </select>
94
95
96        <button type="submit">Crear</button>
97      </form>

```

Verificamos nuestra API de Publications

```
TS publications.api.ts X
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

frontend

localhost:5173/create-publication

Crear Publicación

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

localhost:5173/create-publication

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

localhost:5173/create-publication

Crear Publicación

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

! Seleccione un elemento de la lista.

Listar publicaciones en el frontend

Crear vista para listar publicaciones

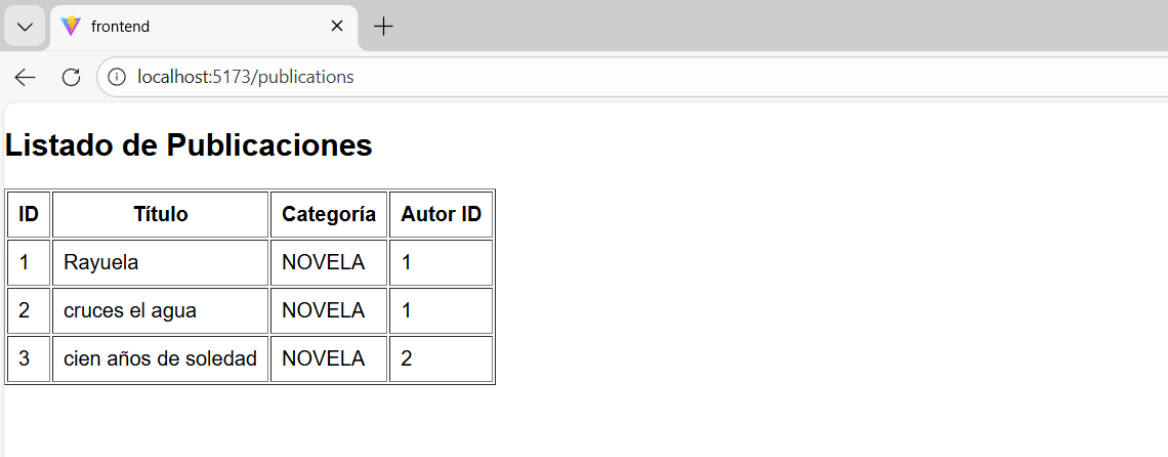
```
15     return (  
16       <div>  
17         <h2>Listado de Publicaciones</h2>  
18  
19         {error && <p>{error}</p>}  
20  
21         <table border={1} cellPadding={8}>  
22           <thead>  
23             <tr>  
24               <th>ID</th>  
25               <th>Título</th>  
26               <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   }  
44 }
```

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 </Router>
```

Evidenciamos en el navegador

<http://localhost:5173/publications>



frontend

localhost:5173/publications

Listado de Publicaciones

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

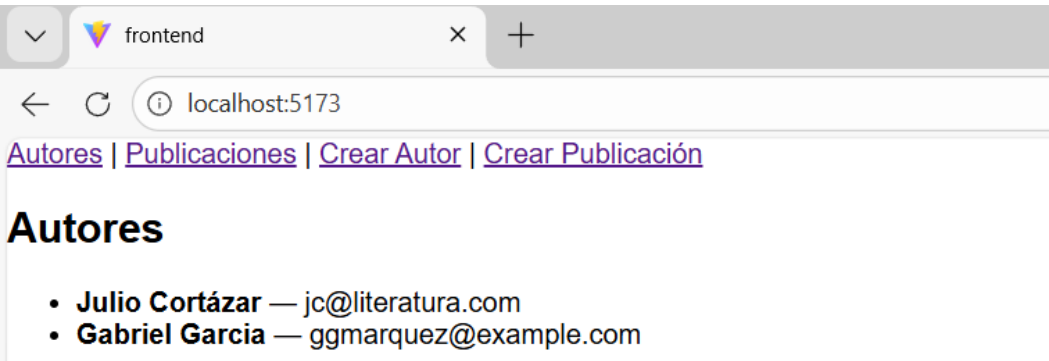
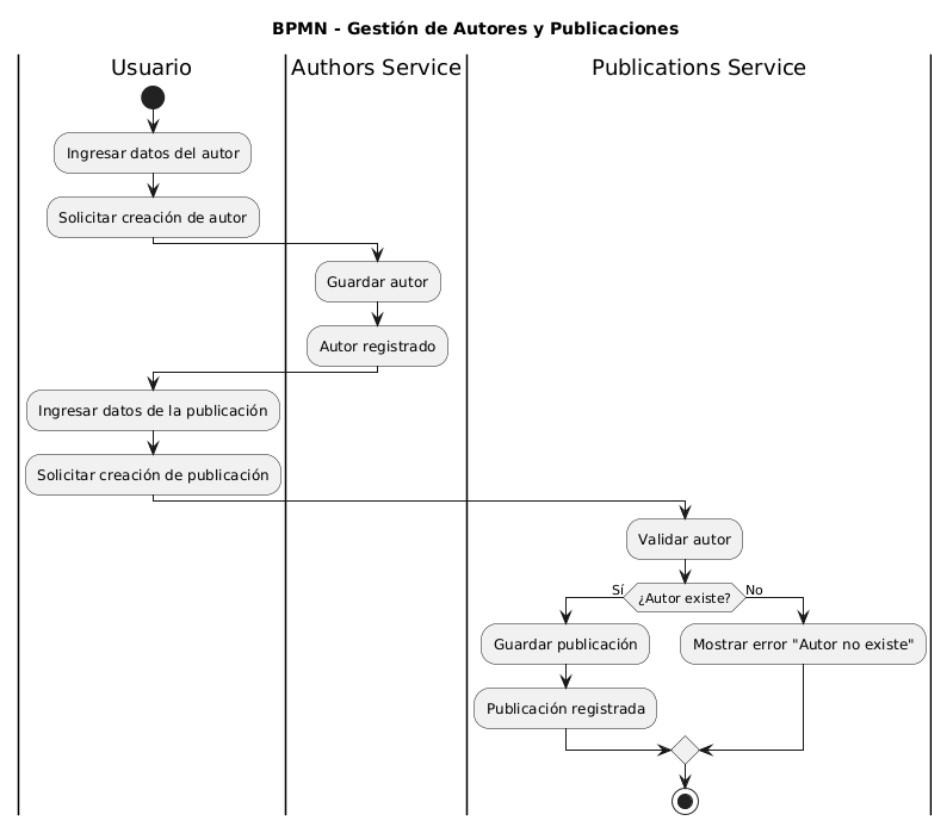
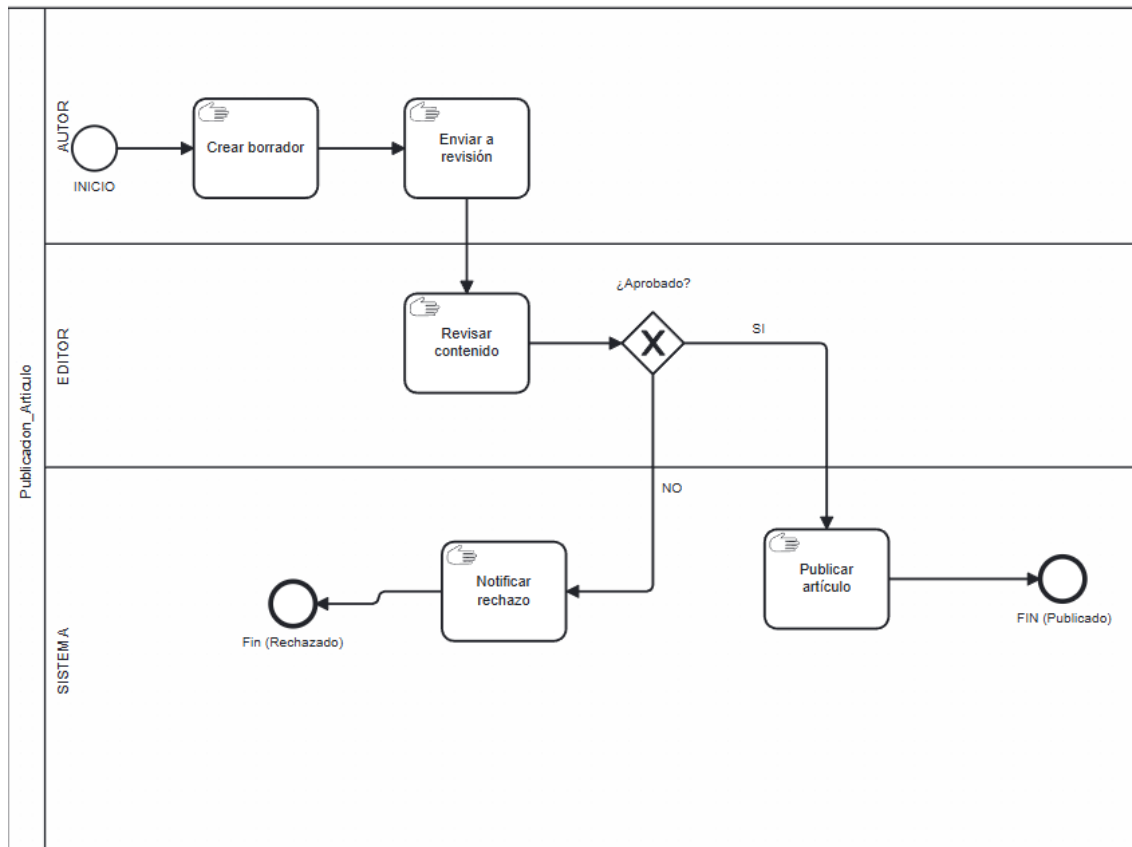


Diagrama BPMN



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.

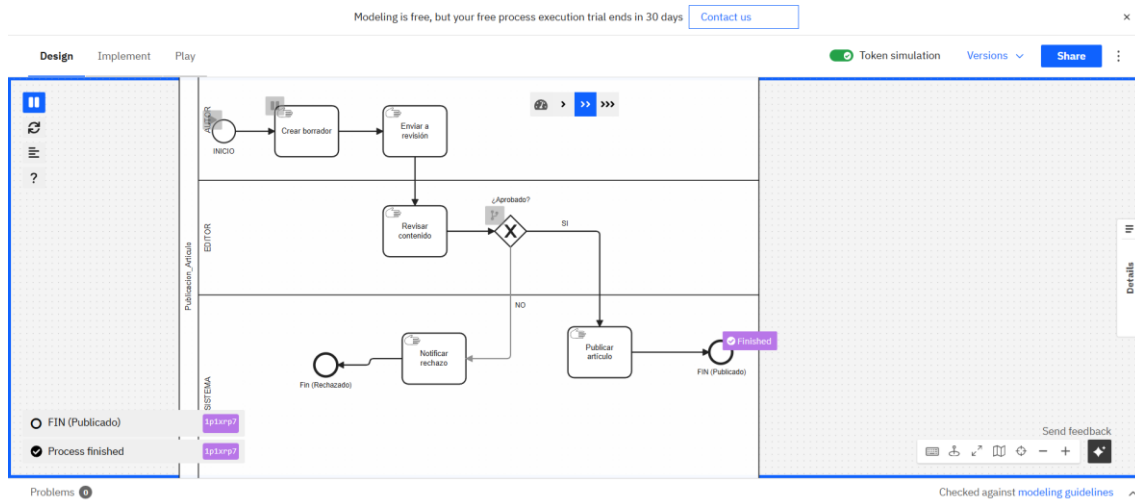


Este modelo nos ayuda a visualizar las actividades, decisiones y resultados del proceso. Además, mediante la **simulación de tokens**, es posible validar el flujo del proceso y comprobar que las decisiones se ejecutan de forma correcta o no, asegurando que el proceso está bien.

Decisión de aprobación (Gateway exclusivo XOR):

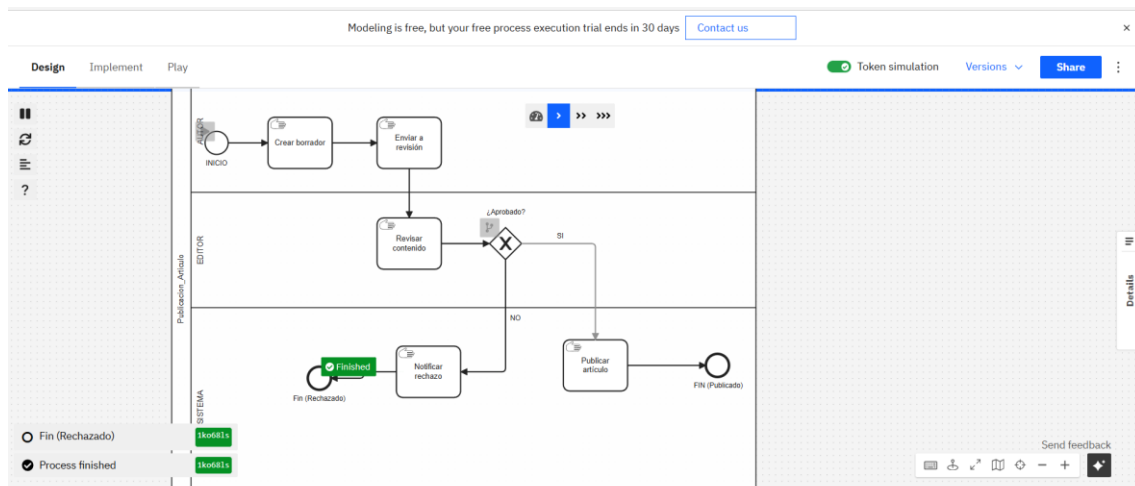
Si aprueba: preparar publicación → publicar → fin (publicado)

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.