

UNIVERSIDAD DE LAS FUERZAS ARMADAS-ESPE



DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

INGENIERÍA EN SOFTWARE

APLICACIONES DISTRIBUIDAS

NRC: 14930

TEMA:

Microservicios y Angular 17

GRUPO Nro. 2

INTEGRANTES:

- ALISSON CLAVIJO
- ADRIAN IZA
- MIGUEL MORALES
- CRISTOPHER ZAMBRANO

DOCENTE:

Ing. Dario. Morales .

Sangolquí, Julio 21, 2024

APLICACIONES DISTRIBUIDAS

Tabla de Contenidos

Tabla de Contenidos.....	1
1. Introducción.....	2
1.1. Objetivo General.....	2
1.2. Objetivos Específicos.....	2
2. Diagramas.....	3
2.1. Diagramas de arquitectura.....	3
2.2. Diagrama flujo de datos.....	4
3. Decisiones de diseño.....	4
4. Desarrollo.....	4
4.1. Backend.....	4
4.2. Frontend.....	4
5. Problemas encontrados y cómo se resolvieron.....	5
6. Ejecución.....	5
7. Conclusiones y Recomendaciones.....	6
7.1 Conclusiones.....	6
7.2 Recomendaciones.....	6
8. Bibliografía.....	6

APLICACIONES DISTRIBUIDAS

1. Introducción

El desarrollo de sistemas informáticos eficientes y escalables es crucial en el ámbito educativo, especialmente cuando se trata de la gestión de matrículas y estudiantes. En los últimos años, la arquitectura de microservicios ha ganado popularidad debido a su capacidad para mejorar la modularidad, escalabilidad y resiliencia de las aplicaciones complejas (Newman, 2015).

Este proyecto se centra en la creación de un Sistema de Matrículas y Estudiantes utilizando una arquitectura de microservicios y el framework Angular. La elección de una arquitectura de microservicios permite una mejor gestión de los recursos y una mayor flexibilidad en el desarrollo y despliegue de los componentes del sistema (Fowler & Lewis, 2014). Al emplear Angular para el frontend, se garantiza una experiencia de usuario dinámica y receptiva (Seshadri, 2018), mientras que la gestión de la base de datos se distribuye entre MySQL y PostgreSQL para optimizar el almacenamiento de datos de usuarios y cursos, respectivamente.

1.1.Objetivo General

Desarrollar un Sistema de Matrículas y Estudiantes basado en una arquitectura de microservicios utilizando Angular para el frontend y MySQL y PostgreSQL para la gestión de bases de datos, que permita la gestión eficiente de estudiantes y cursos, y facilite el proceso de matriculación.

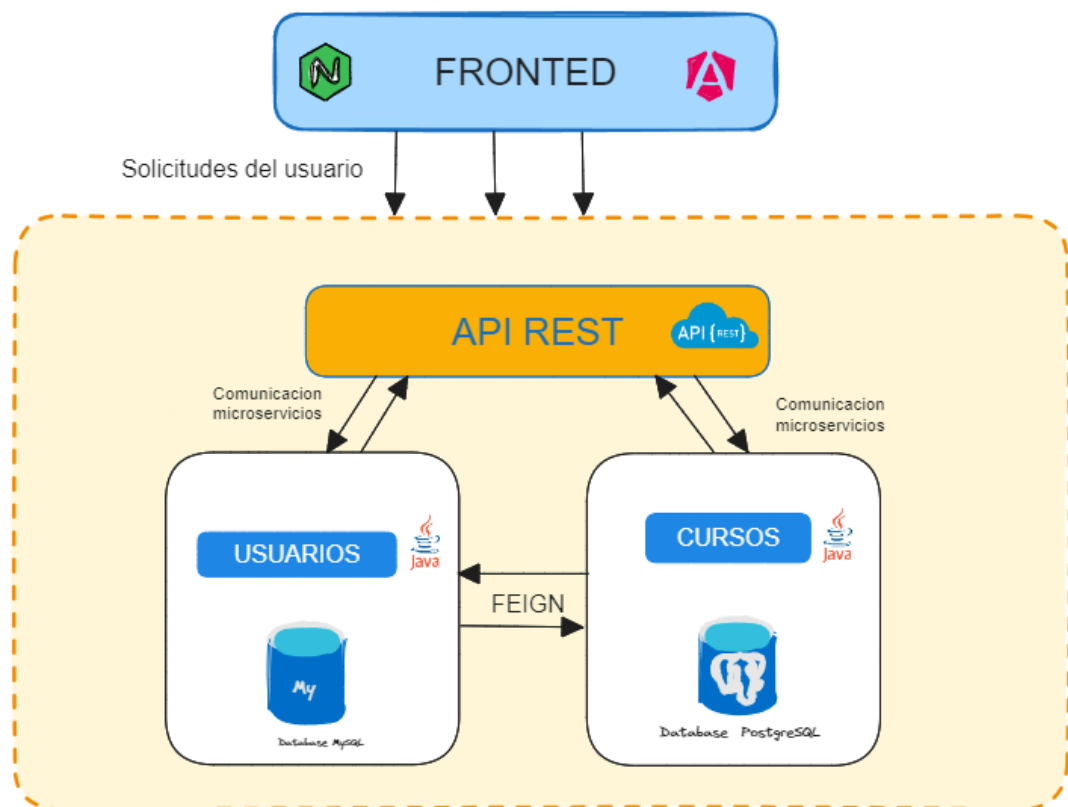
1.2.Objetivos Específicos

- Diseñar y desarrollar microservicios que permitan la creación, actualización, eliminación y visualización de registros de estudiantes.
- Crear microservicios dedicados a la gestión de cursos, incluyendo la creación, actualización, eliminación y visualización de cursos, utilizando PostgreSQL como sistema de gestión de base de datos.
- Implementar funcionalidades que permitan a los estudiantes matricularse en los cursos disponibles a través de una interfaz desarrollada en Angular.

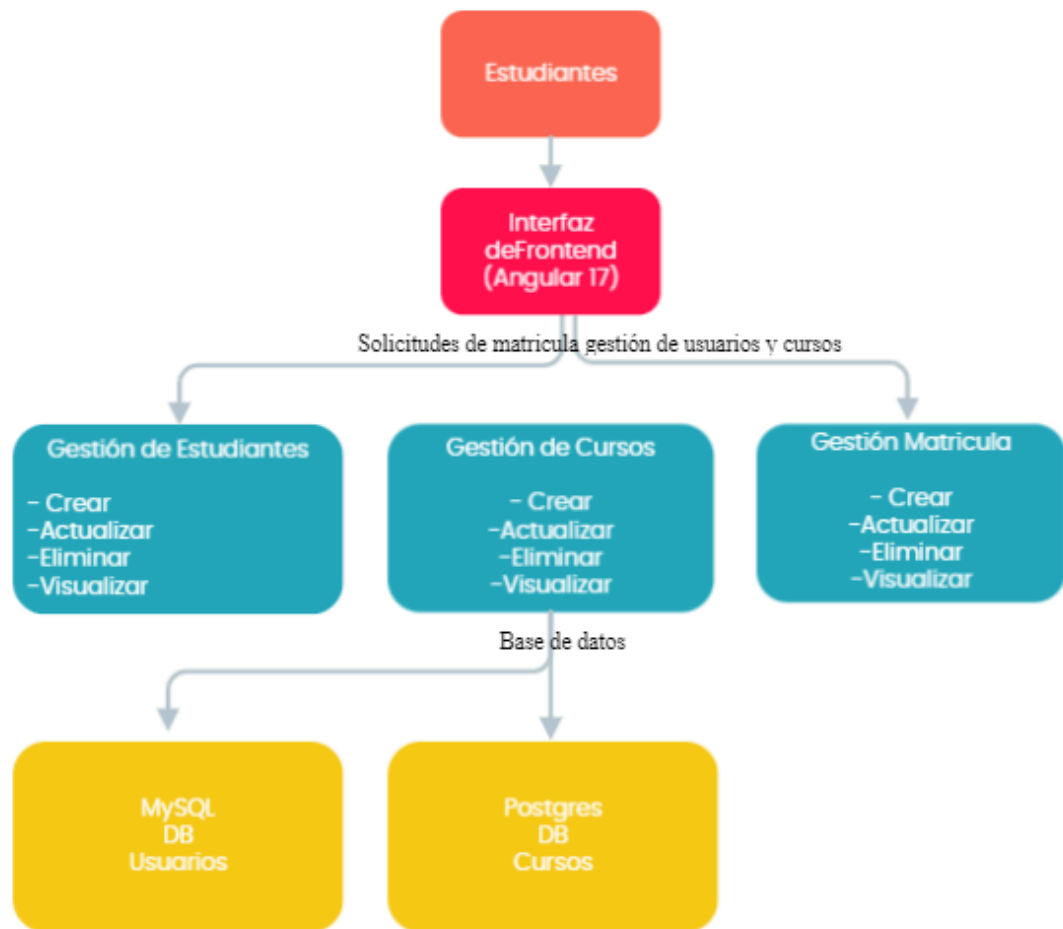
APLICACIONES DISTRIBUIDAS**2. Diagramas**

El diagrama de arquitectura es fundamental para visualizar la estructura general del sistema y cómo interactúan sus componentes. Este diagrama facilita la identificación de posibles cuellos de botella y puntos de fallo, además de ayudar a planificar la escalabilidad y mantenimiento del sistema (Rozanski & Woods, 2011).

El diagrama de flujo describe el flujo de datos y las operaciones dentro del sistema, permitiendo a los desarrolladores y demás interesados visualizar el proceso de matriculación y gestión de estudiantes y cursos. Este diagrama es esencial para identificar y resolver problemas en el proceso, optimizar el rendimiento y asegurar que todas las funciones del sistema se ejecuten de manera eficiente (Yourdon, 1989).

2.1. Diagramas de arquitectura

APLICACIONES DISTRIBUIDAS

2.2. Diagrama flujo de datos**3. Decisiones de diseño****3.1. Arquitectura de microservicios**

Este proyecto se ha dividido en dos micro servicios (msvc-usuarios y msvc-cursos), cada uno es responsable de la gestión de los usuarios y de los cursos respectivamente.

3.2. Diseño de entidades

Las entidades como 'User' y 'Course' con anotaciones JPA para mapear las tablas de la base de datos. También se aplicaron anotaciones como '@NotEmpty' y '@Email'.

3.3. Servicios y repositorios

APLICACIONES DISTRIBUIDAS

- 3.3.1. **Servicios:** En los servicios es donde se implementa la lógica de los negocios, además, se implementó la inyección de dependencias para manejar la interacción entre los componentes.
- 3.3.2. **Repositorios:** Utilizamos Spring Data JPA para manejar la persistencia de datos con 'JpaRepository', lo que nos ayuda con la implementación de operaciones CRUD.

3.4. Comunicación entre Servicios

- 3.4.1. La comunicación entre Servicios msvc-usuarios y msvc-cursos se gestionó usando lo que es el cliente HTTP Feign, lo cual permite una integración eficiente y simplificadas entre los distintos servicios del sistema
- 3.4.2. En las tecnologías y Herramientas se usó Feign como ya se mencionó donde es una biblioteca de Java que facilita la implementación de clientes declarativos eliminando la necesidad de escribir código de clientes repetitivos
- 3.4.3. Se usó Spring Cloud OpenFeign para proporcionar integración con Feign permitiendo la configuración y uso de clientes Feign dentro de aplicaciones Spring.

4. Desarrollo

4.1. Backend

Los servicios están divididos entre el servicio de la gestión de usuarios (msvc-users), la cual utiliza una base de datos en MySQL para el almacenamiento de los usuarios y el servicio de gestión de cursos (msvc-courses) utiliza la base de datos PostgreSQL para el almacenamiento de los cursos. Ambos servicios se comunican mediante Feign para las operaciones, además, ambos servicios tienen una configuración de seguridad para permitir consultas desde direcciones admitidas.

4.2. Frontend

Estamos utilizando Angular 17 para desarrollar el frontend de nuestro Sistema de Matrículas y Estudiantes, el cual está diseñado bajo una arquitectura de microservicios.

4.2.1. Angular 17

APLICACIONES DISTRIBUIDAS

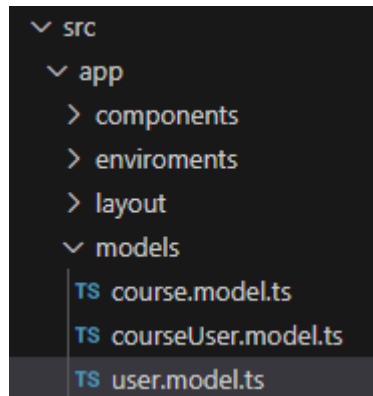
Angular es un framework de diseño de aplicaciones y plataforma de desarrollo para crear aplicaciones de una sola página eficientes y sofisticadas.



Angular, que cuenta con el mantenimiento del equipo dedicado de Google, ofrece un amplio conjunto de herramientas, API y bibliotecas para simplificar y optimizar el flujo de trabajo de desarrollo. Angular ofrece una plataforma sólida sobre la que crear aplicaciones rápidas y confiables que se adaptan tanto al tamaño del equipo como al tamaño de la base del código.

4.2.2. Código Frontend

Creamos dentro de 'src' la carpeta 'models', las tres interfaces que usaremos



Se define una interfaz 'Course' en TypeScript.

```
import { User } from './user.model';
import { CourseUser } from './courseUser.model';

export interface Course {
  id: number;
  name: string;
  courseUsers: CourseUser[];
  users: User[];
```

APLICACIONES DISTRIBUIDAS

```
}
```

Se define una interfaz 'CourseUser' en TypeScript

```
export interface CourseUser {  
  id: number;  
  userId: number;  
}
```

Se define una interfaz 'User' en TypeScript

```
export interface User {  
  id?: number;  
  name?: string;  
  email?: string;  
  password?: string;  
}
```

Definimos un componente Angular llamado 'SidebarComponent', dentro de nuestra carpeta 'layout' el cual se encarga de la barra lateral de navegación de la aplicación.

```
import { Component, OnInit } from '@angular/core';  
import { NavigationEnd, Router, RouterModule } from  
'@angular/router';  
import { SharedModule } from  
'../../components/shared/shared.module';  
  
@Component({  
  selector: 'app-sidebar',  
  standalone: true,  
  imports: [RouterModule, SharedModule],  
  templateUrl: './sidebar.component.html',  
  styleUrls: ['./sidebar.component.css']  
})  
export class SidebarComponent implements OnInit {  
  
  public routes: any[] = [  
    {  
      title: 'Usuarios',  
      icon: 'users',  
      link: ['/users'],  
    },  
  ],  
}
```


APLICACIONES DISTRIBUIDAS

```
{
  title: 'Cursos',
  icon: 'book',
  link: ['/courses'],
},
{
  title: 'Matrícula',
  icon: 'user-plus',
  link: ['/enrollment'],
},
];
private router: Router;

constructor(router: Router) {
  this.router = router;
}

ngOnInit(): void { }
```

Define un componente de Angular llamado 'CoursesComponent', dentro de nuestra carpeta src->app->pages->courses, el cual maneja la lógica relacionada con los cursos, incluyendo la creación, actualización, eliminación y carga de cursos y usuarios.

```
import { Component, OnInit } from '@angular/core';
import { CourseService } from '../../../services/course.service';
import { UserService } from '../../../services/user.service';
import { FormsModule } from '@angular/forms';
import { Course } from '../../../models/course.model';
import { User } from '../../../models/user.model';
import { CourseUser } from '../../../models/courseUser.model';
import { lastValueFrom } from 'rxjs';
import { CommonModule } from '@angular/common';

interface CourseForm {
  id: number;
  name: string;
  courseUsers: CourseUser[];
  users: User[],
```

APLICACIONES DISTRIBUIDAS

```
        selectedUsers: {
            user: User,
            selected: boolean
        }[]
    }

@Component({
    selector: 'app-courses',
    standalone: true,
    imports: [FormsModule, CommonModule],
    templateUrl: './courses.component.html',
    styleUrls: ['./courses.component.css']
})
export class CoursesComponent implements OnInit{

    public courseRows: Course[] = [];
    public userRows: User[] = [];
    public showFormModal = false;
    public courses: Course[] = [];
    message: string | null = null;
    public selectedCourseUsers: User[] = [];

    public courseForm: CourseForm = {
        id: 0,
        name: '',
        courseUsers: [],
        users: [],
        selectedUsers: []
    };

    public userForm: User = {
        id: 0,
        name: '',
        email: ''
    };

    constructor(private courseService: CourseService,
                 private userService: UserService) { }

    ngOnInit(): void {
        this.loadCourses();
    }
}
```

APLICACIONES DISTRIBUIDAS

```
this.loadUsers();

try {
    this.courseService.getCourse().subscribe((courses:
Course[]) => {
        this.courses = courses;
    });
} catch (error) {
    this.handleError(error);
    console.error('Ocurrió un error al obtener los
cursos');
    console.error(error);
}

public async createCourse() {
    try {
        const course = await lastValueFrom(
            this.courseService.createCourse(this.courseForm)
        );
        await this.loadCourses();
    } catch (error) {
        this.handleError(error);
        console.error('Ocurrió un error al crear el curso');
        console.error(error);
    }
}

public async updateCourse() {
    try {
        const course = await lastValueFrom(
            this.courseService.updateCourse({
                id: this.courseForm.id,
                name: this.courseForm.name,
                courseUsers:
this.courseForm.selectedUsers.filter(x => x.selected).map(x => ({
                    id: 0,
                    userId: x.user.id!
                })),
                users: this.courseForm.selectedUsers.map(x =>
x.user)
            })
        );
    } catch (error) {
        this.handleError(error);
        console.error('Ocurrió un error al actualizar el curso');
        console.error(error);
    }
}
```

APLICACIONES DISTRIBUIDAS

```
        ))
    );
    //console.log(course);
    await this.loadCourses();
} catch (error) {
    this.handleError(error);
    console.error('Ocurrió un error al actualizar el
curso');
    console.error(error);
}
}

public async deleteCourse(id: number) {
    try {
        await lastValueFrom(
            this.courseService.deleteCourse(id)
        );
        await this.loadCourses();
    } catch (error) {
        this.handleError(error);
        console.error('Ocurrió un error al eliminar el
curso');
        console.error(error);
    }
}

private async loadCourses() {
    try {
        this.courseRows = await
lastValueFrom(this.courseService.getCourse());
    } catch (error) {
        this.handleError(error);
        console.error('Ocurrió un error al obtener los
cursos');
        console.error(error);
    }
}

private async loadUsers() {
    try {
        const users = await
```

APLICACIONES DISTRIBUIDAS

```
lastValueFrom(this.userService getUsers());

    this.userRows = users;
    this.courseForm.users = [];

    for (const user of this.userRows) {
        this.courseForm.selectedUsers.push({
            user,
            selected: false
        });
    }
} catch (error) {
    this.handleError(error);
    console.error('Ocurrió un error al obtener los usuarios');
    console.error(error);
}

}

public openFormModal(course: Course): void {
    if (course) {
        this.courseForm = {
            id: course.id ?? 0,
            name: course.name ?? '',
            courseUsers: course.courseUsers ?? [],
            users: this.userRows ?? [],
            selectedUsers: this.courseForm.selectedUsers.map(x =>
({ ...x, selected: false }))
        };

        this.courseForm.selectedUsers =
this.courseForm.selectedUsers.map(user => {
            const foundUser = this.courseForm.courseUsers.find(u
=> u.userId === user.user.id);
            return { user: foundUser ?? user.user, selected:
foundUser ? true : false };
        });
    }
}

private handleError(error: any): void {
    switch (error.status) {
```

APLICACIONES DISTRIBUIDAS

```
        case 404:
            this.message = 'User or course not found.';
            break;
        case 400:
            this.message = 'Error: ' + error.error.message;
            break;
        default:
            this.message = 'An unexpected error occurred.';
            console.error('Error enrolling user:', error);
    }
}
```

Define un componente de Angular llamado 'UsersComponent', dentro de nuestra carpeta src->app->pages->users, el cual maneja la lógica relacionada con los cursos, incluyendo la creación, actualización, eliminación y carga de cursos y usuarios.

```
import { Component, OnInit } from '@angular/core';
import { UserService } from '../../services/user.service';
import { FormsModule } from '@angular/forms';
import { User } from '../../models/user.model';
import { lastValueFrom } from 'rxjs';
import { CommonModule } from '@angular/common';

interface UserTableItem {
    user: any;
}

@Component({
    selector: 'app-users',
    standalone: true,
    imports: [FormsModule, CommonModule],
    templateUrl: './users.component.html',
    styleUrls: ['./users.component.css'],
})
export class UsersComponent implements OnInit {
    public userRows: UserTableItem[] = [];
    public showFormModal = false;
```

APLICACIONES DISTRIBUIDAS

```
public users: User[] = [];  
public userForm = {  
  id: 0,  
  name: '',  
  email: '',  
  password: '',  
};  
  
message: string | null = null;  
  
constructor(private userService: UserService) { }  
  
ngOnInit(): void {  
  this.loadUsers();  
  
  try {  
    this.userService.getUsers().subscribe((users: User[])  
=> {  
      this.users = users;  
    });  
  } catch (error) {  
    this.handleError(error);  
    console.error('Ocurrió un error al obtener los  
usuarios');  
    console.error(error);  
  }  
}  
  
public async createUser() {  
  try {  
    const user = await lastValueFrom(  
      this.userService.createUser(this.userForm)  
    );  
    this.userRows.push({ user });  
    await this.loadUsers();  
  } catch (error) {  
    this.handleError(error);  
    console.error('Ocurrió un error al crear el  
usuario');  
    console.error(error);  
  }  
}
```

APLICACIONES DISTRIBUIDAS

```
}

public async updateUser() {
  try {
    const user = await lastValueFrom(
      this.userService.updateUser(this.userForm)
    );
    const index = this.userRows.findIndex((row) =>
row.user.id === user.id);
    this.userRows[index] = { user };
    await this.loadUsers();
  } catch (error) {
    this.handleError(error);
    console.error('Ocurrió un error al actualizar el
usuario');
    console.error(error);
  }
}

public async deleteUser(id: number) {
  try {
    await lastValueFrom(this.userService.deleteUser(id));
    this.userRows = this.userRows.filter((row) =>
row.user.id !== id);
    await this.loadUsers();
  } catch (error) {
    this.handleError(error);
    console.error('Ocurrió un error al eliminar el
usuario');
    console.error(error);
  }
}

private async loadUsers() {
  try {
    const users = await
lastValueFrom(this.userService.getUsers());
    this.userRows = users.map((user: User) => ({ user
}));
  } catch (error) {
    this.handleError(error);
  }
}
```


APLICACIONES DISTRIBUIDAS

```
        console.error('Ocurrió un error al obtener los
usuarios');
        console.error(error);
    }
}

public openFormModal(user: User) {
    if (user) {
        this.userForm = {
            id: user.id ?? 0,
            name: user.name ?? '',
            email: user.email ?? '',
            password: user.password ?? '',
        };
    } else {
        this.userForm = {
            id: 0,
            name: '',
            email: '',
            password: '',
        };
    }
    this.showFormModal = true;
}

private handleError(error: any): void {
    switch (error.status) {
        case 404:
            this.message = 'User or course not found.';
            break;
        case 400:
            this.message = 'Error: ' + error.error.message;
            break;
        default:
            this.message = 'An unexpected error occurred.';
            console.error('Error enrolling user:', error);
    }
}
}
```

APLICACIONES DISTRIBUIDAS

Define un componente de Angular llamado 'EnrollmentComponent', dentro de nuestra carpeta src->app->pages->enrollment, el cual maneja la lógica relacionada con los cursos, incluyendo la creación, actualización, eliminación y carga de cursos y usuarios.

```
import { Component, OnInit } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';
import { EnrollmentService } from
'../../services/enrollment.service';

import { User } from '../../models/user.model';
import { Course } from '../../models/course.model';

@Component({
  selector: 'app-enrollment',
  standalone: true,
  imports: [
    FormsModule,
    CommonModule
  ],
  templateUrl: './enrollment.component.html',
  styleUrls: ['./enrollment.component.css']
})
export class EnrollmentComponent implements OnInit {

  public courses: Course[] = [];
  public users: User[] = [];

  public selectedUserId: number | null = null;
  public selectedCourseId: number | null = null;

  message: string | null = null;

  constructor(private enrollmentService: EnrollmentService) { }

  ngOnInit(): void {
    this.loadCourses();
    this.loadUsers();
  }
}
```

APLICACIONES DISTRIBUIDAS

```
loadUsers(): void {
    this.enrollmentService.getUsers().subscribe(
        (users: User[]) => this.users = users,
        (error) => console.error('Error loading users:',
error)
    );
}

loadCourses(): void {
    this.enrollmentService.getCourses().subscribe(
        (courses: Course[]) => this.courses = courses,
        (error) => console.error('Error loading courses:',
error)
    );
}

enroll(): void {
    if (this.selectedUserId === null || this.selectedCourseId
=== null) {
        this.message = 'Please select both a user and a
course.';
        return;
    }

    this.enrollmentService.enroll(this.selectedUserId,
this.selectedCourseId).subscribe(
        () => {
            this.message = 'User enrolled successfully!';
        },
        (error) => {
            this.handleError(error);
        }
    );
}

private handleError(error: any): void {
    switch (error.status) {
        case 404:
            this.message = 'User or course not found.';
            break;
        case 400:
```

APLICACIONES DISTRIBUIDAS

```
        this.message = 'Error: ' + error.error.message;
        break;
    default:
        this.message = 'An unexpected error occurred.';
        console.error('Error enrolling user:', error);
    }
}
```

4.3. Tecnologías utilizadas

- **Spring Boot:** Framework de Java para desarrollar aplicaciones web y microservicios de manera rápida y con configuración mínima.
- **Java 17:** Versión LTS (Long-Term Support) del lenguaje de programación Java, que ofrece mejoras en el rendimiento, la seguridad y nuevas características del lenguaje.
- **Node.js:** Entorno de ejecución para JavaScript que permite construir aplicaciones escalables y de alto rendimiento.
- **Angular 17:** Framework de desarrollo front-end basado en TypeScript, utilizado para crear aplicaciones web dinámicas
- **Bootstrap:** Framework de CSS para desarrollar interfaces de usuario responsivas y modernas de manera rápida y sencilla.
- **MySQL:** Sistema de gestión de bases de datos relacional, utilizado para gestionar los datos de los usuarios.
- **PostgreSQL:** Sistema de gestión de bases de datos relacional avanzado y de código abierto, utilizado para gestionar los datos de los cursos.

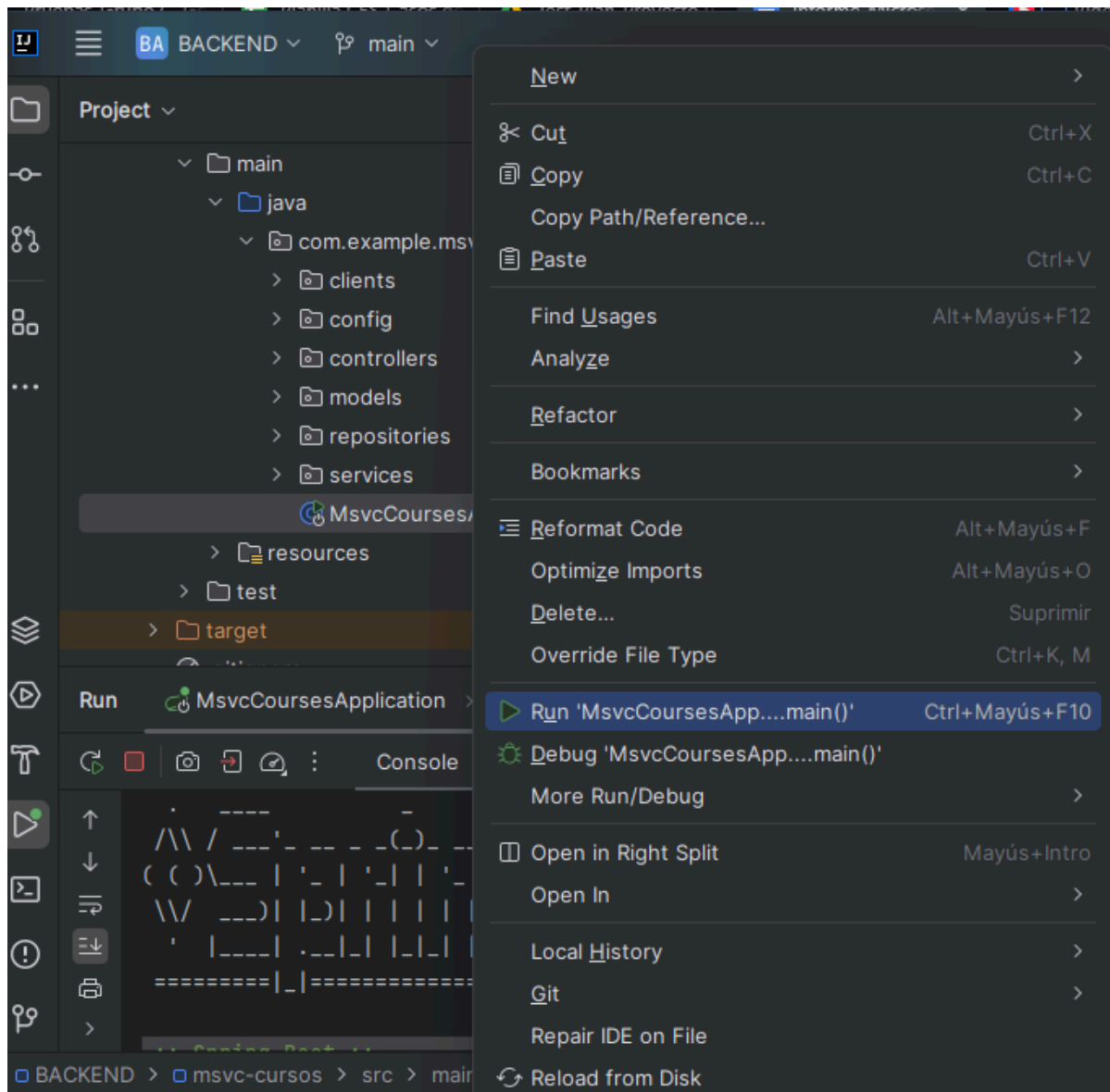
-

5. Ejecución

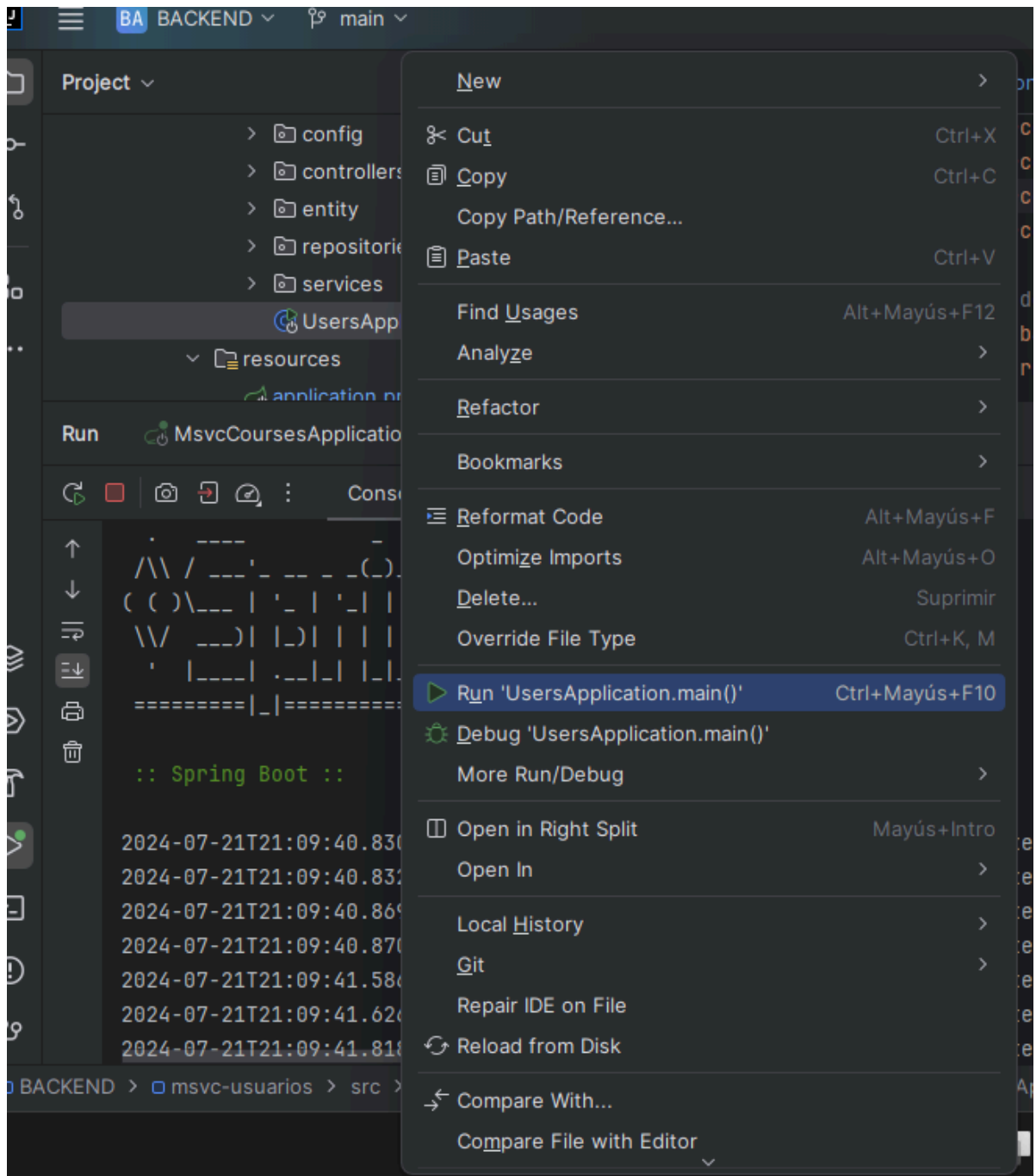
5.1. Primero Ejecutamos el backend

- Abrimos IntelliJ IDEA
- Ejecutamos MSVC-CURSOS y MSVC-USUARIOS

APLICACIONES DISTRIBUIDAS



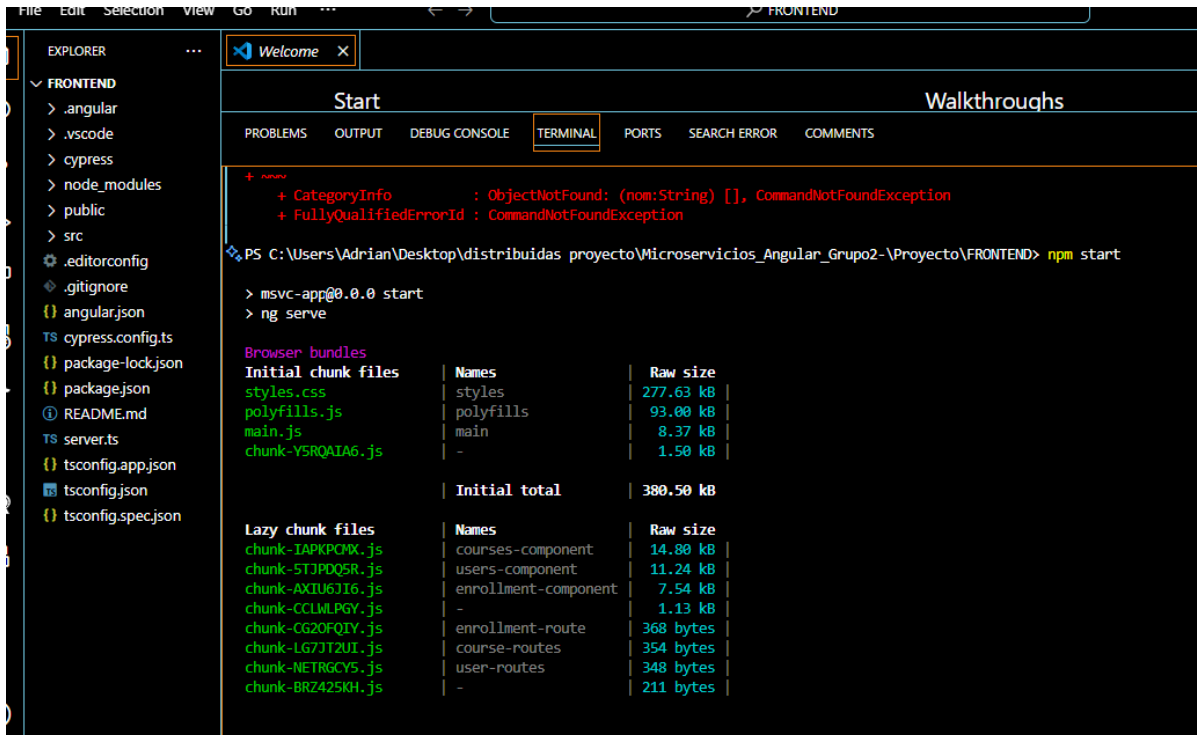
APLICACIONES DISTRIBUIDAS

**Ejecutamos el Frontend**

Abrimos nuestro IDE en este caso visual code, instalamos los módulos con el comando “npm i”

APLICACIONES DISTRIBUIDAS

usamos el comando “npm start” para iniciar



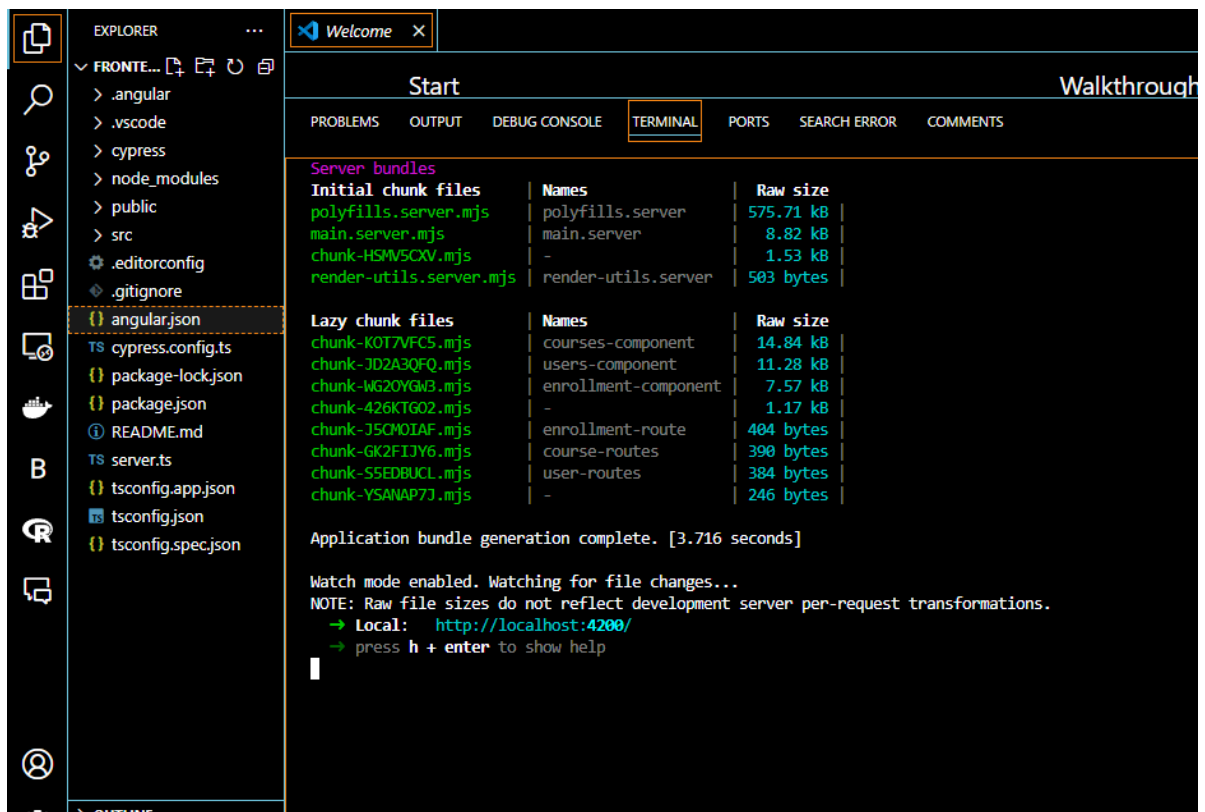
```
PS C:\Users\Adrian\Desktop\distribuidas proyecto\Microservicios_Angular_Grupo2-\Proyecto\FRONTEND> npm start

> msvc-app@0.0.0 start
> ng serve

Browser bundles
Initial chunk files | Names | Raw size
styles.css | styles | 277.63 kB
polyfills.js | polyfills | 93.00 kB
main.js | main | 8.37 kB
chunk-Y5RQIA6.js | - | 1.50 kB

Initial total | 380.50 kB

Lazy chunk files | Names | Raw size
chunk-IAPKPOIX.js | courses-component | 14.80 kB
chunk-STJPDQ5R.js | users-component | 11.24 kB
chunk-AXIU6JI6.js | enrollment-component | 7.54 kB
chunk-CCLWLPGY.js | - | 1.13 kB
chunk-CG20FQIY.js | enrollment-route | 368 bytes
chunk-LG7JT2UI.js | course-routes | 354 bytes
chunk-NETRGYS.js | user-routes | 348 bytes
chunk-BRZ425KH.js | - | 211 bytes
```



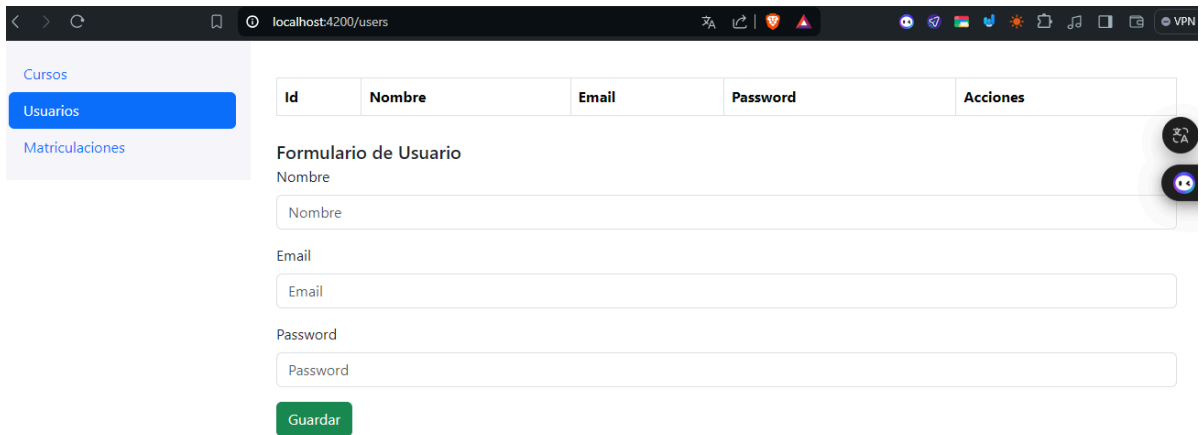
```
Server bundles
Initial chunk files | Names | Raw size
polyfills.server.mjs | polyfills.server | 575.71 kB
main.server.mjs | main.server | 8.82 kB
chunk-HSMVSCXV.mjs | - | 1.53 kB
render-utils.server.mjs | render-utils.server | 503 bytes

Lazy chunk files | Names | Raw size
chunk-KOT7VFC5.mjs | courses-component | 14.84 kB
chunk-JD2A3QFQ.mjs | users-component | 11.28 kB
chunk-WG20YGB3.mjs | enrollment-component | 7.57 kB
chunk-426KTG02.mjs | - | 1.17 kB
chunk-J5CMOIAF.mjs | enrollment-route | 404 bytes
chunk-GK2FIJY6.mjs | course-routes | 390 bytes
chunk-S5EDBUCL.mjs | user-routes | 384 bytes
chunk-YSANAP7J.mjs | - | 246 bytes

Application bundle generation complete. [3.716 seconds]

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
→ Local: http://localhost:4200/
→ press h + enter to show help
```

APLICACIONES DISTRIBUIDAS



Id	Nombre	Email	Password	Acciones
----	--------	-------	----------	----------

Formulario de Usuario

Nombre

Email

Password

Guardar

6. Conclusiones y Recomendaciones

7.1 Conclusiones

- La arquitectura de microservicios, dividida en servicios de usuarios y cursos, permite una alta cohesión y baja dependencia, facilitando el mantenimiento y la escalabilidad del sistema.
- El uso de Feign para la comunicación entre microservicios asegura una integración robusta y eficiente.

7.2 Recomendaciones

- Continuar optimizando la seguridad y el rendimiento del sistema para garantizar la protección de los datos y una experiencia de usuario fluida.
- Mantener un enfoque en la escalabilidad y flexibilidad del sistema para adaptarse a futuras necesidades y cambios en el entorno educativo.

7. Bibliografía

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

Fowler, M., & Lewis, J. (2014). *Microservices*. Retrieved from martinfowler.com.

Seshadri, S. (2018). *Angular Up and Running: Learning Angular Step by Step*. O'Reilly Media.

APLICACIONES DISTRIBUIDAS

- Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.*
- Fowler, M., & Lewis, J. (2014). Microservices. Retrieved from martinowler.com.*
- Seshadri, S. (2018). Angular Up and Running: Learning Angular Step by Step. O'Reilly Media.*
- Rozanski, N., & Woods, E. (2011). Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley.*
- Yourdon, E. (1989). Modern Structured Analysis. Prentice Hall.*
- What is angular? (n.d.). Angular.dev. Retrieved July 19, 2024, from <https://angular.dev/overview>*