



Guía de Actividades Práctico-Experimentales Nro. 002

1. Datos Generales

Asignatura	Desarrollo Basado en Plataformas
Ciclo	5 A
Unidad	1
Resultado de aprendizaje de la unidad	Discute cómo los estándares Web influyen en el desarrollo de software, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.
Práctica Nro.	003
Nombre del Docente	Edison Leonardo Coronel Romero
Fecha	viernes 17 de octubre
Horario	07h30 – 10h30
Lugar	Aula 422
Tiempo planificado en el Sílabo	3 horas

2. Título:

Implementación del flujo de autenticación y autorización en el backend, aplicando mecanismos de seguridad (JWT u OAuth2), validaciones, CORS y principios OWASP Top 10, e incorporación de estos componentes al modelo C4.

3. Objetivo:

Configurar e implementar un mecanismo de autenticación y autorización seguro en el backend del proyecto.

Aplicar políticas CORS, validaciones y manejo de errores según buenas prácticas OWASP.

Documentar el proceso mediante pruebas Postman/Swagger y actualizar los diagramas C4 (Container y Component) reflejando los puntos de seguridad.

4. Materiales y reactivos (Si aplica):

- Computador con acceso a Internet.
- Framework backend (Django REST Framework / Express.js / Spring Boot).
- IDE (VS Code / IntelliJ IDEA).
- Git, GitKraken con flujo GitFlow.
- Postman o Swagger para pruebas.”.

5. Equipos y herramientas

- Equipos personales
- IDE: Visual Studio Code / IntelliJ IDEA.
- Git, GitKraken (flujo GitFlow), Postman o Swagger.
- Repositorio remoto en GitHub.

6. Procedimiento / Metodología

Inicio

- Presentación de los objetivos y repaso de los conceptos JWT/OAuth2.
- Conformación de equipos y revisión de la rama de desarrollo (**feature/security**).

Desarrollo

1. Configuración de entorno

- Instalar dependencias necesarias para seguridad.

Dependencias JWT

```

ATHLETIA athletia > package-lock.json > () packages > () node_modules/@types/jsonwebtoken
athletia
├── .vscode
├── dist
└── node_modules
    ├── src
    └── test
        ├── env
        ├── .gitignore
        ├── .prettierrc
        ├── eslint.config.mjs
        ├── nest-cli.json
        └── package-lock.json
            ├── package.json
            ├── README.md
            ├── tsconfig.build.json
            └── tsconfig.json
    └── docs
        ├── README.md

```

```

3044      "packages": [
3045          "node_modules/@types/json-schema": {
3046              "integrity": "sha512-5+fp8P8MFNC+AyZCDxrB2pkZFPGzqQMUzpSeuiVLvm8VMcorNYavBqoFcxK8bQz4Qsbn4oUEEcHg==",
3047              "dev": true,
3048              "license": "MIT"
3049          },
3050      ],
3051      "node_modules/@types/jsonwebtoken": {
3052          "version": "9.0.10",
3053          "resolved": "https://registry.npmjs.org/@types/jsonwebtoken/-/jsonwebtoken-9.0.10.tgz",
3054          "integrity": "sha512-asxh1G9mf+IoStypjanR7iKTv0gxQ1ov/jfrXgkS/E000Fn18orbmGCn0672NHR3kXHwpAwf",
3055          "license": "MIT",
3056          "dependencies": {
3057              "@types/ms": "*",
3058              "@types/node": "*"
3059          },
3060      },
3061      "node_modules/@types/methods": {
3062          "version": "1.1.4",
3063          "resolved": "https://registry.npmjs.org/@types/methods/-/methods-1.1.4.tgz",
3064          "integrity": "sha512-ymXWVrDiCxtBE3+RlrrP533E70eA+9qu7zdWohu0mgujkytzf4HQF96b8nwHlqhuf4ykX61IGI",
3065          "dev": true,
3066          "license": "MIT"
3067      },

```

Dependencias CORS

```

ATHLETIA athletia > package-lock.json > () packages > () node_modules/core-util-is > license
athletia
├── .vscode
├── dist
└── node_modules
    ├── src
    └── test
        ├── env
        ├── .gitignore
        ├── .prettierrc
        ├── eslint.config.mjs
        ├── nest-cli.json
        └── package-lock.json
            ├── package.json
            ├── README.md
            ├── tsconfig.build.json
            └── tsconfig.json
    └── docs
        ├── README.md

```

```

5063      "packages": [
5064          "node_modules/core-util-is": {
5065              "dev": true,
5066              "license": "MIT"
5067          },
5068          "node_modules/cors": {
5069              "version": "2.8.5",
5070              "resolved": "https://registry.npmjs.org/cors/-/cors-2.8.5.tgz",
5071              "integrity": "sha512-KIHBLJqu73RGr/hnbr09uBeixNGuvSQjuL/jdFVs/KFSIH1hwD1ng7zOHx+YrEfInLG7q4n6C",
5072              "license": "MIT",
5073              "dependencies": {
5074                  "object-assign": "^4",
5075                  "vary": "^1"
5076              },
5077              "engines": {
5078                  "node": "> 0.10"
5079              }
5080          },
5081          "node_modules/cosmiconfig": {
5082              "version": "8.3.6",
5083              "resolved": "https://registry.npmjs.org/cosmiconfig/-/cosmiconfig-8.3.6.tgz",
5084              "integrity": "sha512-kcZ6+w5QzcJ3P1Mt+830Uv+0HFqzHIX8DuxG6eZ5RGMERoLqp4BuGjhHLyGK+kF5XvkvqBSm/",
5085              "dev": true,
5086              "license": "MIT",
5087          },
5088      ],

```



UNL

Universidad
Nacional
de Loja

FEIRNNR - Carrera de computación

Dependencias Argon2 (Se utilizó en lugar de Bcrypt)

```
athletia > package-lock.json > packages
  "packages": [
    "node_modules/append-field": {
      ...
    },
    "node_modules/arg": {
      "version": "4.1.3",
      "resolved": "https://registry.npmjs.org/arg/-/arg-4.1.3.tgz",
      "integrity": "sha512-58S9QDqG0xx27ywPSt9fxjivJYI432YCwfDMFz+7RAqUrZef7LrKQZ3LHLowCS4FLNbplp53:",
      "devOptional": true,
      "license": "MIT"
    },
    "node_modules/argon2": {
      "version": "0.44.0",
      "resolved": "https://registry.npmjs.org/argon2/-/argon2-0.44.0.tgz",
      "integrity": "sha512-ZHgN3S55sihsQo0dbx0A5qpi2R31z7HZDzny3if0yj8bzZnpZN02gpmhnhRG01V/d555RwBc",
      "hasInstallScript": true,
      "license": "MIT",
      "dependencies": {
        "@phc/format": "^1.0.0",
        "cross-env": "^10.0.0",
        "node-addon-api": "8.5.0",
        "node-gyp-build": "4.8.4"
      },
      "engines": {
        "node": ">=16.17.0"
      }
    }
  ]
```

- Crear archivo .env con claves secretas (no versionadas).

Archivo .env

```
athletia > .env
1 SERVER_PORT=3000
2
3 DB_USERNAME=postgres
4 DB_PASSWORD=*****
5 DB_PORT=5432
6 DB_NAME=gym_app
7
8 JWT_SECRET_KEY_ACCESS=*****
9 JWT_SECRET_KEY_REFRESH=*****
10
11 GOOGLE_CLIENT_ID=63339420000000000000000000000000
12 GOOGLE_CLIENT_SECRET=GCPXXXXXXXXXXXXXXXXXXXXXX
13 GOOGLE_CALLBACK_URL=http://localhost:3000/auth/login
```

Archivo .env creado en el entorno de desarrollo con las respectivas claves secretas a utilizar en el proyecto.

2. Implementación del flujo JWT / OAuth2

- Crear rutas /auth/login y /auth/register.

Ruta /auth/login

```
athletia > src > auth > auth.controller.ts > ...
36 @ApiTags('auth')
37 export class AuthController {
38   constructor(private authService: AuthService) {}
39
40   // if the result of processing the request is an error,
41   // an exception will be thrown and handled by the exception filter
42   // otherwise, a successful response will be returned
43
44   @Public()
45   @Post('login')
46   @ApiOperation({ summary: 'Sign in with email and password' })
47   @ApiBody({ type: LoginRequest })
48   @SwaggerResponse({ status: 200, description: 'Token response', type: TokenResponse })
49   @HttpCode(200)
50   async signIn(@Body() loginRequest: LoginRequest): Promise<TokenResponse> {
51     const tokenResponse = await this.authService.signIn(loginRequest);
52     return tokenResponse;
53   }
54 }
```

Ruta /auth/register-account

```

athletia > src > auth > auth.controller.ts > ...
37  export class AuthController {
38    async signIn(@Body() LoginRequest: LoginRequest): Promise<TokenResponse> {
39      return tokenResponse;
40    }
41
42    @Public()
43    @Post('register-account')
44    @ApiOperation({ summary: 'Register a new account' })
45    @ApiBody({ type: RegisterAccountRequest })
46    @SwaggerResponse({ status: 201, description: 'Account created', schema: { example: { success: true, ...
47    async registerAccount(
48      @Body() registerRequest: RegisterAccountRequest,
49    ): Promise<ApiResponse<AccountIdOnly>> {
50      const accountSaved =
51        await this.authService.registerAccount(registerRequest);
52      return ApiResponse.success(
53        { accountId: accountSaved.accountId },
54        accountSaved.message,
55      );
56    }
57  }

```

- Generar token JWT con exp, iat y roles de usuario.

Generación Token JWT dentro de auth.module

```

athletia > src > auth > auth.service.ts > AuthService > createTokenResponse
28  export class AuthService {
29    private createJwtPayload(account: Account): UserPayload {
30      const { email, role } = account;
31      return { email, role };
32    }
33
34    private async createTokenResponse(
35      payload: UserPayload,
36    ): Promise<TokenResponse> {
37      return {
38        accessToken: await this.jwtService.signAsync(payload, {
39          expiresIn: jwtConstants.accessExpiration,
40        }),
41        refreshToken: await this.jwtService.signAsync(payload, {
42          expiresIn: jwtConstants.refreshExpiration,
43          secret: process.env.JWT_SECRET_KEY_REFRESH,
44        }),
45        accountId: payload.sub,
46      };
47    }
48  }
49
50
51
52
53
54
55
56
57
athletia > src > auth > auth.module.ts > ...
5  import { AccountsModule } from 'src/users/accounts/accounts.module';
6  import { PassportModule } from '@nestjs/passport';
7  import { GoogleStrategy } from './strategies/google.strategy';
8  import { MailService } from 'src/common/mail/mail.service';
9
10 @Module({
11   imports: [
12     AccountsModule,
13     PassportModule.register({ session: false }),
14     JwtModule.register({
15       global: true,
16       secret: process.env.JWT_SECRET_KEY_ACCESS || 'defaultSecretKey',
17     }),
18     AccountsModule,
19   ],
20   providers: [
21     AuthService,
22     GoogleStrategy,
23     MailService,
24   ],
25   controllers: [AuthController],
26   exports: [AuthService],
27 })
28 export class AuthModule {}

```



UNL

Universidad
Nacional
de Loja

1859

FEIRNNR - Carrera de computación

Definición del iat del token JWT

Debugger

JSON WEB TOKEN (JWT)

Valid JWT
Kakarotto

Invalid Signature
Paccha #####

eyJhbGciOiJIUzI1NiIsInRScC16IkpxVCJ9.eyJzdWIiOiJjY2EwMzU3My05ODFhLTRlZNETYtY5Ni1mNjI4NjQxNGMzMGUlCJlbwFpbCI6ImFkbWluLmpncmFzb0B1bWFpbC5jb20iLCjb2xIjo1YWRtaW4iLCJpVQIOjE3NjExMDIwMDo5ImV4cCI6MTc2MTEwNTYwNH0..brBw-or47CksDxvutywDE4--LxYdyl9eMxw9IM93vM

JSON CLAIMS TABLE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

DECODED PAYLOAD

JSON CLAIMS TABLE

```
{
  "sub": "cca03573-981a-4eea-a696-f6286414c",
  "email": "admin.jgraso@email.com",
  "role": "admin",
  "iat": 1761102004,
  "exp": 1761105604
}
```

Snipping Tool

Screenshot copied to clipboard
Automatically saved to screenshots folder.

Definición exp del JWT

ATHLETIA

```
athletia > src > auth > constants.ts ...
5  export const messages = {
20    verificationEmailSent: 'Verification email sent',
21    emailAlreadyVerified: 'Email is already verified',
22    tooManyVerificationRequests: 'Too many verification email requests. Try again later.',
23  };

25  export const jwtConstants = {
26    secret: process.env.JWT_SECRET_KEY,
27    // expiration in seconds
28    refreshExpiration: 60 * 60 * 24 * 7, // 7 days
29    accessExpiration: 60 * 60, // 1 hour
30  };
31
32  export const domain = process.env.DOMAIN || 'http://localhost:3000';
33
```

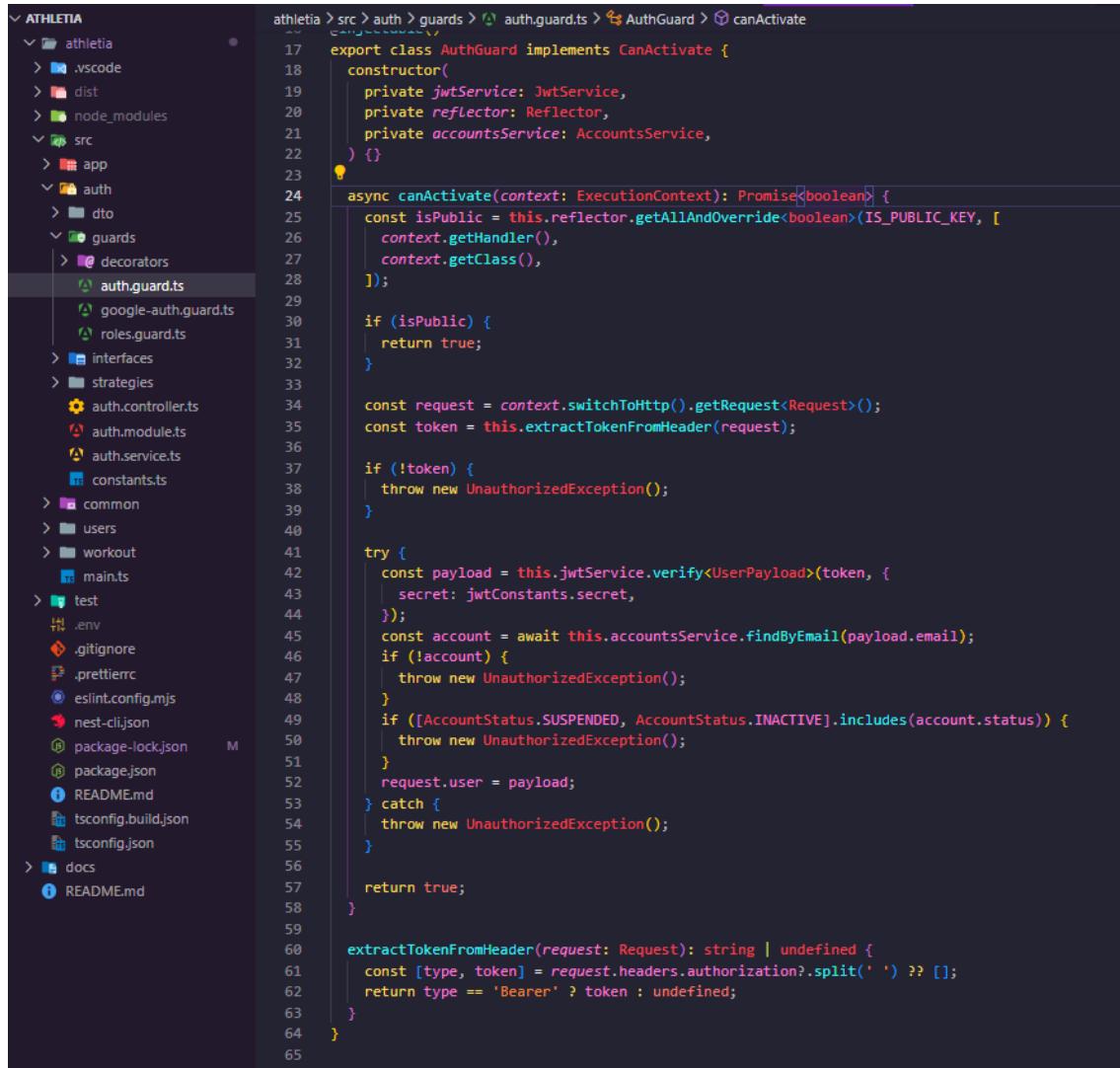
JWTPayload con Roles de Usuario

ATHLETIA

```
athletia > src > auth > auth.service.ts > AuthService
23 // import { GoogleUser } from './strategies/google.strategy';
24
25 dotenv.config();
26
27 @Injectable()
28 export class AuthService {
29   constructor(
30     private accountsService: AccountsService,
31     private jwtService: JwtService,
32     private mailService: MailService,
33   ) {}
34
35   private createJwtPayload(account: Account): UserPayload {
36     return {
37       sub: account.id,
38       email: account.email,
39       role: account.role,
40     };
41   }
42
43   private async createTokenResponse()
```

- Crear middleware de verificación de token y roles (RBAC).

Middleware de Verificación de Token



```

athletia > src > auth > guards > auth.guard.ts > AuthGuard > canActivate
17 export class AuthGuard implements CanActivate {
18   constructor(
19     private jwtService: JwtService,
20     private reflector: Reflector,
21     private accountsService: AccountsService,
22   ) {}
23
24   canActivate(context: ExecutionContext): Promise<boolean> {
25     const isPublic = this.reflector.getAllAndOverride<boolean>(IS_PUBLIC_KEY, [
26       context.getHandler(),
27       context.getClass(),
28     ]);
29
30     if (isPublic) {
31       return true;
32     }
33
34     const request = context.switchToHttp().getRequest<Request>();
35     const token = this.extractTokenFromHeader(request);
36
37     if (!token) {
38       throw new UnauthorizedException();
39     }
40
41     try {
42       const payload = this.jwtService.verify<UserPayload>(token, {
43         secret: jwtConstants.secret,
44       });
45       const account = await this.accountsService.findByEmail(payload.email);
46       if (!account) {
47         throw new UnauthorizedException();
48       }
49       if ([AccountStatus.SUSPENDED, AccountStatus.INACTIVE].includes(account.status)) {
50         throw new UnauthorizedException();
51       }
52       request.user = payload;
53     } catch {
54       throw new UnauthorizedException();
55     }
56
57     return true;
58   }
59
60   extractTokenFromHeader(request: Request): string | undefined {
61     const [type, token] = request.headers.authorization?.split(' ') ?? [];
62     return type === 'Bearer' ? token : undefined;
63   }
64 }
65

```

Roles (RBAC)



```

athletia > src > auth > guards > roles.guard.ts > ...
1 import { CanActivate, ExecutionContext, Injectable } from '@nestjs/common';
2 import { Reflector } from '@nestjs/core';
3 import { Request } from 'express';
4 import { ROLES_KEY } from 'src/auth/guards/decorators/roles.decorator';
5
6 @Injectable()
7 export class RolesGuard implements CanActivate {
8   constructor(private reflector: Reflector) {}
9
10  canActivate(context: ExecutionContext): boolean {
11    const roles = this.reflector.get<string[]>(ROLES_KEY, context.getHandler());
12    if (!roles) {
13      return true;
14    }
15    const request = context.switchToHttp().getRequest<Request>();
16    const user = request.user;
17    return user && roles.includes(user.role);
18  }
19

```



3. Configuración CORS y validaciones

- Definir orígenes permitidos y métodos HTTP.

Orígenes Permitidos y Métodos HTTP

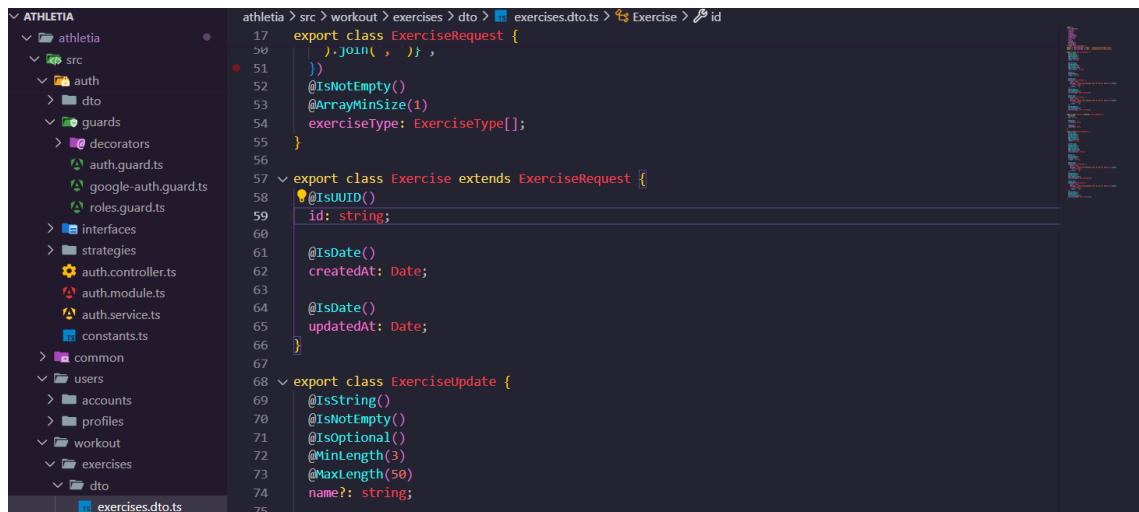
```
athletia > src > main.ts > bootstrap > config
athletia > src > main.ts
7  async function bootstrap() {
8    const app = await NestFactory.create(AppModule);
9    // Secure HTTP headers
10   app.use(
11     helmet({
12       // Keep defaults; customize as needed
13       // Example: only enable HSTS in production behind HTTPS
14       hsts: process.env.NODE_ENV === 'production' ? undefined : false,
15       // Hide X-Powered-By
16       hidePoweredBy: true,
17       // Basicreferrer policy
18       refererPolicy: { policy: 'no-referrer' },
19       // Cross-Origin Resource Policy (adjust for your static hosting if needed)
20       crossOriginResourcePolicy: { policy: 'cross-origin' },
21     }),
22   );
23   app.useGlobalPipes(
24     new ValidationPipe({
25       whitelist: true, // elimina propiedades que no estén en el DTO
26       forbidNonWhitelisted: true, // lanza error si hay propiedades extra
27       transform: true, // convierte los objetos a clases (útil para usar `class-transformer`)
28     }),
29   );
30   app.enableCors({
31     origin: 'http://localhost:3001', // dominio simpático del frontend
32     methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
33     preflightContinue: false,
34     optionsSuccessStatus: 204,
35   });
36
37   // Swagger / OpenAPI
38   const config = new DocumentBuilder()
39     .setTitle('AthletIA API')
40     .setDescription('API documentation for AthletIA')
41     .setVersion(process.env.npm_package_version || '0.0.1')
42     .addBearerAuth({ type: 'http', scheme: 'bearer', bearerFormat: 'JWT' }, 'access-token')
43     .build();
44   const document = SwaggerModule.createDocument(app, config);
45   SwaggerModule.setup('api', app, document);
46   const port = parseInt(process.env.SERVER_PORT!);
47   await app.listen(port);
48 }
void bootstrap();
```

- Agregar validación de entrada (request body y params).

Validación de Entrada

```
23   app.useGlobalPipes(
24     new ValidationPipe({
25       whitelist: true, // elimina propiedades que no estén en el DTO
26       forbidNonWhitelisted: true, // lanza error si hay propiedades extra
27       transform: true, // convierte los objetos a clases (útil para usar `class-transformer`)
28     }),
29   );
```

EJEMPLO DTO: exercises.dto



```

ATHLETIA
└── src
    ├── auth
    ├── dto
    ├── guards
    ├── interfaces
    ├── strategies
    └── workout
        └── exercises
            └── dto
                └── exercises.dto.ts

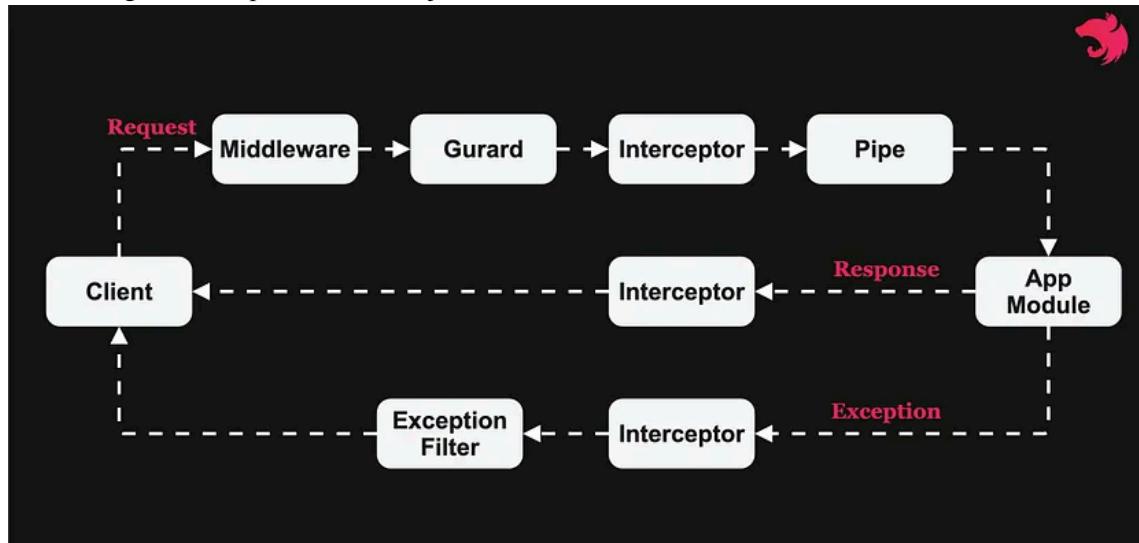
```

```

athletia > src > workout > exercise > dto > exercises.dto.ts > Exercise > id
17 export class ExerciseRequest {
18     ...
19 }
20 @IsNotEmpty()
21 @ArrayMinSize(1)
22 exerciseType: ExerciseType[];
23 }
24
25 export class Exercise extends ExerciseRequest {
26     ...
27     @IsUUID()
28     id: string;
29
30     @IsDate()
31     createdAt: Date;
32
33     @IsDate()
34     updatedAt: Date;
35 }
36
37 export class ExerciseUpdate {
38     ...
39     @IsString()
40     @IsNotEmpty()
41     @IsOptional()
42     @MinLength(3)
43     @MaxLength(50)
44     name?: string;
45 }
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75

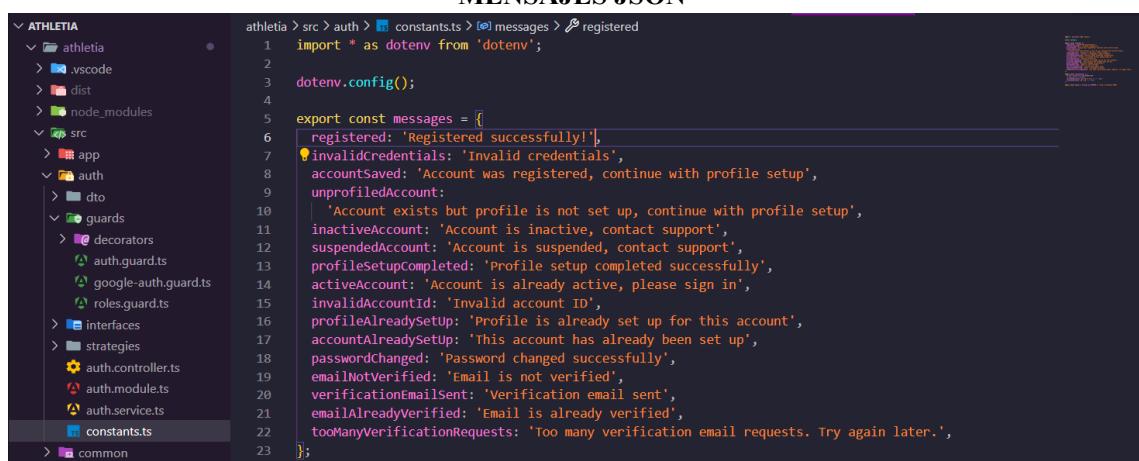
```

- Implementar manejo de errores uniforme (códigos HTTP y mensajes JSON). La tecnología NestJS permite el manejo de errores de forma automática.



Ciclo de Vida de una petición HTTP en NestJS [4]

MENSAJES JSON



```

ATHLETIA
└── src
    ├── auth
    ├── dto
    ├── guards
    ├── interfaces
    ├── strategies
    └── constants
        └── constants.ts

```

```

athletia > src > auth > constants.ts > messages > registered
1 import * as dotenv from 'dotenv';
2
3 dotenv.config();
4
5 export const messages = [
6     registered: 'Registered successfully!',
7     invalidCredentials: 'Invalid credentials',
8     accountSaved: 'Account was registered, continue with profile setup',
9     unprofiledAccount:
10         'Account exists but profile is not set up, continue with profile setup',
11     inactiveAccount: 'Account is inactive, contact support',
12     suspendedAccount: 'Account is suspended, contact support',
13     profileSetupCompleted: 'Profile setup completed successfully',
14     activeAccount: 'Account is already active, please sign in',
15     invalidAccountId: 'Invalid account ID',
16     profileAlreadySetup: 'Profile is already set up for this account',
17     accountAlreadySetup: 'This account has already been set up',
18     passwordChanged: 'Password changed successfully',
19     emailNotVerified: 'Email is not verified',
20     verificationMailSent: 'Verification email sent',
21     emailAlreadyVerified: 'Email is already verified',
22     tooManyVerificationRequests: 'Too many verification email requests. Try again later.'
23 ];

```



UNL

Universidad
Nacional
de Loja

FEIRNNR - Carrera de computación

4. Pruebas y verificación

- Probar login y rutas protegidas con Postman/Swagger.

Prueba de Login en Postman

The screenshot shows the Postman interface with the following details:

- Collection:** AthletIA
- Request:** POST /auth/login
- Body:** raw JSON ({{admin-user}})
- Response Status:** 200 OK
- Response Body:**

```
1 {  
2     "accessToken": "eyJhbGciOiJIUzI1NiIsInR6cC16IkpxVC39.  
3         eyJzdWI1IiZodVmNz11SmM0YyLTmMQtYmV1ZSlkyzhj0MwMTB1YTaiLCJlbWFpbCI6ImFkbWluLmpncmFzb0BlMFpbC5jb20ilCJ  
4         yb2x1IjoiYWRtaW4ilCjoYXQ10jE3NjEyTQxNzMsInV4ccC16Mt-2MTi1Nzc3M38.  
5             LLS2pJw.x04x_u9aRVlQkcmfmlwNPtNtatM2D8wug",  
6     "refreshToken": "eyJhbGciOiJIUzI1NiIsInR6cC16IkpxVC39.  
7         eyJzdWI1IiZodVmNz11SmM0YyLTmMQtYmV1ZSlkyzhj0MwMTB1YTaiLCJlbWFpbCI6ImFkbWluLmpncmFzb0BlMFpbC5jb20ilCJ  
8         yb2x1IjoiYWRtaW4ilCjoYXQ10jE3NjEyTQxNzMsInV4ccC16Mt-2MTg100k3M38.  
9             Q1nizJF9J03ghlvU83jF9isG1YoAgRjm0Xm_x0lu",  
10    "accountId": "3865f79ee-f062-4f34-beee-dc8c9ed10ea0"
```

Prueba de Register en Postman

The screenshot shows the Postman interface with the following details:

- Collection:** AthletIA
- Request:** POST /auth/register-account
- Body:** raw JSON ({{normal-user}})
- Response Status:** 201 Created
- Response Body:**

```
1 {  
2     "success": true,  
3     "message": "Account was registered, continue with profile setup",  
4     "data": {  
5         "accountId": "3eadb8b-1d56-47c3-9269-6bd4711e28e9"  
6     }  
7 }
```

- Documentar resultados (capturas de respuesta 200, 401, 403).

Código de éxito 200

The screenshot shows the Postman interface with a successful API call. The URL is `https://AthletIA / Workout / Exercises / GetAll`. The method is `GET` and the endpoint is `{{domain}} /workout/exercises`. The Headers tab shows 6 hidden headers. The Body tab shows a JSON response with 16 lines of code. The response is 200 OK with a 13 ms duration and 1.25 KB size.

```

1 {
2   "success": true,
3   "message": "Exercises retrieved successfully",
4   "data": [
5     {
6       "id": "4a7198c0-81a4-4f49-a173-e964dab4e127",
7       "name": "T Row",
8       "description": "Exercise for building a strong Back.",
9       "video": "https://www.youtube.com/shorts/nwdU1Kb5Ms4",
10      "createdAt": "2025-10-24T02:27:15.426Z",
11      "updatedAt": "2025-10-24T02:27:15.426Z",
12      "exerciseType": [
13        "bodybuilding",
14        "strength",
15        "powerlifting"
16      ]
    }
  ]
}
  
```

Código de Error 401

The screenshot shows an unauthorized API call. The URL is `https://AthletIA / auth / login`. The method is `POST` and the endpoint is `{{domain}} /auth/login`. The Headers tab shows 8 headers. The Body tab shows a raw JSON payload with 4 lines of code. The response is 401 Unauthorized with a 141 ms duration and 931 B size.

```

1 {
2   "email": "admin.jgraso@email.com",
3   "password": "administrator123"
4 }
  
```

```

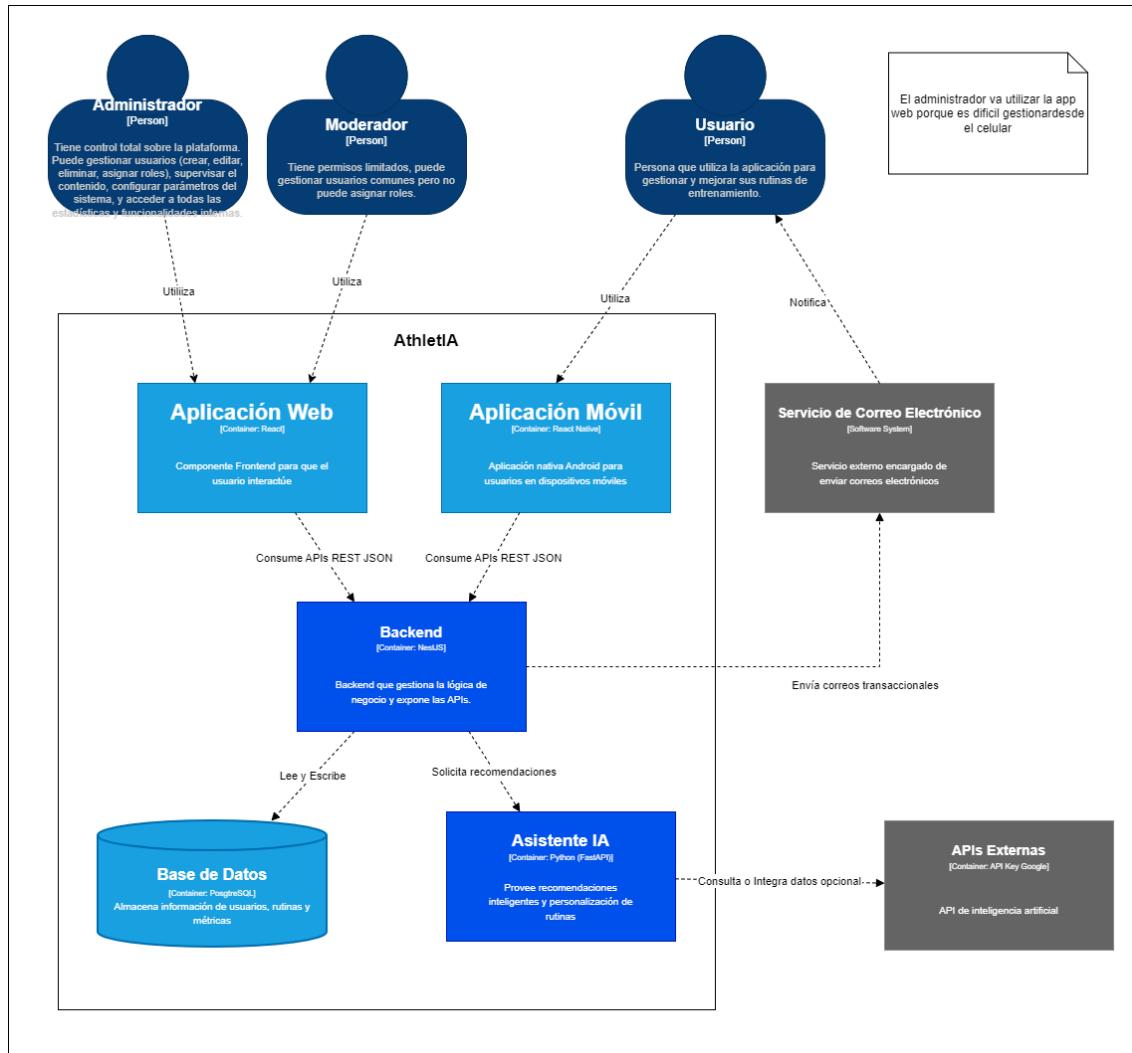
1 {
2   "message": "Unauthorized",
3   "statusCode": 401
4 }
  
```



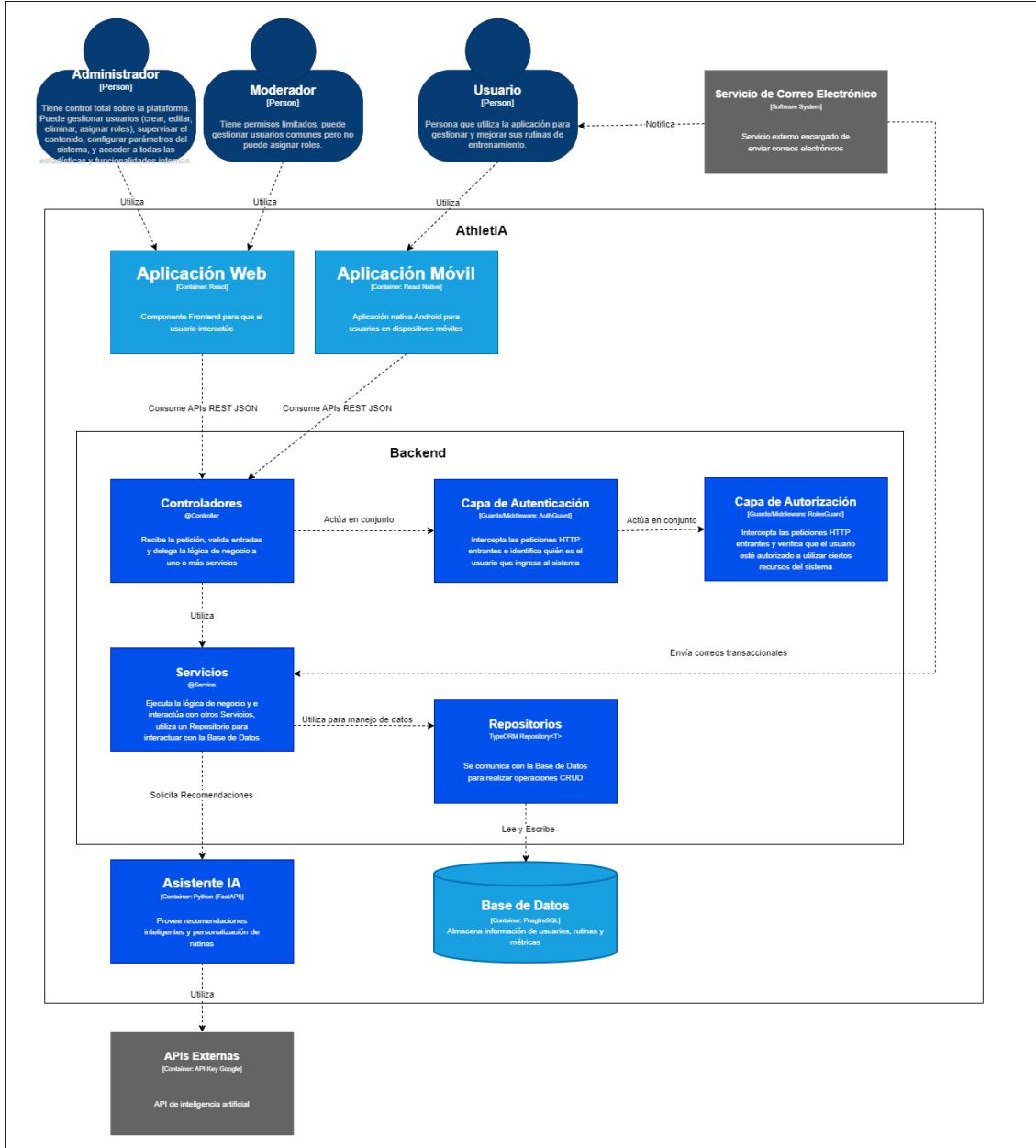
5. Modelo C4 con seguridad

- Actualizar diagramas Container y Component para incluir los servicios de autenticación y módulos de seguridad.

Modelo C4 - Nivel 2 - Contenedor



Modelo C4 - Nivel 3 - Component





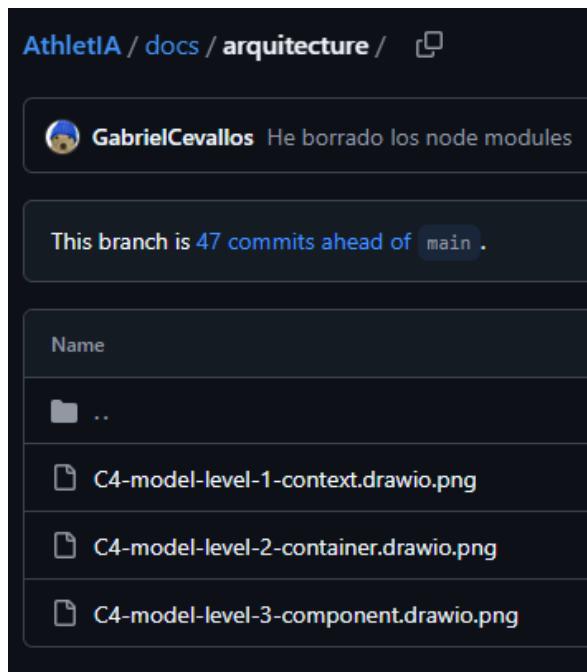
UNL

Universidad
Nacional
de Loja

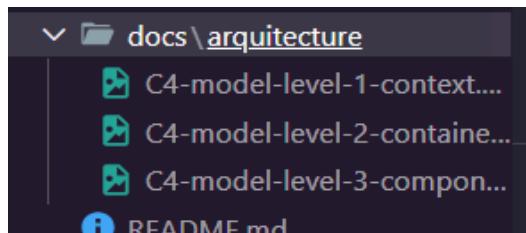
FEIRNNR - Carrera de computación

- Guardar los diagramas actualizados en [/docs/architecture/](#).

Diagramas Actualizados en Repositorio GitHub



Diagramas Actualizados en el Proyecto



Cierre

- Socialización de resultados entre equipos.
- Retroalimentación docente sobre configuración y buenas prácticas OWASP.

7. Resultados esperados:

- Backend con flujo JWT u OAuth2 funcional.
- Políticas CORS y validaciones implementadas.
- Colección Postman/Swagger con pruebas de autenticación.
- Diagramas C4 actualizados mostrando componentes de seguridad.
- Evidencias (capturas de pantalla + README actualizado).

8. Preguntas de Control:

1. **¿Cuál es la diferencia fundamental entre autenticación y autorización dentro de un sistema backend?**

La autenticación verifica las identidades de los usuarios, mientras que la autorización controla sus derechos de acceso dentro de una aplicación [1].

2. **¿Qué ventajas ofrece JWT frente a sesiones tradicionales de servidor y qué vulnerabilidades puede tener?**

Ventajas:

- **Interoperabilidad:** Un JWT puede transmitirse fácilmente entre varias aplicaciones, incluso si utilizan tecnologías diferentes [2].
- **Gestión simplificada:** A diferencia de las sesiones tradicionales, no es necesario almacenar un estado en el lado del servidor [2].
- **Autenticación eficiente:** Simplemente verificando la forma del token se autentica al usuario, sin necesidad de consultar una base de datos o un sistema de sesiones [2].
- **Formato compacto y legible:** El formato (JSON codificado en base64url) es una forma compacta y legible de transportar datos, útil en arquitecturas distribuidas o APIs [2].

Vulnerabilidades:

- **Firma del JWT no verificada:** Ocurre cuando el servidor no comprueba la firma del token y simplemente decodifica el contenido para leerlo [2].
- **Explotación de algoritmo “none”:** Un atacante modifica la cabecera del token para establecer el algoritmo como “none”. Si el servidor está mal configurado acepta este algoritmo, valida el token sin verificar ninguna firma [2].
- **Secretos débiles y ataques de fuerza bruta:** Si se utiliza un algoritmo simétrico, la clave secreta compartida puede ser adivinada mediante ataques de fuerza bruta si es demasiado simple o predecible [2].

3. **Explique cómo CORS protege (o restringe) la comunicación entre cliente y servidor.**

CORS introduce el acceso medido, lo que ayuda a reducir el riesgo de amenazas e infracciones de seguridad. Estas son las formas en que lo hace.

- **Evita el acceso no autorizado:** CORS especifica qué dominios pueden acceder a sus recursos, lo que reduce el riesgo de exposición de datos [3].
- **Mitiga la violación de datos:** CORS controla el acceso a través de encabezados definidos. Eso garantiza que los intrusos no puedan acceder a los datos [3].
- **Directivas predefinidas:** Las directivas de CORS solo permiten solicitudes específicas. Al hacerlo, protege el sistema de ataques y violaciones de seguridad [3].

4. **Mencione tres vulnerabilidades del OWASP Top 10 que podrían afectar su API y cómo las mitigaría.**

1. Inyección (A03)

- **¿Qué pasa?:** Consultas SQL/NoSQL o comandos con datos no válidos permiten ejecución de código/inyección y robo/modificación de datos
- **Mitigación:**
 - Validar y sanitizar entradas con class-validator + ValidationPipe
 - Evitar eval/exec y construir queries dinámicos solo con listas blancas.
 - Revisar logs para detectar patterns y añadir pruebas que intenten payloads de inyección.

**2. Broken Access Control (A01)**

- **¿Qué pasa?:** Usuarios acceden a recursos ajenos (IDs, endpoints, admin) por falta de verificación de roles/propietario.
- **Mitigación:**
 - Implementar guards y middleware RBAC (RolesGuard) y verificar propiedad del recurso (owner checks) en services.
 - No confiar solo en datos del cliente.
 - Principio de menor privilegio en endpoints y datos retornados.
 - Probar escenarios: escalado de privilegios, acceso directo por id, endpoints admin.

3. Cryptographic Failures (A02)

- **¿Qué pasa?:** Secretos débiles, tokens no rotados, contraseñas almacenadas en texto, errores con trazas que exponen información.
- **Mitigación:**
 - Hashing de contraseñas y evitar logging de contraseñas ni tokens.
 - Usar JTW secret fuerte en .env, expiración corta para access tokens y refresh tokens con almacenamiento seguro.
 - No publicar .env en repo, revisar .gitignore, evitar mensajes de error con stack trace en producción.
 - Usar rotación de claves y revocación de refresh tokens (lista negra) si es necesario.

5. ¿En qué parte del modelo C4 se deben representar las capas o componentes de seguridad y por qué?

En nuestro caso lo colocamos en la capa 3 (nivel de componentes) porque al expandir el módulo del backend se muestra la existencia de la capa de autenticación y autorización.

6. ¿Qué buenas prácticas debe seguir al almacenar contraseñas y manejar tokens en su proyecto?

- Hashing de contraseñas con Argon2.
- No almacenar ni hacer logging de contraseñas en texto plano.
- Validar fuerza de contraseña y limitar intentos de login
- Tener un reset seguro, es decir un token de un solo uso con respiración corta y guardar el token hashed.
- Access tokens cortos, refresh tokens más largos pero rotables y guardados después del hashing en BD.
- HTTPS, CORS restringidos.
- No subir .env a repositorio

9. Conclusiones:

Se implementó de manera exitosa el flujo de autenticación mediante JWT y OAuth2, lo que incluye la instalación de las dependencias y la configuración necesaria para poder ejecutar estas funcionalidades, dentro de lo cual se incluye también el uso de variables de entorno para la gestión de claves secretas e información que no debe ser de conocimiento público.

Se aplicaron algunas prácticas del desarrollo seguro como el manejo de restricciones para orígenes desconocidos, la distinción entre los entornos de desarrollo y producción, utilización demiddlewares para verificar la autenticidad de los emisores de las peticiones HTTP así como sus permisos dentro de nuestro sistema y una intensiva sesión de pruebas en la herramienta Postman que nos permite asegurar un funcionamiento correcto, seguro y limpio del módulo de autenticación y autorización.

10. Recomendaciones:

- Utilizar el archivo .gitignore para evitar exponer las claves secretas al repositorio remoto.
- Antes de utilizar cualquier biblioteca o herramienta, verificar que no esté obsoleta y que no presente fallos de seguridad o vulnerabilidades graves conocidas.
- Utilizar diagramas en lugar de texto para comprender de mejor manera el flujo de los procesos de autenticación y autorización.
- Realizar pruebas con la herramienta Postman para los endpoints de autenticación.
- Automatizar procesos repetitivos en el entorno de desarrollo, por ejemplo la preparación del IDE.
- Documentar de manera clara el esquema de las peticiones para evitar un uso incorrecto de la API.
- Utilizar IA generativa de manera ética y responsable para agilizar los procesos de construcción y pruebas.
- Configurar el entorno de ejecución antes de probar el sistema (en la fase de desarrollo).

11. Evaluación

Criterio	2 – Logro Alto	1 – Logro Medio	0 – Bajo / Sin Evidencia
1. Implementación de autenticación y autorización (JWT/OAuth2)	Flujo completo, tokens válidos y rutas protegidas operativas.	Flujo parcial o errores de validación de token.	No implementa autenticación funcional.
2. Configuración de CORS y validaciones	Configuración correcta, verificada en pruebas.	Parcial o con advertencias en consola.	Sin configuración verificable.
3. Aplicación de principios OWASP Top 10	Checklist completo y medidas de mitigación documentadas.	Checklist parcial o sin evidencia de mitigación.	No evidencia revisión OWASP.
4. Actualización del modelo C4 (Container y Component)	Diagramas actualizados y coherentes con las modificaciones de seguridad.	Diagramas incompletos o sin claridad en los componentes de seguridad.	No presenta actualización del C4.
5. Documentación y entrega de evidencias	PDF y README completos con capturas y referencias a la implementación.	Entrega parcial o poco clara.	No entrega evidencias o sin documentación.



12. Bibliografía

- OWASP Foundation. (2023). OWASP Top 10 – Web Application Security Risks.
 - Auth0. JWT Handbook. <https://auth0.com/learn/json-web-tokens>
 - Spring Security / Django Auth / Express JWT docs.
 - PlantUML / Mermaid Model C4 Reference.
- [1] R. Raj, “Demystifying authentication and authorization in backend systems”, *Medium*, 10-may-2024. [En línea]. Disponible en: <https://medium.com/@rohitraj1912000/demystifying-authentication-and-authorization-in-backend-systems-52489c3fae8c>. [Consultado: 23-oct-2025].
- [2] “JWT (JSON Web Token): Vulnerabilities, common attacks and security best practices”, *VAADATA - Ethical Hacking Services*, 30-abr-2025. [En línea]. Disponible en: <https://www.vaadata.com/blog/jwt-json-web-token-vulnerabilities-common-attacks-and-security-best-practices/>. [Consultado: 23-oct-2025].
- [3] “Secure API configurations: Key CORS headers for safe resource sharing”, *Contentstack*. [En línea]. Disponible en: <https://www.contentstack.com/blog/tech-talk/secure-api-configurations-key-cors-headers-for-safe-resource-sharing>. [Consultado: 23-oct-2025].
- [4] Dan. “NestJS Request Lifecycle: A Complete Guide to the Architecture”. Medium. Accedido el 23 de octubre de 2025. [En línea]. Disponible: <https://medium.com/@daiki01240/nestjs-request-lifecycle-a-complete-guide-to-the-architecture-5ada9666867a>

13. Elaboración y Aprobación

Elaborado por	Edison L Coronel Romero Docente	
Aprobado por	Edison L Coronel Romero Director de Carrera	