

**PRUEBA****PRACTICO****Programación Avanzada**

Nombre: Esteban Rivera

Introducción

Desarrollo de un sistema Web que Gestione inventario de una empresa "XWY"

Este proyecto consiste en un **sistema web de inventario y préstamos** desarrollado con una arquitectura en capas, orientado a la gestión de artículos, clientes, empleados y préstamos dentro de una organización. Fue construido con un enfoque moderno tanto en el backend como en el frontend, garantizando seguridad, escalabilidad y una interfaz amigable para el usuario.

Tecnologías y Arquitectura

- **Backend:** ASP.NET Core Web API (.NET 8), Entity Framework Core, JWT para autenticación, arquitectura por capas (Entities, DataAccess, Business, API)
- **Frontend:** HTML, CSS moderno, JavaScript puro (sin frameworks externos), utilizando fetch para llamadas a la API y localStorage para gestionar sesiones.
- **Base de datos:** SQL Server, con relaciones correctamente modeladas entre usuarios, empleados, roles, artículos y préstamos.

Control de acceso

El sistema diferencia entre:

- **Admin:** puede crear, editar, eliminar y visualizar todos los datos.
- **Operator:** puede consultar y registrar información limitada (como préstamos o clientes).
- **Autenticación:** basada en JWT, con claims personalizados para rol y employeeId.

Funcionalidades principales

- Registro y login de usuarios (con hash de contraseña).
- Gestión completa de artículos, clientes, empleados y préstamos.
- Vistas protegidas por rol con políticas [Authorize(Policy = "...")].
- Dashboard visual con resumen y gráficas.
- Exportación a PDF/Excel y diseño responsive.
- Alertas visuales, confirmaciones para editar/eliminar y validaciones claras.

Requerimientos Funcionales

ID	Descripción	Criterio de aceptación
RF1	Gestión de usuarios y roles	
RF1.1	Registro de nuevo usuario con nombre, email y contraseña	Al enviar el formulario, el usuario queda creado en la base de datos con rol por defecto "Operador".
RF1.2	Inicio de sesión	Permite autenticar con email/contraseña; redirige según rol.



RF1.3	Asignación y cambio de rol	Sólo usuarios “Administrador” pueden asignar roles (Administrador u Operador).
RF1.4	Control de acceso	Páginas y acciones protegidas según rol; intento no autorizado redirige a “Acceso denegado”.
RF2 Gestión de artículos		
RF2.1 Crear artículo Formulario con campos Código, Nombre, Categoría, Estado, Ubicación; guarda en BD.		
RF2.2 Editar artículo Permite modificar todos los campos salvo ID; valida unicidad de código.		
RF2.3 Eliminar artículo Sólo si no existe un préstamo activo; muestra confirmación.		
RF2.4 Listar y paginar artículos Despliega tabla con paginación y orden por Nombre o Código.		
RF2.5 Filtrar y buscar artículos Filtrado por categoría y estado, búsqueda por nombre o código en tiempo real.		
RF3 Gestión de préstamos		
RF3.1 Solicitar préstamo Selección de artículo disponible y fecha de entrega; crea registro con estado “Pendiente”.		
RF3.2 Aprobación / rechazo Usuario Administrador aprueba o rechaza; cambia estado del préstamo.		
RF3.3 Registrar devolución Asignar fecha de devolución; cambia estado a “Devuelto” y libera artículo.		
RF3.4 Consultar historial Vista con listado de todos los préstamos (activos e históricos), filtrable por usuario, artículo o estado.		
RF4 Generación de reportes		
RF4.1 Exportar listado de artículos a PDF Botón “Exportar PDF” genera documento con columnas clave.		
RF4.2 Exportar listado de préstamos a Excel Botón “Exportar Excel” genera archivo .xlsx con datos de préstamo.		

Requerimientos No Funcionales

Categoría	Requerimiento
Seguridad	• Contraseñas hasheadas (BCrypt o ASP .NET Identity).• Protección CSRF/XSS en formularios. • HTTPS obligatorio.
Rendimiento	• Tiempo de respuesta \leq 200 ms para operaciones CRUD. • Soportar al menos 50 usuarios concurrentes sin degradación.
Usabilidad	• UI responsive (Bootstrap o Tailwind). • Mensajes de validación claros. • Navegación consistente y accesible.
Mantenibilidad	• Código organizado por capas y proyectos. • Uso de SOLID y patrones Repository/Unit of Work. • Comentarios en código y README.
Escalabilidad	• Arquitectura modular monolítica, preparada para futura migración a microservicios. • Dependencias inyectadas.
Disponibilidad	• Tasa de disponibilidad objetivo 99.5 %. • Backups automáticos diarios de la BD. • Manejo de errores centralizado (Serilog).
Portabilidad	• Compatible con Windows y Linux (.NET 6+). • Contenerización opcional con Docker.
Localización	• Soporte para español e inglés (recursos .resx).
Compatibilidad	• SQL Server 2019+. • Navegadores modernos: Chrome, Edge, Firefox.
Auditoría	• Registro de acciones críticas (login, aprobación de préstamo, exportación).

1. Contexto y descripción

La aplicación será una solución de Inventario que permita:

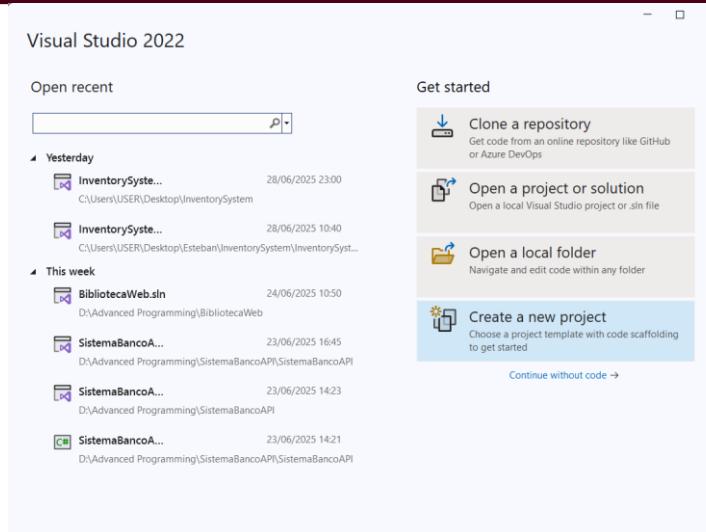


Ilustración 1: Crear un nuevo proyecto

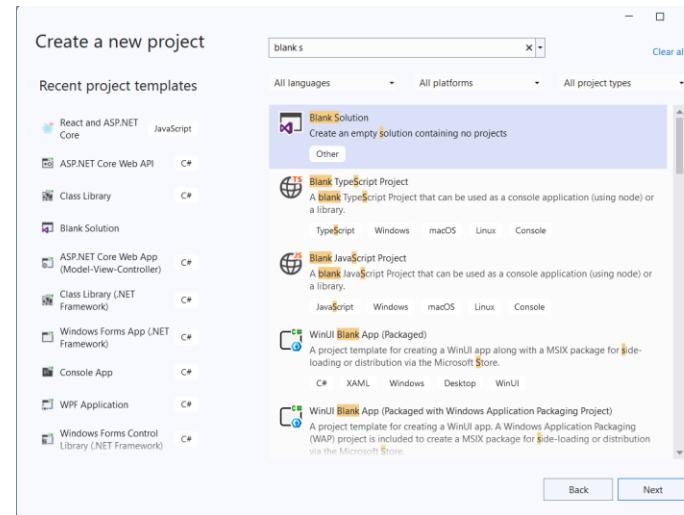


Ilustración 2: Solución vacía para n-capas

- Gestionar usuarios y roles.
- Dar de alta, editar y listar artículos.
- Registrar préstamos y devoluciones.
- **Generar reportes sencillos (PDF o Excel). (examen) – no aplica**

Implementa una **arquitectura N-capas** con al menos las capas:

1. Presentación API

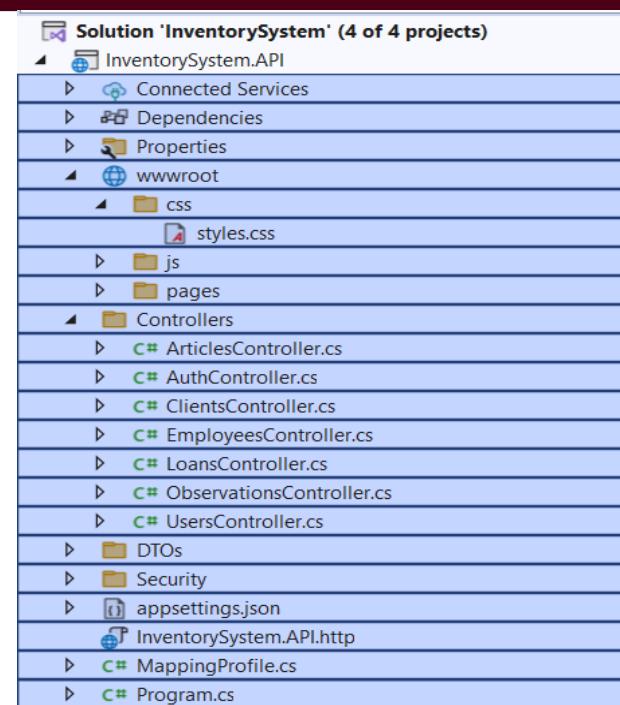


Ilustración 3: Capa de presentación API

2. Lógica de negocio (proyecto de clases)

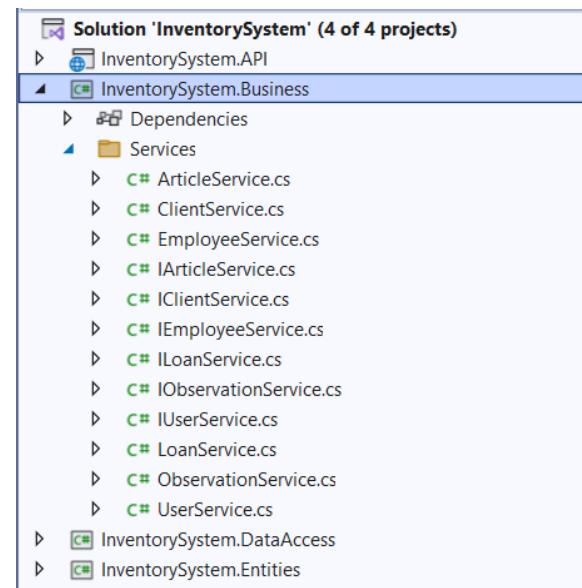


Ilustración 4: Capa de negocio – Class Library

3. Acceso a datos (Entity Framework Core sobre SQL Server)

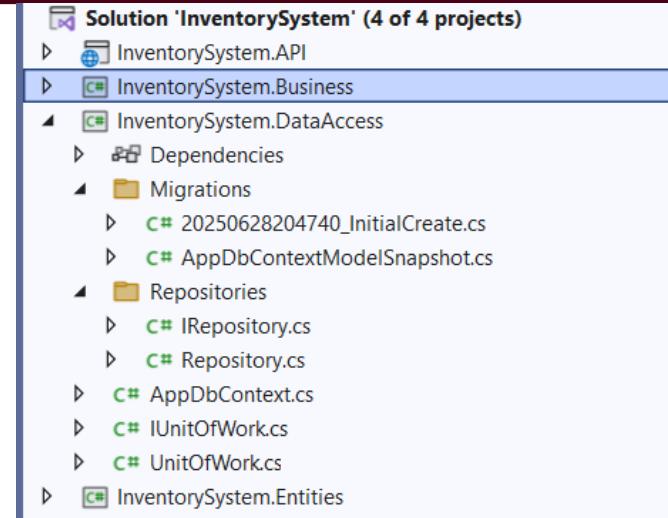


Ilustración 5: Capa de enlace de datos – Class Library

4. Capa de entidades/modelos (clases POCO)

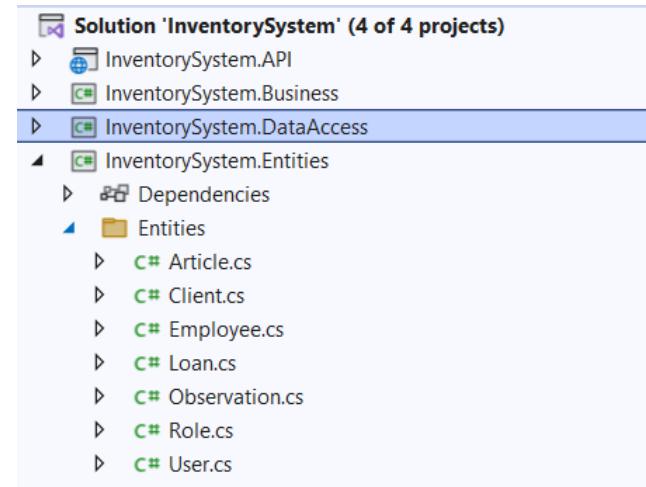


Ilustración 6: Capa entities - Class Library

2. Requisitos funcionales

1. Autenticación y autorización

- Registro e inicio de sesión de usuarios con roles (Administrador, Operador).



```
AuthController.cs
16 [ApiController]
17 [Route("api/{controller}")]
18 public class AuthController : ControllerBase
19 {
20     private readonly AppDbContext _context;
21     private readonly JwtSettings _jwt;
22 
23     public AuthController(AppDbContext context, IOptions<JwtSettings> jwtOptions)
24     {
25         _context = context;
26         _jwt = jwtOptions.Value;
27     }
28 
29     [HttpPost("login")]
30     public IActionResult Login([FromBody] LoginDto dto)
31     {
32         // Simulación de búsqueda de usuario con contraseña hasheada
33         var user = _context.Users
34             .FirstOrDefault(u => u.Email == dto.Email && u.PasswordHash == Convert.ToBase64String(Encoding.UTF8.GetBytes(dto.Password)));
35 
36         if (user == null)
37             return Unauthorized(new { message = "Invalid credentials" });
38 
39         var employee = _context.Employees.FirstOrDefault(e => e.UserId == user.UserId);
40         var roleName = employee != null
41             ? _context.Roles.FirstOrDefault(r => r.RoleId == employee.RoleId)?.Name ?? "Unknown"
42             : "Operator"; // Default for non-admins
43 
44         var token = GenerateToken(user.Email, roleName);
45 
46         return Ok(new
47         {
48             token,
49             employeeId = employee?.EmployeeId ?? Guid.Empty, // Nunca null
50             role = roleName
51         });
52     }
53 
54     [AllowAnonymous]
55     [HttpPost("register")]
56     public async Task<ActionResult> Register([FromBody] CreateUserDto dto)
57     {
58         var emailExists = await _context.Users.AnyAsync(u => u.Email == dto.Email);
59         if (emailExists)
60             return BadRequest(new { message = "Email already exists." });
61     }
62 }
```

Ilustración 7: Controllers para el ingreso de usuarios

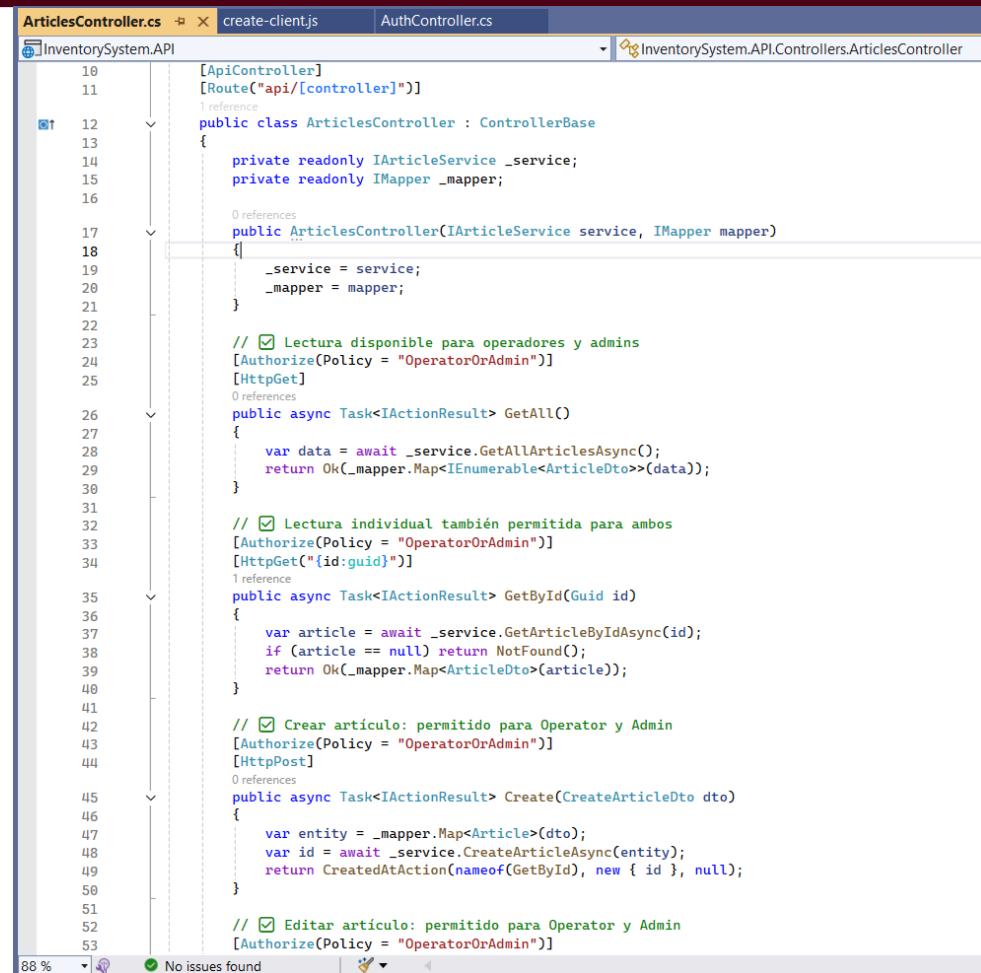
- Control de acceso a páginas según rol.

```
const token = localStorage.getItem('token');
if (!token) window.location.href = "/pages/login.html";
if (role !== "Admin") {
    alert("Access denied");
    window.location.href = "/pages/dashboard.html";
}
```

Ilustración 8: Método JavaScript para control de acceso según rol

2. Gestión de Artículos

- CRUD de Artículo (Código, Nombre, Categoría, Estado, Ubicación).



```
10 [ApiController]
11 [Route("api/[controller]")]
12 1 reference
13 public class ArticlesController : ControllerBase
14 {
15     private readonly IArticleService _service;
16     private readonly IMapper _mapper;
17 
18     public ArticlesController(IArticleService service, IMapper mapper)
19     {
20         _service = service;
21         _mapper = mapper;
22     }
23 
24     // [Authorize(Policy = "OperatorOrAdmin")]
25     [HttpGet]
26     public async Task<IActionResult> GetAll()
27     {
28         var data = await _service.GetAllArticlesAsync();
29         return Ok(_mapper.Map<IEnumerable<ArticleDto>>(data));
30     }
31 
32     // [Authorize(Policy = "OperatorOrAdmin")]
33     [HttpGet("{id:guid}")]
34     public async Task<IActionResult> GetById(Guid id)
35     {
36         var article = await _service.GetArticleByIdAsync(id);
37         if (article == null) return NotFound();
38         return Ok(_mapper.Map<ArticleDto>(article));
39     }
40 
41     // [Authorize(Policy = "OperatorOrAdmin")]
42     [HttpPost]
43     public async Task<IActionResult> Create(CreateArticleDto dto)
44     {
45         var entity = _mapper.Map<Article>(dto);
46         var id = await _service.CreateArticleAsync(entity);
47         return CreatedAtAction(nameof(GetById), new { id }, null);
48     }
49 
50     // [Authorize(Policy = "OperatorOrAdmin")]
51     [HttpPut("{id:guid}")]
52     public async Task<IActionResult> Update(Guid id, UpdateArticleDto dto)
53     {
54         var entity = _mapper.Map<Article>(dto);
55         var result = await _service.UpdateArticleAsync(id, entity);
56         if (result == null) return NotFound();
57         return Ok(result);
58     }
59 }
```

Ilustración 9: Gestión de artículos según el rol

- Búsqueda y filtrado por categoría y estado.

```
// Filtros
function populateFilters() {
    const clientSelect = document.getElementById("clientFilter");
    clients.forEach(c => {
        const opt = document.createElement("option");
        opt.value = c.clientId;
        opt.textContent = `${c.firstName} ${c.lastName}`;
        clientSelect.appendChild(opt);
    });

    const articleSelect = document.getElementById("articleFilter");
    articles.forEach(a => {
        const opt = document.createElement("option");
        opt.value = a.articleId;
        opt.textContent = a.name;
        articleSelect.appendChild(opt);
    });

    document.querySelectorAll(".filters select").forEach(select => {
        select.addEventListener("change", () => {
            const client = document.getElementById("clientFilter").value;
            const article = document.getElementById("articleFilter").value;
            const status = document.getElementById("statusFilter").value;

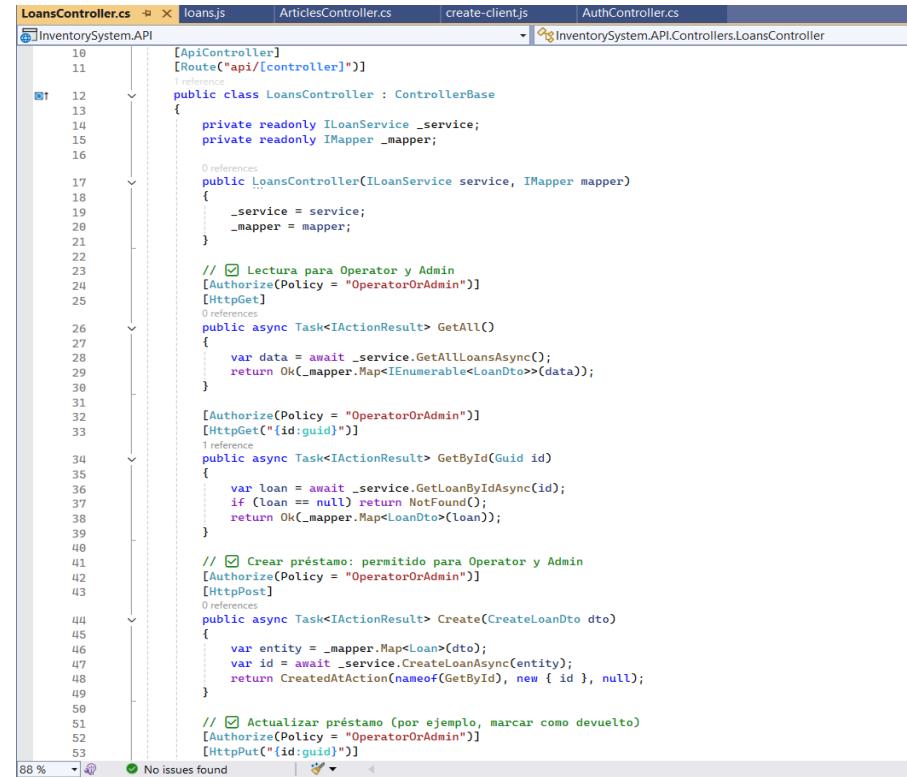
            const filtered = loans.filter(l =>
                (!client || l.clientId === client) &&
                (!article || l.articleId === article) &&
                (!status || l.status === status)
            );

            renderTable(filtered);
        });
    });
}
```

Ilustración 10: Métodos JavaScript para filtros

3. Gestión de Préstamos

- Solicitud y aprobación de préstamos.



```
LoansController.cs  X  loans.js  ArticlesController.cs  create-clientjs  AuthController.cs
InventorySystem.API  InventorySystem.API.Controllers.LoansController
10 [ApiController]
11 [Route("api/[controller]")]
12 public class LoansController : ControllerBase
13 {
14     private readonly ILoanService _service;
15     private readonly IMapper _mapper;
16 
17     public LoansController(ILoanService service, IMapper mapper)
18     {
19         _service = service;
20         _mapper = mapper;
21     }
22 
23     // 🔍 Lectura para Operator y Admin
24     [Authorize(Policy = "OperatorOrAdmin")]
25     [HttpGet]
26     public async Task<IActionResult> GetAll()
27     {
28         var data = await _service.GetAllLoansAsync();
29         return Ok(_mapper.Map<IEnumerable<LoanDto>>(data));
30     }
31 
32     [Authorize(Policy = "OperatorOrAdmin")]
33     [HttpGet("{id:guid}")]
34     public async Task<IActionResult> GetById(Guid id)
35     {
36         var loan = await _service.GetLoanByIdAsync(id);
37         if (loan == null) return NotFound();
38         return Ok(_mapper.Map<LoanDto>(loan));
39     }
40 
41     // 🚧 Crear préstamo: permitido para Operator y Admin
42     [Authorize(Policy = "OperatorOrAdmin")]
43     [HttpPost]
44     public async Task<IActionResult> Create(CreateLoanDto dto)
45     {
46         var entity = _mapper.Map<Loan>(dto);
47         var id = await _service.CreateLoanAsync(entity);
48         return CreatedAtAction(nameof(GetById), new { id }, null);
49     }
50 
51     // 🔍 Actualizar préstamo (por ejemplo, marcar como devuelto)
52     [Authorize(Policy = "OperatorOrAdmin")]
53     [HttpPut("{id:guid}")]
54 }
```

Ilustración 11: Control de préstamos



- o Registro de fechas de entrega y devolución.

The screenshot shows a code editor with multiple tabs: LoansController.cs, loans.js, ArticlesController.cs, create-clients.cs, and AuthController.cs. The loans.js tab is active and displays the following code:

```
// Acciones sobre préstamos
tableBody.addEventListener("click", async (e) => {
  if (e.target.tagName !== "BUTTON" || !e.target.dataset.id) return;

  const id = e.target.dataset.id;
  let update = null;
  let endpoint = "https://localhost:7127/api/loans/${id}";
  let method = "PUT";
  let action = "";

  if (e.target.classList.contains("return-btn")) {
    if (!confirm("Mark this loan as returned?")) return;
    update = {
      returnedAt: new Date().toISOString(),
      status: "Returned"
    };
    action = "marked as returned";
  }

  if (e.target.classList.contains("approve-btn")) {
    if (!confirm("Approve this loan?")) return;
    endpoint += "/status";
    update = "Approved";
    action = "approved";
  }

  if (e.target.classList.contains("reject-btn")) {
    if (!confirm("Reject this loan?")) return;
    endpoint += "/status";
    update = "Rejected";
    action = "rejected";
  }

  if (e.target.classList.contains("delete-btn")) {
    if (!confirm("Delete this loan?")) return;
    try {
      const res = await fetch(`https://localhost:7127/api/loans/${id}`, {
        method: "DELETE",
        headers: { Authorization: `Bearer ${token}` }
      });
      if (!res.ok) throw new Error();
      showToast("Loan deleted successfully!", "success");
      e.target.closest("tr").remove();
    } catch {
      showToast("Failed to delete loan.", "error");
    }
    return;
  }
});
```

Ilustración 12: Métodos JavaScript para validar fechas de devolución

4. Reportes

- o Exportar lista de artículos y préstamos en PDF o Excel vía un botón.

(No aplica)

3. Entregables obligatorios

1. Solución Visual Studio

- o Solution con proyectos:

- MyApp.Presentation (Core Web API)

- Controllers

1. ArticlesController

Permite a operadores y admins gestionar artículos (incluye exportación a PDF desde frontend).



```
namespace InventorySystem.API.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ArticlesController : ControllerBase
    {
        private readonly IArticleService _service;
        private readonly IMapper _mapper;

        public ArticlesController(IArticleService service, IMapper mapper)
        {
            _service = service;
            _mapper = mapper;
        }

        // ☑ Lectura disponible para operadores y admins
        [Authorize(Policy = "OperatorOrAdmin")]
        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            var data = await _service.GetAllArticlesAsync();
            return Ok(_mapper.Map<IEnumerable<ArticleDto>>(data));
        }

        // ☑ Lectura individual también permitida para ambos
        [Authorize(Policy = "OperatorOrAdmin")]
        [HttpGet("{id:guid}")]
        public async Task<IActionResult> GetById(Guid id)
        {
            var article = await _service.GetArticleByIdAsync(id);
            if (article == null) return NotFound();
            return Ok(_mapper.Map<ArticleDto>(article));
        }

        // ☑ Crear artículo: permitido para Operador y Admin
        [Authorize(Policy = "OperatorOrAdmin")]
        [HttpPost]
        public async Task<IActionResult> Create(CreateArticleDto dto)
        {
            var entity = _mapper.Map<Article>(dto);
            var id = await _service.CreateArticleAsync(entity);
            return CreatedAtAction(nameof(GetById), new { id }, null);
        }
    }
}
```

2. AuthController

Maneja la autenticación (/login) y registro de usuarios (/register). Genera JWTs e incluye el employeeId y role en las respuestas.

```
public class AuthController : ControllerBase
{
    private readonly AppDbContext _context;
    private readonly JwtSettings _jwt;

    public AuthController(AppDbContext context, IOptions<JwtSettings> jwtOptions)
    {
        _context = context;
        _jwt = jwtOptions.Value;
    }

    [HttpPost("login")]
    public IActionResult Login([FromBody] LoginDto dto)
    {
        // Simulación de búsqueda de usuario con contraseña hasheada
        var user = _context.Users
            .FirstOrDefault(u => u.Email && u.PasswordHash == Convert.ToBase64String(Encoding.UTF8.GetBytes(dto.Password)));

        if (user == null)
            return Unauthorized(new { message = "Invalid credentials" });

        var employee = _context.Employees.FirstOrDefault(e => e.UserId == user.UserId);
        var roleName = employee != null
            ? _context.Roles.FirstOrDefault(r => r.RoleId == employee.RoleId)?.Name ?? "Unknown"
            : "Operator"; // ☐ default for non-admins

        var token = GenerateToken(user.Email, roleName);

        return Ok(new
        {
            token,
            employeeId = employee?.EmployeeId ?? Guid.Empty, // ☐ nunca null
            role = roleName
        });
    }

    [AllowAnonymous]
    [HttpPost("register")]
    public async Task<IActionResult> Register([FromBody] CreateUserDto dto)
    {
        var emailExists = await _context.Users.AnyAsync(u => u.Email == dto.Email);
        if (emailExists)
            return BadRequest(new { message = "Email already exists." });
    }
}
```



3. ClientsController

Administra clientes (crear, editar, eliminar) según el rol del usuario autenticado.

The screenshot shows a code editor with the ClientsController.cs file open. The code is written in C# and defines a controller for managing clients in an inventory system API. The controller uses dependency injection for IClientService and IMapper. It includes methods for getting all clients, getting a client by ID, and creating a new client. Authorization is handled using [Authorize] attributes and policy-based roles (OperatorOrAdmin). The code editor interface shows tabs for ClientsController.cs, AuthController.cs, and ArticlesController.cs. A status bar at the bottom indicates 88% completion and no issues found.

```
7
8     <namespace InventorySystem.API.Controllers
9     {
10        [ApiController]
11        [Route("api/[controller]")]
12        public class ClientsController : ControllerBase
13        {
14            private readonly IClientService _service;
15            private readonly IMapper _mapper;
16
17            public ClientsController(IClientService service, IMapper mapper)
18            {
19                _service = service;
20                _mapper = mapper;
21            }
22
23            // ☑ Lectura permitida para ambos roles
24            [Authorize(Policy = "OperatorOrAdmin")]
25            [HttpGet]
26            public async Task<IActionResult> GetAll()
27            {
28                var data = await _service.GetAllClientsAsync();
29                return Ok(_mapper.Map<IEnumerable<ClientDto>>(data));
30            }
31
32            // ☑ Lectura individual permitida para ambos
33            [Authorize(Policy = "OperatorOrAdmin")]
34            [HttpGet("{id:guid}")]
35            public async Task<IActionResult> GetById(Guid id)
36            {
37                var client = await _service.GetClientByIdAsync(id);
38                if (client == null) return NotFound();
39                return Ok(_mapper.Map<ClientDto>(client));
40            }
41
42            // ☑ Crear cliente: Admin y Operator
43            [Authorize(Policy = "OperatorOrAdmin")]
44            [HttpPost]
45            public async Task<IActionResult> Create(CreateClientDto dto)
46            {
47                var entity = _mapper.Map<Client>(dto);
48                var id = await _service.CreateClientAsync(entity);
49                return CreatedAtAction(nameof(GetById), new { id }, null);
50            }
51        }
52    }
53
```

4. EmployeesController

Expuesto para admins que gestionan empleados. Permite listar, crear y modificar roles de empleados.



```
EmployeesController.cs  X ClientsController.cs  AuthController.cs  ArticlesController.cs
InventorySystem.API  EmployeesController.cs

7
8  namespace InventorySystem.API.Controllers
9  {
10     [ApiController]
11     [Route("api/[controller]")]
12 
13     public class EmployeesController : ControllerBase
14     {
15         private readonly IEmployeeService _service;
16         private readonly IMapper _mapper;
17 
18         public EmployeesController(IEmployeeService service, IMapper mapper)
19         {
20             _service = service;
21             _mapper = mapper;
22         }
23 
24         // Solo Admin puede listar empleados
25         [Authorize(Policy = "AdminOnly")]
26         [HttpGet]
27 
28         public async Task<IActionResult> GetAll()
29         {
30             var data = await _service.GetAllEmployeesAsync();
31             return Ok(_mapper.Map<IEnumerable<EmployeeDto>>(data));
32         }
33 
34         // Solo Admin puede ver un empleado
35         [Authorize(Policy = "AdminOnly")]
36         [HttpGet("{id:guid}")]
37 
38         public async Task<IActionResult> GetById(Guid id)
39         {
40             var employee = await _service.GetEmployeeByIdAsync(id);
41             if (employee == null) return NotFound();
42             return Ok(_mapper.Map<EmployeeDto>(employee));
43         }
44 
45         // Solo Admin puede crear empleados
46         [Authorize(Policy = "AdminOnly")]
47         [HttpPost]
48 
49         public async Task<IActionResult> Create(CreateEmployeeDto dto)
50         {
51             var entity = _mapper.Map<Employee>(dto);
52             var id = await _service.CreateEmployeeAsync(entity);
53             return CreatedAtAction(nameof(GetById), new { id }, null);
54         }
55     }
56 }

88 %  No issues found
```

5. LoansController

Controla el ciclo de vida de los préstamos: registro, devolución, rechazo o aprobación. Solo admins pueden eliminar préstamos.

```
LoansController.cs  X EmployeesController.cs  ClientsController.cs  AuthController.cs  ArticlesController.cs
InventorySystem.API  LoansController.cs

7
8  namespace InventorySystem.API.Controllers
9  {
10     [ApiController]
11     [Route("api/[controller]")]
12 
13     public class LoansController : ControllerBase
14     {
15         private readonly ILoanService _service;
16         private readonly IMapper _mapper;
17 
18         public LoansController(ILoanService service, IMapper mapper)
19         {
20             _service = service;
21             _mapper = mapper;
22         }
23 
24         // Lectura para Operator y Admin
25         [Authorize(Policy = "OperatorOrAdmin")]
26         [HttpGet]
27 
28         public async Task<IActionResult> GetAll()
29         {
30             var data = await _service.GetAllLoansAsync();
31             return Ok(_mapper.Map<IEnumerable<LoanDto>>(data));
32         }
33 
34         [Authorize(Policy = "OperatorOrAdmin")]
35         [HttpGet("{id:guid}")]
36 
37         public async Task<IActionResult> GetById(Guid id)
38         {
39             var loan = await _service.GetLoanByIdAsync(id);
40             if (loan == null) return NotFound();
41             return Ok(_mapper.Map<LoanDto>(loan));
42         }
43 
44         // Crear préstamo: permitido para Operator y Admin
45         [Authorize(Policy = "OperatorOrAdmin")]
46         [HttpPost]
47 
48         public async Task<IActionResult> Create(CreateLoanDto dto)
49         {
50             var entity = _mapper.Map<Loan>(dto);
51             var id = await _service.CreateLoanAsync(entity);
52             return CreatedAtAction(nameof(GetById), new { id }, null);
53         }
54 }

88 %  No issues found
```



6. ObservationsController

Permite añadir observaciones a un préstamo y listarlas por ID de préstamo.

The screenshot shows a code editor with the file 'ObservationsController.cs' open. The code is written in C# and defines a controller for managing observations in a loan system. It includes imports for AutoMapper, various DTOs, business services, entities, and authentication. The controller itself is annotated with [ApiController] and [Route("api/[controller]")]. It contains two main methods: 'GetByLoan' which retrieves observations for a specific loan ID, and 'Create' which adds a new observation. Both methods use the IMapper interface to map between DTOs and entities, and the IObservationService to interact with the database. Authorization is handled via policy checks for 'OperatorOrAdmin'.

```
ObservationsController.cs  ×  LoansController.cs  EmployeesController.cs  ClientsController.cs  AuthController.cs  Ar
InventorySystem.API  ↴ InventorySystem.API.Controllers.ObservationsController
1  1  using AutoMapper;
2  2  using InventorySystem.API.DTOs;
3  3  using InventorySystem.Business.Services;
4  4  using InventorySystem.Entities.Entities;
5  5  using Microsoft.AspNetCore.Authorization;
6  6  using Microsoft.AspNetCore.Mvc;
7
8  7  namespace InventorySystem.API.Controllers
9  8  {
10 10  [ApiController]
11 11  [Route("api/[controller]")]
12 12  public class ObservationsController : ControllerBase
13 13  {
14 14      private readonly IObservationService _service;
15 15      private readonly IMapper _mapper;
16
17 17      public ObservationsController(IObservationService service, IMapper mapper)
18 18      {
19 19          _service = service;
20 20          _mapper = mapper;
21 21      }
22
23 23  // Ver observaciones: permitido para Operator y Admin
24 24  [Authorize(Policy = "OperatorOrAdmin")]
25 25  [HttpGet("loan/{loanId:guid}")]
26 26  public async Task<IActionResult> GetByLoan(Guid loanId)
27 27  {
28 28      var data = await _service.GetObservationsByLoanAsync(loanId);
29 29      return Ok(_mapper.Map<IEnumerable<ObservationDto>>(data));
30 30  }
31
32 32  // Crear observación: permitido para Operator y Admin
33 33  [Authorize(Policy = "OperatorOrAdmin")]
34 34  [HttpPost]
35 35  public async Task<IActionResult> Create(CreateObservationDto dto)
36 36  {
37 37      var entity = _mapper.Map<Observation>(dto);
38 38      var id = await _service.AddObservationAsync(entity);
39 39      return Created("", new { id });
40 40  }
41 41  }
42 42 }
```

88 % No issues found

7. UsersController

Permite a los admins listar, crear, editar y eliminar usuarios. Depende de UserService y usa AutoMapper para transformar entre DTOs y entidades.



The screenshot shows the code editor for the UsersController.cs file within the InventorySystem.API project. The code implements a RESTful API for managing users. It includes methods for getting all users, getting a user by ID, and creating a new user. The 'AdminOnly' attribute is used to restrict access to certain endpoints.

```
1  [ApiController]
2  [Route("api/[controller]")]
3  1 reference
4  public class UsersController : ControllerBase
5  {
6      private readonly IUserService _service;
7      private readonly IMapper _mapper;
8
9      0 references
10     public UsersController(IUserService service, IMapper mapper)
11     {
12         _service = service;
13         _mapper = mapper;
14     }
15
16     // ✅ Solo Admin puede ver todos los usuarios
17     [Authorize(Policy = "AdminOnly")]
18     [HttpGet]
19     0 references
20     public async Task<IActionResult> GetAll()
21     {
22         var users = await _service.GetAllUsersAsync();
23         var dtos = _mapper.Map<IEnumerable<UserDTO>>(users);
24         return Ok(dtos);
25     }
26
27     // ✅ Solo Admin puede ver un usuario
28     [Authorize(Policy = "AdminOnly")]
29     [HttpGet("{id:guid}")]
30     1 reference
31     public async Task<IActionResult> GetById(Guid id)
32     {
33         var user = await _service.GetUserByIdAsync(id);
34         if (user == null) return NotFound();
35         return Ok(_mapper.Map<UserDTO>(user));
36     }
37
38     // ✅ Solo Admin puede crear usuarios
39     [Authorize(Policy = "AdminOnly")]
40     [HttpPost]
41     0 references
42     public async Task<IActionResult> Create([FromBody] CreateUserDto dto)
43     {
44         // Validar email duplicado
45         try
46         {
47             var hashed = HashPassword(dto.Password);
48             var id = await _service.CreateUserWithRoleAsync(dto.FirstName, dto.LastName, dto.Email, hashed, dto.Role);
49         }
50     }
51
52     0 issues found
```

- DTOs
 - 1. Article

Permiten manejar la creación y modificación de artículos

```
UpdateArticleDto.cs ArticleDto.cs CreateArticleDto.cs UpdateArticleDto.cs ArticleDto.cs CreateArticleDto.cs
InventorySystem.API
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para crear un articulo.
    /// </summary>
    2 references
    public class CreateArticleDto
    {
        0 references
        public string Code { get; set; } = null!;
        0 references
        public string Name { get; set; } = null!;
        0 references
        public string Category { get; set; } = null!;
        0 references
        public string Status { get; set; } = null!;
        0 references
        public string Location { get; set; } = null!;
    }
}

InventorySystem.API
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para representar un articulo.
    /// </summary>
    3 references
    public class ArticleDto
    {
        0 references
        public Guid ArticleId { get; set; }
        0 references
        public string Code { get; set; } = null!;
        0 references
        public string Name { get; set; } = null!;
        0 references
        public string Category { get; set; } = null!;
        0 references
        public string Status { get; set; } = null!;
        0 references
        public string Location { get; set; } = null!;
    }
}

UpdateArticleDto.cs ArticleDto.cs CreateArticleDto.cs
InventorySystem.API
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para actualizar un articulo existente.
    /// </summary>
    2 references
    public class UpdateArticleDto
    {
        0 references
        public string Code { get; set; } = null!;
        0 references
        public string Name { get; set; } = null!;
        0 references
        public string Category { get; set; } = null!;
        0 references
        public string Status { get; set; } = null!;
        0 references
        public string Location { get; set; } = null!;
    }
}
```

2. Client

DTOs para gestionar clientes desde el frontend



```
InventorySystem.API
ClientDto.cs
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para representar un cliente.
    /// </summary>
    public class ClientDto
    {
        public Guid ClientId { get; set; }
        public string FirstName { get; set; } = null!;
        public string LastName { get; set; } = null!;
        public string Email { get; set; } = null!;
        public string Phone { get; set; } = null!;
    }
}

InventorySystem.API
CreateClientDto.cs
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para crear un cliente.
    /// </summary>
    public class CreateClientDto
    {
        public string FirstName { get; set; } = null!;
        public string LastName { get; set; } = null!;
        public string Email { get; set; } = null!;
        public string Phone { get; set; } = null!;
    }
}

InventorySystem.API
UpdateClientDto.cs
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para actualizar datos de un cliente.
    /// </summary>
    public class UpdateClientDto
    {
        public string FirstName { get; set; } = null!;
        public string LastName { get; set; } = null!;
        public string Email { get; set; } = null!;
        public string Phone { get; set; } = null!;
    }
}

InventorySystem.API
ClientDto.cs
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para representar un cliente.
    /// </summary>
    public class ClientDto
    {
        public Guid ClientId { get; set; }
        public string FirstName { get; set; } = null!;
        public string LastName { get; set; } = null!;
        public string Email { get; set; } = null!;
        public string Phone { get; set; } = null!;
    }
}
```

3. Loan (Préstamo)

Estructuras de datos para crear préstamos y representar su estado.

```
InventorySystem.API
LoanDto.cs
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para representar un préstamo.
    /// </summary>
    public class LoanDto
    {
        public Guid LoanId { get; set; }
        public Guid ArticleId { get; set; }
        public Guid EmployeeId { get; set; }
        public Guid ClientId { get; set; }
        public DateTime RequestedAt { get; set; }
        public DateTime DeliveredAt { get; set; }
        public DateTime? ReturnedAt { get; set; }
        public string Status { get; set; } = null!;
    }
}

InventorySystem.API
CreateLoanDto.cs
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para crear un préstamo.
    /// </summary>
    public class CreateLoanDto
    {
        public Guid ArticleId { get; set; }
        public Guid EmployeeId { get; set; }
        public Guid ClientId { get; set; }
        public DateTime DeliveredAt { get; set; }
        public string Status { get; set; } = null!; // Estado inicial (e.g., \"Pending\")
    }
}

InventorySystem.API
UpdateLoanDto.cs
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para actualizar un préstamo existente.
    /// </summary>
    public class UpdateLoanDto
    {
        public DateTime? ReturnedAt { get; set; } // Fecha de devolución
        public string Status { get; set; } = null!; // Nuevo estado
    }
}
```

4. Empleado

DTOs para manejar empleados, asignar roles y contratar.



```
EmployeeDto.cs
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para representar un empleado.
    /// </summary>
    public class EmployeeDto
    {
        public Guid EmployeeId { get; set; } // Identificador de empleado
        public Guid UserId { get; set; } // FK al usuario
        public int RoleId { get; set; } // FK al rol
        public DateTime HireDate { get; set; } // Fecha de ingreso
    }
}

CreateEmployeeDto.cs
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para crear un empleado.
    /// </summary>
    public class CreateEmployeeDto
    {
        public Guid UserId { get; set; }
        public int RoleId { get; set; }
        public DateTime HireDate { get; set; } // Puede omitirse y asignarse automáticamente en el servicio
    }
}

UpdateEmployeeDto.cs
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para actualizar un empleado existente.
    /// </summary>
    public class UpdateEmployeeDto
    {
        public int RoleId { get; set; }
        public DateTime HireDate { get; set; }
    }
}
```

5. User

Se usan para enviar y recibir información de usuario sin exponer datos sensibles como contraseñas hasheadas.



```
InventorySystem.API.DTOS
UpdateUserDto.cs
CreateUserDto.cs
UserDTO.cs
UpdateUserDto.cs
InventorySystem.API.DTOS
CreateUserDto.cs
UserDTO.cs
InventorySystem.API.DTOS
UpdateUserDto.cs
CreateUserDto.cs
UserDTO.cs
```

```
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para representar un usuario en peticiones/respuestas.
    /// </summary>
    public class UserDTO
    {
        public Guid UserId { get; set; }
        public string FirstName { get; set; } = null!;
        public string LastName { get; set; } = null!;
        public string Email { get; set; } = null!;
        public DateTime CreatedAt { get; set; }
        public string? Role { get; set; } // △ Esto debe estar presente
    }
}

namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para crear un nuevo usuario.
    /// </summary>
    public class CreateUserDto
    {
        public string FirstName { get; set; } = null!;
        public string LastName { get; set; } = null!;
        public string Email { get; set; } = null!;
        public string Password { get; set; } = null!; // Plain text, se convertirá a hash
        public string Role { get; set; } = null!;
    }
}

namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para actualizar un usuario existente.
    /// </summary>
    public class UpdateUserDto
    {
        public string FirstName { get; set; } = null!;
        public string LastName { get; set; } = null!;
        public string Email { get; set; } = null!;
        public string Role { get; set; } = null!;
    }
}
```

6. Observaciones

Representan observaciones añadidas por operadores o admins.

```
InventorySystem.API
CreateObservationDto.cs
ObservationDto.cs
InventorySystem.API
CreateObservationDto.cs
ObservationDto.cs
```

```
InventorySystem.API.DTOs
CreateObservationDto.cs
ObservationDto.cs
InventorySystem.API.DTOs
CreateObservationDto.cs
ObservationDto.cs
```

```
namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para representar una observación.
    /// </summary>
    public class ObservationDto
    {
        public Guid ObservationId { get; set; }
        public Guid LoanId { get; set; }
        public string Text { get; set; } = null!;
        public DateTime CreatedAt { get; set; }
    }
}

namespace InventorySystem.API.DTOs
{
    /// <summary>
    /// DTO para crear una nueva observación.
    /// </summary>
    public class CreateObservationDto
    {
        public Guid LoanId { get; set; }
        public string Text { get; set; } = null!;
    }
}
```

- Security

1. JwtSettings

Configuración para la emisión de JWTs: clave secreta, duración, emisor, audiencia.

```
InventorySystem.API
JwtSettings.cs
InventorySystem.API
JwtSettings.cs
```

```
InventorySystem.API.Security
JwtSettings.cs
InventorySystem.API.Security
JwtSettings.cs
```

```
namespace InventorySystem.API.Security
{
    public class JwtSettings
    {
        public string Key { get; set; } = null!;
        public string Issuer { get; set; } = null!;
        public string Audience { get; set; } = null!;
        public int DurationInMinutes { get; set; }
    }
}
```



- Program.cs

Es el punto de entrada del backend ASP.NET Core. Aquí se configura el pipeline de la aplicación (servicios, middleware, rutas, autenticación, etc.). Porque define cómo se comporta la app al iniciar. Registra los servicios (como AutoMapper, DbContext, UnitOfWork, controladores), habilita CORS, Swagger, JWT, etc.

```
Program.cs  ✘ ×
InventorySystem.API
13 // -----
14 // 1. EF Core - DbContext
15 // -----
16 builder.Services.AddDbContext<AppDbContext>(options =>
17     options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
18
19 // -----
20 // 2. JWT Authentication
21 // -----
22 builder.Services.Configure<JwtSettings>(builder.Configuration.GetSection("JwtSettings"));
23 var jwt = builder.Configuration.GetSection("JwtSettings").Get<JwtSettings>();
24 var key = Encoding.UTF8.GetBytes(jwt.Key);
25
26 builder.Services.AddAuthentication(options =>
27 {
28     options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
29     options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
30 })
31     .AddJwtBearer(options =>
32 {
33     options.TokenValidationParameters = new TokenValidationParameters
34     {
35         ValidateIssuer = true,
36         ValidateAudience = true,
37         ValidateLifetime = true,
38         ValidateIssuerSigningKey = true,
39         ValidIssuer = jwt.Issuer,
40         ValidAudience = jwt.Audience,
41         IssuerSigningKey = new SymmetricSecurityKey(key)
42     };
43 });
44
45 // -----
46 // 3. Authorization Policies
47 // -----
48 builder.Services.AddAuthorization(options =>
49 {
50     options.AddPolicy("AdminOnly", policy => policy.RequireRole("Admin"));
51     options.AddPolicy("OperatorOrAdmin", policy => policy.RequireRole("Admin", "Operator"));
52 });
53
54
55 // -----
56 // 4. AutoMapper
57 // -----
58 builder.Services.AddAutoMapper(typeof(MappingProfile));
59 builder.Services.AddAutoMapper(typeof(Program));
60
```

- Appsetting.json

Archivo de configuración principal del backend. Contiene ajustes que pueden cambiar por entorno (dev, prod), como cadenas de conexión, configuración del JWT, y más. Permite separar la lógica de configuración del código fuente. Aporta flexibilidad y seguridad (por ejemplo, al definir claves y duración del token JWT).

```
appsettings.json*  ✘ ×
Schema: https://www.schemastore.org/appsettings.json
1 {
2     "ConnectionStrings": {
3         "DefaultConnection": "Server=DESKTOP-622UG1M\SQLDEV;Database=InventoryDb;Trusted_Connection=True;MultipleActiveResultSets=true"
4     },
5     "JwtSettings": {
6         "Key": "SuperSecureJwtKey1234567890Token!!", // ✋ 32 caracteres = 256 bits
7         "Issuer": "InventorySystem",
8         "Audience": "InventorySystemUser",
9         "DurationInMinutes": 60
10    },
11    "Logging": {
12        "LogLevel": {
13            "Default": "Information",
14            "Microsoft.AspNetCore": "Warning"
15        }
16    },
17    "AllowedHosts": "*"
18 }
/*Key: "supersecretkey1234567890",*/
```

- MappingProfile

Sirve para definir todas las reglas de mapeo entre entidades del dominio (User, Client, etc.) y sus respectivos DTOs (UserDto, CreateUserDto, etc.). Permite convertir fácilmente un Client en un ClientDto y viceversa, sin tener que escribir a mano cada asignación de propiedad.

```

MappingProfile.cs  ↗ X
InventorySystem.API
1  using InventorySystem.API.DTOs;
2  using InventorySystem.Entities.Entities;
3  using AutoMapper;
4
5  namespace InventorySystem.API
6  {
7      /// <summary>
8      /// Perfil de AutoMapper para mapear entre entidades y DTOs.
9      /// </summary>
10     public class MappingProfile : Profile
11     {
12         public MappingProfile()
13         {
14             // Mapeos básicos
15             CreateMap<Article, ArticleDto>().ReverseMap();
16             CreateMap<Client, ClientDto>().ReverseMap();
17             CreateMap<Loan, LoanDto>().ReverseMap();
18             CreateMap<Employee, EmployeeDto>().ReverseMap();
19             CreateMap<Observation, ObservationDto>().ReverseMap();
20             CreateMap<User, UserDTO>();
21
22             CreateMap<UpdateUserDto, User>();
23             CreateMap<CreateUserDto, User>();
24             CreateMap<User, UserDTO>();
25
26             CreateMap<UpdateClientDto, Client>();
27             CreateMap<UpdateArticleDto, Article>();
28             //CreateMap<Role, RoleDto>().ReverseMap();
29
30             // Otros mapeos que uses
31             CreateMap<CreateArticleDto, Article>();
32             CreateMap<CreateClientDto, Client>();
33             CreateMap<CreateLoanDto, Loan>();
34             CreateMap<CreateUserDto, User>();
35
36         }
37     }
38

```

- Front-End (wwwroot)

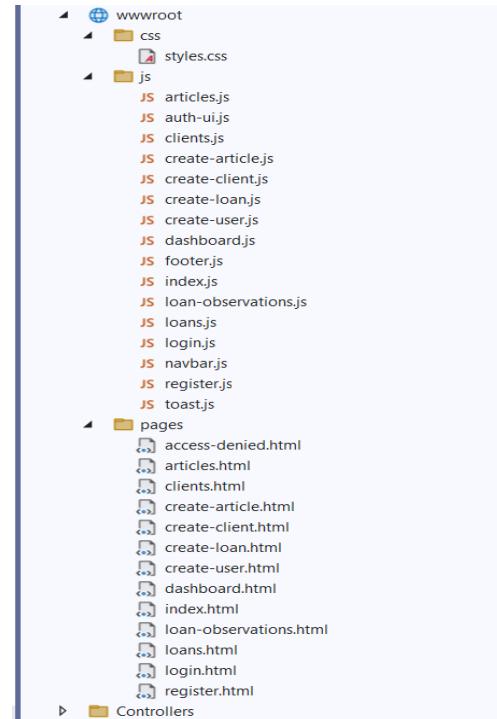


Ilustración 13: Estructura del Front-End



■ MyApp.Business

- Services

1. IArticleService

```
IArticleService.cs  X  IObservationService.cs
InventorySystem.Business  InventorySystem.Business.Services.IA  UpdateArticleAsync(Article)
1  using InventorySystem.Entities.Entities;
2
3  namespace InventorySystem.Business.Services
4  {
5      /// <summary>
6      /// Interface para la lógica de artículos.
7      /// </summary>
8
9
10     public interface IArticleService
11     {
12         Task<IEnumerable<Article>> GetAllArticlesAsync();
13         Task<Article?> GetArticleByIdAsync(Guid id);
14         Task<Guid> CreateArticleAsync(Article article);
15         Task UpdateArticleAsync(Article article);
16         Task DeleteArticleAsync(Guid id);
17     }
}
```

2. ArticleService

Administra artículos y se asegura de aplicar las reglas necesarias antes de su almacenamiento.



```
public class ArticleService : IArticleService
{
    private readonly IUnitOfWork _uow;

    public ArticleService(IUnitOfWork uow)
    {
        _uow = uow;
    }

    public async Task<IEnumerable<Article>> GetAllArticlesAsync()
    {
        return await _uow.Articles.GetAllAsync();
    }

    public async Task<Article?> GetArticleByIdAsync(Guid id)
    {
        return await _uow.Articles.GetByIdAsync(id);
    }

    public async Task<Guid> CreateArticleAsync(Article article)
    {
        var all = await _uow.Articles.GetAllAsync();
        if (all.Any(a => a.Code == article.Code))
            throw new InvalidOperationException("Article code must be unique.");

        await _uow.Articles.AddAsync(article);
        await _uow.CompleteAsync();
        return article.ArticleId;
    }

    public async Task UpdateArticleAsync(Article article)
    {
        _uow.Articles.Update(article);
        await _uow.CompleteAsync();
    }

    public async Task DeleteArticleAsync(Guid id)
    {
        var existing = await _uow.Articles.GetByIdAsync(id);
        if (existing != null)
        {
            _uow.Articles.Remove(existing);
            await _uow.CompleteAsync();
        }
    }
}
```

3. IClientService

```
using InventorySystem.Entities.Entities;
namespace InventorySystem.Business.Services
{
    /// <summary>
    /// Interface para la lógica de clientes.
    /// </summary>
    public interface IClientService
    {
        Task<IEnumerable<Client>> GetAllClientsAsync();
        Task<Client?> GetClientByIdAsync(Guid id);
        Task<Guid> CreateClientAsync(Client client);
        Task UpdateClientAsync(Client client);
        Task DeleteClientAsync(Guid id);
    }
}
```



4. ClientService

Administra las operaciones CRUD de clientes. Encapsula validaciones antes de llamar al repositorio.

```
ClientService.cs  ✘  ✘  IClientService.cs
InventorySystem.Business  InventorySystem.Business.Services.ClientService
public class ClientService : IClientService
{
    private readonly IUnitOfWork _uow;

    public ClientService(IUnitOfWork uow)
    {
        _uow = uow;
    }

    public async Task<IEnumerable<Client>> GetAllClientsAsync()
    {
        return await _uow.Clients.GetAllAsync();
    }

    public async Task<Client?> GetClientByIdAsync(Guid id)
    {
        return await _uow.Clients.GetByIdAsync(id);
    }

    public async Task<Guid> CreateClientAsync(Client client)
    {
        var all = await _uow.Clients.GetAllAsync();
        if (all.Any(c => c.Email == client.Email))
            throw new InvalidOperationException("Client email already exists.");

        await _uow.Clients.AddAsync(client);
        await _uow.CompleteAsync();
        return client.ClientId;
    }

    public async Task UpdateClientAsync(Client client)
    {
        _uow.Clients.Update(client);
        await _uow.CompleteAsync();
    }

    public async Task DeleteClientAsync(Guid id)
    {
        var existing = await _uow.Clients.GetByIdAsync(id);
        if (existing != null)
        {
            _uow.Clients.Remove(existing);
            await _uow.CompleteAsync();
        }
    }
}
```

5. IEmployeeService



```
IEmployeeService.cs
InventorySystem.Business    InventorySystem.Business.Services.IE    UpdateEmployee
1     using InventorySystem.Entities.Entities;
2
3     namespace InventorySystem.Business.Services
4     {
5         /// <summary>
6         /// Interface para manejar lógica de empleados.
7         /// </summary>
8         public interface IEmployeeService
9         {
10            Task<IEnumerable<Employee>> GetAllEmployeesAsync();
11            Task<Employee?> GetEmployeeByIdAsync(Guid id);
12            Task<Guid> CreateEmployeeAsync(Employee employee);
13            Task UpdateEmployeeAsync(Employee employee);
14            Task DeleteEmployeeAsync(Guid id);
15        }
16    }
```

6. EmployeeService

Lógica para manejar empleados: alta, edición, baja. Crea empleados asociados a usuarios existentes y les asigna roles.

```
EmployeeService.cs
InventorySystem.Business    InventorySystem.Business.Services.EmployeeService
1     public class EmployeeService : IEmployeeService
2     {
3         private readonly IUnitOfWork _unitOfWork;
4
5         public EmployeeService(IUnitOfWork unitOfWork)
6         {
7             _unitOfWork = unitOfWork;
8         }
9
10        public async Task<IEnumerable<Employee>> GetAllEmployeesAsync()
11        {
12            return await _unitOfWork.Employees.GetAllAsync();
13        }
14
15        public async Task<Employee?> GetEmployeeByIdAsync(Guid id)
16        {
17            return await _unitOfWork.Employees.GetByIdAsync(id);
18        }
19
20        public async Task<Guid> CreateEmployeeAsync(Employee employee)
21        {
22            employee.HireDate = DateTime.UtcNow;
23            await _unitOfWork.Employees.AddAsync(employee);
24            await _unitOfWork.CompleteAsync();
25            return employee.EmployeeId;
26        }
27
28        public async Task UpdateEmployeeAsync(Employee employee)
29        {
30            _unitOfWork.Employees.Update(employee);
31            await _unitOfWork.CompleteAsync();
32        }
33
34        public async Task DeleteEmployeeAsync(Guid id)
35        {
36            var existing = await _unitOfWork.Employees.GetByIdAsync(id);
37            if (existing != null)
38            {
39                _unitOfWork.Employees.Remove(existing);
40                await _unitOfWork.CompleteAsync();
41            }
42        }
43    }
44
45    } No issues found
```



7. ILoanService

```
ILoanService.cs  X
InventorySystem.Business  InventorySystem.Business.Services.IL  UpdateLoanSta
1  using InventorySystem.Entities.Entities;
2
3  namespace InventorySystem.Business.Services
4  {
5      /// <summary>
6      /// Interface para la lógica de préstamos.
7      /// </summary>
8
9      public interface ILoanService
10     {
11         Task<IList<Loan>> GetAllLoansAsync();
12         Task<Loan?> GetLoanByIdAsync(Guid id);
13         Task<Guid> CreateLoanAsync(Loan loan);
14         Task UpdateLoanAsync(Loan loan);
15         Task DeleteLoanAsync(Guid id);
16         Task UpdateLoanStatusAsync(Guid id, string newStatus);
17     }
18 }
```

8. LoanService

Implementa reglas para crear y modificar préstamos. Se asegura de registrar fechas, verificar disponibilidad, y validar estados (ej. no se puede marcar como "Returned" dos veces).

```
LoanService.cs  X
InventorySystem.Business  InventorySystem.Business.Services.LoanService
9
10    public class LoanService : ILoanService
11    {
12        private readonly IUnitOfWork _uow;
13
14        public LoanService(IUnitOfWork uow)
15        {
16            _uow = uow;
17        }
18
19        public async Task<IList<Loan>> GetAllLoansAsync()
20        {
21            return await _uow.Loops.GetAllAsync();
22        }
23
24        public async Task<Loan?> GetLoanByIdAsync(Guid id)
25        {
26            return await _uow.Loops.GetByIdAsync(id);
27        }
28
29        public async Task<Guid> CreateLoanAsync(Loan loan)
30        {
31            loan.RequestedAt = DateTime.UtcNow;
32            // Aquí podríamos hacer validaciones (disponibilidad del artículo, etc.)
33            await _uow.Loops.AddAsync(loan);
34            await _uow.CompleteAsync();
35            return loan.LoanId;
36        }
37
38        public async Task UpdateLoanAsync(Loan loan)
39        {
40            _uow.Loops.Update(loan);
41            await _uow.CompleteAsync();
42        }
43
44        public async Task DeleteLoanAsync(Guid id)
45        {
46            var existing = await _uow.Loops.GetByIdAsync(id);
47            if (existing != null)
48            {
49                _uow.Loops.Remove(existing);
50                await _uow.CompleteAsync();
51            }
52        }
53    }
54 }
```

9. IUserService



```
IUserService.cs
InventorySystem.Business
InventorySystem.Business.Services.IUserService
Up

1  using InventorySystem.Entities.Entities;
2
3  namespace InventorySystem.Business.Services
4  {
5      /// <summary>
6      /// Interface que define operaciones de negocio sobre usuarios.
7      /// </summary>
8
9      public interface IUserService
10     {
11         Task<IList<User>> GetAllUsersAsync();
12         Task<User?> GetUserByIdAsync(Guid id);
13         Task<Guid> CreateUserAsync(User user); // Opcional si lo usás
14         Task<Guid> CreateUserWithRoleAsync(string firstName, string lastName, string email, string passwordHash, string roleName);
15         Task UpdateUserAsync(Guid userId, string firstName, string lastName, string email, string roleName);
16         Task DeleteUserAsync(Guid id);
17     }
18 }
19
20 
```

10. UserService

Encapsula la lógica para crear, actualizar y eliminar usuarios. Asegura que los emails sean únicos y aplica reglas antes de persistir datos.

```
UserService.cs
InventorySystem.Business
InventorySystem.Business.Services.UserService
Up

10  public class UserService : IUserService
11  {
12      private readonly IUnitOfWork _unitOfWork;
13
14      public UserService(IUnitOfWork unitOfWork)
15      {
16          _unitOfWork = unitOfWork;
17      }
18
19      public async Task<IList<User>> GetAllUsersAsync()
20      {
21          // Aquí podríamos añadir paginación, filtros, etc.
22          return await _unitOfWork.Users.GetAllAsync();
23      }
24
25
26      public async Task<User?> GetUserByIdAsync(Guid id)
27      {
28          return await _unitOfWork.Users.GetByIdAsync(id);
29      }
30
31      public async Task<Guid> CreateUserAsync(User user)
32      {
33          // Validar datos, e.g. email único
34          var allUsers = await _unitOfWork.Users.GetAllAsync();
35
36          if (allUsers.Any(u => u.Email == user.Email))
37              throw new InvalidOperationException("Email already exists.");
38
39          user.CreatedAt = DateTime.UtcNow;
40
41          await _unitOfWork.Users.AddAsync(user);
42          await _unitOfWork.CompleteAsync();
43          return user.UserId;
44      }
45
46      public async Task UpdateUserAsync(Guid userId, string firstName, string lastName, string email, string roleName)
47      {
48          var user = await _unitOfWork.Users.GetByIdAsync(userId);
49          if (user == null)
50              throw new InvalidOperationException("User not found.");
51
52          user.FirstName = firstName;
53      }
54
55
56
57
58
59
59 
```

11. IObservationService



The screenshot shows a code editor with two tabs: `IObservationService.cs` and `UserService.cs`. The `IObservationService.cs` tab is active, displaying the following C# code:

```
1     using InventorySystem.Entities.Entities;
2
3     namespace InventorySystem.Business.Services
4     {
5         /// <summary>
6         /// Interface para la lógica de observaciones.
7         /// </summary>
8         public interface IObservationService
9         {
10             Task<IList<Observation>> GetObservationsByLoanAsync(Guid loanId);
11             Task<Guid> AddObservationAsync(Observation observation);
12         }
13     }
```

12. ObservationService

Lógica de negocio para agregar y listar observaciones asociadas a préstamos.

The screenshot shows a code editor with three tabs: `ObservationService.cs`, `IObservationService.cs`, and `UserService.cs`. The `ObservationService.cs` tab is active, displaying the following C# code:

```
1     using InventorySystem.DataAccess;
2     using InventorySystem.Entities.Entities;
3
4     namespace InventorySystem.Business.Services
5     {
6         /// <summary>
7         /// Implementación de reglas de negocio para observaciones.
8         /// </summary>
9         public class ObservationService : IObservationService
10        {
11            private readonly IUnitOfWork _unitOfWork;
12
13            public ObservationService(IUnitOfWork unitOfWork)
14            {
15                _unitOfWork = unitOfWork;
16            }
17
18            public async Task<IList<Observation>> GetObservationsByLoanAsync(Guid loanId)
19            {
20                // Podríamos filtrar directamente en la consulta
21                var all = await _unitOfWork.Observations.GetAllAsync();
22                return all.Where(o => o.LoanId == loanId);
23            }
24
25            public async Task<Guid> AddObservationAsync(Observation observation)
26            {
27                observation.CreatedAt = DateTime.UtcNow;
28                await _unitOfWork.Observations.AddAsync(observation);
29                await _unitOfWork.CompleteAsync();
30                return observation.ObservationId;
31            }
32        }
33    }
```

- MyApp.DataAccess

- Repositories
- 1. IRepository

Repositorio genérico que define operaciones comunes: GetAllAsync, GetByIdAsync, AddAsync, Update, Remove.



```
namespace InventorySystem.DataAccess.Repositories
{
    public interface IRepository<T> where T : class
    {
        Task<IEnumerable<T>> GetAllAsync();
        Task<T?> GetByIdAsync(Guid id);
        Task AddAsync(T entity);
        void Update(T entity);
        Task Remove(T entity);
    }
}
```

2. Repository

Repositorio genérico que define operaciones comunes: GetAllAsync, GetByIdAsync, AddAsync, Update, Remove.

```
public class Repository<T> : IRepository<T> where T : class
{
    protected readonly AppDbContext _context;
    protected readonly DbSet<T> _dbSet;

    public Repository(AppDbContext context)
    {
        _context = context;
        _dbSet = context.Set<T>();
    }

    public async Task<IEnumerable<T>> GetAllAsync()
    {
        // Devuelve todos los registros
        return await _dbSet.ToListAsync();
    }

    public async Task<T?> GetByIdAsync(Guid id)
    {
        // Busca por clave primaria
        return await _dbSet.FindAsync(id);
    }

    public async Task AddAsync(T entity)
    {
        // Añade la entidad al Change Tracker
        await _dbSet.AddAsync(entity);
    }

    public void Update(T entity)
    {
        // Marca la entidad como modificada
        _dbSet.Update(entity);
    }

    public void Remove(T entity)
    {
        // Marca la entidad como eliminada
        _dbSet.Remove(entity);
    }
}
```

- ApplicationDbContext

Contexto EF Core. Configura las relaciones entre entidades y mapea las clases a tablas.



```
AppDbContext.cs
public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options)
    {
    }

    // DbSets para cada entidad
    public DbSet<User> Users { get; set; } = null!;
    public DbSet<Role> Roles { get; set; } = null!;
    public DbSet<Employee> Employees { get; set; } = null!;
    public DbSet<Client> Clients { get; set; } = null!;
    public DbSet<Article> Articles { get; set; } = null!;
    public DbSet<Loan> Loans { get; set; } = null!;
    public DbSet<Observation> Observations { get; set; } = null!;

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        // Configuración de la relación uno a uno User-Employee
        modelBuilder.Entity<User>()
            .HasOne(u => u.Employee)
            .WithOne(e => e.User)
            .HasForeignKey<Employee>(e => e.UserId);

        // Configuración de la relación uno a muchos Role-Employee
        modelBuilder.Entity<Role>()
            .HasMany(r => r.Employees)
            .WithOne(e => e.Role)
            .HasForeignKey(e => e.RoleId);

        // Configuración de la relación uno a muchos Employee-Loan
        modelBuilder.Entity<Employee>()
            .HasMany(e => e.Loans)
            .WithOne(l => l.Employee)
            .HasForeignKey(l => l.EmployeeId);

        // Configuración de la relación uno a muchos Client-Loan
        modelBuilder.Entity<Client>()
            .HasMany(c => c.Loans);
    }
}
```

- IUnitOfWork

Coordina múltiples repositorios y permite ejecutar SaveChangesAsync() de forma transaccional.
Evita que cada servicio maneje directamente el DbContext.

```
IUnitOfWork.cs
using InventorySystem.DataAccess.Repositories;
using InventorySystem.Entities.Entities;

namespace InventorySystem.DataAccess
{
    /// <summary>
    /// Interface para coordinar múltiples repositorios y transacciones.
    /// </summary>
    public interface IUnitOfWork : IDisposable
    {
        IRepository<User> Users { get; }

        IRepository<Role> Roles { get; }

        IRepository<Employee> Employees { get; }

        IRepository<Client> Clients { get; }

        IRepository<Article> Articles { get; }

        IRepository<Loan> Loans { get; }

        IRepository<Observation> Observations { get; }

        /// <summary>
        /// Persiste todos los cambios pendientes en la base de datos.
        /// </summary>
        Task<int> CompleteAsync();
    }
}
```

- UnitOfWork

Coordina múltiples repositorios y permite ejecutar SaveChangesAsync() de forma transaccional.
Evita que cada servicio maneje directamente el DbContext.



The screenshot shows two code files side-by-side in a code editor:

- IUnitOfWork.cs**: A partial interface definition for a unit of work. It defines methods for getting repositories for User, Role, Employee, Client, Article, Loan, and Observation.
- UnitOfWork.cs**: A concrete implementation of **IUnitOfWork**. It contains a private field `_context` of type `AppDbContext`. It has a constructor that takes `AppDbContext` and initializes `_context`. It also implements the methods defined in **IUnitOfWork**, such as `CompleteAsync()` which returns the number of affected rows from `SaveChangesAsync()`, and `Dispose()` which disposes the `DbContext`.

- MyApp.Entities

- Entities
 - 1. Article

Representa un artículo disponible para préstamo. Se relaciona con Loan y contiene información como código, nombre, estado y ubicación.

The screenshot shows the **Article.cs** file:

```
namespace InventorySystem.Entities.Entities
{
    public class Article
    {
        public Guid ArticleId { get; set; } // Identificador único
        public string Code { get; set; } = null!; // Código interno
        public string Name { get; set; } = null!; // Descripción o nombre
        public string Category { get; set; } = null!; // Categoría del artículo
        public string Status { get; set; } = null!; // Estado (e.g., Available, Loaned)
        public string Location { get; set; } = null!; // Ubicación física

        // Préstamos asociados a este artículo
        public ICollection<Loan> Loans { get; set; } = new List<Loan>();
    }
}
```

- 2. Client

Representa a un cliente externo al sistema. Tiene información de contacto y una relación uno a muchos con Loan.



```
Client.cs  Article.cs
InventorySystem.Entities
namespace InventorySystem.Entities.Entities
{
    public class Client
    {
        public Guid ClientId { get; set; } // Identificador único de cliente
        public string FirstName { get; set; } = null!; // Nombre de pila
        public string LastName { get; set; } = null!; // Apellido
        public string Email { get; set; } = null!; // Correo electrónico
        public string Phone { get; set; } = null!; // Teléfono de contacto
        public ICollection<Loan> Loans { get; set; } = new List<Loan>();
    }
}
```

3. Empleado

Representa a un empleado. Tiene una relación uno a uno con User y muchos a uno con Role. Además, está relacionado con Loan, ya que un empleado gestiona préstamos.

```
Employee.cs  Client.cs  Article.cs
InventorySystem.Entities
namespace InventorySystem.Entities.Entities
{
    public class Employee
    {
        public Guid EmployeeId { get; set; } // Identificador único de empleado
        public Guid UserId { get; set; } // FK hacia User
        public int RoleId { get; set; } // FK hacia Role
        public DateTime HireDate { get; set; } // Fecha de ingreso
        public User User { get; set; } = null!; // Usuario asociado
        public Role Role { get; set; } = null!; // Rol asignado
        public ICollection<Loan> Loans { get; set; } = new List<Loan>(); // Préstamos realizados
    }
}
```

4. Préstamo

Modelo de un préstamo de un artículo a un cliente. Se relaciona con Article, Employee, Client y Observation.

```
Loan.cs  Employee.cs  Client.cs  Article.cs
InventorySystem.Entities
namespace InventorySystem.Entities.Entities
{
    public class Loan
    {
        public Guid LoanId { get; set; } // Identificador único de préstamo
        public Guid ArticleId { get; set; } // FK hacia Article
        public Guid EmployeeId { get; set; } // FK hacia Employee
        public Guid ClientId { get; set; } // FK hacia Client
        public DateTime RequestedAt { get; set; } // Fecha de solicitud
        public DateTime DeliveredAt { get; set; } // Fecha de entrega
        public DateTime? ReturnedAt { get; set; } // Fecha de devolución (opcional)
        public string Status { get; set; } = null!; // Estado (e.g., Pending, Completed)
        public Article Article { get; set; } = null!;
        public Employee Employee { get; set; } = null!;
        public Client Client { get; set; } = null!;
        public ICollection<Observation> Observations { get; set; } = new List<Observation>();
    }
}
```

5. Rol



Define un rol dentro del sistema (Admin, Operator). Se relaciona con Employee para asignar permisos según el rol.

```
Role.cs  Observation.cs  Loan.cs  Employee.cs  Client.cs  Article.cs
InventorySystem.Entities
namespace InventorySystem.Entities.Entities
{
    public class Role
    {
        public int RoleId { get; set; }          // PK autoincremental
        public string Name { get; set; } = null!; // Nombre del rol (e.g., Admin, Clerk)
        public ICollection<Employee> Employees { get; set; } = new List<Employee>();
    }
}
```

6. Observación

Representa una anotación o comentario sobre un préstamo. Se relaciona uno a muchos con Loan.

```
Role.cs  Observation.cs  Loan.cs  Employee.cs  Client.cs  Article.cs
InventorySystem.Entities
namespace InventorySystem.Entities.Entities
{
    public class Observation
    {
        public Guid ObservationId { get; set; }      // Identificador único
        public Guid LoanId { get; set; }                // FK hacia Loan
        public string Text { get; set; } = null!;        // Texto de la observación
        public DateTime CreatedAt { get; set; }         // Fecha de creación
        public Loan Loan { get; set; } = null!;
    }
}
```

7. Usuario

Representa a un usuario del sistema. Tiene datos básicos como nombre, email, contraseña hasheada y fecha de creación. Se relaciona uno a uno con Employee si el usuario es un empleado.

```
User.cs  Role.cs  Observation.cs  Loan.cs  Employee.cs  Client.cs  Article.cs
InventorySystem.Entities
namespace InventorySystem.Entities.Entities
{
    public class User
    {
        public Guid UserId { get; set; }          // Identificador único
        public string FirstName { get; set; } = null!; // Nombre de pila
        public string LastName { get; set; } = null!; // Apellido
        public string Email { get; set; } = null!;   // Correo electrónico
        public string PasswordHash { get; set; } = null!; // Hash de la contraseña
        public DateTime CreatedAt { get; set; }       // Fecha de alta
        public Employee? Employee { get; set; }
    }
}
```

- (Opcional) MyApp.Tests (No aplica)

2. Scripts de base de datos

- Script SQL para crear la BD y tablas (usuarios, roles, artículos, préstamos).

No aplica un script para DataBase ya que se aplican migraciones para cubrirla.



- Migratios

1. 20250628204740_InitialCreate

```
20250628204740_InitialCreate.cs  20250628204740_InitialCreate.cs
InventorySystem.DataAccess          InventorySystem.DataAccess.Migrations.InitialCreate

7  {
8      /// <inheritdoc />
9      1 reference
10     public partial class InitialCreate : Migration
11     {
12         /// <inheritdoc />
13         0 references
14         protected override void Up(MigrationBuilder migrationBuilder)
15         {
16             migrationBuilder.CreateTable(
17                 name: "Articles",
18                 columns: table => new
19                 {
20                     ArticleId = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
21                     Code = table.Column<string>(type: "nvarchar(max)", nullable: false),
22                     Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
23                     Category = table.Column<string>(type: "nvarchar(max)", nullable: false),
24                     Status = table.Column<string>(type: "nvarchar(max)", nullable: false),
25                     Location = table.Column<string>(type: "nvarchar(max)", nullable: false)
26                 },
27                 constraints: table =>
28                 {
29                     table.PrimaryKey("PK_Articles", x => x.ArticleId);
30                 });
31
32             migrationBuilder.CreateTable(
33                 name: "Clients",
34                 columns: table => new
35                 {
36                     ClientId = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
37                     FirstName = table.Column<string>(type: "nvarchar(max)", nullable: false),
38                     LastName = table.Column<string>(type: "nvarchar(max)", nullable: false),
39                     Email = table.Column<string>(type: "nvarchar(max)", nullable: false),
40                     Phone = table.Column<string>(type: "nvarchar(max)", nullable: false)
41                 },
42                 constraints: table =>
43                 {
44                     table.PrimaryKey("PK_Clients", x => x.ClientId);
45                 });
46
47             migrationBuilder.CreateTable(
48                 name: "Roles",
49                 columns: table => new
50                 {
51                     RoleId = table.Column<int>(type: "int", nullable: false)
52                         .Annotation("SqlServer:Identity", "1, 1"),
53                     Name = table.Column<string>(type: "nvarchar(max)", nullable: false)
54                 },
55                 constraints: table =>
56             );
57         }
58
59     }
59 %
```

2. AppDbContextModelSnapshot

```
AppDbContext..ISnapshot.cs  20250628204740_InitialCreate.cs
InventorySystem.DataAccess          InventorySystem.DataAccess.Migrations.AppDbContextMo
AppDbContext..ISnapshot.cs        AppDbContext..ISnapshot.cs
InventorySystem.DataAccess        InventorySystem.DataAccess.Migrations.AppDbContextMo

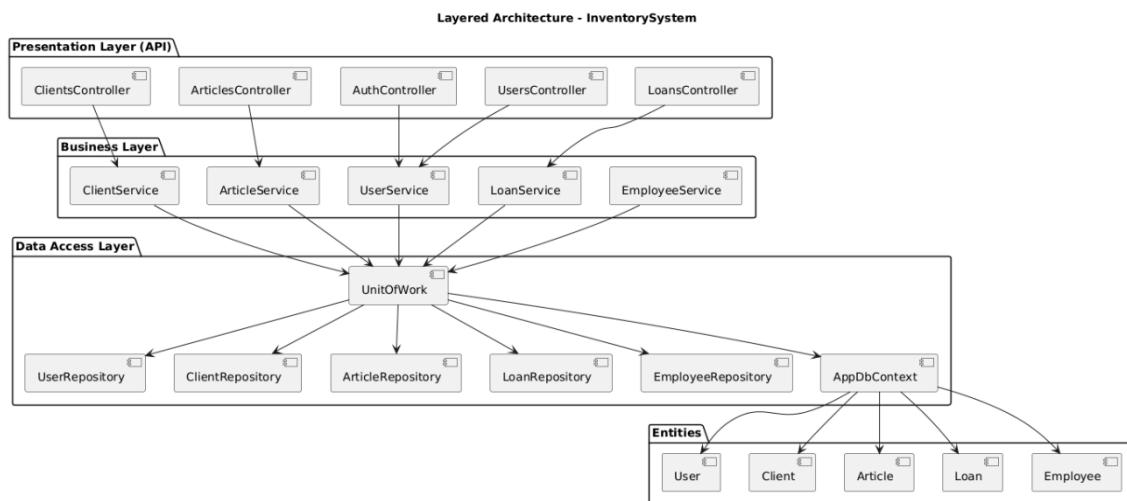
13 [DbContext(typeof(AppDbContext))]
14 0 references
15 partial class AppDbContextModelSnapshot : ModelSnapshot
16 {
17     0 references
18     protected override void BuildModel(ModelBuilder modelBuilder)
19     {
20         #pragma warning disable 612, 618
21         modelBuilder
22             .HasAnnotation("ProductVersion", "9.0.6")
23             .HasAnnotation("Relational:MaxIdentifierLength", 128);
24
25         SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder);
26
27         modelBuilder.Entity("InventorySystem.Entities.Entities.Article", b =>
28         {
29             b.Property<Guid>("ArticleId")
30                 .ValueGeneratedOnAdd()
31                 .HasColumnType("uniqueidentifier");
32
33             b.Property<string>("Category")
34                 .IsRequired()
35                 .HasColumnType("nvarchar(max)");
36
37             b.Property<string>("Code")
38                 .IsRequired()
39                 .HasColumnType("nvarchar(max)");
40
41             b.Property<string>("Location")
42                 .IsRequired()
43                 .HasColumnType("nvarchar(max)");
44
45             b.Property<string>("Name")
46                 .IsRequired()
47                 .HasColumnType("nvarchar(max)");
48
49             b.Property<string>("Status")
50                 .IsRequired()
51                 .HasColumnType("nvarchar(max)");
52
53             b.HasKey("ArticleId");
54
55             b.ToTable("Articles");
56         });
57
58         modelBuilder.Entity("InventorySystem.Entities.Entities.Client", b =>
59         {
60             b.Property<Guid>("ClientId")
61                 .ValueGeneratedOnAdd();
62         });
63     }
63 %
```

3. Diagramas UML

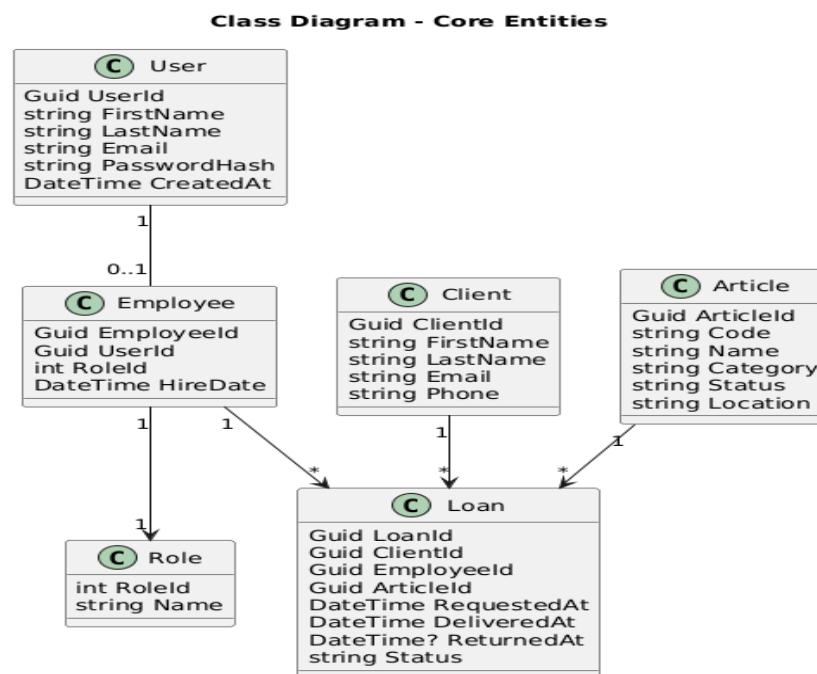
Diagramas generados con PlantUML (<https://plantuml.com/es/class-diagram>)

En el repositorio de GitHub se añadirá los códigos para su implementación.

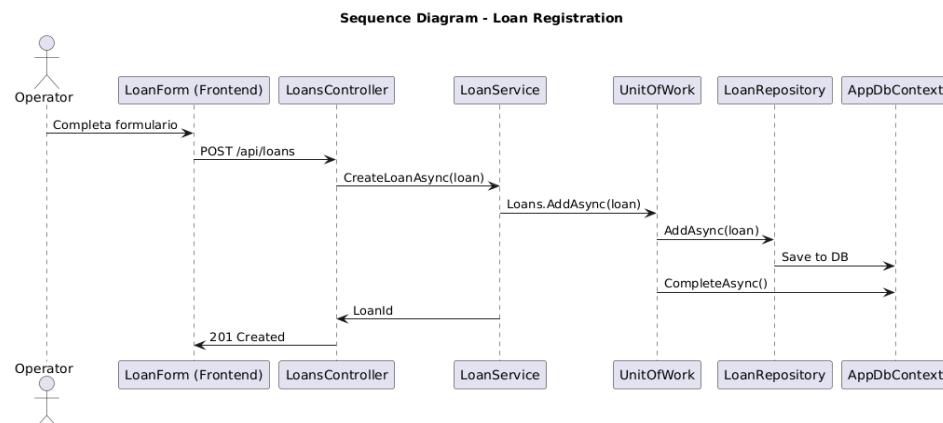
- **Diagrama de capas** mostrando dependencias.



- **Diagrama de clases** de entidades principales (Usuario, Artículo, Préstamo).



- **Diagrama de secuencia** de un flujo de préstamo.



4. Documentación técnica

En el repositorio de GitHub se implementará un read me detallado con los puntos solicitados.

- README con:
 - Instrucciones de despliegue.
 - Descripción de capas y patrones usados (Repository, Unit of Work).
 - Ejemplos de llamadas a la API interna si aplica (Swagger).

5. Pruebas automatizadas

- Visualización del Front-End
 - 1. Swagger



The image displays two separate instances of the Swagger UI interface, each showing a list of API endpoints for different modules.

Top Window (Articles API):

- Articles:**
 - GET /api/Articles
 - POST /api/Articles
 - GET /api/Articles/{id}
 - PUT /api/Articles/{id} (highlighted in orange)
 - DELETE /api/Articles/{id} (highlighted in red)
- Auth:**
 - POST /api/Auth/login
 - POST /api/Auth/register
- Clients:**
 - GET /api/Clients
 - POST /api/Clients
 - GET /api/Clients/{id}
 - PUT /api/Clients/{id} (highlighted in orange)
 - DELETE /api/Clients/{id} (highlighted in red)
- Employees:**
 - GET /api/Employees

Bottom Window (Loans API):

- Loans:**
 - GET /api/Loans
 - POST /api/Loans
 - GET /api/Loans/{id}
 - PUT /api/Loans/{id} (highlighted in orange)
 - DELETE /api/Loans/{id} (highlighted in red)
 - PUT /api/Loans/{id}/status
- Observations:**
 - GET /api/Observations/loan/{loanId}
 - POST /api/Observations
- Users:**
 - GET /api/Users
 - POST /api/Users
 - GET /api/Users/{id}
 - PUT /api/Users/{id} (highlighted in orange)
 - DELETE /api/Users/{id} (highlighted in red)
 - POST /api/Users/register

2. Aplicativo Web

- LogIn

A screenshot of a web browser displaying a login page. The title bar shows the URL <https://localhost:7127/paginas/Login.html>. The main content is a dark-themed login form with a central modal window. The modal has a header "Welcome Back" with a padlock icon. It contains two input fields: "Email" and "Password", both with placeholder text. Below the fields is a blue "Login" button. At the bottom of the modal, there is a link "Don't have an account? [Register](#)".

- Registrarse



Register New User

First Name

Last Name

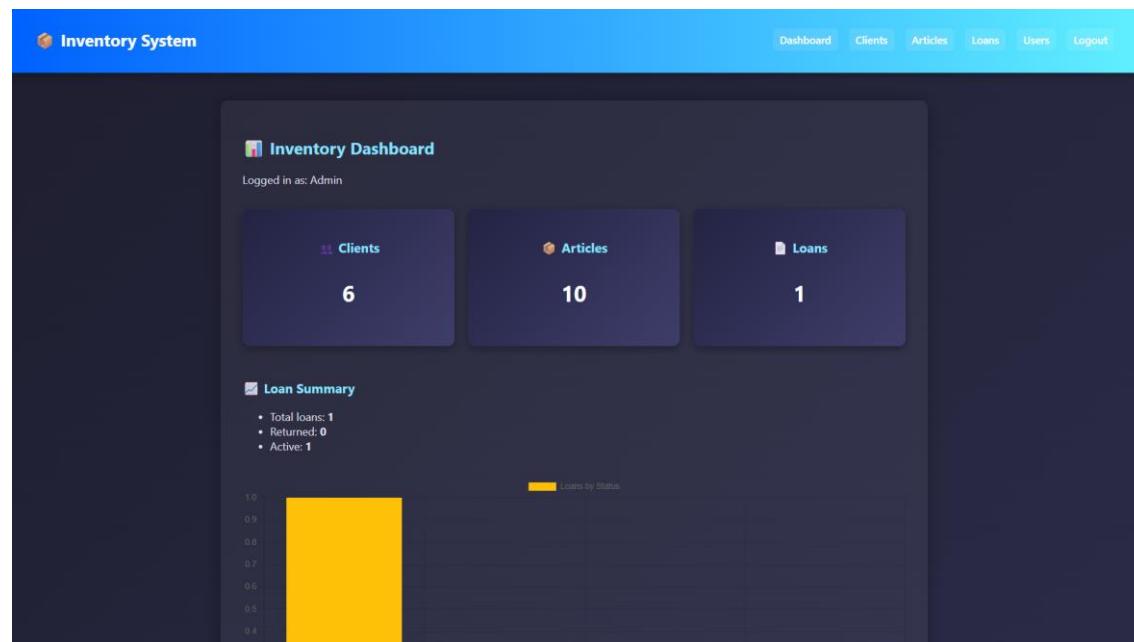
operator@system.com

...

Register

Already have an account? [Login here](#)

- Dashboard



- Clientes



Inventory System

Add Client

Registered Clients

Full Name	Email	Phone	Actions
Sofia Ramírez	ana.ramirez@example.com	601234567	<button>Edit</button> <button>Delete</button>
Elena López	elena.lopez@example.com	605678901	<button>Edit</button> <button>Delete</button>
Elena López	elena.lopez@example.com	605678901	<button>Edit</button> <button>Delete</button>
Maria Gómez	maria.gomez@example.com	603456789	<button>Edit</button> <button>Delete</button>
Carlos Mora	carlos.mora@example.com	604567890	<button>Edit</button> <button>Delete</button>
Carlos Mora	carlos.mora@example.com	604567890	<button>Edit</button> <button>Delete</button>

- Artículos

Inventory System

Add Article

Articles List

Code	Name	Category	Status	Location	Actions
ART-004	Router TP-Link	Networking	Available	Shelf B8	<button>Save</button> <button>Delete</button>
ART-005	Teclado Logitech	Accessories	Available	Shelf C1	<button>Edit</button> <button>Delete</button>
ART-004	Router TP-Link	Networking	Available	Shelf B2	<button>Edit</button> <button>Delete</button>
ART-002	Monitor Samsung	Electronics	Available	Shelf A2	<button>Edit</button> <button>Delete</button>

- Préstamos



The screenshot shows a dark-themed web application. At the top, there's a blue header bar with the title "Inventory System". Below it is a navigation bar with links: Dashboard, Clients, Articles, Loans, Users, and Logout. A central modal dialog box is open, asking "localhost:7127 says Delete this loan?". It has two buttons: "OK" and "Cancel". The main content area is titled "Loans" and contains a table with columns: Client, Article, Delivery, Return, Status, and Actions. One row in the table is highlighted, showing "Sofía Ramírez" as the Client, "Router TP-Link" as the Article, "2025-06-29" as the Delivery date, and "Pending" as the Status. In the Actions column for this row, there are four buttons: "Mark Returned" (with a checkmark), "Delete" (with a trash icon), "Approve" (with a checkmark), and "Reject" (with a cross icon). At the bottom of the page, a copyright notice reads "© 2025 Inventory System. All rights reserved."

- Usuarios

The screenshot shows a dark-themed web application. At the top, there's a blue header bar with the title "Inventory System". Below it is a navigation bar with links: Dashboard, Clients, Articles, Loans, Users, and Logout. A central modal dialog box is open, titled "Create New User". It contains fields for First Name, Last Name, Email (set to "operator@system.com"), and a password field (partially visible as "..."). Below these fields is a "Role:" dropdown menu with "Select Role" selected. At the bottom of the modal is a large blue "Create" button. To the right of the modal, there's a section titled "Existing Users" with a table. The table has columns: Name, Email, and Role. It lists two users: "Aby" (Email: aby@gmail.com, Role: Admin) and "Tuston" (Email: tuston@gmail.com, Role: Operator). There is a "Save" button next to the "Aby" row. At the bottom right of the page, there's a "Logout" link.

- Pruebas
- Autenticación en swagger
 - 1. Generación de token

The screenshot shows a dark-themed Swagger API documentation page. At the top, there's a table with columns: Code and Details. A row for status code 200 is expanded, showing the "Response body" which contains a JSON object with fields like "token", "employeeId", and "role". Below this, there's a "Download" button. Further down, there's a "Response headers" section with a JSON object containing "Content-Type", "Date", and "Server" fields. At the bottom, there's a "Responses" section with a table for status code 200, showing "OK" under the "Description" column and "No links" under the "Links" column.

- 2. Método Get en Usuarios



Code Details

200 Response body

```
[{"articleId": "12345678-ef-c6db-40c1-9cf1-180a127f4932", "code": "ART-004", "name": "Router TP-Link", "category": "Networking", "status": "Available", "location": "Shelf B2"}, {"articleId": "f123e287-8cb7-4a2c-9077-274fdd44666", "code": "ART-005", "name": "Teclado Logitech", "category": "Accessories", "status": "Available", "location": "Shelf C1"}, {"articleId": "933193c-32ad-4742-b46a-2dc5ffffe11a", "code": "ART-006", "name": "Router TP-Link", "category": "Networking", "status": "Available", "location": "Shelf B2"}, {"articleId": "ea0f04d27-6344-41b9-8b85-5576a3424b27", "code": "ART-002", "name": "Monitor Samsung", "category": "Peripherals", "status": "Available", "location": "Shelf A1"}]
```

Response headers

```
content-type: application/json; charset=utf-8
date: Sun, 29 Jun 2025 19:45:46 GMT
server: Kestrel
```

Responses

Code	Description
200	OK

Links
No links

Enlace a GitHub: [esteebaan/InventorySystem: Proyecto - Prueba Final](#)

En el repositorio de GitHub se implementa un ReadMe con las instrucciones necesarias para el uso y despliegue del proyecto, adicional a eso se agrega un script SQL para pruebas en la base de datos y usuarios creados para pruebas.

6. Anexos

- **Prompts usados para AI (ChatGPT)**

1. **Prompts del plan de implementación y arquitectura**

- "Ayudame a estructurar una arquitectura por capas para un sistema web en ASP.NET Core Web API, sin usar Razor ni MVC, solamente Web API y frontend estático en wwwroot."
- "Quiero desacoplar la lógica de negocio de los controladores usando interfaces y servicios. ¿Cómo puedo organizar las carpetas y proyectos en Visual Studio?"
- "Definí por favor la clase **AppDbContext** y configurá las relaciones entre entidades como User, Employee, Role, Loan, etc. con Fluent API."

2. **Prompts técnicos sobre desarrollo de funcionalidades**

- "Mostrame cómo implementar un endpoint para registrar usuarios con rol por defecto 'Operator', y cómo incluir el **employeeId** y **role** en la respuesta del login."
- "Necesito que el DTO de creación de usuario (**CreateUserDto**) tenga el campo **role** como string, pero que el backend lo convierta correctamente para guardar en la entidad **Employee**."
- "Cómo configuro AutoMapper para mapear **CreateClientDto**, **UpdateClientDto** y **ClientDto** con la entidad **Client**."

3. **Prompts sobre interacción con la API**

- "Mostrame un ejemplo de llamada a la API /api/clients con token JWT desde Postman, incluyendo los headers necesarios."



- "Cómo puedo enviar una solicitud POST al endpoint `/api/loans` con los campos `clientId`, `articleId`, `employeeId`, `deliveredAt`, `status` usando fetch desde el frontend."

4. Prompts sobre configuración y despliegue

- "Cómo configuro el `appsettings.json` para que incluya las claves del JWT, y cómo lo inyecto en `Program.cs` usando `IOptions<JwtSettings>`."
- "Mostrame cómo generar automáticamente la base de datos con `dotnet ef migrations` y `dotnet ef database update` usando la terminal."

5. Prompts sobre JWT y autenticación

- "No me está generando bien el token JWT, solo me devuelve el token pero no el `employeeId` ni el `role`. ¿Cómo arreglo el `AuthController` para incluirlos en la respuesta?"
- "¿Cómo agrego el rol en los claims del token JWT para poder usar `[Authorize(Policy = "AdminOnly")]`?"
- "Ayudame a configurar `JwtSettings` en `appsettings.json` y a inyectarlo correctamente en `Program.cs` usando `builder.Services.Configure<JwtSettings>`."

6. Prompts sobre errores de métodos y controladores

- "Me está tirando `AutoMapperMappingException: Missing type map configuration or unsupported mapping` al actualizar un cliente. Revisá el mapping del `UpdateClientDto` con `Client`."
- "Cuando intento crear un préstamo me dice 'Error registering loan'. Ya estoy enviando el `employeeId`, ¿qué puede estar mal en el backend?"
- "No me está cargando el resumen en el dashboard, revisá por qué el `fetch('/api/loans')` no me devuelve nada."

7. Prompts sobre estructura en capas y DTOs

- "¿Está bien que los DTOs se usen en la capa Business? Porque ahora estoy usando `CreateUserDto` en `UserService` pero me da error de acoplamiento."
- "¿Cómo hago para que los controladores usen solo DTOs y la capa Business trabaje únicamente con entidades? No quiero que la capa de negocio conozca los DTOs."

8. Prompts sobre errores en migraciones / EF Core

- "Me sale `Metadata file ...InventorySystem.Business.dll could not be found`, ¿cómo soluciono ese error de compilación para poder hacer la migración?"
- "Estoy intentando hacer `dotnet ef migrations add InitialCreate` pero me tira error porque no encuentra la clase `AppDbContext`. ¿Qué proyecto debería tener como startup y cómo lo configuro en VS?"