

**Universidad ORT Uruguay
Facultad de Ingeniería**

Obligatorio 1 de Diseño de Aplicaciones 2

Agustín Juárez - 236487, Agustín Campón - 233006

Tutor: Nicolas Fierro

[Repositorio con la solución](#)

2023

Evidencia de la aplicación de Clean Code

Comenzando con la evidencia de la aplicación de Clean Code en nuestro sistema, a continuación mostraremos la aplicación basándonos en una clase creada para el controlador de nuestra WEB API para los usuarios del sistema.

```
1. public class UserController : ControllerBase
2. {
3.     private readonly IUserService _userService;
4.     private readonly IRoleService _roleService;
5.     private readonly IUserRoleService _userRoleService;
6.     private readonly ISessionService _sessionService;
7.
8.     public UserController(IUserService userService, IRoleService
roleService, IUserRoleService userRoleService, ISessionService
sessionService)
9.     {
10.         _userService = userService;
11.         _roleService = roleService;
12.         _userRoleService = userRoleService;
13.         _sessionService = sessionService;
14.     }
15.
16.     // Index - Get all users (/api/users)
17.     [AuthenticationFilter]
18.     [RoleFilter(RoleType.Admin, RoleType.Blogger)]
19.     [HttpGet]
20.     public IActionResult GetUsers([FromQuery] UserSearchCriteriaModel
searchCriteria)
21.     {
22.         var retrievedUsers =
23.         _userService.GetAllUsers(searchCriteria.ToEntity());
24.         return Ok(retrievedUsers.Select(u => new UserModelOut(u)));
25.     }
26.
27.     // Get users ranking (/api/users/ranking)
28.     [AuthenticationFilter]
29.     [RoleFilter(RoleType.Admin)]
30.     [HttpGet("ranking")]
31.     public IActionResult GetUsersRanking([FromQuery] string startDate,
[FromQuery] string endDate)
32.     {
33.         try
34.         {
35.             DateTime start = DateTime.Parse(startDate);
36.             DateTime end = DateTime.Parse(endDate);
37.
38.             if(start >= end)
39.             {
40.                 return BadRequest("Invalid date format");
41.             }
42.         }
43.     }
44. }
```

```

42.         var retrievedUsers = _userService.GetUsersRanking(start,
end);
43.         return Ok(retrievedUsers.Select(u => new UserModelOut(u)));
44.     }
45.     catch (FormatException)
46.     {
47.         return BadRequest("Invalid date format");
48.     }
49.
50. }
51.
52. // Get user activities (/api/users/activities)
53. [AuthenticationFilter]
54. [HttpGet("activities")]
55. public IActionResult GetUserActivities()
56. {
57.     User currentUser = _sessionService.GetCurrentUser();
58.     List<Comment> comments = new List<Comment>();
59.     currentUser.Articles.ToList().ForEach(a =>
60.     {
61.         if(a.Comments != null)
62.         {
63.             List<Comment> newComments = a.Comments.Where(c =>
!c.IsViewed && !c.Author.Equals(currentUser)).ToList();
64.             comments.AddRange(newComments);
65.         }
66.     });
67.     return Ok(comments.Select(c => new CommentDetailModel(c)));
68. }
69.
70. // Show - Get specific user (/api/users/{id})
71. [AuthenticationFilter]
72. [RoleFilter(RoleType.Admin)]
73. [HttpGet("{id}", Name = "GetUser")]
74. public IActionResult GetUserById(int id)
75. {
76.     try
77.     {
78.         var retrievedUser = _userService.GetSpecificUser(id);
79.         return Ok(new UserModelOut(retrievedUser));
80.     }
81.     catch (ResourceNotFoundException e)
82.     {
83.         return NotFound(e.Message);
84.     }
85. }
86.
87. // Create - Create new user (/api/users)
88. [HttpPost]
89. public IActionResult CreateUser([FromBody] UserModelIn newUser)
90. {
91.     try
92.     {
93.         EnsureRolesHasValues(newUser.roles.Count);
94.         EnsureRolesExists(newUser);
95.

```

```

96.         // 1) Creo User
97.         var createdUser =
            _userService.CreateUser(newUser.ToCreateEntity());
98.
99.         foreach (int roleValue in new HashSet<int>(newUser.roles))
100.        {
101.            var role = _roleService.GetSpecificRole(roleValue);
102.            var userRole = new UserRole()
103.            {
104.                User = createdUser,
105.                Role = role
106.            };
107.
108.            _userRoleService.CreateUserRole(userRole);
109.        }
110.
111.
112.        var userModel = new UserModelOut(createdUser);
113.        return CreatedAtRoute("GetUser", new { id = userModel.Id
            }, userModel);
114.    }
115.    catch (InvalidResourceException e)
116.    {
117.        return BadRequest(e.Message);
118.    }
119.    catch (DuplicateResourceException e)
120.    {
121.        return Conflict(e.Message);
122.    }
123.    catch (ResourceNotFoundException e)
124.    {
125.        return NotFound(e.Message);
126.    }
127.    }
128.
129.    // Update - Update specific user (/api/users/{id})
130.    [AuthenticationFilter]
131.    [HttpPut("{id}")]
132.    public IActionResult Update(int id, [FromBody] UserModelIn
        updatedUser)
133.    {
134.        try
135.        {
136.            User currentUser = _sessionService.GetCurrentUser();
137.
138.            if(currentUser.Id == id ||
                currentUser.IsInRole(RoleType.Admin))
139.            {
140.                EnsureRolesHasValues(updatedUser.roles.Count);
141.                EnsureRolesExists(updatedUser);
142.
143.                // 1) Creo User
144.                var retrievedUser = _userService.UpdateUser(id,
                    updatedUser.ToUpdateEntity());
145.

```

```

146.         foreach (int roleValue in new
147.             HashSet<int>(updatedUser.roles))
148.         {
149.             var role =
150.                 _roleService.GetSpecificRole(roleValue);
151.             var userRole = new UserRole()
152.             {
153.                 User = retrievedUser,
154.                 Role = role
155.             };
156.             _userRoleService.CreateUserRole(userRole);
157.             return Ok(new UserModelOut(retrievedUser));
158.         } else
159.         {
160.             return Unauthorized("You are not authorized to
161.                 perform this action");
162.         }
163.     catch (InvalidResourceException e)
164.     {
165.         return BadRequest(e.Message);
166.     }
167.     catch (ResourceNotFoundException e)
168.     {
169.         return NotFound(e.Message);
170.     }
171.     catch (DuplicateResourceException e)
172.     {
173.         return Conflict(e.Message);
174.     }
175. }
176.
177. // Delete - Delete specific user (/api/users/{id})
178. [AuthenticationFilter]
179. [HttpDelete("{id}")]
180. [RoleFilter(RoleType.Admin)]
181. public IActionResult Delete(int id)
182. {
183.     try
184.     {
185.         _userService.DeleteUser(id);
186.         return NoContent();
187.     }
188.     catch (ResourceNotFoundException e)
189.     {
190.         return NotFound(e.Message);
191.     }
192. }
193.
194.
195.
196. private void EnsureRolesHasValues(int rolesLength)
197. {

```

```

198.         if (rolesLength == 0) throw new
InvalidResourceException("User should have at least one role");
199.     }
200.
201.     private void EnsureRolesExists(UserModelIn user)
202.     {
203.         foreach (int roleValue in new HashSet<int>(user.roles))
204.         {
205.             var role = _roleService.GetSpecificRole(roleValue);
206.         }
207.     }
208. }

```

Desde la perspectiva de Clean Code, algunas cosas buenas sobre la clase UserController proporcionada son:

- La clase sigue una convención de nomenclatura coherente y utiliza nombres significativos y claros para sus métodos y propiedades, lo que mejora la legibilidad.
- La clase utiliza la inyección de constructor para administrar las dependencias y sigue el Principio de responsabilidad única (SRP) al tener una responsabilidad única: administrar las operaciones relacionadas con el usuario.
- La clase sigue el Principio Abierto/Cerrado (OCP) al permitir una fácil extensión a través de la herencia y la decoración, como lo demuestra el uso de filtros en los métodos del controlador.
- Los métodos en la clase son pequeños y enfocados, con parámetros claros de entrada y salida y un número limitado de responsabilidades.
- La clase incluye manejo de excepciones para manejar posibles errores y proporcionar comentarios al usuario.
- La clase incluye una validación básica de las entradas, lo que ayuda a garantizar que el sistema se comporte correctamente y evita la propagación de errores.

Siempre hay lugar para mejoras, pero en general, la clase UserController proporcionada parece estar bien estructurada y es fácil de leer, comprender y mantener.

Evidencia de la aplicación de TDD

Para estas funcionalidades se usó la estrategia de outside-in, normalmente esta estrategia ejercita todo el sistema desde el exterior y puede involucrar múltiples capas del sistema, como la interfaz de usuario, la capa del controlador y la capa de servicio.

Para eso comenzamos escribiendo los primeros unit test lo cual fallaron, luego se escribió lo mínimo indispensable para que la prueba pase y por último el refactor para conservar todos los posibles escenarios y que la prueba matchee totalmente con la funcionalidad.

Un usuario debe poder registrarse y modificar su perfil

A continuación, captura de pantalla de Commit con las pruebas unitarias para la clase UserController, la cual contiene las funcionalidades para crear un nuevo usuario y modificarlo.

Commit: [989c314303ffaa963f410dbdd6a9548d29bb1c6b](#)

```
[TestMethod]
public void UpdateUser_UpdatesUserAndReturnsOk()
{
    var userToUpdate = CreateUser(1);
    var updatedUserModel = new UserModelIn
    {
        FirstName = "UpdatedFirstname",
        LastName = "UpdatedLastname",
        Password = "UpdatedPassword",
        Email = "updatedemail@gmail.com",
        Username = "updatedusername",
        roles = new List<int> { 1 }
    };

    _userServiceMock.Setup(service => service.UpdateUser(userToUpdate.Id, It.IsAny<User>())).Returns(userToUpdate);
    _roleServiceMock.Setup(service => service.GetSpecificRole(1)).Returns(new Role { Id = 1, RoleType = RoleType.Admin});

    _userServiceMock.Setup(service => service.CreateUserRole(It.IsAny<UserRole>())).Returns(new UserRole());

    var controller = new UserController(_userServiceMock.Object, _roleServiceMock.Object, _userServiceMock.Object);

    var result = controller.Update(userToUpdate.Id, updatedUserModel) as OkObjectResult;
    Assert.IsNotNull(result);
    Assert.AreEqual((int)HttpStatusCode.OK, result.StatusCode);

    var userModelOut = result.Value as UserModelOut;
    Assert.IsNotNull(userModelOut);
}
```

Un usuario debe poder loguearse y desloguearse del sistema

A continuación, captura de pantalla de Commit con las pruebas unitarias para la clase SessionController, la cual contiene las funcionalidades para loguearse y desloguearse del sistema.

Commit: [ed75828ee3a37389733f8763a1e31e9aeacbbe3d](#)

```
[TestClass]
public class SessionControllerTest
{
    private Mock<ISessionService> _sessionServiceMock;

    [TestInitialize]
    public void Setup()
    {
        _sessionServiceMock = new Mock<ISessionService>();
    }

    [TestMethod]
    public void Login_ValidCredentials_ReturnsToken()
    {
        // Arrange
        var sessionModel = new SessionModel { Email = "test@example.com", Password = "password" };
        Guid token = Guid.NewGuid();
        _sessionServiceMock.Setup(service => service.Authenticate(sessionModel.Email, sessionModel.Password)).Returns(token);
        var controller = new SessionController(_sessionServiceMock.Object);

        // Act
        var result = controller.Login(sessionModel) as OkObjectResult;

        // Assert
        Assert.IsNotNull(result);
        Assert.AreEqual((int)HttpStatusCode.OK, result.StatusCode);
    }
}
```

Un usuario logueado en el sistema puede realizar comentarios acerca de un artículo público y también responder comentarios.

A continuación, captura de pantalla de Commit con las pruebas unitarias para la clase CommentController, la cual contiene las funcionalidades para comentar un artículo.

Commit: [888fecad3a084ee815c9d86e08e46d23c6d6a3d9](#)


```

[TestMethod]
public void CreateCommentReplyShouldReturnBadRequestWhenCommentAlreadyHasReply()
{
    // Arrange
    var commentId = 1;
    var reply = new CommentReplyModel { Content = "test reply" };
    var comment = new Comment { Id = commentId, Reply = new Comment(), Author = new User(), Arti
    var currentUser = new User();

    _commentServiceMock.Setup(x => x.GetSpecificComment(It.IsAny<int>())).Returns(comment);
    _sessionServiceMock.Setup(x => x.GetCurrentUser(null)).Returns(currentUser);

    var controller = new CommentController(_articleServiceMock.Object, _commentServiceMock.Object

    // Act
    var result = controller.CreateCommentReply(commentId, reply) as BadRequestObjectResult;

    // Assert
    Assert.IsNotNull(result);
    Assert.AreEqual("Cannot reply to a replied comment", result.Value);
}

```

Se debe tener la posibilidad de buscar un artículo a partir de un texto

A continuación, captura de pantalla de Commit con las pruebas unitarias para la clase ArticleController, la cual contiene las funcionalidades para buscar un artículo a partir de un texto dado.

Commit: [6c3f11f3bce650caf44419b63bdb69a1e353fd8](https://github.com/6c3f11f3bce650caf44419b63bdb69a1e353fd8)

```

Blog.Domain.Tests/ArticleSearchCriteriaTests.cs
@@ -10,7 +10,7 @@ public class ArticleSearchCriteriaTests
    [TestMethod]
    public void Criteria_WithNullTitleAndContent_ReturnsTrue()
    {
        var articleSearchCriteria = new ArticleSearchCriteria { Title = null, Content = null };
        var articleSearchCriteria = new ArticleSearchCriteria { q = null };
        Expression<Func<Article, bool>> criteriaExpression = articleSearchCriteria.Criteria();
        Func<Article, bool> criteriaFunc = criteriaExpression.Compile();

    @@ -21,7 +21,7 @@ public void Criteria_WithNullTitleAndContent_ReturnsTrue()
    [TestMethod]
    public void Criteria_WithMatchingTitleAndContent_ReturnsTrue()
    {
        var articleSearchCriteria = new ArticleSearchCriteria { Title = "Test title", Content = "Test content" };
        var articleSearchCriteria = new ArticleSearchCriteria { q = "Test title" };
        Expression<Func<Article, bool>> criteriaExpression = articleSearchCriteria.Criteria();
        Func<Article, bool> criteriaFunc = criteriaExpression.Compile();

    @@ -32,7 +32,7 @@ public void Criteria_WithMatchingTitleAndContent_ReturnsTrue()
    [TestMethod]
    public void Criteria_WithMismatchedTitleAndContent_ReturnsFalse()
    {
        var articleSearchCriteria = new ArticleSearchCriteria { Title = "Wrong title", Content = "Wrong content" };
        var articleSearchCriteria = new ArticleSearchCriteria { q = "Wrong title" };
        Expression<Func<Article, bool>> criteriaExpression = articleSearchCriteria.Criteria();
        Func<Article, bool> criteriaFunc = criteriaExpression.Compile();

```

La aplicación debe contar con un módulo de administración de usuarios que permite crear, eliminar y modificar usuarios.

A continuación, captura de pantalla de Commit con las pruebas unitarias para la clase UserController, la cual contiene las funcionalidades para crear, eliminar y modificar usuarios.






Commit: [989c314303ffaa963f410dbdd6a9548d29bb1c6b](#)

```
96 +
97 +
98 +     [TestMethod]
99 +     public void CreateUser_CreatesUserAndReturnsCreatedUser()
100 +     {
101 +         var newUser = new UserModelIn
102 +         {
103 +             FirstName = "Firstname",
104 +             LastName = "Lastname",
105 +             Password = "Password",
106 +             Email = "email@gmail.com",
107 +             Username = "username",
108 +             roles = new List<int> { 1, 2 }
109 +         };
110 +
111 +         var createdUser = CreateUser(1);
112 +
113 +         _userServiceMock.Setup(service => service.CreateUser(It.IsAny<User>())).Returns(createdUser);
114 +         _roleServiceMock.Setup(service => service.GetSpecificRole(1)).Returns(new Role { Id = 1, RoleType = RoleType.Admin });
115 +         _roleServiceMock.Setup(service => service.GetSpecificRole(2)).Returns(new Role { Id = 2, RoleType = RoleType.User });
116 +
117 +         _userServiceMock.Setup(service => service.CreateUserRole(It.IsAny<UserRole>())).Returns(new UserRole { Id = 1, RoleId = 1, UserId = 1 });
118 +
119 +         var controller = new UserController(_userServiceMock.Object, _roleServiceMock.Object, _userServiceMock.Object);
120 +
121 +         var result = controller.CreateUser(newUser) as CreatedAtRouteResult;
122 +
123 +         Assert.IsNotNull(result);
124 +
125 +         var createdUserModel = result.Value as UserModelOut;
126 +
127 +         _userServiceMock.Verify(service => service.CreateUser(It.IsAny<User>()), Times.Once());
128 +
129 +         Assert.AreEqual(createdUser.Id, createdUserModel.Id);
130 +         Assert.AreEqual(newUser.Username, createdUserModel.Username);
131 +     }
132 +
133 +
134 +
135 +     [TestMethod]
136 +     public void UpdateUser_UpdatesUserAndReturnsOk()
137 +     {
138 +         var userToUpdate = CreateUser(1);
139 +         var updatedUserModel = new UserModelIn
140 +         {
141 +             FirstName = "UpdatedFirstname"
```

Evidencia de Cobertura de Código










Con respecto a la cobertura total del código, logramos un 93% total, siendo un buen valor que puede asegurar el buen funcionamiento del sistema.

Total

Symbol	Coverage (%) ▾	Uncovered/Total Stmts.
▼  Total	93%	58/887
>  Blog.Services	95%	11/223
>  Blog.Domain	95%	13/255
>  Blog.WebApi	92%	25/312
>  Blog.DataAccess	91%	9/97

Services

En el paquete de Services se logró un 95% de cobertura de código.

Symbol	Coverage (%) ▾	Uncovered/Total Stmts.
▼  Total	93%	58/887
▼  Blog.Services	95%	11/223
▼  Blog.Services	95%	11/223
>  RoleService	100%	0/17
>  UserRoleService	100%	0/9
>  CommentService	97%	1/32
>  UserService	97%	2/76
>  SessionService	94%	2/31
>  ArticleService	90%	6/58

Domain

En el paquete de Domain se logró un 95% de cobertura de código.

Symbol	Coverage (%) ▾	Uncovered/Total Stmts.
▼ Total	93%	58/887
> Blog.Services	95%	11/223
▼ Blog.Domain	95%	13/255
▼ {} Blog.Domain	95%	13/255
> Comment	100%	0/35
> Role	100%	0/15
> {} SearchCriteria	100%	0/17
> {} Exceptions	100%	0/9
> Article	95%	2/42
> User	94%	7/118
> {} Entities	80%	2/10
> UserRole	78%	2/9


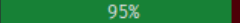
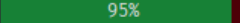
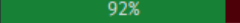
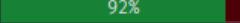
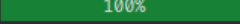
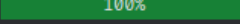
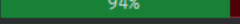
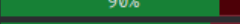
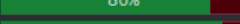
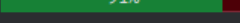
DataAccess

En el paquete de DataAccess se logró un 91% de cobertura de código.

Symbol	Coverage (%) ▾	Uncovered/Total Stmts.
▼ Total	93%	58/887
> Blog.Services	95%	11/223
> Blog.Domain	95%	13/255
> Blog.WebApi	92%	25/312
▼ Blog.DataAccess	91%	9/97
▼ {} Blog.DataAccess	91%	9/97
> ArticleRepository	100%	0/9
> RoleRepository	100%	0/10
> SessionRepository	100%	0/10
> UserRoleRepository	100%	0/3
> CommentRepository	89%	3/28
> BaseRepository<T>	88%	3/25
> UserRepository	75%	3/12

WebApi

En el paquete de WebApi se logró un 92% de cobertura de código.

Symbol	Coverage (%) ▾	Uncovered/Total Stmts.
▼ 📁 Total	93% 	58/887
> 📁 Blog.Services	95% 	11/223
> 📁 Blog.Domain	95% 	13/255
▼ 📁 Blog.WebApi	92% 	25/312
▼ () Blog.WebApi.Controllers	92% 	25/312
> 📁 RoleController	100% 	0/12
> 📁 SessionController	100% 	0/20
> 📁 UserController	94% 	7/126
> 📁 ArticleController	90% 	9/90
> 📁 CommentController	86% 	9/64
> 📁 Blog.DataAccess	91% 	9/97