Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio 2 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 =
   _userService.GetAllUsers(searchCriteria.ToEntity());
23.        return Ok(retrievedUsers.Select(u => new UserModelOut(u)));
24.    }
25.
26.    // Get users ranking (/api/users/ranking)
27.    [AuthenticationFilter]
28.    [RoleFilter(RoleType.Admin)]
29.    [HttpGet("ranking")]
30.    public IActionResult GetUsersRanking([FromQuery] string startDate,
   [FromQuery] string endDate)
31.    {
32.        try
33.        {
34.            DateTime start = DateTime.Parse(startDate);
35.            DateTime end = DateTime.Parse(endDate);
36.
37.            if(start >= end)
38.            {
39.                return BadRequest("Invalid date format");
40.            }
41.
```

```
            var retrievedUsers = _userService.GetUsersRanking(start, end);
            return Ok(retrievedUsers.Select(u => new UserModelOut(u)));
        }
        catch (FormatException)
        {
            return BadRequest("Invalid date format");
        }

    }

    // Get user activities (/api/users/activities)
    [AuthenticationFilter]
    [HttpGet("activities")]
    public IActionResult GetUserActivities()
    {
        User currentUser = _sessionService.GetCurrentUser();
        List<Comment> comments =  new List<Comment>();
        currentUser.Articles.ToList().ForEach(a =>
        {
            if(a.Comments != null)
            {
                List<Comment> newComments = a.Comments.Where(c => !c.IsViewed && !c.Author.Equals(currentUser)).ToList();
                comments.AddRange(newComments);
            }
        });
        return Ok(comments.Select(c => new CommentDetailModel(c)));
    }

    // Show - Get specific user (/api/users/{id})
    [AuthenticationFilter]
    [RoleFilter(RoleType.Admin)]
    [HttpGet("{id}", Name = "GetUser")]
    public IActionResult GetUserById(int id)
    {
        try
        {
            var retrievedUser = _userService.GetSpecificUser(id);
            return Ok(new UserModelOut(retrievedUser));
        }
        catch (ResourceNotFoundException e)
        {
            return NotFound(e.Message);
        }
    }

    // Create - Create new user (/api/users)
    [HttpPost]
    public IActionResult CreateUser([FromBody] UserModelIn newUser)
    {
        try
        {
            EnsureRolesHasValues(newUser.roles.Count);
            EnsureRolesExists(newUser);

```

```
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
    HashSet<int>(updatedUser.roles))
147.                     {
148.                         var role =
    _roleService.GetSpecificRole(roleValue);
149.                         var userRole = new UserRole()
150.                         {
151.                             User = retrievedUser,
152.                             Role = role
153.                         };
154.
155.                         _userRoleService.CreateUserRole(userRole);
156.                     }
157.                     return Ok(new UserModelOut(retrievedUser));
158.                 } else
159.                 {
160.                     return Unauthorized("You are not authorized to
    perform this action");
161.                 }
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

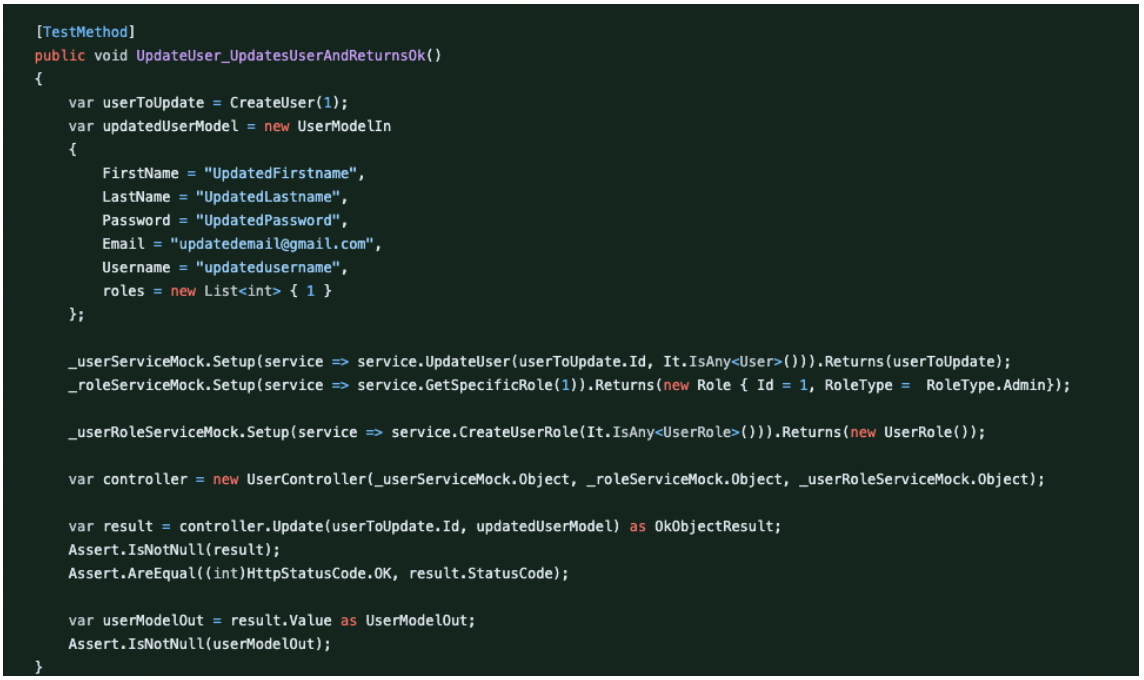
Para estas funcionalidades se usó la estrategia de outside-in, normalmente esta estrategia ejercita todo el sistema desde el exterior y puede involucrar múltiples capas del sistema, como la interfaz de usuario, la capa del controlador y la capa de servicio.

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});

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

    var controller = new UserController(_userServiceMock.Object, _roleServiceMock.Object, _userRoleServiceMock.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.

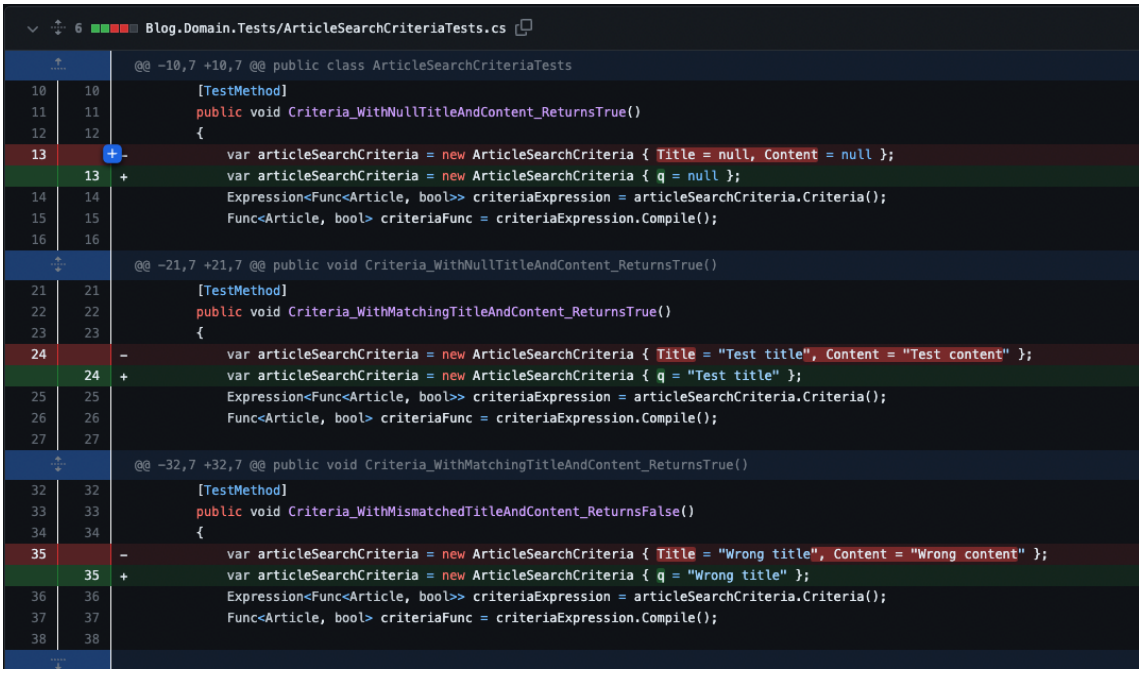
Commit: 888fecad3a084ee815c9d86e08e46d23c6d6a3d9

```
[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.Objec

    // 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.

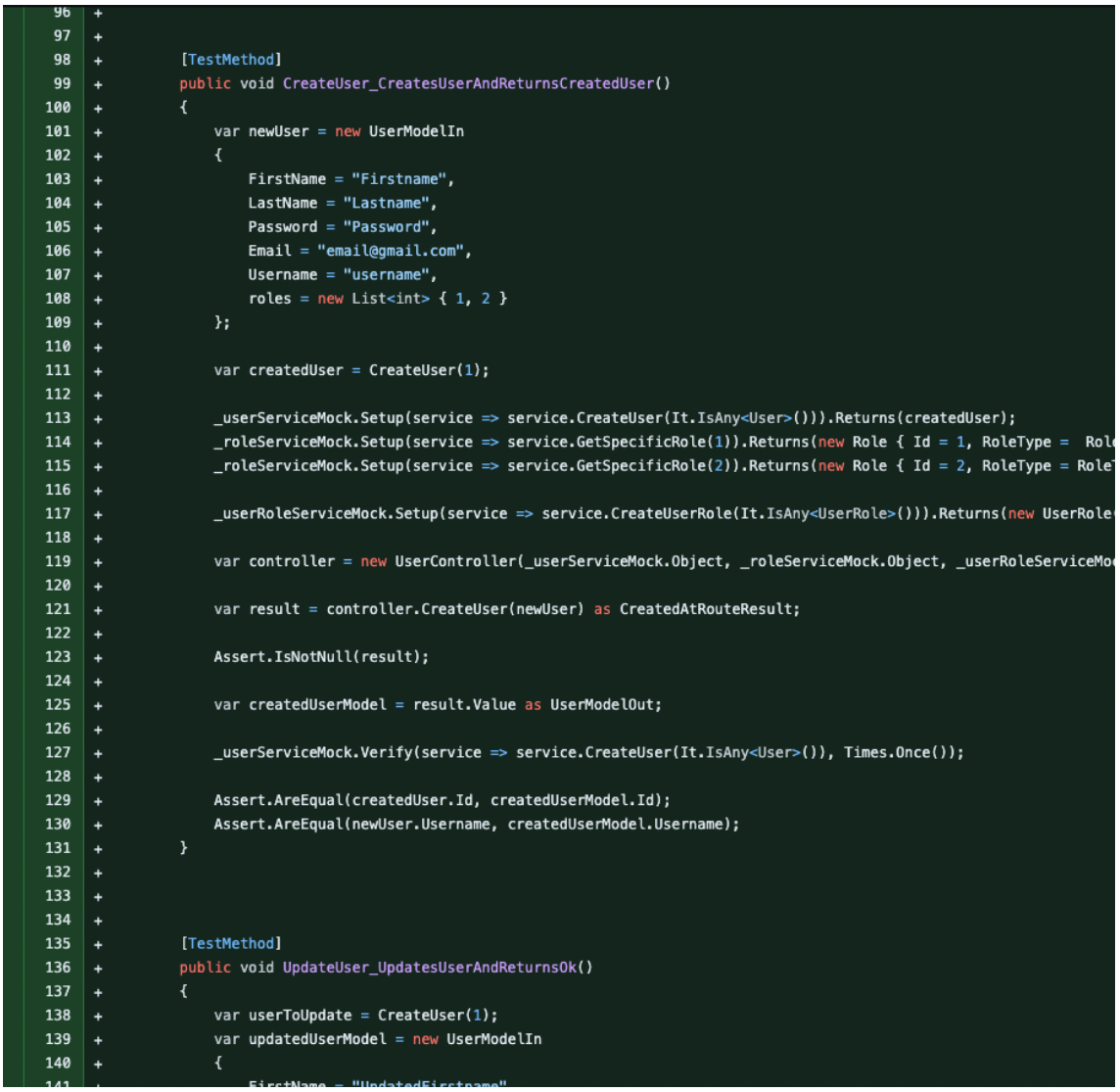
Commit: [c6c3f11f3bce650caf44419b63bdb69a1e353fd8]



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

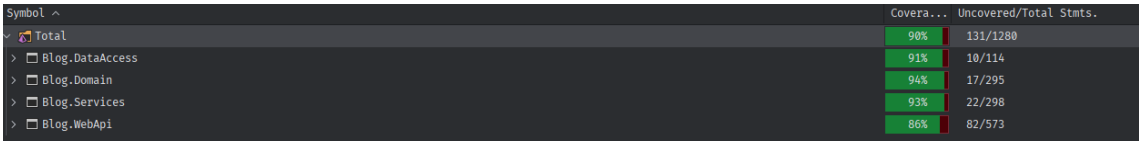


```
+
+
+       [TestMethod]
+       public void CreateUser_CreatesUserAndReturnsCreatedUser()
+       {
+           var newUser = new UserModelIn
+           {
+               FirstName = "Firstname",
+               LastName = "Lastname",
+               Password = "Password",
+               Email = "email@gmail.com",
+               Username = "username",
+               roles = new List<int> { 1, 2 }
+           };
+
+           var createdUser = CreateUser(1);
+
+           _userServiceMock.Setup(service => service.CreateUser(It.IsAny<User>())).Returns(createdUser);
+           _roleServiceMock.Setup(service => service.GetSpecificRole(1)).Returns(new Role { Id = 1, RoleType =  Rol
+           _roleServiceMock.Setup(service => service.GetSpecificRole(2)).Returns(new Role { Id = 2, RoleType = Role
+
+           _userRoleServiceMock.Setup(service => service.CreateUserRole(It.IsAny<UserRole>())).Returns(new UserRole
+
+           var controller = new UserController(_userServiceMock.Object, _roleServiceMock.Object, _userRoleServiceMo
+
+           var result = controller.CreateUser(newUser) as CreatedAtRouteResult;
+
+           Assert.IsNotNull(result);
+
+           var createdUserModel = result.Value as UserModelOut;
+
+           _userServiceMock.Verify(service => service.CreateUser(It.IsAny<User>()), Times.Once());
+
+           Assert.AreEqual(createdUser.Id, createdUserModel.Id);
+           Assert.AreEqual(newUser.Username, createdUserModel.Username);
+       }
+
+
+
+       [TestMethod]
+       public void UpdateUser_UpdatesUserAndReturnsOk()
+       {
+           var userToUpdate = CreateUser(1);
+           var updatedUserModel = new UserModelIn
+           {
+               FirstName = "UpdatedFirstname"
```

Evidencia de Cobertura de Código

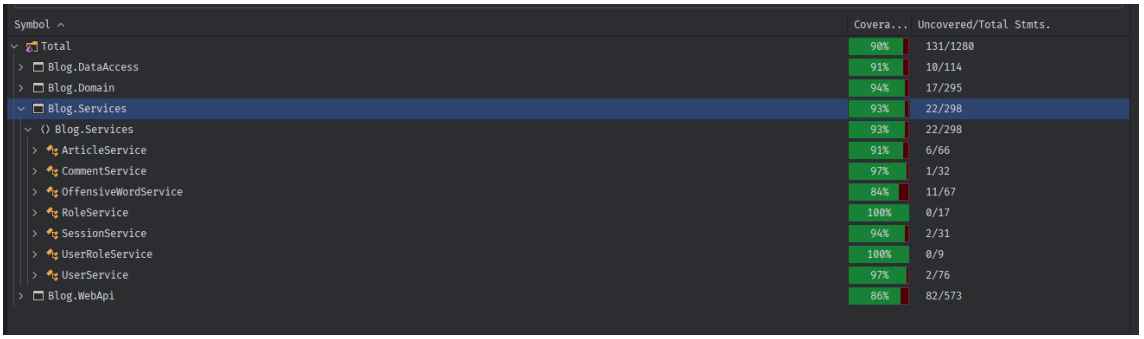
Con respecto a la cobertura total del código, logramos un 90% total, siendo un buen valor que puede asegurar el buen funcionamiento del sistema.

Total



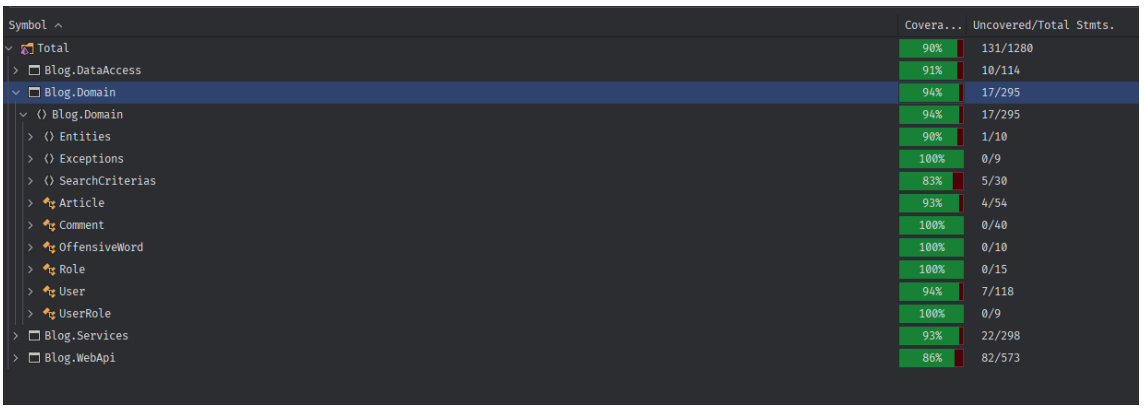
Services

En el paquete de Services se logró un 93% de cobertura de código.



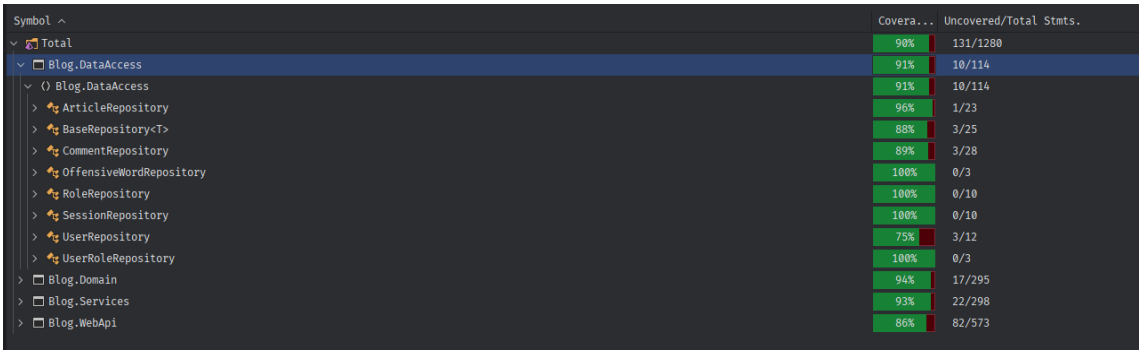
Domain

En el paquete de Domain se logró un 94% de cobertura de código.



DataAccess

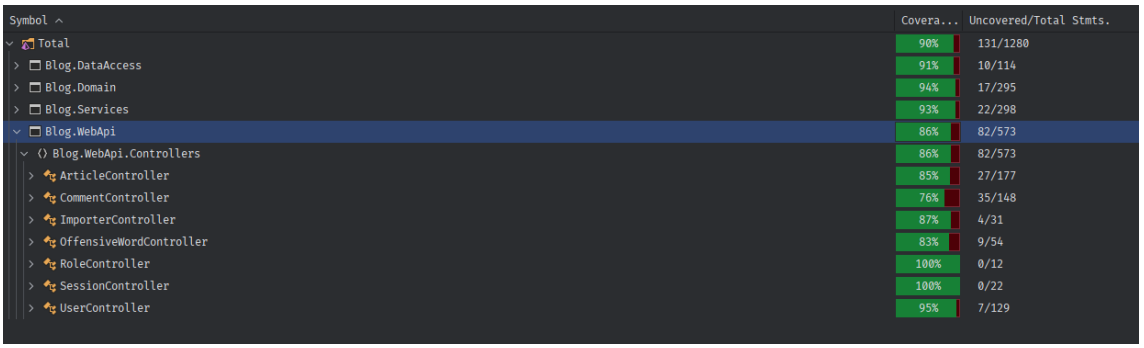
En el paquete de DataAccess se logró un 91% de cobertura de código.



Symbol	Covera...	Uncovered/Total Stmts.
Total	90%	131/1280
Blog.DataAccess	91%	10/114
{} Blog.DataAccess	91%	10/114
ArticleRepository	96%	1/23
BaseRepository<T>	88%	3/25
CommentRepository	89%	3/28
OffensiveWordRepository	100%	0/3
RoleRepository	100%	0/10
SessionRepository	100%	0/10
UserRepository	75%	3/12
UserRoleRepository	100%	0/3
Blog.Domain	94%	17/295
Blog.Services	93%	22/298
Blog.WebApi	86%	82/573

WebApi

En el paquete de WebApi se logró un 86% de cobertura de código.



Symbol	Covera...	Uncovered/Total Stmts.
Total	90%	131/1280
Blog.DataAccess	91%	10/114
Blog.Domain	94%	17/295
Blog.Services	93%	22/298
Blog.WebApi	86%	82/573
{} Blog.WebApi.Controllers	86%	82/573
ArticleController	85%	27/177
CommentController	76%	35/148
ImporterController	87%	4/31
OffensiveWordController	83%	9/54
RoleController	100%	0/12
SessionController	100%	0/22
UserController	95%	7/129