Universidad ORT Uruguay

Facultad de Ingeniería

Obligatorio Ingeniería de Software Ágil 2

Entregado como requisito para la obtención del título de Ingeniero en Sistemas

Milena dos Santos – 254813

Guzmán Dupont – 230263

Julieta Sarantes – 251105

Tutores: Alvaro Ortas, Carina Fontán

2023

Identificación y justificación de los bugs

Para esta nueva etapa se pidió la corrección de tres bugs, encontrados en la entrega anterior, a elección por los estudiantes.
Los bugs elegidos debían ser aquellos de mayor severidad y que involucraran a todas las capas del sistema, es decir desde el backend hasta el frontend.

Los bugs elegido fueron los siguientes:

- Crear una invitación.
- Invitación con nombre de usuario único en el sistema.
- Comprar medicamentos de farmacias distintas.

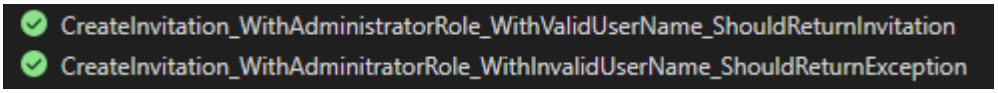
Id 3: Crear una invitación

Severidad: Media (entrega 1), Alta (entrega 2)
Responsable: Guzmán Dupont.
Descripción: No permite crear una nueva Invitación, se intenta agregar completando los tres campos: farmacia existente, un nombre de usuario único y el rol del invitado, pero cuando elegimos el rol del invitado como administrador y le damos al botón para crear la invitación lanza error de: "Farmacia no requerida", cuando sí es requerida al elegir el rol administrador dado que la letra no especifica lo contrario.
Cambio de severidad: Para la primera entrega consideramos que la severidad debería ser media, sin embargo, este error le ocasiona una limitante importante al usuario ya que no le permite la creación de otros usuario administradores al sistema.
Solución: Solucionamos el bug permitiendo la creación de una invitación con el nombre de una farmacia con un rol administrador, a su vez, denegamos la creación de una invitación sin farmacia para todo rol ingresado en la invitación. Esto se codificó en la carpeta *PharmaGo.BusinessLogic* dentro de la clase *InvitationManager*.
Código de casos de prueba:

```
[TestMethod]
[ExpectedException(typeof(InvalidResourceException))]
0 references
public void CreateInvitation_WithAdministratorRole_WithoutPharmacy_ShouldReturnException()
{
    var token = "Test";
    var user = new User() { Id = 1, Role = new Role() { Name = "Administrator" } };
    Invitation nullInvitation = null;
    Role adminRole = new Role() { Name = "Administrator" };
    int invitationID = 1;
    var newInvitation = new Invitation();
    Pharmacy pharmacy = null;
    var invitation = new Invitation()
    {
        Id = invitationID,
        UserName = "jcastro@test.com.uy",
        UserCode = "123456",
        Pharmacy = null,
        Role = new Role() { Name = "Administrator" },
        Created = DateTime.Now
    };

    _sessionMock.Setup(session => session.GetOneByExpression(It.IsAny<Expression<Func<Session, bool>>>()))
        .Returns(new Session());
    _userMock.Setup(user => user.GetOneDetailByExpression(It.IsAny<Expression<Func<User, bool>>>())).Returns(user);

    _roleMock.Setup(r => r.GetOneByExpression(It.IsAny<Expression<Func<Role, bool>>>())).Returns(adminRole);


    var result = _invitationManager.CreateInvitation(token, invitation);

}
```

```
[TestMethod]
0 references
public void CreateInvitation_WithAdministratorRole_WithPharmacy_ShouldReturnInvitation()
{
    var token = "Test";
    var user = new User() { Id = 1, Role = new Role() { Name = "Administrator" } };
    Invitation nullInvitation = null;
    Role adminRole = new Role() { Name = "Administrator" };
    int invitationID = 1;
    var newInvitation = new Invitation();
    Pharmacy pharmacy = new Pharmacy() { Id = 1, Name = "PharmaGo", Address = "Montevideo" };
    var invitation = new Invitation()
    {
        Id = invitationID,
        UserName = "jcastro@test.com.uy",
        UserCode = "123456",
        Pharmacy = new Pharmacy() { Id = 1, Name = "PharmaGo", Address = "Montevideo" },
        Role = new Role() { Name = "Administrator" },
        Created = DateTime.Now
    };

    _sessionMock.Setup(session => session.GetOneByExpression(It.IsAny<Expression<Func<Session, bool>>>()))
        .Returns(new Session());
    _userMock.Setup(user => user.GetOneByExpression(It.IsAny<Expression<Func<User, bool>>>())).Returns(new User() { UserName = null });
    _userMock.Setup(user => user.GetOneDetailByExpression(It.IsAny<Expression<Func<User, bool>>>())).Returns(user);

    _invitationMock.Setup(i => i.GetOneByExpression(It.IsAny<Expression<Func<Invitation, bool>>>())).Returns(newInvitation);

    _roleMock.Setup(r => r.GetOneByExpression(It.IsAny<Expression<Func<Role, bool>>>())).Returns(adminRole);

    _pharmacyMock.Setup(i => i.GetOneByExpression(It.IsAny<Expression<Func<Pharmacy, bool>>>())).Returns(pharmacy);

    _invitationMock.Setup(i => i.InsertOne(It.IsAny<Invitation>()));

    _invitationMock.Setup(i => i.Save());

    var result = _invitationManager.CreateInvitation(token, invitation);

    Assert.AreEqual(result.Pharmacy.Name, invitation.Pharmacy.Name);
    Assert.AreEqual(result.Id, invitationID);
}
```

Como se puede apreciar los tests quedaron en verde



Id 15: Invitación con nombre de usuario único en el sistema

Severidad: Alta
Responsable: Julieta Sarantes
Descripción: Como administrador quiero mandar una invitación validando que el nombre de usuario sea único en el sistema. No se está validando si existe un usuario con el nombre ingresado, solo si hay una invitación con ese nombre.
Solución: Para la solución buscamos primero donde debería hacerse el cambio, analizando el código detenidamente. Por lo tanto ingresamos a la carpeta *BusinessLogic*, dentro de esta a *PharmaGo.BusinessLogic* y por último a *InvitationManager*, aquí se logró visualizar que no se estaba contemplando el caso en el que ya existiera un usuario con ese nombre en alguna parte del sistema, sólo si ya estaba creada una invitación con este.
Entonces, nos movimos a *InvitationManagerTest* en donde aplicamos TDD para probar la nueva funcionalidad, luego de esto hicimos los cambios necesarios para que el test pasará de la fase RED a GREEN.

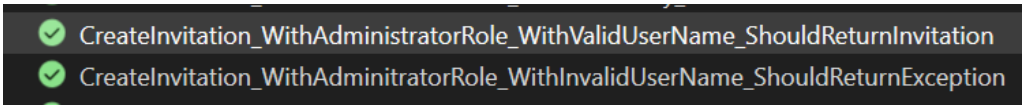
Lo que hicimos fue buscar si ya no existía un usuario con el mismo *UserName,* tanto en los usuarios creados como en los invitados, poniendo una condición de que si este usuario ya existía (user != null) tire una excepción.

Código de casos de prueba:

```
[TestMethod]
0 referencias
public void CreateInvitation_WithAdministratorRole_WithValidUserName_ShouldReturnInvitation()
{
    var token = "Test";
    var user = new User() { Id = 1,UserName = "Pablo5", Role = new Role() { Name = "Administrator" } };
    Invitation nullInvitation = null;
    User user2 = null;
    Role adminRole = new Role() { Name = "Administrator" };
    int invitationID = 1;
    var newInvitation = new Invitation();
    Pharmacy pharmacy = new Pharmacy() { Id = 1, Name = "PharmaGo", Address = "Montevideo" };
    var invitation = new Invitation()
    {
        Id = invitationID,
        UserName = "jcastro@test.com.uy",
        UserCode = "123456",
        Pharmacy = new Pharmacy() { Id = 1, Name = "PharmaGo", Address = "Montevideo" },
        Role = new Role() { Name = "Administrator" },
        Created = DateTime.Now
    };

    _sessionMock.Setup(session => session.GetOneByExpression(It.IsAny<Expression<Func<Session, bool>>>()))
        .Returns(new Session());
    _userMock.Setup(user => user.GetOneByExpression(It.IsAny<Expression<Func<User,bool>>>())).Returns(user2);
    _userMock.Setup(user => user.GetOneDetailByExpression(It.IsAny<Expression<Func<User, bool>>>())).Returns(user);

    _invitationMock.Setup(i => i.GetOneByExpression(It.IsAny<Expression<Func<Invitation, bool>>>())).Returns(newInvitation);

    _roleMock.Setup(r => r.GetOneByExpression(It.IsAny<Expression<Func<Role, bool>>>())).Returns(adminRole);

    _pharmacyMock.Setup(i => i.GetOneByExpression(It.IsAny<Expression<Func<Pharmacy, bool>>>())).Returns(pharmacy);

    _invitationMock.Setup(i => i.InsertOne(It.IsAny<Invitation>()));

    _invitationMock.Setup(i => i.Save());

    var result = _invitationManager.CreateInvitation(token, invitation);

    Assert.AreEqual(result.Pharmacy.Name, invitation.Pharmacy.Name);
    Assert.AreEqual(result.Id, invitationID);
}
```

```
[TestMethod]
[ExpectedException(typeof(InvalidResourceException))]
0 referencias
public void CreateInvitation_WithAdminitratorRole_WithInvalidUserName_ShouldReturnException()
{
    var token = "Test";
    var user = new User() { Id = 1, UserName = "Pablo1", Role = new Role() { Name = "Administrator" } };
    var invitation = new Invitation()
    {
        Id = 1,
        UserName = "Pablo1",
        Created = DateTime.Now
    };

    _sessionMock.Setup(session => session.GetOneByExpression(It.IsAny<Expression<Func<Session, bool>>>())).Returns(new Session());
    _userMock.Setup(user => user.GetOneDetailByExpression(It.IsAny<Expression<Func<User, bool>>>())).Returns(user);
    var result = _invitationManager.CreateInvitation(token, invitation);
}
```

Como se puede apreciar los tests quedaron en verde:

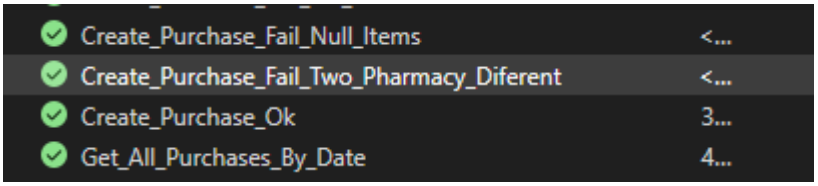


Id 16: Comprar medicamentos de farmacias distintas

Severidad: Alta
Responsable: Milena dos Santos Díaz
Descripción: Como usuario anónimo quiero poder realizar una compra de distintos medicamentos siempre y cuando estos pertenezcan a la misma farmacia, esta validación no estaba, ya que si intentaba realizar una compra con medicamentos de distintas farmacias.
Solución: Para la solución buscamos primero donde debería hacerse el cambio, analizando el código detenidamente.
Nos encontramos con que dentro de las validaciones no había ninguna que validará que no hayan dos farmacias distintas en la misma compra, por lo que con TDD realizamos el test, fallo ya que no estaba la validación, cuando la agregamos funcionó pero se rompieron tests anteriores ya que en los tests habían compras con distintas farmacias, por lo que al solucionar eso funcionaron correctamente todos los tests. Luego de que los tests estén en verde, pasamos a Refactorizar el código.

Código de casos de prueba:

```
[TestMethod]
[ExpectedException(typeof(InvalidResourceException))]
public void Create_Purchase_Fail_Two_Pharmacy_Diferent()
{
    //Arrange


    Drug drug4 = new Drug { Id = 4, Deleted = false, Code = "XL329", Name = "Loratadina", Prescription = false, Price = 100, Stock = 50, Quantity = 10, UnitMeasure
    Pharmacy pharmacy4 = new Pharmacy { Id = 4, Name = "Farmacia 4", Address = "Av. Italia 12345", Users = new List<User>(), Drugs = new List<Drug> { drug4 } };
    Drug drug3 = new Drug { Id = 3, Deleted = false, Code = "RS500", Name = "Perifar", Prescription = false, Price = 250, Stock = 50, Quantity = 20, UnitMeasure = u
    Pharmacy pharmacy3 = new Pharmacy { Id = 3, Name = "Farmacia 3", Address = "Av. Italia 3333", Users = new List<User>(), Drugs = new List<Drug> { drug3 } };
    PurchaseDetail purchaseDetail3 = new PurchaseDetail { Id = 1, Quantity = 2, Price = new decimal(100), Drug = drug3, Pharmacy = pharmacy3, Status = "Pending" };
    PurchaseDetail purchaseDetail4 = new PurchaseDetail { Id = 2, Quantity = 2, Price = new decimal(100), Drug = drug4, Pharmacy = pharmacy4, Status = "Pending" };
    //PurchaseDetail purchaseDetail2 =                                    = 2, Price = new decimal(100), Drug = drug2, Pharmacy = pharmacy2, Status = "Pending" }
    List<PurchaseDetail> purchaseDetai  PurchaseDetail.PurchaseDetail()
    purchaseDetailsFail.Add(purchaseDe
    purchaseDetailsFail.Add(purchaseDetail4);
    var purchaseFail = new Purchase
    {
        Id = 3,
        BuyerEmail = "roberto.perez@gmail.com",
        PurchaseDate = new DateTime(2022, 09, 19, 14, 34, 44),
        TotalAmount = new decimal(450),
        details = purchaseDetailsFail
    };

    //Act
    var response = _purchasesManager.CreatePurchase(purchaseFail);
}
```



```
public Purchase CreatePurchase(Purchase purchase)
{
    string validEmail = @"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$";
    Regex rgEmail = new(validEmail);
    if (string.IsNullOrEmpty(purchase.BuyerEmail) || !rgEmail.IsMatch(purchase.BuyerEmail))
        throw new InvalidResourceException("Invalid Email");

    if ((purchase.details == null || purchase.details.Count == 0))
        throw new InvalidResourceException("The list of items can't be empty");

    if (purchase.PurchaseDate == DateTime.MinValue)
        throw new InvalidResourceException("The purchase date is a mandatory field");

    bool differentPharmacy = DistintasFarmaciasEnMismaCompra(purchase.details);
    decimal total = 0;
    foreach (var detail in purchase.details)
    {
        int pharmacyId = detail.Pharmacy.Id;
        if (pharmacyId <= 0)
            throw new ResourceNotFoundException($"Pharmacy Id is a mandatory field");

        var pharmacy = _pharmacysRepository.GetOneByExpression(x => x.Id == pharmacyId);
        if (pharmacy is null)
            throw new ResourceNotFoundException($"Pharmacy {detail.Pharmacy.Id} not found");

        if (detail.Quantity <= 0)
            throw new InvalidResourceException("The Quantity is a mandatory field");

        string drugCode = detail.Drug.Code;
        var drug = pharmacy.Drugs.FirstOrDefault(x => x.Code == drugCode && x.Deleted == false);
        if (drug is null)
            throw new ResourceNotFoundException($"Drug {drugCode} not found in Pharmacy {pharmacy.Name}");

        detail.Pharmacy = pharmacy;
        total = total + (drug.Price * detail.Quantity);
        detail.Price = drug.Price;
        detail.Drug = drug;
        detail.Status = PENDING;
    }
    purchase.TotalAmount = total;
    purchase.TrackingCode = generateTrackingCode();
    _purchasesRepository.InsertOne(purchase);
    _purchasesRepository.Save();

    return purchase;
}
```

```
private bool DistintasFarmaciasEnMismaCompra(ICollection<PurchaseDetail> details)
{
    bool differentPharmacy = false;
    int firstPharmacyId = 0;
    firstPharmacyId = details.First().Pharmacy.Id;
    foreach (var detail in details)
    {
        if (detail.Pharmacy.Id != firstPharmacyId)
        {
            differentPharmacy = true;
        }
    }
    if (differentPharmacy)
    {
        throw new InvalidResourceException("There are products from different pharmacies in the purchase");
    }
    return differentPharmacy;
}
```