UNIVERSIDAD ORT
Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Obligatorio 1

Diseño de aplicaciones 2

Lucas Castro - Nº 218709
Ricardo Poladura - Nº 238052

Grupo: N5A

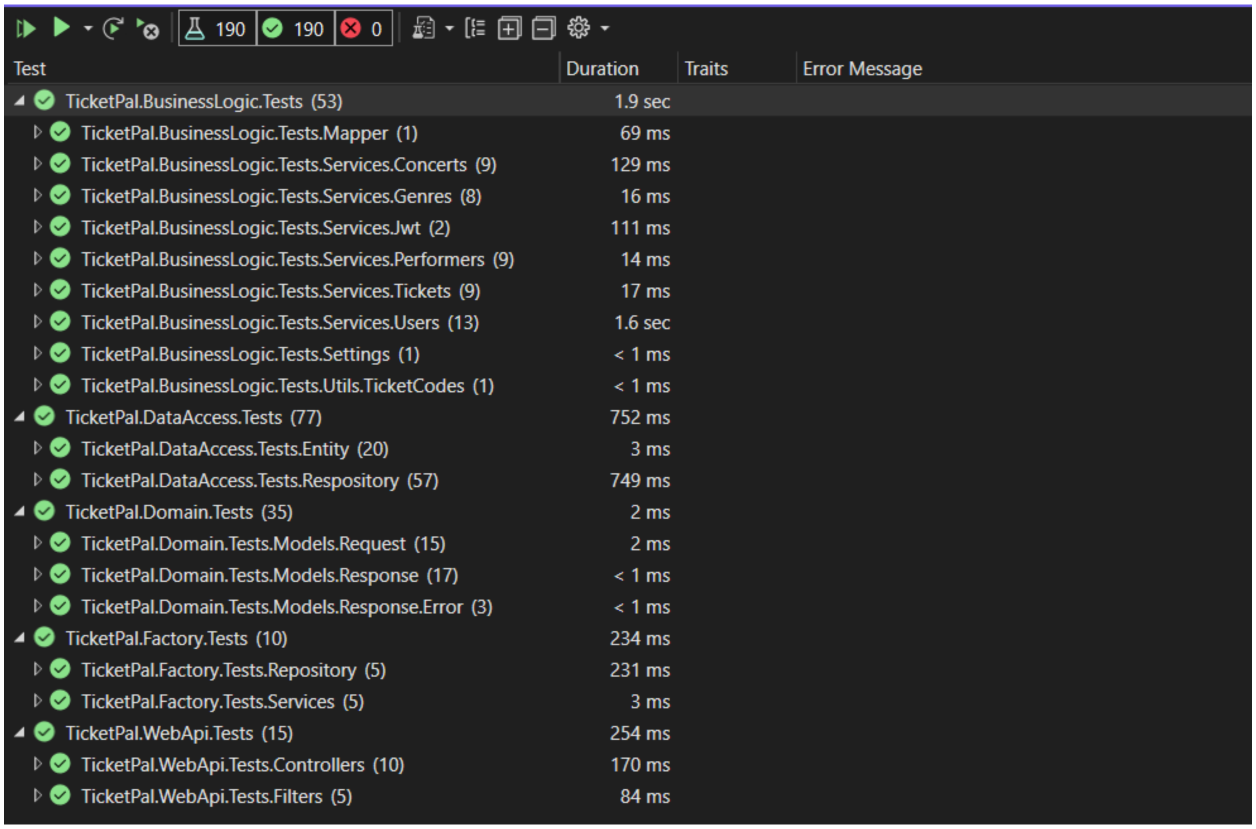
Docentes: Ignacio Valle – Nicolás Fierro – Nicolás Blanco

EVIDENCIA DE LA APLICACIÓN DE TDD Y CLEAN CODE

La estrategia utilizada para aplicar TDD en nuestro obligatorio fue la de outside-in, que consiste en escribir un test, con lo mínimo necesario, que falle, para luego escribir el código necesario para que el test pase, y luego refactorizar, volviendo a escribir otro test que falle, y así sucesivamente (“Red-Green-Refactor”). Y se van realizando las iteraciones necesarias de este bucle interno hasta conseguir pasar el test de aceptación.

Por regla general, casi toda la implementación del obligatorio se hizo aplicando TDD. Por un tema de tiempo, solamente la implementación de algunos Requests y Responses, se hizo sin aplicar TDD.

Como tuvimos algunos problemas con la implementación de algunos controladores, los refactor de los tests de dichas clases provocaron algunas fallas en otros tests que anteriormente estaban funcionando correctamente.



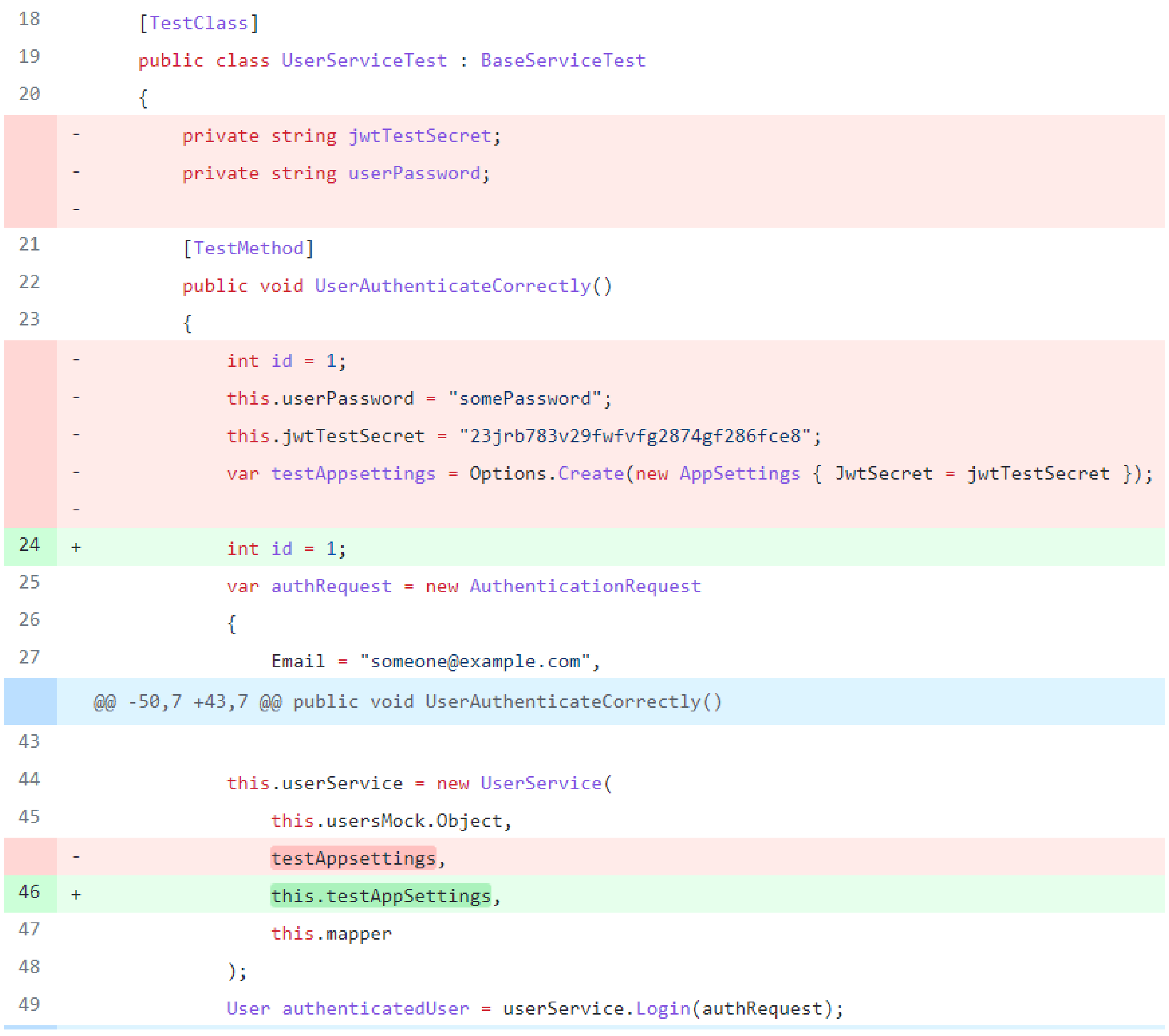
```
190   190   0
Test                                                   Duration   Traits   Error Message
TicketPal.BusinessLogic.Tests (53)                      1.9 sec
  TicketPal.BusinessLogic.Tests.Mapper (1)              69 ms
  TicketPal.BusinessLogic.Tests.Services.Concerts (9)   129 ms
  TicketPal.BusinessLogic.Tests.Services.Genres (8)     16 ms
  TicketPal.BusinessLogic.Tests.Services.Jwt (2)        111 ms
  TicketPal.BusinessLogic.Tests.Services.Performers (9) 14 ms
  TicketPal.BusinessLogic.Tests.Services.Tickets (9)    17 ms
  TicketPal.BusinessLogic.Tests.Services.Users (13)     1.6 sec
  TicketPal.BusinessLogic.Tests.Settings (1)            < 1 ms
  TicketPal.BusinessLogic.Tests.Utils.TicketCodes (1)   < 1 ms
TicketPal.DataAccess.Tests (77)                         752 ms
  TicketPal.DataAccess.Tests.Entity (20)                3 ms
  TicketPal.DataAccess.Tests.Respository (57)           749 ms
TicketPal.Domain.Tests (35)                             2 ms
  TicketPal.Domain.Tests.Models.Request (15)            2 ms
  TicketPal.Domain.Tests.Models.Response (17)           < 1 ms
  TicketPal.Domain.Tests.Models.Response.Error (3)      < 1 ms
TicketPal.Factory.Tests (10)                            234 ms
  TicketPal.Factory.Tests.Repository (5)                231 ms
  TicketPal.Factory.Tests.Services (5)                  3 ms
TicketPal.WebApi.Tests (15)                             254 ms
  TicketPal.WebApi.Tests.Controllers (10)               170 ms
  TicketPal.WebApi.Tests.Filters (5)                    84 ms
```

La cobertura de código no la pudimos medir, debido a que teníamos instalado el Visual Studio 2022 Community, que no cuenta con esa funcionalidad. Entendemos que debería ser alta, ya que la mayoría del código escrito se realizó mediante TDD.

Pruebas de aplicación de TDD y Clean Code

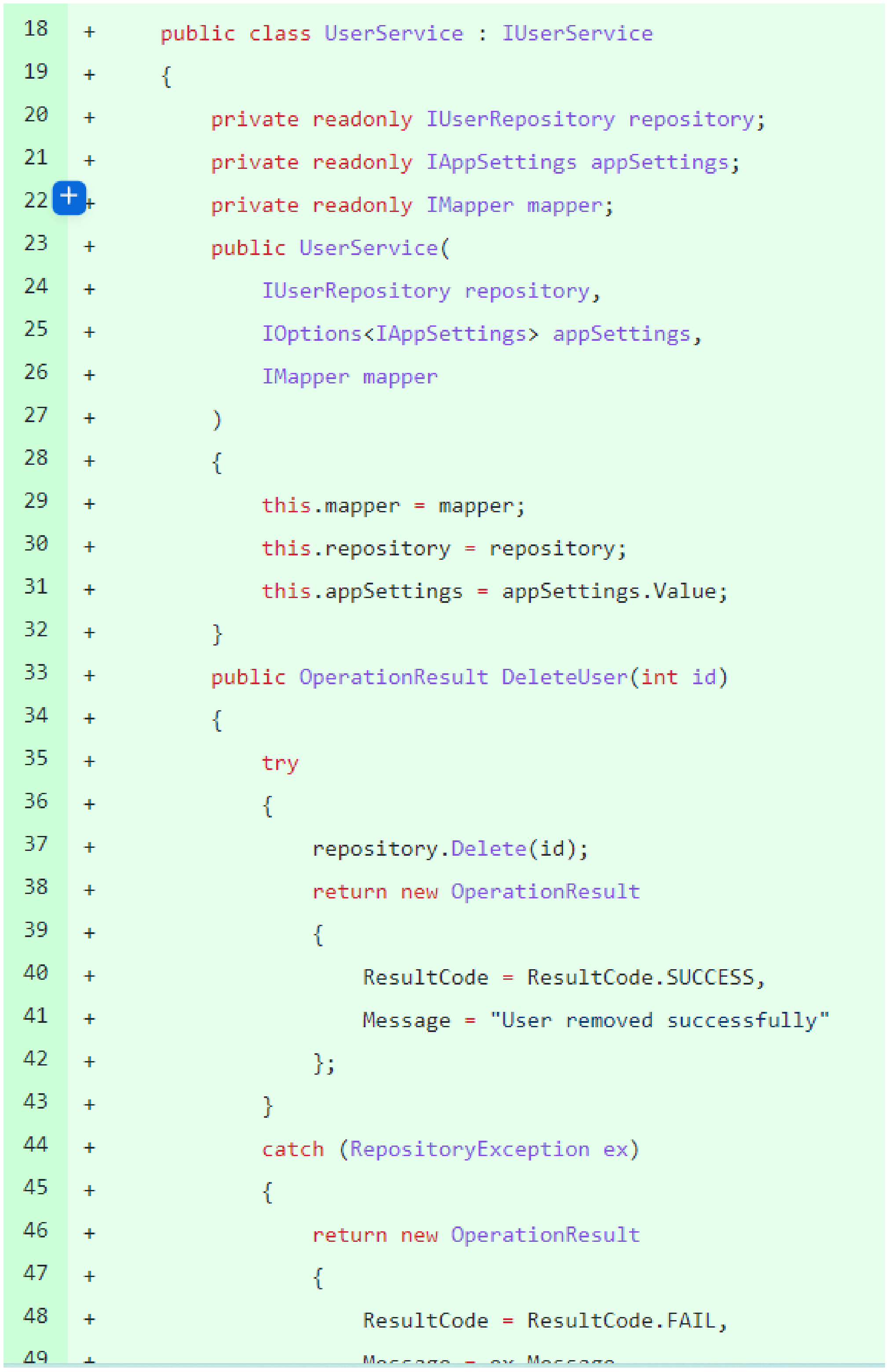
Ejemplos de aplicación de TDD y Clean code en la implementación del obligatorio:

Commit 570b1ee (Primero se terminan de implementar los tests de UserServiceTests):

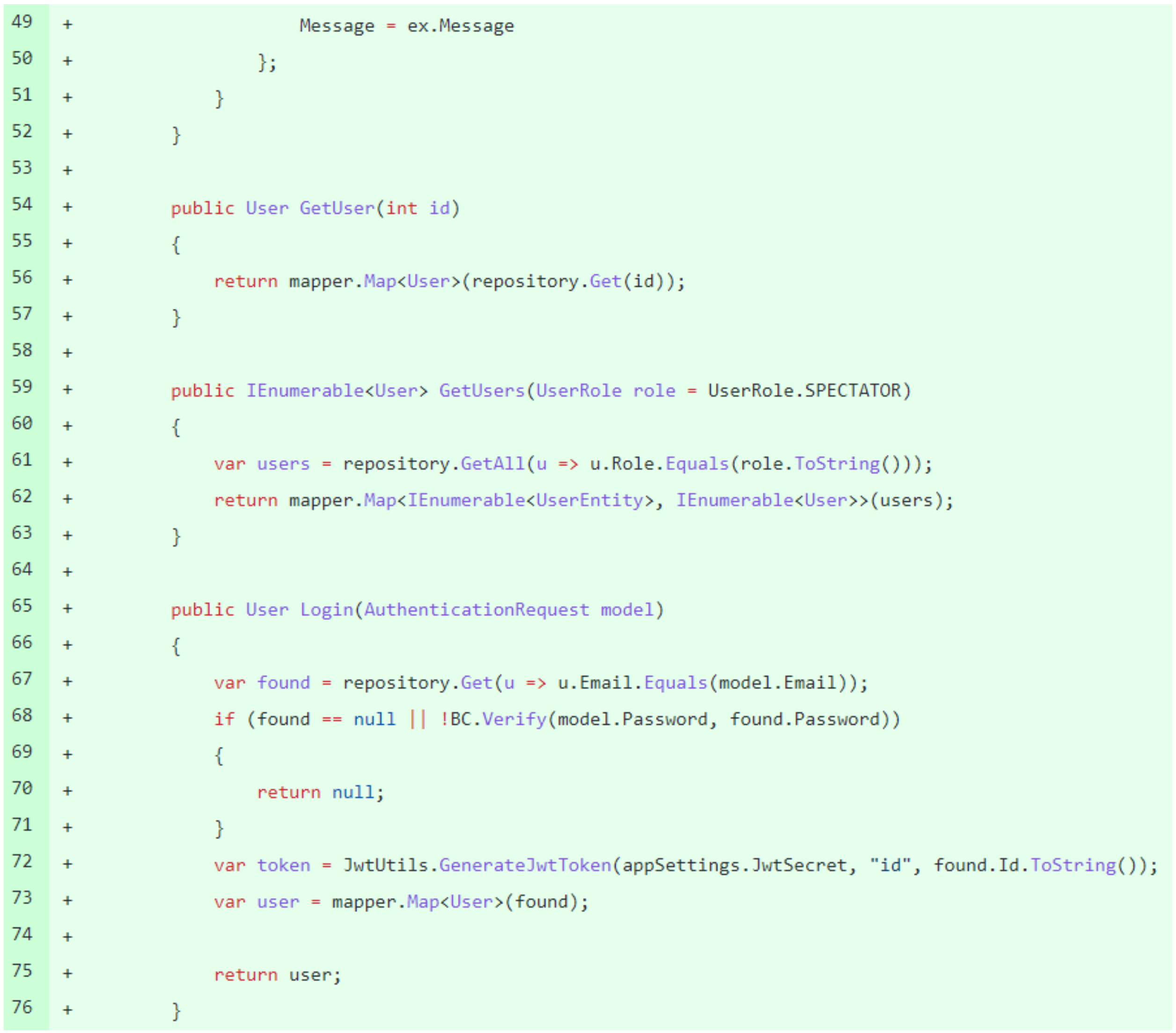


```diff
      [TestClass]
      public class UserServiceTest : BaseServiceTest
      {
-         private string jwtTestSecret;
-         private string userPassword;
-
          [TestMethod]
          public void UserAuthenticateCorrectly()
          {
-             int id = 1;
-             this.userPassword = "somePassword";
-             this.jwtTestSecret = "23jrb783v29fwfvfg2874gf286fce8";
-             var testAppsettings = Options.Create(new AppSettings { JwtSecret = jwtTestSecret });
-
+             int id = 1;
              var authRequest = new AuthenticationRequest
              {
                  Email = "someone@example.com",
@@ -50,7 +43,7 @@ public void UserAuthenticateCorrectly()

              this.userService = new UserService(
                  this.usersMock.Object,
-                 testAppsettings,
+                 this.testAppSettings,
                  this.mapper
              );
              User authenticatedUser = userService.Login(authRequest);
```

Commit e22c195 (Se implementa la clase UserService):



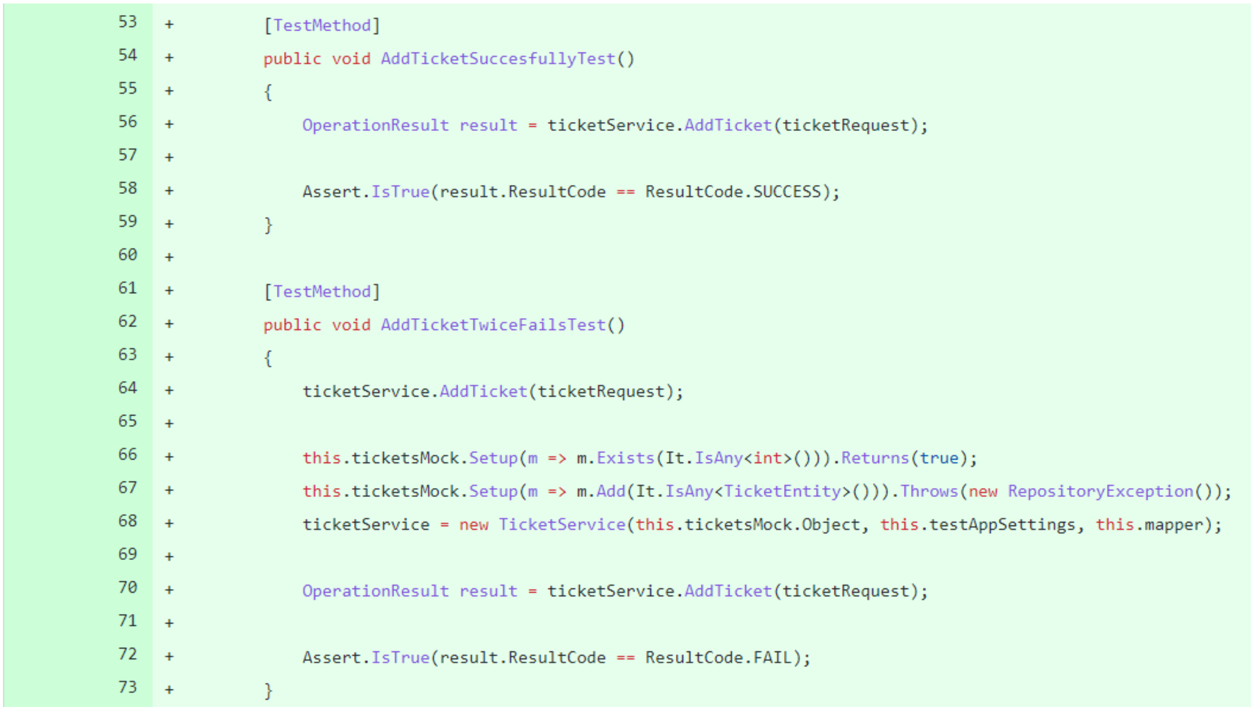
```
+    public class UserService : IUserService
+    {
+        private readonly IUserRepository repository;
+        private readonly IAppSettings appSettings;
+        private readonly IMapper mapper;
+        public UserService(
+            IUserRepository repository,
+            IOptions<IAppSettings> appSettings,
+            IMapper mapper
+        )
+        {
+            this.mapper = mapper;
+            this.repository = repository;
+            this.appSettings = appSettings.Value;
+        }
+        public OperationResult DeleteUser(int id)
+        {
+            try
+            {
+                repository.Delete(id);
+                return new OperationResult
+                {
+                    ResultCode = ResultCode.SUCCESS,
+                    Message = "User removed successfully"
+                };
+            }
+            catch (RepositoryException ex)
+            {
+                return new OperationResult
+                {
+                    ResultCode = ResultCode.FAIL,
+                    Message = ex.Message
```



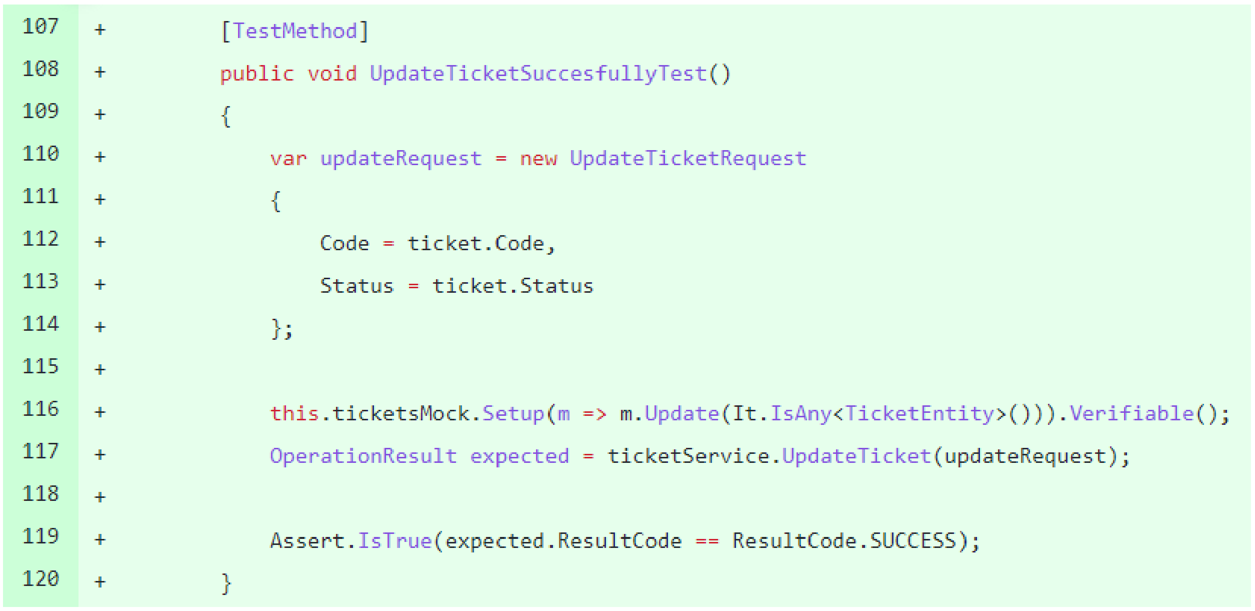
```
+                    Message = ex.Message
+                };
+            }
+        }
+
+        public User GetUser(int id)
+        {
+            return mapper.Map<User>(repository.Get(id));
+        }
+
+        public IEnumerable<User> GetUsers(UserRole role = UserRole.SPECTATOR)
+        {
+            var users = repository.GetAll(u => u.Role.Equals(role.ToString()));
+            return mapper.Map<IEnumerable<UserEntity>, IEnumerable<User>>(users);
+        }
+
+        public User Login(AuthenticationRequest model)
+        {
+            var found = repository.Get(u => u.Email.Equals(model.Email));
+            if (found == null || !BC.Verify(model.Password, found.Password))
+            {
+                return null;
+            }
+            var token = JwtUtils.GenerateJwtToken(appSettings.JwtSecret, "id", found.Id.ToString());
+            var user = mapper.Map<User>(found);
+
+            return user;
+        }
```

Commit 0a22225 (Se crearon las pruebas de la clase TicketServiceTests):

Clase TicketServiceTests:

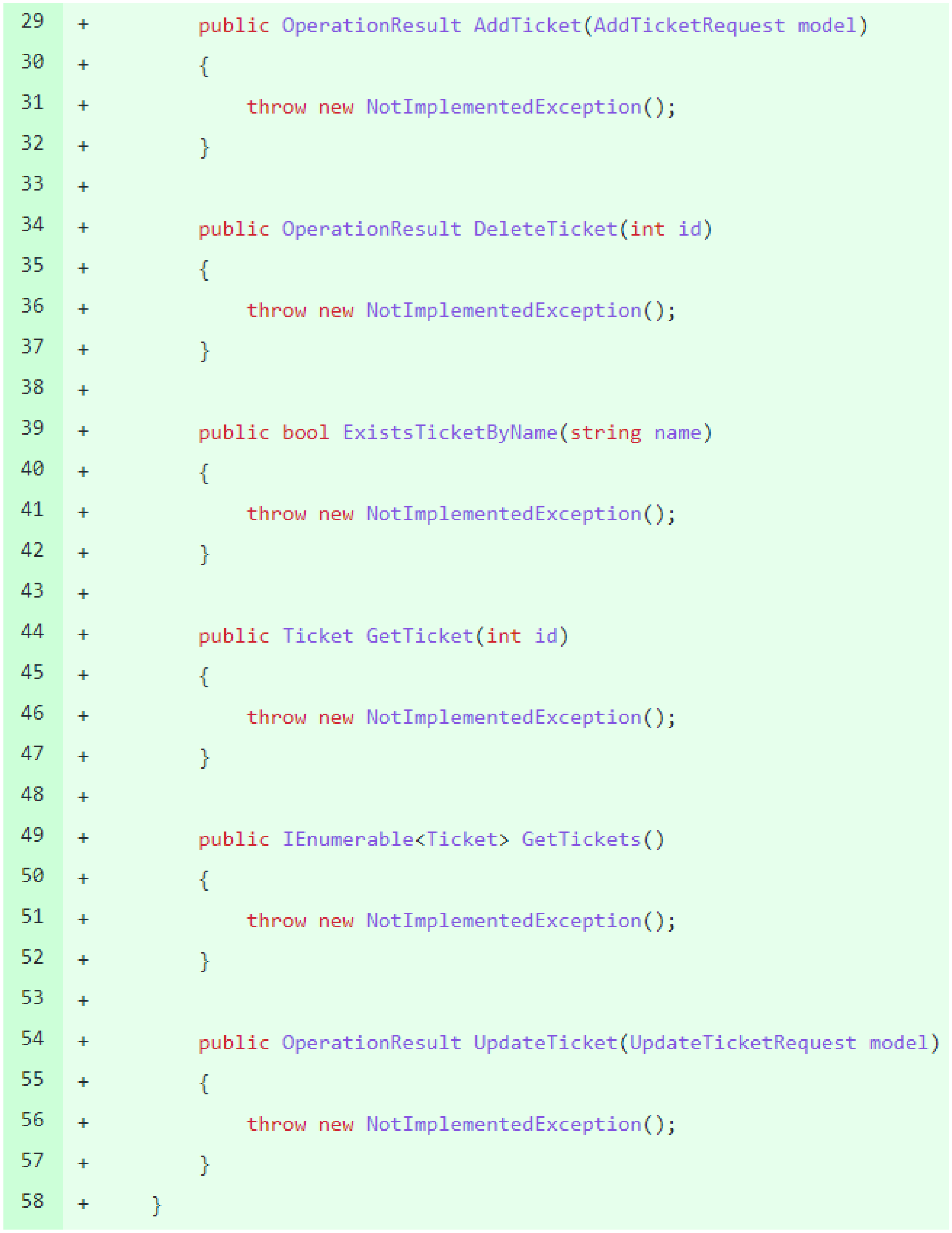


```
+        [TestMethod]
+        public void AddTicketSuccesfullyTest()
+        {
+            OperationResult result = ticketService.AddTicket(ticketRequest);
+
+            Assert.IsTrue(result.ResultCode == ResultCode.SUCCESS);
+        }
+
+        [TestMethod]
+        public void AddTicketTwiceFailsTest()
+        {
+            ticketService.AddTicket(ticketRequest);
+
+            this.ticketsMock.Setup(m => m.Exists(It.IsAny<int>())).Returns(true);
+            this.ticketsMock.Setup(m => m.Add(It.IsAny<TicketEntity>())).Throws(new RepositoryException());
+            ticketService = new TicketService(this.ticketsMock.Object, this.testAppSettings, this.mapper);
+
+            OperationResult result = ticketService.AddTicket(ticketRequest);
+
+            Assert.IsTrue(result.ResultCode == ResultCode.FAIL);
+        }
```



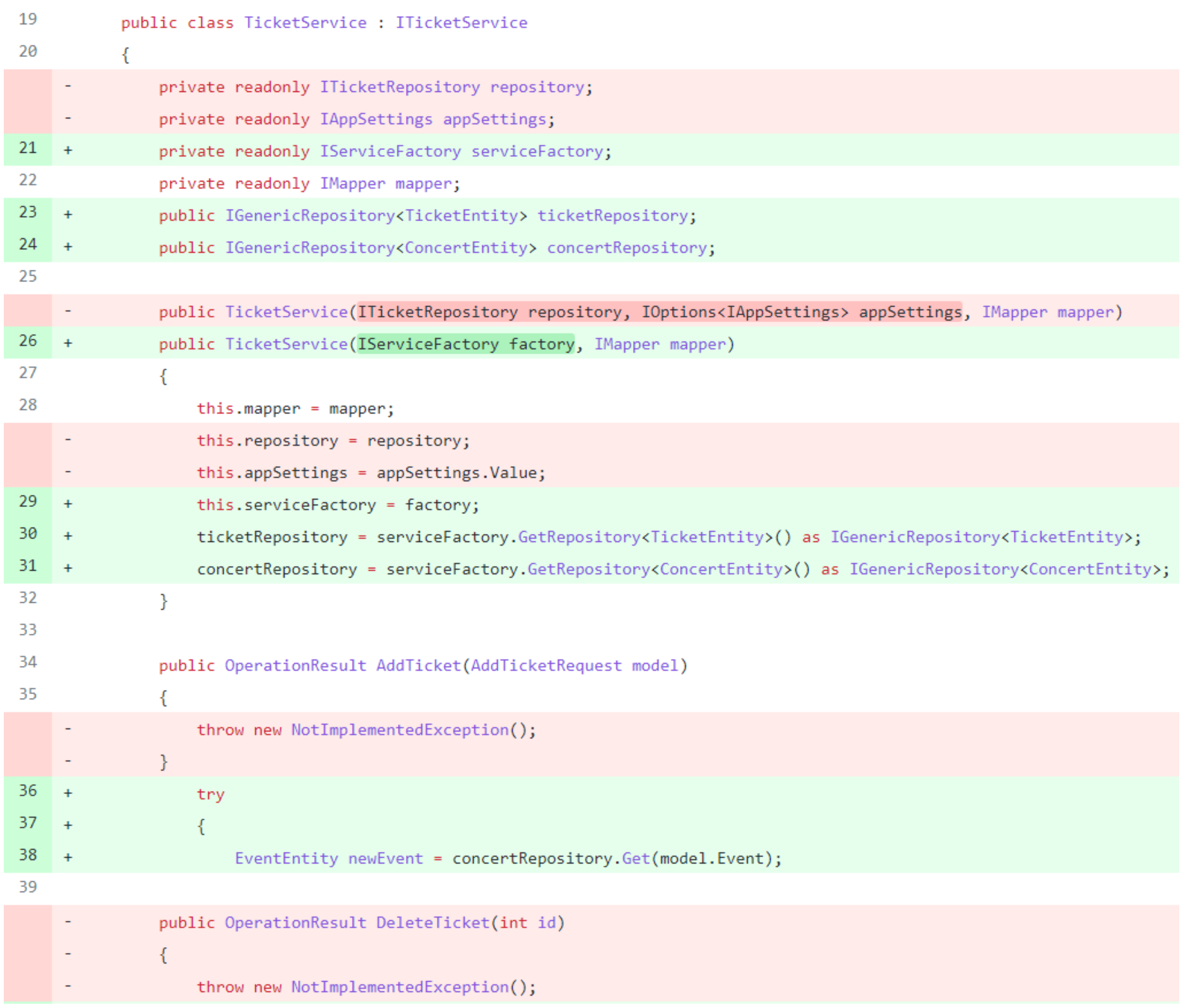
```
+        [TestMethod]
+        public void UpdateTicketSuccesfullyTest()
+        {
+            var updateRequest = new UpdateTicketRequest
+            {
+                Code = ticket.Code,
+                Status = ticket.Status
+            };
+
+            this.ticketsMock.Setup(m => m.Update(It.IsAny<TicketEntity>())).Verifiable();
+            OperationResult expected = ticketService.UpdateTicket(updateRequest);
+
+            Assert.IsTrue(expected.ResultCode == ResultCode.SUCCESS);
+        }
```

Clase TicketService:



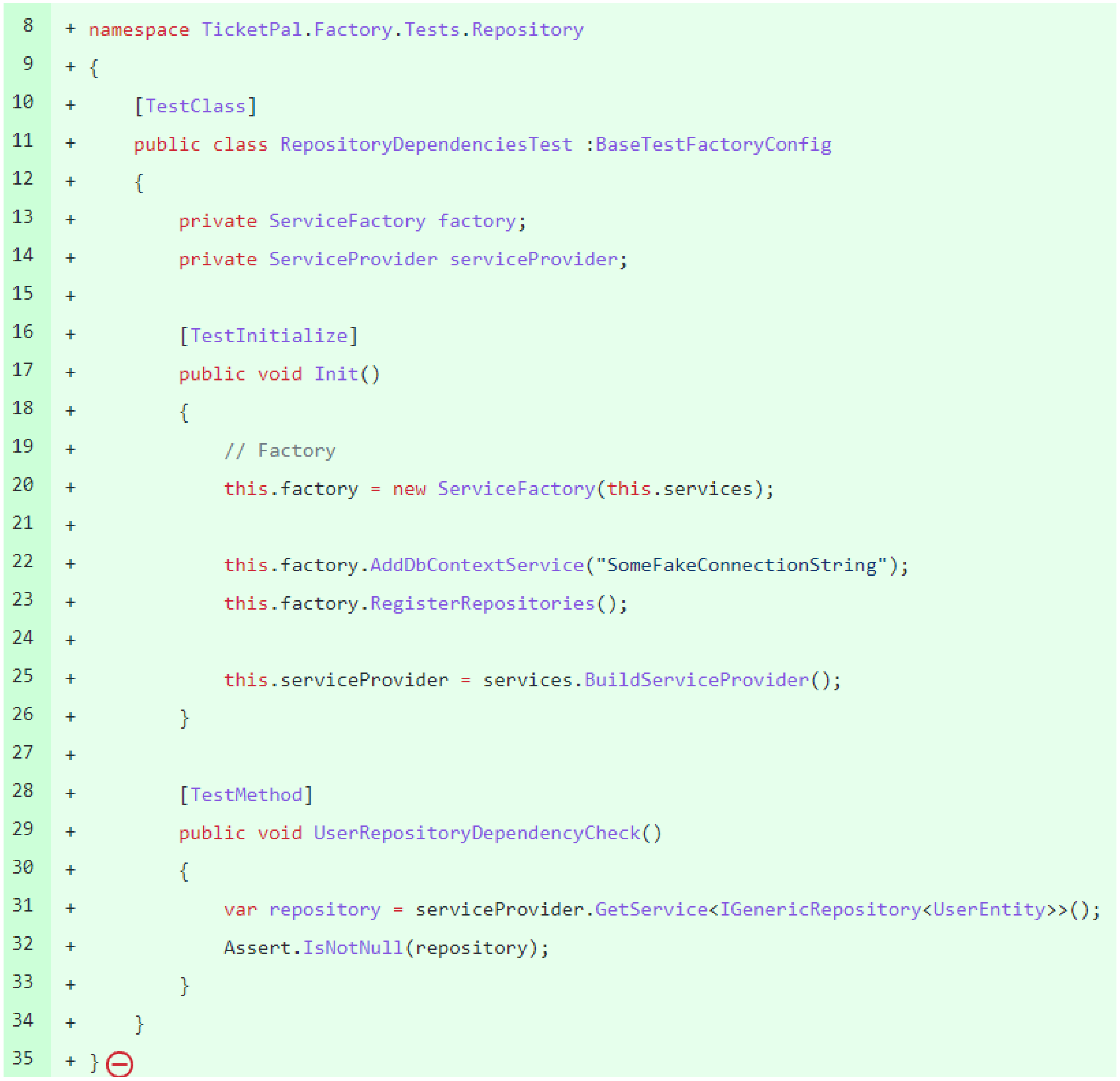
```
+        public OperationResult AddTicket(AddTicketRequest model)
+        {
+            throw new NotImplementedException();
+        }
+
+        public OperationResult DeleteTicket(int id)
+        {
+            throw new NotImplementedException();
+        }
+
+        public bool ExistsTicketByName(string name)
+        {
+            throw new NotImplementedException();
+        }
+
+        public Ticket GetTicket(int id)
+        {
+            throw new NotImplementedException();
+        }
+
+        public IEnumerable<Ticket> GetTickets()
+        {
+            throw new NotImplementedException();
+        }
+
+        public OperationResult UpdateTicket(UpdateTicketRequest model)
+        {
+            throw new NotImplementedException();
+        }
+    }
```

Commit 8fa6670 (Se implementa la clase TicketService):



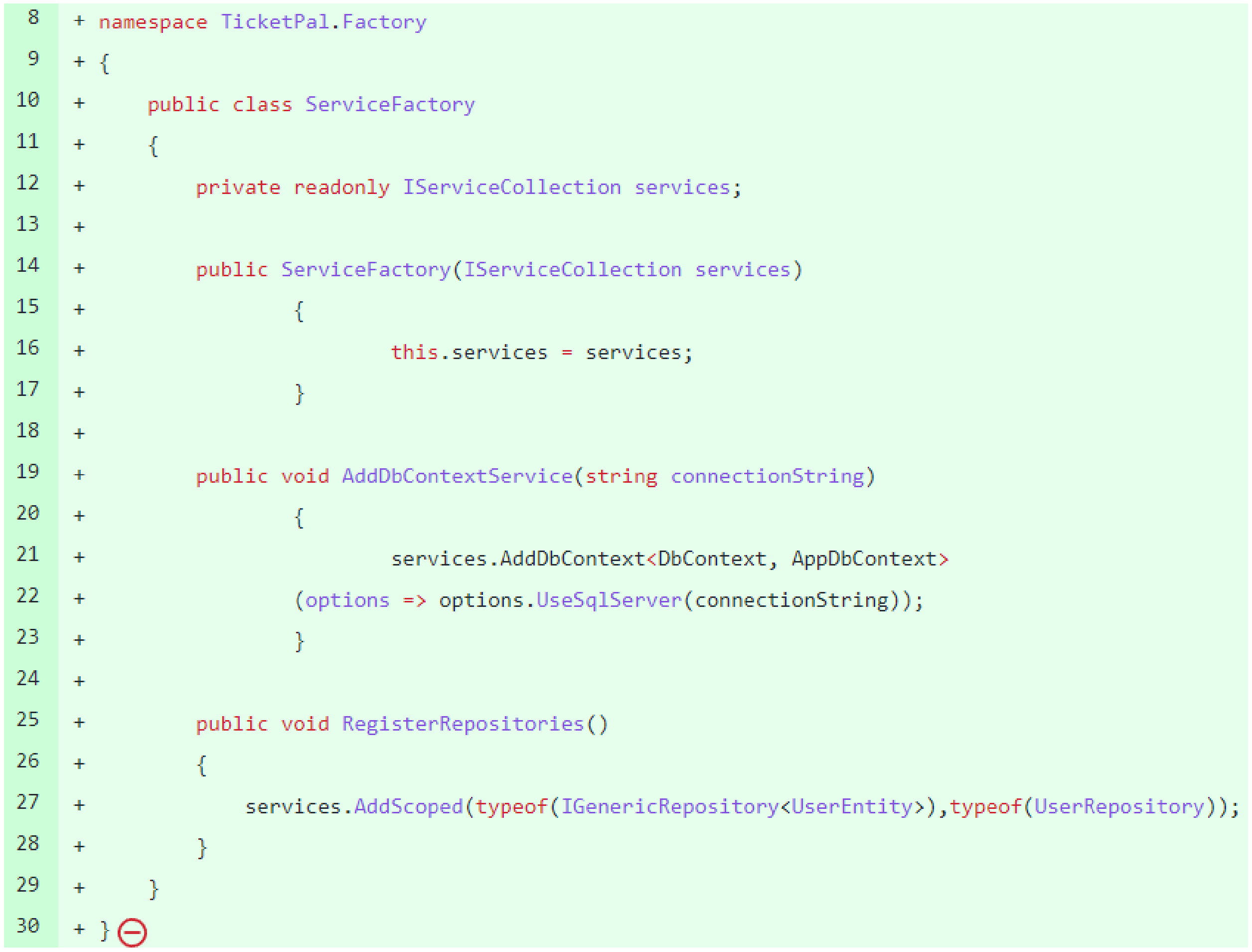
```
    public class TicketService : ITicketService
    {
-       private readonly ITicketRepository repository;
-       private readonly IAppSettings appSettings;
+       private readonly IServiceFactory serviceFactory;
        private readonly IMapper mapper;
+       public IGenericRepository<TicketEntity> ticketRepository;
+       public IGenericRepository<ConcertEntity> concertRepository;

-       public TicketService(ITicketRepository repository, IOptions<IAppSettings> appSettings, IMapper mapper)
+       public TicketService(IServiceFactory factory, IMapper mapper)
        {
            this.mapper = mapper;
-           this.repository = repository;
-           this.appSettings = appSettings.Value;
+           this.serviceFactory = factory;
+           ticketRepository = serviceFactory.GetRepository<TicketEntity>() as IGenericRepository<TicketEntity>;
+           concertRepository = serviceFactory.GetRepository<ConcertEntity>() as IGenericRepository<ConcertEntity>;
        }

        public OperationResult AddTicket(AddTicketRequest model)
        {
-           throw new NotImplementedException();
-       }
+           try
+           {
+               EventEntity newEvent = concertRepository.Get(model.Event);

-       public OperationResult DeleteTicket(int id)
-       {
-           throw new NotImplementedException();
```

Commit 14563ee (Se crean tests para utilización de ServiceFactory):



```
+ namespace TicketPal.Factory.Tests.Repository
+ {
+     [TestClass]
+     public class RepositoryDependenciesTest :BaseTestFactoryConfig
+     {
+         private ServiceFactory factory;
+         private ServiceProvider serviceProvider;
+
+         [TestInitialize]
+         public void Init()
+         {
+             // Factory
+             this.factory = new ServiceFactory(this.services);
+
+             this.factory.AddDbContextService("SomeFakeConnectionString");
+             this.factory.RegisterRepositories();
+
+             this.serviceProvider = services.BuildServiceProvider();
+         }
+
+         [TestMethod]
+         public void UserRepositoryDependencyCheck()
+         {
+             var repository = serviceProvider.GetService<IGenericRepository<UserEntity>>();
+             Assert.IsNotNull(repository);
+         }
+     }
+ }
```

Commit 129884f (Se crea clase ServiceFactory):



```
+ namespace TicketPal.Factory
+ {
+     public class ServiceFactory
+     {
+         private readonly IServiceCollection services;
+
+         public ServiceFactory(IServiceCollection services)
+                 {
+                         this.services = services;
+                 }
+
+         public void AddDbContextService(string connectionString)
+                 {
+                         services.AddDbContext<DbContext, AppDbContext>
+                 (options => options.UseSqlServer(connectionString));
+                 }
+
+         public void RegisterRepositories()
+         {
+             services.AddScoped(typeof(IGenericRepository<UserEntity>),typeof(UserRepository));
+         }
+     }
+ }
```