Taller de Lenguajes II – 2025
Programador Universitario / Licenciatura en informática / Ingeniería en Informática
TP Nro 10

CONTENIDOS

- Inyección de dependencias
- Autenticación y Autorización

Introducción

El objetivo es migrar el sistema de Presupuestos a una arquitectura desacoplada mediante **Inyección de Dependencias (DI)** y construir un **Módulo de Autenticación y Autorización por Roles** funcional que use variables de **Sesión** para el control de acceso.

Seguiremos trabajando en el mismo repositorio del TP anterior,

Reglas de Acceso:

- **Administrador:** Acceso total (CRUD) a **Productos** y **Presupuestos**.
- **Cliente:** Acceso de **solo Lectura** (Index, Details) a **Presupuestos**.
- **No Logueado:** Redirigir a /Login/Index.
- **Error UX:** Un usuario *logueado* sin permiso debe ser redirigido a una acción **AccesoDenegado**.

Desarrollo Paso a Paso del TP Nro 10

Fase 1: Infraestructura y Desacoplamiento (DI y Base de Datos)

Esta fase define las abstracciones (interfaces), el modelo de datos para usuarios y configura el contenedor de Inyección de Dependencias.

1. Creación de Interfaces de Abstracción

Cree la carpeta Interfaces para definir las Interfaces necesarias

- **Interfaces de Repositorios Existentes:** Cree las interfaces **IProductoRepository.cs** e **IPresupuestoRepository.cs** con todos los métodos CRUD
  Nota: Modifique sus clases de repositorio (ProductoRepository, PresupuestoRepository) para que implementen explícitamente estas interfaces
- **Interfaz de Usuarios: Defina la interfaz IUserRepository.cs** con el método GetUser(string username, string password) (debe devolver el objeto Usuario completo).

```
C#

namespace MVC.Interfaces;

public interface IUserRepository
{
    // Retorna el objeto Usuario si las credenciales son válidas, sino null.
    Usuario GetUser(string username, string password);
}
```

- **Interfaz de Seguridad: Defina la interfaz IAuthenticationService.cs** con los métodos: Login(user, pass), Logout(), IsAuthenticated() y HasAccessLevel(rol).

```
C#

namespace MVC.Interfaces;

public interface IAuthenticationService
{
    bool Login(string username, string password);
    void Logout();
    bool IsAuthenticated();
    // Verifica si el usuario actual tiene el rol requerido (ej. "Administrador").
    bool HasAccessLevel(string requiredAccessLevel);
}
```

2. Creación de la Tabla de Usuarios en la Base de Datos

- **Acción:** En **SQLiteStudio** o su herramienta de DB, cree la tabla **Usuarios**.
- **Campos:**
  - Id (Numérico)
  - Nombre (Texto)
  - User (Texto)
  - Pass (Texto)
  - Rol (Texto)
- **Poblar:** Inserte registros de prueba, al menos uno con Rol = "Administrador" y uno con Rol = "Cliente".

3. Implementación del Modelo y Repositorio de Usuarios

- **Modelo:** Cree el Modelo de Dominio **Usuario.cs** (incluyendo el campo Rol).
- **Repositorio:** Cree la clase **UsuarioRepository.cs** que implemente **IUserRepository** y contenga la lógica para buscar al usuario en la tabla Usuarios por credenciales.

```
C#

namespace MVC.Repositorios;

public class UsuarioRepository : IUserRepository
{
    // Lógica para conectar con la DB y buscar por user/pass.
    public Usuario GetUser(string usuario, string contrsena)
    {
         Usuario user = null;

        //Consulta SQL que busca por Usuario Y Contrasena
        const string sql = @"
                    SELECT Id, Nombre, User, Pass, Rol
                    FROM Usuarios
                    WHERE User = @Usuario AND Pass = @Contrasena";

        using var conexion = new SqliteConnection(CadenaConexion);
        conexion.Open();

               using var comando = new SqliteCommand(sql, conexion);

        // Se usan parámetros para prevenir inyección SQL
```

```
                comando.Parameters.AddWithValue("@Usuario", usuario);
                comando.Parameters.AddWithValue("@Contrasena",
contrasena);

                using var reader = comando.ExecuteReader();

                if (reader.Read())
                {
                    // Si el lector encuentra una fila, el usuario
existe y las credenciales son correctas
                    user = new Usuario
                    {
                        Id = reader.GetInt32(0),
                        Nombre = reader.GetString(1),
                        User = reader.GetString(2),
                        Pass = reader.GetString(3),
                        Rol = reader.GetString(4)
                    };
                }
                return user;
    }
```

4. Configuración de Servicios en Program.cs

- **Usings:** Asegúrese de incluir los using correctos para sus Interfaces, Repositorios y Servicios.
- **Servicios de Sesión:**
  - Agregue: builder.Services.AddHttpContextAccessor();
  - Agregue: builder.Services.AddSession(...) (con las opciones recomendadas, como IdleTimeout).
- **Registro de DI (AddScoped):** Registre **TODAS** las interfaces y sus implementaciones:
  - builder.Services.AddScoped<IProductoRepository, ProductoRepository>();
  - builder.Services.AddScoped<IPresupuestoRepository, PresupuestoRepository>();
  - builder.Services.AddScoped<IUserRepository, UsuarioRepository>();
  - builder.Services.AddScoped<IAuthenticationService, AuthenticationService>();
- **Pipeline de Sesión:** Asegúrese de que el orden en la configuración de la aplicación sea correcto:
  - app.UseRouting();
  - **app.UseSession();**
  - app.UseAuthorization();

C#

```
// Asegúrese de agregar los using necesarios
using Microsoft.AspNetCore.Http;
using MVC.Interfaces;
using MVC.Repositorios;
using MVC.Services;

var builder = WebApplication.CreateBuilder(args);
```

```
// Servicios de Sesión y Acceso a Contexto (CLAVE para la autenticación)
builder.Services.AddHttpContextAccessor();
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(30);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

// Registro de la Inyección de Dependencia (TODOS AddScoped)
builder.Services.AddScoped<IProductoRepository, ProductoRepository>();
builder.Services.AddScoped<IPresupuestoRepository, PresupuestoRepository>();
builder.Services.AddScoped<IUserRepository, UsuarioRepository>();
builder.Services.AddScoped<IAuthenticationService, AuthenticationService>();

// ... (Otros servicios y configuración MVC) ...

var app = builder.Build();

// Configuración del Pipeline de Middleware
// El orden es CRUCIAL: UseSession debe ir ANTES de UseRouting/UseAuthorization
app.UseSession(); //  → Habilita el uso de la sesión

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();
app.UseAuthorization(); // Necesario si usa atributos, aunque aquí lo haremos manual

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

5. Inyección en Controladores Existentes

- **Refactorización:** Modifique los constructores de **TODOS** los controladores (PresupuestosController y ProductosController) para que reciban sus dependencias (todos los Repositorios y el nuevo servicio de autenticación) **exclusivamente por interfaces** como parámetros.

Fase 2: Módulo de Autenticación Central

Esta fase implementa la lógica de sesión y el controlador de acceso (Login).

6. Implementación de AuthenticationService

- **Clase Concreta:** Implemente **AuthenticationService.cs** e inyecte **IUserRepository** y **IHttpContextAccessor** en el constructor.
- **Lógica de Login:** En el método Login(), si el usuario es válido, devuelve true y carga todas las variables de sesion relacionadas al usuario logueado

- **Lógica de Chequeo:** Los métodos `IsAuthenticated()` y `HasAccessLevel(rol)` deben leer la variable de Sesión `"Rol"` a través del `IHttpContextAccessor`.

C#

```
using MVC.Interfaces;
namespace MVC.Services;
public class AuthenticationService : IAuthenticationService
{
    private readonly IUserRepository _userRepository;
    private readonly IHttpContextAccessor _httpContextAccessor;
    //private readonly HttpContext context;

     public AuthenticationService(IUserRepository userRepository,
IHttpContextAccessor httpContextAccessor)
    {
        _userRepository = userRepository;
        _httpContextAccessor = httpContextAccessor;
       // context = _httpContextAccessor.HttpContext;
    }
    public bool Login(string username, string password)
    {
        var context = _httpContextAccessor.HttpContext;
        var user = _userRepository.GetUser(username,password);
        if (user != null)
        {
            if (context == null)
            {
                  throw new InvalidOperationException("HttpContext
no está disponible.");
            }
            context.Session.SetString("IsAuthenticated", "true");
            context.Session.SetString("User", user.User);
            context.Session.SetString("UserNombre", user.Nombre);
            context.Session.SetString("Rol", user.Rol);
            //es el tipo de acceso/rol admin o cliente
            return true;
        }
        return false;
```

```
        }
        public void Logout()
        {
            var context = _httpContextAccessor.HttpContext;

            if (context == null)
            {
                    throw new InvalidOperationException("HttpContext no
    está disponible.");
            }
           /* context.Session.Remove("IsAuthenticated");
            context.Session.Remove("User");
            context.Session.Remove("UserNombre");
            context.Session.Remove("Rol");
            */
            context.Session.Clear();
        }
        public bool IsAuthenticated()
        {
            var context = _httpContextAccessor.HttpContext;
            if (context == null)
            {
                    throw new InvalidOperationException("HttpContext no
    está disponible.");
            }
               return context.Session.GetString("IsAuthenticated") ==
    "true";
        }
        public bool HasAccessLevel(string requiredAccessLevel)
        {
            var context = _httpContextAccessor.HttpContext;
            if (context == null)
            {
                    throw new InvalidOperationException("HttpContext no
    está disponible.");
            }
                        return   context.Session.GetString("Rol")   ==
    requiredAccessLevel;
        }
    }
```

7. Implementación del `LoginController` y Vistas

- **ViewModel:** Cree el **`LoginViewModel`** (con `Username` y `Password`).
- **Controlador:** Cree **`LoginController.cs`** e inyecte **`IAuthenticationService`**.
- **Acciones:**
  - Implemente `[HttpGet] Index()` para mostrar la vista de login.
  - Implemente `[HttpPost] Login(ViewModel)` que llama a `_authService.Login()`. Redirige a `/Home/Index` si es exitoso o muestra error.
  - Implemente `Logout()` que llama a `_authService.Logout()` y redirige a `/Login/Index`.

C#

```
// implementación del LoginController.cs
using Microsoft.AspNetCore.Mvc;
using MVC.Interfaces;
using MVC.ViewModels;

public class LoginController : Controller
{
    private readonly IAuthenticationService _authenticationService;

    public LoginController(IAuthenticationService authenticationService)
    {
        _authenticationService = authenticationService;
    }

    // [HttpGet] Muestra la vista de login
    public IActionResult Index()
    {
        // ... (Crear LoginViewModel)
        return View(new LoginViewModel());
    }

    // [HttpPost] Procesa el login
    [HttpPost]
    public IActionResult Login(LoginViewModel model)
    {
        if (string.IsNullOrEmpty(model.Username) || string.IsNullOrEmpty(model.Password))
        {
            model.ErrorMessage = "Debe ingresar usuario y contraseña.";
            return View("Index", model);
        }

        if (_authenticationService.Login(model.Username, model.Password))
        {
            return RedirectToAction("Index", "Home");
        }
```

```
            model.ErrorMessage = "Credenciales inválidas.";
            return View("Index", model);
        }

        // [HttpGet] Cierra sesión
        public IActionResult Logout()
        {
            _authenticationService.Logout();
            return RedirectToAction("Index");
        }
    }
```

Fase 3: Autorización por Rol y Control de Acceso

Esta fase aplica las reglas de negocio de seguridad en cada acción de los controladores de negocio.

8. Acción de Acceso Denegado (UX de Error)

- **Acción:** Agregue la acción **public IActionResult AccesoDenegado()** en el **PresupuestosController.cs**.
- **Vista:** Cree la vista **AccesoDenegado.cshtml** con un mensaje informativo para usuarios logueados que intentan acceder sin permiso. Haga lo mismo para ProductosControllers.cs (personalice el mje de error que quiera mostrar)

C# AccesoDenegado.cshtml de presupuestos

```
@model dynamic
@{
    ViewData["Title"] = "Acceso Denegado";
}

<div class="row">

    <div class="col-md-10 mx-auto text-center">

        <h1 class="text-danger">❌ Acceso Denegado (Error 403)</h1>

        <hr />
```

```
            <h2>Lo sentimos.</h2>

            <p>            Su rol de **Cliente** no tiene permisos
    para acceder a esta funcionalidad.
                Esta sección está reservada para usuarios con el rol
    de **Administrador**.
            </p>

            <div class="mt-4">

                <a asp-action="Index" class="btn btn-primary">Volver
    a Listado de Presupuestos</a>

                <a asp-controller="Login" asp-action="Logout"
    class="btn btn-secondary">Cerrar Sesión</a>

            </div>

        </div>
    </div>
```

9. Protección de ProductosController (Acceso solo para Administrador)

- Al inicio de **TODAS** las acciones de ProductosController (Index, Details, Create, Edit, Delete), aplique la siguiente lógica:
    1. Si **!_authService.IsAuthenticated()**, redirigir a /Login/Index.
    2. Si **!_authService.HasAccessLevel("Administrador")**, redirigir a **Productos/AccesoDenegado**.

```
C#

public class ProductosController : Controller
{
    private  IProductoRepository _repo; //= new ProductoRepository();
    private  IAuthenticationService _authService;
 public ProductosController(IProductoRepository prodRepo,
IAuthenticationService authService)
    {
        //_repo = new ProductoRepository();
        _repo =prodRepo;

        _authService=authService;
```

```
    }

    public IActionResult Index()
    {

       // Aplicamos el chequeo de seguridad
        var securityCheck = CheckAdminPermissions();
        if (securityCheck != null) return securityCheck;

        List<Producto> productos = _repo.GetAll();
        return View(productos);
    }

    private IActionResult CheckAdminPermissions()
    {
        // 1. No logueado? -> vuelve al login
        if (!_authService.IsAuthenticated())
        {
            return RedirectToAction("Index", "Login");
        }

        // 2. No es Administrador? -> Da Error
        if (!_authService.HasAccessLevel("Administrador"))
        {
                    // Llamamos a AccesoDenegado (llama a la vista
correspondiente de Productos)
            return RedirectToAction(nameof(AccesoDenegado));
        }
        return null; // Permiso concedido
    }
    public IActionResult AccesoDenegado()
{
    // El usuario está logueado, pero no tiene el rol suficiente.
    return View();
}
// resto del código con las correspondientes correcciones
```

Taller de Lenguajes II – 2025
Programador Universitario / Licenciatura en informática / Ingeniería en Informática
TP Nro 10

10. Protección de PresupuestosController (Habilitando permisos según rol)

- **Acciones de Modificación:** Al inicio de las acciones que cambian datos (Create, Edit, Delete, AddDetalle), aplique la lógica del punto 9 (solo **Administrador**).
- **Acciones de Lectura:** Al inicio de Index y Details, implemente el chequeo para ambos roles, redirigiendo a AccesoDenegado solo si no es ni Administrador ni Cliente:
    1. Si **!_authService.IsAuthenticated()**, redirigir a /Login/Index.
    2. Si **!(_authService.HasAccessLevel("Administrador") || _authService.HasAccessLevel("Cliente"))**, redirigir a **Presupuestos/AccesoDenegado**.

C#

```
 // ... (El resto de los using)
using MVC.Interfaces;
namespace MVC.Controllers;
public class PresupuestosController : Controller
{
        private  IPresupuestoRepository _repo; //= new
PresupuestoRepository();
        // Se necesita el repositorio de Productos para llenar los
Dropdowns
        private  IProductoRepository _productoRepo; //= new
ProductoRepository();
        private  IAuthenticationService _authService;
    public PresupuestosController(IPresupuestoRepository repo,
IProductoRepository prodRepo, IAuthenticationService authService)
        {
            _repo =repo;
            _productoRepo=prodRepo;
            _authService=authService;
        }

    //
------------------------------------------------------------------------
    // 1. LISTAR (INDEX)
    //
------------------------------------------------------------------------
    public IActionResult Index()
        {
            // Comprobación de si está logueado
        if (!_authService.IsAuthenticated())
        {
            return RedirectToAction("Index", "Login");
        }

        // Verifica Nivel de acceso que necesite validar
        if (_authService.HasAccessLevel("Administrador") ||
_authService.HasAccessLevel("Cliente") )
        {
            //si es es valido entra sino vuelve a login
        var presupuestos = _repo.GetAll();
        return View(presupuestos);
        }
        else
```

```
        {
            return RedirectToAction("Index", "Login");
        }


        }


 public IActionResult Create()
        {
         // Comprobación de si está logueado
           if (!_authService.IsAuthenticated())
        {

            return RedirectToAction("Index", "Login");
        }

         // Verifica Nivel de acceso
        if (!_authService.HasAccessLevel("Administrador"))
        {

            return RedirectToAction(nameof(AccesoDenegado));
        }
           // Se retorna un VM vacío para el formulario
            return View(new PresupuestoViewModel());
        }


   // ... (El resto de las acciones)
}
```