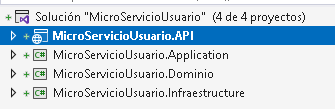
**Uso Plataforma de Desarrollo.**

**Creación de la estructura de un API utilizando la plataforma de desarrollo.**

Al finalizar el desarrollo de una PI con el objetivo de validar usuarios se tendría el proyecto de la siguiente forma:



Para demostrar el uso de la plataforma vamos a crear una solución que permita generar la validación de usuario o logueo (Api de seguridad).

Vamos a utilizar una arquitectura limpia con las siguientes capas:

* Infraestrutura
* Dominio
* Application
* API

1. **Crear solución MicroServicioUsuario**

Abrir Visual Studio 2022 Comunity y crear una solución en blanco con el siguiente nombre MicroServicioUsuario.

1. **Crear proyecto MicroServicioUsuario.Infraestructure**

Se agregar un nuevo proyecto de tipo librería con el nombre MicroServicioUsuario.Infraestructure este proyecto debe tener la versión .Net 5

1. **Crear proyecto MicroServicioUsuario.Dominio**

Se agregar un nuevo proyecto de tipo librería con el nombre MicroServicioUsuario.Dominio este proyecto debe tener la versión .Net 5

1. **Crear proyecto MicroServicioUsuario.Application**

Se agregar un nuevo proyecto de tipo librería con el nombre MicroServicioUsuario.Application este proyecto debe tener la versión .Net 5

1. **Crear proyecto MicroServicioUsuario.API**

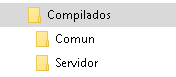
Se agregar un nuevo proyecto de tipo librería con el nombre MicroServicioUsuario.API este proyecto debe tener la versión .Net 5

1. **Descargar la plataforma de desarrollo.**

La plataforma de desarrollo debe estar en un repositorio git en el cual se debe descargar, este repositorio git debe estar bloqueado para permitir cambios y debe cumplir con una estrategia de ramas como git flow.

Al momento de descargar la plataforma se debe restaurar los paquetes nuget y compilar en su totalidad la solución. Por el momento cada dll se está generando en las carpetas /bin de cada proyecto.

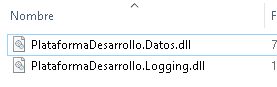
Se recomienda crear la siguiente estructura de carpetas fuera de la solución de la plataforma



En la carpeta Servidor se debe copiar las siguientes DLL:

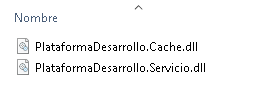
* PlataformaDesarrollo.Datos.dll
* PlataformaDesarrollo.Logging.dll

Quedando de la siguiente forma:



En la carpeta Comun se debe copiar las siguientes DLL:

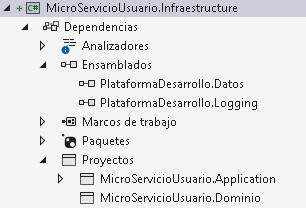
* PlataformaDesarrollo.Cache.dll
* PlataformaDesarrollo.Servicio.dll



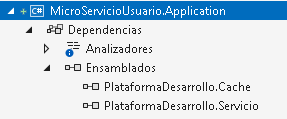
Con esta estructura se realizara las referencias hacia las DLL desde el API que se va a construir.

1. **Referencias de la plataforma de desarrollo.**

En el proyecto MicroServicioUsuario.Infraestructure se debe realizar la referencia hacia la DLL PlataformaDesarrollo.Datos.dll como se muestra en la imagen.



En el proyecto MicroServicioUsuario.Application se debe realizar las siguientes referencias:



1. **Implementación proyecto MicroServicioUsuario.Infraestructure**

Se implementa la clase UsuarioRepository

|  |
| --- |
| public class UsuarioRepository : IUsuarioRepository  {  private readonly SeguridadContext \_contexto;  private readonly IEjecutorDatos \_ejecutorDatos;  public UsuarioRepository(SeguridadContext contexto, IEjecutorDatos ejecutorDatos)  {  \_contexto = contexto;  \_ejecutorDatos = ejecutorDatos;  }  public async Task<Usuario> ObtenerUsuario(Usuario usuario)  {  \_ejecutorDatos.CadenaConexion = Constantes.CadenaConexionSeguridad;  \_ejecutorDatos.Modulo = Constantes.ModuloSeguridad;  ParametrosEjecucion parametrosEjecucion = new ParametrosEjecucion  {  NombreProcedimiento = Constantes.pConsultaUsuario,  DapperParametros = new[]{    new SqlParametrosDapper("@Username",System.Data.DbType.String, 20,usuario.Username),  new SqlParametrosDapper("@Contrasenia",System.Data.DbType.String, 20, usuario.Contrasenia)  }  };  Usuario resultado = await \_ejecutorDatos.ExecuteScalarAsync<Usuario>(parametrosEjecucion);  return resultado;  }  } |

1. **Implementación proyecto MicroServicioUsuario.Dominio**

Dentro de este proyecto se implementa las siguientes clases IUsuarioRepository

|  |
| --- |
| public interface IUsuarioRepository  {  public Task<Usuario> ObtenerUsuario(Usuario usuario);  } |

1. **Implementación proyecto MicroServicioUsuario.Application**

Se implementa la interfaz IUsuarioService.cs

|  |
| --- |
| public interface IUsuarioService  {  Task<Response<Usuario>> ObtenerUsuario(Usuario usuario);  } |

Se implementa la siguiente clase UsuarioService.cs

|  |
| --- |
| public class UsuarioService : IUsuarioService  {  private readonly ILogger \_logger;  private readonly IUsuarioRepository \_usuarioRepository;  public UsuarioService(ILogger logger, IUsuarioRepository usuarioRepository)  {  \_logger = logger;  \_usuarioRepository = usuarioRepository;  }  public async Task<Response<Usuario>> ObtenerUsuario(Usuario usuario)  {  try  {  Usuario usuarioRespuesta = await \_usuarioRepository.ObtenerUsuario(usuario);  return Response<Usuario>.Ok(usuarioRespuesta, Constantes.Constantes.MensajeSolicitudProcesoCorrectamente);  }  catch (Exception ex)  {  \_logger.Error(ex, "Error ObtenerUsuario");  return Response<Usuario>.Error(ex);  }  }  } |

1. **Implementación proyecto MicroServicioUsuario.Application**

Se implementa el siguiente controlador

|  |
| --- |
| [Route("api/usuario")]  [ApiController]  public class UsuarioController : ControllerBase  {  private readonly IUsuarioService usuarioService;  public UsuarioController(IUsuarioService usuarioService)  {  this.usuarioService = usuarioService;  }  [HttpPost]  [Route("ConsultarColaboradorActivo")]  public async Task<ActionResult> ObtenerUsuario(Usuario usuario)  {  Response<Usuario> usuarioRes = await usuarioService.ObtenerUsuario(usuario);  return StatusCode((int)usuarioRes.Status, usuarioRes);  }  } |

Para la configuración del sitio se debe realizar lo siguiente en Startup:

|  |
| --- |
| public class Startup  {  public Startup(IConfiguration configuration)  {  Configuration = configuration;  }  public IConfiguration Configuration { get; }  // This method gets called by the runtime. Use this method to add services to the container.  public void ConfigureServices(IServiceCollection services)  {  ContenedorComun.RegistrarServicios(services);  services.AddControllers();  services.AddSingleton(Log.Logger);  services.AddScoped<IUsuarioService, UsuarioService>();  services.AddScoped<IUsuarioRepository, UsuarioRepository>();  services.AddSwaggerGen(c =>  {  c.SwaggerDoc("v1", new OpenApiInfo { Title = "MicroServicioUsuario.API", Version = "v1" });  });  }  // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.  public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  {  if (env.IsDevelopment())  {  app.UseDeveloperExceptionPage();  app.UseSwagger();  app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "MicroServicioUsuario.API v1"));  }  app.UseRouting();  app.UseAuthorization();  app.UseEndpoints(endpoints =>  {  endpoints.MapControllers();  });  }  } |

Simulación, pruebas y evaluación