

Universidad ORT Uruguay  
Facultad de Ingeniería

# **Diseño de aplicaciones 2**

## **Obligatorio 2**

Santiago Larralde 217922  
Cristhian Maciel 163471

Entregado como requisito de la materia Diseño de Aplicaciones 2  
20 de junio de 2019

# Índice

<b>Reporte de defectos</b>	<b>3</b>
<b>2. Análisis de métricas</b>	<b>3</b>
2.1 Abstracción	4
2.2 Inestabilidad	4
2.3 Zona de dolor	5
<b>3. Endpoint expuestos</b>	<b>6</b>
<b>4. Clean Code y estándares</b>	<b>8</b>
<b>5. Cobertura de las pruebas.</b>	<b>9</b>
<b>6. Diagramas</b>	<b>9</b>
6.1 Diagrama de clases de Reports.Domain	9
6.2 Diagrama de paquetes	10
6.3 Diagrama de Componentes	11
6.4 Acceso a datos	12
6.5 Diagrama de deploy	13
<b>7. Manual de instalación</b>	<b>14</b>

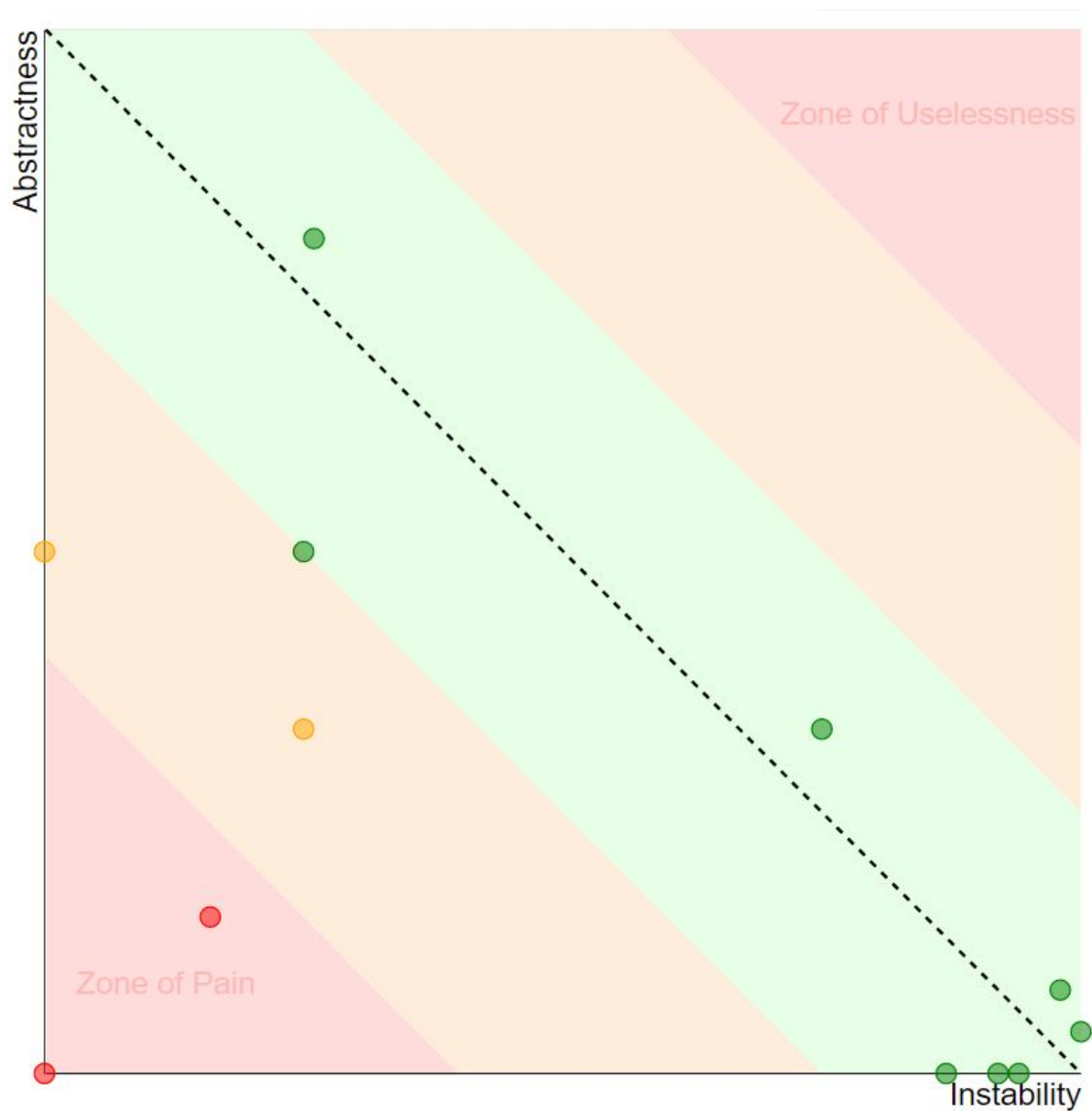
# 1. Reporte de defectos

Si bien la app permite iniciar sesión y/o registrarse, en ambos casos una vez realizada la acción es necesario recargar la pagina para que se actualice lo que el usuario ve.

En cuanto a funcionalidad la app permite realizar prácticamente todas las funcionalidades pedidas, salvo la del importador, la cual por temas de tiempo no fue implementada.

El manejo de errores no es el indicado desde el front, si una request falla por el motivo que sea no se le informa al usuario (desde el backend si son manejadas).

## 2. Análisis de métricas



Paquete	Distancia	Abstracción	Inestabilidad
Logger.Domain	0.71	0	0
Domain	0,49	0,15	0,16
DataAccess.Interface	0,35	0,5	0
Logger.Interface	0,29	0,33	0,25
DataAccess.Logger.interface	0,18	0,5	0,25
Logger	0,09	0	0,87
DbConnections	0,06	0,33	0,75
DataAccess.Logger	0,05	0	0,92
WebApi	0,03	0,04	1
DataAccess	0,04	0,08	0,98
BussinessLogic.Interface	0,04	0,8	0,26
BusinessLogic	0,04	0	0,94

## 2.1 Abstracción

La abstracción presenta valores entre 0 y 1. Siendo el 0 un paquete 100% concreto y 1 100% abstracto.

Los paquetes Domain, Logger y DataAccess.Logger y BussinessLogic no contienen ninguna clase abstracta ni interfaz, por lo que su valor es 0. El valor de Domain, la WebApi y DataAccess son muy cercano a 0.

## 2.2 Inestabilidad

El concepto de inestabilidad mide qué tan estables es un paquete, comparando dependencias entrantes con dependencias salientes.

Tenemos los casos extremos en WebApi ya que solo tiene dependencias salientes y por otro lado están Logger.Domain y DataAccess.Interface que solo tienen dependencias entrantes.

## 2.3 Zona de dolor

En la zona de dolor o zona roja tenemos dos paquetes el paquete de dominio, y el paquete de dominio de los logs. Estos son paquetes separados porque la letra tenía el requerimiento que el log fuera independiente, pero a nivel conceptual son lo mismo, las entidades que vamos a persistir. Son paquetes prácticamente 100% concretos y al ser clases de dominios hay muchas clases y paquete que dependen de ellos por lo que un cambio ahí repercute en todo el sistema, por eso es que son muy estables pero tiene un costo muy alto su modificación.

### 3. Endpoint expuestos

Controllers	Resource	Post	Get	Put	Delete
<b>UserController</b>	/users	Crea un nuevo usuario	Devuelve todos los usuarios	-	-
	/users/{id}	-	Devuelve el usuarios identificado con id	Modifica el usuario identificado con id	Elimina el usuario identificado con id
<b>AreaController</b>	/areas	Crea una nueva área	Devuelve todas las áreas	-	-
	/areas{id}	-	Devuelve el área identificad a con id	Modifica el ares identificad a con id	Elimina el área identificada con id
	/areas/{id}/Manager	Agrega un manager al area	Obtiene los manager de un area	-	Elimina los manager de un area
	/areas/{id}/Indicator	Agrega un indicador al area	Obtiene los indicadore s de un area	-	Elimina los indicadores de un area
<b>IndicatorController</b>	/indicators	Crea un nuevo indicador	Devuelve todos los indicadores	-	-
	/indicators{id}	-	Devuelve el indicador identificado con id	Modifica el indicador identificado con id	Elimina el indicador identificado con id
<b>LogsController</b>	/ManagerMoreLogger	-	Devuelve los	-	-

			managers más logueados		
	/IndicatorMoreHidden	-	Devuelve los indicadores más ocultos	-	-
	/ActionLogsByDate	-	Devuelve las acciones en el sistema en una rango de fecha	-	-
<b>CheckIndicatorController</b>	/checkIndicator	Verifica si el indicador es correcto	-	-	-
<b>ManagerController</b>	/id/Indicator	-	Devuelve los indicadores de un manager	Asocia al manager al indicador	-
	/id/IndicatorsSummary	Devuelve el reporte el usuario manager	-	-	-
<b>ProtectedController</b>	/CheckAdmin	Devuelve si el usuario es admin	-	-	-
	/CheckUser	Devuelve si el usuario es manager	-	-	-

TokenController	/Token	Logea al user y devuelve token	-	-	-
-----------------	--------	--------------------------------	---	---	---

## 4. Clean Code y estándares

A lo largo de toda la solución mantuvimos los estándares de codificación c# como por ejemplo que los métodos comienzan todos con mayúscula y utilizan CamelCase, las variables con minúscula y también con CamelCase.

Una de las prácticas más comunes de Clean Code son los nombres mnemotécnicos para las variables, clases y métodos, algo que utilizamos extensivamente en el desarrollo de nuestra app, tanto en el backend como en el frontend.

La app está escrita en inglés, por lo que todos los métodos y variables también lo están. Además, cada variable y nombre de método fueron escritos pensando que fueran lo más nemotécnico posible.

La UI también está en inglés para consistencia al trabajo.

Los métodos contruidos intentan tener una única responsabilidad

Otra práctica de Clean Code es pasar no más de tres parámetros a un método.

En la mayoría de los casos respetamos esto, pasando a lo sumo 2 parámetros por método. Las excepciones son los constructores que en algún caso recibe más de 2 parámetros y algunos métodos en controllers.

En general nuestros métodos no superan 20 líneas y las clases no superan las 200 líneas salvo las clases de test.

Para la mantención del repositorio en github mantuvimos un estándar interno de commit, el mismo cuenta en que para cada funcionalidad a desarrollar deberíamos hacer una rama nueva y su nombre sería *feature/funcionalidad*, una vez terminada la funcionalidad se hacía un merge con la rama develop.

Hay ramas que no comienzan con *feature/* y esas son las ramas que no hacen referencia a una funcionalidad en sí, o son refactor de otra rama.

Para crear una rama nueva, debíamos salir siempre desde develop estando esta actualizada para evitar conflictos a posteriori.

La rama Master contiene solo commit uno por obligatorio.

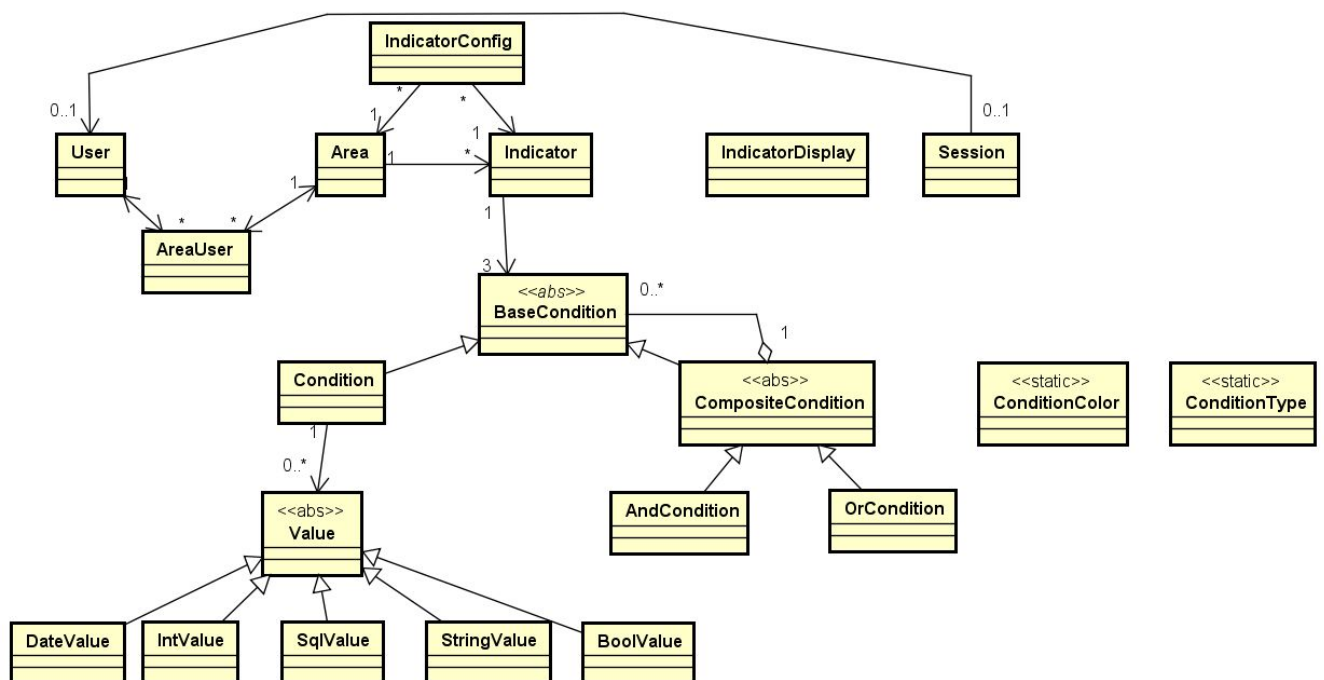


## 5. Cobertura de las pruebas.

Para esta segunda entrega no seguimos con la metodología TDD, la razón es que teníamos muchas funcionalidades atrasadas de la entrega anterior y dimos prioridad a desarrollar funcionalidades y no hacer testing unitario, más allá de eso, si realizamos pruebas con postman y con nuestro front de angular. Si bien somos conscientes que este tipo de pruebas no sustituye a las otras (no son automáticas, tienen costo de ejecución, etc) al menos nos sirvieron para testear nuestra app. Se realizaron también pruebas de postman, las cuales sirvieron como pruebas de integración y pruebas funcionales del software. Las cuales fueron corridas a mano, verificando en la base de datos y comprobando que estaban los resultados correctamente.

## 6. Diagramas

### 6.1 Diagrama de clases de Reports.Domain



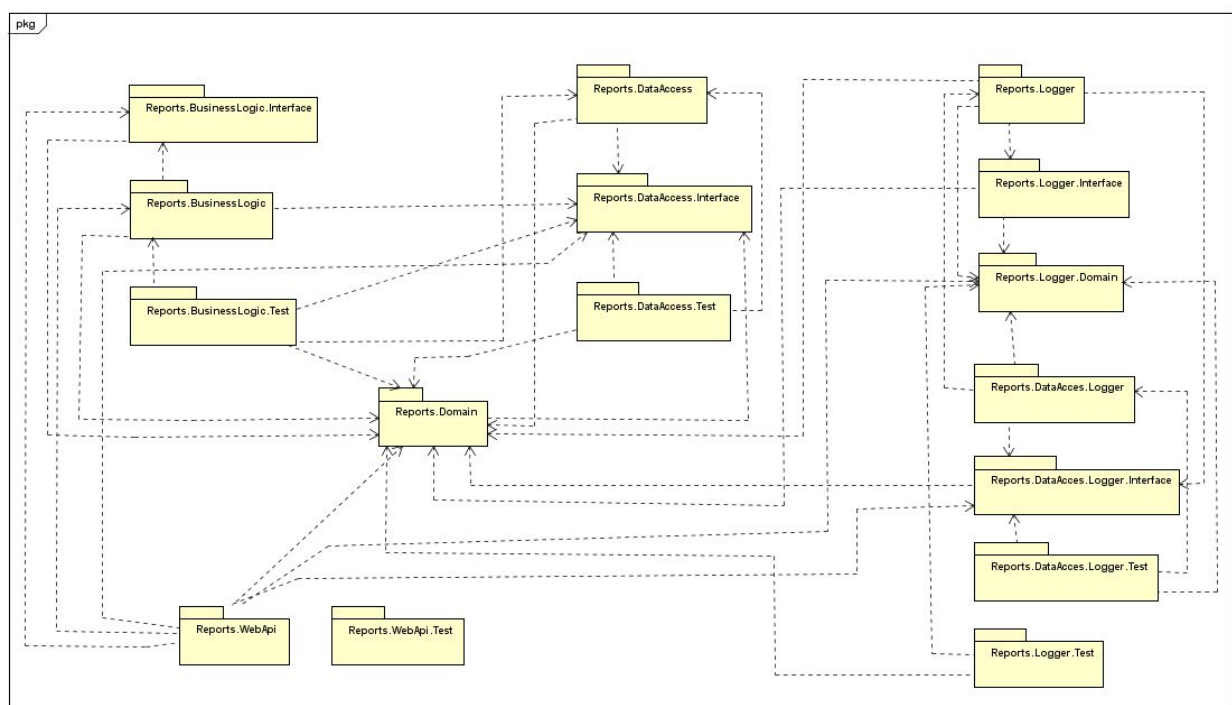
Este paquete es el encargado de contener el dominio del problema, el mismo no cuenta con lógica y solamente cuenta con la estructura de los datos a manejar en el sistema. Almacena las entidades que serán persistidas en la base de datos.

Esto implica que las responsabilidades de las clases son únicas, (mantener sus datos)

Como se observa en el diagrama el acoplamiento es bajo y en su mayoría se da a través de clases abstractas, el acoplamiento entre clases concretas en mínimo.

En el diagrama también se puede observar cómo fue que decidimos modelar los indicadores por intermedio del patrón composite el cual tiene como precondition que la estructura sea arborescente cómo en este caso podría ser el indicador con sus condiciones, si bien es similar a la solución de la primer entrega tiene algunas mejoras que en esta entrega nos permite poder manipular los indicadores en forma correcta.

## 6.2 Diagrama de paquetes



En este diagrama podemos ver cómo se relacionan los paquetes de nuestra solución, decidimos por cuestiones visuales realizar el diagrama más macro, con esto queremos decir que no colocamos las clases dentro de cada paquete ya que haría poco entendible el diagrama en sí, y cómo lo que queremos mostrar es cómo se relacionan los paquetes con esta vista alcanza.

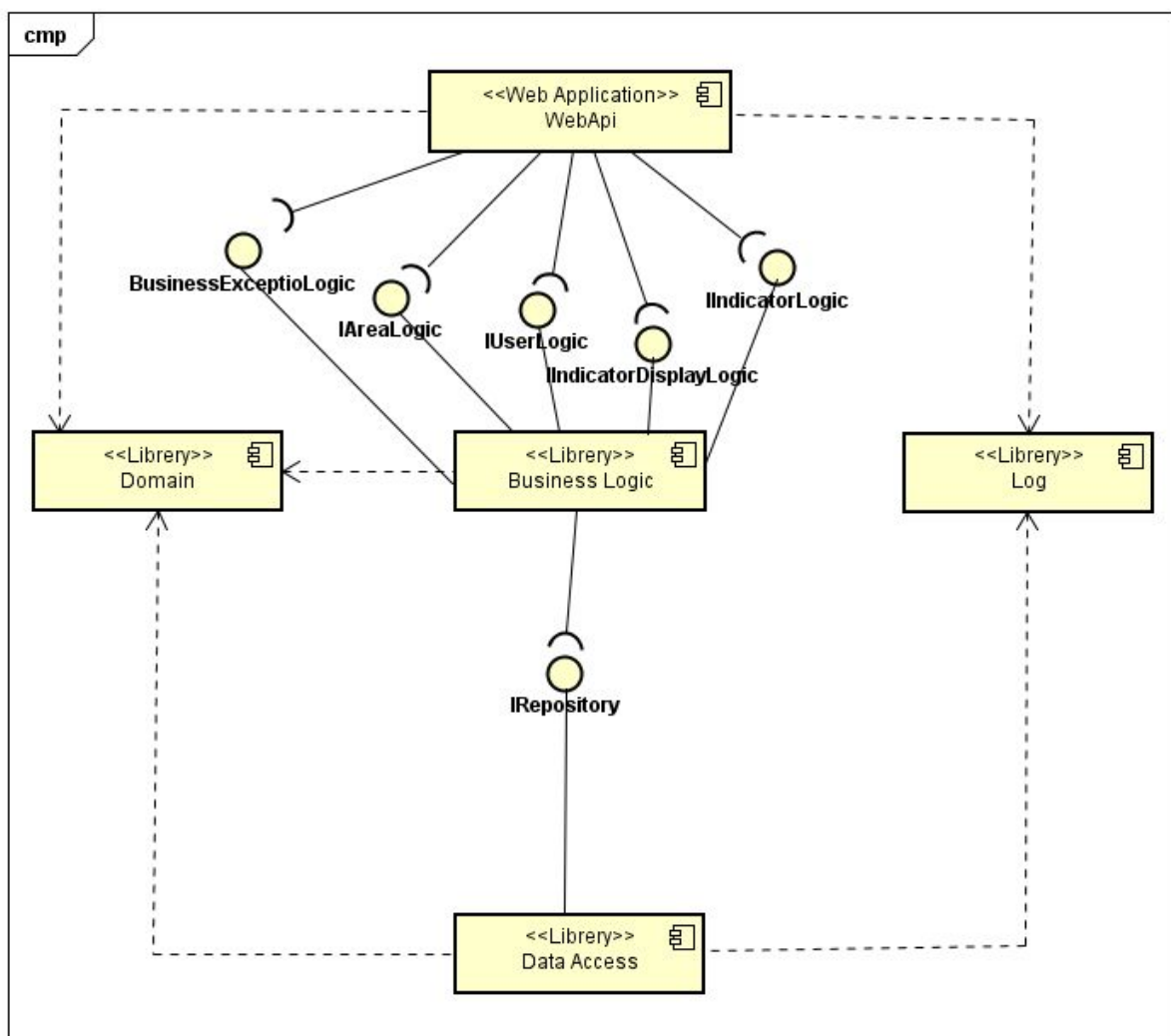
Aclarado esto pasemos a analizar el diagrama. En el mismo se puede visualizar las decisiones tomadas en cuanto a la organización de los paquetes, y sus dependencias en tiempo de compilación.

Si dividimos nuestro diagrama en alto y bajo nivel vemos que el alto nivel, (el dominio y las business logic) no dependen del bajo nivel, webapi y el acceso a datos. La business logic depende del dominio, alto dependiendo del alto, y ambos

dependen del acceso a datos pero por intermedio de interfaces (Reports.DataAccess.interface), es esta forma no violamos DIP (Dependency Inversion Principle).

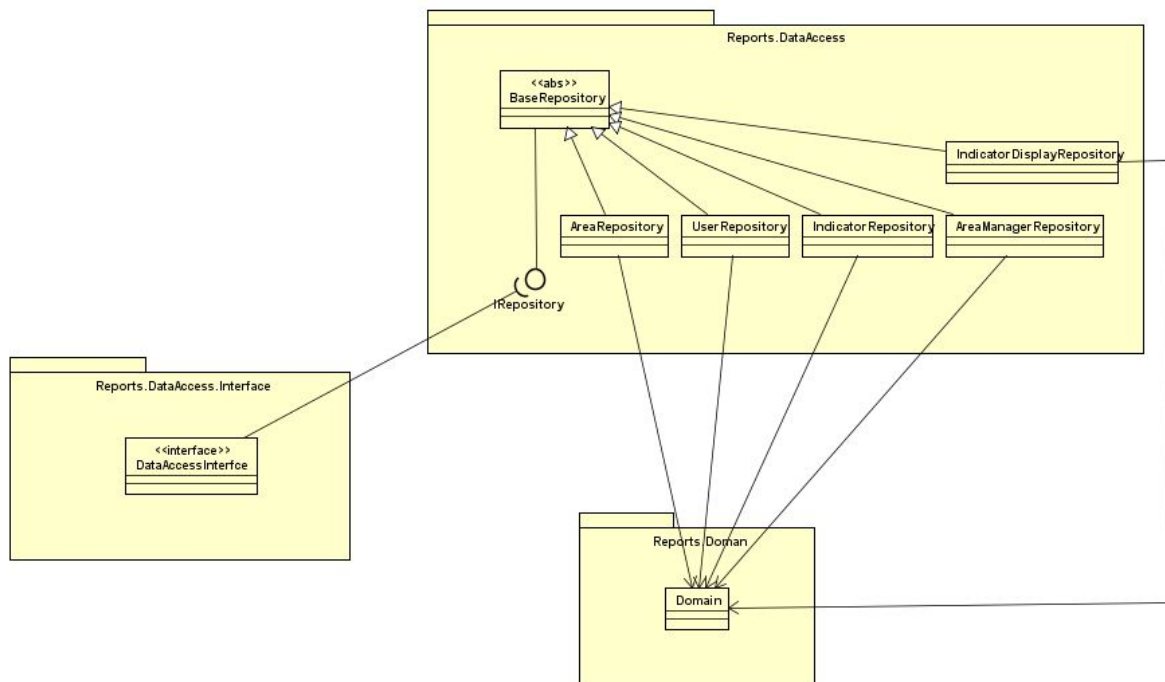
Para cumplir con el requerimiento de la letra para el log se crearon nuevos paquetes, estos son Reports.Logger.Interface en el cual se define la interfaz para trabajar con los logs. Reports.Logger acá está la clase que implementa la interfaz antes mencionada. Reports.Logger.Domain almacena la entidad del log en sí y por último Reports.Logger.Test para hacer pruebas unitarias.

### 6.3 Diagrama de Componentes



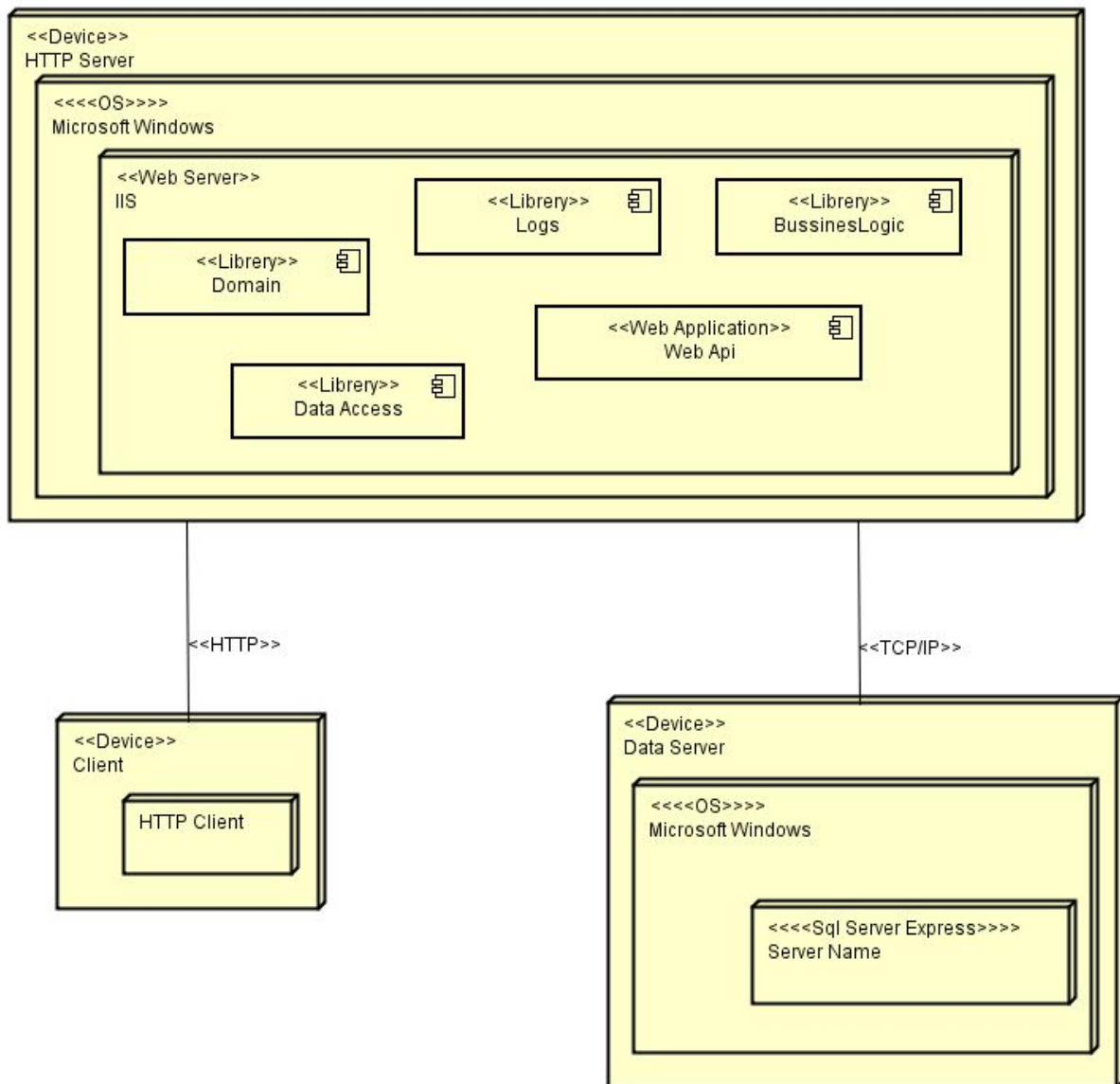
En este diagrama vemos como los componentes de nuestro sistema interactúan entre sí, nuestros componentes son la webapi, el dominio, la lógica de negocios y el acceso a datos, en este diagrama también puede verse el cumplimiento de DIP ya que el alto nivel se relaciona con el bajo a través de interfaces bien definidas.

## 6.4 Acceso a datos




Esto fue resuelto mediante el uso de dos paquetes un paquete que requiere una interfaz llamada **IRepository** genérico la cual define los métodos CRUD para acceso a base, y otro paquete que implementa esa interfaz. En el mismo se encuentra la clase **BaseRepository** que provee dicha interfaz la mismas es de alto nivel y genérica con lo que será extendida por otras clases que customizar los métodos que necesiten de acuerdo a la entidad.

## 6.5 Diagrama de deploy



## 7. Manual de instalación

1. Bajar zip del repositorio y copiarlo en disco D:
2. Iniciar servicio sql server
3. Conectarse a instancia \SQLEXPRESS con sql managment studio
4. Restaurar bases reports y area
5. Iniciar IIS
6. Copiar carpeta del proyecto (release/publish) en C:\inetpub\wwwroot
7. Agregar el sitio la ruta anterior, puerto 8080, nombre reports
8. En grupo de aplicaciones/reports/seleccionar “sin codigo administrado”

 Grupos de aplicaciones

Esta página permite ver y administrar la lista de grupos de aplicaciones del servidor. Los grupos de aplicaciones están asociados a aislamiento entre aplicaciones.

Filtro:	Ir	Mostrar todo	Agrupar por:	Sin agrupar	
Nombre	Estado	Versión de ...	Modo de canal...	Identidad	Aplicaciones
.NET v2.0	Iniciado	v2.0	Integrada	ApplicationPoolId...	0
.NET v2.0 Classic	Iniciado	v2.0	Clásica	ApplicationPoolId...	0
.NET v4.5	Iniciado	v4.0	Integrada	ApplicationPoolId...	0
.NET v4.5 Classic	Iniciado	v4.0	Clásica	ApplicationPoolId...	0
Classic .NET AppPool	Iniciado	v2.0	Clásica	ApplicationPoolId...	0
DefaultAppPool	Iniciado	v4.0	Integrada	ApplicationPoolId...	0
reports	Iniciado	v4.0	Integrada	ApplicationPoolId...	0

Modificar grupo de aplicaciones ? X

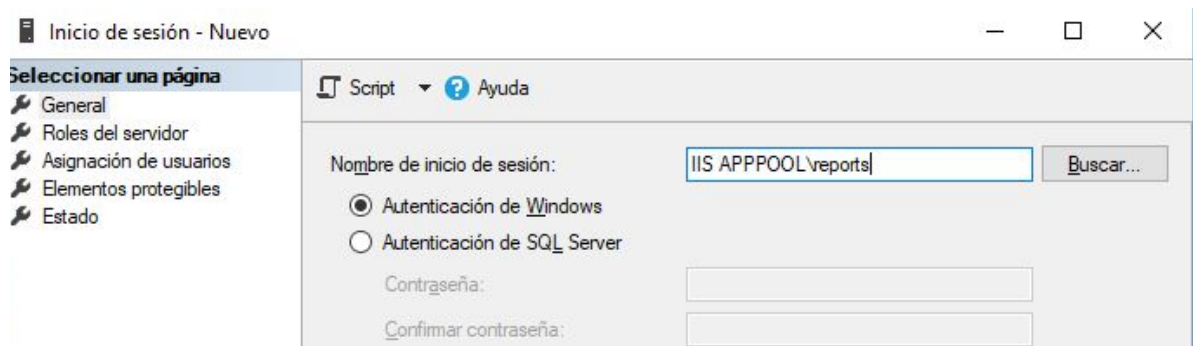
Nombre:

Versión de .NET CLR:

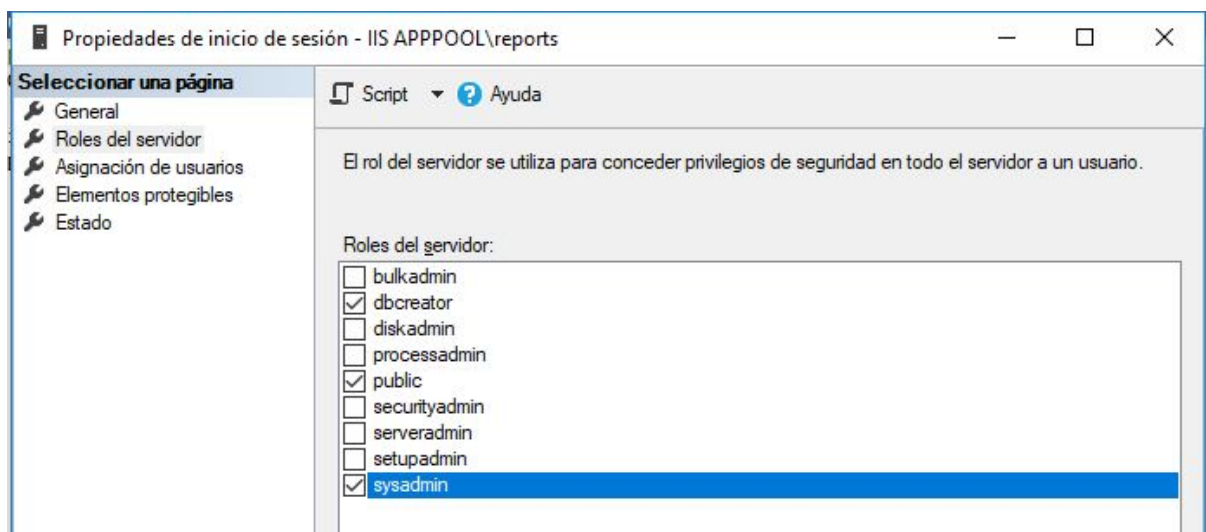
Modo de canalización administrada:

☒ Iniciar grupo de aplicaciones inmediatamente

9. En sql managment studio:
  - a. Seguridad\inicio de sesión\nuevo inicio de sesión
  - b. nombre: IIS APPPOOL\reports



c. En roles marcar publish, sysadmin, dbcreator



10. En las dos maquinas servidor y cliente: deshabilitar conexión ethernet en las dos maquinas y habilitar conexión wifi

11. Loguearse con usuario de gestion

a. <http://wlccentro.ort.edu.uy>

12. Configurar salida sin proxy

13. Deshabilitar el firewall de windows en el servidor

14. Ver la ip asignada por wifi en el servidor

15. Hacer una prueba desde el navegador en la maquina cliente

a. <http://172.29.1.51:8080/api/Areas>

16. En la maquina cliente importar la colección y variables de ambiente en postman. Se puede acceer al repositorio en el servidor

\\172.29.1.51\d\$\defensa\163471-217922-demo\Postman

17. Configurar la variable de ambiente {{domain}} con la url del servidor

a. <http://172.29.1.51:8080>

18. Configurar la variable de ambiente {{AreaConectionString}} con la dirección del servidor
  - a. 172.29.1.51\\SQLEXPRESS
19. Levantar el frontend y conectarse