UNIVERSIDAD ORT URUGUAY – FACULTAD DE INGENIERÍA

Diseño de Aplicaciones 2

Segundo Obligatorio

Bruno Ferrari – N°: 167709

Diego Zuluaga – N°: 173642

Docentes: Ignacio Valle y Gabriel Piffaretti

Grupo: M6B ID

Fecha: Noviembre 2016

Abstract

Trabajo realizado por Bruno Ferrari y Diego Zuluaga como segundo obligatorio de la materia Diseño de Aplicaciones 2. El sistema trataba en su primera versión de una API Rest que ofrece las operaciones necesarias para el manejo de un local de cobranzas. El obligatorio fue construído aplicando TDD en su modalidad Test First y teniendo en cuenta los principios de diseño SOLID, guías de diseño GRASP y estándares de Clean Code. Para el desarrollo del código se utilizó Visual Studio 2015 en su versión Enterprise y Github para el control de versiones. La base de datos fue implementada utilizando Entity Framework (apoyándonos en el software SQL Server 2014 Management Studio para el manejo de la misma).
Esta segunda versión agrega por sobre lo anterior dos interfaces de usuario: una web codificada haciendo uso de AngularJS que consume los servicios ofrecidos por nuestra API Rest (y además agrega las nuevas funcionalidades que fueron solicitadas para esta parte), y una aplicación de Windows Form para uso de los administradores del sistema que permite manejar el nuevo sistema de puntos, importación de productos y consultas del log.
Se mejoraron además los mensajes de error que se muestran al usuario así como también se simplificaron los datos necesarios para que los usuarios hicieran requests a la API. Se incluyen además un mayor número de datos de prueba para facilitar aún más el testeo del sistema.

Índice

Manual de instalación

Pasos a seguir:

1. Asegúrese de tener todos sus servicios SQL iniciados y las características de windows relacionadas al IIS y a .NET activadas. Además, acuérdese de desactivar el antivirus para evitar problemas.
2. En la carpeta "*Release"* de la Root del CD se encuentran dos sub-carpetas, *"todo_pagos_production"* (la cual contiene la Web Api publicada) y "*Aplicación Administrativa"* (que contiene versión release de la aplicación de escritorio). Mueva estas carpetas a donde usted le quede más cómodo para utilizarlas, pero retírelas del CD.
3. Haga click derecho en la carpeta "*todo_pagos_production"* y dele permisos totales al usuario "*IIS_IUSRS"* y al usuario "*Todos".*
4. Publique en IIS el sitio web de nombre *todo_pagos_production* apuntando a la carpeta del mismo nombre. Acuérdese de seleccionar un puerto que usted sepa esté libre y de no tocar la configuración de ningún otro campo.
5. Abra Microsoft SQL Server Management Studio y conéctese a su motor de base de datos.
6. Ejecute el script "TodoPagosDB PRUEBAS" que se encuentra dentro de la carpeta del mismo nombre, que a su vez está en la carpeta "Scripts y Backups" del Root del CD.
7. Haga click derecho sobre la base de datos "TodoPagosDB" que aparece luego de ejecutar el script anterior, y seleccione la opción "Nueva Consulta".
8. Aquí ejecute el script "Configuración IIS APPPOOL".
9. Luego, haga click derecho sobre su motor de base de datos, click en propiedades, click en Permisos, y concédale todos los permisos al usuario "IIS APPPOOL\todo_pagos_production". Luego haga click en Aceptar para conservar los cambios.
10. Modifique el connection string del archivo Web.xml (dentro de la carpeta todo_pagos_production) y del archivo "AdminUserInterface.exe.xml" con el nombre de su motor de base de datos
11. En IIS, haga click en el sitio web creado anteriormente (*"todo_pagos_production"*) y seleccione "Reiniciar" en la columna a la derecha.
12. ¡Ya está listo para utilizar nuestro sistema!

    a. Si desea utilizar el sitio web, entre a su navegador preferido y escriba como dirección *"http://-IP.DE.SU.MAQUINA-:-PUERTO.ESCOGIDO-"*. Ejemplo:

       - http://172.16.5.14:8004

b. Si desea utilizar la aplicación de escritorio, haga doble click en el archivo “AdminUserInterface.exe” que se encuentra en la carpeta “*Aplicación Administrativa”* mencionada anteriormente.

Manual de uso de la API de TodoPagos

Bienvenido al manual de usuario de la api creada por Bruno Ferrari y Diego Zuluaga! En este manual tomaremos un poco de su tiempo para enseñarle la manera óptima de usar TodoPagos API, además de proveer un par de ejemplos por si usted mismo desea probarlo.

Antes de probar la api acuérdese de seguir la guía de instalación al pie de la letra, o tendrá problemas a la hora de usar el sistema.

Sin más preambulo, empezemos!

Uris expuestas

Nuestra api cuenta con 5 uris expuestas al público, cada una ofreciendo un servicio distinto de las demás. Estas son:

1. /api/v1/login
2. /api/v1/users
3. /api/v1/providers
4. /api/v1/payments
5. /api/v1/query/earnings
6. /api/v1/clients

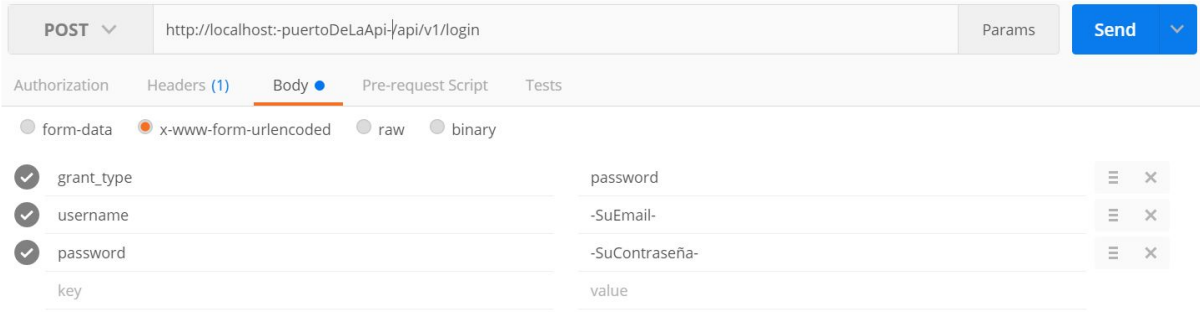
Recuerde que todas ellas se escriben solamente en minúscula. Ahora, pasemos a describir las acciones que se puede realizar con cada una.

1) /api/v1/login

El propósito de esta URI es iniciar sesión en el sistema a través de un token de autenticación.

Primera Parte: Conseguir el token

Uno debe hacer un POST a la misma con los siguientes datos en postman:



- En -puertoDeLaApi-, debe colocar el número de puerto en el que la api fue montada
- En -SuEmail- y -SuContraseña-, va el mail y la contraseña con los que usted fue registrado en la base de datos. Se suministrará un usuario administrador de correo diego@bruno.com y contraseña “HOLA1234” y un usuario cajero de correo nacho@gabriel.com de contraseña “HOLA1234” para poder iniciar sesion por primera vez y si lo desea, probar partes del sistema sin necesidad de crear otros usuarios

Luego de hacer este post, postman devolverá el siguiente mensaje si usted suministró sus datos de forma correcta:

```
Status: 200 (OK)

{
                                                                    "access_token":
"uTqdBClTGsF5ldDFTwZIo7R0UOQrsa4W2MMoxp8PhystrOoFfFqIERPgZ9re4-zPGjlj3u9-Z3azQsqxI35cq6GNR8
7HrGmbDXozmTg1vXvbs2SxhmNXuFAf_Q9N2jirJOlRYMVg_YaD_SSsOXOjOL-43drmb7_KOlB2Pw73HSwhp7c0pIrhl
UsaRrRg_NBPau1yY8Gw33FL0r3wDOc70yMvVq6lYWKSxMPbbjG-0EI",
  "token_type": "bearer",
  "expires_in": 86399
}
```

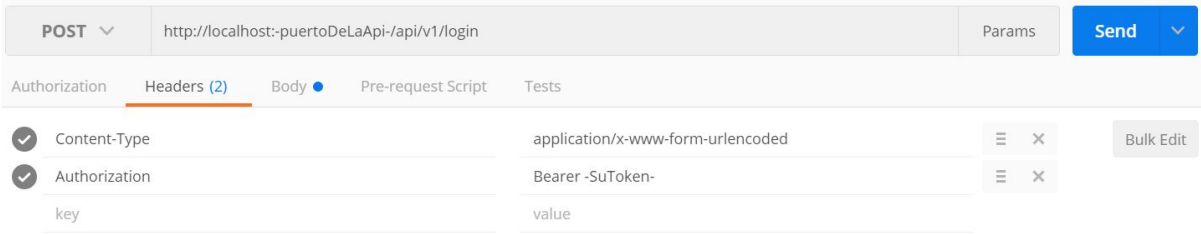
La cadena de caracteres seguida de `"access_token"` (y sin las comillas) es su token de autorización, que caducará en un día aproximadamente. Si ingresó sus datos mal, o si no está ingresado en el sistema, postman devolverá:

```
Status: 400 (Bad Request)
{
  "error": "invalid_grant",
  "error_description": "The user name or password is incorrect."
}
```

Si este error persiste, contacte al proveedor de este software.

Segunda Parte: Agregar el token al header

Luego de tener su token, valla a la sección de *header* del postman, e ingrese la siguiente información:



¡Y pronto! ¡Ya puede utilizar TodoPagosApi tranquilamente! Simplemente acuérdese de no modificar su header mientras siga utilizando la aplicación o deberá realizar este proceso una segunda vez.

2) /api/v1/users

El propósito de esta sección es manejar a los usuarios ingresados en el sistema de TodoPagosApi.

Esta URI cuenta con 4 acciones posibles: GET, POST, PUT y DELETE.

Acuérdese que para utilizar los servicios brindados en esta URI en particular, debe haber iniciado sesión y tener rol de Administrador. Si usted no ha iniciado sesión o no cuenta con rol de Administrador, postman dirá:

```
Status: 401 (Unauthorized)
{
  "Message": "Authorization has been denied for this request."
}
```

GET

Usted puede realizar un GET a */api/v1/users* para obtener los usuarios ingresados actualmente en el sistema. TodoPagosApi devolverá un JSON con la lista de usuarios ingresados en el sistema, con código:

```
Status: 200 (OK)
```

Tenga en cuenta que las contraseñas y los salts de los usuarios apareceran como *null* para evitar problemas de seguridad en el sistema.

Nota:

Si usted conoce la ID de un usuario ingresado en el sistema, y desea conseguir solo su información, puede hacer un GET de a cualquiera de las siguientes URI:

- */api/v1/users/x*
- */api/v1/users?ID=x*
- */api/v1/users/?ID=x*

Siendo *x* la ID del usuario que usted conoce. Esta llamada devolverá la información particular del usuario con ID *x*. Si el usuario de ID *x* no es encontrado en el sistema, postman dirá:

```
Status: 404 (Not Found)
```

Comunicando que ese usuario no se encuentra ingresado en el sistema.

Nota 2:

Si usted desea saber los roles de un usuario en particular, puede hacer un GET de la siguiente URI:

- */api/v1/users/getRoles?userEmail=x*

Siendo x el email del usuario del cual se desea conocer los roles. Esta llamada devolverá una lista con los nombres de los roles que el usuario tiene. Si ese email no está en la base de datos, postman devolverá:

```
Status: 404 (Not Found)
```

POST

Usted puede realizar un POST a */api/v1/users* suministrando un nuevo usuario al sistema, y este mismo será agregado. La información del mismo debe ser enviada en el *Body* del mensaje, de esta forma:

- Formato: RAW (JSON)

- Contenido:

```
{
        "Name" : "-NombreDelNuevoUsuario-",
        "Email" : "-EmailDelNuevoUsuario-",
        "Password" : "-ContraseñaDelNuevoUsuario-",
        "Roles" : [-RolesDelNuevoUsuario-]
}
```

- Si todo sale bien, postman retornará `Status 201 (Created)`, y la información del usuario creado (con la contraseña y salt en null para evitar problemas de seguridad)

Recordar que:

- El email es único, no puede haber dos usuarios con el mismo email
- La contraseña debe tener mínimo 8 caracteres y tener al menos una mayúscula y un número
- Los posibles Roles son:
    - AdminRole
    - CashierRole

EJEMPLO:

```
{
        "Name" : "Raul",
        "Email" : "raulito@gmail.com",
        "Password" : "Contra123",
        "Roles" : ["CashierRole", "AdminRole"]
}
```

POSIBLES ERRORES:

- Si el usuario nuevo ingresado tiene un email ya presente en la base de datos, postman retornará:
    - `Status: 400 (Bad Request)`
- Si la contraseña del usuario no es lo suficientemente segura, postman retornará:
    - `Status: 400 (Bad Request)`
- Si el usuario nuevo no tiene toda la información completa o no tiene formato correcto, postman retornará:
    - `Status: 400 (Bad Request)`

PUT

Usted puede realizar un PUT a */api/v1/users* suministrando un usuario modificado al sistema, y este mismo será actualizado. La información del mismo debe ser enviada en el *Body* del mensaje, de esta forma:

- Formato: RAW (JSON)

- Contenido:

```
{
        "ID" : -IDDelUsuarioAModificar-,
        "Name" : "-NombreDelNuevoUsuario-",
        "Email" : "-EmailDelNuevoUsuario-",
        "Password" : "-ContraseñaDelNuevoUsuario-",
        "Roles" : [“-UnRolDelNuevoUsuario-”]
```

}

(Tenga en cuenta que usted puede dejar campos vacíos con “” si no desea actualizar esa información)

Además, en la URI, se debe incluir la ID del usuario (que debe ser la misma que en el body del mensaje, ya que las ID no se pueden cambiar), de una de estas formas:

- */api/v1/users/x*
- */api/v1/users?ID=x*
- */api/v1/users/?ID=x*

Si todo sale bien, postman dirá:

```
Status: 204 (No Content)
```

EJEMPLO:

Path: http://localhost:4128/api/v1/users/1

```
{
        "ID" : 1,
        "Name" : "Hola",
        "Email" : "",
        "Password" : "",
        "Roles" : []
}
```

Esto modificará solo el nombre del usuario de ID 1.

POSIBLES ERRORES:

- Si el usuario que se suministra no tiene todos los campos especificados:
    - `Status: 400 (Bad Request)`
- Si la ID del Body no es la misma que la ID del Path:
    - `Status: 400 (Bad Request)`
- Si hay un usuario ya en la base de datos con el nuevo correo:
    - `Status: 400 (Bad Request)`
- Si no hay un usuario con esa ID en el sistema:
    - `Status: 404 (Not Found)`

DELETE

Usted puede realizar un DELETE a */api/v1/users* suministrando la ID de un usuario que se quiere eliminar, de cualquiera de las siguientes formas:

- */api/v1/users/x*
- */api/v1/users?ID=x*
- */api/v1/users/?ID=x*

Si todo sale bien, postman contestará:

```
Status: 204 (No Content)
```

POSIBLES ERRORES:

- Si usted está tratando de eliminarse a sí mismo:
    - `Status: 404 (Not Found)`
- Si el usuario no existe en la base de datos:
    - `Status: 404 (Not Found)`

3) /api/v1/providers

El propósito de esta sección es manejar a los proveedores ingresados en el sistema de TodoPagosApi.

Esta URI cuenta con 4 acciones posibles: GET, POST, PUT y DELETE.

Acuérdese que para utilizar los servicios brindados en esta URI en particular, debe haber iniciado sesión y tener rol de Administrador (excepto para los GET, estos se pueden hacer siendo Cajero). Si usted no ha iniciado sesión o no cuenta con rol necesario , postman dirá:

```
Status: 401 (Unauthorized)
{
  "Message": "Authorization has been denied for this request."
}
```

GET

Usted puede realizar un GET a */api/v1/providers* para obtener los proveedores ingresados actualmente en el sistema. TodoPagosApi devolverá un JSON con la lista de proveedores ingresados en el sistema, con código:

```
Status: 200 (OK)
```

Nota:

Si usted desea obtener la información de solo los proveedores que están activos, usted puede hacer un GET a cualquiera de los siguientes Paths:

- */api/v1/providers*?getActiveProviders=true
- */api/v1/providers/*?getActiveProviders=true

Por lo contrario, si desea obtener la información de solo los proveedores que están eliminados, usted puede hacer un GET a cualquiera de los siguientes Paths:

- */api/v1/providers*?getActiveProviders=false
- */api/v1/providers/*?getActiveProviders=false

Por último, si usted conoce la ID de un proveedor ingresado en el sistema, y desea conseguir solo su información, puede hacer un GET de a cualquiera de las siguientes URI:

- */api/v1/providers/x*
- */api/v1/providers?ID=x*
- */api/v1/providers/?ID=x*

Siendo *x* la ID del proveedor que usted conoce. Esta llamada devolverá la información particular del proveedor con ID *x*. Si este proveedor no es encontrado en el sistema, postman dirá:

```
Status: 404 (Not Found)
```

Comunicando que ese proveedor no se encuentra ingresado en el sistema.

POST

Usted puede realizar un POST a */api/v1/providers* suministrando un nuevo proveedor al sistema, y este mismo será agregado. La información del mismo debe ser enviada en el *Body* del mensaje, de esta forma:

- Formato: RAW (JSON)

- Contenido:

```
{
        "Commission" : -Número Que Representa Porcentaje De Ganancia Por Pago-,
```

```
        "Name" : -Nombre Proveedor-,
        "Fields" : [{
                "Type" : -Que tipo de field es-,
                "Name" : -Nombre del Field-
        }]
}
```

- Si todo sale bien, postman retornará `Status 201 (Created)`, y la información del proveedor creado

Recordar que:

- El nombre de un proveedor no se puede repetir entre los proveedores activos
- Comission debe estar entre 0 y 100 inclusive
- El proveedor brindado no puede estar ya marcado como eliminado
- Los posibles Type de Fields son :
    - TextField
    - NumberField
    - DateField

EJEMPLO:

```
{
        "Commission" : 2,
        "Name" : "Antel",
        "Fields" : [{
                "Type" : "NumberField",
                "Name" : "Cedula"
        }]
}
```

POSIBLES ERRORES:

- Si el proveedor nuevo ingresado tiene un nombre ya presente en un proveedor activo de la base de datos, postman retornará:
    - `Status: 400 (Bad Request)`
- Si comission es negativa o por arriba de 100
    - `Status: 400 (Bad Request)`
- Si el proveedor nuevo no tiene toda la información completa o no tiene formato correcto, postman retornará:
    - `Status: 400 (Bad Request)`

PUT

Usted puede realizar un PUT a */api/v1/providers* suministrando un proveedor modificado al sistema, y este mismo será actualizado. La información del mismo debe ser enviada en el *Body* del mensaje, de esta forma:

- Formato: RAW (JSON)
- Contenido:

```
{
	“ID” : -ID del proveedor a modificar-
	"Commission" : -Número Que Representa Porcentaje De Ganancia Por Pago-,
	"Name" : -Nombre Proveedor-,
	"Fields" : [{
		"Type" : -Que tipo de field es-,
		"Name" : -Nombre del Field-
	}]
}
```

(Tenga en cuenta que usted puede dejar campos vacíos con “” si no desea actualizar esa información. Para no actualizar comission, debe poner un número negativo o por arriba de 100, o debe poner el valor anterior. Recuerde que no puede actualizar un provider de modo que su estado quede como inactivo/eliminado, para eso existe el DELETE. No se puede modificar un proveedor marcado como desactivado/eliminado)

Además, en la URI, se debe incluir la ID del proveedor (que debe ser la misma que en el body del mensaje, ya que las ID no se pueden cambiar), de una de estas formas:

- */api/v1/providers/x*
- */api/v1/providers?ID=x*
- */api/v1/providers/?ID=x*

Si todo sale bien, postman dirá:

```
Status: 204 (No Content)
```

EJEMPLO:

Path: http://localhost:4128/api/v1/providers/1

```
{
	“ID” : 1,
	"Commission" : 101,
	"Name" : "AntelData",
	"Fields" : []
}
```

Esto modificará solo el nombre del proveedor de ID 1.

POSIBLES ERRORES:

- Si el proveedor que se suministra no tiene todos los campos especificados:
    - `Status: 400 (Bad Request)`
- Si la ID del Body no es la misma que la ID del Path:
    - `Status: 400 (Bad Request)`
- Si hay un proveedor activo ya en la base de datos con el nuevo nombre:
    - `Status: 400 (Bad Request)`
- Si el proveedor a modificar está marcado como desactivado/eliminado:
    - `Status: 400 (Bad Request)`
- Si no hay un proveedor con esa ID en el sistema:
    - `Status: 404 (Not Found)`

DELETE

Usted puede realizar un DELETE a */api/v1/providers* suministrando la ID de un proveedor que se quiere eliminar, de cualquiera de las siguientes formas:

- */api/v1/providers/x*
- */api/v1/providers?ID=x*
- */api/v1/providers/?ID=x*

Si todo sale bien, postman contestará:

```
Status: 204 (No Content)
```

Recuerde que eliminar un proveedor significa que el mismo se marcará como desactivado/eliminado en la base de datos, ningún dato de él será perdido (esto es para evitar problemas de consistencia con pagos guardados)

POSIBLES ERRORES:

- Si el proveedor no existe en la base de datos:
    - `Status: 404 (Not Found)`

4) /api/v1/payments

El propósito de esta sección es manejar a los pagos ingresados en el sistema de TodoPagosApi.

Esta URI cuenta con 2 acciones posibles: GET y POST..

Acuérdese que para utilizar los servicios brindados en esta URI en particular, debe haber iniciado sesión y tener rol de Administrador o de Cajero. Si usted no ha iniciado sesión o no cuenta con rol necesario, postman dirá:

```
Status: 401 (Unauthorized)
{
  "Message": "Authorization has been denied for this request."
}
```

GET

Usted puede realizar un GET a */api/v1/payments* para obtener los pagos ingresados actualmente en el sistema. TodoPagosApi devolverá un JSON con la lista de pagos ingresados en el sistema, con código:

```
Status: 200 (OK)
```

Nota:

Si usted conoce la ID de un pago ingresado en el sistema, y desea conseguir solo su información, puede hacer un GET de a cualquiera de las siguientes URI:

- */api/v1/payments/x*
- */api/v1/payments?ID=x*
- */api/v1/payments/?ID=x*

Siendo *x* la ID del pago que usted conoce. Esta llamada devolverá la información particular del pago con ID *x*. Si este no es encontrado en el sistema, postman dirá:

```
Status: 404 (Not Found)
```

Comunicando que el mismo no se encuentra ingresado en el sistema.

POST

Usted puede realizar un POST a */api/v1/payments* suministrando un nuevo pago al sistema, y este mismo será agregado. La información del mismo debe ser enviada en el *Body* del mensaje, de esta forma:

- Formato: RAW (JSON)

- Contenido:

```
{

        "AmountPaid" : -Plata pagada-,
        "PayMethod" : {

                "Type" : -Un PayMethod-,
                "PayDate" : -Fecha en formato correcto-
        },
        "Receipts" : [{

                "Amount" : -Total A Pagar en Factura-,
                "ReceiptProviderID" : -ID del proveedor de la factura-,
                "CompletedFields" : [{

                        "Type" : -Tipo de Field-,
                        "Data" : -Datos en forma de string-
                        "Name" : -Nombre Field-
                }]
        }]
}
```

- Si todo sale bien, postman retornará `Status 201 (Created)`, y la información del pago creado

Recordar que:

- Un nuevo pago no puede contener una factura que ya se encuentra ingresada en el sistema
- Las facturas solo pueden tener proveedores que estén ingresados en el sistema
- Las facturas deben tener los mismos capos (pero completos con datos) que su proveedor.
- Los dos tipos posibles de PayMethod son:
    - DebitPayMethod (Si se usa este, AmountPaid y la suma de todas las Amounts de todos los receipts deben ser iguales)
    - CashPayMethod (Si se usa este, AmountPaid debe ser mayor o igual a la suma de todas las Amounts de todos los receipts)
- Los posibles Type de Fields son (a los tres se les debe pasar la información entre comillas, no importa su tipo):
    - TextField (El Data de estos debe tener un string)
    - NumberField (El Data de estos debe tener un número en string)
    - DateField (El Data de estos debe tener una fecha en string en formato ISO 8061, es decir, yyyy-MM-ddTHH:mm:ssZ)

EJEMPLOS:

```
{
        "AmountPaid" : 250,
        "PayMethod" : {
                "Type" : "DebitPayMethod",
                "PayDate" : "2008-09-22T14:01:54Z"
        },
        "Receipts" : [{
                "Amount" : 250,
                "ReceiptProviderID" : 1,
                "CompletedFields" : [{
                        "Type" : "NumberField",
                        "Data" : "4901883",
                        "Name" : "Cedula"
                }]
        }]
}

{
        "AmountPaid" : 250,
        "PayMethod" : {
                "Type" : "DebitPayMethod",
                "PayDate" : "2008-09-22T14:01:54Z"
        },
        "Receipts" : [{
                "Amount" : 250,
                "ReceiptProviderID" : 3,
                "CompletedFields" : [{
                                "Data" : "4901883",
                                "Type" : "NumberField",
                                "Name" : "Cedula"
                        },{
                                "Data" : "2016-09-22T14:01:54Z",
                                "Type" : "DateField",
                                "Name" : "Fecha"
                        },{
                                "Data" : "Diego",
                                "Type" : "TextField",
                                "Name" : "Nombre"
                        }]
        }]
}
```

POSIBLES ERRORES:

- Si el pago nuevo ingresado tiene una factura que ya está en la base de datos:
    - `Status: 400 (Bad Request)`
- Si un Field está lleno de información no válida:
    - `Status: 400 (Bad Request)`
- Si el proveedor de alguna factura está marcado como eliminado en la base de datos:
    - `Status: 400 (Bad Request)`
- Si alguna fecha no está en formato correcto
    - `Status: 400 (Bad Request)`
- Si el pago nuevo no tiene toda la información completa o no tiene formato correcto, postman retornará:
    - `Status: 400 (Bad Request)`

5) /api/v1/query/earnings/

El propósito de esta sección es conseguir reportes de las ganancias del sistema de TodoPagosApi.

A esta URI solo se le puede hacer GET, pero en dos sub rutas posibles:

- /api/v1/query/earnings/allEarnings
- /api/v1/query/earnings/earningsPerProvider

Acuérdese que para utilizar los servicios brindados en esta URI en particular, debe haber iniciado sesión y tener rol de Administrador. Si usted no ha iniciado sesión o no cuenta con rol necesario, postman dirá:

```
Status: 401 (Unauthorized)
{
  "Message": "Authorization has been denied for this request."
}
```

Acordarse que las ganancias se calculan por nombre de provider, entonces pagos hechos por “Antel” eliminado y “Antel” activo se contarán juntos.

GET /api/v1/query/earnings/allEarnings

Usted puede realizar un GET a /api/v1/query/earnings/allEarnings para obtener un número que representa las ganancias totales del sistema TodoPagosApi. Se le pueden pasar fecha desde y fecha hasta para obtener las ganancias totales en cierto periodo de tiempo. Al hacerlo, Postman devolverá:

```
Status: 200 (OK)

{
        250
}
```

Siendo ese número las ganancias totales del sistema entre las fechas solicitadas
Nota:

Como las fechas son opcionales, estas son las formas en que se puede llamar

- /api/v1/query/earnings/allEarnings?to=yyyy-MM-ddTHH:mm:ssZ
- /api/v1/query/earnings/allEarnings?from=yyyy-MM-ddTHH:mm:ssZ
- /api/v1/query/earnings/allEarnings?from=yyyy-MM-ddTHH:mm:ssZ&to=yyyy-MM-ddTHH:mm:ssZ
- /api/v1/query/earnings/allEarnings

Las fechas default son to=(la fecha de hoy) y from=DateTime.Min (`0001-01-01T00:00:00Z`)

El formato para pasarle las fechas es:

- ❖ yyyy-MM-ddTHH:mm:ssZ

EJEMPLO:

- /api/v1/query/earnings/allEarnings?to=2008-09-22T14:01:54Z

GET /api/v1/query/earnings/earningsPerProvider

Usted puede realizar un GET a /api/v1/query/earnings/earningsPerProvider para obtener un número las ganancias del sistema TodoPagosApi separados por proveedor. Se le pueden pasar fecha desde y fecha hasta para obtener las ganancias de cada proveedor en cierto periodo de tiempo. Al hacerlo, Postman devolverá:

```
Status: 200 (OK)

{
        “Antel” : 250
}
```

Siendo las ganancias de TodoPagosApi solo las provistas por el proveedor Antel, e igual a 250.

Nota:

Como las fechas son opcionales, estas son las formas en que se puede llamar

- /api/v1/query/earnings/earningsPerProvider?to=yyyy-MM-ddTHH:mm:ssZ
- /api/v1/query/earnings/earningsPerProvider?from=yyyy-MM-ddTHH:mm:ssZ
- /api/v1/query/earnings/earningsPerProvider?from=yyyy-MM-ddTHH:mm:ssZ&to=yyyy-MM-ddTHH:mm:ssZ
- /api/v1/query/earnings/earningsPerProvider

Las fechas default son to=(la fecha de hoy) y from=DateTime.Min (`0001-01-01T00:00:00Z`)

El formato para pasarle las fechas es cualquiera de estos dos (los espacios en blanco cambiados con %20):

- yyyy-MM-ddTHH:mm:ssZ

EJEMPLO:

- /api/v1/query/earnings/earningsPerProvider?from=2008-09-22T14:01:54Z

6) /api/v1/clients

```
{
        "Name" : "Diego",
        "IDCard" : "49018830",
        "PhoneNumber" : "26666666",
        "Address" : "Cuareim 1451",
        "Points" : 10
}
```

El propósito de esta sección es manejar a los clientes ingresados en el sistema de TodoPagosApi.

Esta URI cuenta con 4 acciones posibles: GET, POST, PUT y DELETE.

Acuérdese que para utilizar los servicios brindados en esta URI en particular, debe haber iniciado sesión y tener rol de Administrador o Cajero. Si usted no ha iniciado sesión o no cuenta con alguno de estos roles, postman dirá:

```
Status: 401 (Unauthorized)
{
  "Message": "Authorization has been denied for this request."
}
```

GET

Usted puede realizar un GET a */api/v1/clients* para obtener los clientes ingresados actualmente en el sistema. TodoPagosApi devolverá un JSON con la lista de clientes ingresados, con código:

```
Status: 200 (OK)
```

Nota:

Si usted conoce la ID de un cliente ingresado en el sistema, y desea conseguir solo su información, puede hacer un GET de a cualquiera de las siguientes URI:

- */api/v1/clients/x*
- */api/v1/clients?ID=x*
- */api/v1/clients/?ID=x*

Siendo *x* la ID del cliente que usted conoce. Esta llamada devolverá la información particular del cliente con ID *x*. Si el cliente de ID *x* no es encontrado en el sistema, postman dirá:

```
Status: 404 (Not Found)
```

Comunicando que ese usuario no se encuentra ingresado en el sistema.

POST

Usted puede realizar un POST a */api/v1/clients* suministrando un nuevo cliente al sistema, y este mismo será agregado. La información del mismo debe ser enviada en el *Body* del mensaje, de esta forma:

- Formato: RAW (JSON)

- Contenido:

```
{
        "Name" : -Un nombre-,
       "IDCard" : -Una cédula de identidad uruguaya-,
       "PhoneNumber" : -Un número de teléfono o celular-,
       "Address" : -Una dirección-
}
```

- Si todo sale bien, postman retornará `Status 201 (Created)`, y la información del cliente creado

Recordar que:

- La cédula es única, no puede haber dos clientes con la misma.

EJEMPLO:

```
{
        "Name" : "Diego",
       "IDCard" : "49018830",
       "PhoneNumber" : "26666666",
       "Address" : "Cuareim 1451"
}
```

POSIBLES ERRORES:

- Si el cliente nuevo ingresado tiene una cédula ya presente en la base de datos, postman retornará:
    - `Status: 400 (Bad Request)`

PUT

Usted puede realizar un PUT a */api/v1/clients* suministrando un cliente modificado al sistema, y este mismo será actualizado. La información del mismo debe ser enviada en el *Body* del mensaje, de esta forma:

- Formato: RAW (JSON)

- Contenido:

```
{
        "ID" : -Una id-,
         "Name" : -Un nombre-,
       "IDCard" : -Una cédula de identidad uruguaya-,
       "PhoneNumber" : -Un número de teléfono o celular-,
       "Address" : -Una dirección-,
       "Points" : -Una cantidad de puntos-
}
```

(Tenga en cuenta que usted puede dejar campos vacíos con “” si no desea actualizar esa información. Para no modificar la cantidad de puntos, puede colocar un número negativo)

Además, en la URI, se debe incluir la ID del cliente (que debe ser la misma que en el body del mensaje, ya que las ID no se pueden cambiar), de una de estas formas:

- */api/v1/clients/x*
- */api/v1/clients?ID=x*
- */api/v1/clients/?ID=x*

Si todo sale bien, postman dirá:

```
Status: 204 (No Content)
```

EJEMPLO:

Path: http://localhost:4128/api/v1/users/1

```
{
        "ID" : 1,
        "Name" : "Diego",
       "IDCard" : "",
       "PhoneNumber" : "",
        "Address" : “”,
      "Points" : -1
}
```

Esto modificará solo el nombre del usuario de ID 1.

POSIBLES ERRORES:

- Si el cliente que se suministra no tiene todos los campos especificados:
    - `Status: 400 (Bad Request)`
- Si la ID del Body no es la misma que la ID del Path:
    - `Status: 400 (Bad Request)`
- Si hay un cliente ya en la base de datos con la nueva cédula:
    - `Status: 400 (Bad Request)`
- Si no hay un cliente con esa ID en el sistema:
    - `Status: 404 (Not Found)`

DELETE

Usted puede realizar un DELETE a */api/v1/clients* suministrando la ID de un cliente que se quiere eliminar, de cualquiera de las siguientes formas:

- */api/v1/clients/x*
- */api/v1/clients?ID=x*
- */api/v1/clients/?ID=x*

Si todo sale bien, postman contestará:

```
Status: 204 (No Content)
```

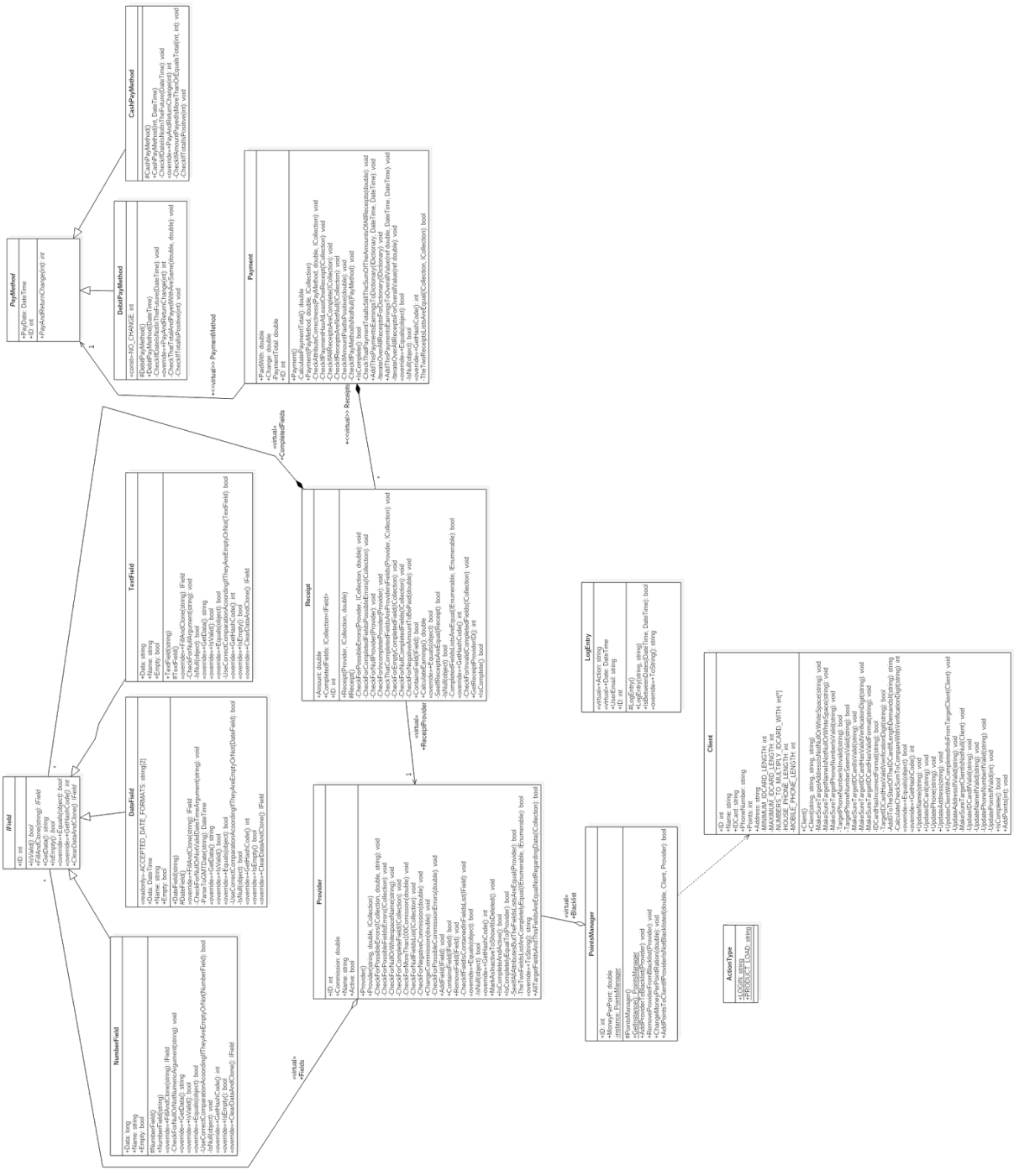
POSIBLES ERRORES:

- Si el cliente no existe en la base de datos:
    - `Status: 404 (Not Found)`

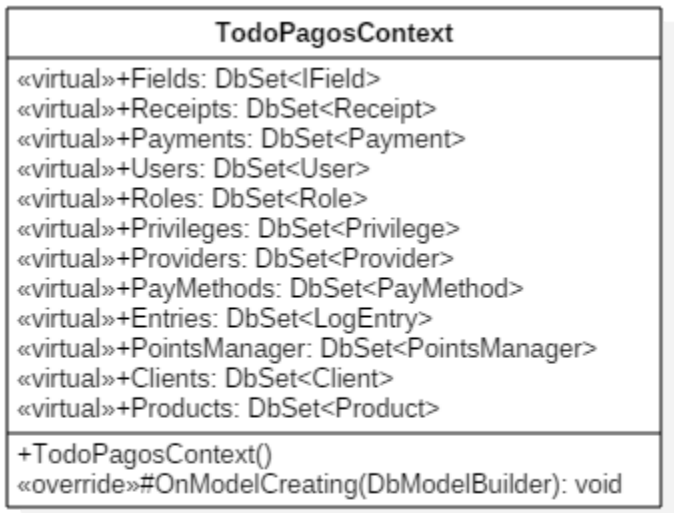
Descripción del Diseño Propuesto

Diagramas de Clases

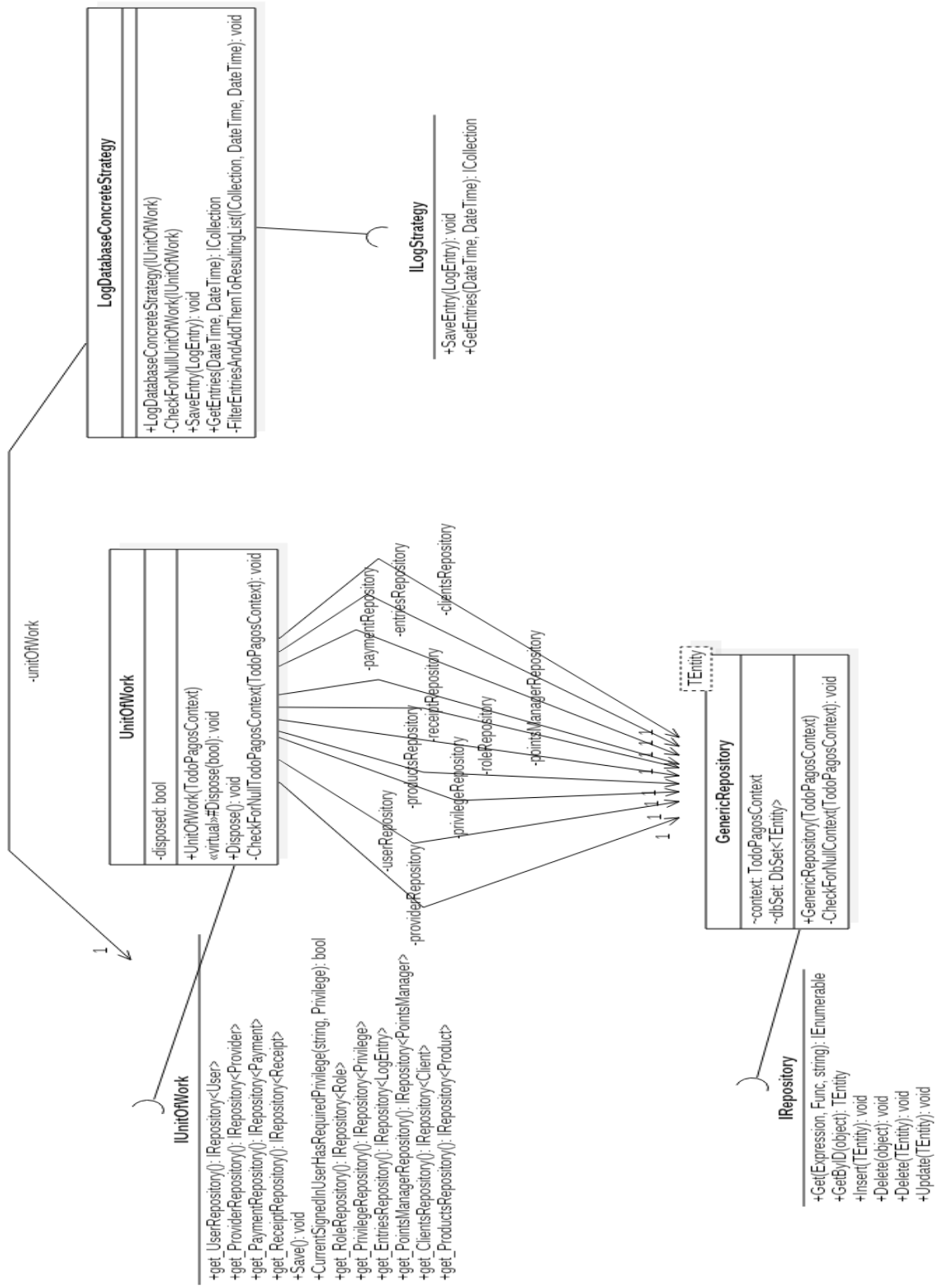
TodoPagos.Domain



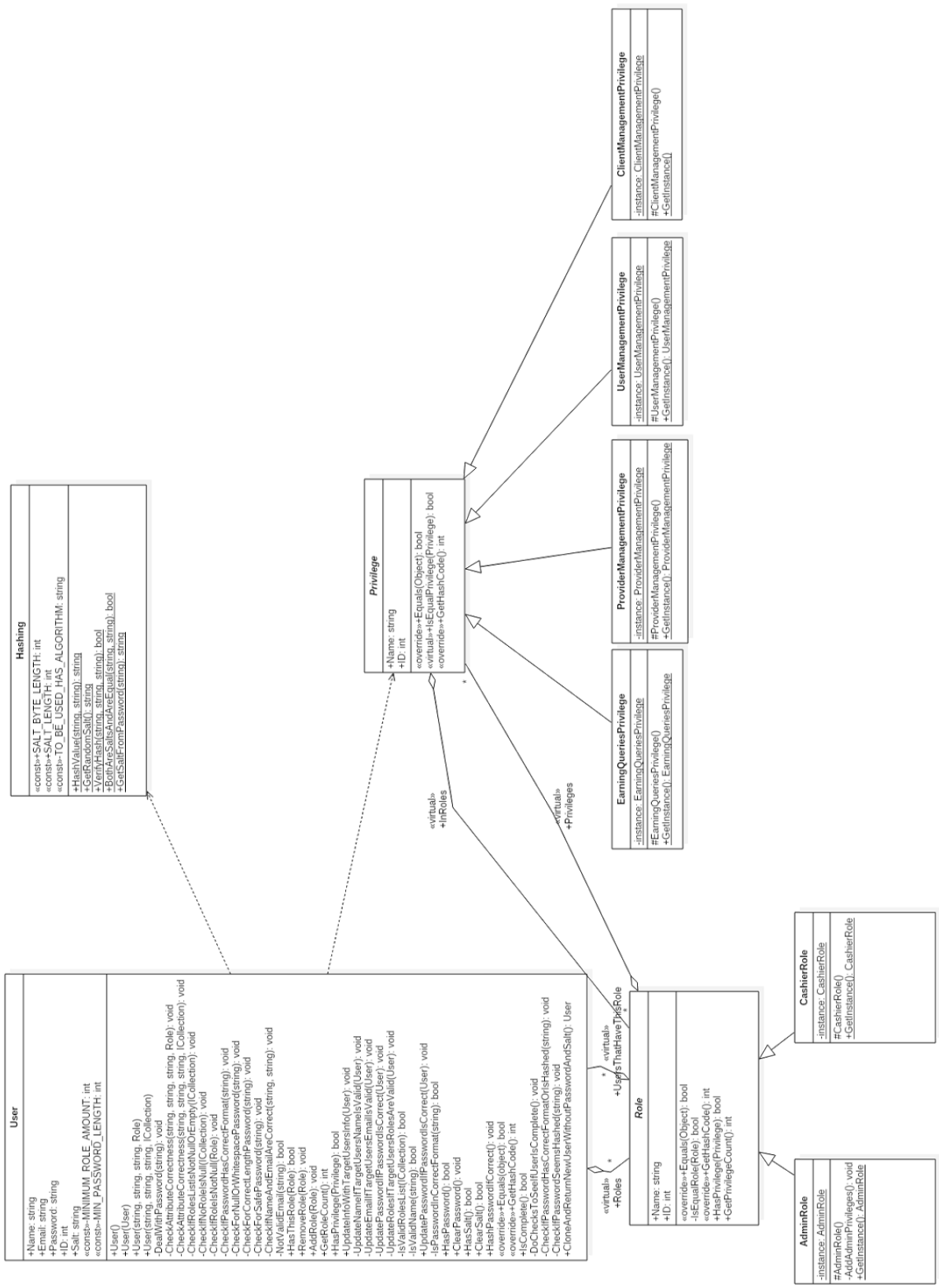
TodoPagos.Domain.DataAccess



TodoPagos.Domain.Repository

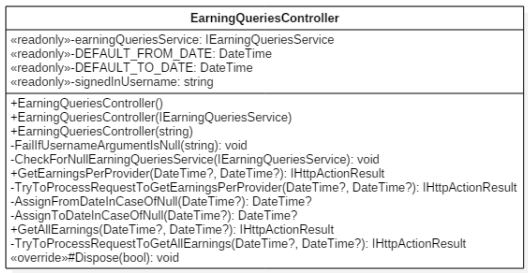


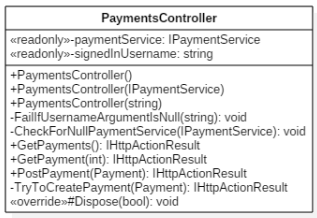
TodoPagos.UserAPI

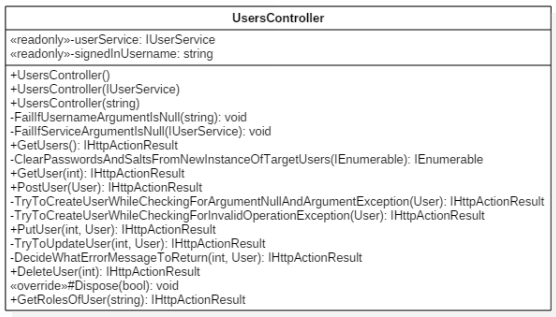


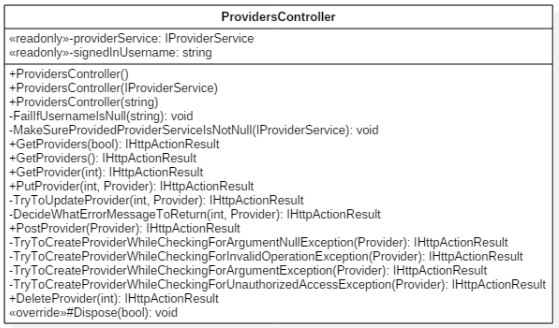
TodoPagos.Web.Api

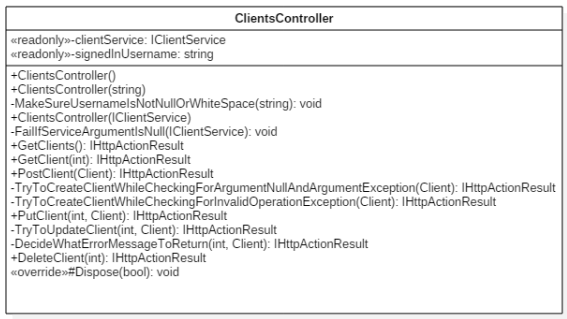
TodoPagos.Web.Api.Controllers



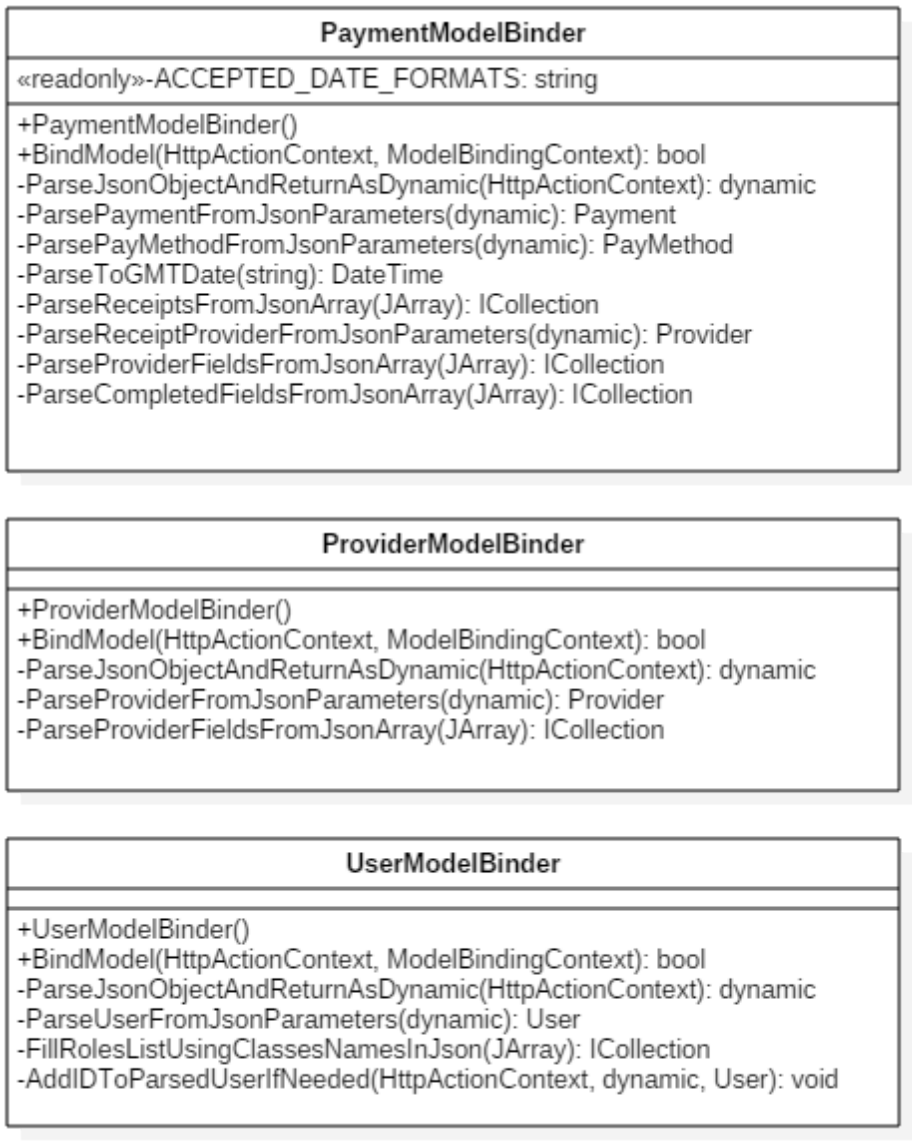




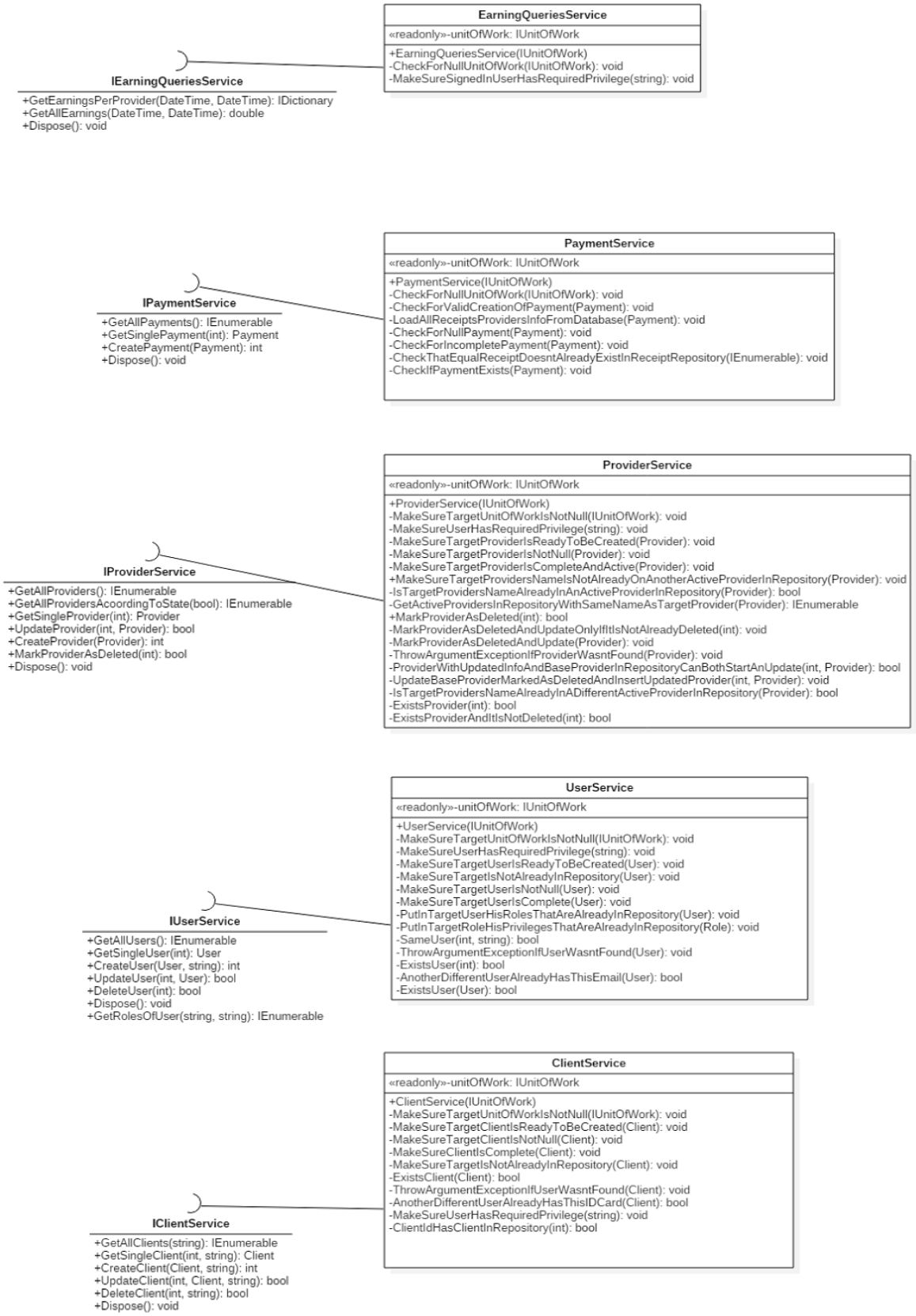




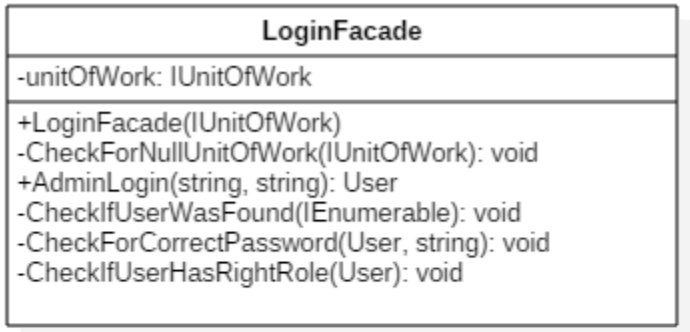
TodoPagos.Web.Api.Models

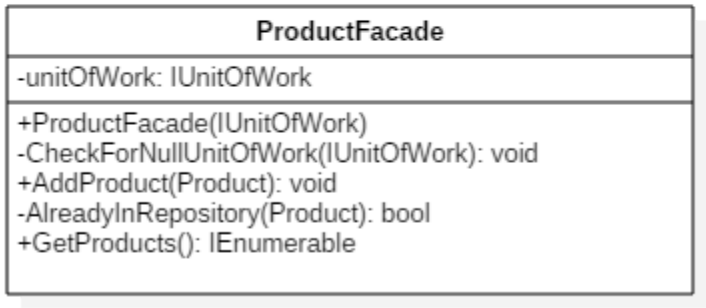


TodoPagos.Web.Services



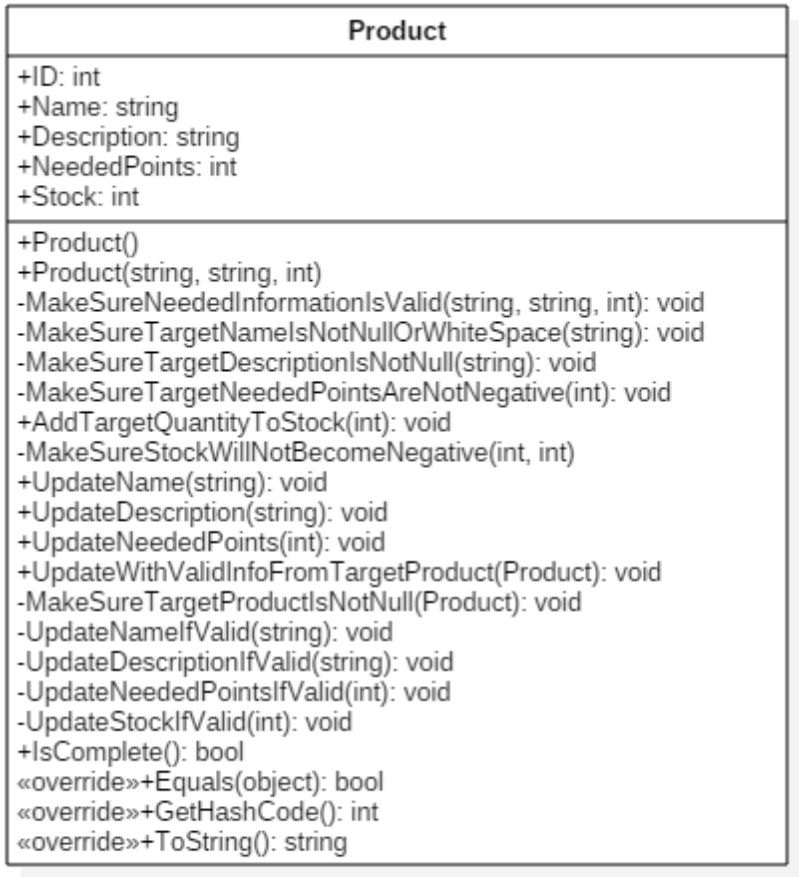
TodoPagos.AdminForm.Logic



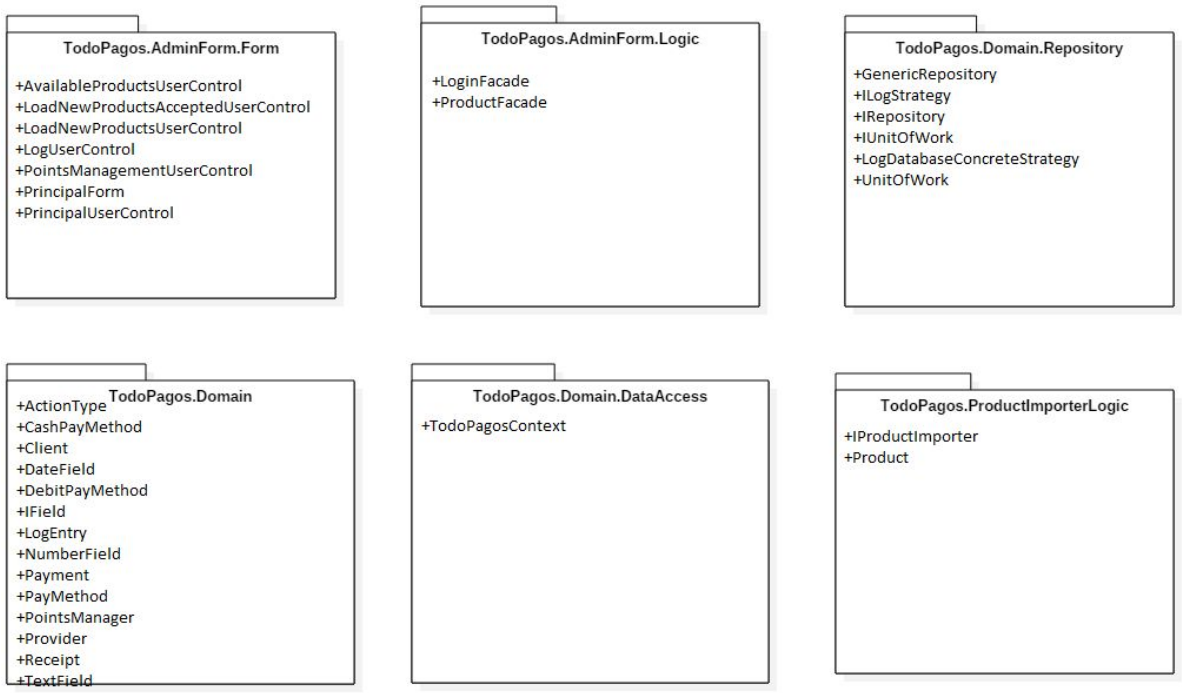


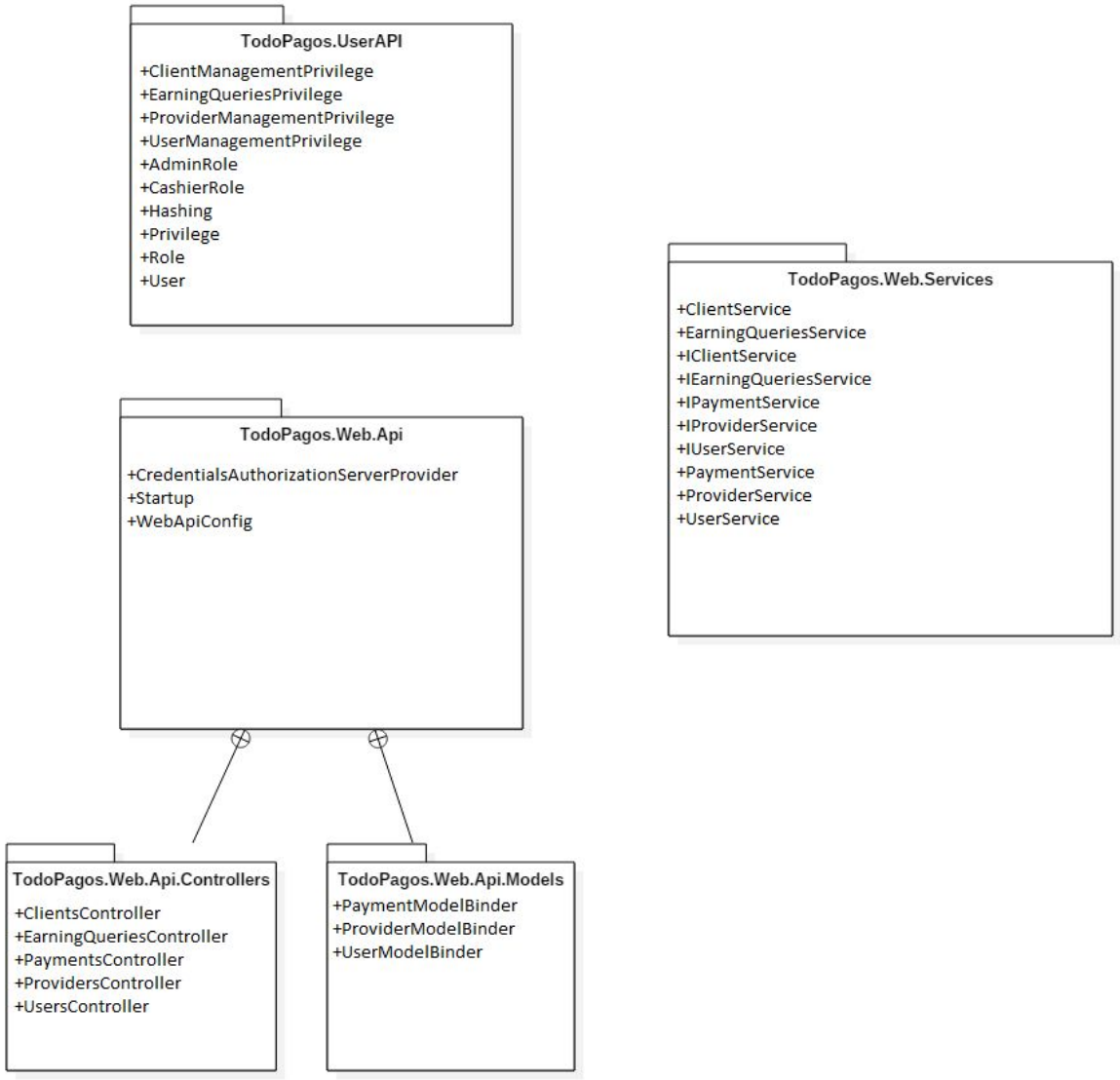
TodoPagos.AdminForm.Form

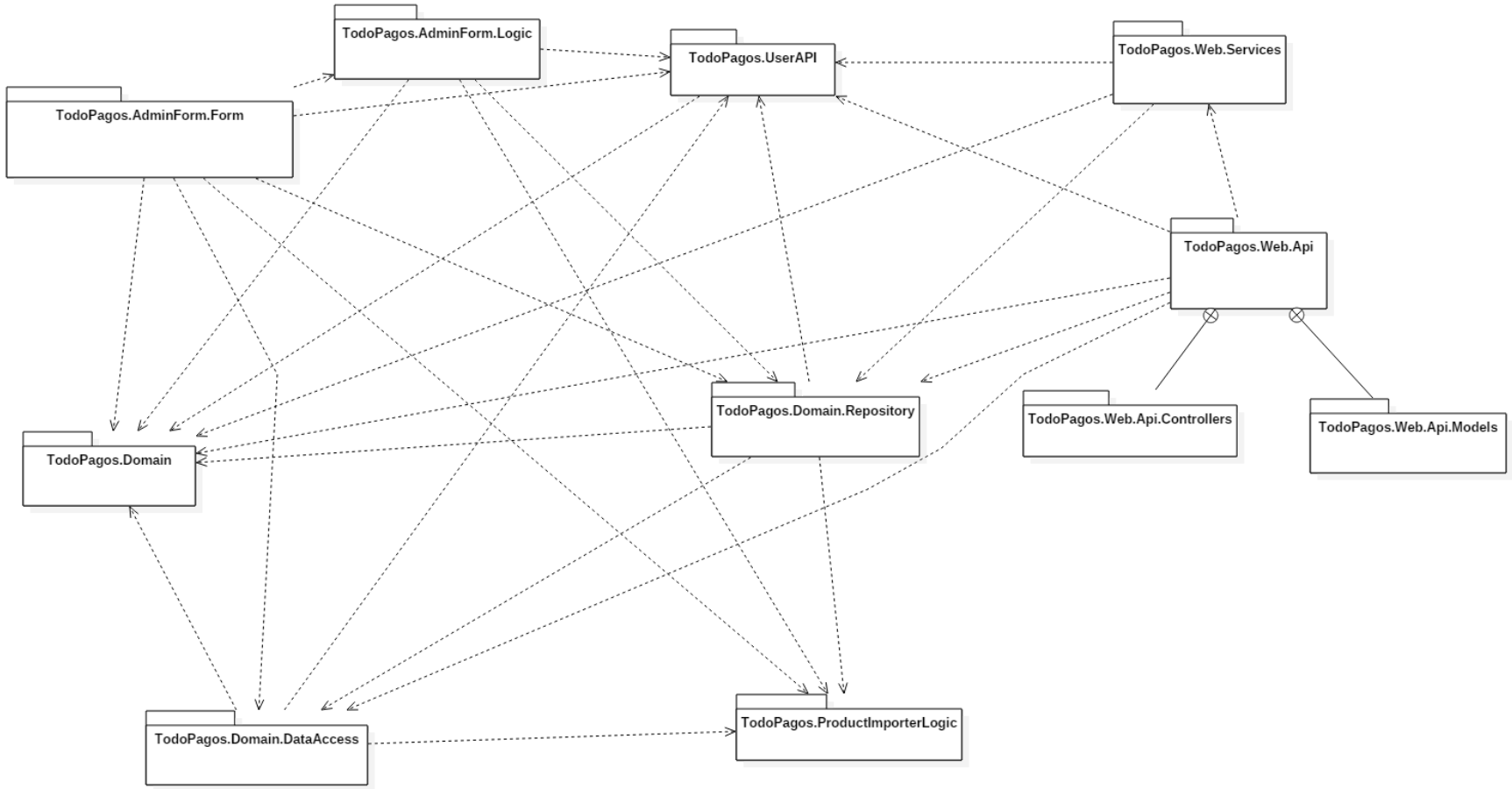
TodoPagos.ProductImporterLogic



Diagramas de Paquetes

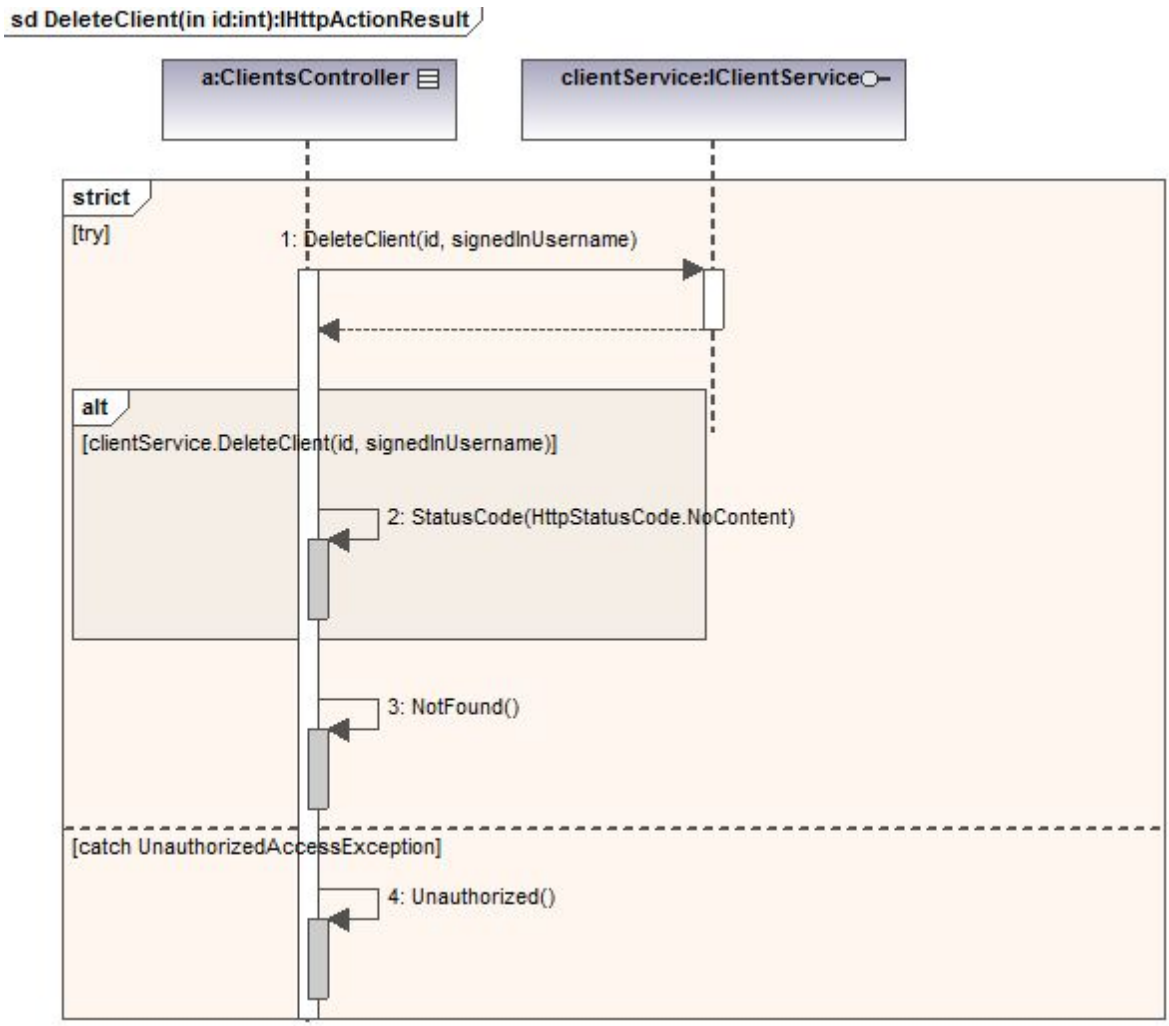



TodoPagos.UserAPI
+ClientManagementPrivilege
+EarningQueriesPrivilege
+ProviderManagementPrivilege
+UserManagementPrivilege
+AdminRole
+CashierRole
+Hashing
+Privilege
+Role
+User
TodoPagos.Web.Services
+ClientService
+EarningQueriesService
+IClientService
+IEarningQueriesService
+IPaymentService
+IProviderService
+IUserService
+PaymentService
+ProviderService
+UserService
TodoPagos.Web.Api
+CredentialsAuthorizationServerProvider
+Startup
+WebApiConfig
TodoPagos.Web.Api.Controllers
+ClientsController
+EarningQueriesController
+PaymentsController
+ProvidersController
+UsersController
TodoPagos.Web.Api.Models
+PaymentModelBinder
+ProviderModelBinder
+UserModelBinder

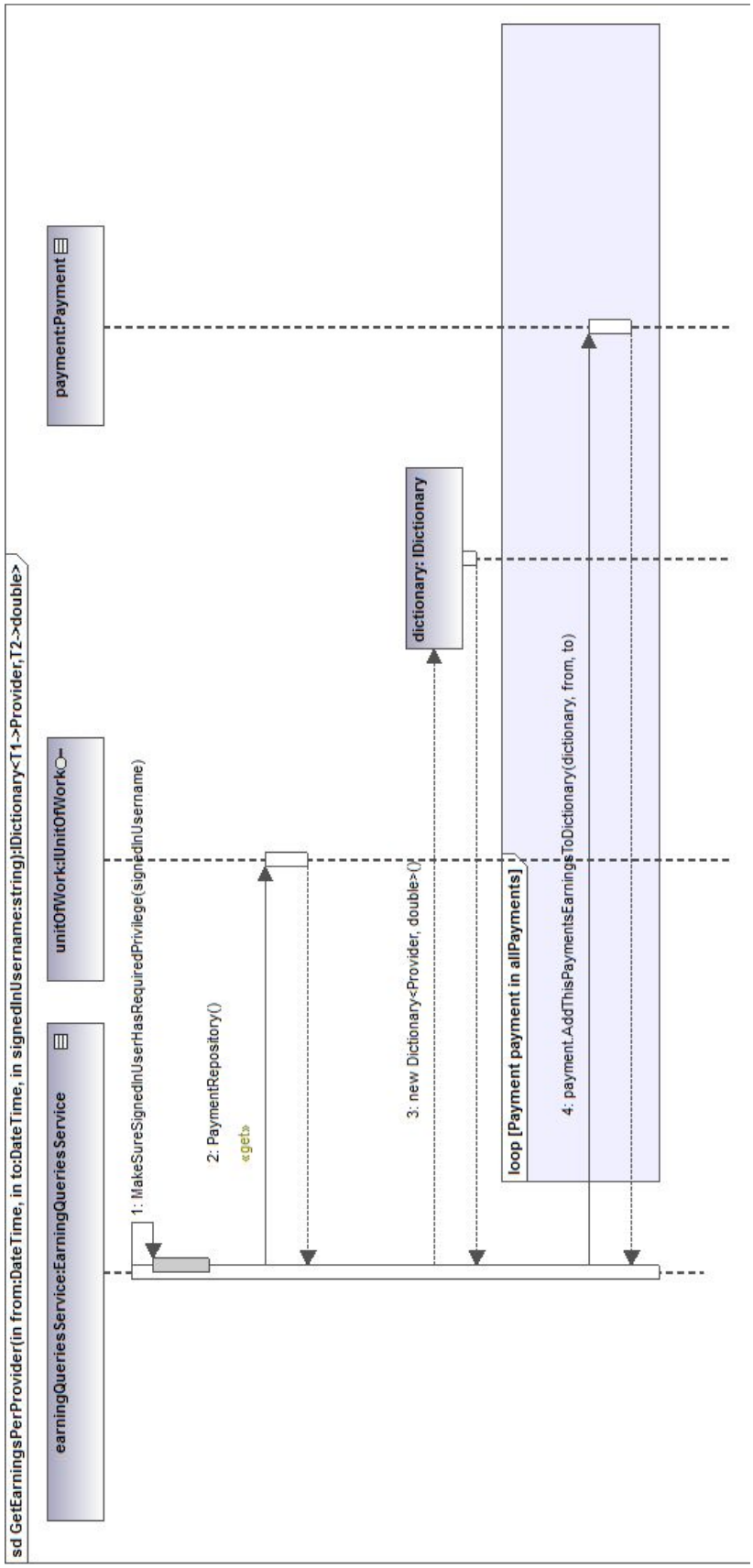

TodoPagos.AdminForm.Logic
TodoPagos.UserAPI
TodoPagos.Web.Services
TodoPagos.AdminForm.Form
TodoPagos.Web.Api
TodoPagos.Domain.Repository
TodoPagos.Web.Api.Controllers
TodoPagos.Web.Api.Models
TodoPagos.Domain
TodoPagos.Domain.DataAccess
TodoPagos.ProductImporterLogic

Diagramas de Interacción

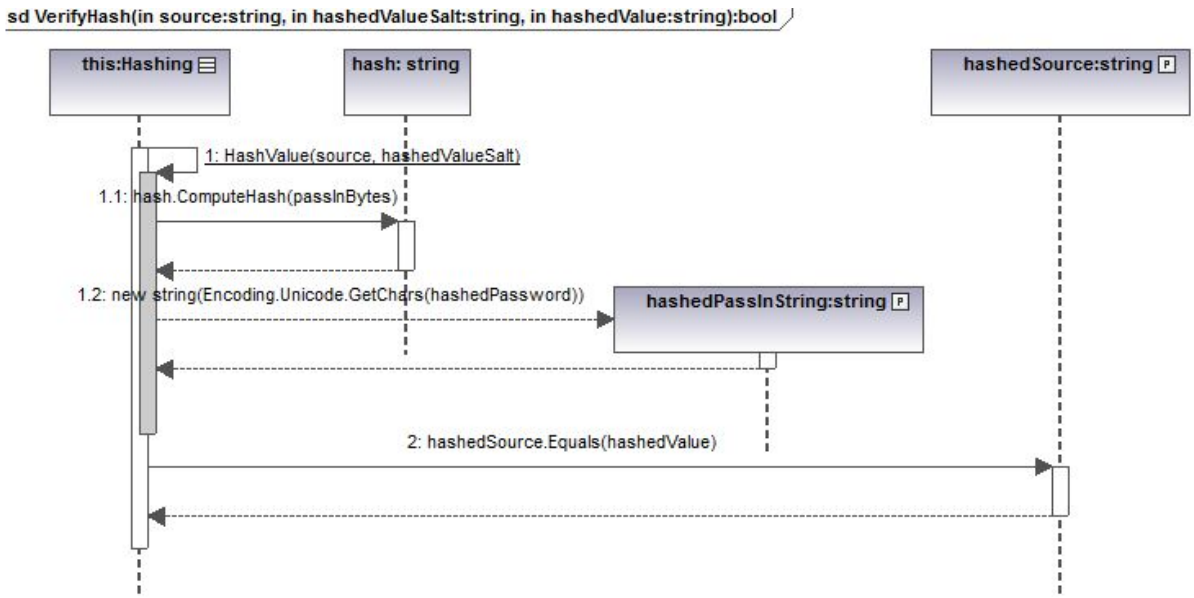
DeleteClient



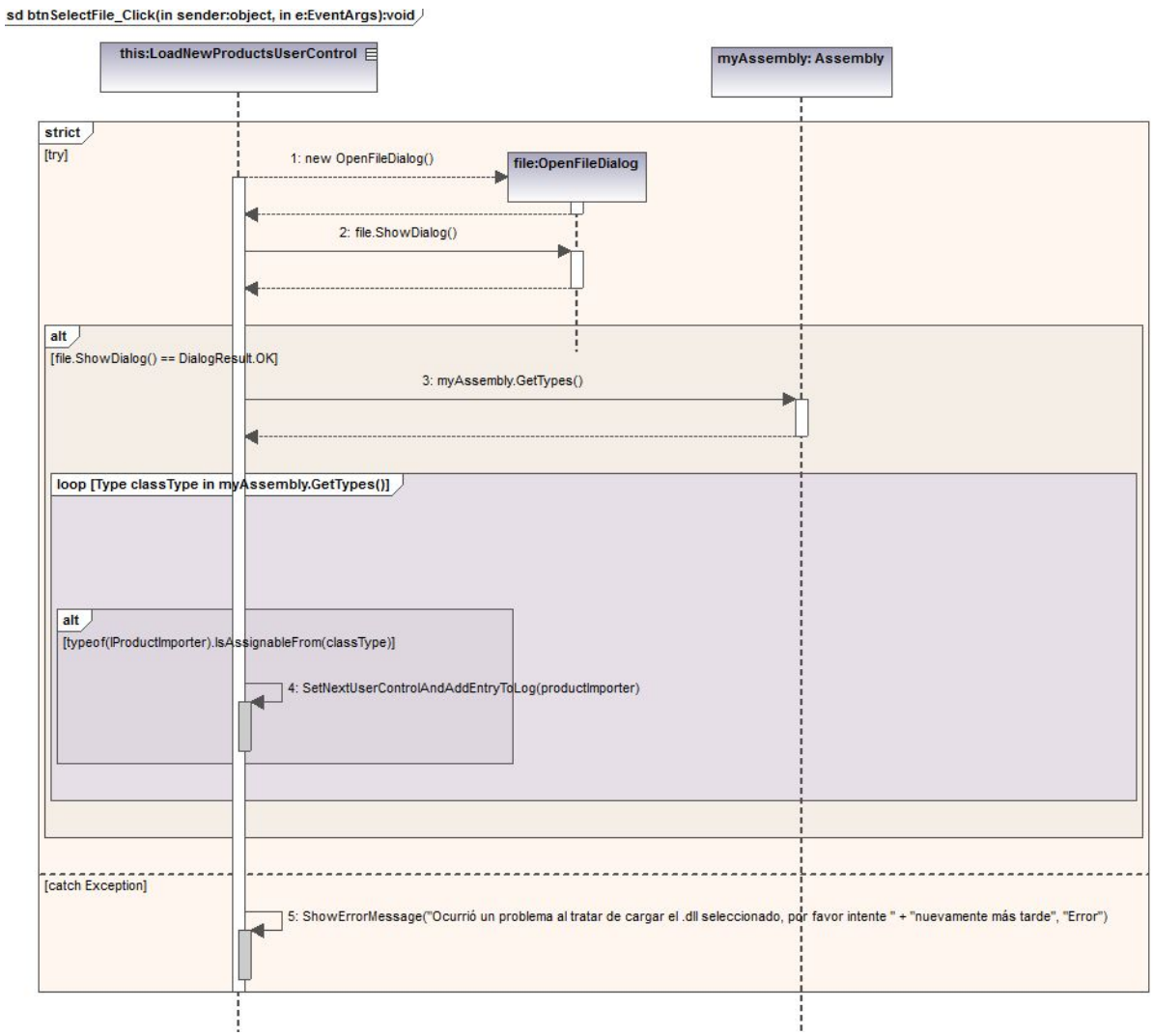
GetEarningsPerProvider



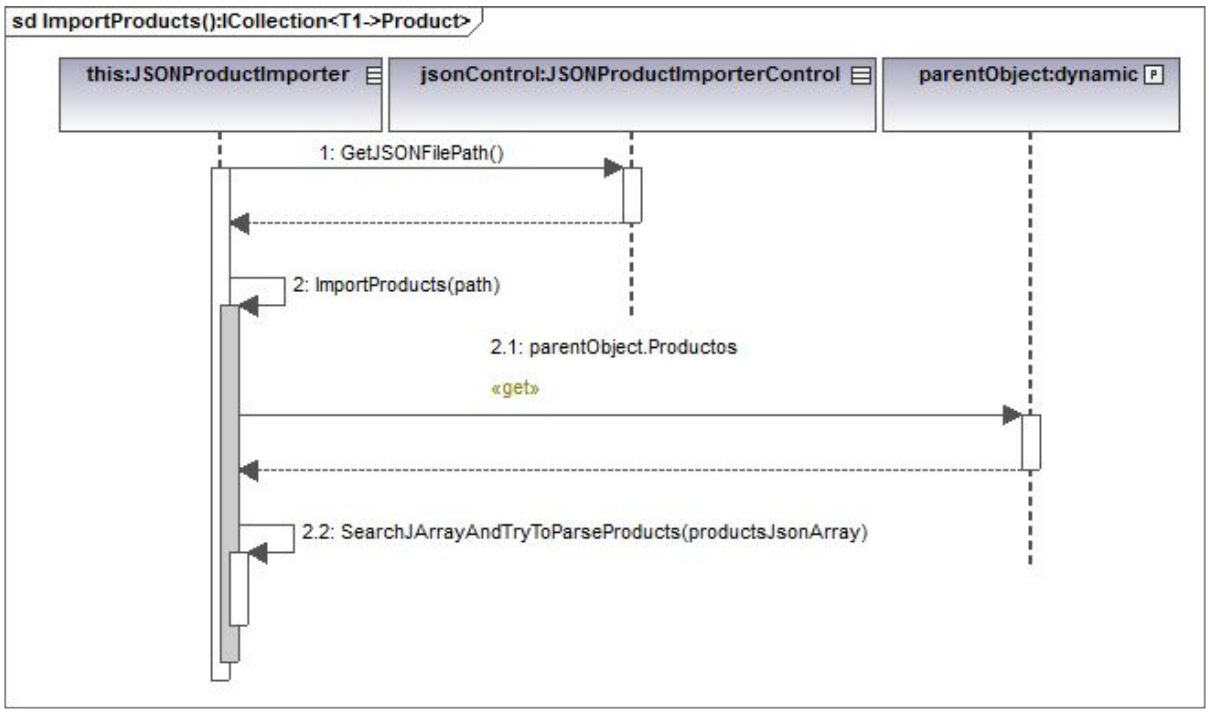
VerifyHash


sd VerifyHash(in source:string, in hashedValueSalt:string, in hashedValue:string):bool
this:Hashing
hash: string
hashedSource:string
1: HashValue(source, hashedValueSalt)
1.1: hash.ComputeHash(passInBytes)
1.2: new string(Encoding.Unicode.GetChars(hashedPassword))
hashedPassInString:string
2: hashedSource.Equals(hashedValue)

btnSelectFile_click



ImportProducts



AddThisPaymentsEarningsToDictionary

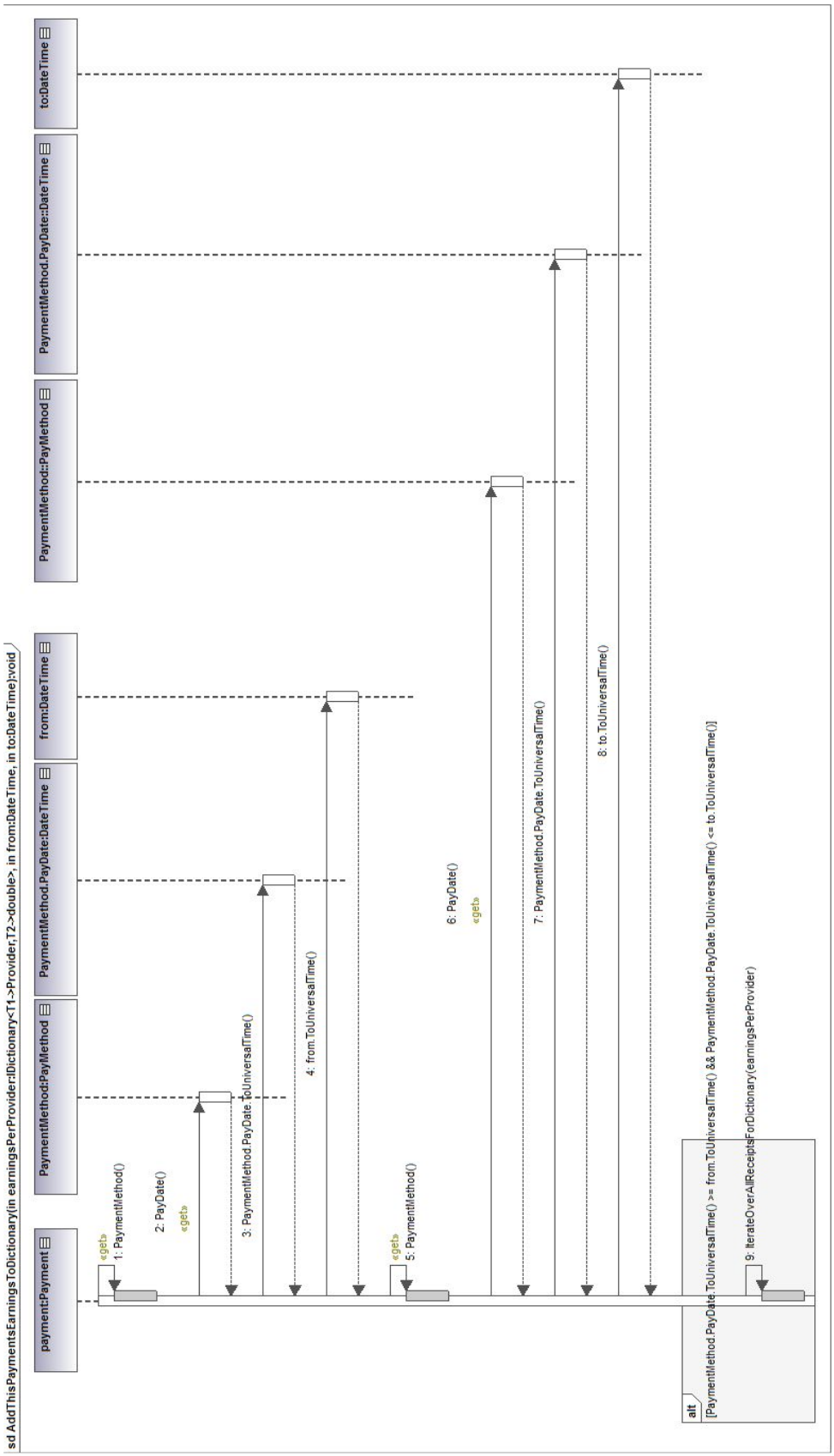


Diagrama de Componentes

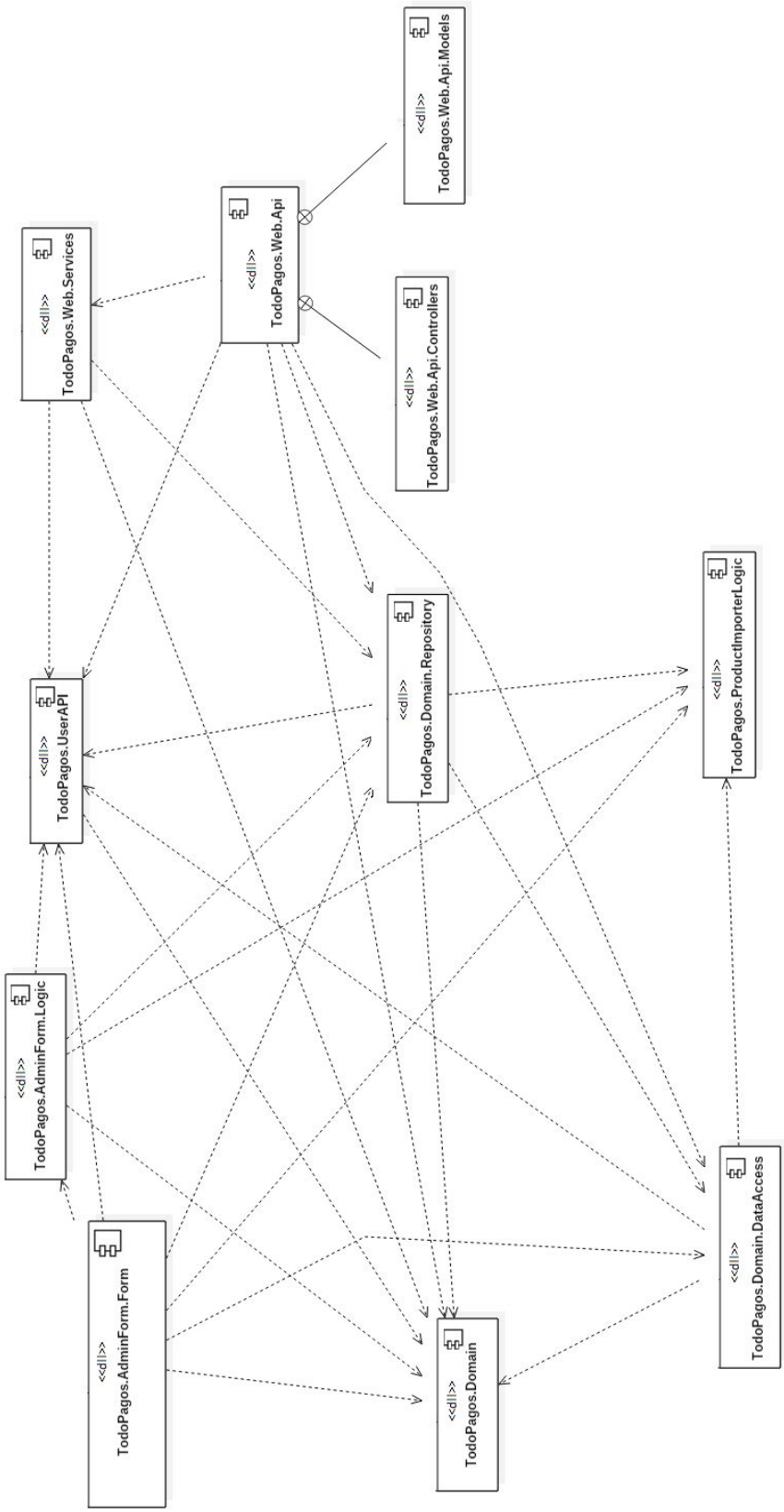
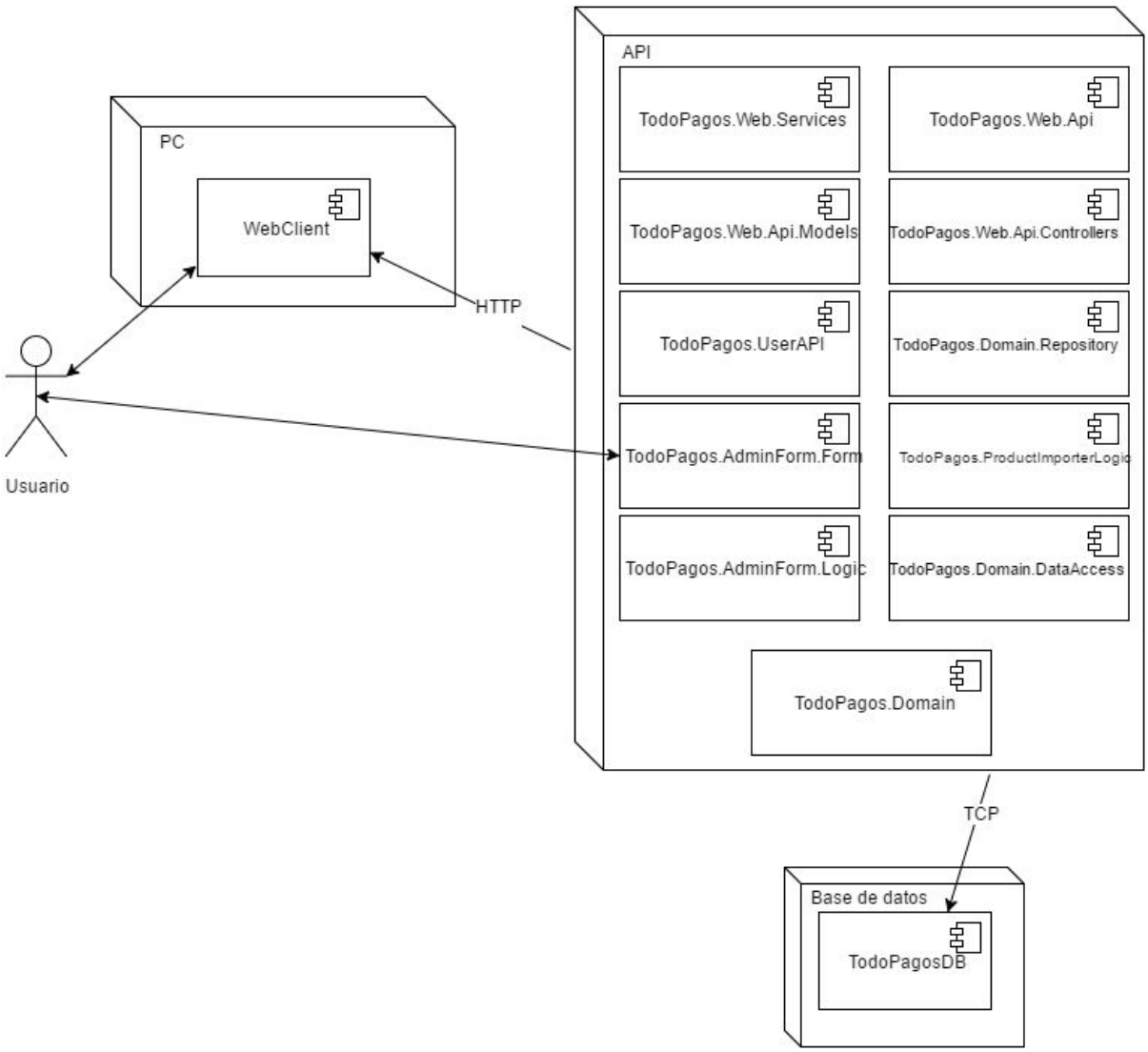


Diagrama de Entrega



Justificación de las decisiones de Diseño

Modelado de Roles

Al modelar los usuarios y los distintos roles que estos pueden asumir (al día de hoy cajero o administrador), decidimos apoyarnos en dos papers (referenciados al final de este documento y que fueron investigados a partir de la sugerencia de Ignacio Valle de utilizar Role Subtyping) que proponen una solución elegante al problema de los subtipos y polimorfismo en Object-Role Modelling.
En nuestro caso, decidimos adaptar lo aprehendido de la siguiente forma: se modeló una clase abstracta Role, que define las operaciones que todos los roles del sistema deberán tener. Luego, se definieron los roles concretos CashierRole y AdminRole. Para manejar los permisos de cada Role, se modeló una clase abstracta Privilege que define las operaciones que todos los privilegios del sistema deberán tener. La clase Role entonces posee una colección de Privilege. De esta manera, en el constructor de cada Role se agregan a su colección de privilegios aquellos que el Role comprende. Luego lo que resta es en la clase User agregar una property que sea una colección de Role.
Así, pudimos dejar abierta la posibilidad de que se agreguen nuevos roles y privilegios al sistema y que éstos se puedan agregar a los usuarios ya existentes, si fuera necesario. Además, permitió un manejo bastante simple y preciso sobre los permisos de acceso a las diferentes operaciones de la API por los distintos tipos de usuario (se pregunta si el usuario posee el Privilege necesario para acceder, y se busca a ese Privilege en los Role que el usuario tiene. Si se encuentra el Privilege, entonces se deja acceder al usuario y si no lo tiene, se tira una excepción que imposibilita el acceso a la operación).

Tipo de Dato de NumberField

En la clase NumberField, se optó por colocar el tipo de dato como de tipo long, para así evitarnos cualquier problema de almacenamiento (admite números decimales, números mayores al máximo Integer).

Hasheo de Contraseñas

Decidimos hashear las contraseñas para mayor seguridad de los usuarios. Se implementó una clase Hashing que se encarga de esto. El algoritmo utilizado fue SHA256.
Para esto decidimos utilizar también un salt que acompañe al hash para aumentar la seguridad de la aplicación.

Blanqueo de Contraseñas

Cuando se hace una query sobre la base de datos pidiendo algún usuario, decidimos devolverlo poniendo en null el hash y el salt que corresponden a su contraseña. Se optó por no implementar una View (es decir, aplicar el modelo MVC) ya que consideramos que fabricar una nueva clase solamente para borrar los datos correspondientes a la contraseña hubiera aumentado la complejidad innecesariamente.

Cálculo del Total de Payment

Decidimos que el cálculo del total de cada Payment se hará en la aplicación cliente que se hará en el segundo obligatorio en vez de necesitar pedírselo al servidor.

Patrón Singleton en Role y Privilege

Se decidió utilizar el patrón Singleton en las clases concretas derivadas de la clase abstracta pura Role y de la clase abstracta pura Privilege. Esto se realizó ya que no nos interesaba que hubieran múltiples instancias de un mismo privilegio o de un mismo rol porque el sistema puede manejarse perfectamente manteniendo una única instancia de cada privilegio y rol permitiéndonos mejorar la eficiencia al hacer operaciones sobre las tablas correspondientes en la base de datos.

Patrón Facade

Decidimos utilizar el patrón Facade en nuestro obligatorio para definir una interfaz por la cual nuestro sistema se comunicará con las diferentes aplicaciones cliente (en este caso WebApi, pero en un futuro se podría quitar WebApi e implementar un cliente Windows Form, por ejemplo). Nuestras “fachadas” son los Services (UserService, ProviderService, PaymentService y EarningQueriesService). A través de ellas se puede acceder a las distintas operaciones que nuestro sistema ofrece.
Este patrón nos permite asegurarnos de que las diferentes aplicaciones cliente no conozcan qué es lo que hay detrás de los Service (se agrega una capa de abstracción), mientras que, a la vez, provee un punto de acceso bien definido a nuestro sistema (que es lo que permite no acoplarse a ninguna aplicación cliente en específico).

Patrón Repository

Se implementó el patrón Repository para poder agregar una capa de abstracción entre nuestro código y el mecanismo de persistencia elegido (Entity Framework). De esta manera, logramos que nuestro código fuera ignorante del mecanismo de persistencia elegido. Si en

un futuro se hicieran cambios mayores a Entity Framework, o se deseara cambiar el mecanismo de persistencia, entonces nuestro código no se vería afectado gracias a la implementación de este patrón.

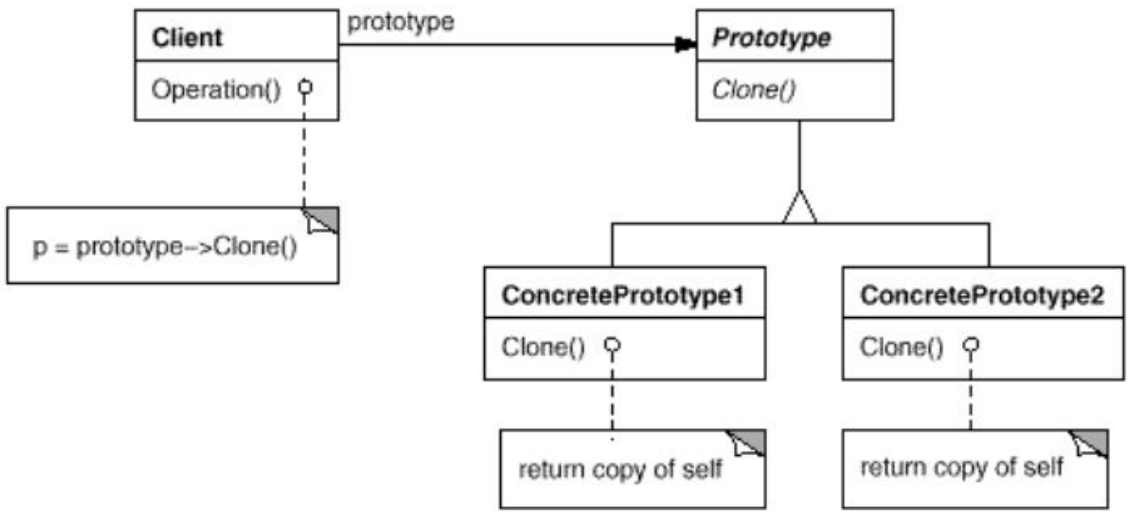
Repositorio Genérico y Unit Of Work

Junto con el patrón Repository, se implementó un repositorio genérico (utilizando Generics) ya que el crear una clase Repository para cada entidad resulta en mucho código redundante. Además, para asegurarnos de que todos los repositorios usen el mismo contexto y así evitar los problemas que podrían surgir a la hora de actualizar diferentes tipos en una misma transacción (si cada tipo utiliza un contexto diferente entonces uno podría fallar y el otro no), utilizamos la clase UnitOfWork.

Patrón Prototype

Antes de entrar en detalle sobre la implementación de este patrón, nos gustaría definirlo brevemente ya que no es un patrón que hayamos dado ni en Diseño de Aplicaciones 1 ni en lo que va de Diseño de Aplicaciones 2.
El patrón Prototype es un patrón creacional que tiene como objectivo crear nuevos objectos duplicándolos, clonando una instancia creada previamente. La situación que este patrón ayuda a resolver es cuando es impreciso abstraer la lógica que decide qué tipos de objetos se utilizarán, de la lógica que luego usarán esos objetos en su ejecución. A su vez, el patrón diferencia entre dos tipos de clonación posibles: superficial o profunda. Nosotros utilizamos la clonación profunda (al clonar se devuelve una nueva instancia del objeto).

A continuación, una imagen de la estructura propuesta por el libro Design Patterns de GoF:



En nuestro caso, la implementación de este patrón fue en la clase IField y sus derivadas (NumberField, DateField y TextField), además de completamente inconsciente. Queríamos resolver el problema de tener distintos tipos de IField y poder manejar los mismos de manera uniforme, sin preguntar por el tipo concreto (es decir, evitar hacer RTTI, Run-Time Type Identification). Además queríamos que el llenar un IField concreto con un dato devolviera un IField, para así poder desprendernos del tipo concreto ni bien se terminara con la validación correspondiente a dicho tipo concreto en la clase IField concreta. Para esto, implementamos un método llamado FillAndClone(), que todos las clases que heredan de IField deben implementar. Este método recibe un string y se encarga de parsear dicho string al tipo de dato concreto y luego devolver un IField lleno y despojado del tipo.

Mediante todo esto pudimos evitar realizar RTTI mientras a la vez permitíamos extensibilidad a futuro, en el caso de que se agregaran nuevos tipos de IField.
Mientras investigábamos los distintos patrones, nos encontramos con el patrón Prototype, que resultó ser casi-exactamente lo que habíamos implementado en la clase IField.

Repetición de Role en la Base de Datos

Al comenzar a utilizar nuestra aplicación y crear múltiples usuarios, se notará que en vez de mantener un AdminRole y un CashierRole, se crearán múltiples roles AdminRole y múltiples roles CashierRole (tantos como usuarios con cada Role existan), en vez de mantener solamente un objeto AdminRole y un objeto CashierRole. Decidimos dejar que se creen múltiples objetos Role ya que si en futuro quisiéramos agregarle algún atributo que varíe por usuario, entonces el cambio sería trivial, mientras que si mantuviéramos un único objeto de cada Role, entonces este cambio tendría un gran impacto de cambio. Si bien pecamos de algo de genericidad especulativa aquí, consideramos que el costo/beneficio de dejar que los objetos Role se repitan en la base de datos era más alto que el de mantener un único objeto de cada Role y arriesgarnos a que luego este objeto tuviera que ser modificado por alguna razón y así obligarnos a tener que volver a guardar todos los objetos Role.

Catch de Exception

En los Models, se observará que hicimos un catch de Exception (es decir, del tipo más general de las excepciones), que en general es una mala práctica. Somos conscientes de ello, pero lo justificamos argumentando que existen muchos tipos distintos de excepciones que los métodos que se encargan de parsear el JSON recibido podrían tirar, y el hacer un catch de cada tipo distinto de excepción solamente para devolver false (si se detecta una excepción en los models se devuelve false, que luego es controlado por los Service), agregaba complejidad al código de forma innecesaria, por lo que se optó por capturar Exception de manera general y devolver false en el caso de que alguna excepción fuera capturada.

UserAPI

Decidimos separar todo lo concerniente al manejo de usuarios (Privilege, Role, User) en un assembly separado, ya que observamos que este tipo de manejo de roles y permisos podría ser reusado en cualquier otro sistema que también deba manejar usuarios con permisos. Por esto, se optó por no acoplar esta lógica con la lógica del paquete Domain y separarlo.

Mecanismo de Persistencia

Se utilizó Entity Framework para implementar la persistencia de nuestra aplicación, apoyándonos en SQL Server 2014 Management Studio para la manipulación de la base de datos (tal y como fue pedido por la cátedra).

Acoplamiento a Entity Framework

El acoplarnos a Entity Framework nos obligó a cambiar algunas decisiones de diseño que habíamos tomado, detalladas a continuación:

- Imposibilidad de utilizar interfaces en el Domain: Entity Framework no mapea interfaces, por lo que nos vimos obligados a cambiar todo lo que sería una interfaz en el paquete Domain por una clase abstracta pura (que Entity Framework sí puede mapear).

- Properties públicas: Para que Entity Framework funcione correctamente y fuera capaz de mapear las entidades, todos los objetos del Domain debían tener Properties públicas.

- Constructores sin parámetros protected: Con el fin de que Entity Framework pudiese mapear las entidades, dicho Framework exige que todas las entidades deben tener constructores sin parámetros con un nivel de visibilidad protected o superior.

Privilegios de CashierRole

Ahora mismo, CashierRole cuenta con solo un privilegio (el pertinente a poder manejar a los clientes de la base de datos). Sin embargo, eso no significa que no pueda agregar pagos ni traer proveedores. Simplemente, todos los permisos que tendrán tanto un administrador como un cajero (con excepción del previamente mencionado) todavía no fueron especificados.
Al tener un mecanismo que permite agregar privilegios y roles de manera fácil y rápida (aunque se necesite volver a compilar nuevamente), optamos por esta decisión ya que el haberle colocado privilegios a la clase CashierRole (por ejemplo: un privilegio que le permita

agregar un pago) hubiera pecado de algo de genericidad especulativa (si en el futuro nos agregan un usuario que solamente pueda ver sus pagos, por ejemplo), pero además nos hubiera aumentado la complejidad del código de forma innecesaria en esta etapa del proyecto (al agregar un pago y similares operaciones que el cajero puede hacer, deberíamos haber agregado un control sobre si tiene el privilegio correspondiente).

Sobre las Excepciones

Se notará que para la mayoría de las excepciones se utilizó un throw de ArgumentException y en menor medida otras como InvalidOperationException, entre otras. Esto fue porque: primero, la excepción ArgumentException (u otras excepciones del sistema) se adecuaba correctamente al problema que estábamos tratando de controlar en todo momento y segundo, porque consideramos que la fabricación de excepciones propias y la complejidad que todo esto agrega no estaba justificada por el uso que le íbamos a dar a las mismas. Por esto se decidió a favor de utilizar las exepciones que brinda System.

Acoplamiento a OWin

Decidimos utilizar la especificación de OWin y Katana como su implementación ya que nos pareció que OAuth 2.0, que era la otra opción que podíamos llegar a implementar, le agregaba mucho más complejidad a nuestra solución (había que cambiar mucho de nuestro código para poder acoplarnos a ella), que la complejidad que OWin terminó agregando (casi nula). La especificación de OWin era suficiente para el objetivo al que queríamos llegar y además no encontramos diferencias en seguridad entre ambos protocolos de autenticación, por lo que decidimos utilizar OWin por sobre OAuth 2.0.

Patrón Strategy

Se hizo uso del patrón Strategy para poder lograr la mayor extensibilidad posible en la implementación del sistema de logueo de acciones. El uso de este patrón permite el agregar una nueva forma de persistir los datos del Log sin tener que modificar nada de lo ya existente. Además, el uso del patrón no solamente simplifica esto si no también el cambio entre los distintos mecanismos de persistencia que pudiesen existir.

Patrón Singleton en PointsManager

Con el fin de asegurarnos de mantener una única instancia de la clase PointsManager en todo momento en el sistema, se optó por implementar el patrón Singleton, que resuelve exactamente este problema.

AngularJS

Por pedido externo utilizamos el framework de AngularJS para desarrollar la página web que consumirá nuestra API.

Acoplamiento a AngularJS

El acoplamiento a dicho framework simplificó enormemente el trabajo necesario para comunicar nuestra API con nuestra página web sin obligarnos a hacer cambios en el sistema ya existente. Sin embargo, cabe resaltar que como nos acoplamos a la versión 1 de AngularJS, nuestro sistema podría quedar deprecated en un futuro cercano, en el cual habría que evaluar el costo de mudar es sistema a AngularJS 2 o a otro framework web.

Bootstrap

Decidimos hacer uso de Bootstrap para facilitarnos el modelado de la página web y de esa manera poder concentrarnos en brindar la funcionalidad pedida mientras a la vez logramos ofrecer un diseño fluido y moderno sin dedicarle mayor esfuerzo.

Acoplamiento a Bootstrap

El acoplamiento a Bootstrap solamente comprende el agregar un archivo .css y hacer referencia a él desde nuestra página web, dejando así abierta la posibilidad de poder usar las clases definidas por el framework en nuestro código HTML para lograr obtener un buen modelo de nuestra página web. Así, el acoplamiento a dicho framework tuvo un impacto mínimo.

Idioma de las interfaces de usuario

Si bien el código de la aplicación se encuentra en idioma inglés en toda su extensión, se decidió por utilizar el idioma español en todo lo que se le muestra al usuario (página web, la aplicación de Windows Form y mensajes de error) ya que la aplicación está claramente orientada al uso en Uruguay (requerimientos sobre cédula, por ejemplo).

Sobre los números de teléfono

En línea con el punto anterior, se optó por soportar únicamente teléfono uruguayos (ya sea de línea o celular). No se realizaron checkeos sobre las distintas compañías de celular uruguayas ni por número de región (interior o capital) y tampoco se tomaron en cuenta los números internacionales, para evitar así un aumento de la complejidad importante.

Sobre el checkeo de las cédulas

Como la validación sobre las cédulas uruguayas no está prevista que cambie en el futuro cercano, se decidió por dejar dicha validación en la clase Client.

Mecanismo de importación de productos

Se definió una clase Product y una interfaz IProductImporter, que son los dos elementos que un desarrollador ajeno a nuestro sistema necesita para poder proveernos de una nueva forma de importación de productos. Decidimos que para utilizar un .dll de terceros se debe seleccionar manualmente a través de un file explorer, ya que consideramos que la importación de productos no será un recurso usado tan usualmente como para necesitar colocar los .dll en una carpeta para que todas las veces que se use el programa se deban traer.

Uso de Reflection

Con el fin de poder importar los productos desde la aplicación de Windows Form y desde un .dll externo, se utilizó este mecanismo que nos permite acceder en tiempo de ejecución a dichos componentes externos a nuestra aplicación.

Mecanismos de importación de Productos provistos

XML

Se podrán cargar productos de archivos .xml. Los mismos se seleccionarán desde la interfaz de usuario. Deben tener el siguiente formato:

```
<?xml version="1.0" encoding="utf-8" ?>
<Productos>
        <Producto Nombre="Z">
        <Descripcion>texto</Descripcion>
        <CantidadPuntos>número entero</CantidadPuntos>
        <Stock>número entero</Stock>
        </Producto>
        <ProductoNombre="Y">
       ...
        </Producto>
 ...
</Productos>
```

Vale la pena resaltar que las tags “Descripcion”, “CantidadPuntos” y “Stock” no precisan ir en ese orden, pueden ir en cualquiera. Además, los espacios en blanco entre tags tampoco influencian. Ejemplo:

Este producto:

```
<Productos>
        <Producto Nombre="Z">
        <Descripcion>texto</Descripcion>

        <CantidadPuntos>número entero</CantidadPuntos>

        <Stock>número entero</Stock>
        </Producto>
</Productos>
```

Será igual a este:

```
<Productos>


        <Producto Nombre="Z">
        <Stock>número entero</Stock>
        <CantidadPuntos>número entero</CantidadPuntos>
        <Descripcion>texto</Descripcion>
        </Producto>


</Productos>
```

JSON

Se podrán cargar productos de archivos .json. Los mismos se seleccionarán desde la interfaz de usuario. Deben tener el siguiente formato:

```
{
	"Productos" : [
		{
			"Nombre" : texto,
			"Descripcion" : texto,
			"CantidadPuntos" : número entero,
			"Stock" : número entero
		},
		{
			"Nombre" : texto,
			"Descripcion" : texto,
			"CantidadPuntos" : número entero,
			"Stock" : número entero
		}
		...
	]
}
```

Vale la pena resaltar que las tags “Nombre”, “Descripcion”, “CantidadPuntos” y “Stock” no precisan ir en ese orden, pueden ir en cualquiera. Además, los espacios en blanco entre tags tampoco influencian. Ejemplo:

Este producto:

```
{


	"Productos" : [
		{
			"Nombre" : texto,

			"Descripcion" : texto,
			"CantidadPuntos" : número entero,
			"Stock" : número entero
		}
	]
}
```

Será igual a este:

```
{
	"Productos" : [
		{

			"CantidadPuntos" : número entero,
			"Nombre" : texto,
```

```
                    "Stock" : número entero,

                    "Descripcion" : texto



                }
        ]
}
```

Modelo de tablas de la estructura de la base de datos

Justificación de Clean Code

Nuestro obligatorio cumple con los estándares predefinidos en el libro “Clean Code” de Robert C. Martin. Aplicamos en todo momento la regla del Boy Scout (“Deja el lugar más limpio de lo que lo encontraste”) mientras desarrollábamos aplicando TDD, para poder asegurarnos de que nuestro código cumpiera con los estándares propuestos por el libro de Martin en sus capítulos 1-10 y 12, enumerados a continuación:

Nombres:

- Revelan intención.
- Evitan la desinformación.
- Hacen distinciones significativas.
- Son pronunciables.
- Son fáciles de buscar.
- No presentan “encodings” (notación húngara, prefijos).
- Los nombres de las clases no presentan verbos.
- Los nombres de los métodos presentan al menos un verbo.
- Refieren siempre igual al mismo concepto.
- Refieren al problema de la vida real (Ejemplo: Payment, Receipt, etc).
- Evitan mapas mentales.

Funciones:

- Son pequeñas (o lo más pequeñas posible, a veces no se pudo seguir el estándar purista pero igual se las mantuvo lo más pequeñas posible).
- Hacen una única cosa.
- No tienen demasiados argumentos (máximo 3 argumentos, con pocas excepciones a funciones con 4 argumentos).
- Tienen nombres descriptivos, que explican la función que cumple.
- Tiran excepciones en vez de retornar códigos de error.

Comentarios:

- No tenemos ninguno, ya que consideramos que las desventajas (redundacia, imprecisión, mantenimiento, ruido, entre otras) superaban ampliamente a la posible ventaja de poder explicar algo mediante un comentario.

Formato:

- Formato horizontal y vertical del código cuidados: no se poseen funciones largas ni en anchura (se realizaron saltos de línea en el caso de que alguna fuera muy larga) ni en longitud.

Objetos y estructuras de datos:

- Siempre se trató de cumplir con la Ley de Demeter: los objetos no hablan con extraños sino que únicamente con sus amigos más cercanos.

Manejo de errores:

- Se utilizan excepciones y no se retornan códigos de error.
- No hubieron pasajes de null por parámetro (excepto en los Tests, para poder probar justamente el manejo de errores).
- No se retornó null.

Tests Unitarios:

- Se cumplieron con las tres leyes de TDD (no escribir código de producción hasta que se tenga una prueba que falle; no escribir más que una prueba unitaria que falle; no escribir más código de producción que el suficiente para que la prueba pase) durante el desarrollo de todo el obligatorio.
- Clean Tests: los tests poseen variables con nombres que siguen los estándares de Clean Code, son concisos y testean lo que su nombre indica.
- Cumplen FIRST (son rápidas, repetibles, se autovalidan, independientes y escritas justo antes que el código de producción, o muy cerca de la construcción del mismo.

Clases:

- Son pequeñas (ninguna excede las 300 líneas de código, la mayoría tiene entre 50-150 líneas).
- Tienen un único eje de cambio.

A continuación algunas imágenes del código que demustran el seguimiento de los anteriores estándares:

- **Nombres adecuados al estándar:**

```
private IUnitOfWork unitOfWork;

5 referencias | Bruno Ferrari, Hace 7 días | 1 autor, 1 cambio
public LoginFacade(IUnitOfWork aUnitOfWork)
{
    CheckForNullUnitOfWork(aUnitOfWork);
    unitOfWork = aUnitOfWork;
}

1 referencia | Bruno Ferrari, Hace 7 días | 1 autor, 1 cambio
private void CheckForNullUnitOfWork(IUnitOfWork aUnitOfWork)
{
    if (aUnitOfWork == null) throw new ArgumentException();
}

4 referencias | Bruno Ferrari, Hace 7 días | 1 autor, 2 cambios
public User AdminLogin(string email, string password)
{
    IEnumerable<User> relatedUser = unitOfWork.UserRepository.Get(u => u.Email.Equals(email), null, "");
    CheckIfUserWasFound(relatedUser);
    CheckForCorrectPassword(relatedUser.First(), password);
    CheckIfUserHasRightRole(relatedUser.First());
    return relatedUser.First();
}

1 referencia | Bruno Ferrari, Hace 7 días | 1 autor, 1 cambio
private void CheckIfUserWasFound(IEnumerable<User> relatedUser)
{
    if (relatedUser.Count() == 0) throw new ArgumentException();
}
```

- **Funciones y Manejo de Errores adecuados al estándar:**

```
2 referencias | Diego Zuluaga, Hace 3 días | 1 autor, 1 cambio
private void MakeSureTargetAddressIsNotNullOrWhiteSpace(string targetAddress)
{
    if (string.IsNullOrWhiteSpace(targetAddress))
    {
        throw new ArgumentException("La dirección de un cliente no puede ser vacía");
    }
}

2 referencias | Diego Zuluaga, Hace 12 días | 1 autor, 1 cambio
private void MakeSureTargetNameIsNotNullOrWhiteSpace(string targetName)
{
    if (string.IsNullOrWhiteSpace(targetName))
    {
        throw new ArgumentException("El nombre de un cliente no puede ser vacío");
    }
}

2 referencias | Diego Zuluaga, Hace 9 días | 1 autor, 2 cambios
private void MakeSureTargetPhoneNumberIsValid(string targetPhoneNumber)
{
    if (TargetPhoneNumberIsInvalid(targetPhoneNumber))
    {
        throw new ArgumentException("El número de teléfono del cliente no es válido");
    }
}
```

- **Tests adecuados al estándar:**

```
[TestMethod]
0 referencias | Diego Zuluaga, Hace 5 días | 1 autor, 1 cambio
public void BeAbleToReturnSingleClientInRepository()
{
    Client singleClient = TEST_CLIENTS.First();
    IHttpActionResult actionResult = CONTROLLER.GetClient(singleClient.ID);
    OkNegotiatedContentResult<Client> contentResult = (OkNegotiatedContentResult<Client>)actionResult;

    Assert.AreEqual(contentResult.Content, singleClient);
}

[TestMethod]
0 referencias | Diego Zuluaga, Hace 5 días | 1 autor, 1 cambio
public void FailWithNotFoundIfSingleClientIdDoesntExistInRepository()
{
    IHttpActionResult actionResult = CONTROLLER.GetClient(0);
    Assert.IsInstanceOfType(actionResult, typeof(NotFoundResult));
}
```

- **Estándar de objetos y estructuras de datos cumplido (Ley de Demeter):**

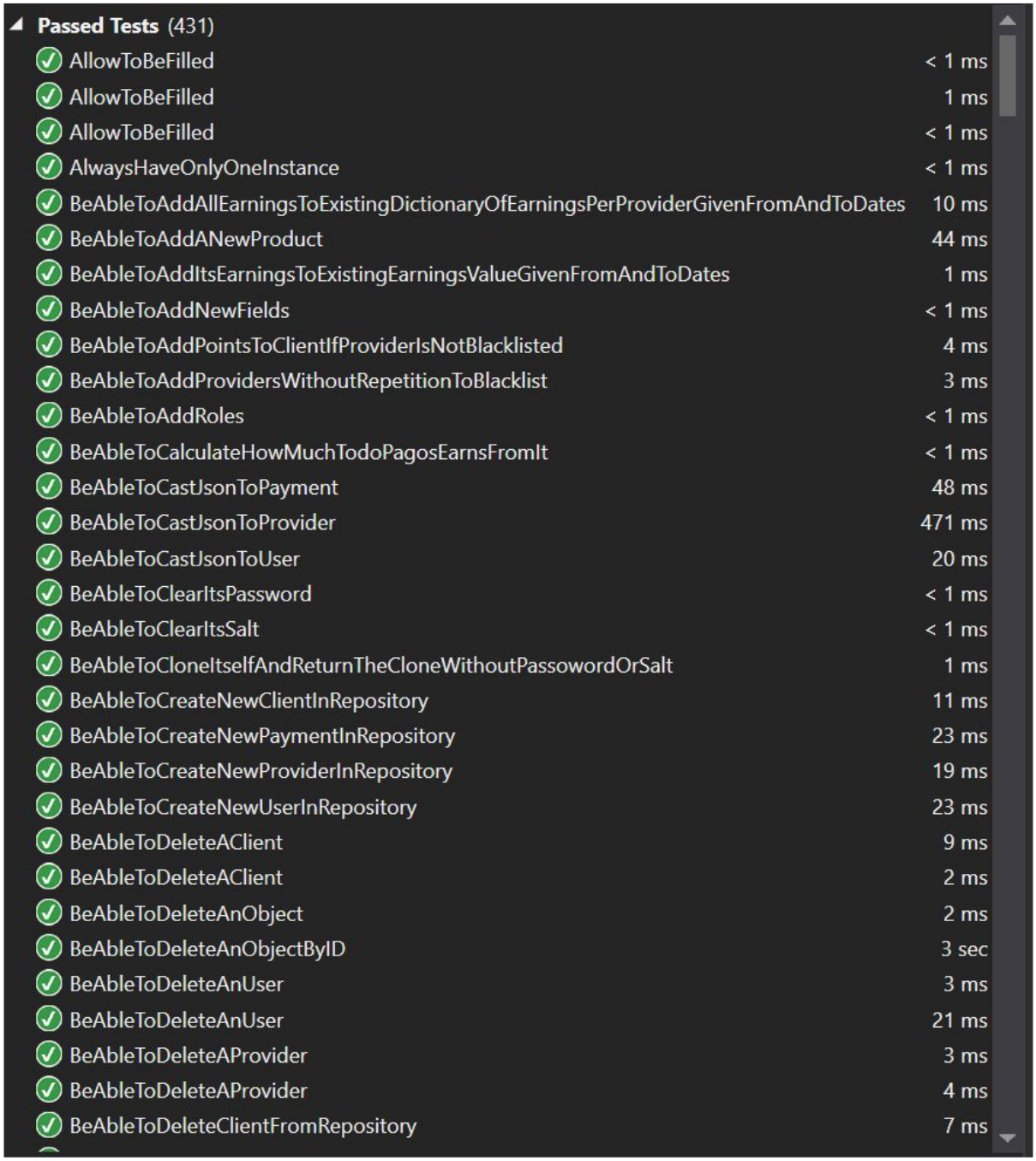
```
private void CheckForEmptyCompletedField(ICollection<IField> completedFields)
{
    foreach (IField oneField in completedFields)
    {
        if (oneField.IsEmpty())
        {
            throw new ArgumentException();
        }
    }
}
```

Sobre los nombres de las pruebas:

Decidimos utilizar un approach bastante innovador que descubrimos en Internet al buscar información sobre equipos de trabajo que utilizan TDD, que consiste en definir el nombre de la clase de prueba como <EntidadAProbar>Should y el nombre de cada método de prueba como BeAbleTo…,FailWhen…, Return…, entre otros. De esta manera, si alguien del equipo lee primero el nombre de la clase de prueba y luego el nombre del test, se obtiene una frase que indica exactamente cuál es la función que se introdujo a partir de este test. A continuación, un ejemplo de esto en nuestro obligatorio:

Resultados de la ejecución de las pruebas

Nuestro obligatorio posee 431 pruebas unitarias:

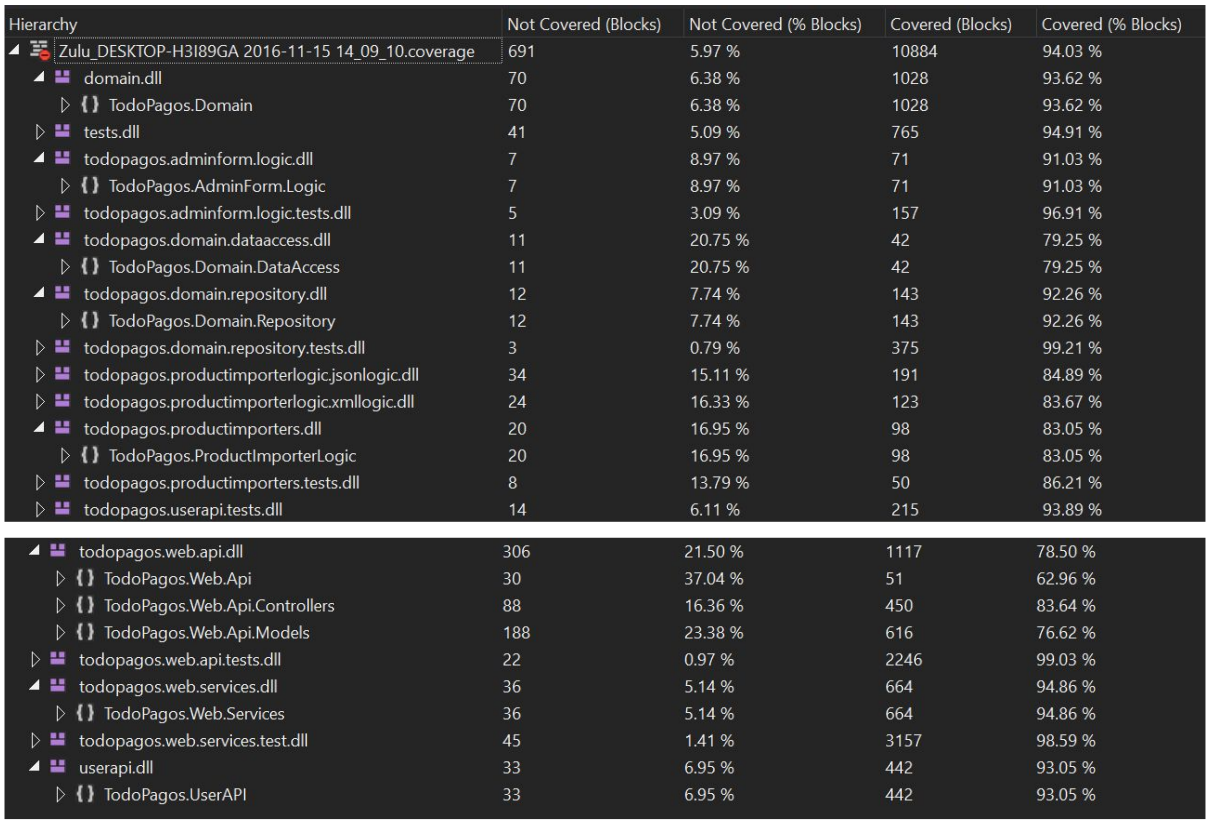


Se omitirá la lista específica que detalla cada test y su nombre ya que al ser más de 400 quedaría extremadamente extensa.
Las 431 pruebas anteriores se encuentran en diversos paquetes de prueba:

- TodoPagos.Domain.Tests
- TodoPagos.Domain.Repository.Tests
- TodoPagos.UserAPI.Tests
- TodoPagos.Web.API.Tests
- TodoPagos.Web.API.Tests.ControllerIntegrationTests
- TodoPagos.Web.API.Tests.ControllerUnitTests
- TodoPagos.Web.API.Tests.ModelBinderTests
- TodoPagos.Web.Services.Tests
- TodoPagos.AdminForm.Logic.Tests
- TodoPagos.ProductImporterLogic.Tests

Cobertura del Código

El análisis efectuado por la herramienta de análisis de cobertura de código que incluye Visual Studio 2015 en su versión Enterprise arrojó el siguiente resultado:



Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
Zulu_DESKTOP-H3I89GA 2016-11-15 14_09_10.coverage	691	5.97 %	10884	94.03 %
domain.dll	70	6.38 %	1028	93.62 %
TodoPagos.Domain	70	6.38 %	1028	93.62 %
tests.dll	41	5.09 %	765	94.91 %
todopagos.adminform.logic.dll	7	8.97 %	71	91.03 %
TodoPagos.AdminForm.Logic	7	8.97 %	71	91.03 %
todopagos.adminform.logic.tests.dll	5	3.09 %	157	96.91 %
todopagos.domain.dataaccess.dll	11	20.75 %	42	79.25 %
TodoPagos.Domain.DataAccess	11	20.75 %	42	79.25 %
todopagos.domain.repository.dll	12	7.74 %	143	92.26 %
TodoPagos.Domain.Repository	12	7.74 %	143	92.26 %
todopagos.domain.repository.tests.dll	3	0.79 %	375	99.21 %
todopagos.productimporterlogic.jsonlogic.dll	34	15.11 %	191	84.89 %
todopagos.productimporterlogic.xmllogic.dll	24	16.33 %	123	83.67 %
todopagos.productimporters.dll	20	16.95 %	98	83.05 %
TodoPagos.ProductImporterLogic	20	16.95 %	98	83.05 %
todopagos.productimporters.tests.dll	8	13.79 %	50	86.21 %
todopagos.userapi.tests.dll	14	6.11 %	215	93.89 %
todopagos.web.api.dll	306	21.50 %	1117	78.50 %
TodoPagos.Web.Api	30	37.04 %	51	62.96 %
TodoPagos.Web.Api.Controllers	88	16.36 %	450	83.64 %
TodoPagos.Web.Api.Models	188	23.38 %	616	76.62 %
todopagos.web.api.tests.dll	22	0.97 %	2246	99.03 %
todopagos.web.services.dll	36	5.14 %	664	94.86 %
TodoPagos.Web.Services	36	5.14 %	664	94.86 %
todopagos.web.services.test.dll	45	1.41 %	3157	98.59 %
userapi.dll	33	6.95 %	442	93.05 %
TodoPagos.UserAPI	33	6.95 %	442	93.05 %

Como puede observarse, se obtuvo un 94.03% de cobertura de código por sobre todos los paquetes del sistema (incluyendo los paquetes de prueba).

La cobertura de los paquetes que interesa probar (es decir, de los paquetes que no contienen pruebas unitarias) fue la siguiente:

- TodoPagos.Domain: 93.62%
- TodoPagos.Domain.DataAccess: 79.25%
- TodoPagos.Domain.Repository: 92.26%
- TodoPagos.UserAPI: 93.05%
- TodoPagos.Web.Api: 62.96%
- TodoPagos.Web.Api.Controllers: 83.64%

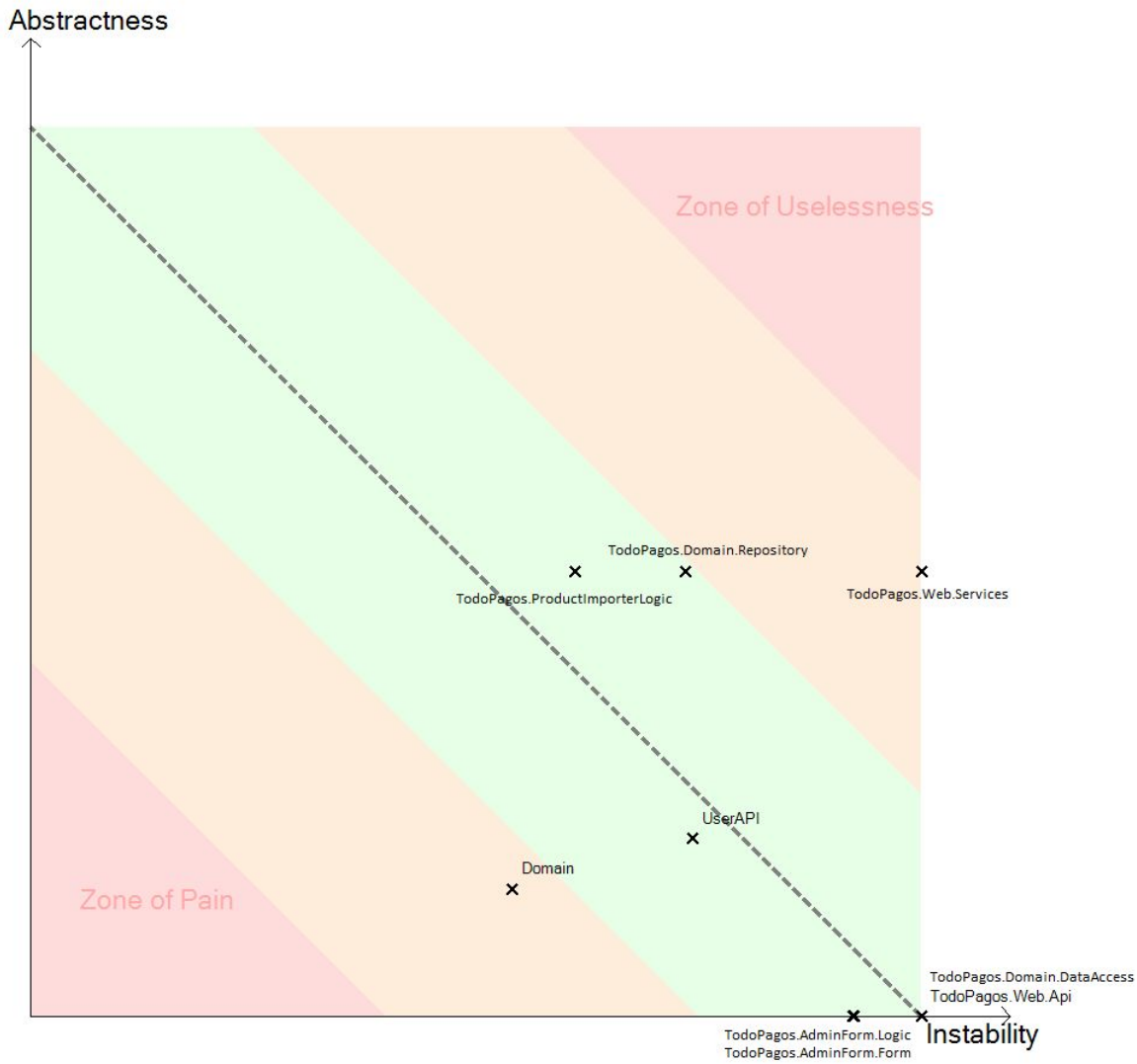
- TodoPagos.Web.Api.Models: 76.62%
- TodoPagos.Web.Services: 94.86%
- TodoPagos.AdminForm.Logic: 91.03%
- TodoPagos.ProductImporterLogic: 83.05%

El análisis que hacemos sobre estos resultados es el siguiente: si bien se adoptó TDD en su modalidad Tests First para el desarrollo del sistema, llegar al 100% de la cobertura de código es un objetivo que conlleva un esfuerzo desproporcionadamente más grande que el que conlleva apuntar a un 90% o 95% de cobertura de código. De todos modos, nuestro objetivo siempre fue apuntar a un 100% de la cobertura de código ya que toda nueva funcionalidad era introducida por una prueba unitaria. Sin embargo, no se pudo llegar a este objetivo (aunque sí al de sobrepasar holgadamente el 90% de cobertura de código) debido a determinadas funciones que eran imposibles (o extremadamente difíciles y rebuscadas) de probar. A modo de ejemplo, los constructores sin parámetros de los Controllers en el paquete TodoPagos.Web.Api.Controllers que son los utilizados al ejecutar la aplicación en modo Release no fueron probados ya que en su interior hacen una llamada a un método de HTTPContext, y al no poder simular mediante un mock esta llamada no pudimos hacer prueba unitaria que probara el correcto funcionamiento de los mismos (se notará que en las clases se crearon constructores “de prueba” para poder probar el resto de la funcionalidad de los Controllers).

Reporte de Métricas

Utilizando la herramienta NDepend logramos obtener determinadas métricas sobre nuestro obligatorio. Aclaramos que para reducir la complejidad del análisis, no tuvimos en cuenta los paquetes de Test.

Gráfica de Abstracción en función de Inestabilidad



Como puede observarse en la gráfica anterior, nuestro obligatorio posee solamente dos componentes que se encuentran por fuera de la zona deseable: Domain y TodoPagos.Web.Services. Esto se debe, en el caso de Domain, porque el mismo posee pocos elementos abstractos mientras a la vez es referenciado y él referencia también a otros componentes.

En el caso de TodoPagos.Web.Services, el mismo se encuentra allí ya que si bien posee elementos abstractos, el mismo depende de muchos otros componentes y se podría verse afectado en el caso de que alguno de estos componentes de los cuales él depende cambie. Esto se podría mejorar haciendo que las clases de este paquete dependan de abstracciones en vez de clases concretas.
El resto de los componentes se encuentran en la zona deseable, por lo que consideramos bastante correcto nuestro obligatorio dado resultado arrojado por la herramienta en esta parte.

Cohesión - Inestabilidad - Abstracción

Nombre del Assembly	Cohesión	Inestabilidad	Abstracción
Domain	1	0.52	0.14
UserAPI	2	0.66	0.2
TodoPagos.ProductImporterLogic	1	0.61	0.5
TodoPagos.Domain.DataAccess	1	0.8	0
TodoPagos.Domain.Repository	1.5	0.66	0.5
TodoPagos.Web.Services	0.6	0.91	0.5
TodoPagos.Web.Api	1.04	1	0
TodoPagos.AdminForm.Logic	0.5	0.93	0
TodoPagos.AdminForm.Form	0.89	1	0

NOTA: En la métrica de Cohesión, tomaremos como aceptable un resultado de 1.5 a 4.0.

Primero analizaremos los resultados de la métrica de Cohesión. Aquí todos los assemblies excepto UserAPI y TodoPagos.Domain.Repository dieron un resultado por debajo de lo aceptable. Esto quiere decir que las clases dentro de nuestros paquetes no se relacionan demasiado entre sí. Esto podría solucionarse creando nuevos paquetes y moviendo dichas clases a esos nuevos paquetes. Esto agregaría mucha complejidad a nuestra solución de todas maneras, ya que ya contamos con más de 20 paquetes (incluídos los paquetes de Test).

Pasaremos ahora a analizar el resultado de la métrica de Estabilidad. Aquellos paquetes con un valor cercano o igual a 1 (TodoPagos.Web.Services, TodoPagos.Web.Api, TodoPagos.AdminForm.Logic, TodoPagos.AdminForm.Form) son muy inestables porque dependen de muchos otros paquetes. Sin embargo, son fáciles de cambiar ya que impactan en pocos paquetes. El resto de los paquetes se encuentran en la zona media (entre 0.5 y

0.8), lo que significa que sí dependen de otros paquetes pero a la vez otros paquetes dependen de ellos (el impacto de cambio en estos paquetes puede ser alto ya que hay otros paquetes que dependen de los mismos).

Finalmente, analizaremos el resultado de la métrica de Abstracción. Aquí simplemente notar que tenemos 4 paquetes que son completamente concretos: TodoPagos.Web.Api, TodoPagos.Domain.DataAccess, TodoPagos.AdminForm.Logic, TodoPagos.AdminForm.Form, mientras que el resto de los paquetes contienen elementos abstractos.

Referencias

[1] R. Martin, *Clean code*. Upper Saddle River, NJ: Prentice Hall, 2009.

[2]C. Ltd, "Naming Test Classes and Methods | Codurance", *Codurance*, 2016. [Online]. Available: https://codurance.com/2014/12/13/naming-test-classes-and-methods/. [Accessed: 10- Oct- 2016].

[3]E. Gamma, *Design patterns*. Reading, Mass.: Addison-Wesley, 1995.

[4] T.A. Halpin and H.A. Proper, Subtyping and Polymorphism in Object-Role Modelling, Data & Knowledge Engineering, 15(3), 251–281, 1995. [Online]. Available: http://www.orm.net/pdf/subpoly.pdf. [Accessed: 08- Oct- 2016].

[5] Dr. Terry Halpin, BSc, DipEd, BA, MLitStud, PhD, Subtyping: conceptual and logical issues. Database Newsletter vol 23, no 6., 1995 [Online]. Available: http://www.orm.net/pdf/subtype.pdf. [Accessed: 10- Oct- 2016].

[6]"Prototype (patrón de diseño)", *Es.wikipedia.org*, 2016. [Online]. Available: https://es.wikipedia.org/wiki/Prototype_(patr%C3%B3n_de_dise%C3%B1o). [Accessed: 11- Oct- 2016].

[7]"ORT-DA2/Clase-4", *GitHub*, 2016. [Online]. Available: https://github.com/ORT-DA2/Clase-4. [Accessed: 14- Oct- 2016].

[8] “Métricas”, Aulas - Universidad ORT Uruguay, 2016. [Online]. Available: https://aulas.ort.edu.uy/pluginfile.php/215431/mod_resource/content/0/Metricas.pdf. [Accessed: 15- Nov- 2016].

Anexo

Datos de Prueba

Tabla Users:

	ID	Name	Email	Password	Salt
1	1	Brulu	diego@bruno.com	[illegible]	[illegible]
2	2	Eric	eric@ort.com	[illegible]	[illegible]
3	3	Nacho	ignacio.valle@gmail.com	[illegible]	[illegible]
4	4	Gabriel	gabo@ort.com	[illegible]	[illegible]

Nota: Todos estos usuarios tienen contraseña “HOLA1234”

Tabla UserRoles:

	User_ID	Role_ID
1	1	1
2	3	1
3	2	2
4	3	2
5	4	2

Tabla Roles:

	ID	Name	Discriminator
1	1	Admin	AdminRole
2	2	Cajero	CashierRole

Tabla RolePrivileges:

	Role_ID	Privilege_ID
1	1	1
2	1	2
3	1	3
4	1	4
5	2	4

Tabla Receipts:

	ID	Amount	ReceiptProvider_ID	Payment_ID
1	1	250	1	1
2	2	200	2	2
3	3	100	3	2
4	4	800	4	3
5	5	800	5	4

Tabla Providers:

	ID	Commission	Name	Active	PointsManager_ID
1	1	10	OSE	0	NULL
2	2	15	Antel	1	NULL
3	3	20	Ancap	1	NULL
4	4	50	Banco Central	1	NULL
5	5	90	OSE	1	NULL

Tabla Privileges:

	ID	Name	Discriminator
1	1	User Managment	UserManagementPrivilege
2	2	Register Provider	ProviderManagementPrivilege
3	3	Earning Queries	EarningQueriesPrivilege
4	4	Client Manager	ClientManagementPrivilege

Tabla PayMethods:

	ID	PayDate	Type	Discriminator
1	1	2016-11-15 16:39:00.000	DebitPayMethod	DebitPayMethod
2	2	2016-11-15 16:42:00.000	CashPayMethod	CashPayMethod
3	3	2016-11-15 16:43:00.000	DebitPayMethod	DebitPayMethod
4	4	2012-01-20 17:43:00.000	CashPayMethod	CashPayMethod

Tabla Payments:

	ID	PaidWith	Change	Total	PaymentMethod_ID
1	1	250	0	250	1
2	2	500	200	300	2
3	3	800	0	800	3
4	4	1000	200	800	4

Tabla LogEntries:

	ID	Action	Date	UserEmail
1	1	LOGIN	2016-11-15 19:34:13.163	diego@bruno.com
2	2	LOGIN	2016-11-15 19:34:27.253	diego@bruno.com
3	3	LOGIN	2016-11-15 19:49:12.793	diego@bruno.com
4	4	LOGIN	2016-11-15 19:49:33.690	diego@bruno.com
5	5	LOGIN	2016-11-15 19:52:16.527	diego@bruno.com
6	6	LOGIN	2016-11-15 19:53:25.237	diego@bruno.com

Tabla IFields:

	ID	Name	Type	Data	Data1	Data2	Discriminator	Receipt_ID	Provider_ID
1	1	Fecha	DateField	NULL	NULL	NULL	DateField	NULL	1
2	2	RUT	NumberField	NULL	NULL	NULL	NumberField	NULL	2
3	3	Nombre Lugar	TextField	NULL	NULL	NULL	TextField	NULL	3
4	4	Nombre Sucursal	TextField	NULL	NULL	NULL	TextField	NULL	4
5	5	Código	NumberField	NULL	NULL	NULL	NumberField	NULL	4
6	6	Fecha	DateField	2015-11-15 14:24:30.000	NULL	NULL	DateField	1	NULL
7	7	Fecha Registro	DateField	NULL	NULL	NULL	DateField	NULL	5
8	8	Nombre Empleado	NumberField	NULL	NULL	NULL	NumberField	NULL	5
9	9	RUT	NumberField	NULL	123	NULL	NumberField	2	NULL
10	10	Nombre Lugar	TextField	NULL	NULL	Piedras Blancas	TextField	3	NULL
11	11	Nombre Sucursal	TextField	NULL	NULL	La Blanqueada	TextField	4	NULL
12	12	Código	NumberField	NULL	123	NULL	NumberField	4	NULL
13	13	Fecha Registro	DateField	2012-05-15 14:24:30.000	NULL	NULL	DateField	5	NULL
14	14	Nombre Empleado	NumberField	NULL	1234	NULL	NumberField	5	NULL

Tabla Clients:

	ID	Name	IDCard	PhoneNumber	Points	Address
1	1	Gustavo	11111111	26015909	0	Calle 1234
2	2	Alicia	49018830	099898909	0	Hola 123
3	3	Pablo	12345672	23456789	0	Algun Lugar 1