

Diseño de Aplicaciones 2

Primer Obligatorio



Bruno Ferrari – N°: 167709



Diego Zuluaga – N°: 173642

Docentes: Ignacio Valle y Gabriel Piffaretti

Grupo: M6B ID

Fecha: Octubre 2016

Abstract

Trabajo realizado por Bruno Ferrari y Diego Zuluaga como primer obligatorio de la materia Diseño de Aplicaciones 2. El sistema trata de una API Rest que ofrece las operaciones necesarias para el manejo de un local de cobranzas. El obligatorio fue construido 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).

Índice

<u>Abstract</u>	2
<u>Manual de Usuario</u>	5
<u>Descripción del Diseño Propuesto</u>	23
<u>Diagramas de Clases</u>	23
<u>Diagrama de Paquetes</u>	31
<u>Diagramas de Interacción</u>	32
<u>Diagrama de Componentes</u>	37
<u>Diagrama de Entrega</u>	38
<u>Justificación de las decisiones de Diseño</u>	39
<u>Modelo de tablas de la estructura de la base de datos</u>	45
<u>Justificación de Clean Code</u>	46
<u>Resultados de la ejecución de las pruebas</u>	50
<u>Referencias</u>	53
<u>Anexo</u>	54

Manual de Usuario

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

Antes de probar la api acuerdese que debe primero cambiar el *connection string* del archivo web.config de web api, y segundo debe montar la api de manera satisfactoria.

Sin más preambulo, empecemos!

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

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:

POST ▼ http://localhost:puertoDeLaApi/api/v1/login Params Send ▼

Authorization Headers (1) **Body** Pre-request Script Tests

☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary

key	value
grant_type	password
username	-SuEmail-
password	-SuContraseña-

- 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":
  "uTqdBClTGsF5ldDFTwZIo7R0U0Qrsa4W2MMoxp8Phystr0oFffqIERPgZ9re4-zPGjlj3u9-Z3azQsqxI35cq6GNR8
  7HrGmbDXozmTg1vXvbs2SxhmNXuFAf_Q9N2jirJ0lRYMVg_YaD_SSsOX0jOL-43drmb7_K0lB2Pw73HSwhp7c0pIrhl
  UsaRrRg_NBPau1yY8Gw33FL0r3wD0c70yMvVq6lYWKSxMPbbjG-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:

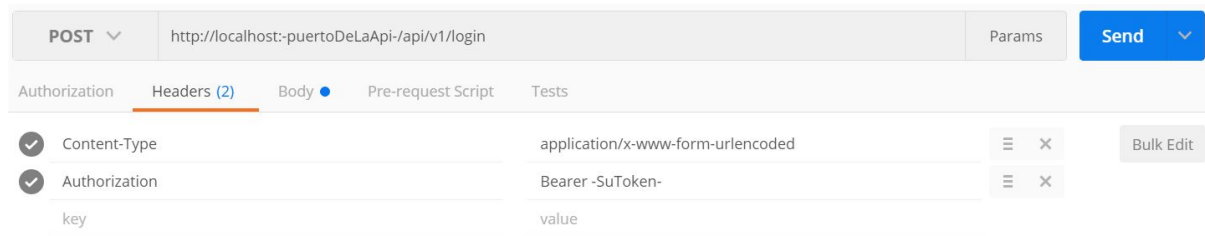
Status: 400 (Bad Request)

```
{
  "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 acuérdesse 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.

Acuérdesse 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.

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-,
  "Active" : -Si esta eliminado o no-,
  "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
- La comision 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",
  "Active" : true,
  "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 el proveedor está marcado como eliminado/desactivado:
 - Status: 400 (Bad Request)
- Si la comision 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-,
    "Active" : -Si esta eliminado o no-,
    "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 la comisión, debe poner un número negativo o por arriba de 100, o debe poner el valor anterior. Recuerde que no puede actualizar un proveedor 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",
    "Active" : true,
    "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 nuevo proveedor o 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-,
    "ReceiptProvider" : {
        "ID" : -La Id del proveedor-,
        "Commission" : -Comission del proveedor-,
        "Name" : -Nombre de proveedor-,
        "Active" : -Estado Proveedor-,
        "Fields" : [{
            "Type" : -Tipo de Field-,
            "Name" : -Nombre Field-
        }]
    },
    "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 ddd, dd MMM yyyy HH:mm:ss GMT" o "ddd, d MMM yyyy HH:mm:ss GMT")

EJEMPLO:

```
{
  "AmountPaid" : 250,
  "PayMethod" : {
    "Type" : "DebitPayMethod",
    "PayDate" : "Mon, 15 Sep 2008 09:30:41 GMT"
  },
  "Receipts" : [{
    "Amount" : 250,
    "ReceiptProvider" : {
      "ID" : 1,
      "Commission" : 2,
      "Name" : "Antel",
      "Active" : true
      "Fields" : [{
        "Type" : "NumberField",
        "Name" : "Cedula"
      }]
    },
    "CompletedFields" : [{
      "Type" : "NumberField",
      "Data" : "4901883"
      "Name" : "Cedula"
    }]
  }]
}

{
  "AmountPaid" : 250,
  "PayMethod" : {
    "Type" : "DebitPayMethod",
    "PayDate" : "Mon, 15 Sep 2008 09:30:41 GMT"
  },
  "Receipts" : [{
    "Amount" : 250,
    "ReceiptProvider" : {
      "ID" : 3,
      "Commission" : 20,
      "Name" : "Antel",
```

```

        "Active" : true,
        "Fields" : [{
            "Type" : "NumberField",
            "Name" : "Cedula"
        }, {
            "Type" : "DateField",
            "Name" : "Fecha"
        }, {
            "Type" : "TextField",
            "Name" : "Nombre"
        }
    ],
    "CompletedFields" : [{
        "Data" : "4901883",
        "Type" : "NumberField",
        "Name" : "Cedula"
    }, {
        "Data" : "Wed, 19 Oct 2016 09:30:41 GMT",
        "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=ddd,%20dd%20MMM%20yyyy%20HH:mm:ss%20GMT`
- `/api/v1/query/earnings/allEarnings?from=ddd,%20dd%20MMM%20yyyy%20HH:mm:ss%20GMT`
- `/api/v1/query/earnings/allEarnings?from=ddd,%20dd%20MMM%20yyyy%20HH:mm:ss%20GMT&to=ddd,%20dd%20MMM%20yyyy%20HH:mm:ss%20GMT`
- `/api/v1/query/earnings/allEarnings`

Las fechas default son to=(la fecha de hoy) y from=DateTime.Min (1/1/0001 12:00:00 AM)

El formato para pasarle las fechas es:

- ❖ `ddd, dd MMM yyyy HH:mm:ss GMT`
- ❖ `ddd, d MMM yyyy HH:mm:ss GMT`

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=ddd,%20dd%20MMM%20yyyy%20HH:mm:ss%20GMT`
- `/api/v1/query/earnings/earningsPerProvider?from=ddd,%20dd%20MMM%20yyyy%20HH:mm:ss%20GMT`
- `/api/v1/query/earnings/earningsPerProvider?from=ddd,%20dd%20MMM%20yyyy%20HH:mm:ss%20GMT&to=ddd,%20dd%20MMM%20yyyy%20HH:mm:ss%20GMT`
- `/api/v1/query/earnings/earningsPerProvider`

Las fechas default son to=(la fecha de hoy) y from=DateTime.Min (1/1/0001 12:00:00 AM)

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

- ❖ `ddd, dd MMM yyyy HH:mm:ss GMT`
- ❖ `ddd, d MMM yyyy HH:mm:ss GMT`

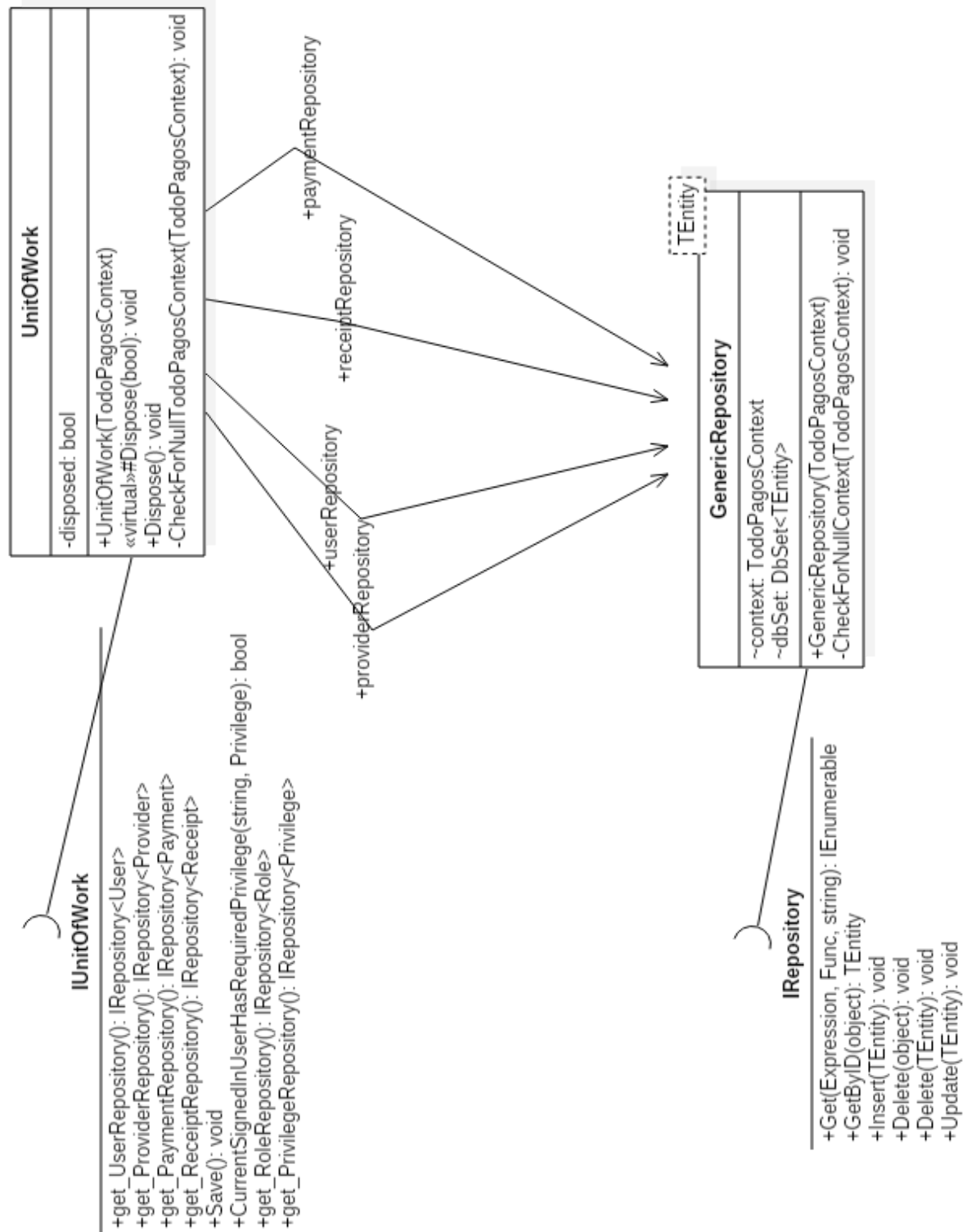
TodoPagos.Domain



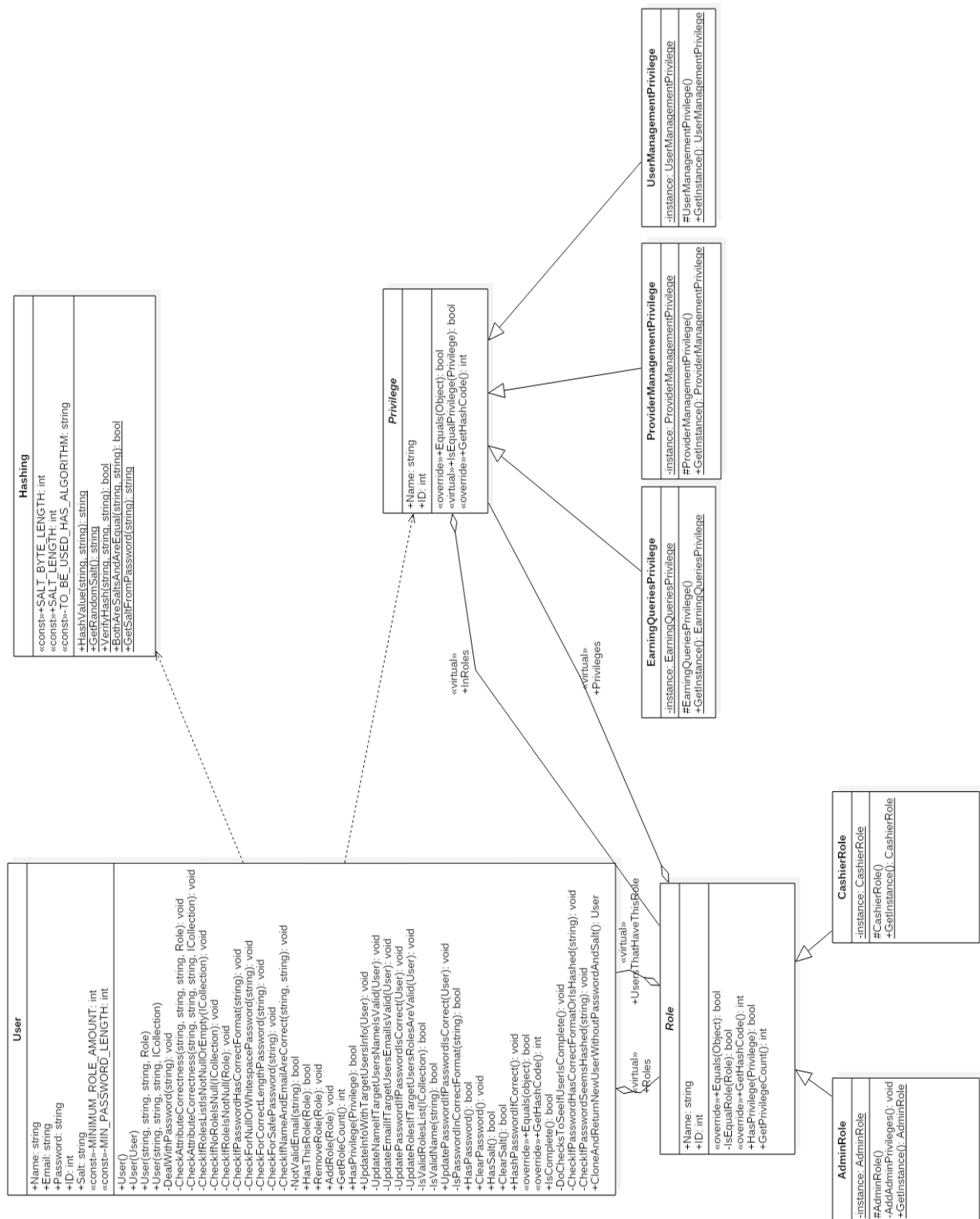
TodoPagos.Domain.DataAccess

TodoPagosContext
+Fields: DbSet<IField> +Receipts: DbSet<Receipt> +Payments: DbSet<Payment> +Users: DbSet<User> +Providers: DbSet<Provider> +PayMethods: DbSet<PayMethod> +Roles: DbSet<Role> +Privileges: DbSet<Privilege>
+TodoPagosContext() «override»#OnModelCreating(DbModelBuilder): void

TodoPagos.Domain.Repository



TodoPagos.UserAPI



TodoPagos.Web.Api

Startup
+Configuration(IAppBuilder): void +ConfigureOAuth(IAppBuilder): void

WebApiConfig
+Register(HttpConfiguration): void

CredentialsAuthorizationServerProvider
«override»+ValidateClientAuthentication(OAuthValidateClientAuthenticationContext): Task «override»+GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext): Task

TodoPagos.Web.Api.Controllers

EarningQueriesController
<pre>«readonly»-earningQueriesService: IEarningQueriesService «readonly»-DEFAULT_FROM_DATE: DateTime «readonly»-DEFAULT_TO_DATE: DateTime «readonly»-signedInUsername: string +EarningQueriesController() +EarningQueriesController(IEarningQueriesService) +EarningQueriesController(string) -FailIfUsernameArgumentIsNullOrEmpty(string): void -CheckForNullOrEmptyEarningQueriesService(IEarningQueriesService): void +GetEarningsPerProvider(DateTime?, DateTime?): IHttpActionResult -TryToProcessRequestToGetEarningsPerProvider(DateTime?, DateTime?): IHttpActionResult -AssignFromDateInCaseOfNullOrEmpty(DateTime?): DateTime? -AssignToDateInCaseOfNullOrEmpty(DateTime?): DateTime? +GetAllEarnings(DateTime?, DateTime?): IHttpActionResult -TryToProcessRequestToGetAllEarnings(DateTime?, DateTime?): IHttpActionResult «override»#Dispose(bool): void</pre>

PaymentsController
<pre>«readonly»-paymentService: IPaymentService «readonly»-signedInUsername: string +PaymentsController() +PaymentsController(IPaymentService) +PaymentsController(string) -FailIfUsernameArgumentIsNullOrEmpty(string): void -CheckForNullOrEmptyPaymentService(IPaymentService): void +GetPayments(): IHttpActionResult +GetPayment(int): IHttpActionResult +PostPayment(Payment): IHttpActionResult -TryToCreatePayment(Payment): IHttpActionResult «override»#Dispose(bool): void</pre>

ProvidersController
<pre>«readonly»-providerService: IProviderService «readonly»-signedInUsername: string +ProvidersController() +ProvidersController(IPProviderService) +ProvidersController(string) -FailIfUsernameIsNullOrEmpty(string): void -MakeSureProviderProviderServiceIsNotNull(IPProviderService): void +GetProviders(bool): IHttpActionResult +GetProviders(): IHttpActionResult +GetProvider(int): IHttpActionResult +PutProvider(int, Provider): IHttpActionResult -TryToUpdateProvider(int, Provider): IHttpActionResult -DecideWhatErrorMessageToReturn(int, Provider): IHttpActionResult +PostProvider(Provider): IHttpActionResult -TryToCreateProviderWhileCheckingForArgumentNullException(Provider): IHttpActionResult -TryToCreateProviderWhileCheckingForInvalidOperationException(Provider): IHttpActionResult -TryToCreateProviderWhileCheckingForArgumentException(Provider): IHttpActionResult -TryToCreateProviderWhileCheckingForUnauthorizedAccessException(Provider): IHttpActionResult +DeleteProvider(int): IHttpActionResult «override»#Dispose(bool): void</pre>

UserController
<pre>«readonly»-userService: IUserService «readonly»-signedInUsername: string +UserController() +UserController(IPUserService) +UserController(string) -FailIfUsernameArgumentIsNullOrEmpty(string): void -FailIfServiceArgumentIsNullOrEmpty(IPUserService): void +GetUsers(): IHttpActionResult -ClearPasswordsAndSaltsFromNewInstanceOfTargetUsers(IEnumerable): IEnumerable +GetUser(int): IHttpActionResult +PostUser(User): IHttpActionResult -TryToCreateUserWhileCheckingForArgumentNullAndArgumentOutOfRangeException(User): IHttpActionResult -TryToCreateUserWhileCheckingForInvalidOperationException(User): IHttpActionResult +PutUser(int, User): IHttpActionResult -TryToUpdateUser(int, User): IHttpActionResult -DecideWhatErrorMessageToReturn(int, User): IHttpActionResult +DeleteUser(int): IHttpActionResult «override»#Dispose(bool): void</pre>

TodoPagos.Web.Api.Models

PaymentModelBinder
«readonly»-ACCEPTED_DATE_FORMATS: string
+PaymentModelBinder() +BindModel(HttpContext, ModelBindingContext): bool -ParseJsonObjectAndReturnAsDynamic(HttpContext): dynamic -ParsePaymentFromJsonParameters(dynamic): Payment -ParsePayMethodFromJsonParameters(dynamic): PayMethod -ParseToGMTDate(string): DateTime -ParseReceiptsFromJsonArray(JArray): ICollection -ParseReceiptProviderFromJsonParameters(dynamic): Provider -ParseProviderFieldsFromJsonArray(JArray): ICollection -ParseCompletedFieldsFromJsonArray(JArray): ICollection

ProviderModelBinder
+ProviderModelBinder() +BindModel(HttpContext, ModelBindingContext): bool -ParseJsonObjectAndReturnAsDynamic(HttpContext): dynamic -ParseProviderFromJsonParameters(dynamic): Provider -ParseProviderFieldsFromJsonArray(JArray): ICollection

UserModelBinder
+UserModelBinder() +BindModel(HttpContext, ModelBindingContext): bool -ParseJsonObjectAndReturnAsDynamic(HttpContext): dynamic -ParseUserFromJsonParameters(dynamic): User -FillRolesListUsingClassesNamesInJson(JArray): ICollection -AddIDToParsedUserIfNeeded(HttpContext, dynamic, User): void

TodoPagos.Web.Services

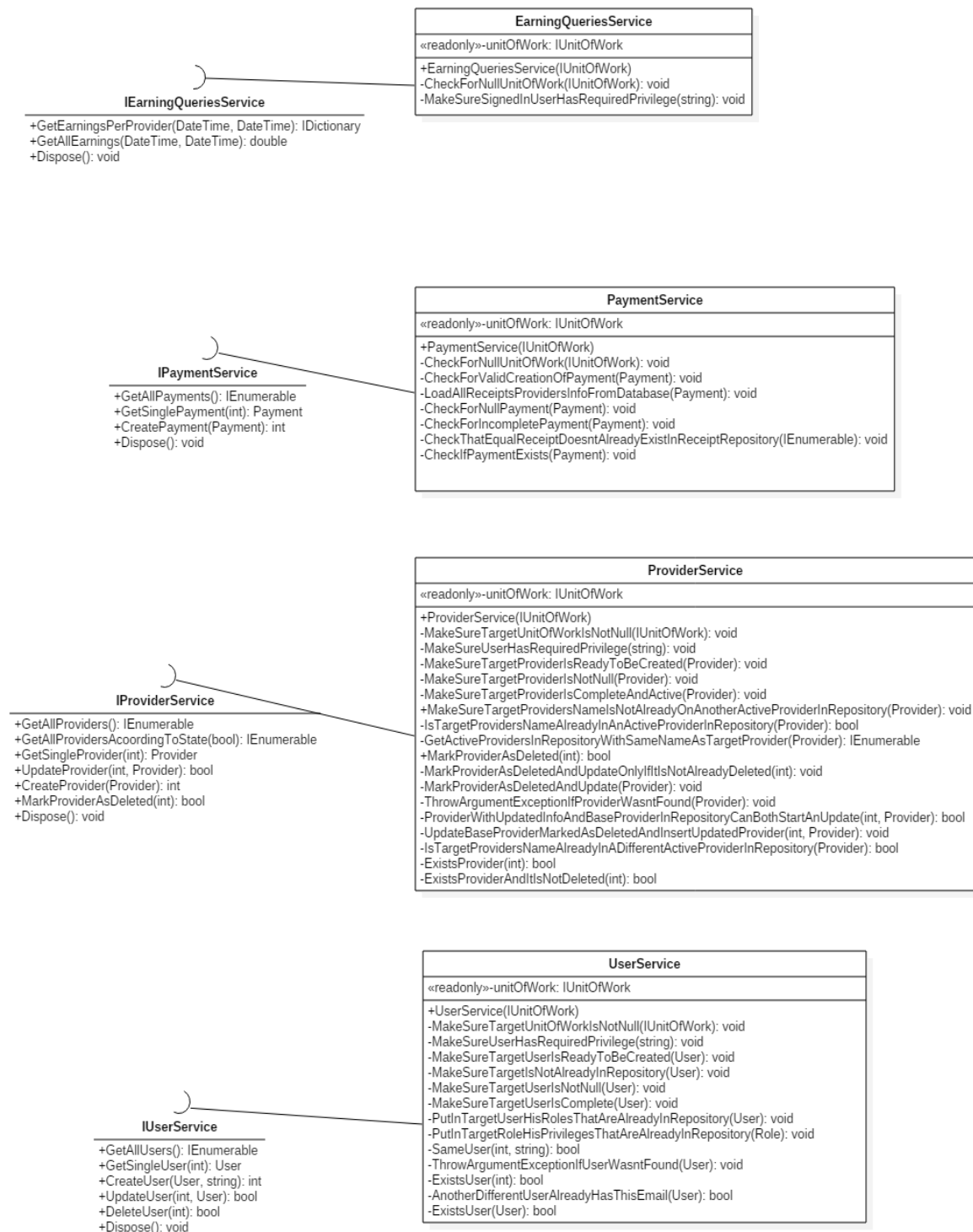
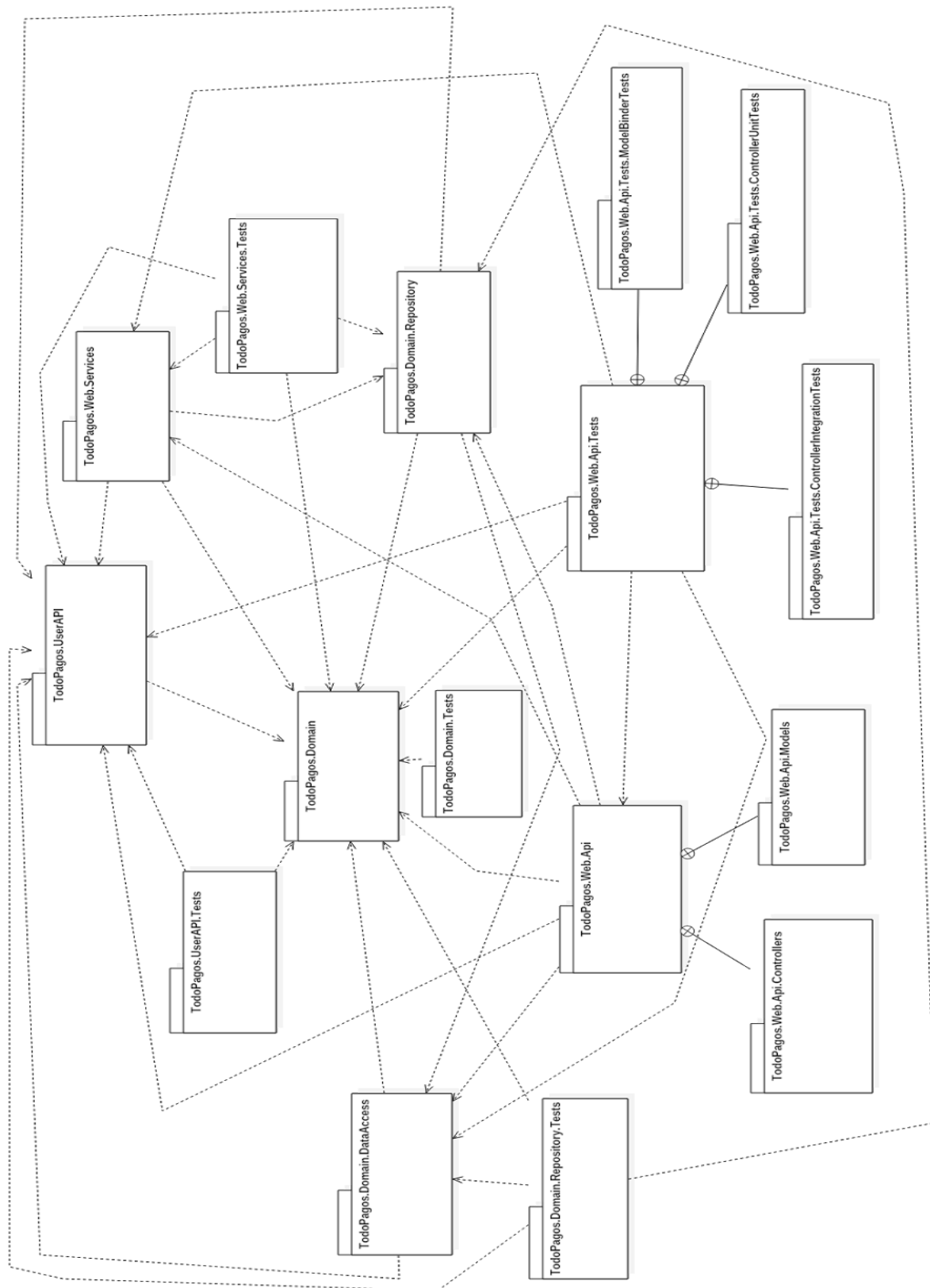
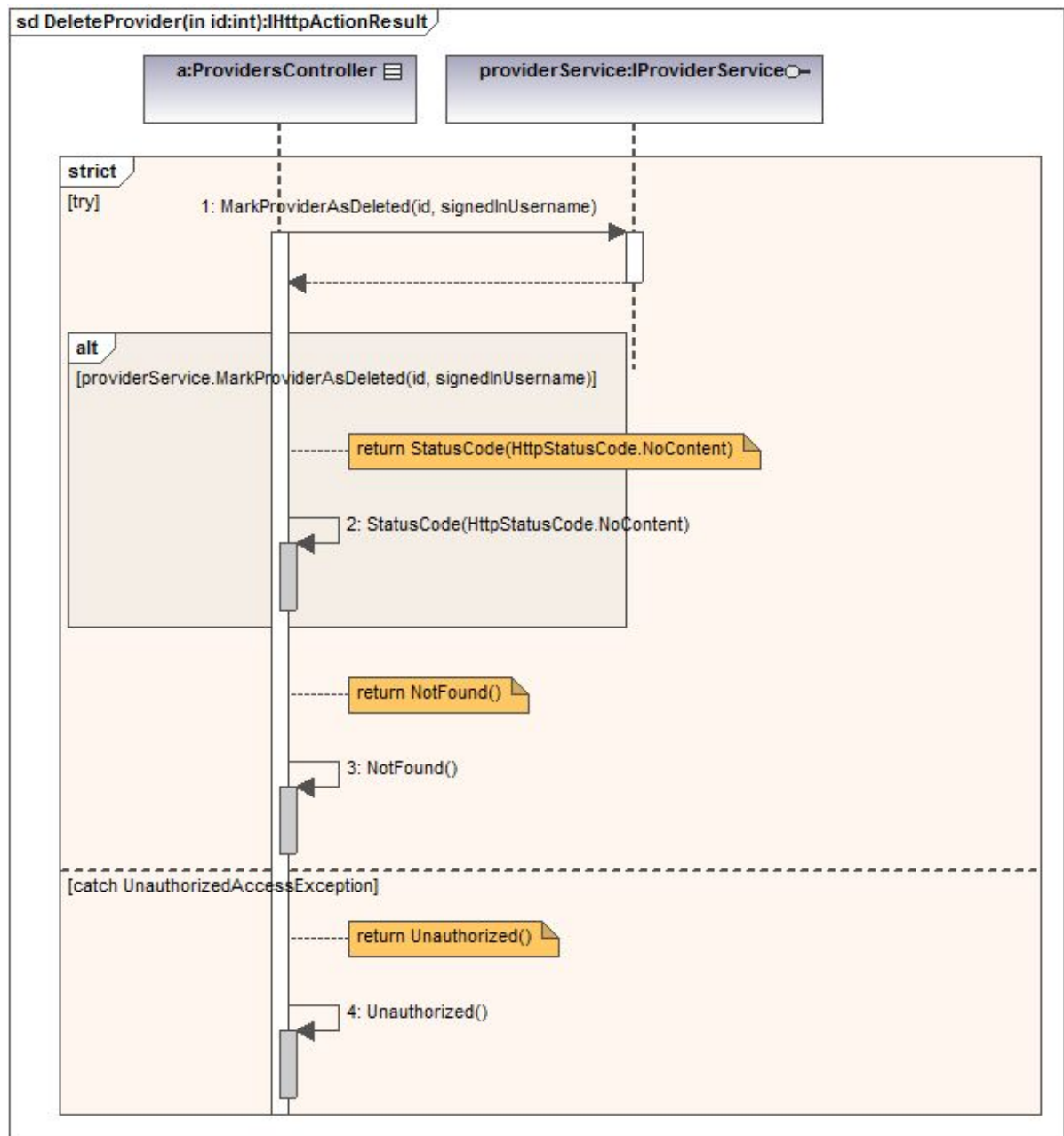


Diagrama de Paquetes

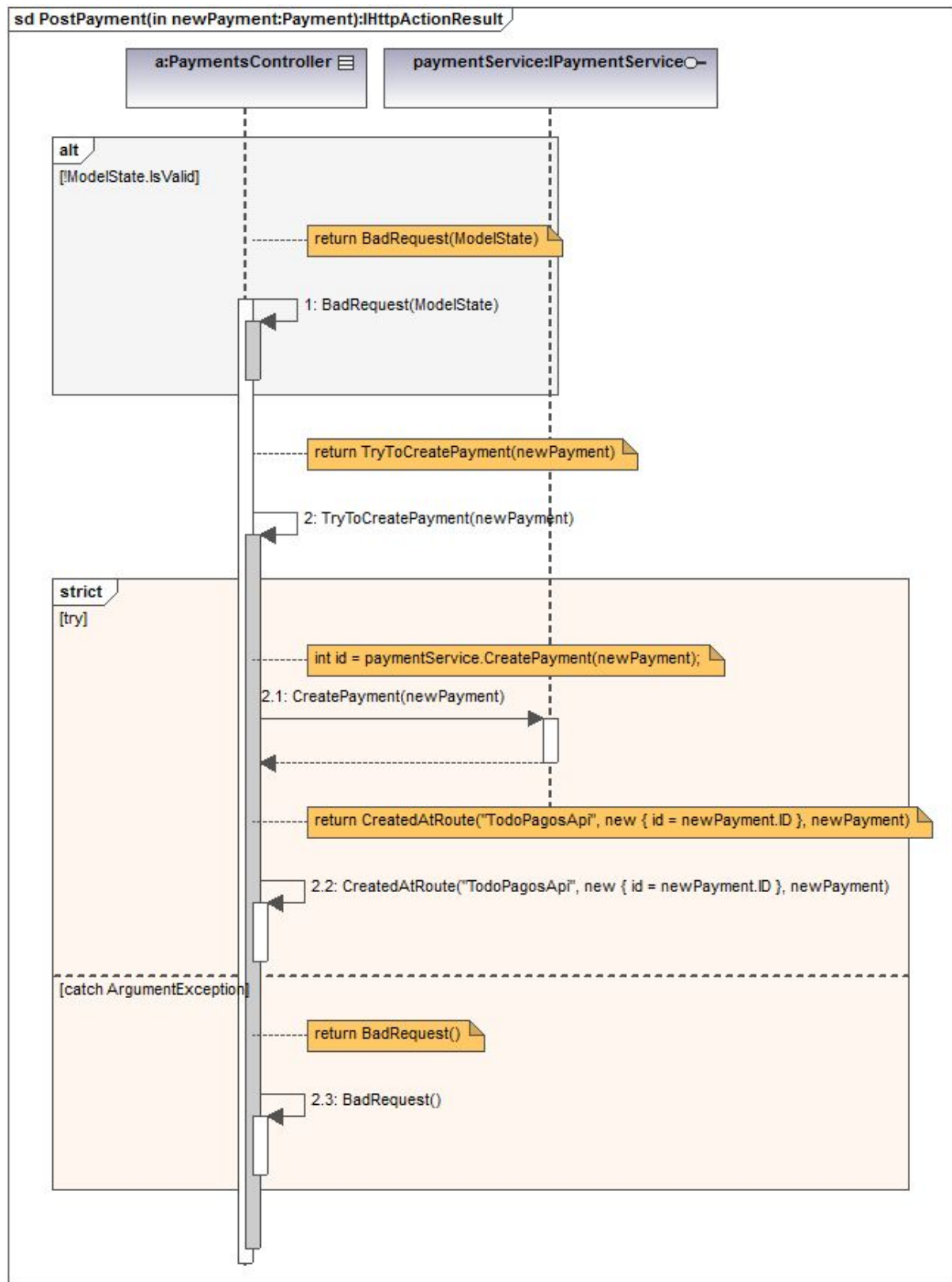


Diagramas de Interacción

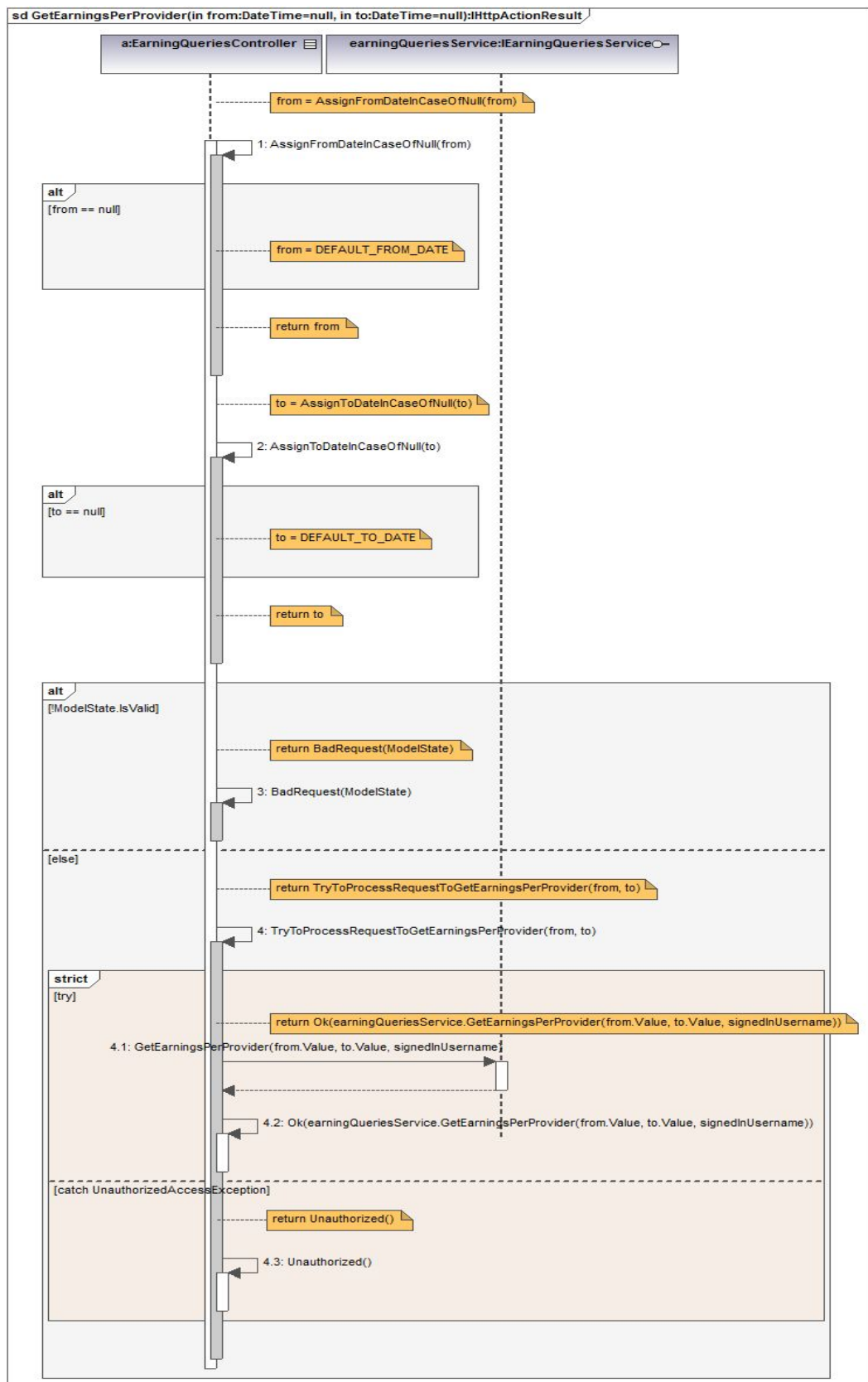
DeleteProvider



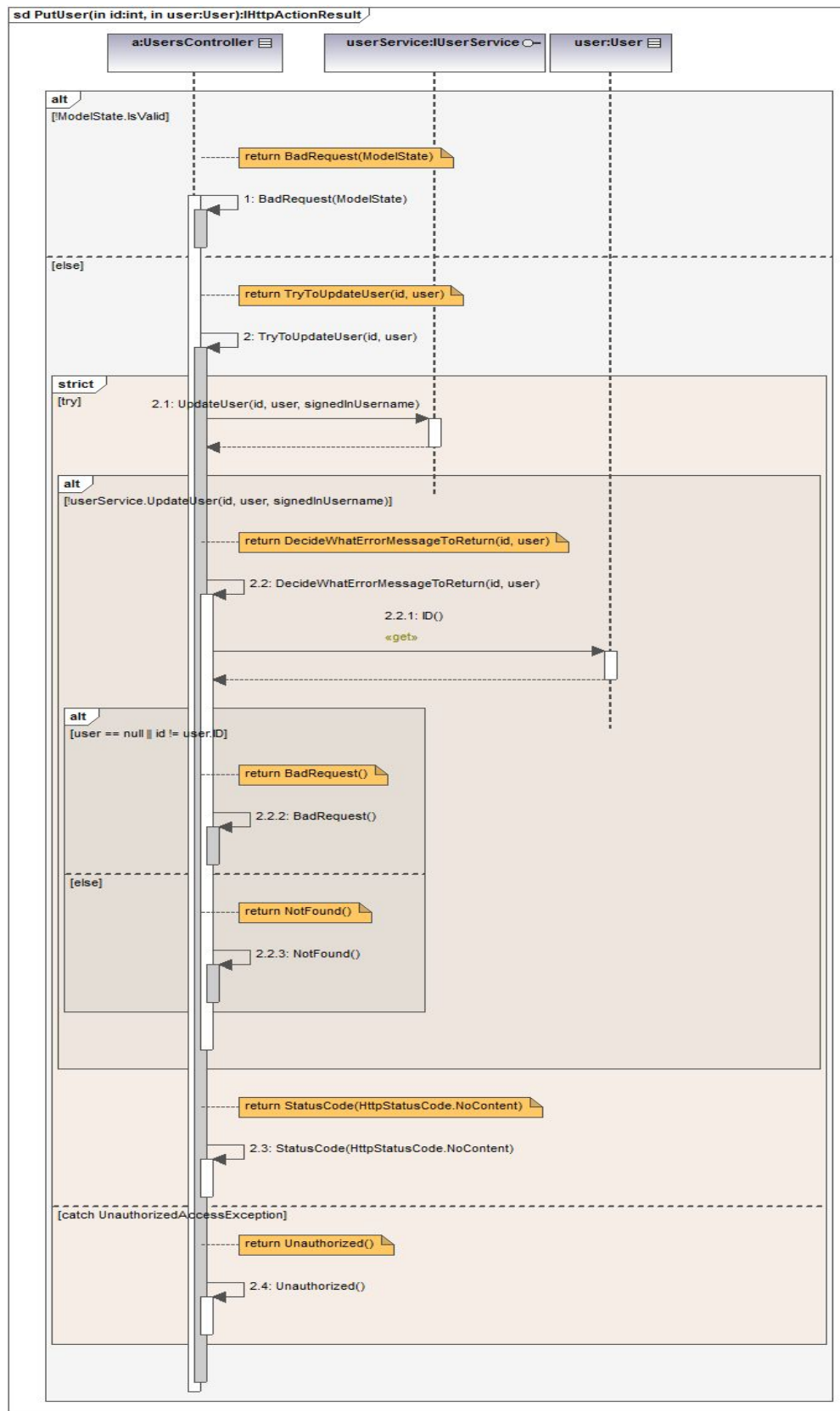
PostPayment



GetEarningsPerProvider



PutUser



GetAllProviders

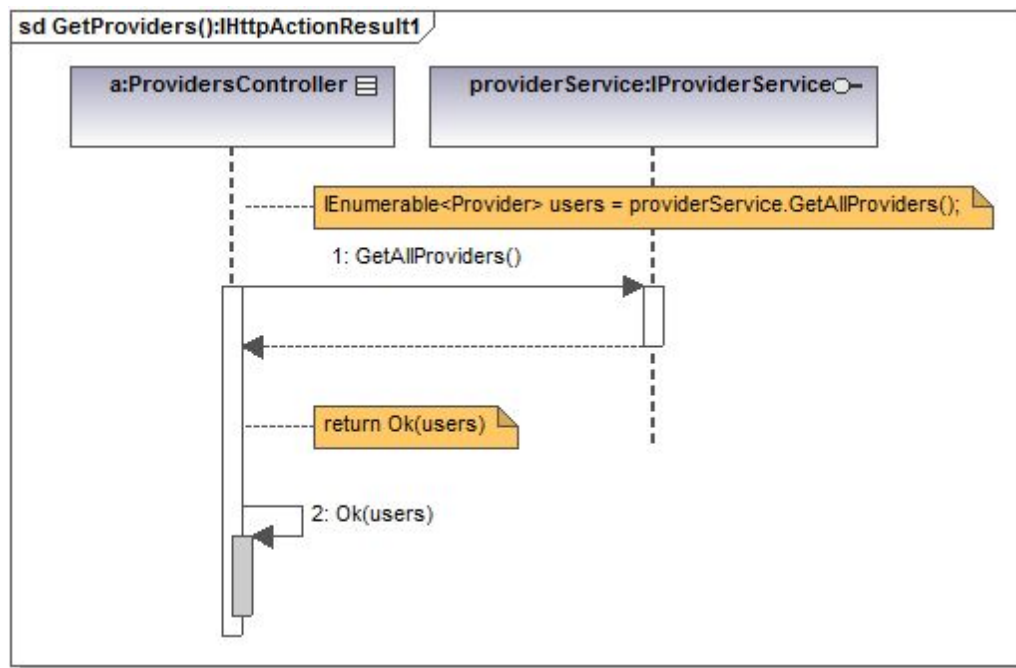


Diagrama de Componentes

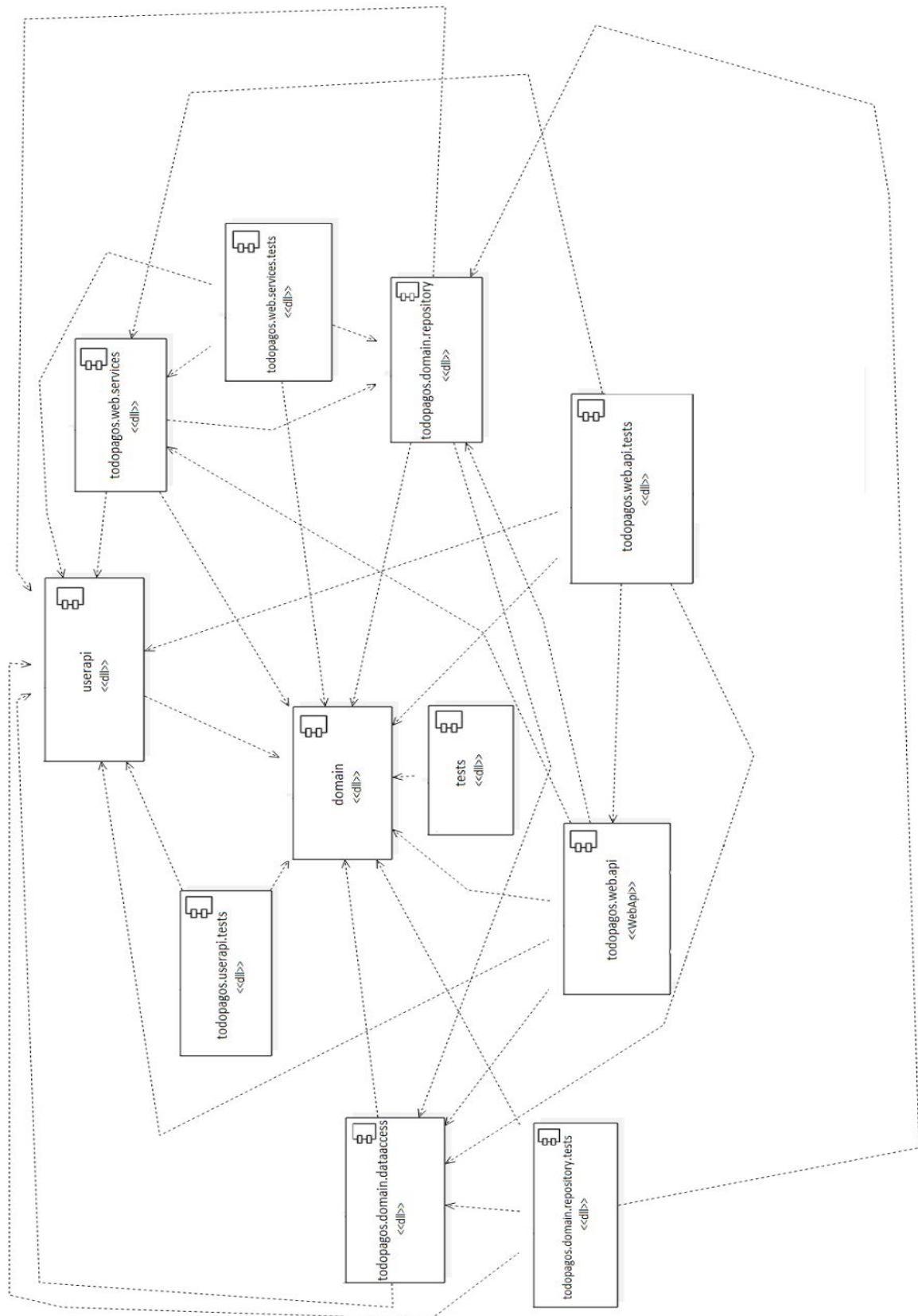
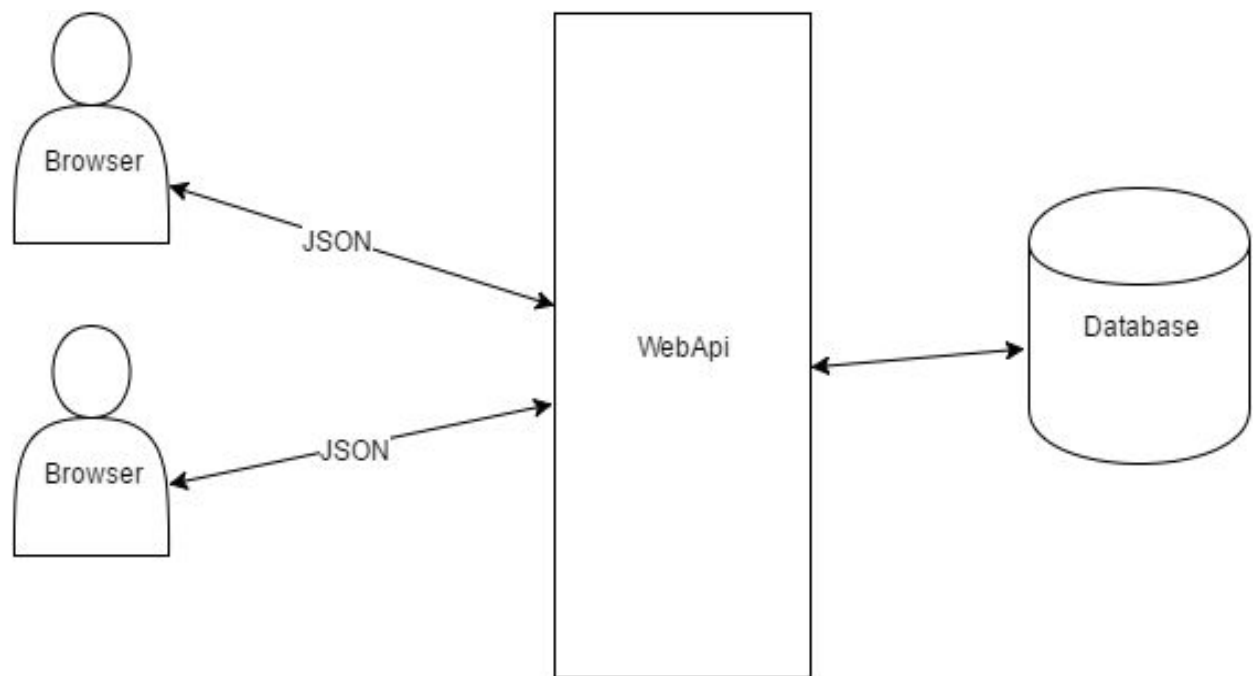


Diagrama de Entrega



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 apprehendido 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

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

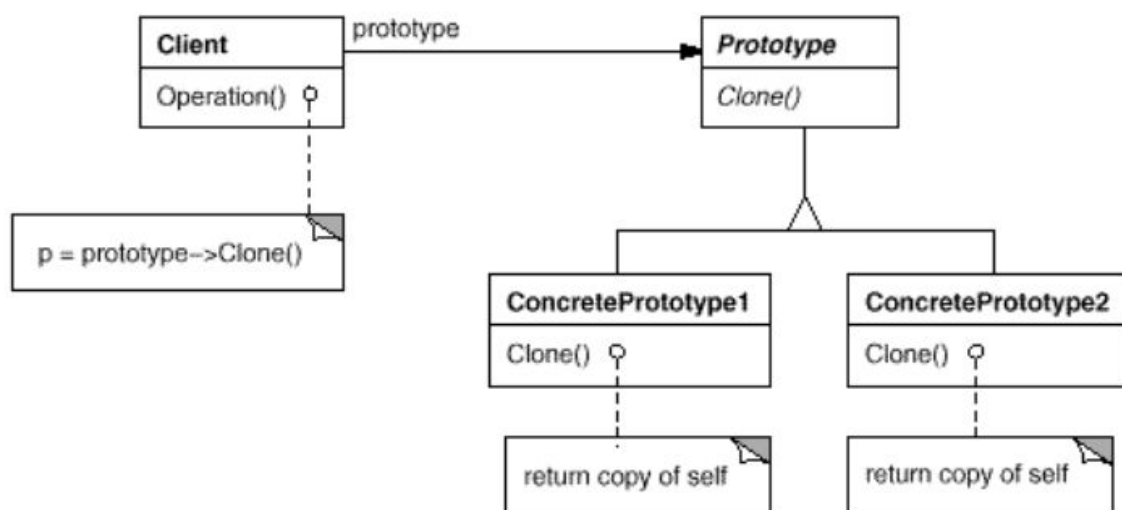
Junto con el patrón Repository, se implementó un repositorio genérico (utilizando Generics) ya que el crear una clase Repository para cada entidad resulta en mucho código redundante. Además, para asegurarnos de que todos los repositorios usen el mismo contexto y así evitar los problemas que podrían surgir 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 objetivo crear nuevos objetos 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 todas 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

Se observará que la clase CashierRole no agrega ningún privilegio al momento. Decidimos tomar como base el rol del cajero y agregar los privilegios en la clase AdminRole, que corresponde al rol del administrador.

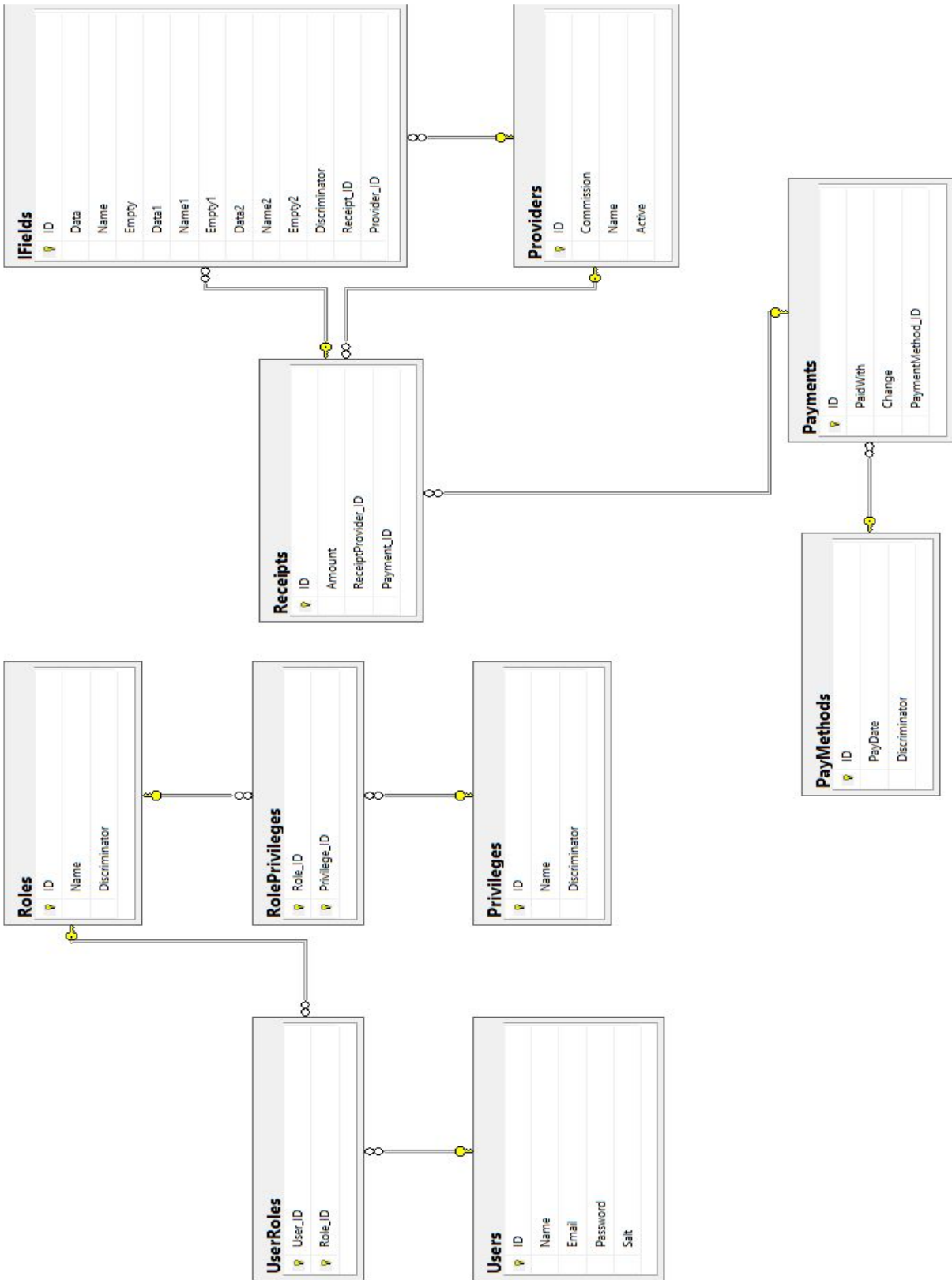
Al tener un mecanismo que permite agregar privilegios y roles de manera fácil y rápida (aunque se necesite volver a compilar nuevamente), optamos por esta decisión ya que el haberle colocado privilegios a la clase CashierRole (por ejemplo: un privilegio que le permita agregar un pago) hubiera pecado 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 excepciones que brinda `System`.

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 cumpliera 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 (redundancia, 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 240 líneas de código, la mayoría tiene entre 50-150 líneas).
- Tienen un único eje de cambio.

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

- **Nombres adecuados al estándar:**

```

9 referencias | 7/7 pasando | Diego Zuluaga, Hace 2 días | 1 autor, 4 cambios
public int CreateUser(User newUser)
{
    MakeSureTargetUserIsReadyToBeCreated(newUser);
    unitOfWork.UserRepository.Insert(newUser);
    unitOfWork.Save();
    return newUser.ID;
}

1 referencia | Diego Zuluaga, Hace 2 días | 1 autor, 1 cambio
private void MakeSureTargetUserIsReadyToBeCreated(User targetUser)
{
    MakeSureTargetUserIsNotNull(targetUser);
    MakeSureTargetUserIsComplete(targetUser);
}

1 referencia | Diego Zuluaga, Hace 2 días | 1 autor, 1 cambio
private void MakeSureTargetUserIsNotNull(User targetUser)
{
    if (targetUser == null)
    {
        throw new ArgumentNullException();
    }
}

1 referencia | Diego Zuluaga, Hace 2 días | 1 autor, 1 cambio
private void MakeSureTargetUserIsComplete(User targetUser)
{
    if (!targetUser.IsComplete())
    {
        throw new ArgumentException();
    }
}

```

- Funciones y Manejo de Errores adecuados al estándar:

```

6 referencias | 4/4 pasando | Bruno Ferrari, Hace 2 días | 1 autor, 4 cambios
public Payment GetSinglePayment(int paymentId)
{
    Payment payment = unitOfWork.PaymentRepository.GetByID(paymentId);
    CheckIfPaymentExists(payment);
    return payment;
}

1 referencia | 0 cambios | 0 autores, 0 cambios
private void CheckIfPaymentExists(Payment payment)
{
    if (payment == null) throw new ArgumentException();
}

```

- Tests adecuados al estándar:

```

[TestClass]
0 referencias | Bruno Ferrari, Hace 5 días | 2 autores, 8 cambios
public class PaymentsControllerShould
{
    [TestMethod]
    ✓ | 0 referencias | Bruno Ferrari, Hace 5 días | 1 autor, 1 cambio
    public void RecieveAPaymentServiceOnCreation()
    {
        var mockPaymentService = new Mock<IPaymentService>();

        PaymentsController controller = new PaymentsController(mockPaymentService.Object);
    }

    [TestMethod]
    [ExpectedException(typeof(ArgumentException))]
    ✓ | 0 referencias | Bruno Ferrari, Hace 5 días | 1 autor, 1 cambio
    public void FailCreationIfServiceIsNull()
    {
        IPaymentService service = null;

        PaymentsController controller = new PaymentsController(service);
    }
}

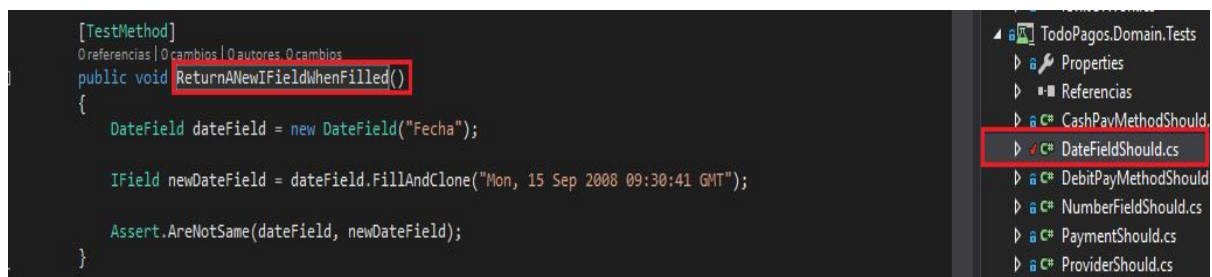
```

- 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();
        }
    }
}
```

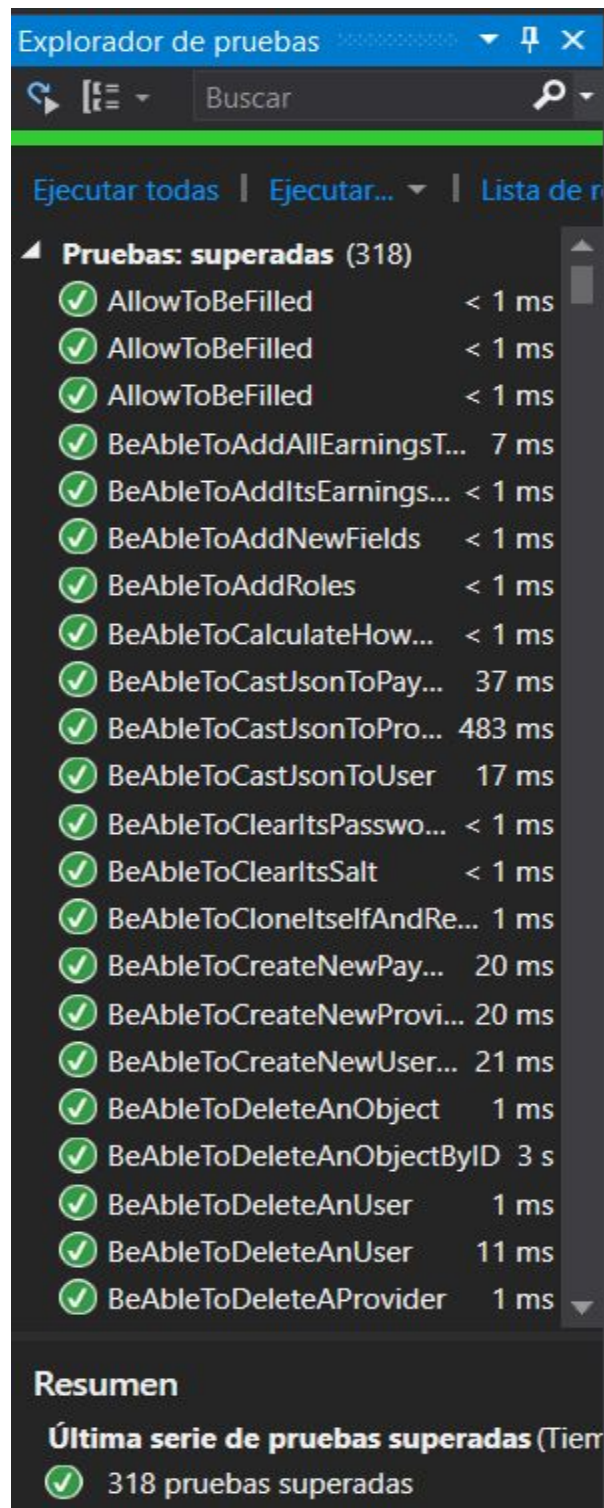
Sobre los nombres de las pruebas:

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 318 pruebas unitarias:



The screenshot shows the 'Explorador de pruebas' (Test Explorer) window in Visual Studio. The window title is 'Explorador de pruebas'. Below the title bar, there is a search bar with the text 'Buscar'. The main area displays a list of tests under the heading 'Pruebas: superadas (318)'. Each test is preceded by a green checkmark icon. The tests listed include 'AllowToBeFilled', 'BeAbleToAddAllEarningsT...', 'BeAbleToAddItsEarnings...', 'BeAbleToAddNewFields', 'BeAbleToAddRoles', 'BeAbleToCalculateHow...', 'BeAbleToCastJsonToPay...', 'BeAbleToCastJsonToPro...', 'BeAbleToCastJsonToUser', 'BeAbleToClearItsPasswo...', 'BeAbleToClearItsSalt', 'BeAbleToCloneItselfAndRe...', 'BeAbleToCreateNewPay...', 'BeAbleToCreateNewProvi...', 'BeAbleToCreateNewUser...', 'BeAbleToDeleteAnObject', 'BeAbleToDeleteAnObjectByID', 'BeAbleToDeleteAnUser', and 'BeAbleToDeleteAProvider'. The execution times for these tests are shown on the right side of each entry. At the bottom of the window, there is a 'Resumen' (Summary) section that states 'Última serie de pruebas superadas (Tierr)' and '318 pruebas superadas' with a green checkmark icon.

Test Name	Execution Time
AllowToBeFilled	< 1 ms
AllowToBeFilled	< 1 ms
AllowToBeFilled	< 1 ms
BeAbleToAddAllEarningsT...	7 ms
BeAbleToAddItsEarnings...	< 1 ms
BeAbleToAddNewFields	< 1 ms
BeAbleToAddRoles	< 1 ms
BeAbleToCalculateHow...	< 1 ms
BeAbleToCastJsonToPay...	37 ms
BeAbleToCastJsonToPro...	483 ms
BeAbleToCastJsonToUser	17 ms
BeAbleToClearItsPasswo...	< 1 ms
BeAbleToClearItsSalt	< 1 ms
BeAbleToCloneItselfAndRe...	1 ms
BeAbleToCreateNewPay...	20 ms
BeAbleToCreateNewProvi...	20 ms
BeAbleToCreateNewUser...	21 ms
BeAbleToDeleteAnObject	1 ms
BeAbleToDeleteAnObjectByID	3 s
BeAbleToDeleteAnUser	1 ms
BeAbleToDeleteAnUser	11 ms
BeAbleToDeleteAProvider	1 ms

Resumen
Última serie de pruebas superadas (Tierr)
318 pruebas superadas

Se omitirá la lista específica que detalla cada test y su nombre ya que al ser más de 300 quedaría extremadamente extensa.

Las 318 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

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:

Resultados de la cobertura de código				
Bruno_DESKTOP-JI2MV3T 2016-10-19 16_34				
Jerarquía	No cubiertos (bloques)	No cubiertos (% de bloques)	Cubiertos (bloques)	Cubiertos (% de bloques)
▲ Bruno_DESKTOP-JI2MV3T 2016-10-19 16_...	336	3,99 %	8079	96,01 %
▲ domain.dll	30	3,86 %	747	96,14 %
▸ { } TodoPagos.Domain	30	3,86 %	747	96,14 %
▸ tests.dll	31	4,52 %	655	95,48 %
▲ todopagos.domain.dataaccess.dll	7	15,56 %	38	84,44 %
▸ { } TodoPagos.Domain.DataAccess	7	15,56 %	38	84,44 %
▲ todopagos.domain.repository.dll	2	1,89 %	104	98,11 %
▸ { } TodoPagos.Domain.Repository	2	1,89 %	104	98,11 %
▸ todopagos.domain.repository.tests.dll	2	0,72 %	275	99,28 %
▸ todopagos.userapi.tests.dll	14	6,76 %	193	93,24 %
▲ todopagos.web.api.dll	143	11,80 %	1069	88,20 %
▸ { } TodoPagos.Web.Api	28	40,58 %	41	59,42 %
▸ { } TodoPagos.Web.Api.Controllers	58	14,57 %	340	85,43 %
▸ { } TodoPagos.Web.Api.Models	57	7,65 %	688	92,35 %
▸ todopagos.web.api.tests.dll	17	0,96 %	1762	99,04 %
▲ todopagos.web.services.dll	36	6,70 %	501	93,30 %
▸ { } TodoPagos.Web.Services	36	6,70 %	501	93,30 %
▸ todopagos.web.services.test.dll	33	1,41 %	2315	98,59 %
▲ userapi.dll	21	4,76 %	420	95,24 %
▸ { } TodoPagos.UserAPI	21	4,76 %	420	95,24 %

Como puede observarse, se obtuvo un 96.01% 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: 96.14%
- TodoPagos.Domain.DataAccess: 84.44%
- TodoPagos.Domain.Repository: 98.11%
- TodoPagos.UserAPI: 95.24%
- TodoPagos.Web.Api: 59.42%
- TodoPagos.Web.Api.Controllers: 85.43%
- TodoPagos.Web.Api.Models: 92.35%
- TodoPagos.Web.Services: 93.30%

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).

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_\(patr%C3%B3n_de_dise%C3%B1o\)](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].

Anexo

Datos de Prueba

Users

- Diego Ferrari, diego@bruno.com, HOLA1234, Administrador
- Gabriel Valle, nacho@gabriel.com, HOLA1234, Cajero
- Eric Méndez, eric@santiago.com, HOLA1234, Administrador y Cajero

Providers

- Antel, 10%, Activo
- OSE, 15%, Inactivo
- OSE, 20%, Activo

IFields

- Antel: un NumberField de nombre "Cedula", un TextField de nombre "Nombre" y un DateField de nombre "Fecha"
- OSE (Inactivo): un NumberField de nombre "RUT" y un DateField de nombre "Fecha".
- OSE: un NumberField de nombre "RUT"
- Payment (pagado con 250): un NumberField de nombre "Cedula" y de dato "4901883", un TextField de nombre "Nombre" y de dato "Diego" y un DateField de nombre "Fecha" y de dato "19/10/2016"
- Payment (pagado con 3000): un NumberField de nombre "RUT" y de dato "123"

Payments

- ID: 1, pagado con 250, cambio 0 y PayMethod DebitPayMethod
- ID: 2, pagado con 3000, cambio 200 y PayMethod CashPayMethod

PayMethods

- El DebitPayMethod ingresado tiene fecha 15/09/2008
- El CashPayMethod ingresado tiene fecha 15/09/2008