
OBLIGATORIO 2 - DepoQuick

Diseño 1

Link al repositorio:

https://github.com/IngSoft-DA1-2023-2/281835_281542_250230.git

Integrantes:

- Agustin Do Canto (250230)
- Alejo Fraga (281542)
- Sebastian Vega (281835)

Docentes:

- Facundo Aracet
- Gaston Mousques
- Martin Radovitzky

Junio 2024

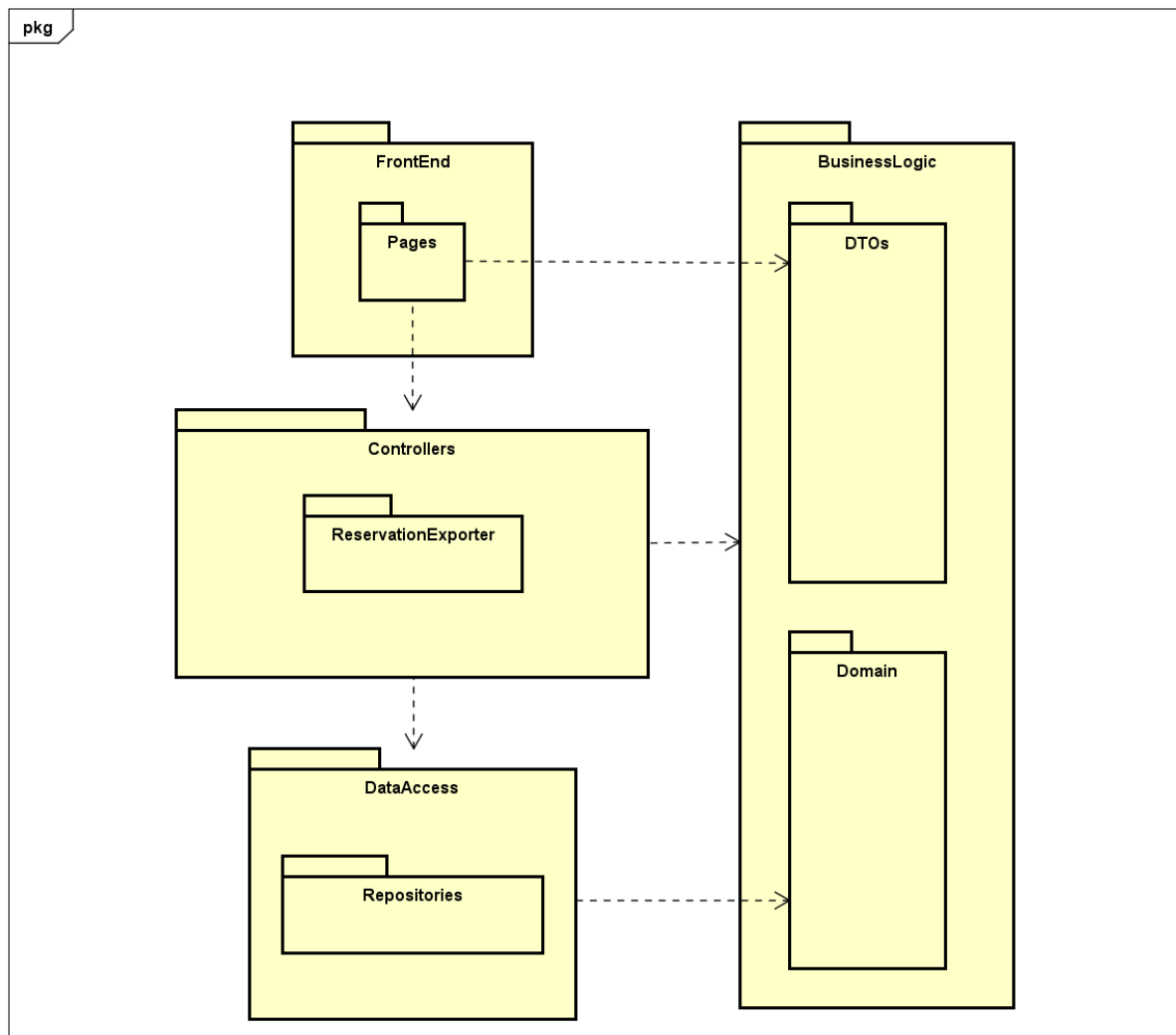
Universidad ORT Uruguay

Facultad de Ingeniería

ÍNDICE

1. Justificación del Diseño.....	1
Separación de facade en múltiples controladores.....	2
Acoplamiento de la capa de acceso a datos y la lógica de negocio (DataAccess y BusinessLogic).....	4
Alternativa considerada: Utilización de Data Transfer Objects (DTOs).....	5
Decisión tomada.....	5
2. Tabla de asignación de responsabilidades (namespaces).....	6
3. Cobertura de las pruebas unitarias.....	7
4. Modelo de tablas de la base de datos.....	8
5. Nuevas Funcionalidades.....	9
Gestión de disponibilidad de reservas.....	9
Disponibilidad de depósito.....	9
Pagos.....	11
Reporte de reservas.....	11
Notificaciones.....	12
6. Análisis de Dependencias (Reporte de reservas).....	13
7. Manejo de errores.....	14
8. Diagramas de Interacción.....	15
Diagrama de secuencia : Reserva de depósito y Pago.....	15
Diagrama de secuencia : Calculo de precio de deposito.....	15
Diagrama de secuencia: Reporte de reservas.....	16
INSTALACIÓN.....	16
ANEXOS.....	17
[Anexo 1].....	17
[Anexo 2].....	19
Diagramas de clases.....	19
Diagrama de clases (BusinessLogic).....	19
Diagrama de clases (DataAccess).....	20
Diagrama de clases (Controllers).....	20
[Anexo 3].....	21
Diagrama de relación entre clases (Controllers -> BusinessLogic).....	21
[Anexo 4].....	22
[Anexo 5].....	25
Logo DEPOQUICK.....	25

1. Justificación del Diseño

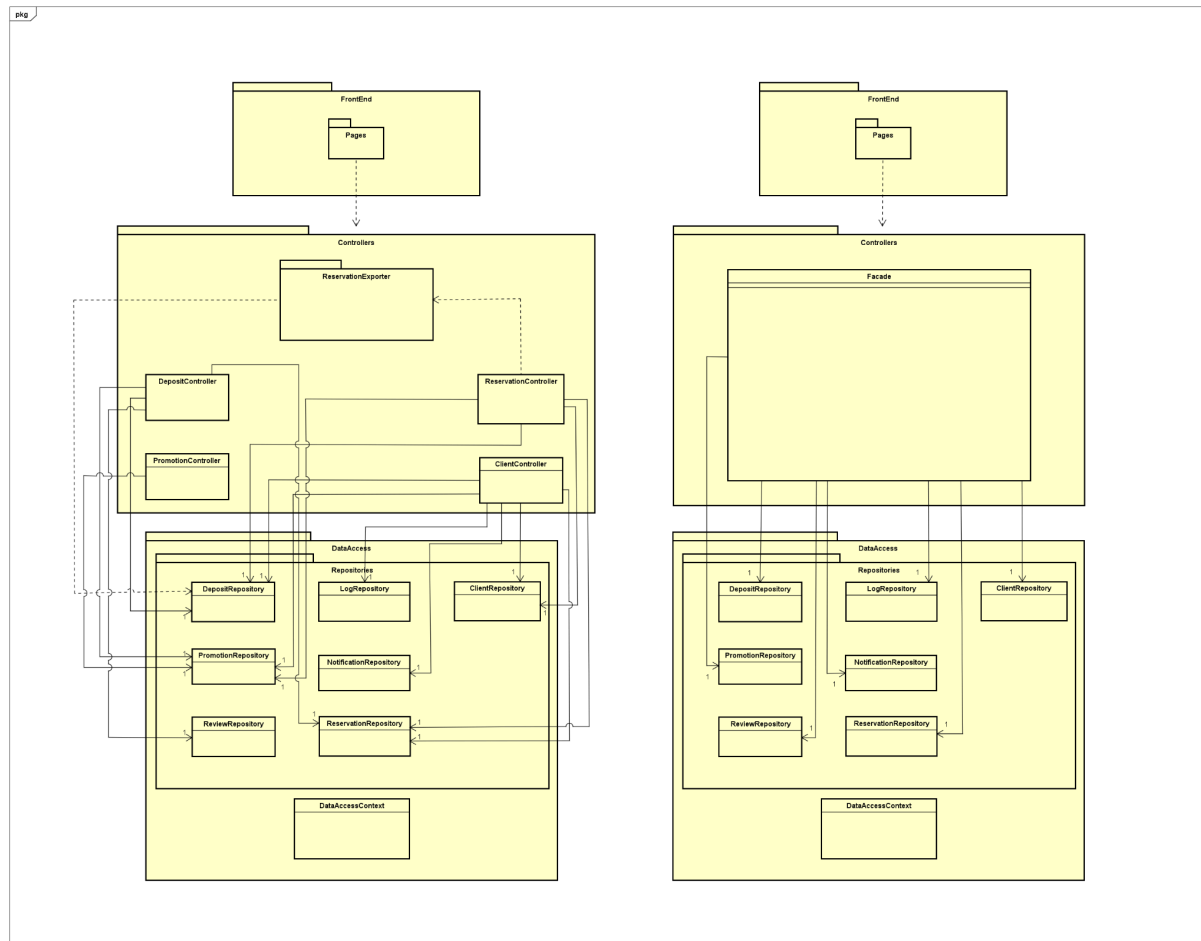


En el diseño de la segunda versión de DEPOQUICK, cada página se comunica con los controladores de depósito, reserva, promoción y cliente a través de **objetos de transferencia de datos** (DTOs), siguiendo el principio de "Alta Cohesión y Bajo Acoplamiento" de GRASP. Al utilizar DTOs, se promueve la cohesión al agrupar los datos necesarios para una tarea específica, mientras se logra un bajo acoplamiento al permitir que el frontend y la lógica de negocio evolucionen de manera independiente.

Además, para el almacenamiento y consulta de datos, los controladores se comunican con los repositorios ubicados en el proyecto DataAccess (capa de acceso a datos), los cuales interactúan directamente con la base de datos a través de Entity Framework. Este enfoque sigue el patrón de diseño GRASP "Controlador", que asigna la responsabilidad de las interacciones entre los objetos de la aplicación a clases controladoras específicas, evitando acoplamientos innecesarios.

Nota: Diagramas en detalle de la composición de cada paquete en [ANEXO 2]

Separación de facade en múltiples controladores



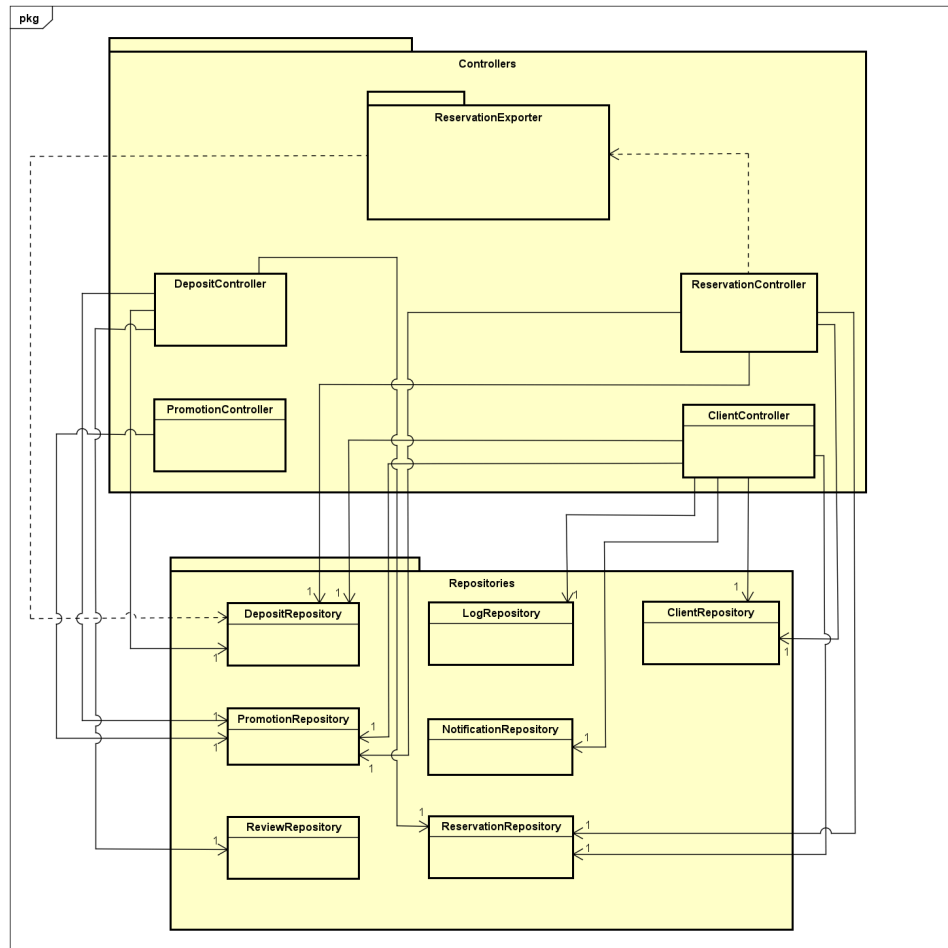
Las entidades Cliente, Promoción y Depósito son fundamentales y autónomas dentro del sistema, no dependen de otras entidades y sirven como pilares centrales para la lógica de la aplicación.

Su independencia y relevancia en el sistema se reflejan en la asignación de controladores exclusivos para cada una de ellas (PromotionController, DepositController y ClientController), lo que permite gestionar de manera específica y eficaz las operaciones relacionadas con estas, favoreciendo al principio de "Responsabilidad Única" de SOLID.

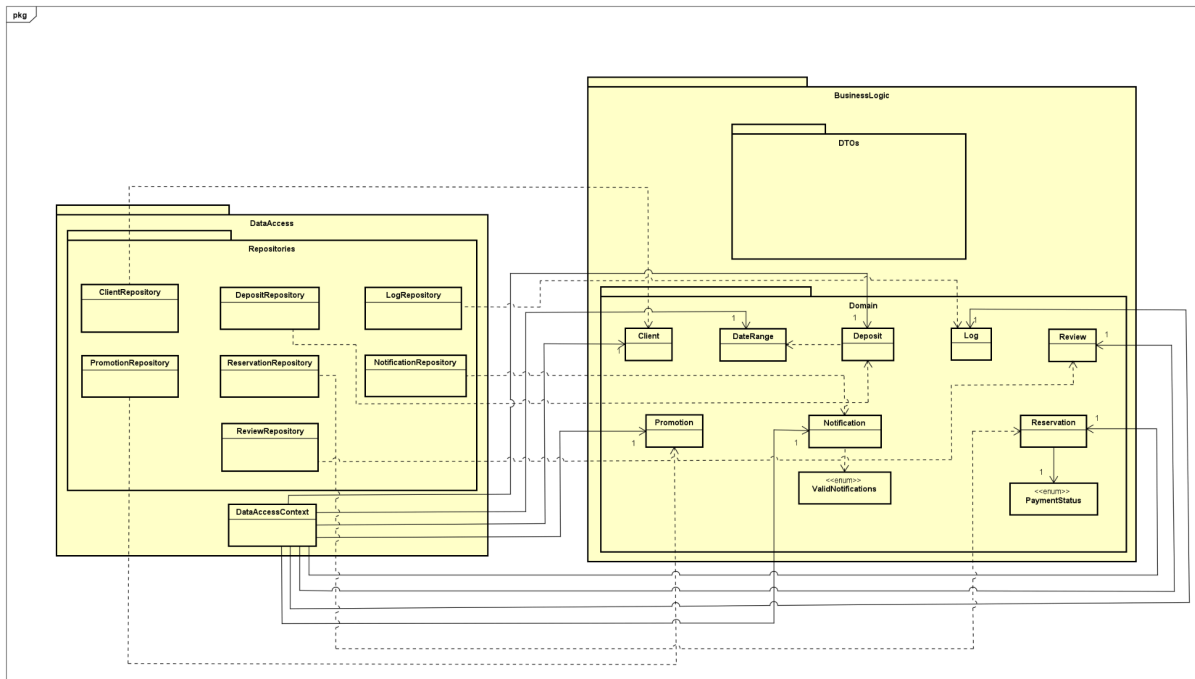
Además, se ha considerado que la lógica asociada a la entidad de reserva (Reservation) es más compleja y genera mayor dependencia con otras partes del sistema. Por este motivo, se determinó la necesidad de asignar un controlador exclusivo para gestionar la lógica relacionada con las reservas, lo que resultó en la creación de ReservationController.

Esta estructuración refleja un enfoque más cohesivo y desacoplado, permitiendo que cada entidad mencionada tenga un controlador dedicado que gestione sus operaciones de forma independiente.

Por otra parte es importante destacar que a pesar de que no existe dependencia entre controladores, en ocasiones estos se comunican con los mismos repositorios generando mayor acoplamiento con los mismos.



Acoplamiento de la capa de acceso a datos y la lógica de negocio (DataAccess y BusinessLogic)



Al utilizar las propias entidades del dominio de la aplicación (BusinessLogic.Domain) en la capa de acceso a datos (DataAccess), se obtiene una representación directa y fiel de los objetos y estructuras de datos inherentes al negocio. Esto implica una integración más estrecha entre la capa de acceso a datos y la lógica de negocio, permitiendo una manipulación eficiente de los datos sin la necesidad de transformaciones complejas.

Ventajas:

- **Simplicidad en el diseño:** Al interactuar directamente con las entidades del dominio, se evita la duplicación y la necesidad de crear entidades adicionales para representar los datos.
- **Mantenimiento simplificado:** La gestión directa de las entidades del dominio reduce la complejidad al eliminar la necesidad de sincronizar estructuras de datos separadas.

Desafíos:

- **Acoplamiento alto:** Al acoplar directamente la capa de acceso a datos a las entidades del dominio, la flexibilidad y la capacidad de evolución de la aplicación podrían verse comprometidas, ya que los cambios en las entidades del dominio tendrían un impacto directo en la capa de acceso a datos.

Alternativa considerada: Utilización de Data Transfer Objects (DTOs)

La utilización de DTOs implica la introducción de estructuras de datos independientes, diseñadas específicamente para el intercambio de información entre las capas de la aplicación. Estos objetos encapsulan datos necesarios para operaciones específicas y brindan una mayor flexibilidad en el manejo de la información debido a su capacidad para adaptar la representación de los datos según las necesidades específicas de cada capa.

Ventajas:

- **Desacoplamiento entre capas:** Al utilizar DTOs, se establece un nivel de desacoplamiento entre la capa de acceso a datos y la lógica de negocio, lo que facilita la evolución independiente de ambas capas. Permitiendo de esta forma realizar cambios en ambas capas con un esfuerzo menor.
- **Flexibilidad en el intercambio de datos:** Los DTOs permiten adaptar la representación de los datos a las necesidades específicas de cada capa, por ejemplo, se podrían implementar todas las “shadow properties” necesarias sin ver afectada la cobertura de las clases de dominio.
- **Reducción de Datos Transportados:** Los DTOs pueden contener únicamente la información necesaria para una operación específica, lo que permite reducir la cantidad de datos transferidos entre capas y sistemas, mejorando el rendimiento.

Desafíos:

- **Complejidad adicional:** La introducción de DTOs implica la creación y mantenimiento de estructuras de datos adicionales, lo que puede aumentar la complejidad del diseño y la implementación de la aplicación.
- **Posible necesidad de transformaciones:** La utilización de DTOs puede requerir operaciones de mapeo y transformación entre las entidades del dominio y los DTOs, lo que implica una carga adicional en términos de mantenibilidad.

Decisión tomada




Después de considerar detenidamente las implicaciones y los beneficios de aplicar ambas alternativas, se ha tomado la decisión estratégica de no utilizar DTOs y, en su lugar, emplear directamente las entidades del dominio en la capa de acceso a datos. Esta elección se alinea con la búsqueda de un diseño simple que al tiempo que permite una integración más estrecha entre la lógica de negocio y la capa de acceso a datos.

2. Tabla de asignación de responsabilidades (namespaces)

Project	Namespace	Responsabilidad
BusinessLogic	BusinessLogic	“Lógica de negocio”, contiene a las clases de Dominio y a los objetos de transferencia de datos (DTOs) necesarios para transferir los datos de las clases del dominio.
BusinessLogic	BusinessLogic.Domain	Contiene a las clases del dominio de la aplicación
BusinessLogic	BusinessLogic.DTOs	Contiene todos los DTOs que actúan entre el FrontEnd y los controladores para lograr comunicar ambos paquetes sin acoplarse a las clases del dominio.
Controllers	Controllers	Contiene a las clases controladoras del sistema, las cuales tienen como principal objetivo realizar y delegar tareas tanto para brindarle información al FrontEnd como para recibir información.
Controllers	Controllers.ReservationExporter	Contiene a la jerarquía de clases “Exporter”, “CsvExporter” y “TxtExporter”. Estas clases son las encargadas de crear y exportar los archivos para el Reporte de Reservas..
DataAccess	DataAccess	Capa de acceso a datos, contiene al contexto (DataAccessContext) y a los repositorios.
DataAccess	DataLayer.Repositories	Contiene los “repositorios”, clases encargadas de comunicarse con la base de datos. Concretamente crear, eliminar, buscar y actualizar registros (CRUD).

FrontEnd	Pages	Maneja la interfaz de usuario.
----------	-------	--------------------------------

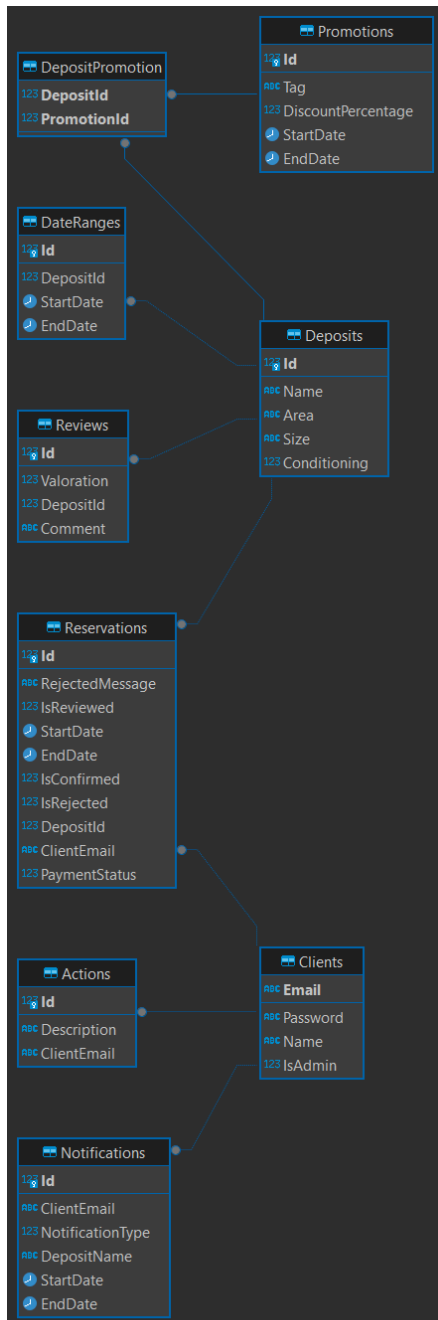
3. Cobertura de las pruebas unitarias

✓  Total	98%	35/1621
>  Controllers	100%	0/881
>  BusinessLogic	95%	35/740

Se logró una cobertura del 100% para las clases controladoras del sistema, mientras que para la lógica de negocio no se alcanzó total cobertura debido a la necesidad de implementar "Shadow Properties" para lograr cumplir las convenciones de Entity Framework.

Justificación e imágenes detalladas adjuntas en [Anexo 1]

4. Modelo de tablas de la base de datos



A continuación se muestra la correspondencia de cada tabla a su respectiva clase del dominio.

Tabla en la base de datos (Clase del dominio):

Deposits (Deposit)
Promotions (Promotion)
DateRanges (DateRange)
Reviews (Review)
Reservations (Reservation)
Actions (Log)
Clients (Client)
Notifications (Notification)

La tabla DepositPromotion surge de la necesidad de representar la relación de muchos a muchos de Depósito (Deposit) con Promoción (Promotion)

5. Nuevas Funcionalidades

Gestión de disponibilidad de reservas

Se agregó el nombre de depósito como atributo único, el cual es utilizado para referenciar a cada depósito durante todo el transcurso de la aplicación (Antiguamente para esto se utiliza el ID de depósito).

Agregar Depósito

Nombre	<input type="text" value="Nuevo deposito"/>
Área	<input type="text" value="A"/>
Tamaño	<input type="text" value="Grande"/>
Climatización	<input checked="" type="radio"/> Si <input type="radio"/> No
Promociones	<div><div>Id: 1 Tag: Promo 1 Id: 2 Tag: Promo 2 Id: 3 Tag: Promo 3 Id: 4 Tag: Promo 4</div><div>Para agregar más de una promoción debe mantener presionada la tecla Ctrl.</div></div>
<input type="button" value="Agregar Depósito"/>	

Disponibilidad de depósito

Se creó la página “Disponibilidad de Depósitos” cuyo uso es exclusivo para el usuario administrador de la aplicación, en esta, el mismo es capaz de asignar disponibilidad sobre los depósitos registrados en la aplicación.

Para poder dar de alta una nueva disponibilidad sobre un depósito, se deben seguir los siguientes pasos:

1. Iniciar sesion en la aplicacion como usuario administrador
2. En el menú de navegación, seleccionar “Disponibilidad de Depósitos” en la sección de administrador
3. Ingresar el rango de fechas deseado
4. Dar click en el botón “Agregar disponibilidad” sobre el depósito deseado.

Una vez efectuado el alta de disponibilidad de una rango de fechas dado, el depósito quedará automáticamente disponible para su reserva en dicho rango de fechas.

Notas y aclaraciones:

- Los rangos de disponibilidad asociados a un depósito pueden no reservarse por completo, es decir si se le agrega disponibilidad desde el día 1 hasta el día 10, se

admitirán reservas cuyos rangos están incluidos en el disponible, por ejemplo se admitirá la reserva del mismo depósito desde el día 1 hasta el 8, dejando aun disponible el depósito para reservarse desde el día 8 al 10.

- No se admitirá reservar un depósito que abarque dos rangos de disponibilidad diferentes, es decir : si un depósito dado tiene disponibilidad desde el día 1 al 5 y desde el 5 al 10, no será posible realizar una única reserva desde el día 1 al 10, se deberá realizar dos reservas independientes en caso de querer reservar la totalidad de los días.

Los depósitos permanecen disponibles según su rango de disponibilidad para todos los usuarios de la aplicación. Se admitirán reservas sobre el mismo depósito y en el mismo rango de fechas siempre y cuando el depósito no tenga ninguna reserva confirmada por el administrador en dichos rangos de fechas.

Al momento de que el administrador confirma una reserva en determinado rango de fechas, se eliminará dicho rango automáticamente del depósito asociado, rechazando de forma automática todas las reservas pendientes existentes sobre el depósito en dicho rango y agregando el siguiente mensaje de rechazo “Se rechazó la reserva automáticamente por superposición de fechas”.

Notas extra:

- Para una mejor experiencia, no se le permitirá al cliente realizar múltiples reservas en el mismo rango de fechas sobre un mismo depósito
- Se le solicita al usuario un rango de fechas a reservas y solamente se le mostrará los depósitos disponibles para ese rango

Reserva de Depositos

Fecha de Inicio



09/06/2024



Fecha de Fin



16/06/2024



Depósitos:

No se encontraron depositos disponibles para esas fechas

Pagos

Al momento de realizar una reserva de un depósito este decide si pagar por la reserva o volver. Si éste decide pagar, se le acreditará la reserva y el estado del depósito pasará a reservado.

Confirmación de reserva

El precio calculado es de 798

Nota: se agrego un descuento por cantidad de dias de 5%

VolverPagar

Cuando la reserva es aceptada por el administrador, el estado del pago depósito pasará a “Capturado”

Reporte de reservas

Se creó la página “Reporte de reservas” cuyo uso es exclusivo para el usuario administrador de la aplicación, en esta, el mismo es capaz de exportar las reservas realizadas por todos los clientes en la aplicación en archivos .txt y .csv.

Para poder realizar un exporte de reservas, se deben seguir los siguientes pasos:

1. Iniciar sesion en la aplicacion como usuario administrador
2. En el menú de navegación, seleccionar “Reporte de Reservas” en la sección de administrador
3. Presionar el botón “Exportar TXT” o “Exportar CSV” dependiendo el formato deseado

Una vez presionado el botón, se mostrará un mensaje de éxito en pantalla y automáticamente se descargara el archivo de extensión especificada en la carpeta “descargas”.

Exportar Reservas

Para exportar la información, presione el botón seleccionando el formato que desea. Se le proporcionará un archivo en la carpeta de descargas con la información completa de las reservas.

Exportar CSV

Exportar TXT

Notificaciones

Se creó la página “Notificaciones” disponible para todos los usuarios de la aplicación, en la misma se mostraran listadas notificaciones relacionadas a la confirmación y rechazo de reservas por parte del administrador.

Los datos mostrados son:

- Estado de la reserva
- Nombre del depósito
- Rango de fechas solicitado
- Estado del pago

Reserva Rechazada

La reserva del deposito JUANCHO desde 6/7/2024 12:00:00 AM hasta 6/10/2024 12:00:00 AM ha sido Rechazada.

El pago se le devolvera.



Nota:

Al presionar en el botón de confirmación, la notificación se eliminará.

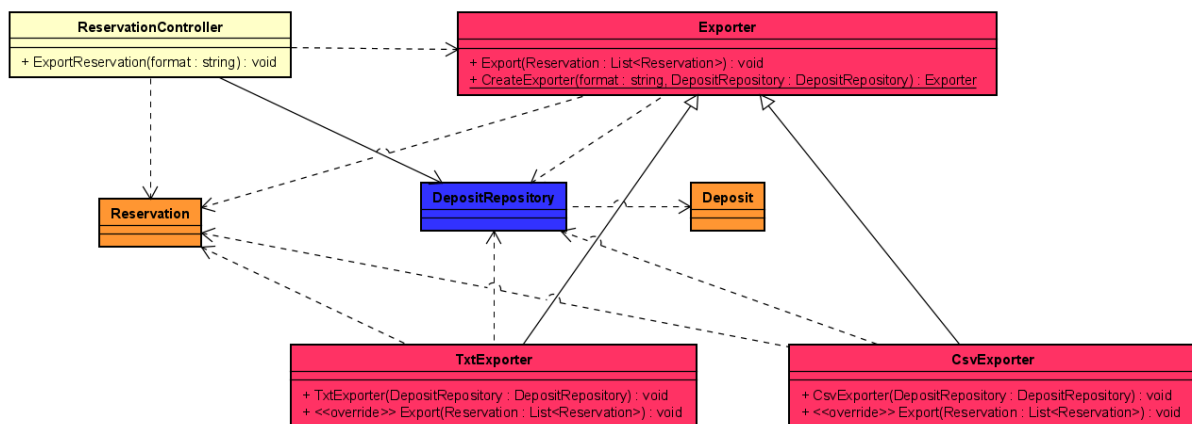
6. Análisis de Dependencias (Reporte de reservas)

```
6 usages 3 tests OK Alejo Fraga
public void ExportReservation(string format)
{
    var reservations : List<Reservation> = GetReservations();
    Exporter exporter = Exporter.CreateExporter(format, DepositRepository);
    exporter.Export(reservations);
}
```

La función ExportReservation se encuentra en la clase ReservationController dentro del paquete Controllers.

La misma recibe por parámetro el tipo de exporte a realizar (TXT o CSV), crea una variable de tipo Exporter asociado a un objeto del tipo TxtExporter o CsvExporter dependiendo el “format” recibido.

Una vez creado el objeto, se llama a la función Export del mismo la cual se encarga de realizar el exporte de reservas.



Nota:

Se muestran únicamente relaciones generadas de la función ExportReservations. La función CreateExporter, se encarga de devolver el objeto deseado dependiendo del formato recibido. De esta forma, se logra que ReservationController dependa únicamente de Exporter dando un uso eficaz del polimorfismo.

Se destaca claramente el LSP (LISKOV SUBSTITUTION PRINCIPLE) de SOLID dado que las clases hijas (TxtExporter y CsvExporter) sustituyen correctamente a su padre (Exporter).

Al mismo tiempo, se cumple OCP (OPEN CLOSED PRINCIPLE) de SOLID, dado que si se necesitará implementar un nuevo tipo, basta con crear una nueva clase que herede de Exporter y sobrescriba el método Export, agregando a su vez una línea más en CreateExporter retornando el nuevo objeto.

```
public static Exporter CreateExporter(string format, DepositRepository depositRepository)
{
    return format switch
    {
        "csv" => new CsvExporter(depositRepository),
        "txt" => new TxtExporter(depositRepository),
        _ => throw new ArgumentException(message: "Formato no soportado") Typo in string:
    };
}
```

Podemos concluir que la distribución de responsabilidades del exporte de reservas es muy buena dado que existe un bajo acoplamiento entre las clases y las mismas tienen un alto nivel de cohesión.

Se muestran las funciones que sobrescriben el método export de Exporter en [Anexo 4]

7. Manejo de errores

Se realiza un manejo exhaustivo de errores durante todo el transcurso de la aplicación, se muestran mensajes amigables y entendibles al usuario.

En el caso de que la conexión a la base de datos se detenga, se llevará al usuario a una página de error controlada donde se le informa cómo proseguir.



Oops.. No hemos logrado conectarte

Asegurate de estar conectado y refresca la pagina

8. Diagramas de Interacción

Diagrama de secuencia : Reserva de depósito y Pago

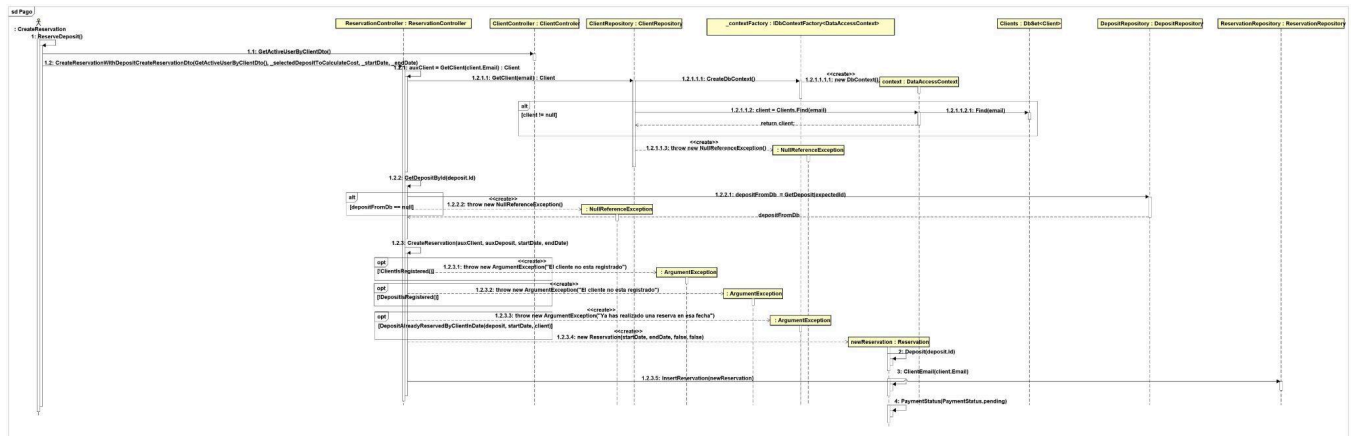


Diagrama de secuencia : Calculo de precio de deposito

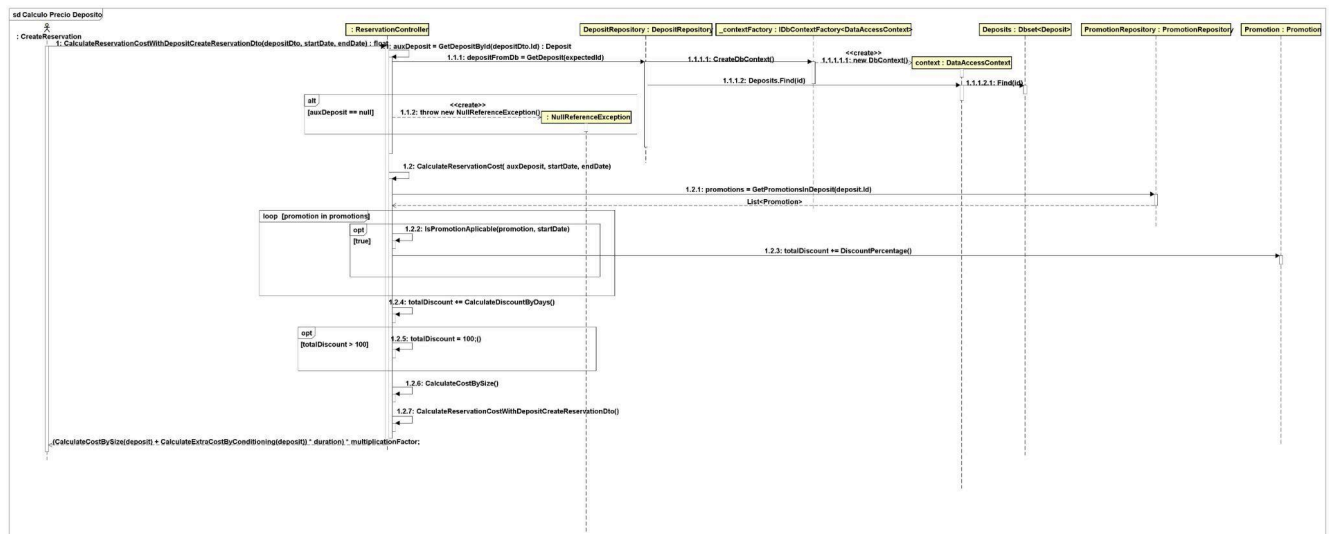
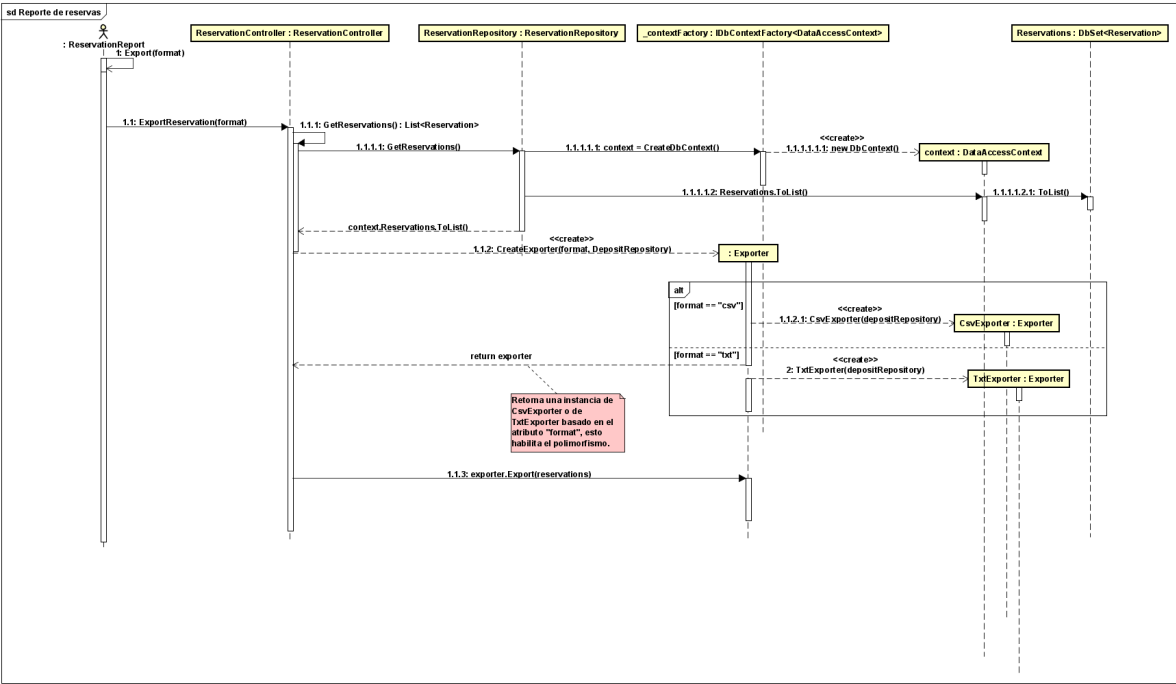


Diagrama de secuencia: Reporte de reservas











9. INSTALACIÓN

Configuración Docker:

Se deberán ejecutar los siguientes comandos:

```
C:\Users\Agustín Do Canto\Desktop\Semestre 5\Diseño 1\OBLIGATORIO 1 REPO\281835_281542_250230>cd Solution
C:\Users\Agustín Do Canto\Desktop\Semestre 5\Diseño 1\OBLIGATORIO 1 REPO\281835_281542_250230\Solution>docker-compose up
```

Una vez levantado el contenedor, se vera asi:

<input type="checkbox"/>	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input type="checkbox"/>	 solution		Running (1/1)	0.31%		6 minutes ago	  
<input type="checkbox"/>	 sqlserver 756b5096ab36	mcr.microsoft.com/azure-sql-edge	Running	0.31%	1433:1433	6 minutes ago	  

Configuración DBeaver y connection string:

Connection string: (ubicado dentro del proyecto FrontEnd)

```
appsettings.json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Server=127.0.0.1;Database=master;User Id=sa;Password=Passw1rd;TrustServerCertificate=True"
  }
}
```

Ingresamos a DBeaver y creamos una nueva conexión:

Con usuario: **sa**

Y contraseña: **Passw1rd**

Cargamos los datos a partir de los archivos DDL.sql y DML.sql ubicados en la raíz del proyecto.

Lanzar la app:

La carpeta Publish contiene el release de la aplicación, la misma está ubicada en la raíz del proyecto.

Dentro de esta buscamos el archivo FrontEnd.exe y lo ejecutamos:

Una vez ejecutado el archivo, se abrirá la siguiente CMD:

```
Seleccionar C:\Users\Agustin Do Canto\Desktop\Semestre 5\Diseno 1\OBLIGATORIO 1 REPO\281835_281542_250230\publish\FrontEnd...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\Agustin Do Canto\Desktop\Semestre 5\Diseno 1\OBLIGATORIO 1 REPO\281835_281542_250230\publish\
```

Para acceder a la aplicación, se puede hacer click mientras se presiona la tecla Control (Ctrl) sobre la URL, o simplemente copiar y pegarla en el navegador. Ambos métodos te llevarán a la misma página y podrás comenzar a utilizar la aplicación.

ANEXOS

[Anexo 1]

▼ { } Domain	92%	35/427
> 📁 Client	98%	3/121
> 📁 Deposit	95%	3/60
> 📁 Promotion	93%	5/74
> 📁 Reservati	91%	6/66
> 📁 Notificati	90%	3/30
> 📁 Review	88%	5/41
> 📁 DateRang	72%	5/18
> 📁 Log	71%	5/17

A continuación se detallan las shadow properties implementadas en sus respectivas clases de dominio:

Client: ActionsInLog, Notifications

2 usages 126 tests OK

```
public List<Log> ActionsInLog { get; set; }
```

2 usages 126 tests OK

```
public List<Notification> Notifications { get; set; }
```

Deposit: Promotions, Reviews, Disponibility

```
public List<Promotion?> Promotions { get; set; }
```

1 usage 81 tests OK

```
public List<Review> Reviews { get; set; } Content
```

1 usage 81 tests OK *

```
public List<DateRange> Disponibility { get; set; }
```

Promotion: Deposits

1 usage More...

```
public List<Deposit> Deposits { get; set; }
```

Reservation: Deposit, Client

2 usages 2 tests OK * More...

```
public Deposit? Deposit { get; set; }
```

15 usages 39 tests OK *

```
public int DepositId { get; set; } Proper
```

```
public Client Client { get; set; } Proper
```

4 usages 40 tests OK *

```
public string ClientEmail { get; set; } P
```

Notification: Client

```
public Client Client { get; set; } Proper
```

2 usages 6 tests OK

```
public string ClientEmail { get; set; }
```

Review: Deposit

```
public Deposit Deposit { get; set; }  
  
2 usages 3 tests OK *  
public int DepositId { get; set; } P
```

DateRange: Deposit

```
public int DepositId { get; set; } P  
  
public Deposit Deposit { get; set; }
```

Log: Client

```
public string ClientEmail { get; set; }  
public Client Client { get; set; } Non
```

[Anexo 2]

Diagramas de clases

Diagrama de clases (BusinessLogic)

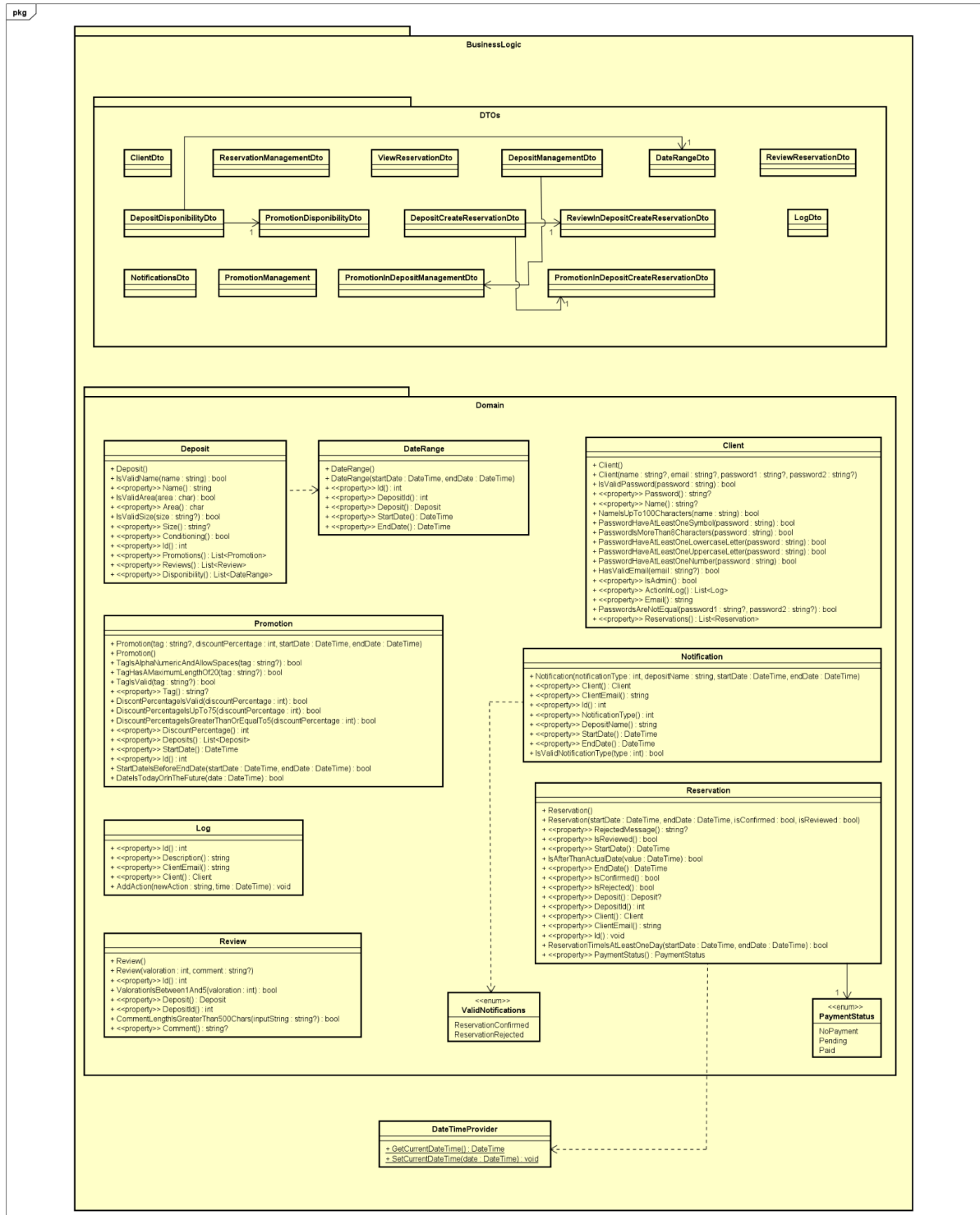


Diagrama de clases (DataAccess)

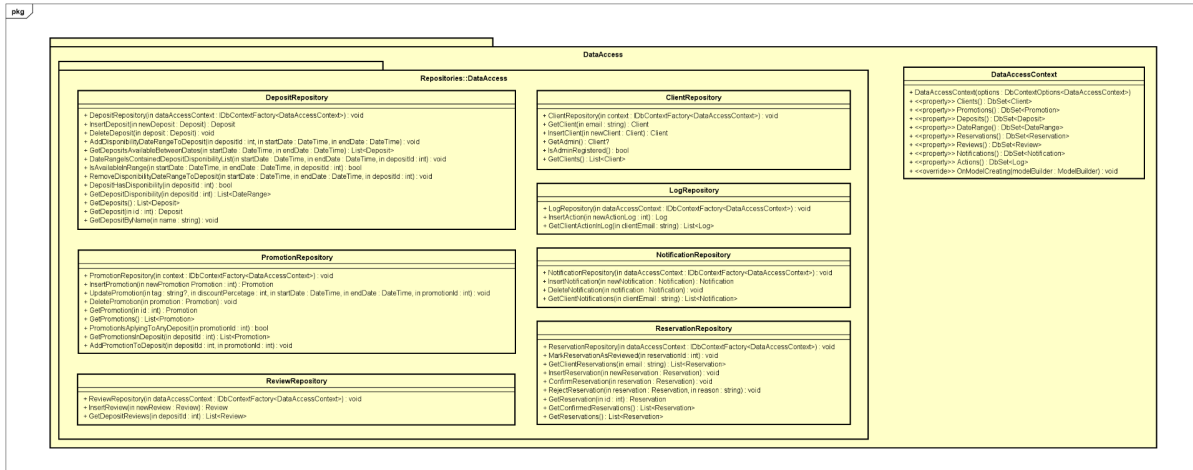
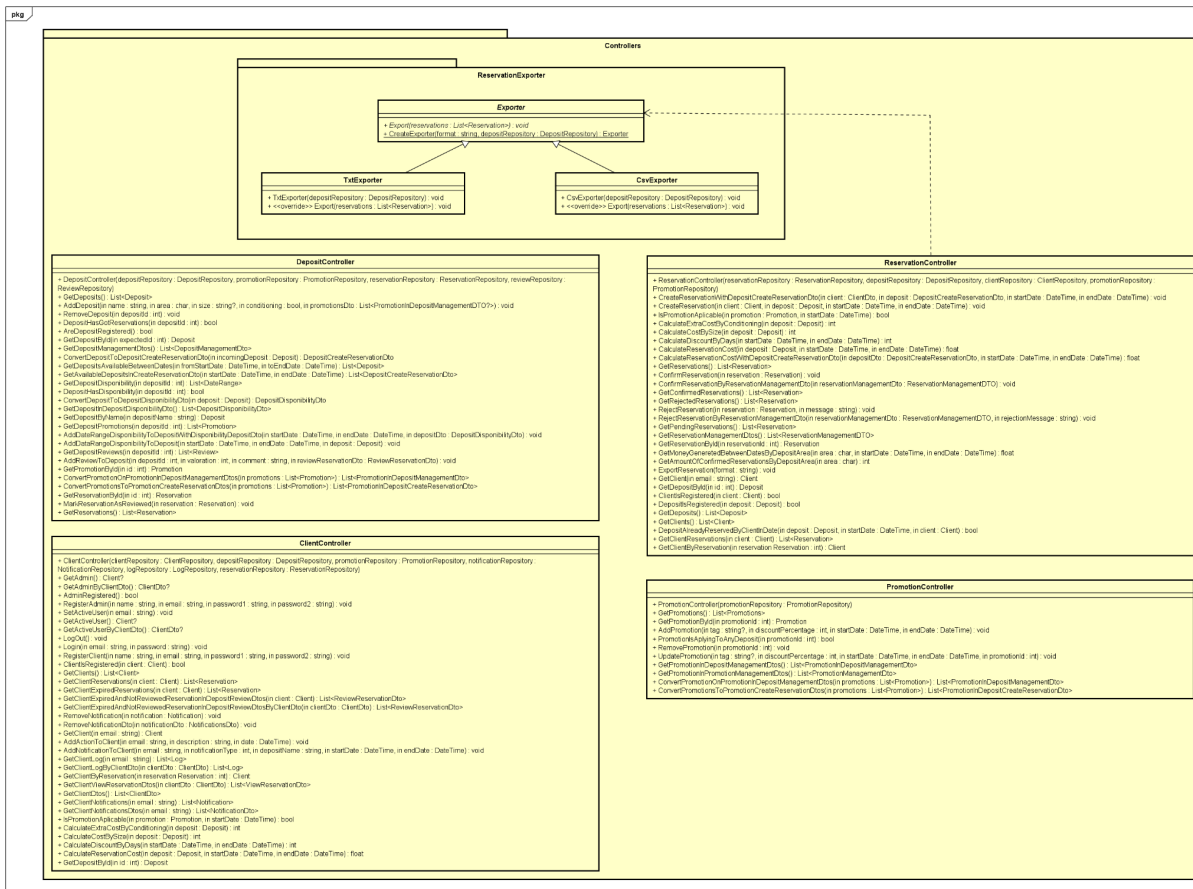
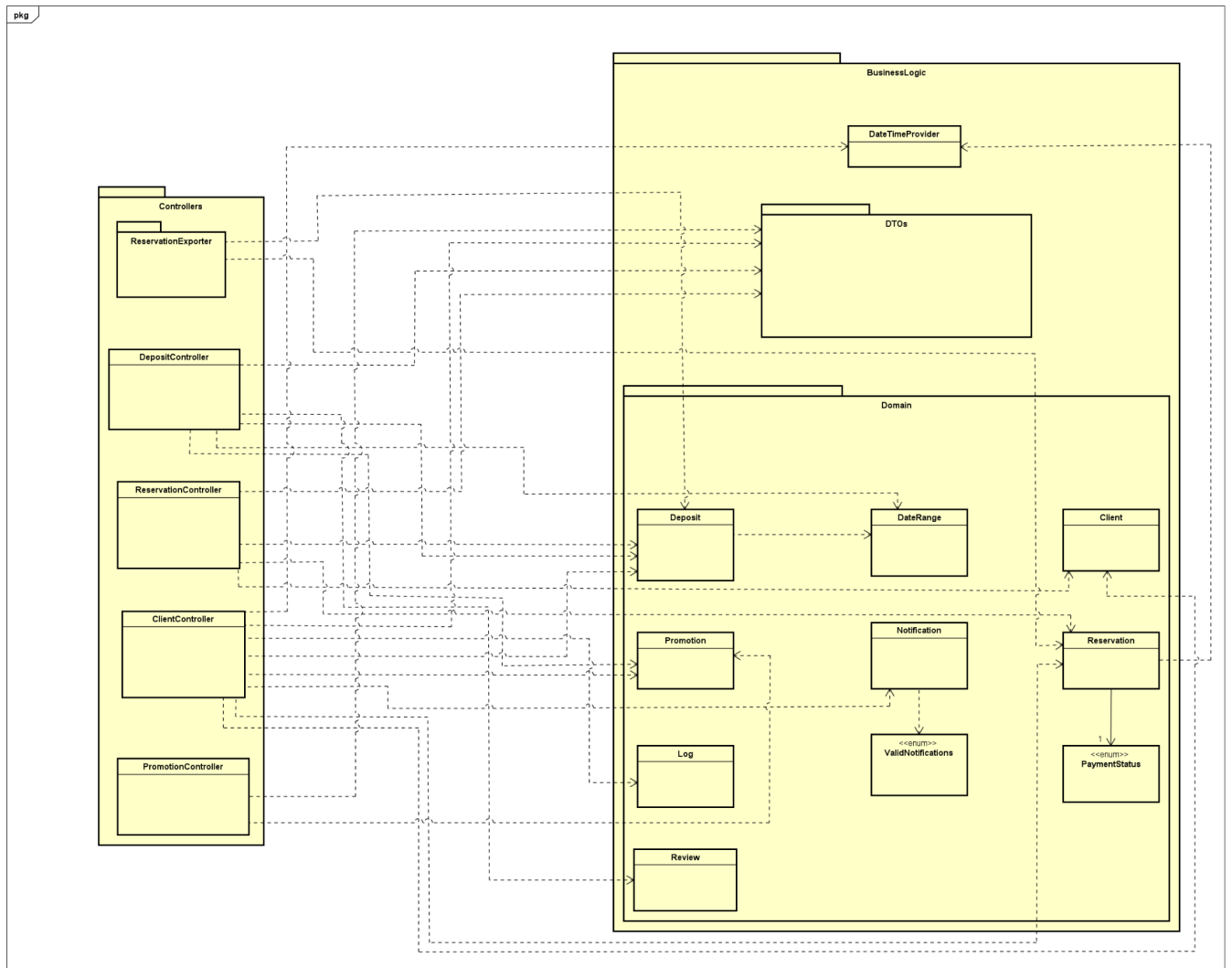


Diagrama de clases (Controllers)



[Anexo 3]

Diagrama de relación entre clases (Controllers -> BusinessLogic)



[Anexo 4]

Función Export en CsvExporter :

```
public override void Export(List<Reservation> reservations)
{
    StringBuilder sb = new StringBuilder();
    sb.AppendLine("Deposito , Reserva , Pago");
    var downloadsPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.UserProfile), "Downloads");
    string baseFileName = "reservas.csv";
    string filePath = Path.Combine(downloadsPath, baseFileName);

    if (File.Exists(filePath))
    {
        int counter = 1;
        while (File.Exists(filePath))
        {
            string newFileName = $"{Path.GetFileNameWithoutExtension(baseFileName)}_{counter}{Path.GetExtension(baseFileName)}";
            filePath = Path.Combine(downloadsPath, newFileName);
            counter++;
        }
    }

    foreach (var reservation in reservations)
    {
        var deposit = DepositRepository.GetDeposit(reservation.DepositId);
        string conditioning = deposit.Conditioning ? "Si" : "No";
        string paymentStatus;
        if (reservation.PaymentStatus == PaymentStatus.Paid)
        {
            paymentStatus = "Capturado";
        }
        else if (reservation.PaymentStatus == PaymentStatus.Pending)
        {
            paymentStatus = "Reservado";
        }
        else
        {
            paymentStatus = "Devuelto";
        }

        sb.AppendLine($"Nombre:({deposit.Name}) Area: ({deposit.Area}) Tamaño: ({deposit.Size}) Calefaccion ({conditioning}) , " + Ty
            $"Id: ({reservation.Id}) Fecha de inicio: ({reservation.StartDate}) Fecha de fin: ({reservation.EndDate}) , " +
            $"Estado del Pago: {paymentStatus}");

        File.WriteAllText(filePath, contents: sb.ToString());
    }
}
```

Función export en TxtExporter:

```
public override void Export(List<Reservation> reservations)
{
    StringBuilder sb = new StringBuilder();
    sb.AppendLine("Deposito \t Reserva \t Pago");
    var downloadsPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.UserProfile), "Downloads");
    string baseFileName = "reservas.txt";
    string filePath = Path.Combine(downloadsPath, baseFileName);

    if (File.Exists(filePath))
    {
        int counter = 1;
        while (File.Exists(filePath))
        {
            string newFileName = $"{Path.GetFileNameWithoutExtension(baseFileName)}_{counter}{Path.GetExtension(baseFileName)}";
            filePath = Path.Combine(downloadsPath, newFileName);
            counter++;
        }
    }
}
```

```
foreach (var reservation in reservations)
{
    var deposit = DepositRepository.GetDeposit(reservation.DepositId);
    string conditioning = deposit.Conditioning ? "Si" : "No";
    string paymentStatus;
    if (reservation.PaymentStatus == PaymentStatus.Paid)
    {
        paymentStatus = "Capturado";
    }
    else if (reservation.PaymentStatus == PaymentStatus.Pending)
    {
        paymentStatus = "Reservado";
    }
    else
    {
        paymentStatus = "Devuelto";
    }

    sb.AppendLine($"Nombre:({deposit.Name}) Area: ({deposit.Area}) Tamaño: ({deposit.Size}) Calefaccion ({conditioning}) \t " +
        $"Id: ({reservation.Id}) Fecha de inicio: ({reservation.StartDate}) Fecha de fin: ({reservation.EndDate}) \t " +
        $"Estado del Pago: {paymentStatus}");

    File.WriteAllText(filePath, sb.ToString());
}
}
```


[Anexo 5]

Logo DEPOQUICK

