



Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio 1

Diseño de aplicaciones 1

Joaquín Corallo – 153651

Martin Cabrera – 185091

Rodrigo Miranda – 233827

Mayo 2024

Link al repositorio de Github:	2
Descripción General:	3
Descripción y justificación de diseño	3
Mantenibilidad y extensibilidad	7
Análisis de dependencias	8
Cobertura de pruebas unitarias	9
Administración de Depósitos y Alta de depósito Casos de Prueba	11
Anexo	11

Link al repositorio de Github:

https://github.com/IngSoft-DA1-2023-2/233827_153651_185091

Descripción General:

DepoQuick es una iniciativa emergente que busca simplificar el proceso de alquiler de depósitos, ofreciendo una solución eficiente y fácil de usar para aquellos que necesitan espacio adicional de almacenamiento. El objetivo principal es desarrollar un Producto Mínimo Viable (MVP) que demuestre la viabilidad de la idea y sirva como base para futuras iteraciones y mejoras.

El sistema de DepoQuick estará compuesto por cuatro componentes principales: la lógica de negocio (Business Logic y controllers en nuestro proyecto), el dominio, los repositorios (Persistence y repositorys en el proyecto), y la interfaz de usuario (desarrollada con Blazor). Estos componentes estarán separados para facilitar el mantenimiento y la escalabilidad del sistema. La aplicación se desarrollará utilizando C# en .NET Core 6, con Blazor Server App como tecnología para la interfaz de usuario. Además, se utilizará GitHub como repositorio de código para el control de versiones.

Bugs Conocidos:

A veces al crear varias Promociones o Depósitos seguidos la aplicación se puede caer debido a un error de Blazor Radezen, lo que hay que hacer aquí es ir creando e intercalando con otras pestañas, no quedarse en la de creación.

Al realizar una booking, las reviews no se muestran correctamente.

Descripción y justificación de diseño

Diagrama de paquetes:

Namespace	Clase	Responsabilidad
Domain	User	Representa la entidad de usuario en el dominio de la aplicación.
Domain	Promotion	Representa una promoción en el sistema.
Domain	Deposit	Representa un depósito en el sistema.

Domain	Booking	Representa una reserva de depósito en el sistema.
Domain	Log	Representa un objeto con información para mostrar el registro de eventos o actividades relevantes en el sistema
Domain	Review	Representa un comentario realizado por el usuario en el sistema.
Domain.Enums	Area	Representa las posibles áreas que pueden ser asignadas a los depósitos.
Domain.Enums	Size	Representa los posibles tamaños que pueden ser asignados a los depósitos.
Domain.Enums	EventType	Representa las posibles acciones del usuario que estamos interesados en guardar para su posterior análisis.
Persistence	UserRepository	Encargada de interactuar con el almacenamiento persistente para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar), relacionadas con la entidad de usuario.
Persistence	PromotionRepository	Responsable de interactuar con el almacenamiento persistente para realizar operaciones CRUD relacionadas con las promociones.
Persistence	DepositRepository	Responsable de interactuar con el almacenamiento persistente para realizar operaciones CRUD relacionadas con los depósitos.
Persistence	BookingRepository	Responsable de interactuar con el almacenamiento persistente para realizar operaciones CRUD relacionadas con las reservas.
Persistence	IRepository	La interfaz proporciona una abstracción sobre las operaciones de almacenamiento, permitiendo que las clases de repositorio concretas

		implementen la lógica específica de acceso a datos para entidades particulares. Aquí definimos funciones básicas como agregar, modificar, borrar y traer una o todas.
Persistence	LogRepository	Responsable de interactuar con el almacenamiento persistente para realizar operaciones CRUD relacionadas con los logs.
Persistence	ReviewRepository	Responsable de interactuar con el almacenamiento persistente para realizar operaciones CRUD relacionadas con los comentarios acerca de las reservas.
BusinessLogic.Controllers	BookingController	Encargada de gestionar las solicitudes relacionadas con las reservas dentro de la aplicación.
BusinessLogic.Controllers	DepositController	Encargada de gestionar las solicitudes relacionadas con los depósitos dentro de la aplicación.
BusinessLogic.Controllers	PromotionController	Encargada de gestionar las solicitudes relacionadas con las promociones dentro de la aplicación.
BusinessLogic.Controllers	ReviewController	Encargada de gestionar las solicitudes relacionadas con los comentarios acerca de los depósitos dentro de la aplicación.
BusinessLogic.Controllers	UserController	Encargada de gestionar las solicitudes relacionadas con los usuarios dentro de la aplicación.
BusinessLogic.Services	LogService	Encargada de gestionar la creación y escritura de registros de logs en el sistema. Proporciona métodos para registrar eventos o actividades relevantes como por ejemplo, LogLogin, LogLogout, etc.
BusinessLogic.Services	PricingService	Encargada de calcular el precio de los depósitos, considerando factores como la duración del depósito, si tiene aire

		acondicionado, el tamaño y cualquier promoción aplicable.
BusinessLogic.Services	StastService	Calcula estadísticas relacionadas con las reservas, como las ganancias totales en un período de tiempo específico y la cantidad de reservas para un mes y año dados.
BusinessLogic.Validators	BookingValidator	Valida la consistencia y la integridad de los objetos de reserva.
BusinessLogic.Validators	PromotionValidator	Valida la consistencia y la integridad de los objetos de promoción
BusinessLogic.Validators	ReviewValidator	Valida la consistencia y la integridad de los objetos que representan los comentarios de los depósitos
BusinessLogic.Validators	UserValidator	Valida la consistencia y la integridad de los objetos de usuario
BusinessLogic.Validators	UuidValidator	Valida que el Uuid sea válido y que cumpla con el estándar
BusinessLogic	SessionLogic	Gestiona la lógica de sesión de usuario, incluyendo el inicio de sesión, cierre de sesión, y la verificación del estado de inicio de sesión.

Descripción de las principales decisiones de diseño tomadas:

Se optó por dividir el proyecto en distintas capas como "Domain", "Business Logic" y "Repository" con el objetivo de reducir el acoplamiento entre los distintos componentes del sistema y proporcionar una clara separación de responsabilidades. La capa de "Domain" encapsula las entidades y objetos de negocio, la capa de "Business Logic" contiene la lógica de negocio y es independiente del almacenamiento, en esta capa es donde colocamos nuestros controladores, y por último la capa "Repository" maneja la interacción con el almacenamiento persistente, por este motivo la llamamos en nuestro proyecto "Persistence". Esta división mejora la cohesión dentro de cada capa, ya que cada una tiene una responsabilidad clara y específica. Para terminar, esta arquitectura nos facilita la evolución y el mantenimiento del código al permitir modificaciones en una

capa sin afectar a las demás, lo que promueve la reutilización, la escalabilidad y la legibilidad del código.

Para la interfaz de usuario utilizamos la herramienta Blazor que nos permite utilizar un mismo lenguaje de código, en este caso C#, tanto para la interfaz de usuario como para la lógica del negocio, persistencia, etc. Esto lo logramos utilizando inyección de dependencias.

Con respecto a la persistencia para este proyecto se optó por guardar los datos únicamente en memoria, con lo cual la sesión y los datos se borrarán al terminar la aplicación, pero si se trató de lograr que el sistema sea escalable y que permita en un futuro adaptarlo para la utilización de mecanismos más complejos como por ejemplo la utilización de bases de datos. Esto lo logramos creando un directorio donde colocamos nuestras clases desacopladas de repositorio.

Análisis de los criterios seguidos para asignar las responsabilidades

Como se mencionó en el punto anterior lo que buscamos al asignar las responsabilidades es desacoplar los distintos componentes. Cada clase y componente del sistema tiene una responsabilidad clara y específica, lo que facilita la comprensión y el mantenimiento del código.

Por ejemplo, tenemos el objeto `deposit` en el dominio que describe a la entidad de depósito, también creamos un enum con los posibles tamaños para que estos ya estén definidos de antemano y no haya lugar a errores de tipeo. Tenemos al controlador que define la lógica de negocios para la creación, edición, borrado, etc de los depósitos. Y el repositorio alineado más a cómo se persisten estos datos. También tenemos los validadores y servicios que están relacionados a nuestra lógica de negocio. Además tenemos excepciones específicas que creamos para que nos muestren información relevante en caso de que ocurra algún error, para poder manejar dicho error en lugar de permitir que la aplicación se caiga por completo.

Todo esto nos permite cumplir con los principios descritos anteriormente y lograr un sistema que sea limpio, mantenible y escalable.

Mantenibilidad y extensibilidad

Un buen ejemplo de cómo logramos esto, es si miramos la capa de repositorio, la misma proporciona una abstracción sobre el acceso a la base de datos, lo que facilita

el cambio de una base de datos a otra sin afectar otras partes de la aplicación. La interfaz IRepository en nuestra aplicación facilita la reutilización del código y reduce el acoplamiento entre los componentes permitiendo una fácil extensión o modificación de la funcionalidad sin afectar otras partes del código. Por ejemplo, si necesitamos cambiar la persistencia en memoria a PostgreSQL, solo debemos crear una nueva implementación de IRepository para PostgreSQL sin modificar el código que utiliza la interfaz. Esto demuestra la extensibilidad y mantenibilidad de nuestra aplicación, ya que podemos realizar cambios en la infraestructura de datos con un impacto mínimo en el resto del sistema.

Análisis de dependencias

Las clases del dominio que interactúan con PricingService son:

- **Deposit:** representa un depósito de almacenamiento y contiene la información relacionada con el depósito que se está evaluando.
- **Promotion:** representa una promoción aplicable a un depósito y contiene la información relacionada con la promoción
- **Size** (enumeración):
 - Es una enumeración que representa los posibles tamaños del depósito (Small, Medium, Large).

En términos de cohesión, PricingService tiene una cohesión alta porque todas sus funcionalidades están enfocadas en calcular el precio de un depósito en base a diversos factores relacionados con el depósito.

Se podría decir que también tiene un nivel moderado de acoplamiento, esto se debe a su dependencia directa de las clases **Deposit** y **Promotion**, ciertos cambios en la estructura de estas clases podrían llegar a requerir un cambio en PricingService para que siga funcionando correctamente. Con el equipo nos dimos cuenta de este acoplamiento pero desafortunadamente la implementación de Interfaces resultaba una reestructura bastante grande para ese entonces.

Cobertura de pruebas unitarias

Nuestro código presenta una cobertura del 97%.

Debido a un error con el uso del controlador de versiones (git), no es posible evidenciar concretamente el uso de TDD para el desarrollo de el requerimiento: **Cálculo de precio del Depósito**, en su defecto, evidenciaremos el uso de TDD para el requerimiento: **Valoraciones y Comentarios**.

Valoraciones y Comentarios:

Para comenzar con el desarrollo de este requerimiento comenzamos analizando qué nuevas clases serán requeridas, basándonos en la arquitectura ya existente. Se definió que necesitamos un ReviewController, para poder manejar la creacion, modificacion y eliminacion de las reviews, un ReviewValidator para poder validar que las review sean correctas y con el formato esperado, y una clase de dominio Reviews, que nos permitirá modelar y asignar atributos a nuestras review.

Una vez creadas estas clases en blanco, comenzamos creando los tests para las funcionalidades de cada una.

Review Controller:

Aquí sabíamos que íbamos a necesitar Crear reviews. Por lo que creamos el test **TestUserCanCreateReview()** y **TestUserCannotCreateReview()**

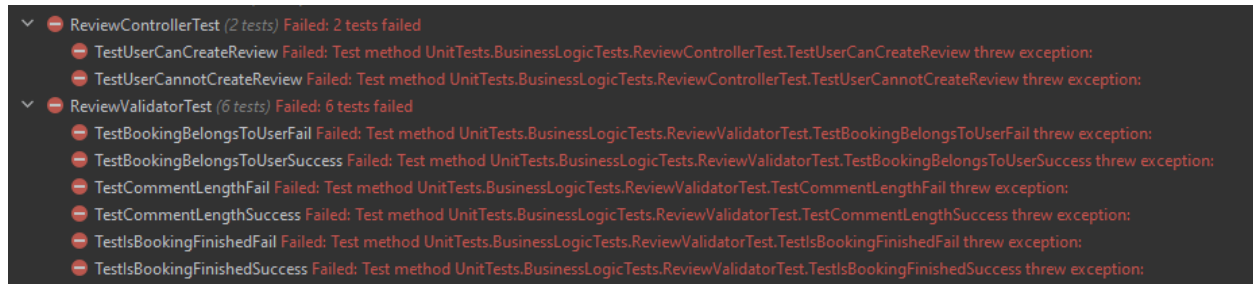
El primer test es efectuado con una review válida (parámetros correctos) mientras que el segundo es con una review invalida (parámetros incorrectos)

Review Validator:

Aquí se anotaron todos los requerimientos que una Review iba a tener que cumplir para ser considerada válida y se crearon los test pertinentes, entre ellos: **TestIsBookingFinishedFail()** y **TestBookingBelongsToUserSuccess()**

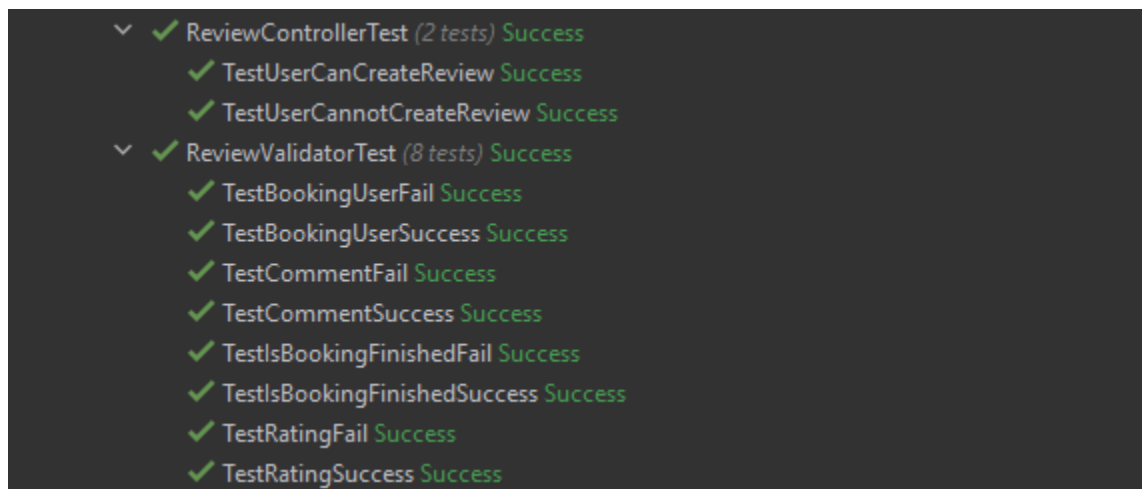
El primer test es encargado de testear si la Reserva ya había terminado, debido a que solo se puede dejar una Review una vez que esto sucede. El segundo test chequea que la reserva pertenezca al usuario logueado actualmente.

En la siguiente imagen se puede observar el resultado de la RED-STAGE:



Paso siguiente comenzó la implementación de los métodos para poder llegar a la GREEN-STAGE, ya sabiendo que Input:Output tiene que tener un método facilita la implementación del mismo, y el desarrollo de los métodos resultó sencillo.

Una vez alcanzada la GREEN-STAGE, los test se ven así:



Lo que sí ocurrió en esta GREEN-STAGE es el Renaming de algunos de los tests.

Finalmente se realizó la etapa REFRACTOR, que principalmente incluye código de Front-End debido a que no había mucho que cambiar.

Administración de Promociones y Alta de depósito

Casos de Prueba

En el caso de Alta de depósito, no hay margen de error, ya que solamente puede seguir el “Happy Path”, ya que la interfaz solo permite seleccionar valores válidos, por lo que no se permite ingresar ningún dato erróneo tampoco omitir campos, ni formatos inválidos. En el caso de prueba de creación de depósito hay unos datos por defecto y si se desea cambiar solo se puede elegir entre las opciones ya validadas previamente.

En los casos de prueba de promociones, intentamos crear una promoción con el campo de label vacía, y se muestra un error “The label is required”.

Otro caso de prueba fue ingresar en porcentaje valores fuera del rango permitido, valores negativos, 0 y mayores a 76 como 100 o más. En todos estos casos lanza un error “The percentage must be between 5 and 75” y no permite crear la promoción.

En el caso de la edición de promoción detectamos un bug que si se desea, modificar el porcentaje a un valor por fuera de los rangos, el sistema muestra un error, y no modifica el campo, pero sin embargo en la interfaz de usuario se muestra la edición como si se hubiera realizado.

Anexo

Es pertinente mencionar que el administrador va a tener acceso a un panel llamado “CREATE TEST DATA”, esto le permite al administrador crear con un botón varios datos de prueba como Usuarios, Bookings y Deposits. Hicimos esta funcionalidad debido a que por tema de validaciones, no era posible hacer una booking que ya haya terminado (ningún usuario puede reservar en el pasado) por lo que esta es la única forma eficaz de comprobar la funcionalidad de Reviews y de Stats.

The screenshot shows the 'Create deposit' form in the 'Depo Quick' application. On the left is a dark blue sidebar with navigation links: Admin Tools, Promotions, Deposits, Book a Deposit, Create a Review, View Bookings, and Logout. The main content area has a title 'Create deposit' and a table with the header 'Area'. The table contains four rows labeled A, B, C, and D. Row A is highlighted in light blue and has a small yellow cursor in the 'Area' column. Below the table, it says 'No records to display'. At the bottom left of the form is a blue 'Create' button.

Area
A
B
C
D

No records to display

Create

Create deposit

Area

D

Size

Big

☒ Air Conditioning

<input type="checkbox"/>	Label	Discount %	Valid From
No records to display.			

Create

Deposit created successfully!

Depo Quick

Admin Tools

Promotions

Create

Manage

Deposits

Create

Manage

Book a Deposit

Create a Review

View Bookings

Logout

About

Create deposit

Area

C

Size

Medium

☒ Air Conditioning

<input checked="" type="checkbox"/>	Label	Discount %	Valid From	Valid To
<input checked="" type="checkbox"/>	20OFFMAYO	20	1/5/2024 18:19:00	30/5/2024 18:19:00

Create

Depo Quick

Admin Tools

Promotions

Create

Manage

Deposits

Create

Manage

Book a Deposit







Create a Review

View Bookings

Logout

About

Manage Deposits

Deposit Id	Area	Size	Air Conditioning	
072c9310-2324-4225-abf...	B	Small	False	 
12ecd275-a7be-4c80-b4d...	C	Medium	True	 
160059aa-edc5-43db-b0...	C	Big	True	 

Depo Quick

Admin Tools

Promotions

Create

Manage

Deposits

Create

Manage

Book a Deposit








Create a Review

View Bookings

Logout

About

Manage Deposits

Deposit Id	Area	Size	Air Conditioning	
072c9310-2324-4225-abf...	<div>B</div>	<div>Medium</div>	<input type="checkbox"/> Air Conditioning	  
12ecd275-a7be-4c80-b4d...	C	Medium	True	 
160059aa-edc5-43db-b0...	C	Big	True	 

Depo Quick

Admin Tools

Promotions

Deposits

Create

Manage

Book a Deposit

Create a Review

View Bookings

Logout

About


Create Booking

Select a deposit to book:

ID	Area	Size	Air Conditioning
e011d499-e03f-...	A	Small	<input checked="" type="checkbox"/>


Book from:

13/5/2024 0:00:00



Book to:

20/5/2024 0:00:00



CHECK BOOKING PRICING

Booking price:

\$465,5

SEND BOOKING REQUEST

Booking request sent successfully! Please wait for confirmation. Thank you!

- ▶ Admin Tools
- ★ Admin Register
- 📄 Logs
- 📊 Statistics
- 📅 Manage Bookings
- ▶ Promotions
- + Create
- 🔍 Manage
- ▶ Deposits
- + Create

Create promotion

Label:

Discount Percentage:

Valid percentages are from 5 to 75

Valid From:

Valid To:

Create

The label is required.

- ▶ Admin Tools
- ★ Admin Register
- 📄 Logs
- 📊 Statistics
- 📅 Manage Bookings
- ▶ Promotions
- + Create
- 🔍 Manage
- ▶ Deposits
- + Create

Create promotion

Label:

Discount Percentage:

Valid percentages are from 5 to 75

Valid From:

Valid To:

Create

The percentage must be between 5 and 75.

- ▶ Admin Tools
- ★ Admin Register
- 📄 Logs
- 📊 Statistics
- 📅 Manage Bookings
- ▶ Promotions
- + Create
- 🔍 Manage
- ▶ Deposits
- + Create

Create promotion

Label:

Discount Percentage:

Valid percentages are from 5 to 75

Valid From:

Valid To:

Create

The 'from' date cannot be greater than the 'to' date.

Depo Quick

Admin Tools

Admin Register

Logs

Statistics

Manage Bookings

Promotions

Create

Manage

About

ManagePromotions

Promotion Id	Label	Percentage	Start Date	End Date	
f10da669-8ebd-417...	<div></div>	10	9/5/2024 20:04:59	30/5/2024 20:04:00	<div><div>✓</div><div>✕</div><div></div></div>
848f9cf7-5a62-400...	<div>Label is required15OFFMAYO</div>	15	11/5/2024 20:12:00	1/5/2024 20:12:00	<div><div></div><div></div><div></div></div>