

Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio 1

Entregado como requisito para la materia Diseño de Aplicaciones 1

[Enlace a Repositorio GitHub](#)

Florencia Brum – 243045
Nicolas Rodriguez – 265403
Rodolfo Presno- 174039

Docentes: Santiago Perez, Pablo Benitez

15/10/2024

Indice

Contents.....	2
Declaración de Autoría	3
Descripción general del trabajo	4
Descripción y justificación del diseño	5
Diagramas	5
Diagrama de paquetes.....	5
Diagrama del proyecto Dominio	5
Asignación de responsabilidades y justificación de criterio	6
Paquete Dominio.Data	6
Paquete Dominio.Interfaces.....	6
Paquete DTOs.UserDTOs	6
Paquete Dominio.Models	7
Paquete Dominio.Services.....	7
Dependencias generadas FrontEnd - BackEnd	8
Mecanismos generales y decisiones de diseño:	8
Responsabilidades.....	10
Mantenibilidad y extensibilidad	10
Análisis de dependencias: Manejo de archivos de importación	10
Cobertura de pruebas Unitarias	11
Pruebas unitarias	11
Cobertura de pruebas	11
Video presentación y ejecución de los casos de prueba unitaria.....	12
Interfaz	12
Funcionalidades no implementadas/ bugs/ Issues para la siguiente entrega.....	13
Bugs	13
Issues	13
Posibles mejoras	13
Conclusión	14
A considerar	14
Usuario administrador	14
Bibliografía	15
Anexos	16
Anexo 1: Diagramas.....	16
Anexo 2: Pruebas Unitarias.....	22
Anexo 3: Imágenes de la interfaz de usuario.	24

Declaración de Autoría

Nosotros, Florencia Brum, Nicolas Rodríguez y Rodolfo Presno, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizamos el Obligatorio 1;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado juntamente con otros, hemos explicado claramente qué fue construido por otros, y qué fue construido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Descripción general del trabajo

La finalidad de este informe es demostrar la aplicación de los conceptos y técnicas aprendidos en el curso Diseño de Aplicaciones 1

El obligatorio consta de la creación de una página web que ayude a organizar las tareas de uno o varios equipos, parecido al programa ya existente Trello. La página ofrece distintas acciones que se pueden realizar, dependiendo de los permisos de cada usuario. Algunas de las acciones son la alta, baja y modificación de: usuarios, equipos, paneles y tareas; siendo los dos primeros exclusivos para administradores.

En el transcurso del proyecto se utilizaron conceptos como la programación orientada a objetos, representación de diagramas de clases y paquetes mediante UML, TDD (ver anexo 2) y la escritura de código aplicando Clean Code.

Entre las técnicas de Clean Code que implementamos destacan:

Funciones cortas: Las funciones que realizamos están enfocadas en realizar una sola tarea y ser cortas.

Nombres claros y Descriptivos: Buscamos que el nombre de nuestras variables, métodos y Properties sean lo más claros posibles, de tal manera que con solo leerlos ya saber a qué se refiere.

Organización en carpetas: buscamos organizar el código en paquetes y carpetas para facilitar la comprensión.

Evitar código duplicado: no se repite el mismo bloque de código en diferentes lugares.

El proyecto se realizó utilizando el framework .NET, el entorno de pruebas MSTest y el lenguaje C#, y en la interfaz se implementó el framework de UI Blazor.

Para el control de versiones se utilizó GitHub y se siguió el flujo de trabajo de Gitflow. Debido a esto, en nuestro repositorio contamos con las ramas:

- **main** (versión estable),
- la rama **develop**, creada por nosotros para el desarrollo y pruebas, en la cual integramos todas las características listas
- las ramas **feature**, en donde creábamos las nuevas características sin afectar el resto del desarrollo.

Para la separación de componentes entre la lógica de negocio y la interfaz de usuario, se crearon los proyectos “Dominio”, dentro del cual se implementó la lógica de negocio con los servicios, modelos, databases o repositorios; e “InterfazWeb”.

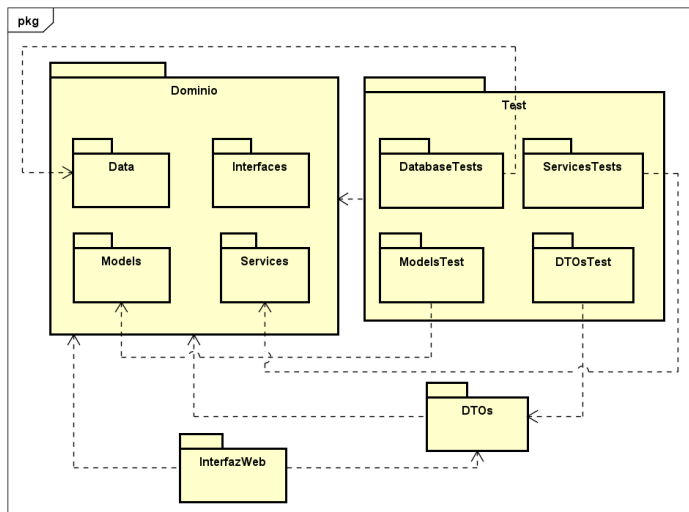
La funcionalidad de la carga de tareas desde un archivo .csv la implementamos basándonos en sugerencias obtenidas de ChatGPT. Esta conversación puede ser encontrada en el siguiente link: [Enlace a Chat](#) o el siguiente link de Youtube: https://youtu.be/O6sb8u_qMuE

Descripción y justificación del diseño

Diagramas

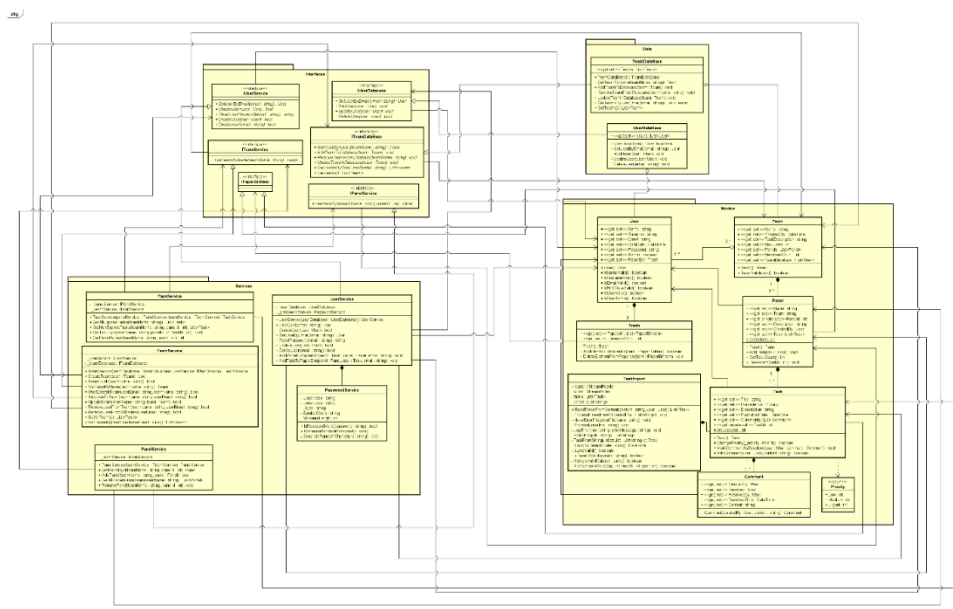
En esta parte se encuentran los diagramas de paquetes y del proyecto Dominio. Para más diagramas dirigirse a la sección de Anexos (ver anexo 2)

Diagrama de paquetes



Como se puede apreciar, separamos el proyecto en 4 paquetes: Dominio, InterfazWeb, DTOs y Tests. A su vez separamos Dominio y Tests en subpaquetes, cada subpaquete representa las carpetas en las que se encuentran las clases de cada proyecto (a nivel programación).

Diagrama del proyecto Dominio



Asignación de responsabilidades y justificación de criterio

Paquete Dominio.Data

El paquete Data funciona de Base de Datos, y las clases TeamDataBase y UserDataBase almacenan las instancias de equipos y usuarios respectivamente.

Paquete Dominio.Data		
Namespace	Clase	Responsabilidad
Dominio.Data	TeamDataBase	Servir de base de datos de Equipos.
Dominio.Data	UserDataBase	Servir de base de datos de Usuarios.

Paquete Dominio.Interfaces

El paquete Interfaces centraliza las interfaces declaradas. Decidimos implementarlas para interactuar con las DataBases de Dominio.Data y los servicios de Dominio.Services.

Paquete Dominio.Interfaces		
Namespace	Clase	Responsabilidad
Dominio.Interfaces	IPanelService	Interfaz para PanelService
Dominio.Interfaces	IPanelBinItem	Interfaz para PanelBinItem
Dominio.Interfaces	ITeamDatabase	Interfaz para TeamDatabase
Dominio.Interfaces	ITeamService	Interfaz para TeamService
Dominio.Interfaces	IUserDatabase	Interfaz para UserDatabase
Dominio.Interfaces	IUserService	Interfaz para UserService

Paquete DTOs.UserDTOs

Para esta primera entrega implementamos DTOs para el manejo de la información de los usuarios.

Namespace	Clase	Responsabilidad
DTOs.UserDTOs	UserLoginDTO	Implementa DTO para los datos del login y sus requerimientos.
DTOs.UserDTOs	UserModifyDTO	Implementa DTO para la modificación de datos de usuario.
DTOs.UserDTOs	UserRegisterDTO	Implementa DTO para el registro de nuevos usuarios y sus requerimientos.

Paquete Dominio.Models

Decidimos centralizar las entidades básicas del en el paquete Models, éstas son los comentarios, tareas, paneles, equipos y usuarios, así como sus datos. También incluimos la clase Trash que usamos para implementar el borrado de elementos, y la clase TaskImport, que contiene métodos para extraer las tareas de un listado de estas en formato .csv.

Namespace	Clase	Responsabilidad
Dominio.Models	Comment	Almacenar datos de un comentario.
Dominio.Models	Panel	Almacenar datos de un panel y sus tareas.
Dominio.Models	Task	Almacenar datos de una tarea y sus comentarios, y métodos para operar sobre los comentarios.
Dominio.Models	TaskImport	Auxiliar en la importación de tareas desde archivos .csv y generar archivos de error.
Dominio.Models	Team	Almacenar datos de equipos, sus usuarios y paneles. Validación de equipos.
Dominio.Models	Trash	Almacenar elementos eliminados (paneles, tareas) por un mismo usuario.
Dominio.Models	User	Almacenar datos de un usuario, y métodos de validación.

Paquete Dominio.Services

En el paquete Services almacenamos los métodos de la lógica de negocio.

Namespace	Clase	Responsabilidad
Dominio.Services	PanelService	Almacenar métodos para obtener y modificar vínculos entre equipos y paneles.
Dominio.Services	PasswordService	Almacenar métodos para validar y generar contraseñas de usuarios.
Dominio.Services	TaskService	Almacenar métodos para obtener y modificar vínculos entre paneles y tareas.
Dominio.Services	TeamService	Almacenar métodos para obtener y modificar equipos, sus usuarios y modificar sus vínculos.
Dominio.Services	UserService	Almacenar métodos auxiliares de usuarios, vínculos con elementos de papelería.

Dependencias generadas FrontEnd - BackEnd

Las dependencias que utilizamos en la aplicación para enviar información desde la interfaz gráfica hacia las capas de lógica de negocio fueron mediante Singletons.

En program.cs creamos todos los singleton necesarios, entre ellos para la clase PasswordService. Estos Singleton son los que le permitieron al proyecto blazor reconocer la capa lógica. Después en cada una de las páginas .razor creadas utilizamos @using para referirnos al namespace de la clase que utilizamos; y @inject para referirnos a la clase en particular.

Mecanismos generales y decisiones de diseño:

Para el diseño del software, utilizamos el patrón de diseño “Arquitectura en Capas”.

Siguiendo este patrón, separamos el programa en 4 capas, donde cada una tiene su propósito específico y conoce únicamente su capa inferior.

Esta separación en capas nos permite asegurar que cada capa se desarrolle y mantenga de forma independiente. Esto nos permite agregar o intercambiar funcionalidades sin afectar otras partes del sistema.

Capa de Interfaz Web

Esta es la capa con la que el usuario interactúa. Aquí se crearon las “vistas” donde cada una de ellas tiene un fin.

Desde cada vista se realizan consultas a los servicios necesarios, así sea como obtener todos los usuarios de un equipo, o todos los paneles de un equipo. Una vez que recibe la información se muestra en pantalla en tiempo real.

Capa de Servicios

En esta capa se encuentra la lógica de negocio. Aquí es donde implementamos las operaciones que la aplicación necesita. Esta capa se comunica con la capa de dominio y Databases/Repositorios, lo que permite la separación entre la capa de dominio y la de interfaz web.

Esta capa consta 5 servicios.

- Panel Service, encargado de manejar los paneles con funciones tales como obtener un panel mediante su id.
- Task Service, encargado de manejar las tareas con funciones como obtener todas las tareas expiradas de un equipo.
- Team Service, encargado de manejar los equipos con funciones como agregar usuario a un equipo
- User Service, encargado de manejar los usuarios con funciones como la de crear usuario.
- Password Service, encargado del manejo de las contraseñas con funciones como la de generar una contraseña aleatoria.

Capa Dominio

En la capa dominio se modelaron los datos en las clases User, Task, Panel, Team, Comment.

En cada entidad no encontraremos métodos complejos, sino más bien validaciones y propiedades para describir a los objetos.

En esta capa no se involucra la lógica de negocio o en cómo se guardan los datos.

Capa Database o Repositorio

En la capa de database o repositorio encontraremos las operaciones para interactuar con la base de datos, o en nuestro caso las operaciones para interactuar con las listas que nos permiten guardar en memoria los datos.

De este modo los servicios no precisan conocer la forma en la que se guardan los datos y se mantiene una separación entre la lógica de negocio y almacenamiento.

Uso de interfaces

Para la implementación de este programa, hicimos enfoque en la utilización de interfaces.

Las interfaces nos permiten el desacoplamiento entre componentes, esto nos ayudó a reducir las dependencias entre clases, y nos permitió realizar modificaciones en las funciones sin tener que afectar a las clases que utilizan esas interfaces.

Además, nos permitió poder realizar pruebas unitarias de manera más efectiva utilizando Mocks. Para ello utilizamos un paquete el cual descargamos e instalamos desde NuGet.

El uso de Mocks nos facilitó para poder simular las interfaces sin tenerlas implementadas.

Por ejemplo, para realizar la prueba unitaria UserService precisábamos la interfaz de IUserDataBase, para ello utilizamos un Mock de IUserDataBase lo cual nos permitió controlar el comportamiento del mismo durante los tests. (ver anexo 2 para imagen)

Por otro lado, el uso de interfaces nos permitió la aplicación de polimorfismo que nos fue muy útil para la papelera de reciclaje de los usuarios.

Definimos una interfaz llamada "IPaperBinItem" donde hicimos que Panel y Task que son los elementos que pueden ser enviados a la papelera, implementaran esta interfaz.

De momento no tiene métodos, pero nos permitió crear una lista de IPaperBinItems e incluir las Tasks y Panels en la misma lista. A futuro podremos implementar un método eliminar donde Tasks Service y Panel Service pueda implementarlo, y que al llamarlo desde PaperBin se eliminen.

Responsabilidades

En cuanto a la asignación de responsabilidades se refiere, nos aseguramos de que cada clase tuviera una única responsabilidad clara y definida.

De esta manera sabemos que cada capa y clase se encarga de una parte específica del sistema.

Entidades del Dominio: Representar y modelar los objetos que son necesarios para la lógica de negocio.

Servicios: implementar la lógica de negocio de cada entidad. Cada servicio interactúa como intermediario entre las entidades y sus respectivos repositorios o databases.

Repositorios: Acceso y almacenamiento de los datos

Mantenibilidad y extensibilidad

Para facilitar y mantener la extensibilidad del proyecto creamos interfaces y clases de services. Nuestro proyecto Blazor interactúa con estas interfaces y no directamente con las clases que utilizamos como database o las clases de cada elemento del programa (Panel, Task, Team, etc.). Esto permite realizar cambios o agregar nuevas funcionalidades de manera sencilla.

El uso de interfaces nos será útil para la segunda entrega a la hora de integrar la base de datos, ya que al haber desacoplado la forma en la que se almacenan los datos tendremos simplemente que cambiar de un sistema de almacenamiento a otro sin tocar parte de la lógica del negocio.

Aun así, se podría mejorar mucho más la mantenibilidad si la interfaz web interactuara únicamente con los Services. Esto también planeamos implementar para la siguiente entrega.

Análisis de dependencias: Manejo de archivos de importación





Para el manejo de archivos de importación se creó la clase TaskImport. Esta clase se encuentra en el dominio e interactúa únicamente con la clase Task. Esta interacción se ve en los atributos de la clase TaskImport, que cuenta con una lista de Task.

Ya que tenemos esta lista de Task en TaskImport, si la clase Task dejara de existir, la clase TaskImport dejaría de funcionar. Lo cual tiene sentido, porque para poder cargar tareas, tiene que existir el elemento tarea.

Cobertura de pruebas Unitarias

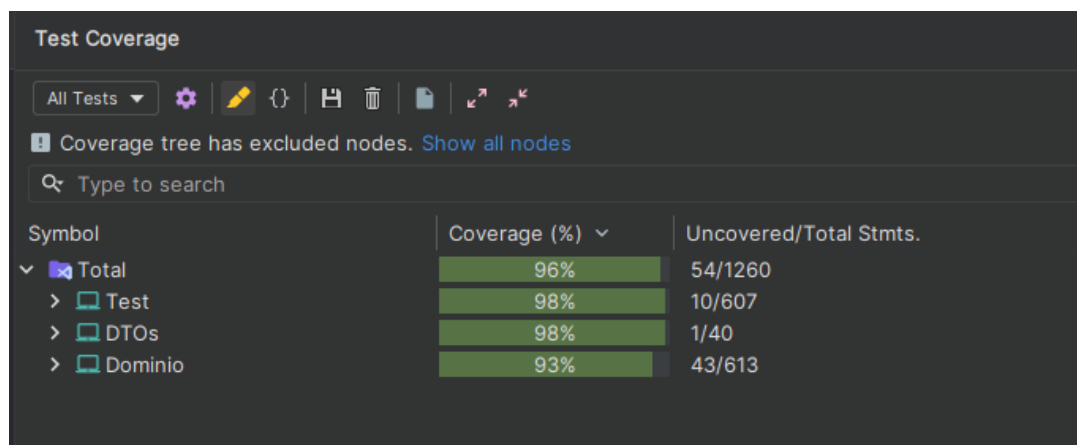
Pruebas unitarias

Para los casos de prueba con los que testear el código escrito, tuvimos en cuenta los datos requeridos en cada clase, y los requerimientos dados en la consigna. Como mencionamos anteriormente, se usó TDD para el avance de la programación y la creación del código.

GREEN: Pasa prueba FromEntity de UserLoginDTO Florence authored and Florence committed 2 days ago	 bb30598 
RED: Falla prueba ToEntity de UserModifyDTO Florence authored and Florence committed 2 days ago	 55e8747 

Ejemplo de TDD para las pruebas de los DTOs

Cobertura de pruebas



En el **anexo 3 - Coverage de las Pruebas Unitarias en Test** mostramos los % de cobertura de los tests implementados. En el caso de los tests del paquete Dominio, cada clase presenta una cobertura del 95% o más salvo un caso: la clase TaskImport, que tiene una cobertura del 69%. Las líneas que no están cubiertas son principalmente las responsables de crear el log de errores al importar tareas, y de algunas validaciones de datos de operaciones intermedias. En un inicio implementamos todas los métodos públicos, lo que permitió testear directamente cada metodo, pero reconsideramos y llegamos a la conclusión de que no era una buena práctica, ya que los métodos públicos deben ser los mínimos necesarios: los que luego serán accedidos por otras clases. Por esto hay validaciones de datos intermedios que no son posibles de testear directamente. En cuanto a la cobertura del log de errores, la clase lee un archivo, y simultáneamente exporta otro archivo .txt a la carpeta de la solución. Lo más exhaustivo hubiera sido testear estos archivos de salida contra los que deberían haberse generado en pruebas, esto no se hizo por la complejidad agregada de leer el nuevo archivo. Se opto por hacer una comprobación manual.

Video presentación y ejecución de los casos de prueba unitaria

Para demostrar la funcionalidad del ABM de usuarios, realizamos un video. [Enlace a video.](#)

Interfaz

Se utilizó Blazor en combinación con Bootstrap para la implementación de la interfaz del proyecto. El uso de Blazor permitió tener todas las partes que conforman la página web (HTML, CSS y C#) en una misma página. Mientras que Bootstrap facilitó el diseño visual.

Decidimos que los equipos, paneles y tareas se vean como tarjetas. Para esto creamos tarjetas personalizadas para cada modelo, una para equipo, una para panel, y dos para tareas, una para las tareas vencidas y la otra para las no vencidas.

Para el manejo de datos utilizamos el elemento EditForm¹, que se puede ver en las clases de registro y modificación de usuarios, equipos, paneles y tareas. Estos EditForm fueron basados en el EditForm dado en clase, pero modificándolo para que cumpla con las necesidades del código.

También, para el manejo de mensajes con los que el usuario tuviera que interactuar, se utilizó modal².

Para el manejo de sesión se creó una clase Session que se encuentra en la carpeta services dentro del proyecto Blazor. Esta misma se utiliza en las clases Login y Logout, y a su vez en todo el proyecto Blazor para asegurarnos que solo los usuarios con permisos adecuados pudieran acceder a las distintas páginas.

Para imágenes de la interfaz, ver anexo 4

¹ EditForm: Se utiliza para crear formularios interactivos y para el manejo de datos.

² Modal: Es un recuadro que aparece sobre la página, bloqueando las funciones de esta y se centra en una acción particular, por lo que el usuario solo puede hacer dicha acción o cerrar la ventana.

Funcionalidades no implementadas/ bugs/ Issues para la siguiente entrega

Bugs

Al momento de nuestras últimas pruebas no tenemos bugs detectados.

Issues

1. En la presente entrega, el primer usuario asignado a cada equipo es el mismo administrador que lo crea. Cambiar este comportamiento para que el administrador pueda asignar a un usuario (regular o no), que luego podrá modificar el equipo independientemente de ser administrador o no. (Derivado de preguntas en el foro)
2. Agregar funcionalidad de poder modificar el equipo al que está asociado un panel. (Derivado de preguntas en el foro)
3. Actualmente se puede elegir la contraseña al registrar un nuevo usuario, cambiarlo a que la misma sea generada automáticamente y se muestre al administrador por única vez. Análogamente modificar el comportamiento al modificar usuario y modificar su contraseña. (Derivado de preguntas en el foro)

Posibles mejoras

1. Si bien las bases de datos de Equipos e Usuarios las implementamos en sus clases ~DataBase, no hicimos lo mismo con Tasks, Comments, y Panel. Pendiente para la siguiente entrega.

Conclusión

En conclusión, este proyecto pudo demostrar la lógica de dominio y la interfaz de usuario, cumpliendo con la mayoría de los requerimientos establecidos. El usuario es capaz de interactuar con la interfaz, lo que demuestra la efectividad de las técnicas de programación orientada a objetos y de TDD aplicadas en el transcurso del desarrollo

A pesar de que, por la falta de tiempo, no se pudieron implementar ciertas clases, lo que se realizó en el proyecto fue suficiente para entender la forma de programar siguiendo TDD, lo cual permite la mantenibilidad y calidad del código.

A considerar

Usuario administrador

El usuario administrador que viene pre-cargado tiene mail admin@taskpanel.com y contraseña Admin123\$

Bibliografía

Bootstrap. (s.f.). *Alerts*. Obtenido de Bootstrap:

<https://getbootstrap.com/docs/4.0/components/alerts/>

Bootstrap. (s.f.). *Cards*. Obtenido de Bootstrap:

<https://getbootstrap.com/docs/4.0/components/card/>

Bootstrap. (s.f.). *Flex*. Obtenido de Bootstrap:

<https://getbootstrap.com/docs/4.0/utilities/flex/>

Bootstrap. (s.f.). *Grid system*. Obtenido de Bootstrap:

<https://getbootstrap.com/docs/5.0/layout/grid/>

Bootstrap. (s.f.). *Modal*. Obtenido de Bootstrap:

<https://getbootstrap.com/docs/4.0/components/modal/>

Bootstrap. (s.f.). *Sizing*. Obtenido de Bootstrap:

<https://getbootstrap.com/docs/4.0/utilities/sizing/>

Microsoft. (26 de 08 de 2024). *Información general de formularios de ASP.NET Core Blazor*. Obtenido de Microsoft Learn - ASP.NET Core:

<https://learn.microsoft.com/es-es/aspnet/core/blazor/forms/?view=aspnetcore-6.0#editform>

Snippet: hacer una ventana modal (modal box) con Javascript. (s.f.). Obtenido de

Didacticode: <https://didacticode.com/snippet-ventana-modal-modal-box-con-javascript/#:~:text=Una%20ventana%20modal%20o%20modal,acci%C3%B3n%20o%20cerrar%20la%20ventana.>

Anexos

Anexo 1: Diagramas.

Nos tomamos la libertad de no incluir todas las flechas cuando se trataba de los subdiagramas de Dominio, para que fuera más fácil la comprensión. Pero en el diagrama completo (que se encuentra en el apartado Diagramas del documento), si se encuentran todas las flechas.

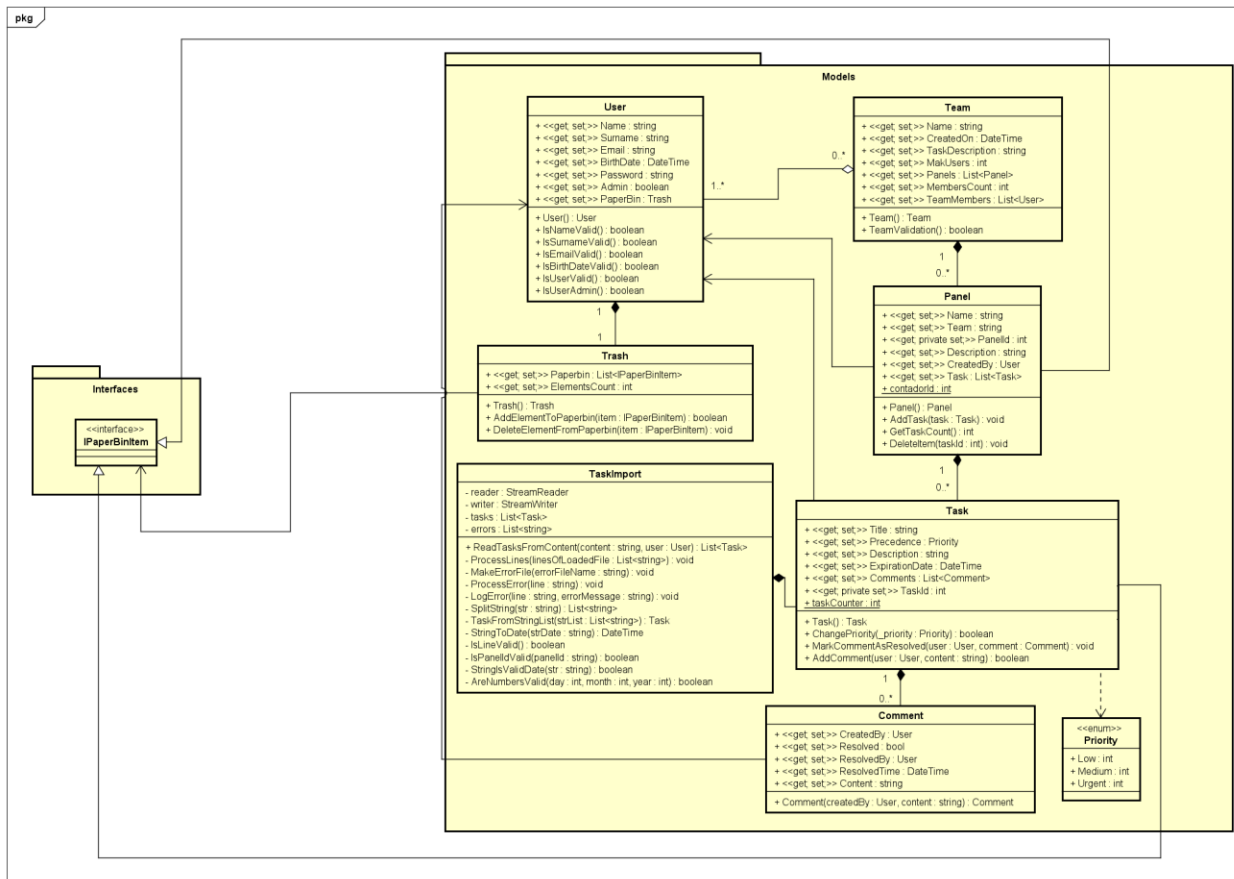


Diagrama de la carpeta Models (dentro de Dominio)

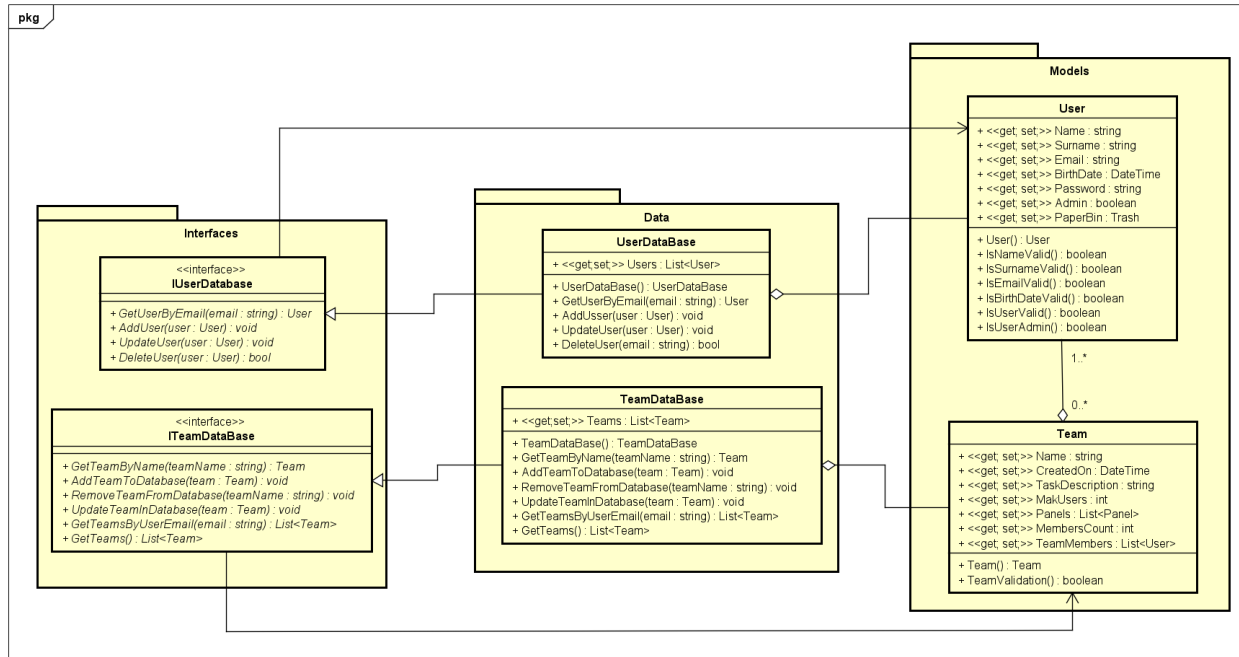


Diagrama de la carpeta Data (dentro de Dominio)

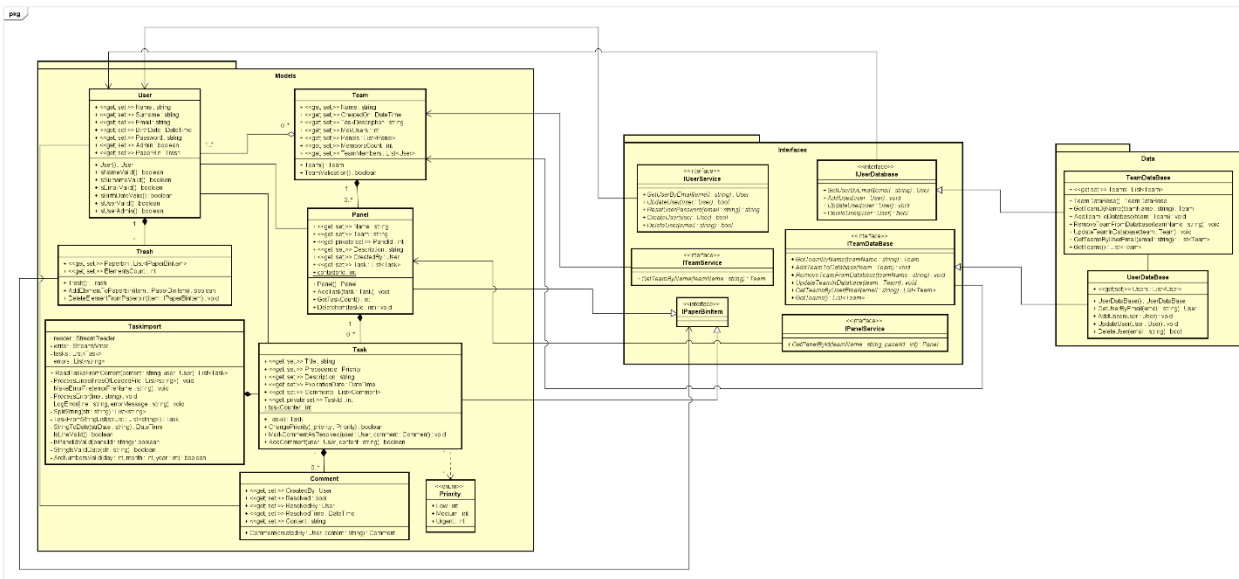


Diagrama de la carpeta Interfaces (dentro de Dominio)

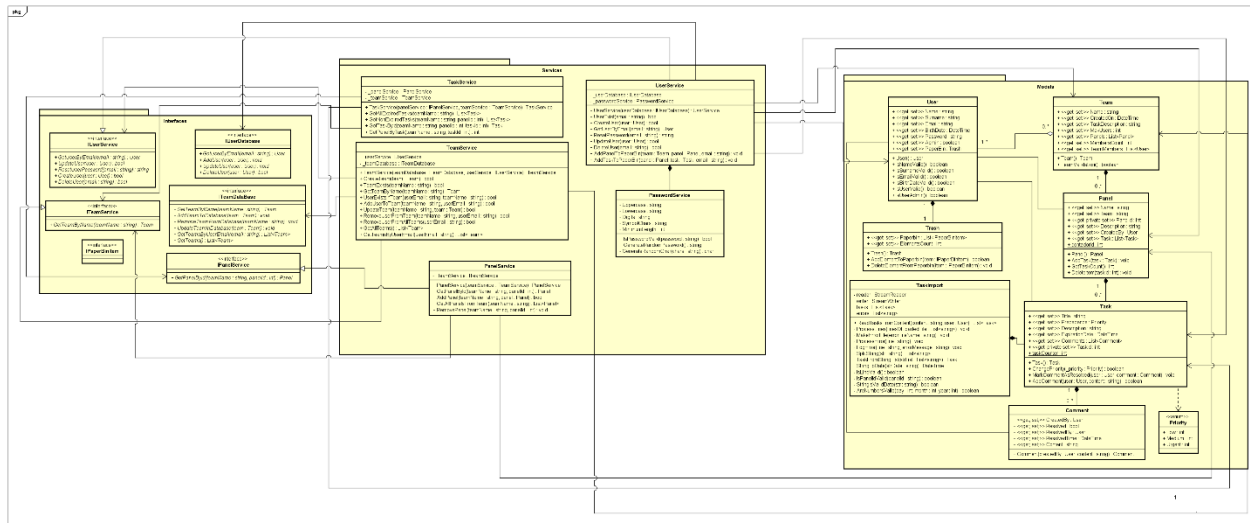
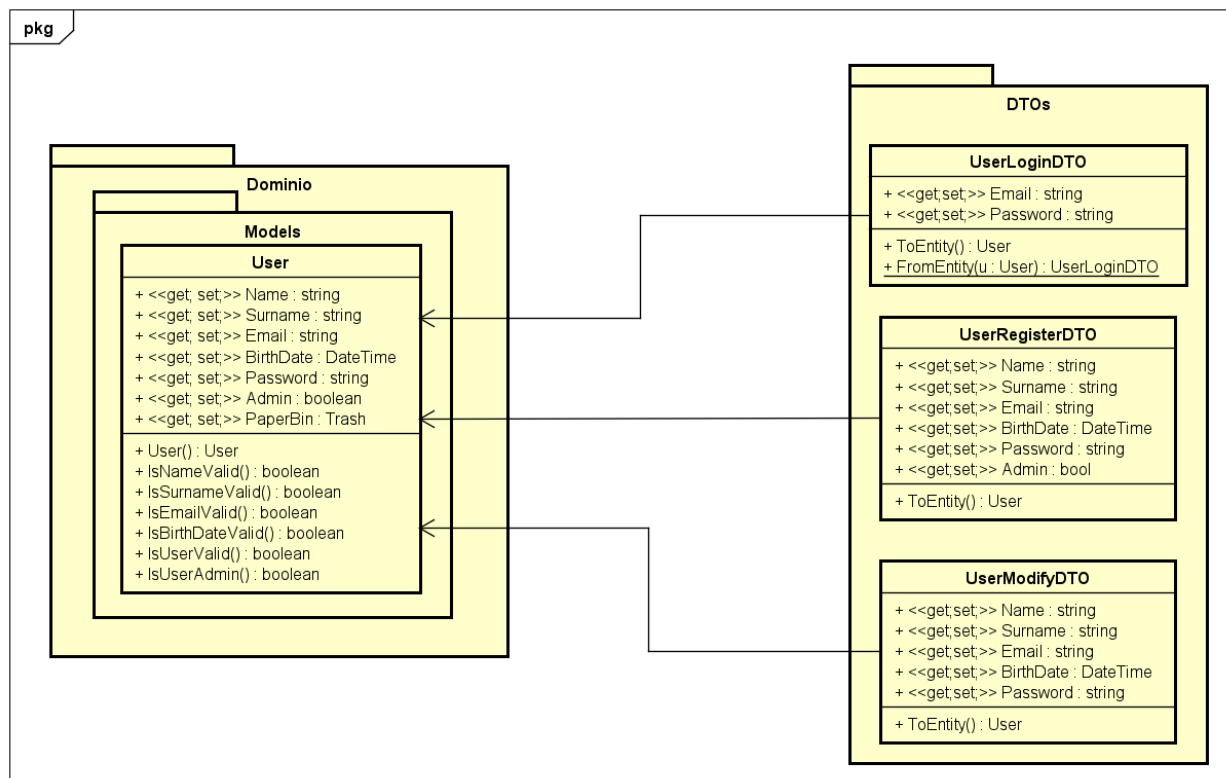


Diagrama de la carpeta Services (dentro de Dominio)



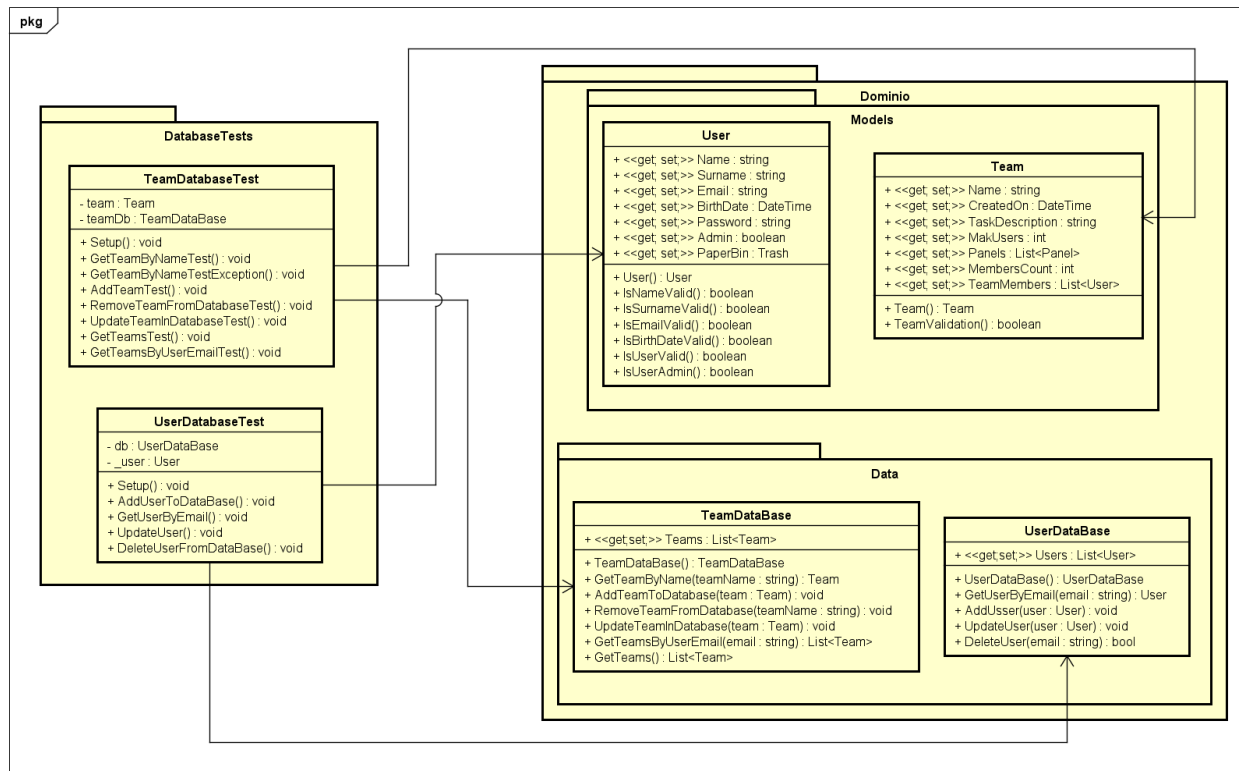


Diagrama de la carpeta DatabaseTests (dentro de Tests)

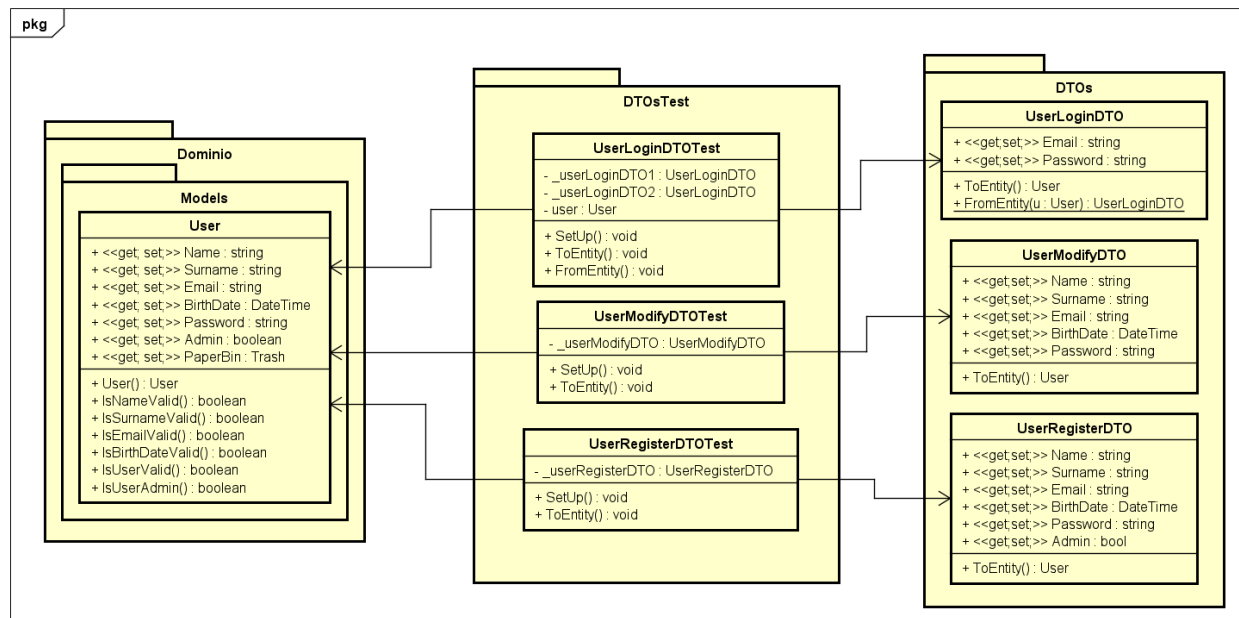
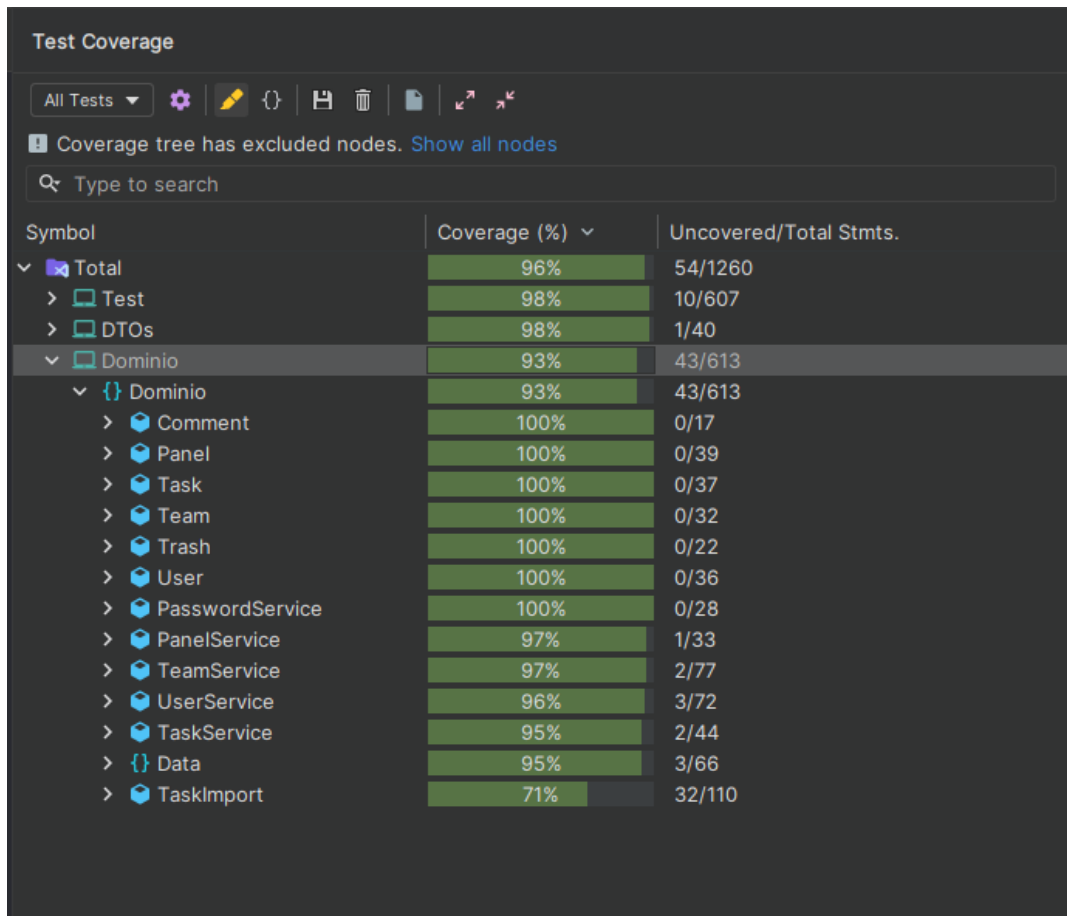
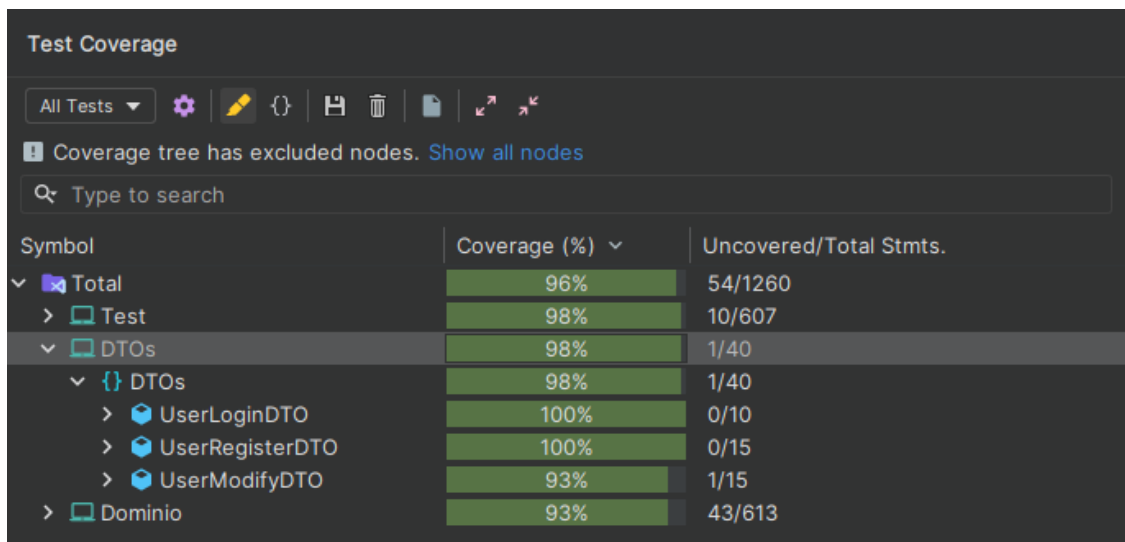


Diagrama de la carpeta DTOsTest (dentro de Tests)

Anexo 2: Pruebas Unitarias

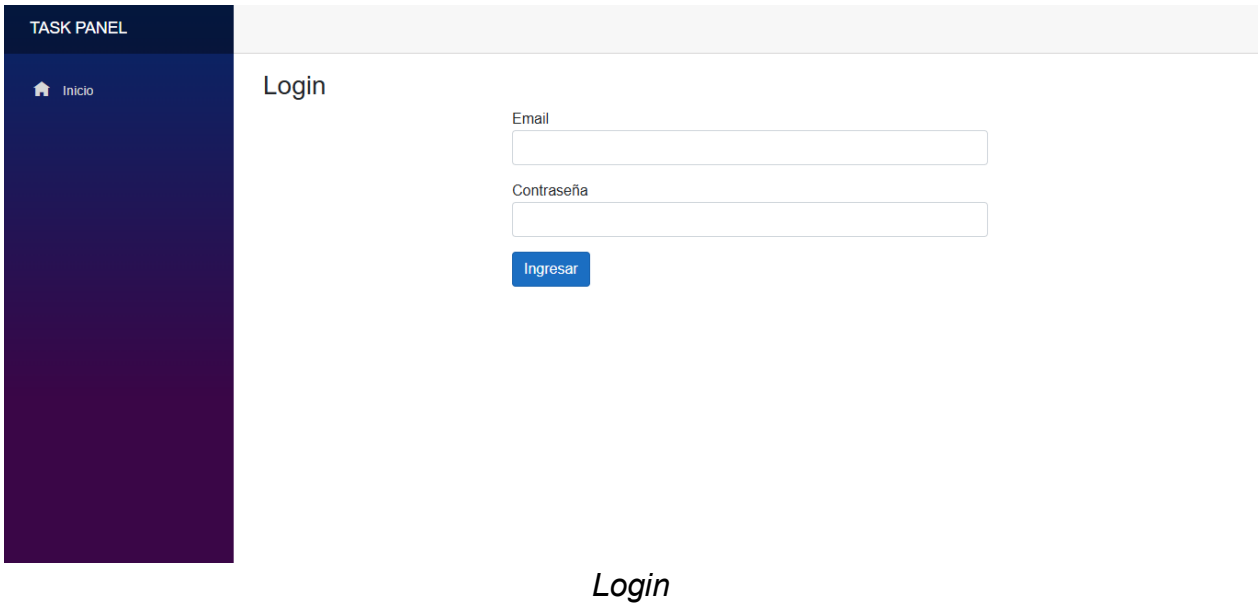


Coverage de las Pruebas Unitarias en Dominio

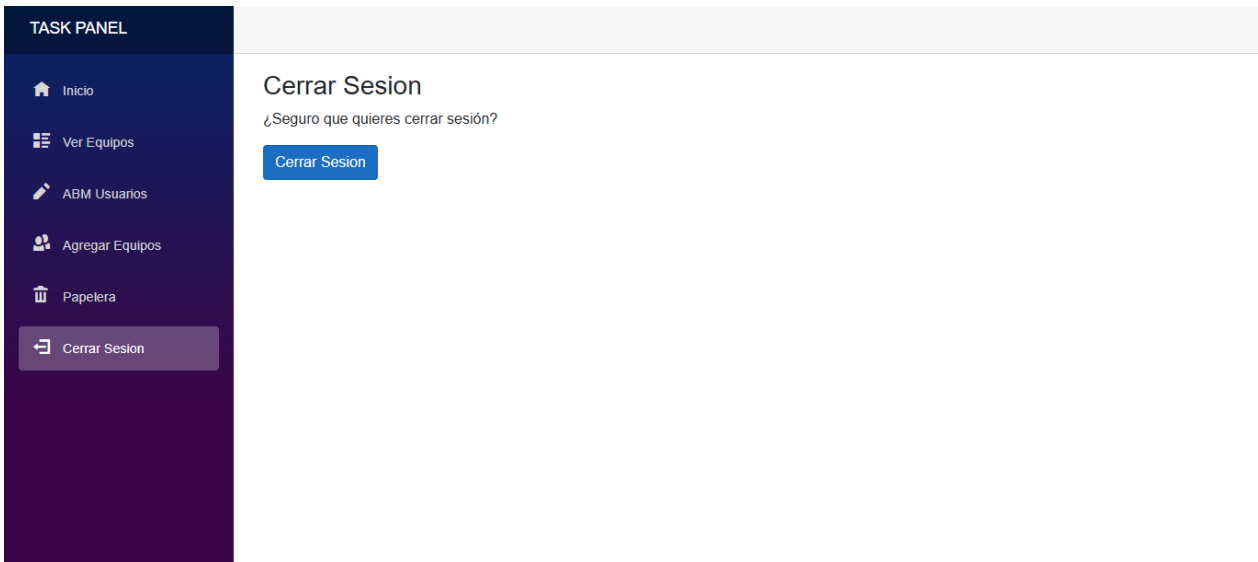


Coverage de las Pruebas Unitarias en DTOs

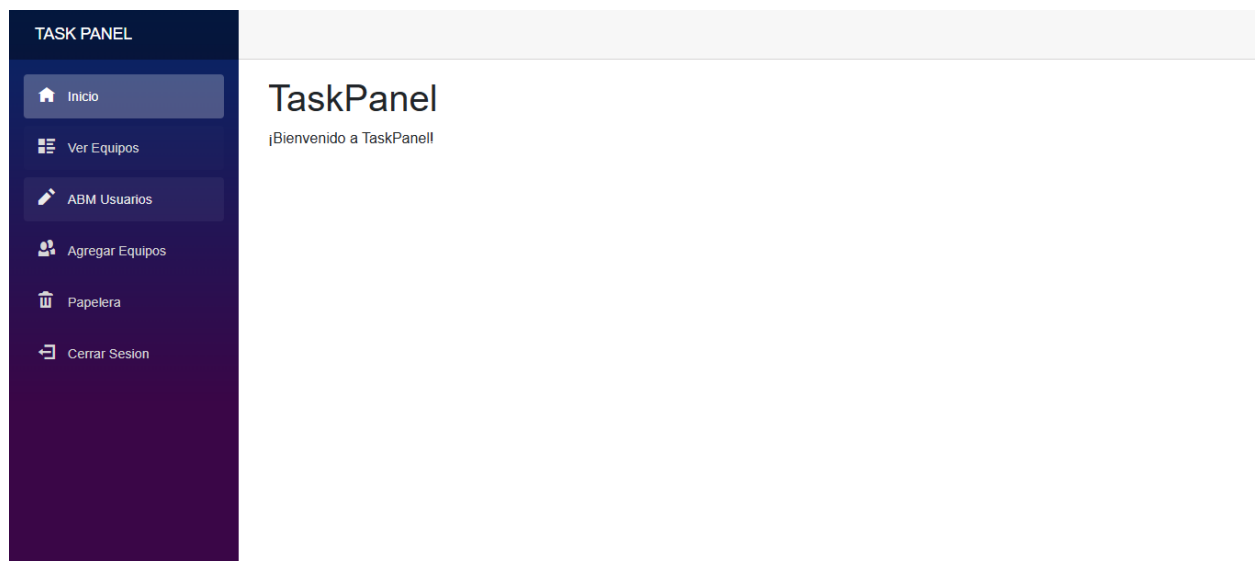
Anexo 3: Imágenes de la interfaz de usuario.



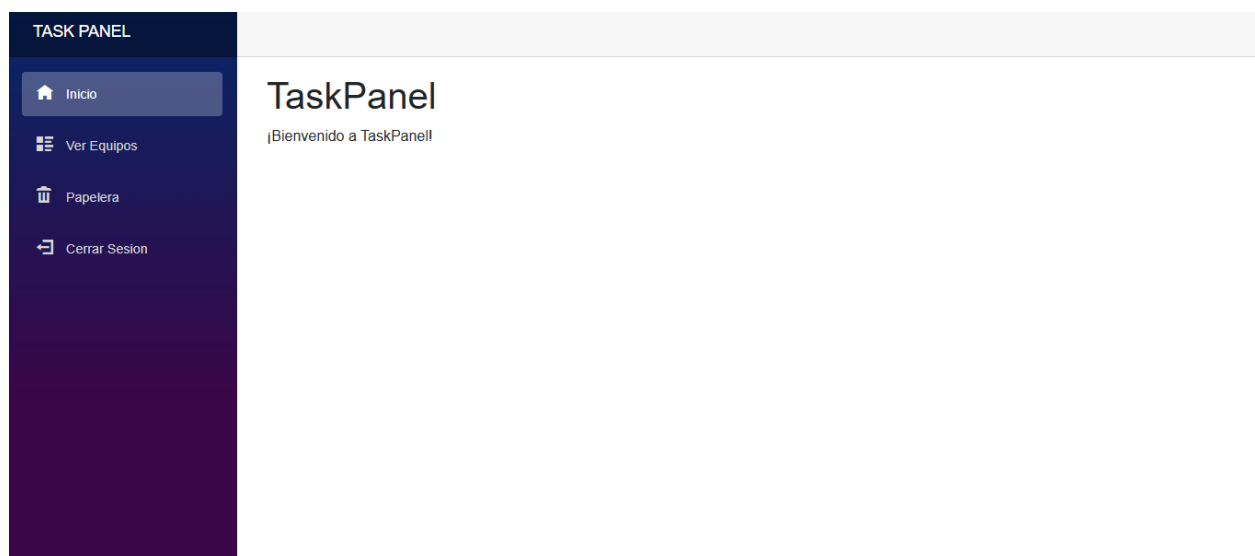
Login



Logout



Inicio usuario administrador



Inicio usuario común

TASK PANEL

Inicio

Ver Equipos

ABM Usuarios

Agregar Equipos

Papelera

Cerrar Sesión

Alta/Baja/Modificación de usuarios

Agregar Usuarios

Nombre	Apellido	Email	Fecha de Nacimiento	Administrador	Acciones
Super	Admin	admin@taskpanel.com	30/8/2000 12:00:00 AM	True	<div>Modificar</div> <div>Eliminar</div>
Lola	Peres	lperes@mail.com	4/1/2004 12:00:00 AM	False	<div>Modificar</div> <div>Eliminar</div>

Volver

Visualización de usuarios (admin)

TASK PANEL

Inicio

Ver Equipos

ABM Usuarios

Agregar Equipos

Papelera

Cerrar Sesión

Modificar Usuario

Nombre

Lola

Apellido

Peres

Fecha de Nacimiento

04/01/2004

Contraseña

Generar contraseña

Lola123+

Guardar Cambios

Atras

Modificación de usuarios (admin)

TASK PANEL

Inicio

Ver Equipos

ABM Usuarios

Agregar Equipos

Papelera

Cerrar Sesión

Alta/Baja/Modificación de usuarios

Agregar Usuarios

Nombre	Apellido	Email	Fecha de Nacimiento	Administrador	Acciones
Super	Admin	admin@taskpanel.com	30/8/2000 12:00:00 AM	True	<div>Modificar</div> <div>Eliminar</div>
Lola	Peres	lperes@mail.com	4/1/2004 12:00:00 AM	False	<div>Modificar</div> <div>Eliminar</div>

Volver

Confirmar Eliminación

¿Estás seguro de que deseas eliminar el usuario lperes@mail.com?

Cancelar

Confirmar

Eliminar usuarios (admin)

TASK PANEL

Inicio

Ver Equipos

ABM Usuarios

Agregar Equipos

Papelera

Cerrar Sesión

Equipos

Equipo1

Fecha de creación: 15/10/2024 3:52:20 AM

Miembros: 2/ 20

Descripción de tareas: Tareas

Paneles

Borrar

Modificar

Visualización de equipos de admin. El de usuarios no tiene el botón de borrar ni el de modificar

TASK PANEL

Inicio

Ver Equipos

ABM Usuarios

Agregar Equipos

Papelera

Cerrar Sesión

Agregar Equipo

Nombre

Cantidad de Usuarios

1

Descripción de tareas

Agregar	Nombre	mail
<input type="checkbox"/>	Lola	lperes@mail.com

Usuarios a agregar: (1)

• Super (admin@taskpanel.com)

Crear Equipo

Volver

Agregar equipos (admin)

TASK PANEL

Inicio

Ver Equipos

ABM Usuarios

Agregar Equipos

Papelera

Cerrar Sesión

Paneles del equipo Equipo1

Crear Panel

Tareas Vencidas

Panel1

DescripcionPanel

Ir

Modificar

Eliminar

Visualización paneles

TASK PANEL

Inicio

Ver Equipos

ABM Usuarios

Agregar Equipos

Papelera

Cerrar Sesión

Crear Panel

Nombre

Descripcion

Confirmar Panel

Volver

Crear panel

TASK PANEL

Inicio

Ver Equipos

Papelera

Cerrar Sesión

Tareas

Crear Tarea

Importar tareas desde .csv

Volver a Paneles

Tarea1

Prioridad: Low

Vencimiento: 24/10/2024 12:00:00 AM

Descripción de la tarea:
DescripciónTarea

Agregar Comentario

Modificar

Eliminar

Visualización de tareas (user). El de admin no tiene el botón de importar tareas

TASK PANEL

Inicio

Ver Equipos

ABM Usuarios

Agregar Equipos

Papelera

Cerrar Sesión

Crear Tarea

Nombre

Tarea1

Prioridad

Low

Descripción

DescripciónTarea

Fecha de expiración

24/10/2024

Crear Tarea

Volver

Crear tarea

TASK PANEL

Inicio

Ver Equipos

Papelera

Cerrar Sesión

Importar Tareas

Choose File

 tareas.csv

Archivo cargado.

Agregar Tarea?	Título	Descripción	Fecha de vencimiento
<input type="checkbox"/>	Contactar al cliente	Contactar al cliente para actualizar el estado del proyecto.	2024/9/10
<input type="checkbox"/>	Pagar proveedores	Revisar planilla de proveedores y pagar.	2024/9/19

Agregar tareas a panel

Volver

Importar Tareas (user)

TASK PANEL

Inicio

Ver Equipos

Papelera

Cerrar Sesión

Tareas vencidas del equipo Equipo1

Contactar al cliente

Prioridad: Low

Vencimiento: 10/9/2024 12:00:00 AM

Descripción de la tarea:
Contactar al cliente para actualizar el estado del proyecto.

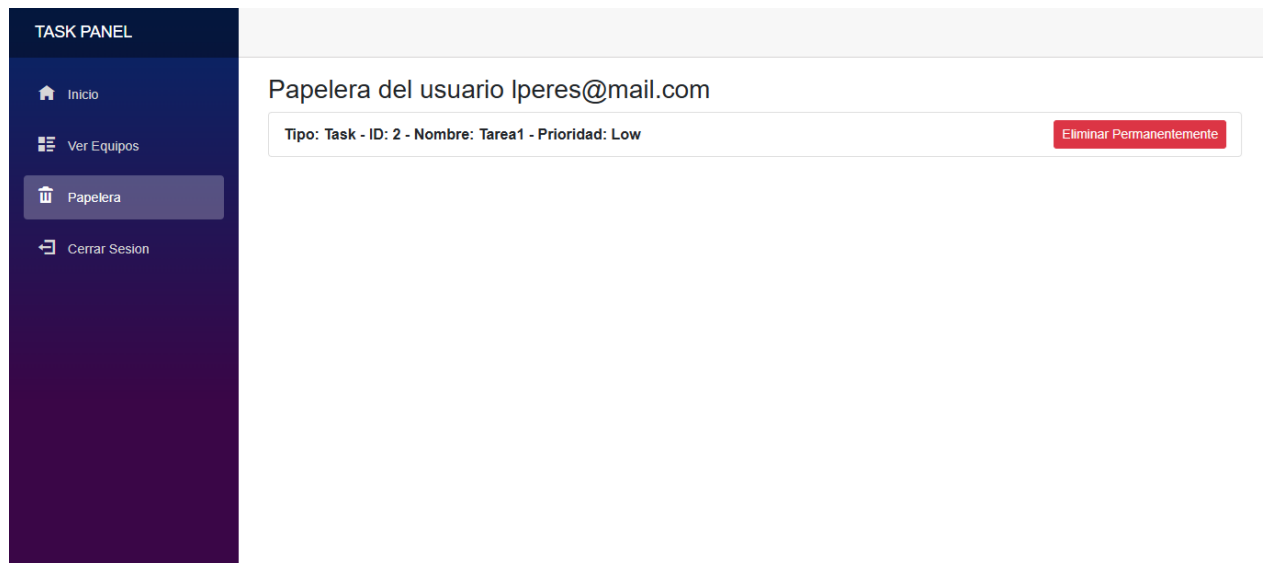
Pagar proveedores

Prioridad: Low

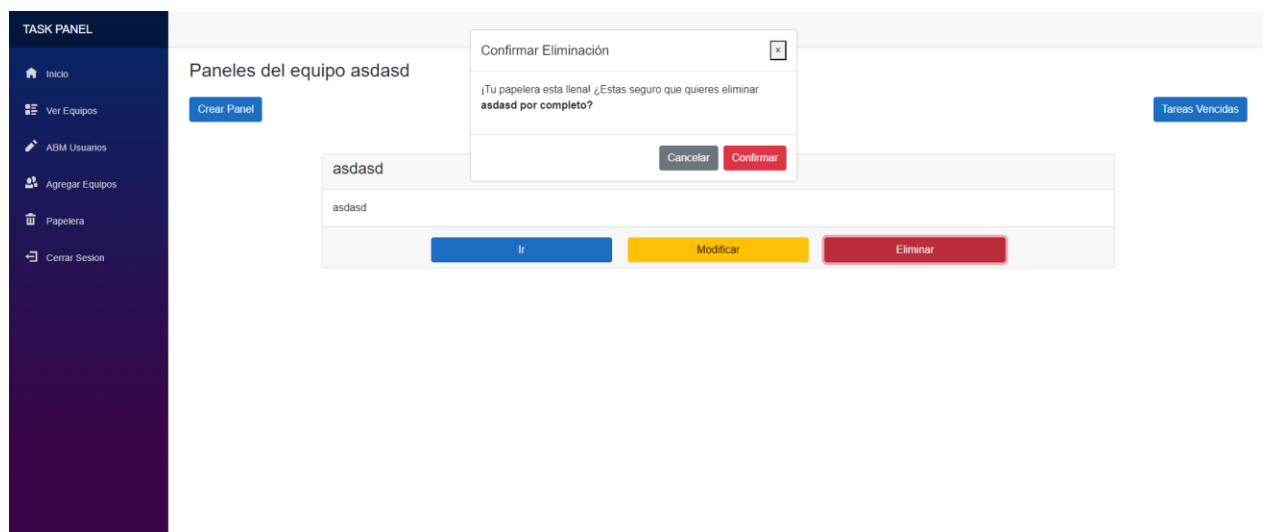
Vencimiento: 19/9/2024 12:00:00 AM

Descripción de la tarea:
Revisar planilla de proveedores y pagar.

Tareas vencidas



Papelera de usuario



Eliminar por completo