UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Módulo para Gestión de Vacaciones: Diseño de un Servicio Orientado a DDD

Autor/es

Héctor Solar Ruiz

Director/es

ROSARIO LÓPEZ GÓMEZ

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2021-22

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Módulo para Gestión de Vacaciones: Diseño de un Servicio Orientado a DDD

Realizado por:

Héctor Solar

Tutelado por:

Rosario López Gómez

Logroño, Julio 2022

Resumen

Resumen

Desarrollo a medida de un módulo que se integre en el programa de la empresa y facilite la gestión, solicitud y aprobación de las vacaciones.

Abstract

Abstract

Custom Development of a module that is integrated into the company's program and facilitates the management, request and approval of vacations.

“Agradecer a todas las personas que de una forma u otra me han ayudado a llegar hasta aquí.

En primer lugar, a mis tutores, Rosario López por su cercanía y dedicación, y Alberto Ortiz, por tutorizarme y echarme siempre una mano.

También querría mencionar a toda la gente de Hiberus que me han acompañado en estos meses y me han facilitado las cosas desde el momento en que llegué.

En segundo lugar, a mi familia, en especial a mis abuelos, mi madre Asun, mi padre Ángel, mi hermana Clara y mi pareja Andrea. A mis amigos y compañeros de carrera “Fumigans“, con quienes he compartido penas y alegrías durante estos años.

Terminar este ciclo ha sido posible gracias a todos ellos”.

Índice

Índice de Figuras

Índice de Tablas

1. Introducción

1.1. Marco del trabajo

Este trabajo fin de grado (TFG) tiene como objetivo la realización de un módulo de servicio que se acople a la aplicación de recursos humanos existente en la empresa, GesTEC, permitiendo de esta forma que se añadan en el futuro nuevas funcionalidades relacionadas con los recursos humanos, desde un punto de vista centralizado y único.

Principalmente el módulo estará orientado a la gestión de las vacaciones de la empresa, en las que el propio empleado pueda solicitar sus días de vacaciones y su gestión sea llevada por el personal de recursos humanos.

El sistema está pensado principalmente tanto para clientes y usuarios dentro de la empresa como personal de recursos humanos o empleados. Se utilizarán datos de la propia empresa y visualmente mantendrá el aspecto corporativo de la misma.

1.2. Motivación y Antecedentes

Con el crecimiento de la empresa, especialmente en aquellas que utilizan el teletrabajo, existen cada vez más factores que requieren una planificación del trabajo y de las horas libres.

Este problema se vuelve más importante si tenemos en cuenta que en muchos casos distintos equipos trabajan de una forma coordinada y deben cumplir unos plazos muy ajustados establecidos por los distintos clientes.

Este TFG tiene como objetivo poder centralizar la gestión de las vacaciones de los empleados de la empresa y facilitar una vista conjunta de los trabajadores y su disponibilidad en las distintas fechas.

1.3. Estructura de la aplicación GesTEC

La aplicación de GesTEC está orientada a la automatización de tareas de recursos humanos. Engloba gran variedad de tareas, desde la gestión interna, al seguimiento de proyectos por parte de clientes externos.

La aplicación sigue una arquitectura Modelo-Vista-Controlador, donde se separa en capas el acceso a datos y la lógica de negocio.

Las tecnologías en las que el proyecto está creado son:

BackEnd: SQL Server y Entity Framework, Docker Desktop, Kitematic (aplicación por la cual nos permite instalar Redmine configurado), para la conversión/mapeo entre clases.

FrontEnd: Bootstrap 4, Datatables para las tablas del front-end, Select2 para los seleccionables de opciones del front-end.

Testing: Moq v4.16.1, para simulación de objetos en las pruebas unitarias, Nunit v3.13.1, para las pruebas unitarias, Effort v2.2.13, para la simulación de base de datos de las pruebas unitarias.

Subsistemas de la aplicación GesTEC

La aplicación de GesTEC es compleja y presenta diversidad de subsistemas internos, aunque podemos resumirlos en 8 subsistemas principales (Figura1):

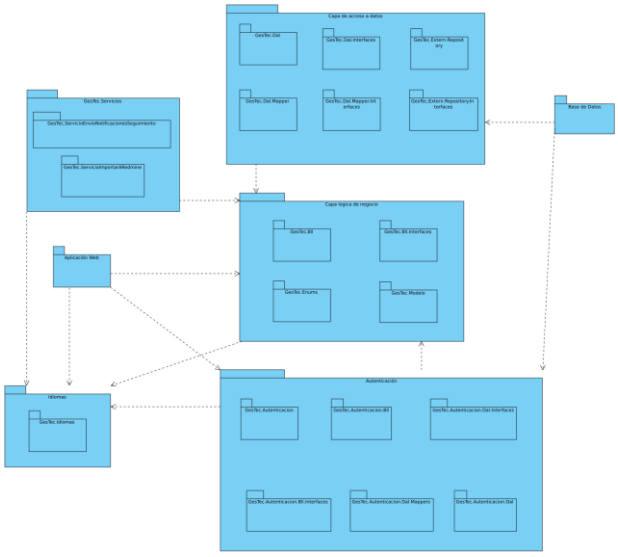


Figura 1: Diagrama de paquetes existente en la documentación de la Aplicación GesTEC

1. Subsistema de Usuarios Técnicos: Gestión de proyectos, datos de clientes y gestión de horas trabajadas.
2. Subsistema de Usuario Manager: Gestión unificada de documentación de seguimiento, progreso de los proyectos etc.
3. Subsistema de Proyectos: Creación, modificación etc, de los proyectos sobre los que se están trabajando.
4. Subsistema De Plantillas: Muestra las diferentes plantillas para los seguimientos de los técnicos.
5. Subsistema de Carga de Datos desde Redmine
6. Ficha Técnica de los trabajadores: Acceso a datos de los trabajadores de la empresa
7. Subsistema de Cliente: Acceso a proyectos activos
8. Usuario Administración: Permite cambiar aspectos técnicos de la aplicación.

1.4. Informe de viabilidad

Para establecer la viabilidad de este proyecto un tema clave es que no existan soluciones alternativas que ofrezcan los servicios solicitados y que resultaran rentables para la propia empresa. Es por ello que, tras estudiar el mercado se observa que, aunque existen diversidad de aplicaciones que prácticamente cumplen las funciones y requerimientos solicitados y que combinan las funcionalidades de recursos humanos,

horarios y tareas, este tipo de aplicaciones son de pago o siguen un programa de subscripción/permanencia, con lo que conllevan un coste que ha sido necesario contemplar.

Se han estudiado las funcionalidades de dos aplicaciones representativas: Kenjo y Factorial (Figura 2).

Figura 2: Aplicaciones de Recursos Humanos Kenjo y Factorial

Como funciones comunes de este tipo de aplicaciones se encuentran la posibilidad de revisar las horas de entrada y salida desde diversos dispositivos (móvil, ordenador, tablet etc), así como establecer la ubicación del trabajador y el proyecto en el que trabaja.

Entre sus usos destaca la posibilidad de adjuntar justificantes y otras funcionalidades como las siguientes (Figura3):

- Control y justificaciones de vacaciones y ausencias
- Cálculo total de horas trabajadas, ausencias, etc.
- Asignaciones de turnos y vacaciones.
- Establecer los empleados que trabajarán en los determinados turnos.
- Solicitud de bajas y peticiones de nuevas vacaciones.

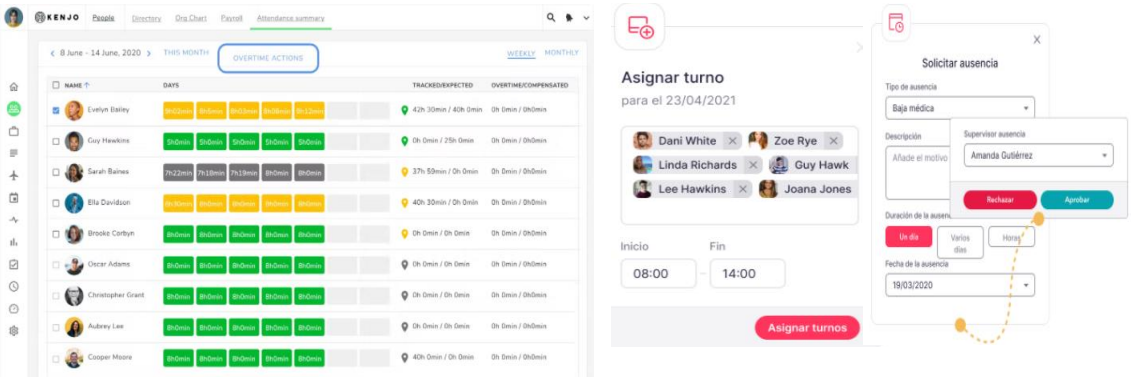


Figura 3: Control Horario

El estudio de las distintas opciones que proporcionan para una empresa de unos 50 empleados indica un precio entre 150-200 € mensuales, lo que anualmente ronda

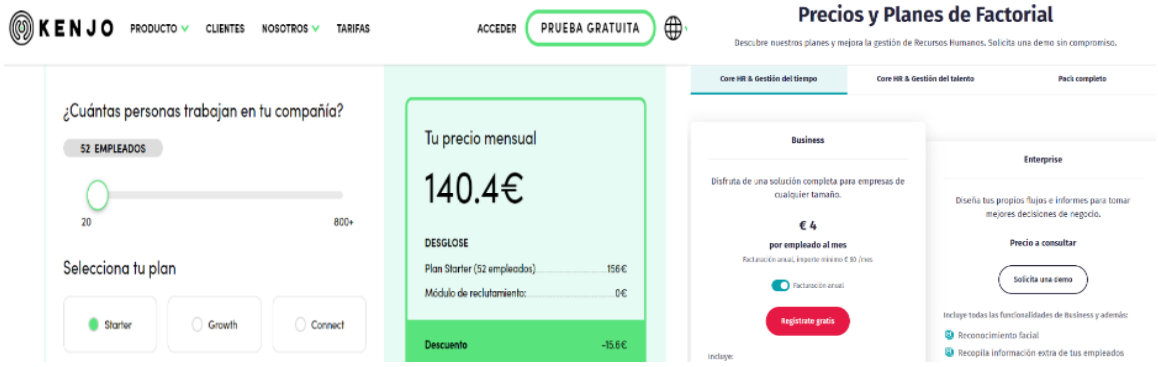


Figura 4: Precios

aproximadamente los 2.000-2.500 €, sin incluir complementos específicos para las necesidades del cliente, como los que se muestran en la Figura 4.

El estudio de las funcionalidades requeridas en este proyecto, para una empresa de unos 50 empleados, suponen un coste entre 150-200 € mensuales, lo que anualmente ronda aproximadamente los 2.000-2.500 €, sin incluir complementos específicos para las necesidades del cliente, como los que se muestran en la Figura 4.

Uno de los problemas encontrados en este tipo de software específico es la falta de conexión directa a la API de SharePoint, que es precisamente la plataforma que utiliza la empresa para almacenar las vacaciones.

Por otra parte, el precio estimado y que muchas de las funcionalidades no se adaptarían exactamente a los requerimientos, así como otras funcionalidades exceden las necesidades permite considerar que un aplicativo propio sería la mejor opción.

Tampoco un software contratado podría ser extensible a nuevas necesidades que puedan ir surgiendo y de cierta forma “atan” a seguir utilizando los servicios contratados, ya que cambiar o migrar los datos es un proceso bastante costoso.

Respecto a la protección de datos, uno de los principales requisitos al realizar la aplicación es que los datos no salgan de la propia empresa, ya que se trata en muchos casos de datos sensibles de los empleados. Aunque en el software analizado sí que garantizan la confidencialidad de los datos, la realización de la aplicación directamente para la propia empresa la dota de mayor confianza y seguridad.

Por todas estas razones, y tras presentar este análisis a la empresa, se decide la realización de una herramienta propia modular que pueda integrarse y complementarse al software utilizado actualmente en la empresa.

2. Planificación

2.1. Objetivos y Alcance del proyecto

Actualmente, la manera de gestionar las vacaciones por la empresa es enviar un correo con un Excel al responsable del equipo y marcar las posibles vacaciones en la plataforma de SharePoint.

El responsable por su parte marca en su ordenador normalmente con otro Excel si puede aprobar o denegar las solicitudes y actualiza, confirmando o no, la aceptación de las distintas solicitudes en SharePoint.

Por tanto, el objetivo principal con esta aplicación es automatizar este proceso, desarrollando e integrando un módulo en el sistema de la empresa, que permita la gestión de las vacaciones y una conexión directa con la API de SharePoint.

2.2. Requisitos del proyecto

Los requisitos del software son las funcionalidades del sistema a desarrollar. Distinguiremos dos tipos de requisitos: funcionales y no funcionales.

2.3. Requisitos funcionales

Un requisito funcional define una función del software o sus componentes descrita como un conjunto de entradas, comportamientos y o salidas:

- **RF-1**: Conectar con la aplicación de GesTEC.
- **RF-2**: Crear prototipos de interfaces para que sean evaluados por el cliente.
- **RF-3**: Se utilizará como modelo de clases los creados en el Proyecto GesTEC.
- **RF-4**: Las solicitudes se persistirán en la base de datos.
- **RF-5**: Comprender y ampliar el diseño de base de datos existentes.
- **RF-6**: Conectar la base de datos con Blazor y procesar datos.
- **RF-7**: Las vacaciones se persistirán en la base de datos.
- **RF-8**: Se guardará el estado de las vacaciones (aprobada, cancelada etc.).
- **RF-9**: Conectar con SharePoint.
- **RF-10**: Leer datos de SharePoint y procesarlos.
- **RF-11**: Diseñar una Api para lectura y procesado de datos.
- **RF-12**: Poder cambiar la configuración del proyecto para poder hacer cambios en las vistas del proyecto.
- **RF-13**: Se podrán importar distintos calendarios de vacaciones.
- **RF-14**: Se establecerán mecanismos para aprobar/cancelar solicitudes.
- **RF-15**: Se podrán enviar solicitudes que serán aceptadas o no.
- **RF-16**: Habrá un sistema de login para poder acceder.
- **RF-17**: Existirán distintos tipos de usuarios con distintos privilegios.
- **RF-18**: Existirá el usuario administrador global que podrá administrar la aplicación.
- **RF-19**: Mostrar los datos de SharePoint.

- **RF-20**: Páginas para importar y mostrar datos de SharePoint.
- **RF-21**: El responsable del equipo podrá ver las vacaciones de su equipo.
- **RF-22**: Las vistas serán responsive.
- **RF-23**: Los resultados de las vacaciones aprobadas del equipo serán generadas en una página centralizada.
- **RF-24**: Para usar la aplicación se deberá estar logueado.
- **RF-25**: Los usuarios registrados podrán loguearse.
- **RF-26**: Se desarrollará un sistema para cambiar los datos del usuario y poder eliminar su cuenta.
- **RF-26**: Las interfaces y procesos estarán documentados con los distintos diagramas pertinentes.

Durante el desarrollo de esta memoria se especificará la sección en la que se ha cubierto cada uno de estos requisitos.

2.4. Requisitos no funcionales

- RNF-1 Robustez: las operaciones de la aplicación no tardarán más de un segundo a excepción de aquellas que requieran conexión o extracción contra la BD.
- RNF-2 Usabilidad: las interfaces serán fáciles de utilizar e intuitivas.
- RNF-3 Privacidad: los usuarios únicamente podrán ver la información que tengan disponible según sus permisos.
- RNF-4 Seguridad: los datos sensibles de la empresa no se harán públicos a los usuarios de la aplicación.
- RNF-5 Disponibilidad: solamente los trabajadores de la empresa podrán acceder a la aplicación.
- RNF-6 Extensibilidad: el sistema será compatible con los anteriores y eventualmente podrá ser ampliado en futuros proyectos.
- RNF-7 Reutilización de componentes: se podrán reutilizar los componentes y código generado para otras tareas que requieran similares servicios.
- RNF-8 Escalabilidad: el sistema podrá crecer si es necesario añadir nuevas operaciones.
- RNF-9 Integrabilidad: se integrará el módulo con el proyecto ya existente.
- RNF-10 Documentación: las distintas secciones estarán bien documentadas.
- RNF-11 Mantenibilidad: se desarrollará el código de una forma que su lectura sea clara y se pueda mantener de una forma sencilla.
- RNF-12 Desacoplamiento y modularidad: el desarrollo estará construido bajouna arquitectura DDD desacoplando sus distintas partes.

2.5. Tecnologías

En este apartado se comentan brevemente las tecnologías por las que se ha optado en este proyecto, así como su justificación. En primer lugar, en los desarrollos de la empresa se opta fundamentalmente por .NET. En la carrera solamente se ha trabajado con esta tecnología en la asignatura de tercer curso Tecnología Orientada a Objetos

(TOO) en la que se dan algunas nociones básicas utilizando el marco de trabajo .NET framework, en concreto la tecnología de Windows Forms (2001) para el desarrollo de aplicaciones de escritorio en Windows.

El interés en este tipo de tecnología ha llevado al estudiante a formarse más allá de lo aprendido en el Grado utilizando otras fuentes y recursos adicionales [1]. Los pequeños desarrollos previos, las pruebas realizadas en su aprendizaje, así como el código de este proyecto pueden consultarse en el repositorio de GitHub de forma pública[1]. Están realizados en la versión del framework de .NET Core 5.

Para el desarrollo web de este proyecto se han utilizado las distintas tecnologías de .NET que se enumeran a continuación.

2.5.1. Marco de Trabajo: .NET Core

.NET Framework, .NET Core, .NET Standard

.NET es una capa de abstracción o entorno de trabajo sobre la que se compilan y ejecutan los lenguajes propios de .NET C#, Visual Basic o F# (Figura 5).

Estos lenguajes son convertidos en tiempo de ejecución en un lenguaje común (CLI) que la plataforma puede interpretar.

Una vez definidos estos conceptos aparece en 2002 el entorno de programación de .NET Framework, principalmente desarrollado para aplicaciones de escritorio en Windows, con el tiempo y evolución de la web surgen nuevas necesidades para el desarrollo de aplicaciones más orientadas a la web.

Aparece Azure la plataforma de Nube de Microsoft en 2010, y comienzan a aparecer nuevas alternativas a WPF y Windows Forms, como son AspNet con MVC.

Por todo esto Microsoft decide sacar en 2016. NetCore que es un cambio en la capa de abstracción de los lenguajes de .NET para mejorar la velocidad y orientar el desarrollo hacia la web, aunque se sigue manteniendo los proyectos en .NET framework. Básicamente las nuevas aplicaciones no utilizan librerías de Windows, sino que las incorporan de forma externa para eliminar esta dependencia.

A partir de .NET Core 5 finalmente el entorno se convierte en un entorno multiplataforma y se comienza a denominar .NET a secas.

Este cambio de paradigma de desarrollo de aplicaciones supone que las diferentes plataformas de .NET Videojuegos: Unity, Android: Xamarin, IOS: Mono, Web: Net Core, se combinen en una única capa de abstracción o bridge con lo cual aparecen finalmente aplicaciones nativas en distintos sistemas operativos.

Por tanto, al igual que otros lenguajes finalmente se pueden realizar diseños multiplataformas nativos (Figura5).

[1] [1] https://github.com/hesolar?tab=repositories

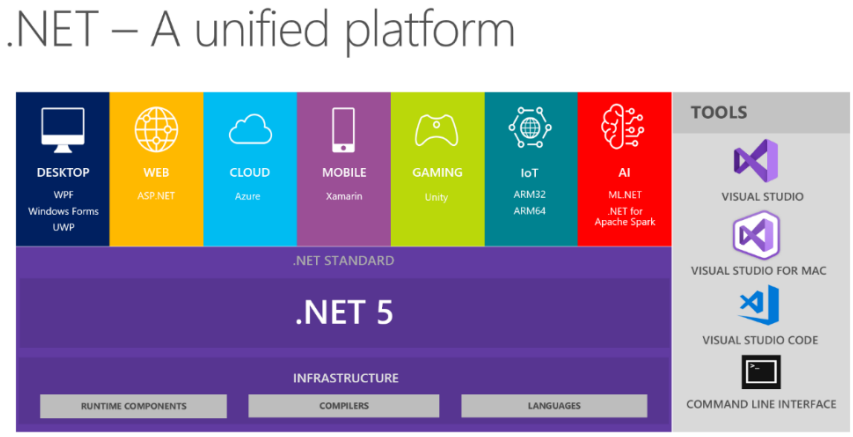


Figura 5: .NET

Para todo el proyecto, se utilizará la nueva versión del Framework .NET 6.0 que fue publicada en diciembre de 2021 de forma preliminar y en enero publicada para la producción, esta versión es LTS (Long Term Support)[2] de esta manera se garantiza que esta función siga teniendo soporte durante gran periodo de tiempo.

Para el desarrollo web de este proyecto he utilizado las distintas tecnologías de .NET que a continuación enumero:

2.5.1. Entorno de desarrollo: Visual Studio

Integrado para Windows y macOS. Es compatible con lenguajes de programación de .NET, tales como F#, C# o Visual Basic (Figura 6).

2.5.2. Entorno de programación: AspNet

ASP.NET (Figura 6) es un entorno de aplicaciones web creado por Microsoft que permite desarrollar aplicaciones para la web sobre la plataforma de .NET.

Figura 6: Tecnologías: Visual Studio, Asp .NET, Blazor

2.5.3. Lenguajes: C#, HTML, CSS y JavaScript

Se utilizará el lenguaje de C# para la toda la lógica de la aplicación (Front-Back). Para la vista se utilizarán las páginas Razor, que son ficheros con extensión *Razor* y combinan C# dentro del marcado de HTML.

[2] Long Term Support: Versiones o ediciones especiales de software diseñadas para tener soportes durante un período más largo que el normal.

Para describir la presentación web de HTML se utilizará Cascading Style Sheets (CSS). La iconografía que identifica estas tres tecnologías se muestra en la Figura 7.

Figura 7 Lenguajes Utilizados:

2.5.4. Tipo de Proyecto: Blazor y MAUI

Como tipo de proyecto se utilizará Blazor (2018). Blazor [2] es un proyecto desarrollado por Microsoft creado para realizar SPAs[3], está pensado para realizar todas las partes de la aplicación web únicamente usando como lenguaje de programación C#, Html y CSS.

Funciona con un sistema de componentes, las cuales tienen un estado y son reutilizables en las distintas partes del proyecto, además pueden ser consumidas por otros tipos de proyectos incluso fuera del entorno .NET, con frameworks nativos en JavaScript como Angular o React.

Blazor funciona como un patrón observer[4], es decir, se muestra al usuario directamente cual es el estado de los objetos de su aplicación, el Html es una representación directa del valor de sus objetos.

El patrón observer es un patrón de tipo comportamiento, define una relación de un objeto con su entorno. Se trata de un patrón de diseño de software que define una dependencia entre objetos del tipo uno a muchos. En este caso, cuando uno de los objetos cambia su estado, notifica este cambio al resto de objetos dependientes (a todo su entorno). Este patrón está relacionado con la asignación de responsabilidades a objetos y el patrón MVVM.

Además, si se desea hacer alguna comprobación o utilizar otros patrones de programación web más convencionales como MVC, podemos añadir controladores y vistas etc en cualquier momento.

Blazor es una alternativa a los nuevos framework de JavaScript spa (React Vue o Angular) y está especialmente pensado para desarrolladores de .NET.

Tipos de proyecto Blazor

Existen dos alternativas en Blazor [2] (Figura 8): Blazor Server Side y Blazor WebAssembly.

Blazor Web Assembly, descarga el código C# en el propio navegador del usuario y se ejecuta del lado de cliente transformando este código en el lenguaje web Assembly,

[3] SPA: Single Page Application es un tipo de aplicación web que ejecuta todo su contenido en una sola página.

mientras que en Server Side se ejecuta el código del lado de servidor y el usuario interactúa con la aplicación utilizando una conexión remota mediante SignalR.

Server Side requiere siempre conexión directa con el servidor, donde se procesan todas las peticiones; Blazor WASM puede ser usado en modo offline como una aplicación de web progresiva (WPA) permitiendo gestionar gran número de usuarios ya que el código se ejecuta en el lado de cliente.

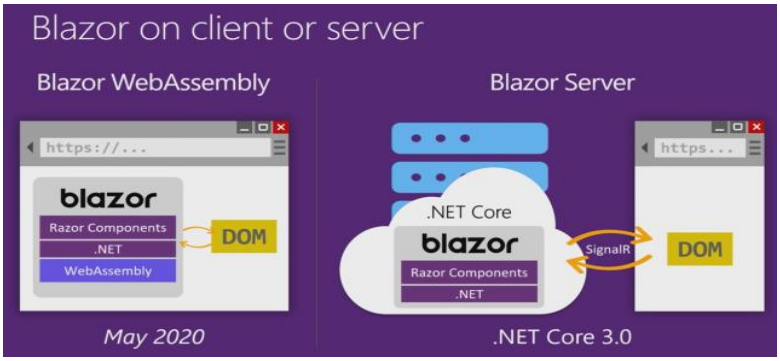


Figura 8: Blazor Server vs Blazor WASM

Blazor WASM debe su nombre al lenguaje de ensamblado Web Assembly (WASM). Está diseñado como un entorno de compilación dinámico para lenguajes de programación permitiendo el despliegue de aplicaciones web cliente/servidor y forma parte de la Open Web platform W3C.

Este lenguaje está soportado por los principales navegadores (Chrome, Edge, Firefox y WebKit) desde 2017 (Figura 9), es Full Cross platform y Nativo en el browser, sin plugins ni extensiones. Cualquier lenguaje se puede compilar a WASM: Javascript, C, C#, Java, etc.

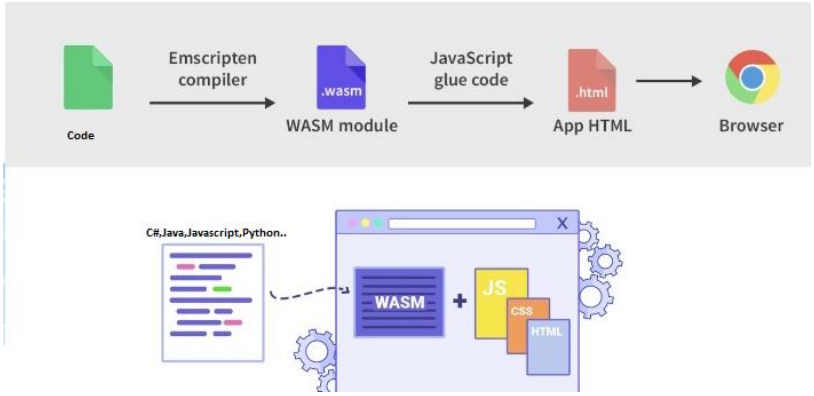


Figura 9: Distintos lenguajes compilados a WASM en el navegador

Los usos de WASM[5] están relacionadas con aquellas aplicaciones que necesiten muchos recursos delegándolos en el cliente y ejecutando código de forma nativa:

Varios casos de uso con WebAssembly muy reconocidos (Figura 10) son **Google Earth**, la biblioteca Javascript de **TensorFlow.js** (para entrenar e implementar modelos de aprendizaje automático) y la librería **OpenCV** de visión por computador creada por Intel.

[5] Usos actuales de WASM descritos en madewithwebassembly.com

WebAssembly ofrece en ellos un rendimiento mucho mejor que ejecutar sobre JavaScript en el navegador. Todas las tareas intensivas de CPU o de procesamiento pesado, como el aprendizaje automático, pueden beneficiarse enormemente de WebAssembly, consiguiendo una aceleración 10 veces mayor en sus predicciones al ejecutar un backend de WebAssembly en comparación con la implementación de JavaScript anterior.

Figura 10: Usos de WASM

No obstante, ambos proyectos utilizan una sintaxis parecida. Para este proyecto, a pesar de las ventajas de WASM, se utilizará Server Side, ya que se requiere de una interacción con el servidor y, por otro lado, no habrá gran cantidad de usuarios ni de peticiones.

Blazor utiliza el patrón MVVM (Model View ViewModel) que se explicará a continuación.

Patrón MVVM(Model-View-ViewModel)

El patrón modelo–vista–modelo de vista es un patrón de arquitectura de software. Se caracteriza por tratar de desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación.

Un estado de aplicación es el estado de un conjunto de variables y sus valores en un momento específico en un contexto específico.

Cuando la aplicación crece se deben compartir variables con diferentes componentes, así como preservar su valor incluso cuando cambie la página mostrada. Ante este problema aparece el patrón MVVM (Figura 11).

Hoy en día se utiliza principalmente para el desarrollo de aplicaciones móviles con Swift, Xamarin o Kotlin.

Modelo (M): Modelo de dominio de la aplicación, son los objetos que representan los datos de la aplicación. En Blazor utilizamos clases .cs

Vista (V): La vista coincide con la interfaz de usuario y su código subyacente. En Blazor, son los componentes de Razor (mezcla de HTML, CSS, C#).

ViewModel(VM): El modelo-vista proporciona la lógica necesaria para actualizar la vista, los datos de modelo y gestionar los eventos y cambios de estado en la aplicación.

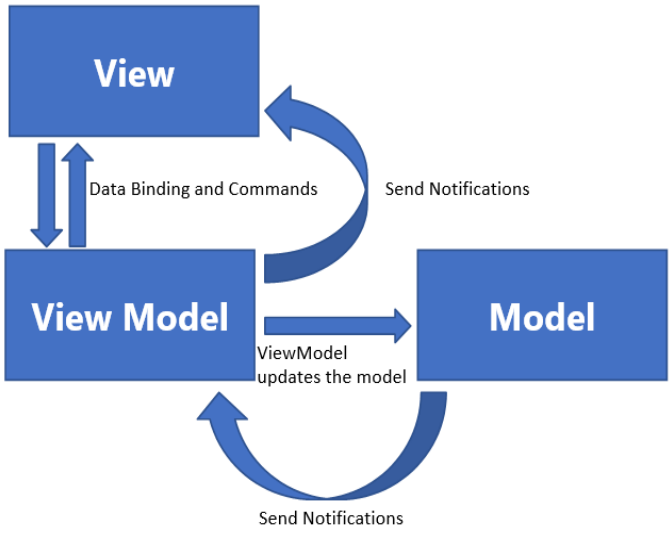


Figura 11: Patrón MVVM

El VM permite que el usuario no tenga que implementar la sincronización entre ambos, el modelo y la vista se actualizarán automáticamente si se produce un cambio en el otro.

Los componentes describen cualquier estado posible de la aplicación, este comportamiento irá cambiando durante el renderizado y el ciclo de vida del mismo (Figura 12).

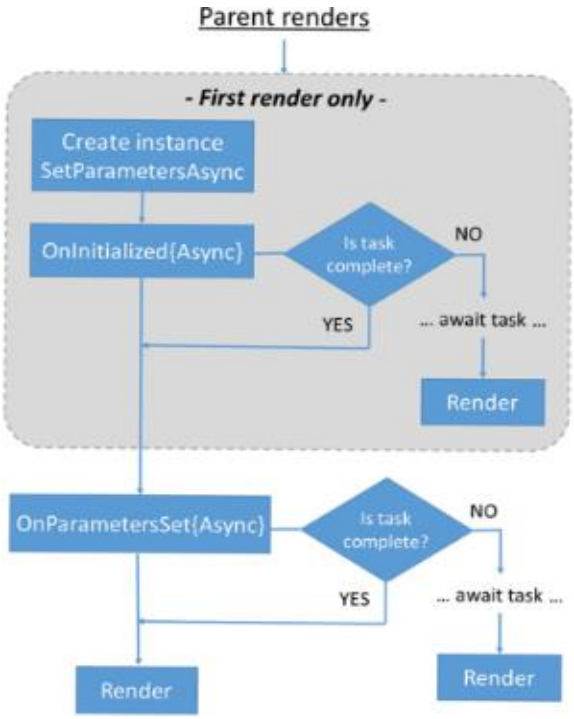


Figura 12: Ciclo de Vida de componentes Blazor

.NET MAUI

MAUI: “la interfaz de usuario de la aplicación multiplataforma de .NET (.NET MAUI) es un marco multiplataforma para crear aplicaciones nativas móviles y de escritorio con C# XAML o componentes de Blazor. Es de código abierto y es la evolución del anterior framework de desarrollo móvil de .NET Xamarin.Forms.” (Fuente: Microsoft docs).

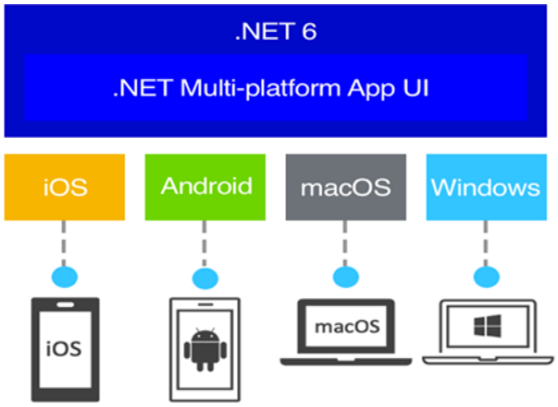


Figura 13: .Net MAUI

Con .NET MAUI (Figura 13), se pueden desarrollar aplicaciones nativas que se pueden ejecutar en Android, iOS, macOS y Windows desde un único código base compartido, utilizando componentes de XAML (lenguaje de marcado), o en el caso de este proyecto, utilizando componentes Blazor.

Como parte final de este proyecto se utilizará este tipo de aplicación.

2.5.5. Librerías de C# y Blazor

Uno de los objetivos de este trabajo era desarrollar el proyecto con lenguajes nativos de .NET sin necesidad de utilizar Javascript únicamente con C#, para ello se han utilizado las componentes de las librerías de Radzen, Blazorise y MudBlazor (Figura 14) las cuales permiten utilizar componentes nativos en Blazor, estos componentes consumen código y datos en C# no hace falta tener que hacer llamadas con Js sino que toda la funcionalidad sucede dentro de la lógica de la aplicación sin llamadas con callbacks estilo JQuery o Ajax.

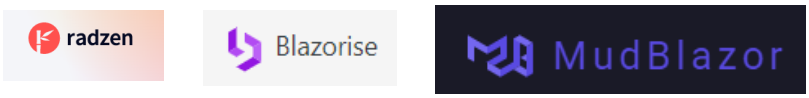


Figura 14: Librerías de componentes nativos de Blazor utilizadas

El uso de estas librerías facilita y agiliza enormemente el desarrollo ya que de esta forma apenas se tiene que programar la lógica y gestión de errores de las distintas componentes.

En cuanto a librerías de C# se han utilizado Automapper que permite copiar el valor de los objetos que tengan propiedades con el mismo nombre y MediatR que permite establecer comunicaciones y relaciones entre objetos (Figura 15).

Figura 15: Librerías Automapper y MediatR

2.5.6. Swagger Open API

Swagger (Open API) es una especificación independiente del lenguaje que sirve para describir API REST [1]. Permite utilizar una API REST sin acceso directo al código fuente Figura 16).

Sus principales objetivos son los siguientes:

- Minimizar la cantidad de trabajo necesaria para conectar los servicios.
- Reducir la cantidad de tiempo necesario para documentar un servicio.

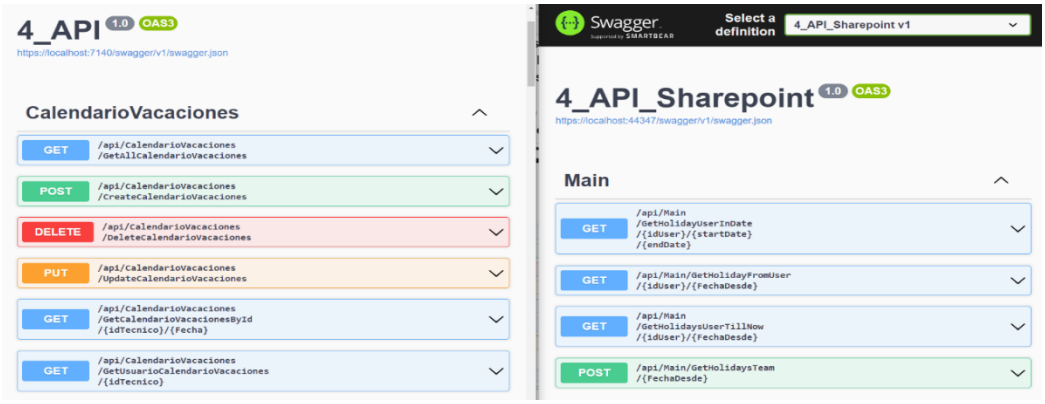


Figura 16: Uso de Swagger-Open API en el proyecto

Permite además hacer peticiones para ver los resultados y parámetros necesarios. En la figura siguiente se puede ver un ejemplo de una petición a la API para obtener las vacaciones (Figura 17):

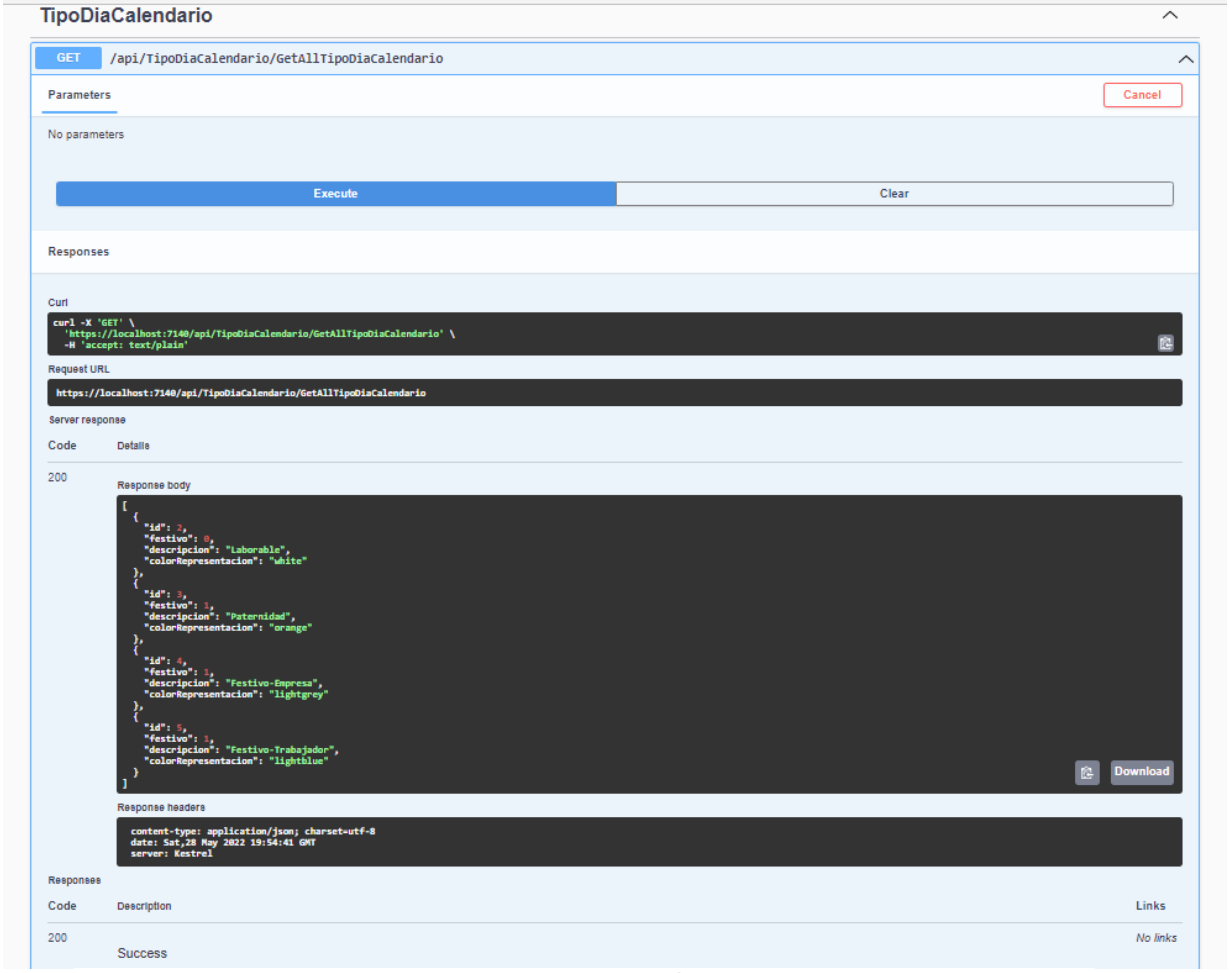


Figura 17: Ejemplo petición API

2.5.7. Base de datos SQL Server con MMSS y Entity Framework

Se utilizará una base de datos relacional en SQL con el motor de base de datos de Microsoft Management Studio.

Entity Framework Core [2] es un asignador de base de datos de objeto (ORM) para .NET (Figura 18). Admite consultas LINQ, seguimiento de cambios, actualizaciones y migraciones de esquemas.

Figura 18: Tecnologías de Base de datos

2.6. Metodologías Ágiles

Trabajar con metodologías en el proceso de desarrollo de software permite detectar y superar las dificultades que vayan apareciendo, organizar tareas y agilizar procesos con el objetivo de mejorar todos los procesos relacionados con el desarrollo.

En este proyecto se utilizarán las metodologías de SCRUM y Kanban durante todo el proyecto.

2.6.1. SCRUM

Utilizar esta metodología tiene como objetivo asegurar el cumplimiento de requisitos del cliente y entregar un producto de calidad en el menor tiempo siguiendo una planificación concisa y estableciendo plazos y objetivos.

El módulo a desarrollar estará integrado con el trabajo del resto del equipo siguiendo una planificación y plazos fijos, por tanto, las distintas etapas de desarrollo se ejecutarán en ciclos de tiempo (Sprint o iteración).

Cada Sprint producirá un resultado completo que será entregado al cliente dentro de un esquema de mejora continua.

2.6.2. KanBan

Las tareas son representadas como tarjetas en un “tablero”, que a su vez representan una historia de usuario, es decir una funcionalidad de la aplicación que se debe realizar durante el sprint, cada una además tiene una prioridad determinada que marcará cuando se va a realizar.

En función del estado (sin hacer, en proceso, hechas, en espera) las tarjetas se distribuirán en columnas según su estado.

Se utilizará la herramienta de office planner donde se creará un tablero que se irá actualizando según las tareas hechas.

El objetivo es poder comprobar de una manera visual y rápida el estado del sprint. Mediante un tablero y distintas columnas podremos observarlo de una manera más rápida.

2.7. Estructura descomposición de paquetes EDT

Se muestra a continuación la estructura de descomposición de paquetes del proyecto (Figura 19):

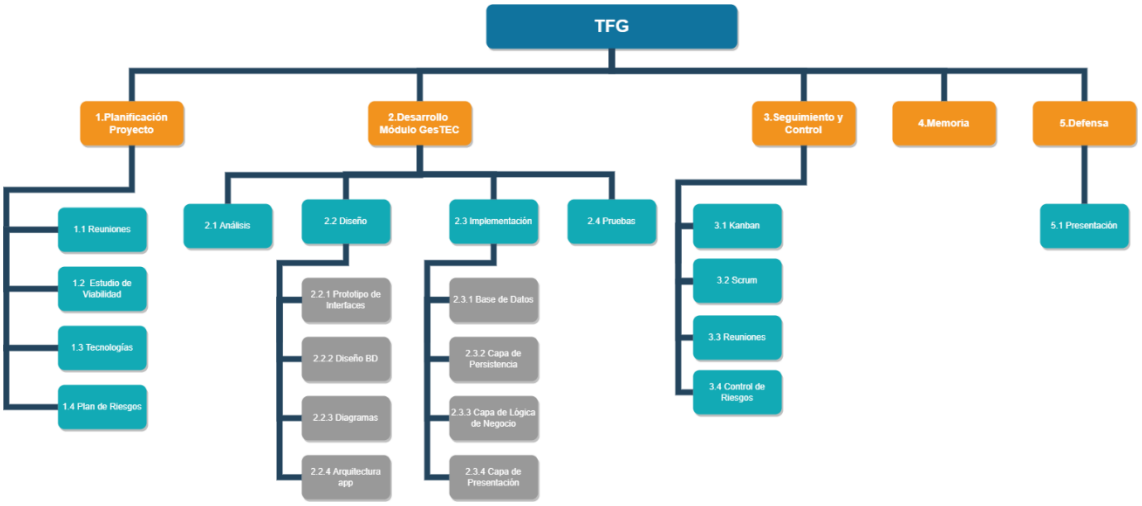


Figura 19: EDT

2.8. Estimación de dedicación

2.8.1. Cronograma: SCRUM aplicado al proyecto

El proyecto se realizará con fecha de inicio el 1 de marzo y terminará el día 1 de julio.

En función de las tareas a realizar se ha distribuido el calendario en Sprints(Figura20) de 50 h, es decir, de 10 días con 5 horas diarias (2 semanas).

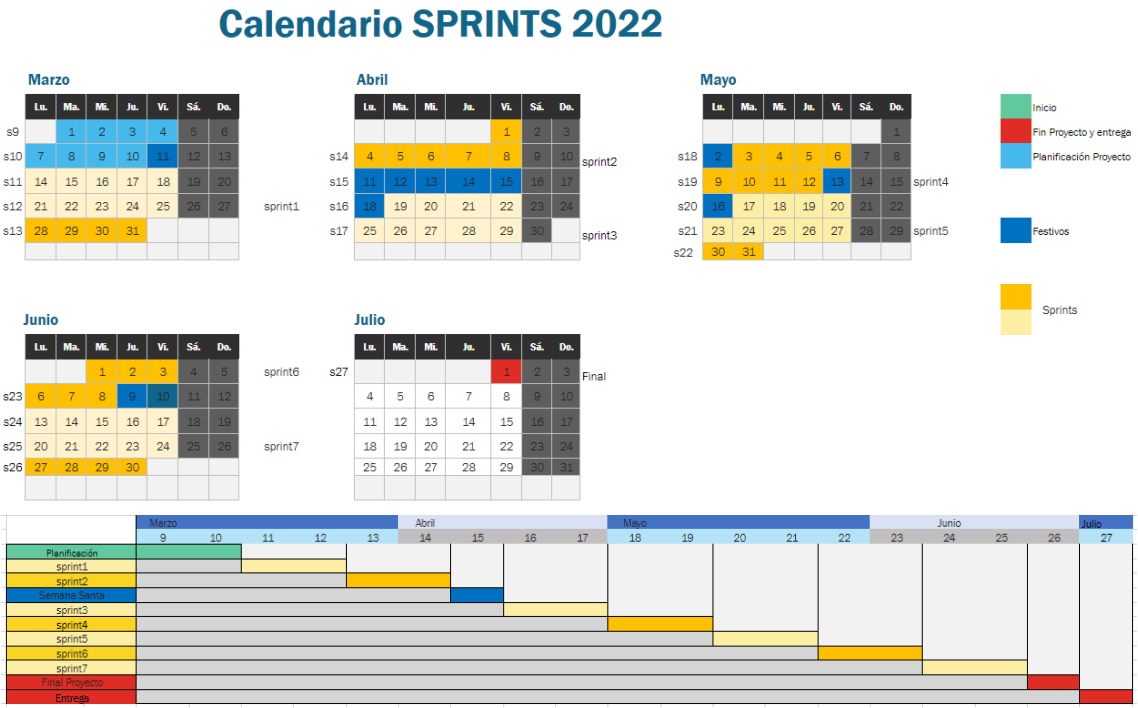


Figura 20: Calendario Sprints Proyecto

Descripción breve de los sprints:

- **Sprint0 Planificación:** Desarrollo del conjunto de documentación de la planificación del proyecto, estudio de viabilidad, análisis de riesgos y elección de tecnologías.
- **Sprint1 [Rfs 1-3]:** Análisis de la aplicación existente en la empresa, bocetos y diseños y realización de la documentación relacionada con los nuevos desarrollos.
- **Sprint2 [Rfs 4-8]:** Comienzo de la implementación del proyecto centrando el desarrollo en la capa de datos.
- **Sprint3 y Sprint4 [Rfs 9-18]:** Desarrollo e implementación de la lógica de la aplicación.
- **Sprint5 [Rfs 19-23]:** Desarrollo del diseño de las interfaces.
- **Sprint6 [Rfs 24-26]:** Evaluación y testeo de todo el proyecto.
- **Sprint7 [Rfs 27]:** Finalización de la memoria del proyecto.
- **Final Proyecto:** Entrega del proyecto, preparación de la defensa y desarrollo de la presentación.

2.8.2. Descripción y estimación de los paquetes de trabajo

En la Tabla1 se muestra la descomposición de paquetes de trabajo y el tiempo estimado para realizar cada uno.

Tabla 1: Diccionario EDT

ID Paquete trabajo	Descripción	FechaInicio	FechaFin	Sprints Realización	Estimación horas	Total Horas
1.Planificación Proyecto	Conjunto de tareas que corresponden a la planificación inicial del proyecto	01/03/2022	13/03/2022			10
1.1 Reuniones	Reuniones asociadas a la planificación inicial del proyecto			s0(planificación)	4	
1.2 Estudio de Viabilidad	Viabilidad del desarrollo del proyecto			s0(planificación)	1	
1.3 Tecnologías	Estudio y análisis de tecnologías que se van a utilizar en el proyecto			s0(planificación)	4	
1.4 Plan de Riesgos	Riesgos y contramedidas del proyecto			s0(planificación)	1	
2.Desarrollo Módulo GesTec	Tareas asociadas al desarrollo del módulo e integración en el proyecto de la	13/03/2022	01/07/2022			185
2.1 Análisis	Estudio previo de la aplicación utilizada por la empresa			s1	20	
2.2 Diseño	Diseño de las distintas partes del módulo	13/03/2022	31/03/2022	s1	23	
2.2.1 Prototipo de Interfaces	Bocetos y diseños completos de las interfaces que se mostrarán al usuario				5	
2.2.2 Diseño BD	Estudio de la BD existente y adición de nuevas tablas				3	
2.2.3 Diagramas	Conjunto de documentación de los diagramas utilizados en el proyecto				12	
2.2.4 Arquitectura de la App	Explicación de la arquitectura DDD seguida				3	
2.3 Implementación	Fase de programación de los aspectos de la aplicación	31/03/2022	31/06/2022	s2-s6		162
2.3.1 Base de Datos	Implementación de las nuevas tablas en la BD			s2	10	
2.3.2 Capa de Persistencia	Desarrollo de los mecanismos para la conexión con la BD			s2	25	
2.3.3 Capa Lógica de Negocio	Programación de la lógica de la aplicación			s3-s4	49	
2.3.4 Capa Presentación	Realización de las interfaces de usuario			s4-s5	50	
2.4 Pruebas	Testeo del correcto funcionamiento del proyecto			s2-s6	28	
3.Seguimiento y Control	Tareas asociadas al correcto desarrollo del proyecto	01/03/2022	01/07/2022	s0(planificación)-final		29
3.1 Kanban	Establecer elementos de Kanban como tableros y distribución de tareas			s0(planificación)	3	
3.2 Scrum	Desarrollo de Sprints, documentación etc.			s0(planificación)-fin	10	
3.3 Reuniones	Reuniones con el cliente y tutora durante todo el proyecto			s0(planificación)-fin	8	
3.4 Plan de Riesgos	Controlar los problemas y dificultades que vayan surgiendo.			s1-fin	8	
4.Memoria	Memoria del TFG	01/03/2022	01/07/2022	s7-fin		70
5.Defensa	Defensa del TFG y finalización del proyecto	27/07/2001	01/07/2022	fin		6
						300

2.8.3. Kanban en el proyecto

El tablero KanBan sobre el que se trabajará contará con las siguientes columnas:

- *Backlog*: Total de tareas del proyecto.
- *Todo*: Tareas a realizar en cada sprint.
- *Waiting*: Tareas “bloqueadas” que no se pueden realizar todavía.
- *Testing*: Tareas completadas que deben superar los test.
- *Done*: Tareas terminadas

En la Figura 21 se muestra un ejemplo de cómo se está trabajando en los datos de la aplicación durante el sprint1, columna *Todo,* en este caso tareas relativas al diseño y creación de BD y el login.

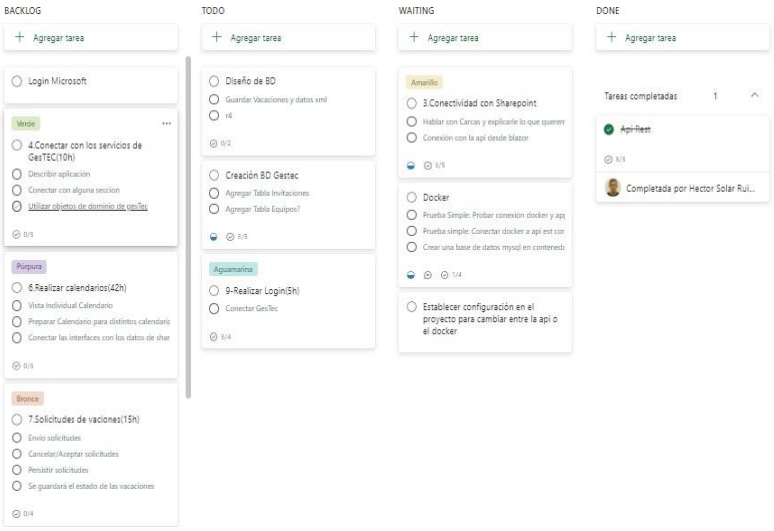


Figura 21: Tablero KanBan

2.8.4. Entregables

- E1.2 Estudio Viabilidad: estudio en el que se explicarán las razones por las que se realiza este proyecto.
- E1.3 Tecnologías para el proyecto: selección de tecnologías para utilizar en este proyecto.
- E1.4 Plan de riesgos: posibles riesgos que aparecerán en el proyecto y estrategias de mitigación.
- E2.1 Análisis: análisis de recursos existentes en la empresa e integración con los nuevos desarrollos.
- E2.2.1 Prototipo Interfaces: diseños para validar con el cliente.
- E2.2.2 Diseño de la BD: nuevos esquemas de la BD y modificaciones.
- E2.2.3 Diagramas: diagramas del proyecto para detallar su funcionamiento.
- E2.2.4 Arquitectura de la App: breve explicación de la arquitectura de software elegida.
- E3.1 Kanban: documentación asociada a la metodología KanBan.
- E3.2 SCRUM documentación asociada a la metodología Sprint.
- E3.3 Reuniones: Documento resumiendo aspectos de las reuniones.
- E4 Memoria: Memoria del TFG en formato pdf.

2.8.5. Hitos del proyecto

En la siguiente figura se muestra los hitos más importantes del proyecto asociados a cada Sprint:

Figura 22: Diagrama de hitos del proyecto

2.9. Plan de riesgos

Durante la realización del proyecto pueden surgir diversos contratiempos y problemas. Se detallan a continuación (Tabla2) los principales riesgos para que puedan ser claramente identificados y así, mitigar su impacto y, si es posible eliminarlos:

Tabla 2: Plan de Riesgos

Riesgo	Tipo de Riesgo	Medida de mitigación	Impacto	Tipo Acción	Riesgo Final
Enfermedad o incidencia personal de algún tipo	RRHH[6]	Priorizar requisitos y dejar para el final del proyecto aquellos que sean menos prioritarios.	Alto	Prevención	Medio
Formación lenta	Tecnológico/ Técnica	Utilizar tecnologías ya conocidas, reforzar los objetivos prioritarios frente a los secundarios.	Bajo	Prevención	Bajo
Pérdida de información	Pérdida de información de memoria o del código del proyecto	Utilizar tecnologías de respaldo y VCS[7]: Memoria: OneDrive Proyecto: GitHub	Alto	Mitigación	Muy bajo
Dificultad en la integración del módulo con el proyecto	Tecnológico/ Técnica	Utilizar arquitectura de software para reducir acoplamientos	Medio	Prevención	Medio
Cambio requisitos	Cliente	Reestablecer la hoja de ruta en cada sprint	Alto	Prevención	Medio
Falta de cumplimiento de plazos	Organizativo	Realizar al final de cada sprint una autocrítica para establecer mejoras en las siguientes fases	Medio	Prevención	Bajo

[6] RRHH: Recursos Humanos
[7] VCS: Sistemas de control de versiones

3. Análisis y diseño

3.1.Estructura Modular de desarrollo

Para este desarrollo se utilizará una arquitectura DDD (Domain Driven Design). Por su novedad, se muestra en qué consiste la arquitectura DDD, y se acompañará su explicación con Figuras del desarrollo realizado que sirvan de ejemplo.

Se presentan a continuación algunas definiciones previas:

Definiciones

1. **Patrones de diseño**: Los patrones de diseño son modelos de resolución que sirven como guía para la búsqueda de soluciones a problemas comunes en el desarrollo de software.
2. **Principios SOLID**: Los principios SOLID son técnicas para utilizar en cualquier aplicación de software independientemente del entorno de programación.

   En inglés, SOLID representa un acrónimo que agrupa cinco principios fundamentales:

   - *Single responsability*: principio de responsabilidad única.
   - *Open close*: principio de abierto y cerrado.
   - *Liskov substitution*: principio de sustitución de Liskov.
   - *Interface segregation*: principio de segregación de interfaces.
   - *Dependency inversion*: principio de inversión de dependencias.

- **Inversión de dependencias:** Los módulos de alto nivel no deberían depender de los de bajo nivel, ambos deberían depender de abstracciones.
- **Inserción/Inyección de dependencias (DI):** La inserción de dependencias es una forma de implementar el principio de inversión de dependencias. Es una técnica para lograr el acoplamiento flexible entre los objetos y sus dependencias. En lugar de crear directamente instancias de colaboradores o de usar referencias estáticas (es decir, usar new...), los objetos que una clase necesita para llevar a cabo sus acciones se proporcionan a la clase (o se "insertan" en ella). A menudo, las clases declaran sus dependencias a través de su constructor, lo que les permite seguir el principio de dependencias explícitas.

3.1.1. DDD (Clean Architecture)

DDD es un tipo de arquitectura de software para desarrollar aplicaciones más mantenibles y con división de responsabilidades. En ella se enfatiza el uso de interfaces que son implementadas por las capas más externas de la aplicación.

En la Figura 23 se representa la arquitectura tradicional en capas frente a la DDD, la comparación está mostrada en la Figura 24.

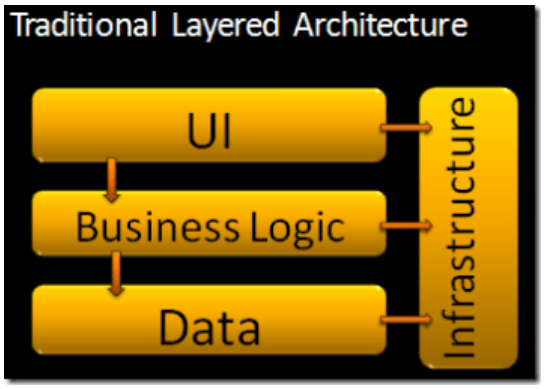


Figura 23: Arquitectura por capas tradicional

Arquitectura tradicional (Por capas)

Cada capa depende de las anteriores, y opcionalmente de alguna infraestructura común y servicios. El gran inconveniente de esta arquitectura en capas de arriba hacia abajo es el acoplamiento que crea.

Cada capa está acoplada a las capas inferiores y a la implementación de la infraestructura. Esta arquitectura crea un acoplamiento innecesario.

El mayor problema (y el más común) es el acoplamiento de la interfaz de usuario y la lógica al acceso a los datos. La interfaz de usuario no puede funcionar si la lógica de negocio no existe. A su vez, la lógica no puede funcionar si no hay acceso a los datos o si estos cambian.

DDD

Frente a este modelo de capas, el módulo construido en este proyecto se basa en la arquitectura DDD [1].

La regla primordial que hace que esta arquitectura funcione es la *regla de dependencia*. Esta regla indica que las dependencias del código fuente solo pueden apuntar hacia adentro del círculo. Nada en un círculo interior puede saber absolutamente nada sobre algo en un círculo exterior. En particular, el nombre de algo declarado en un círculo exterior no debe ser mencionado por el código en un círculo interior. Eso incluye, funciones, clases, variables, o cualquier otra entidad de software nombrada.

Esta arquitectura utiliza el principio de inversión de dependencias anteriormente mencionado, trabajando con las interfaces definidas en las capas internas. La dirección del acoplamiento es hacia el centro. Las capas son independientes y se pueden compilar, testear y ejecutar por separado. El núcleo de la aplicación necesita la implementación de las interfaces del núcleo, y esas clases de implementación residen en los bordes de la aplicación. Finalmente se necesita algún mecanismo para inyectar este código en tiempo de ejecución y resolver los tipos de las interfaces utilizándose para ello la técnica de inyección de dependencias (DI).

Como resultado, en esta arquitectura se puede cambiar la interfaz o la base de datos libremente, sin preocuparse de los acoplamientos que puedan surgir.

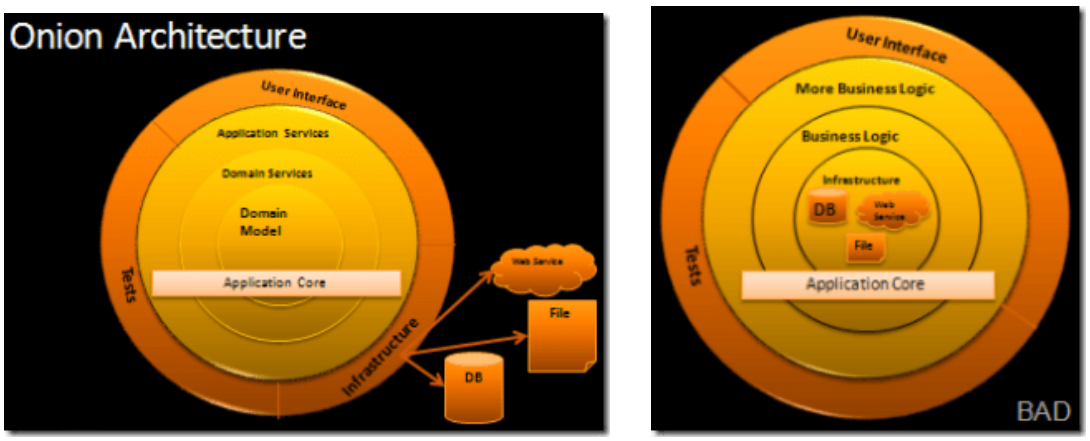


Figura 24: Comparativa arquitectura por capas y arquitectura DDD

En la Figura 25 se muestra una parte del código del proyecto que sirve de ejemplo de uso de esta arquitectura: en la capa de aplicación (Capa 3) hay una clase handler para recibir llamadas de la capa superior (API capa 4). Esta clase recibe un objeto request (petición) que se automapeará como un objeto de dominio (Capa1) mapeando aquellas propiedades con el mismo nombre. Después se comprobará si se puede añadir a la base de datos mediante un objeto repositorio inyectado por dependencias(_repository).

```
namespace Application.Handlers.CommandHandlers;
2 referencias | Hector Solar Ruiz, Hace 6 días | 1 autor, 1 cambio
public class CreateCalendarioVacacionesHandler : IRequestHandler<CreateCalendarioVacacionesCommand, bool> {
    private readonly ICalendarioVacacionesRepository _repository;
    0 referencias | Hector Solar Ruiz, Hace 5 días | 1 autor, 1 cambio
    public CreateCalendarioVacacionesHandler(ICalendarioVacacionesRepository calendarioVacacionesRepository) {
        _repository = calendarioVacacionesRepository;
    }
    0 referencias | Hector Solar Ruiz, Hace 6 días | 1 autor, 1 cambio
    public async Task<bool> Handle(CreateCalendarioVacacionesCommand request, CancellationToken cancellationToken) {
        Core.Entities.CalendarioVacaciones calendarioVacacionesEntity =
            MapperBase<CalendarioVacacionesMappingProfile,Core.Entities.CalendarioVacaciones>.MappEntity(request);
        if (calendarioVacacionesEntity is null) {
            throw new ApplicationException("Issue with mapper");
        }
        return await _repository.AddAsync(calendarioVacacionesEntity);
    }
}
```

Figura 25: Código ejemplo del manejador handler

Este objeto posee toda la funcionalidad del servicio de BD, e implementará una serie de operaciones CRUD de una interfaz superior (*IRepository*). Realmente no se conoce el subtipo de esta interfaz y se trabaja sobre la abstracción *ICalendarioVacacionesRepository*.

No se necesita saber qué tipo es para utilizarla. No es responsabilidad de la capa de aplicación conocer estos detalles, solamente se necesita conocer la signatura de sus operaciones que heredan de la interfaz principal IRepository.

Al margen de subtipos solo se trabaja con interfaces, aunque luego durante el tiempo de ejecución se le inyecta un objeto de tipo *CalendarioVacacionesRepository*, que implementará todas las funciones de la interfaz.

En tiempo de ejecución se resolverán estas interfaces en la capa más externa de aplicación, como puede apreciarse en la Figura 26, con la declaración de las interfaces y en la Figura 27 con su uso y en las siguientes la implementación de estas:

```
namespace Core.Repositories;

16 referencias | Hector Solar Ruiz, Hace 49 minutos | 1 autor, 2 cambios
public interface ICalendarioVacacionesRepository : IRepository<CalendarioVacaciones, Tuple<int, DateTime>> {
    2 referencias | Hector Solar Ruiz, Hace 49 minutos | 1 autor, 1 cambio
    public Task<IReadOnlyList<CalendarioVacaciones>> GetDiaUsuario(int idUsuario);
    2 referencias | Hector Solar Ruiz, Hace 49 minutos | 1 autor, 1 cambio
    public Task<IEnumerable<CalendarioVacaciones>> AddNonSavedItems(IEnumerable<CalendarioVacaciones> listado);
}
```

DDD_TFG - IRepository.cs

```
namespace Core.Repositories.Base;

6 referencias | Hector Solar Ruiz, Hace 5 días | 1 autor, 4 cambios
public interface IRepository<T,TKey> where T : class {
    6 referencias | Hector Solar Ruiz, Hace 10 días | 1 autor, 1 cambio
    Task<IReadOnlyList<T>> GetAllAsync();
    6 referencias | Hector Solar Ruiz, Hace 10 días | 1 autor, 1 cambio
    Task<T> GetByIdAsync(TKey id);
    6 referencias | Hector Solar Ruiz, Hace 5 días | 1 autor, 2 cambios
    Task<bool> AddAsync(T entity);
    6 referencias | Hector Solar Ruiz, Hace 7 días | 1 autor, 2 cambios
    Task<bool> UpdateAsync(T entity);
    6 referencias | Hector Solar Ruiz, Hace 5 días | 1 autor, 1 cambio
    Task<bool> DeleteAsync(TKey entity);
}
```

Figura 26: Declaración de las interfaces en las capas internas

```
namespace Infrastructure.Repositories;
2 referencias | Hector Solar Ruiz, Hace 54 minutos | 1 autor, 5 cambios
public class CalendarioVacacionesRepository : ICalendarioVacacionesRepository {

    CalendarioVacacionesContext _context;
    RepositoryBase<Core.Entities.CalendarioVacaciones, Tuple<int, DateTime>, CalendarioVacacionesContext> baseOperations;
    0 referencias | Hector Solar Ruiz, Hace 6 días | 1 autor, 1 cambio
    public CalendarioVacacionesRepository(CalendarioVacacionesContext context) {
        _context = context;
        baseOperations = new(_context);

    }

    7 referencias | Hector Solar Ruiz, Hace 5 días | 1 autor, 2 cambios
    public async Task<bool> AddAsync(CalendarioVacaciones entity) {
        await _context.Set<CalendarioVacaciones>().AddAsync(entity);
        return await _context.SaveChangesAsync() > 0;
    }


    7 referencias | Hector Solar Ruiz, Hace 54 minutos | 1 autor, 2 cambios
```

Figura 27: Uso de estas interfaces

```
using Application.Handlers.CommandHandlers;
using Core.Repositories;
using Infrastructure.Data;
using Infrastructure.Repositories;
using Microsoft.EntityFrameworkCore;
using System.Reflection;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var connectionString= builder.Configuration.GetConnectionString("TFG_DB");

//DB Context
builder.Services.AddDbContext<CalendarioVacacionesContext>( m => m.UseSqlServer(connectionString));

//Repositories
builder.Services.AddTransient<ICalendarioVacacionesRepository, CalendarioVacacionesRepository>();
```

Figura 28: Inyección dependencias ICalendarioVacacionesRepository y resolución de tipos

Ventajas:

- *Independiente de frameworks*. La arquitectura no depende de problemas o limitaciones de otras capas.
- *Testeable y modular*. Las reglas de negocio se pueden probar sin la interfaz de usuario, la base de datos, el servidor web o cualquier otro elemento externo.
- *Independiente de la interfaz de usuario*. La interfaz de usuario puede cambiar fácilmente, sin cambiar el resto del sistema. Una interfaz de usuario web podría reemplazarse con una interfaz de usuario de consola, por ejemplo, sin cambiar las reglas de negocio.
- *Independiente de la base de datos*. Las reglas de negocio no están vinculadas a la base de datos.
- *Independiente de factores externos*. De hecho, las reglas de negocio simplemente no saben nada sobre “el mundo exterior”.

Desventajas:

- Como cada capa únicamente conoce interfaces y capas anteriores, en ocasiones se tiene que replicar código en cada capa.
- Generación de muchas clases: Al ser estrictos en el principio de SRP nos encontraremos clases con pocas líneas de Código y muy mantenibles, pero también muy numerosas.
- Dificultad de comprensión y tiempo de desarrollo: Utilizar esta arquitectura requiere una formación previa y añade en muchos casos una gran complejidad al proyecto.

Información actualizada puede encontrarse en la propia documentación de Microsoft [2] Ver para ampliación [3] [1].

3.2. Diagrama de paquetes

✓**RF-12** Poder cambiar la configuración del proyecto para poder hacer cambios en las vistas del proyecto

El proyecto está formado por 6 subproyectos o soluciones(Figura29):

- ***Core:*** Proyecto que tiene todas las entidades e interfaces que declaran operaciones sobre ellas.
- ***Infraestructura:*** Capa que implementa las interfaces de la capa anterior y que realiza las operaciones sobre la BD
- ***Aplicación:*** Contiene la lógica de la aplicación.
- ***API:*** Capa de abstracción de la capa de aplicación.
- ***Api SharePoint***: Encargada de leer datos desde el SharePoint de la empresa.
- ***Presentación:*** Capa de presentación e interfaz de la aplicación.

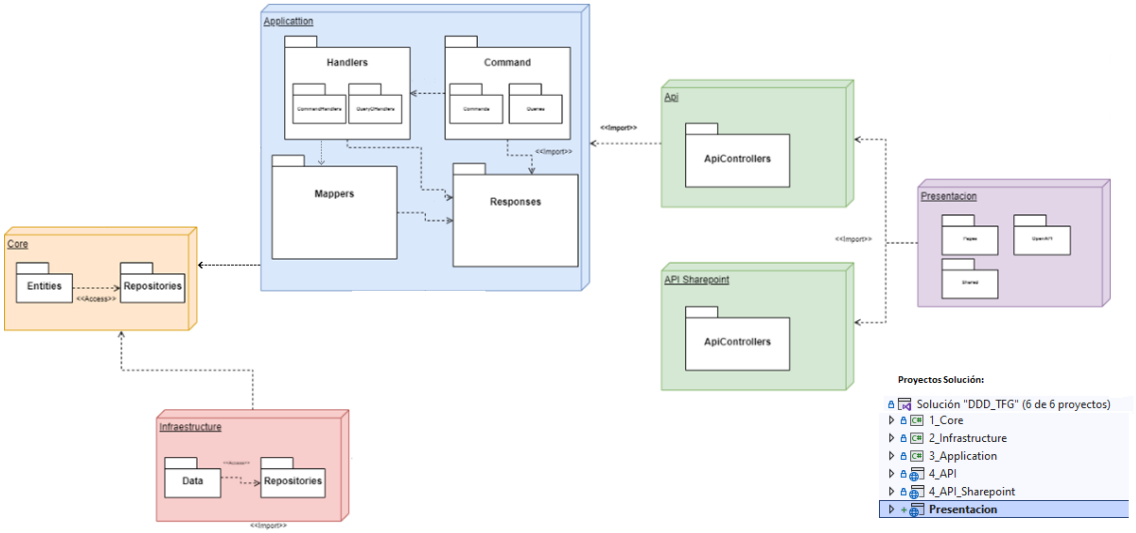


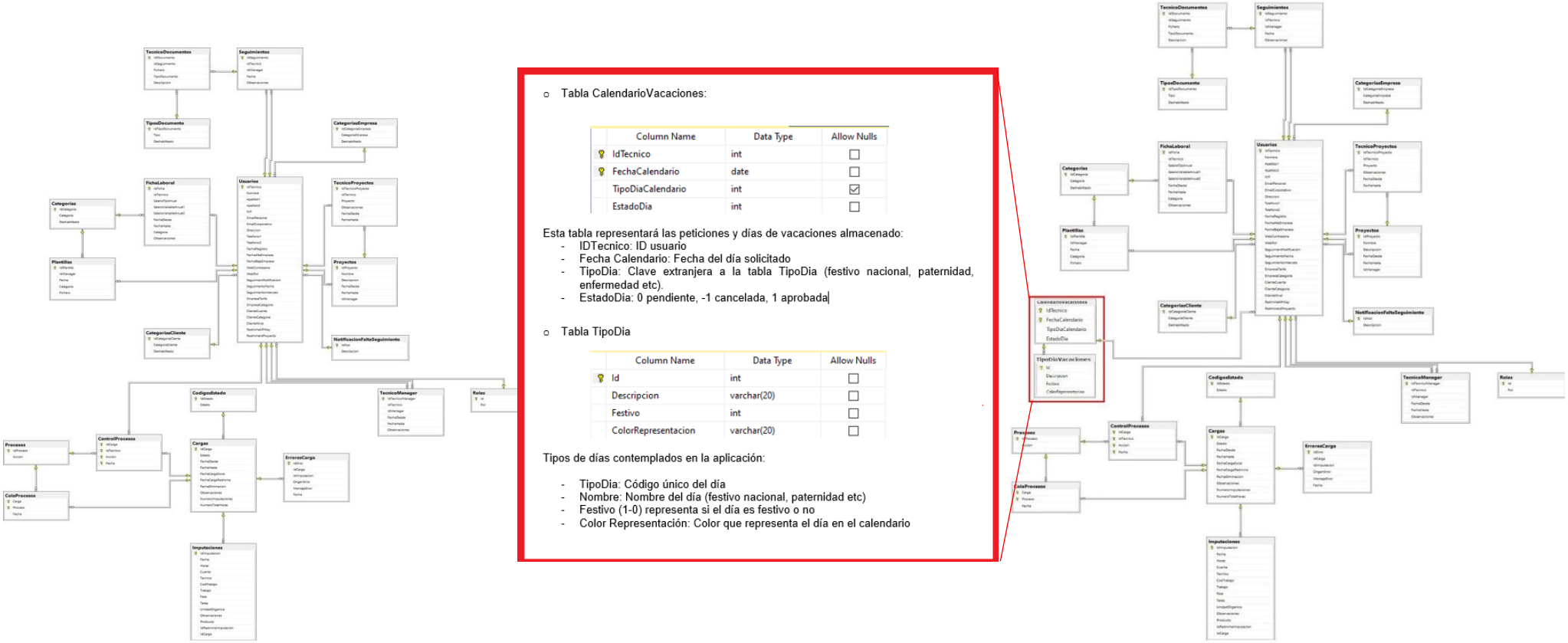
Figura 29: Diagrama de paquetes

Como se ha comentado en el apartado anterior la principal característica de la arquitectura elegida es que no existe ningún tipo de dependencia con la BD.

3.3. Descripción de la BD

- ✓ **RF-4** Las solicitudes se persistirán en la base de datos.
- ✓ **RF-5:** Comprender y ampliar el diseño de base de datos existentes

En la aplicación de GesTec se utilizaba ya un modelo de Base de datos, que se muestra en la figura siguiente. Únicamente se han tenido que añadir dos tablas, la tabla de Calendario Vacaciones y de Tipo Día para gestionar las vacaciones de los usuarios.
En la siguiente figura se puede ver el diagrama de base de datos antes de añadir las dos nuevas tablas calendario Vacaciones y tipo día



- Tabla CalendarioVacaciones:

Column Name	Data Type	Allow Nulls
IdTecnico	int	☐
FechaCalendario	date	☐
TipoDiaCalendario	int	☑
EstadoDia	int	☐

Esta tabla representará las peticiones y días de vacaciones almacenado:
- IDTecnico: ID usuario
- Fecha Calendario: Fecha del día solicitado
- TipoDia: Clave extranjera a la tabla TipoDia (festivo nacional, paternidad, enfermedad etc).
- EstadoDia: 0 pendiente, -1 cancelada, 1 aprobada

- Tabla TipoDia

Column Name	Data Type	Allow Nulls
Id	int	☐
Descripcion	varchar(20)	☐
Festivo	int	☐
ColorRepresentacion	varchar(20)	☐

Tipos de días contemplados en la aplicación:

- TipoDia: Código único del día
- Nombre: Nombre del día (festivo nacional, paternidad etc)
- Festivo (1-0) representa si el día es festivo o no
- Color Representación: Color que representa el día en el calendario

Figura 30: Diagrama de BD

4. Implementación

En esta sección se expondrán brevemente los objetivos y resultados de la implementación realizada en cada sprint.

4.1. Sprint 1

✓ **RF-1**: Conectar con la aplicación de GesTEC.
✓ **RF-2**: Crear prototipos de interfaces para que sean evaluados por el cliente
✓ **RF-3**: Se utilizará como modelo de clases los creados en el Proyecto GesTEC.

Este sprint se ha dedicado a seleccionar y estudiar las distintas tecnologías que se van a utilizar durante el proyecto y tratar de desarrollar una estimación de plazos y costes de uso de estas.

Durante esta primera fase se han tratado de comprender los aspectos de la aplicación GesTEC existente, integrando en el sistema el módulo a desarrollar y se ha procedido a su documentación, recogida en los apartados anteriores de esta Memoria.

Se han presentado y aprobado al cliente los prototipos de las interfaces que se van a realizar. Se han reutilizado los modelos de dominio y de BD existentes en la aplicación GesTEC.

Fechas: 14-27 marzo, semanas 11 y 12.

4.2. Sprint 2

✓ **RF-6:** Conectar la base de datos con Blazor y procesar datos
✓ **RF-7:** Las vacaciones se persistirán en la base de datos.
✓ **RF-8:** Se guardará el estado de las vacaciones (aprobada, cancelada etc.).

Comienzo de la implementación del proyecto centrando el desarrollo en la capa de datos y priorizando los nuevos diseños de BD, reutilizando los datos existentes de la aplicación GesTEC.

La capa de acceso a datos sigue el patrón CQRS, es decir divide responsabilidades de comandos y consultas, un patrón que separa las operaciones de lectura y actualización de datos [4].

Se ha comenzado a utilizar el tablero KanBan con la distribución de tareas por sprint (Figura 31).

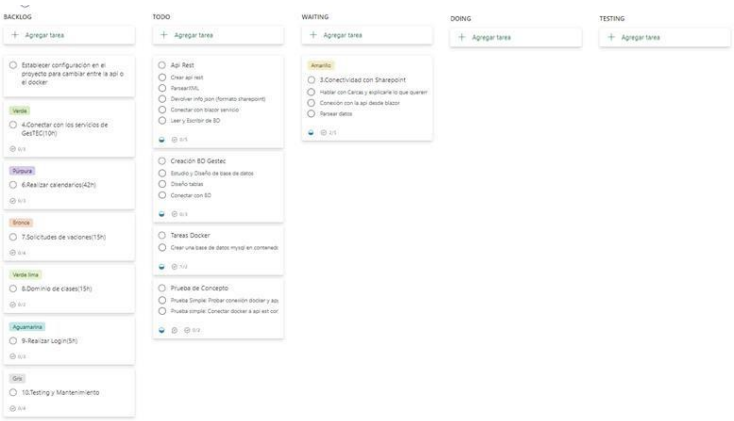


Figura 31: Ejemplo tablero KanBan

Al tratarse de datos sensibles se debe pedir permiso al departamento de RRHH de Hiberus y crear un usuario con los permisos necesarios, como estos permisos son complejos de administrar y costosos se ha proseguido con el proyecto, aunque no se dispone de ellos.

Fechas: 28 marzo-8 abril, semanas 13 y 14.

4.3. Sprint 3

✓ **RF-10**: Leer datos de SharePoint y procesarlos.
✓ **RF-11:** Diseñar una Api para lectura y procesado de datos.
✓ **RF-16:** RF-16 Habrá un sistema de login para poder acceder.
✓ **RF-17:** Existirán distintos tipos de usuarios con distintos privilegios.
✓ **RF-26:** Se desarrollará un sistema para cambiar los datos del usuario y poder eliminar su cuenta

Comienzo del desarrollo, estudio e implementación del esqueleto de los proyectos de la aplicación.

Durante este sprint se ha desarrollado el sistema de Login-Registro utilizando las librerías de AspNet authentication, la cual genera automáticamente el código necesario para gestionar roles y cambios de datos en el perfil (Figura 32).

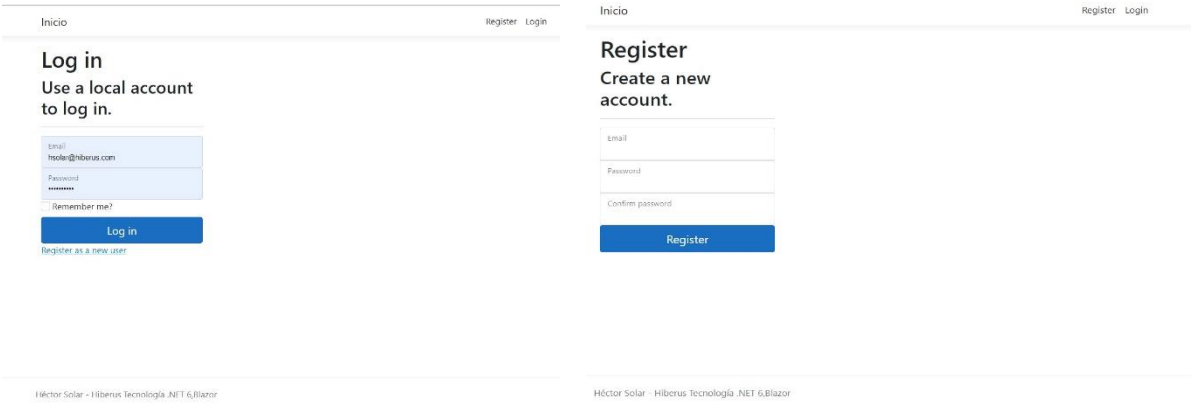


Figura 32: Login

Primeros prototipos y funcionalidades de la aplicación y desarrollo de creación y primeras llamadas a la API de la aplicación además de la realización de la API que lee datos desde SharePoint (Parte izquierda de la Figura 33).

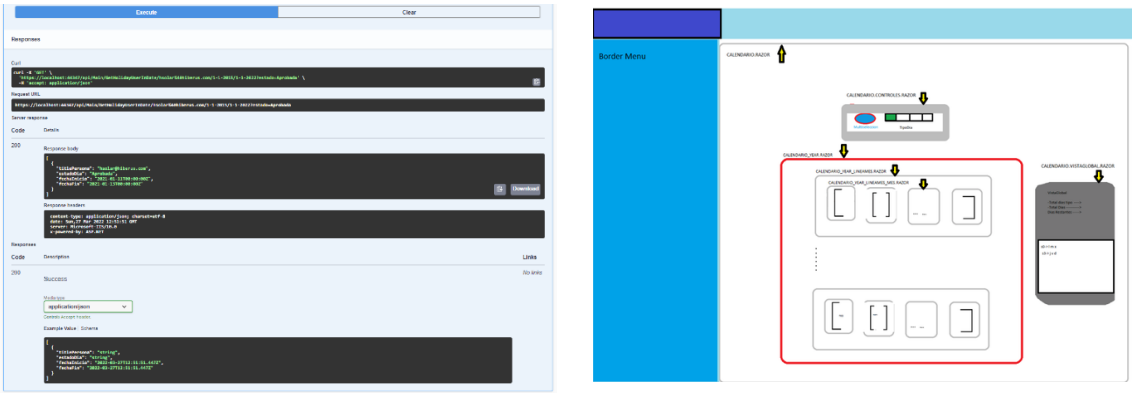


Figura 33: Primeras llamadas a la API y prototipos sprint3

El diseño de los prototipos se trata de un diseño centrado en componentes, es decir que cada página de la aplicación estará formada por pequeños componentes reutilizables con una función única dentro del contexto de la aplicación.

En la ilustración anterior (Figura 33) se puede ver como el boceto de la página de elección de vacaciones estaría formado por diversas componentes indicadas con flechas (parte derecha de la imagen).
En los siguientes ejemplos se puede ver como se ha construido la aplicación:

A continuación, se muestran ejemplos del desarrollo utilizando Blazor y sus componentes:

1. MVVM Blazor

Este es el resultado del archivo de la vista de la página de peticiones, este archivo es de extensión .Razor y combina lenguaje de marcado HTML, CSS y C#:

Se puede observar (Figura 34) como en la vista **PeticionesPage.razor,** el evento click del botón oculta o muestra la componente de tarjeta, es decir, los cambios en los objetos producen directamente un cambio directo en el HTML de la página.

```
@page "/Peticiones"
<h3>Pagina Principal</h3>

<RadzenCard Class="mt-4" Visible>
    <RadzenButton Click="()=>PeticionesPropias.Visible=!PeticionesPropias.Visible"
                  Text="@(PeticionesPropias.Visible? "Ocultar" : "Mostrar")"
                  ButtonStyle="@(PeticionesPropias.Visible? ButtonStyle.Light :  ButtonStyle.Primary)"> <,
    <RadzenCard Class="mt-4" @ref="PeticionesPropias">
        <PeticionesVacacionesPropias />
    </RadzenCard>
</RadzenCard>

<br/>

<RadzenCard Class="mt-4">
   <RadzenButton Click="()=>InteractiveCalendar.Visible=!InteractiveCalendar.Visible"
                  Text="@(InteractiveCalendar.Visible? "Ocultar" : "Mostrar")"
                  ButtonStyle="@(InteractiveCalendar.Visible? ButtonStyle.Light :  ButtonStyle.Primary)">
    <RadzenCard Class="mt-4" @ref="InteractiveCalendar">
       <BotonesTiposDias Multiseleccion=Multiseleccion/>
       @*<ResumenMensual/>*@
    </RadzenCard>
</RadzenCard>
```

Figura 34: Código vista de peticiones .Razor

Esta referencia de componentes lo conseguimos mediante la palabra reservada @ref, que indica que esa componente específica tendrá ese nombre asignado.

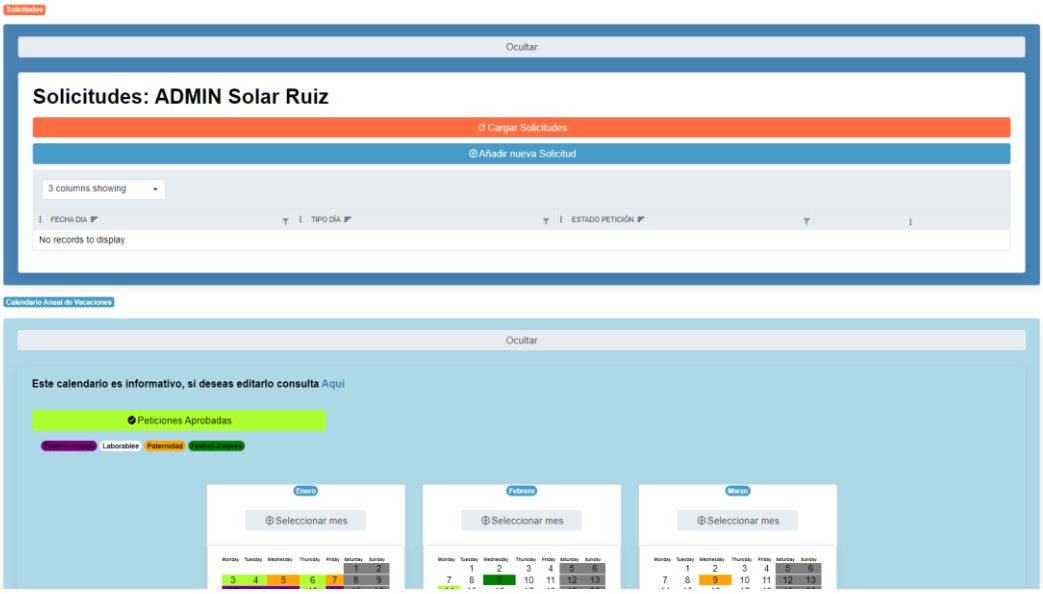


Figura 35: PeticionesPage.razor vista en el navegaodr

La visualización de la petición se puede observar en la Figura 35.
En la Figura34 se puede observar cómo se utilizan componentes de la librería a las que se le añaden manejadores de eventos y atributos que definiremos en el código C# de la aplicación

Para la lógica de la página (ViewModel) se utiliza un archivo con el mismo nombre que el anterior pero terminado en .cs (extensión para archivos C#), opcionalmente, como se también se puede añadir un archivo que terminara .css.

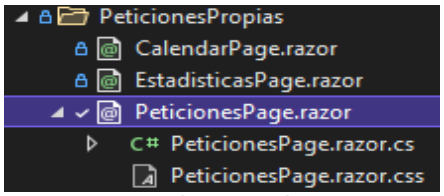


Figura 36: Disposición de las páginas en Blazor

Con esta distribución se puede acceder rápidamente a la parte del código CSS por un lado (estética, eventos, etc), al ViewModel (.cs contiene la lógica de la página C#), y al View con la presentación de la página (.Razor).
La siguiente figura muestra un ejemplo de ViewModel en la aplicación que encapsularía toda la lógica de la página: **PeticionesPage.razor.cs.**
Figura 37 muestra el ejemplo del ViewModel de la página de peticiones:

La Figura 37 muestra el ejemplo del ViewModel de la página de peticiones:

```
namespace BlazorApp2.Pages.PeticionesPropias;
-- referencias | 0 cambios | 0 autores, 0 cambios
public class PeticionesPageBase: ComponentBase {


    [Inject]
    2 referencias | 0 cambios | 0 autores, 0 cambios
    protected  API _api { get; set; }
    -- referencias | 0 cambios | 0 autores, 0 cambios
    public IEnumerable<TipoDiaCalendarioResponse> ColoresBotones { get; set; } = new List<TipoDiaCalendarioResponse>();
    protected RadzenCard TarjetaPeticionesPropias = new();
    protected RadzenCard TarjetaCalendario = new();

    0 referencias | 0 cambios | 0 autores, 0 cambios
    protected override async Task OnInitializedAsync() {
        ColoresBotones = await _api.GetAllTipoDiaCalendarioAsync();
        StateHasChanged();
    }

    1 referencia | 0 cambios | 0 autores, 0 cambios
    public async Task RefreshCalendar() {
        this.TarjetaCalendario.Visible = false;
        StateHasChanged();
        this.TarjetaCalendario.Visible = true;
        StateHasChanged();

    }
}
```

Figura 37: ViewModel de la página

2. Sistema de componentes de Blazor

El sistema de componentes podemos verlo en el siguiente ejemplo. Se ha utilizado la página de "*Ver vacaciones",* similar al ejemplo anterior:

Esta página es una componente, dentro de ella existen, a su vez, otras que pueden tener otras componentes hijas.

Este anidamiento de componentes permite que cada componente pueda ver el estado de sus componentes hijas, modificando su estado y enviando o recibiendo parámetros (por ejemplo, eventos para notificar cambios en la aplicación) que modifican su comportamiento en función de los mismos, lo vemos representados por colores en la figura 38.

A nivel de arquitectura de software cada componente es únicamente responsable de sus funciones, lo que permite aislar la lógica y separar responsabilidades.

Esta fase se ha alargado debido a que implementar la arquitectura DDD de la aplicación costó más tiempo del estimado, en parte debido a la formación y comprensión necesaria para el despliegue de la misma:

Fechas: 28 marzo - 10 abril, semanas 16 y 17.

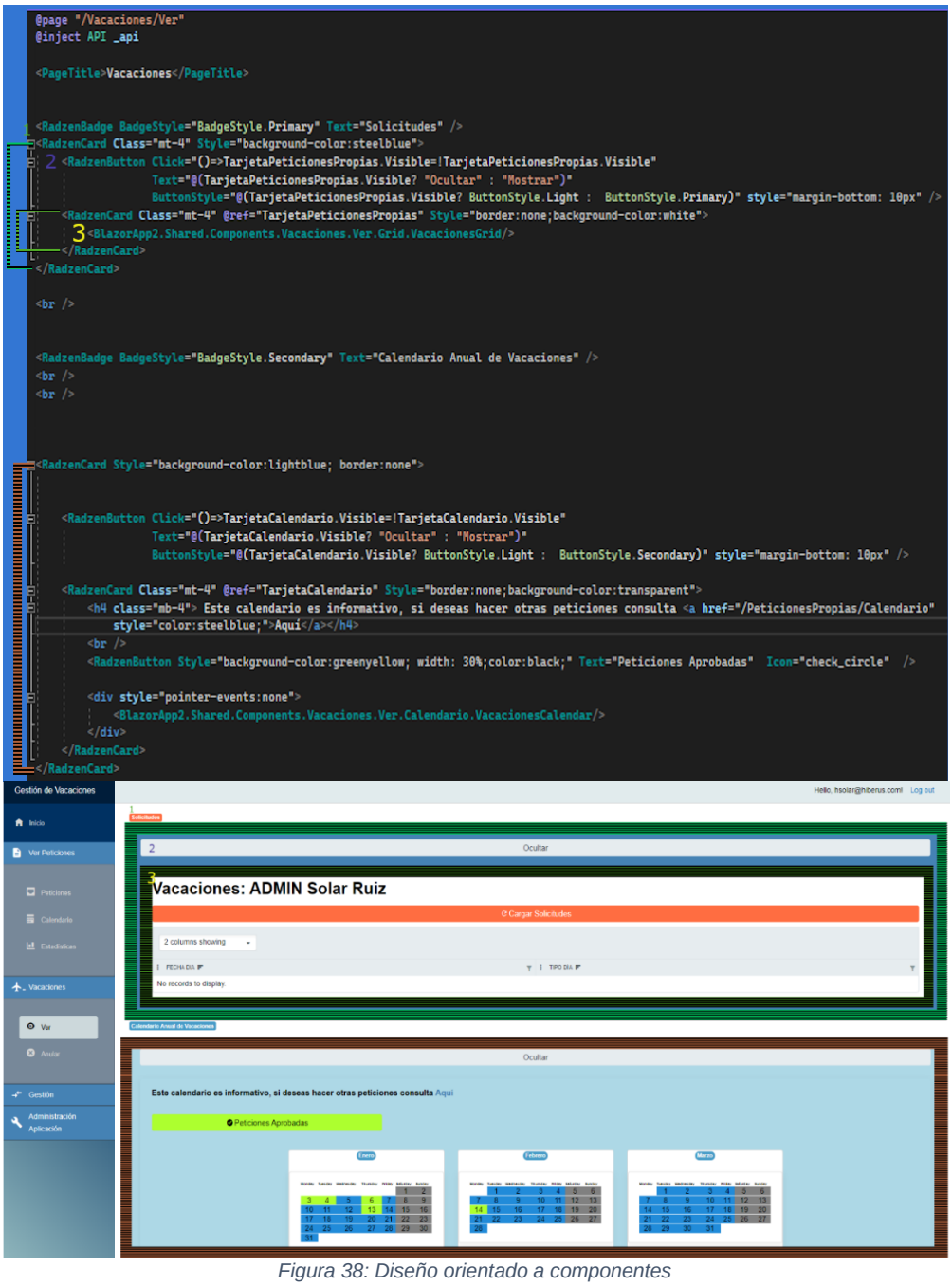


Figura 38: Diseño orientado a componentes

4.4. Sprint 4

- ✓ **RF-13:** Se podrán importar distintos calendarios de vacaciones.
- ✓ **RF-14:** Se establecerán mecanismos para aprobar/cancelar solicitudes.
- ✓ **RF-15:** Se podrán enviar solicitudes que serán aceptadas o no.
- ✓ **RF-18:** Existirá el usuario administrador global que podrá administrar la aplicación.
- ✓ **RF-21:** El responsable del equipo podrá ver las vacaciones de su equipo.
- ✓ **RF-22:** Las vistas serán responsive.
- ✓ **RF-24:** Para usar la aplicación se deberá estar logueado.
- ✓ **RF-25:** Los usuarios registrados podrán loguearse.

Durante este periodo se han terminado de desarrollar las funcionalidades básicas de la aplicación.

Las librerías anteriormente mencionadas facilitan enormemente del desarrollo ya que de esta forma apenas se tiene que programar la lógica y gestión de errores de las distintas componentes.

El resultado de usar estas librerías mejora enormemente la calidad de los desarrollos a nivel estético y a nivel técnico ya que simplifica enormemente los desarrollos.
Estas componentes permiten que el contenido de las páginas sea responsive.

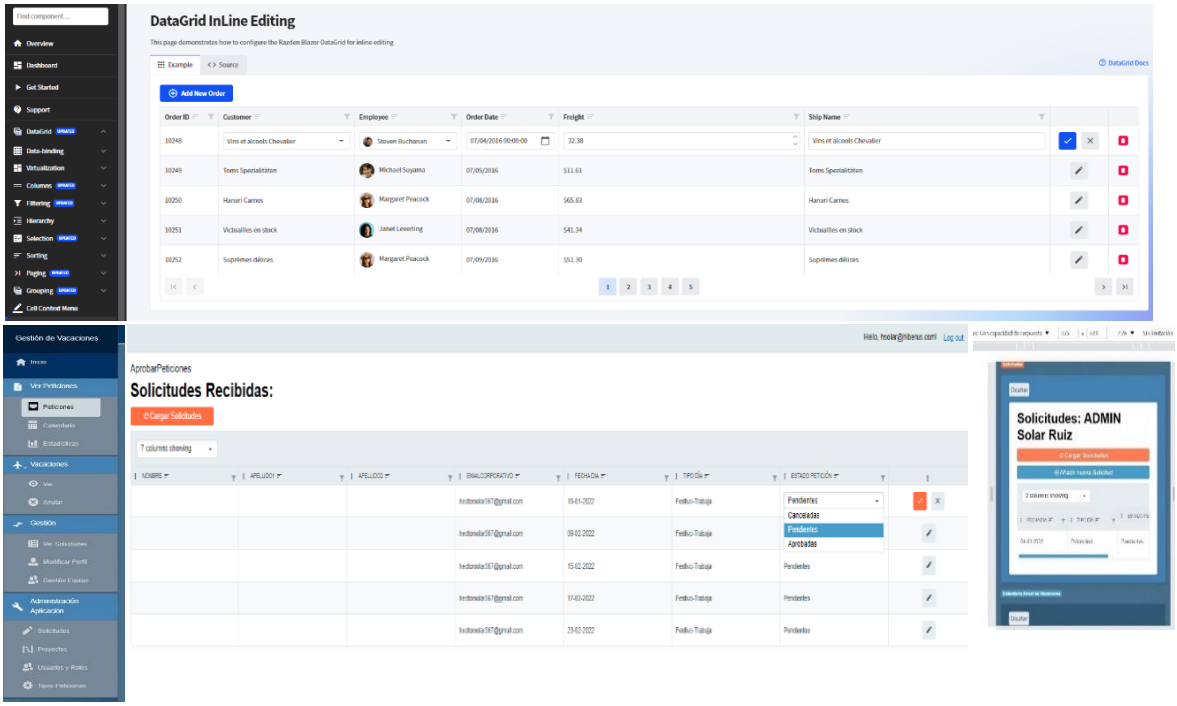
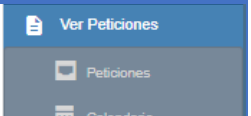
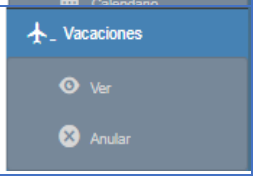


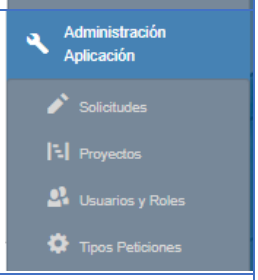
Figura 39:Funcionalidades de la librería Radzen para la aplicación

Se han terminado de realizar las 5 “áreas” principales de la aplicación,

Para los 3 roles en la aplicación, Usuario Registrado, Manager Proyecto y administrador, se podrán acceder a distintas zonas de la aplicación:

Tabla 3: Áreas de la aplicación y roles principales

Nombre Área	Roles que acceden:		
Peticiones	Usuario registrado, Administrador	Acceso y modificación de peticiones de vacaciones.	
Vacaciones	Usuario registrado, Administrador	Ver y cancelar vacaciones concedidas.	

Nombre Área	Roles que acceden:		
Gestión	Manager Proyecto, Administrador	Gestión de datos personales, solicitudes y vacaciones del equipo.	
Administración	Administrador	Modificar datos de la aplicación.	
Perfil	Todos los usuarios	Modificar datos de la cuenta,	

Fechas: 19 abril – 1 mayo, semanas 18 y 19.

4.5. Sprint 5

✓ **RF-23:** Los resultados de las vacaciones aprobadas del equipo serán generadas en una página centralizada.
✓ **RF-28:** Las interfaces y procesos estarán documentados con los distintos diagramas pertinentes.

Durante este sprint se han desarrollado los prototipos e implementadas las distintas interfaces del proyecto. Estas interfaces han sido prototipadas en papel con el fin de ser aceptadas de manera rápida por el cliente y comenzar su desarrollo lo antes posible dentro del sprint.

Algunos de los prototipos y sus desarrollos en este sprint[8] son mostrados a continuación en las figuras 40,41,42.

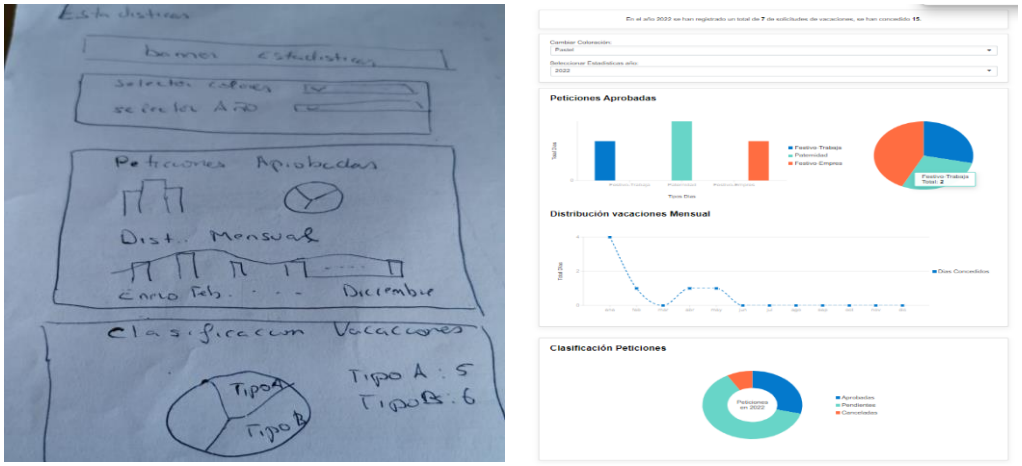
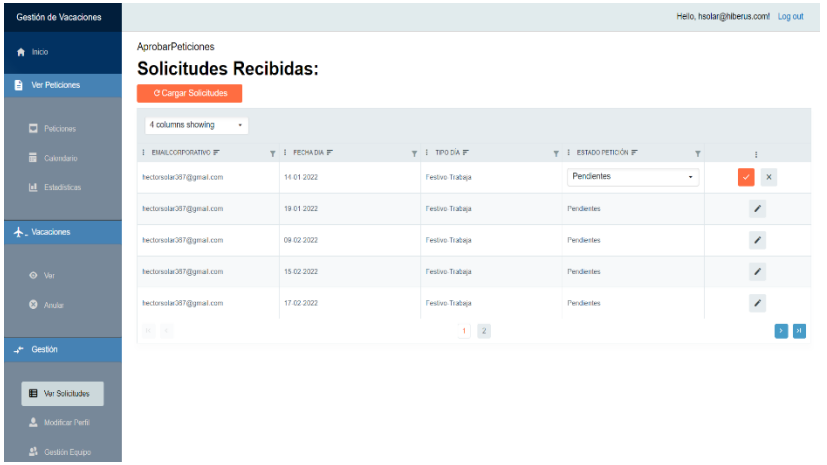


Figura 41:Resultado de la implementación página de estadísticas



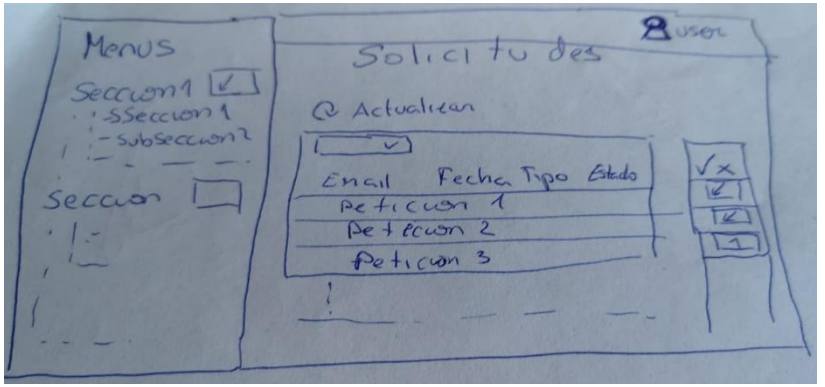


Figura 40:Página de aprobar peticiones

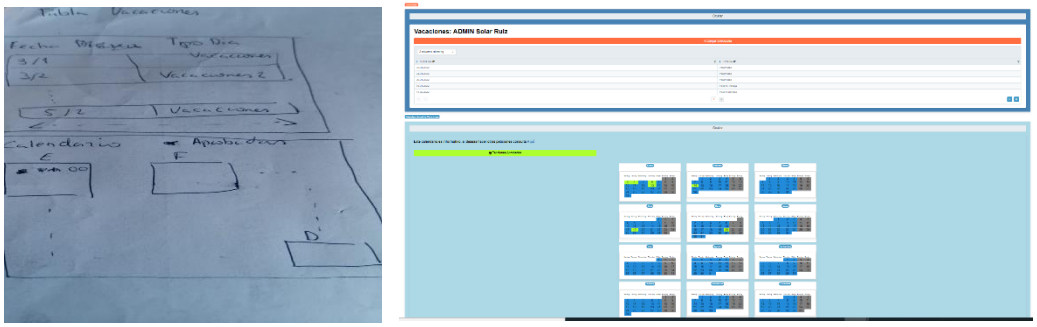


Figura 42:Resultado de página de calendario vacaciones

[8] Nota: Estos son algunos de los resultados tras la finalización del sprint, no se adjuntarán todos los prototipos porque no se consideran necesarios ya que no tienen relevancia final en el proyecto.

4.6. Sprint 6

Durante esta fase del proyecto se ha realizado el testeo y la comprobación de todos los requisitos funcionales y no funcionales de la aplicación.

Se ha comprobado que se satisfacen las necesidades del cliente ante un producto que dispone de las funcionalidades deseadas.

A nivel técnico estas pruebas se pueden consultar en el apartado (5.1 Pruebas), de esta Memoria.

Fechas: 30 mayo –8 junio, semanas 22 y 23.

4.7. Sprint 7

Finalmente, en el sprint 7 se han terminado los aspectos de documentación de la Memoria y la preparación para la presentación del TFG. También se ha documentado de manera más formal la parte de la sección técnica, incluyendo ejemplos propios de la aplicación que ayuden a entender la tecnología (uso de Blazor, MVVM, arquitectura DDD etc).

En el tiempo restante, y con el objeto de ir un poco más allá en el uso de esta toda esta tecnología, se han desarrollado pequeños aspectos con .NET MAUI, que serán explicados mejor en el apartado 5.4.1 Pruebas con .NetMAUI.

Fechas: 13 junio –24 junio, semanas 24 y 25

5.1.Pruebas realizadas

Debido al ajuste necesario en los tiempos para este proyecto, al requerirse algo más formación en la aplicación de algunas partes, ha quedado pendiente automatizar las pruebas de la API que gestiona todos los datos de la aplicación. No obstante, sí que se ha podido ir comprobando manualmente que todos los métodos sobre las entidades funcionaban de forma correcta.

Por ejemplo, en el caso de la entidad Calendario de vacaciones, se dispone de varios métodos. En las siguientes figuras se muestra el método GetAllCalendarioVacaciones, el cual devolverá todos los objetos de este tipo guardados en la BD.

En la BD existen 4 registros distintos de estas solicitudes:

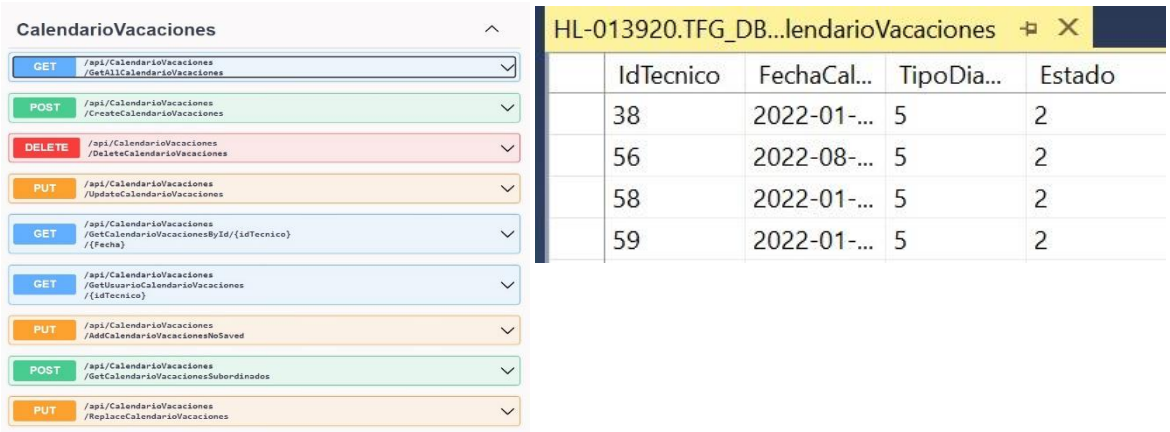


Figura 43: Ejemplo prueba sobre Calendario Vacaciones y sus registros en BD

Tras ejecutar el método en la API, se puede ver que el resultado es el esperado, ya que se obtienen 4 instancias con los mismos datos que existen en la BD:

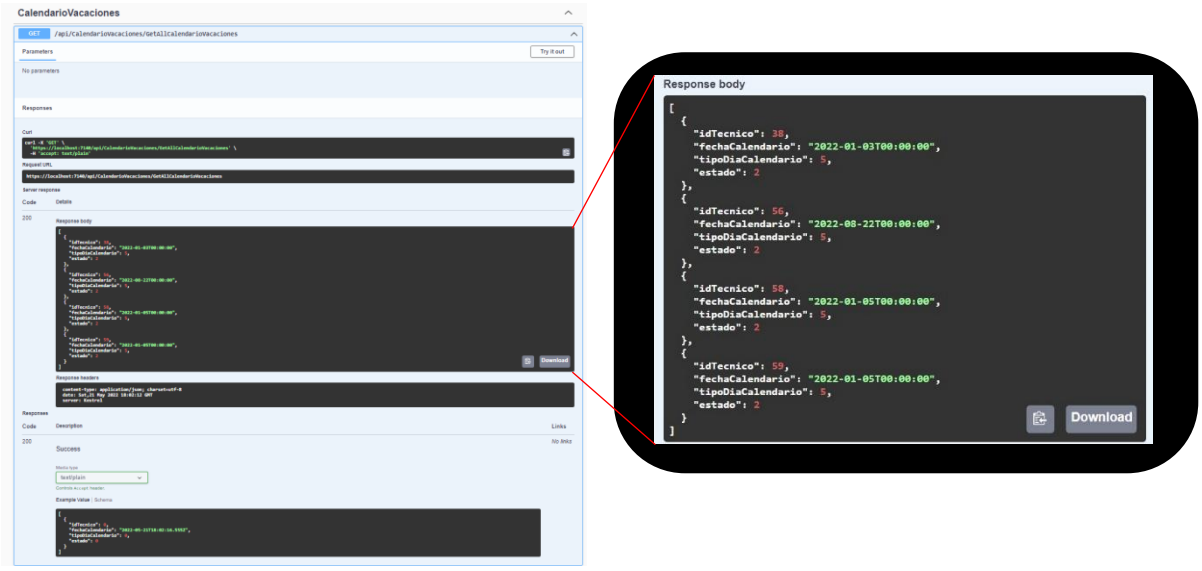


Figura 44: Resultado pruebas sobre la API

Al finalizar este proceso de pruebas una vez comprobado el correcto funcionamiento de la API se utilizó para las distintas funcionalidades de todo el frontend.

En el caso de las pruebas para el frontend, en los distintos sprints se han ido comprobando con el cliente que las funcionalidades de la aplicación se iban cumpliendo, modificando aquellas que resultaban menos convenientes.

En aquellos casos en los que podían producirse errores debido a casos límites que pudieran generar un fallo en la aplicación, se han utilizado la funcionalidad de Blazor de Error Boundary[9].

Como se ha comentado en la descripción de las tecnologías, Blazor tiene siempre un estado mientras los usuarios interactúan con una aplicación y mantienen una conexión con el servidor, lo que se denomina circuito. Blazor trata la mayoría de las excepciones no controladas como graves para el circuito en el que se producen. Si se finaliza un circuito debido a una excepción no controlada, el usuario solo puede seguir interactuando con la aplicación si recarga la página para crear otro circuito.

Este escenario es parecido a cuando una aplicación de escritorio se bloquea debido a una excepción o error. La aplicación bloqueada debe reiniciarse.

Para tratar de recoger estas excepciones no controladas aparece el componente nativo de Blazor Error Boundary:

Los Boundary proporcionan una forma sencilla para controlar las excepciones.

Tienen dos partes:

- **Child Content:** Representa su contenido cuando no se ha producido un error.
- **Error Content:** Representa contenido secundario de error cuando se produce una excepción no controlada en el Child Content.

A continuación, se recrea un ejemplo de este tipo de control de excepciones en la aplicación desarrollada.

En la página de solicitudes de la aplicación (Figura 45 parte izquierda), se añade mediante el botón azul una nueva solicitud, esta solicitud es exactamente igual que la ya existente (Figura 45 parte derecha), es decir con total seguridad al intentar insertar en BD un objeto con la misma clave producirá un error:

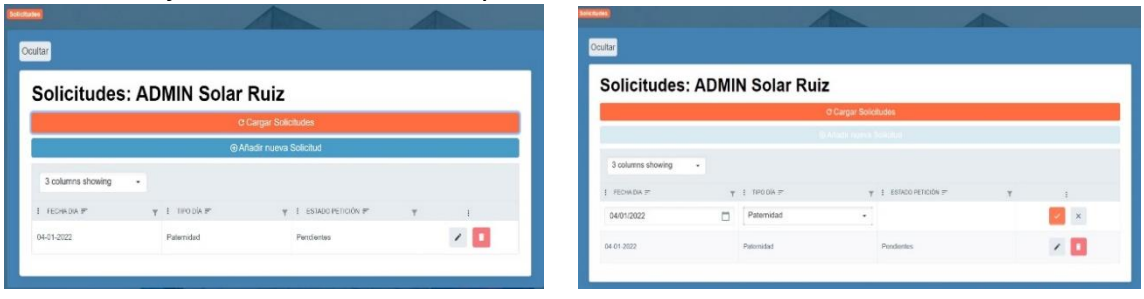


Figura 45: Ejemplo situación de error

[9] Ver: https://docs.microsoft.com/es-es/aspnet/core/blazor/fundamentals/handle-errors?view=aspnetcore-6.0

En el momento que se produce esta excepción es cuando entra en juego el error Boundary ya que al haber detectado una excepción muestra el siguiente *error content* (Figura 46), “No se pudo insertar” en rojo (Figura 47).

```
<EditTemplate Context="order">

    <ErrorBoundary @ref="ErrorBoundaryInsercionesNomodificaciones">
        <ChildContent>
            <RadzenButton Icon="check" ButtonStyle="ButtonStyle.Primary" Class="m-1" Click="@((args) => SaveRow(order))"/>
            <RadzenButton Icon="close" ButtonStyle="ButtonStyle.Light" Class="m-1" Click="@((args) => CancelEdit(order))"/>
        </ChildContent>
        <ErrorContent> <RadzenButton Icon="report" ButtonStyle="ButtonStyle.Danger" Click="RecoverAppState" Text="No se pudo insertar"/>  </ErrorContent>

    </ErrorBoundary>
</EditTemplate>
```

Figura 47: Declaración y partes del Boundary

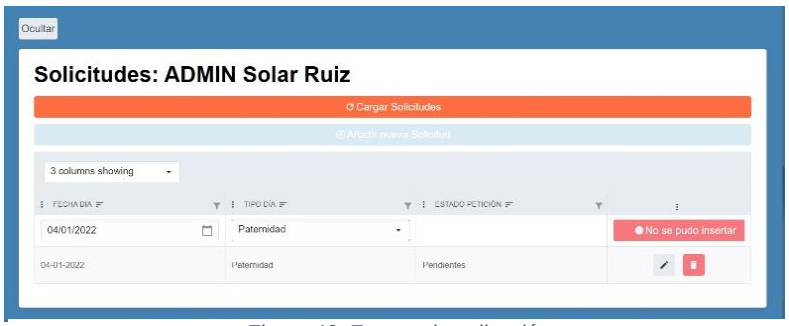


Figura 46: Error en la aplicación

Es decir, aunque existe una excepción debida a la inserción en BD, la aplicación puede seguir utilizándose sin necesidad de recarga ya que el Boundary ha encapsulado este error.

Además, permite recuperar el estado del circuito anterior al error utilizando su método recover.

De esta manera podemos gestionar excepciones sin necesidad de recargar la aplicación de una forma visual y manteniendo el estado de consistencia de la aplicación.

Finalmente, para comprobar el correcto funcionamiento de la BD se han poblado todas las tablas y se han ido comprobando que las restricciones y relaciones de las tablas eran correctas.

5.2. Alcance conseguido

Requisitos Alcanzados

El proyecto ha sido satisfactorio a nivel de cliente y desarrollador.

Se han cumplido prácticamente todos los requisitos deseados por el cliente salvo los siguientes:

- ✗ **RF-9**: Conectar con SharePoint.
- ✗ **RF-19**: Mostrar los datos de SharePoint.
- ✗ **RF-20**: Páginas para importar y mostrar datos de SharePoint.

Incidencias y cambios

Los requisitos 9, 19 y 20 no han podido ser cubiertos debido a factores externos al desarrollo. Ambos tenían una dependencia directa de la utilización de los datos de SharePoint de la empresa. Debido a problemas con los usuarios y permisos internos de la empresa finalmente no se me han facilitado a tiempo los permisos necesarios de acceso y gestión de estos datos. No obstante, la aplicación está lista para su integración tras la creación de un usuario con los permisos necesarios.

5.3. Tiempo

Dedicación Real

Como se puede observar en las figuras siguientes(49 ,49), prácticamente se han mantenido los tiempos de planificación salvo en el sprint 3 que duró una semana más. Esta modificación se vio corregida en el siguiente sprint, recuperando en parte ese tiempo, aunque viéndose afectado parte del plan de pruebas, que se decidió reducirlo, debido al control que presenta el componente nativo de Blazor Error Boundary, comentado en el apartado 5.1 Pruebas realizadas.

La desviación fue debida al tiempo de formación extra requerido para comprender la arquitectura DDD y la complejidad en su implementación.

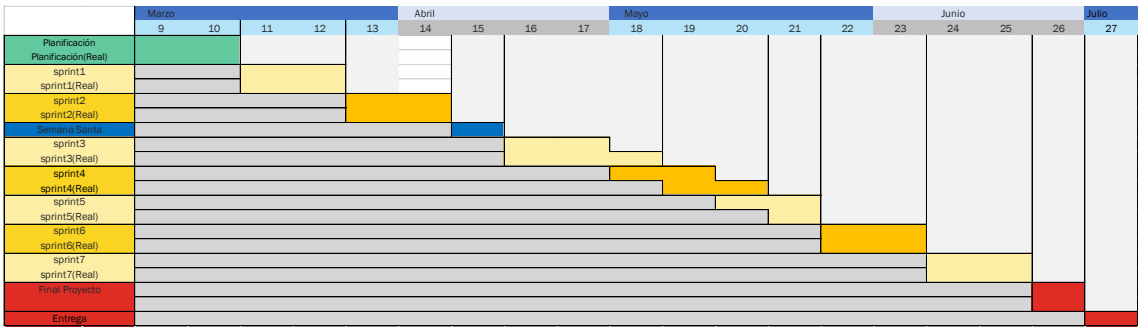


Figura 49: Resultado final de tiempos del proyecto

ID Paquete trabajo		FechaInicio	FechaFin	Sprints Realización	Estimación horas	Horas Reales	Estimación (h)	Reales(h)
1.Planificación Proyecto		01/03/2022	13/03/2022				10	10
1.1 Reuniones				s0(planificación)	4	4		
1.2 Estudio de Viabilidad				s0(planificación)	1	1		
1.3 Tecnologías				s0(planificación)	4	4		
1.4 Plan de Riesgos				s0(planificación)	1	1		
2.Desarrollo Módulo GesTec		13/03/2022	01/07/2022				185	185
2.1 Análisis				s1	20	20		
2.2 Diseño		13/03/2022	31/03/2022	s1	23	23		
	2.2.1 Prototipo de Interfaces				5	2		
	2.2.2 Diseño BD				3	3		
	2.2.3 Diagramas				12	9		
	2.2.4 Arquitectura de la App				3	9		
2.3 Implementación		31/03/2022	31/06/2022	s2-s6	162	154		
	2.3.1 Base de Datos			s2	10	10		
	2.3.2 Capa de Persistencia			s2	25	25		
	2.3.3 Capa Lógica de Negocio			s3-s4	59	72		
	2.3.4 Capa Presentación			s4-s5	40	32		
2.4 Pruebas				s2-s6	28	15		
3.Seguimiento y Control		01/03/2022	01/07/2022	s0(planificación)-final			29	29
3.1 Kanban				s0(planificación)	3	3		
3.2 Scrum				s0(planificación)-fin	10	10		
3.3 Reuniones				s0(planificación)-fin	8	8		
3.4 Plan de Riesgos				s1-fin	8	8		
4.Memoria		01/03/2022	01/07/2022	s7-fin			70	70
5.Defensa		27/07/2001	01/07/2022	fin			6	6
							300	300

Figura 48: Desempeño final de horas

En general, en cuanto al desempeño total de horas, el tiempo dedicado a la arquitectura y la complejidad del backend se ha corregido restando tiempo a la parte de testeo automático e implementando interfaces de menor complejidad estética, no restando funcionalidad a la aplicación.

Reuniones

Las reuniones para gestionar los diversos aspectos del proyecto han sido de dos tipos:

- Reuniones semanales con la tutora del proyecto en la universidad.
- Daily Sprints Meetings y reuniones al final de cada semana con el tutor de la empresa Hiberus.

Gestión de Riesgos

A parte de las modificaciones en el calendario mencionadas en el apartado anterior, no han sucedido otros riesgos o factores externos que hayan hecho que el proyecto se retrase.

Se hace necesario mencionar que, aunque por temas de gestión y permisos internos de la empresa no se ha podido realizar la integración con SharePoint, sí que la API está preparada para integrar los datos de la empresa cuando sea posible.

5.4. Pruebas con .NET MAUI

Como se ha comentado en el apartado de Tecnologías de esta Memoria, lo último en .NET es .NET MAUI con el que se pueden desarrollar aplicaciones nativas que se pueden ejecutar en distintos dispositivos desde un único código base compartido. Durante el sprint 7, y ya fuera del alcance de este proyecto, se ha podido hacer alguna prueba reutilizando el código de las componentes Blazor de este proyecto para su uso con .NET MAUI.

Las pruebas realizadas se han hecho con propósitos de investigación y formación en este tipo de tecnología ya que actualmente es una tecnología muy nueva. No se ha trabajado desde el comienzo del proyecto ya que está totalmente desaconsejado para producción, al tratarse de una tecnología que sólo está disponible en la versión preliminar de .NET 7 que saldrá el próximo año.

El desarrollo realizado, aún con la aparición de errores, indica que se trata de una tecnología prometedora de cara a crear aplicaciones nativas para todos los sistemas operativos y dispositivos.

Se ha podido constatar que el mismo código puede generar aplicaciones en Android y Windows, reutilizando componentes Blazor. No se han conseguido grandes resultados, pero sí se deja entrever que el futuro del framework tendrá aplicaciones nativas multiplataforma e híbridas.

5.5. Resultados del proyecto

El resultado general del proyecto es satisfactorio ya que se han cumplido las necesidades del cliente en los plazos estimados.

La aplicación principal dispone de todas las interfaces y funcionalidades planificadas dentro del alcance del proyecto.

Se han creado las 5 secciones principales que debía tener:

1. Login y registro, datos personales de usuario
2. Realización y comprobación de estado de solicitudes
3. Vista global y anulación de vacaciones
4. Gestión de vacaciones del equipo
5. Administración de la aplicación

La aplicación está apoyada sobre un backend construido con arquitectura DDD que contiene toda la lógica y reglas de negocio de la aplicación, externalizando el acceso a BD.

La lógica de la aplicación está encapsulada sobre una API que el Frontend utiliza.

Y se ha desarrollado también hay otra API que está preparada para integrar los datos de SharePoint cuando así lo decida la empresa.

6. Conclusiones

6.1. Conclusiones y Lecciones Aprendidas

El Trabajo de Fin de Grado ha consistido en la realización de un sistema de gestión de vacaciones y turnos para los empleados de la empresa Hiberus.

El sistema consta de 4 secciones principales:

- Solicitudes de vacaciones: Permite al usuario solicitar vacaciones y ver el estado de sus solicitudes.
- Vacaciones: Muestra la distribución de vacaciones concedidas y permite cancelarlas.
- Gestión de Equipo: Permite aprobar/cancelar las peticiones a las personas del equipo y ver como se distribuyen en un calendario global.
- Administración de la Aplicación: Sección de administración de la aplicación que permite modificar los datos de la aplicación.
- Login, registro y modificación de datos personales de usuario.

Su integración en proyecto más grande (Aplicación GesTEC) se ha realizado de forma paralela, como se ha ido comentando en esta Memoria.

La implementación se ha adaptado a la base de datos existente, el desarrollo inicial del backend es prácticamente nuevo además de su arquitectura, las comunicaciones con la aplicación GesTec y el frontend. Se trata de una aplicación lista para producción.

Como trabajo futuro del proyecto únicamente faltaría desplegarla junto al proyecto de GesTEC, y proceder a toda la parte de testeos unitarios de esta integración.

Como grandes lecciones finales y conocimientos adquiridos destacamos:

- **Tiempo de Formación**

El resultado final de la aplicación ha sido satisfactorio tanto para el cliente como de cara al desarrollo, la principal lección aprendida es que aparte del desarrollo del proyecto se debe estimar también el tiempo de formación necesario.

Este tiempo de usuario realmente es una inversión de tiempo ya que la elección de tecnologías y librerías puede significar en muchos casos el éxito o fracaso de los proyectos, y el propio ciclo de vida del producto verse afectado.

- **Establecer separación de responsabilidades y capas ahorra tiempo**

Aunque hacer una separación estilo DDD ha sido costosa en tiempo de desarrollo, formación y estudio, a la larga si la aplicación aumenta de tamaño ahorra tiempo. Por otro lado, no es sólo cuestión de mantenibilidad de código sino de gestión de errores. Separar responsabilidades permite identificar con mayor rapidez el origen de los errores y solventarlos.

- **Errores Imprevistos y Estimaciones**

A lo largo del proyecto he sido consciente que aquellas tecnologías que no se conocen se debe estimar un tiempo mayor ante posibles errores imprevistos.

Este tiempo se debe contar como tiempo de formación ya que es complicado hacer estimaciones sobre tecnologías que no se conocen.

- **Arquitectura de software**

Una buena arquitectura de capas es la clave para que la aplicación sea mantenible y ampliable en el futuro.

6.2. Conclusiones Más Personales

Toda la cantidad de tecnologías utilizadas en este trabajo son las nuevas tendencias que se utilizan en el desarrollo de aplicaciones con .NET, aunque algunas como el desarrollo de APIS con swagger, nswag etc. son transversales a otras plataformas y lenguajes.

En la carrera (obviando los lenguajes C# y el lenguaje SQL) no se ha podido adquirir ninguna noción de estas tecnologías.

La industria de la tecnología y el desarrollo web cambia de una forma vertiginosa y es prácticamente imposible actualizar los contenidos y tecnologías de un Grado a estas velocidades. Es por ello que he considerado la asignatura de Trabajo Fin de Grado el contexto adecuado para demostrar esta formación propia y exponer como con .NET se pueden realizar aplicaciones web nativas en distintos sistemas operativos y multiplataforma. Esta formación, y por ende, su manifestación en un proyecto real, ha sido altamente satisfactoria.

6.3. Mejoras Futuras

La aplicación de GesTEC está realizada en el framework de .NET framework.

Una vez implementada la aplicación para su uso en la empresa la primera mejora posible sería desplegar el servidor de BD de forma independiente y poder realizar una aplicación multiplataforma con .NET MAUI. Aunque se ha comentado que esta tecnología está en fase de validación sí que se espera que sea el futuro a corto plazo.

Otra posible mejora futura sería utilizar los métodos y funcionalidades de SharePoint para no solamente leer datos sino también realizar la gestión de eliminaciones/modificaciones etc.

Aunque se ha explicado el control de errores que existe en la aplicación, quizás una de las posibles mejoras sería ampliar la funcionalidad de la aplicación y establecer una mejora de errores y redirecciones (y no sólo bloqueo).

Finalmente, desacoplar el CSS del HTML, utilizando un editor de CSS más profesional sería también una ventaja en este proyecto de cara a ajustar resoluciones de distintas pantallas. Debido al tiempo invertido, el CSS no ha sido el aspecto de mayor interés.

7. Bibliografía

[1] T. Corey, «https://www.youtube.com/channel/UC-ptWR16ITQyYOglXyQmpzw,» 2022.

[2] Microsoft, «Blazor de ASP.NET Core,» 2022. [En línea]. Available: https://docs.microsoft.com/es-es/aspnet/core/blazor/?view=aspnetcore-6.0. [Último acceso: 2022].

[3] Microsoft, «Modelos de hospedaje Blazor en ASP.NET Core,» 2022. [En línea]. Available: https://docs.microsoft.com/es-es/aspnet/core/blazor/hosting-models?view=aspnetcore-6.0. [Último acceso: 2022].

[4] Microsoft, «Documentación de la API web de ASP.NET Core con Swagger/OpenAPI,» 2022. [En línea]. Available: https://docs.microsoft.com/es-es/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-6.0. [Último acceso: 2022].

[5] Microsoft, «Tutorial: Introducción con Entity Framework 6 Code First mediante MVC 5,» 2022. [En línea]. Available: https://docs.microsoft.com/es-es/aspnet/core/blazor/blazor-server-ef-core?view=aspnetcore-6.0. [Último acceso: 2022].

[6] J. Palermo, «https://jeffreypalermo.com/,» Julio 2009. [En línea]. Available: https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/. [Último acceso: 2022].

[7] N. Anil, T. Jain, D. Pine, Y. Victor, P. John y M. Wenzel, «Microsoft.com,» 2 Mayo 2022. [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice. [Último acceso: 2022].

[8] R. C. M. Bob), «https://blog.cleancoder.com/,» 13 Agosto 2012. [En línea]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. [Último acceso: 2022].

[9] Microsoft, «docs.microsoft.com,» 2022. [En línea]. Available: https://docs.microsoft.com/es-es/azure/architecture/patterns/cqrs. [Último acceso: 2022].