

E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Título del Trabajo Fin de Grado



Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor: Luis Azcoiti Peña

Director: Jesús Villadangos Alonso

Fecha de la defensa: Pamplona, 07/06/2024

upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Resumen

Este proyecto trata de la construcción del sistema de contratación temporal para justicia navarra. Este proyecto cuenta con requisitos funcionales agrupados en dos apartados o aplicaciones diferentes.

Uno es el portal de gestión, que permitirá a los técnicos de Gobierno de Navarra, según su perfil, elaborar bolsas, ofertas de plaza, ordenar listas y gestionar llamamiento.

Y el otro es el portal de teletramitación, que permitirá que los solicitantes se inscriban a una bolsa, a una oferta de plaza y responder a los llamamientos de manera telemática, así como gestionar sus datos personales como teléfono, estudios o idiomas.

Este proyecto hace uso de marcos de trabajo en el desarrollo de aplicaciones web modernas .Net Core, usado para el backend del sistema, angular como framework para el desarrollo del frontend y entity framework como ORM para simplificar el acceso a base de datos y su manipulación.

Lista de palabras clave

Visual Studio, .Net, .Net Core, ASP.Net Core, Angular, Entity framework, TypeScript, iTextSharp, SPA, SCRUM, Gobierno de Navarra, Justicia Navarra, Registra, CSV, PDF.

Índice

Resumen.....	1
Lista de palabras clave.....	1
1. Introducción y objetivos	5
1.1 Introducción	5
1.2 Objetivos	5
1.2.1 Optimización de los procesos de contratación	5
1.2.2 Facilitar la teletramitación.....	5
1.2.3 Centralización de la gestión de contratación	5
1.2.4 Mejora de la experiencia de usuario.....	6
1.2.5 Automatización de tareas	6
1.2.6 Incremento de la transparencia y trazabilidad.....	6
1.2.7 Adaptación a la transformación digital	6
2. Gestión del proyecto	6
2.1 Metodología Scrum	6
2.2 Horas totales del proyecto por proceso	7
2.3 Mi total de horas por proceso.....	7
3. Requisitos del sistema.....	8
3.1 Requisitos funcionales	8
3.1.1 Multi idioma	8
3.1.2 Registra	8
3.1.3 PDF	8
3.1.4 CSV	9
3.1.5 Almacenar apuntes en la base de datos	9
3.1.6 Envío de emails y SMSs	9
3.2 Requisitos no funcionales.....	9
3.2.1 Usabilidad.....	9
3.3 Diagrama de casos de uso.....	9
3.4 Requisitos Software.....	10
3.4.1 Tecnologías	10
3.4.2 Herramientas	16
4. Análisis y diseño	18
4.1 Base de datos.....	18
4.2 Diagramas.....	20
4.2.1 Diagrama de clases.....	20

4.2.2 Diagramas de flujo	21
4.3 Vertical slice	24
4.4 Patrón CQRS (Command query responsibility segregation)	24
4.5 Componentes.....	24
4.5.1 Proyectos CORE	24
4.5.2 Proyectos Backend, librerías y servicios.....	27
4.5.3 Proyectos Frontend.....	29
4.6 Modificaciones del proyecto	30
4.6.1 Multi idioma	30
4.6.1 Registra	31
4.6.2 Tabla apuntes.....	34
5. Implementación	35
5.1 Multi idioma.....	35
5.2 Registra	38
5.2.1 Alta_Apunte_Telematico.....	38
5.2.2 Adjuntar_Archivo	39
5.2.3 RGE_Actualizar_Hoja_Ruta_UN.....	40
5.2.4 RGE_Anular_Instanceia	40
5.3 PDF.....	41
5.4 CSV	42
5.5 Almacenar apuntes en la base de datos	43
5.6 Proceso de registro en Registra	45
6. Valoración y conclusión	46
6.1 Valoración.....	46
6.2 Conclusión	47
6.3 Líneas futuras	47
7. Bibliografía	47
7.1 Técnicas:.....	47
7.2 Herramientas:	48

Índice figuras

Figura 1 Tabla con las horas totales del proyecto por proceso.....	7
Figura 2 Tabla con mis horas por proceso.....	8
Figura 3 Diagrama de casos de uso	10
Figura 4 IDE visual studio.....	16
Figura 5 IDE SQL server management studio.....	17
Figura 6 Swagger.....	18
Figura 7 Tablas de la base de datos	19
Figura 8 Tablas Apuntes y AspirantesListas	19
Figura 9 Tabla ConfiguracionAvisos	20
Figura 10 Diagrama de clases.....	21
Figura 11 Diagrama de flujo de la creación y ordenación de una bolsa.....	22
Figura 12 Diagrama de flujo de la transformación de petición de contratación hasta el llamamiento	23
Figura 13 Diagrama de flujo del recorrido de los aspirantes por la aplicación	24
Figura 14 Proyecto API.CORE	25
Figura 15 Proyecto GATEWAY	26
Figura 16 Proyecto Application.....	27
Figura 17 Proyecto Infrastructure.....	28
Figura 18 Proyecto Persistence.....	28
Figura 19 Proyecto Domain.....	29
Figura 20 Proyectos Frontend.....	30
Figura 21 Carpeta i18n con los json de idiomas.....	31
Figura 22 Archivo app.module.ts	31
Figura 23 Campos de la clase Apunte	34
Figura 24 Tabla apuntes generada durante la migración.....	35
Figura 25 Constructor portal Teletramitación	36
Figura 26 Archivo home.component.html.....	37
Figura 27 Archivo es.json.....	37
Figura 28 Archivo eu.json	37
Figura 29 Menú home de teletramitación en español.....	38
Figura 30 Menú home de teletramitación en euskera.....	38
Figura 31 Método RegistrarApunte	39
Figura 32 Método AdjuntarArchivo	40
Figura 33 Método ActualizarHojaRuta	40
Figura 34 Método AnularInstancia.....	41
Figura 35 Administrador de paquetes NuGet donde se puede ver el paquete iTextSharp instalado en el proyecto	41
Figura 36 PDF generado mediante iTextSharp	42
Figura 37 Uso método FirmarPdf con CSV	42
Figura 38 Ejemplo firma CSV en PDF.....	43
Figura 39 Contenido del archivo ApunteConfiguration	43
Figura 40 Proyecto de inicio único API.Core.....	43
Figura 41 Donde encontrar la consola de administrador de paquetes en visual studio ..	44
Figura 42 Operación nuevo apunte controlador	44
Figura 43 Método nuevo apunte del archivo ApunteRepository.....	45
Figura 44 Operaciones que se pueden realizar de Apunte en el Swagger.....	45

1. Introducción y objetivos

1.1 Introducción

Nuevos tiempos, nuevas formas de trabajar y nuevas herramientas para hacerlo. El mundo está cambiando y por ende las organizaciones y personas que las integran. La búsqueda constante de la eficiencia y optimización nos plantean este reto. Gobierno de Navarra no es ajeno a estos cambios y a través de esta iniciativa de transformación digital va evolucionando hacia nuevos horizontes. El desarrollo de una nueva herramienta digital nos va a permitir optimizar la productividad, ahorrando tiempo y recursos.

Este trabajo de fin de grado es el desarrollo de una página web para el departamento de justicia del gobierno de Navarra, que busca de este modo automatizar la búsqueda de gente para cubrir bajas temporales en puestos del departamento de justicia, de manera que personas que se hayan presentado anteriormente a alguna oposición puedan optar a estos puestos, todo ello ordenado en función a parámetros como la nota que obtuvieron. Este proyecto cuenta con la necesidad de crear dos portales diferentes para que así los diferentes usuarios puedan usarlo.

De este modo se busca desarrollar dos portales diferentes, para que puedan por una parte los funcionarios que la gestionen, usar el portal de gestión y por otra parte contar con el portal teletramitación, para que estas personas que no obtuvieron ninguna plaza en las anteriores oposiciones puedan optar a un puesto temporal y así cubrir esas bajas.

1.2 Objetivos

1.2.1 Optimización de los procesos de contratación

Este proyecto busca mejorar y agilizar los procesos de contratación temporal en el Gobierno de Navarra, permitiendo a los gestores realizar tareas relacionadas con la elaboración de bolsas, ofertas de plaza, ordenación de listas y gestión de llamamientos de manera más eficiente.

1.2.2 Facilitar la teletramitación

Se pretende facilitar a los solicitantes la realización de trámites relacionados con la inscripción en bolsas, ofertas de plaza y respuesta a llamamientos de forma telemática a través del portal de teletramitación.

1.2.3 Centralización de la gestión de contratación

En este proyecto se busca centralizar la gestión de los procesos de contratación en una plataforma digital, lo que permitirá una mayor organización, control y seguimiento de las distintas etapas del proceso.

1.2.4 Mejora de la experiencia de usuario

Se busca mejorar la experiencia de los diferentes perfiles de usuario proporcionando interfaces intuitivas y funciones que faciliten la realización de tareas y la posibilidad de consultar información relevante.

1.2.5 Automatización de tareas

Se plantea la automatización de ciertas tareas repetitivas o diarias, como la validación de peticiones de contratación, lo que permitirá ahorrar tiempo y recursos para poder usarlos en otras actividades.

1.2.6 Incremento de la transparencia y trazabilidad

Esta plataforma digital permitirá mantener un registro detallado de todas las acciones realizadas en el proceso de contratación, lo que ayudará a aumentar la transparencia y trazabilidad de las acciones tomadas.

1.2.7 Adaptación a la transformación digital

Este proyecto busca seguir la tendencia de transformación digital en las organizaciones, permitiendo al Gobierno de Navarra adoptar nuevas formas de trabajo y herramientas tecnológicas más modernas y rápidas de usar que las tradicionales que usan hasta el momento.

2. Gestión del proyecto

2.1 Metodología Scrum

Esta es una de las metodologías más usadas últimamente en el desarrollo software, durante el desarrollo de este proyecto se han empleado algunas de las cosas que se suelen usar de esta metodología. La que más se ha usado ha sido la de las reuniones dailys y el desarrollo iterativo e incremental.

Cada 2 días se tenía una reunión con todo el equipo, donde se veía que había desarrollado cada uno, que problemas había tenido y se decidía que era lo siguiente que iba a estar haciendo hasta la siguiente reunión. Como digo solían ser 2 días después de la anterior reunión y se trataba de dar suficiente trabajo para estar desarrollando durante ese tiempo hasta la siguiente reunión.

De este modo cada pocos días se añadían nuevas funcionalidades a la aplicación y se revisaba que fuese correcto, además se iba presentando cada cierto tiempo al cliente que en este caso es el gobierno de Navarra los avances realizados para ser validados definitivamente y poder de ese modo en caso de que quisieran algún cambio hacerlo para así ir dejando acabadas esas funcionalidades.

2.2 Horas totales del proyecto por proceso

Proceso	Pendiente	Esfuerzo	Partes totales
Arquitectura de la solución	0h	400h	633,00h
Gestión	100h	500h	429,75h
Análisis funcional	100h	300h	278,50h
Portal de Gestión	400h	1000h	598,50h
Perfil Peticionario	0h	50h	11,00h
Portal de Teletramitación	750h	1000h	239,50h
Integración SAP	0h	40h	81,25h
Ateka	0h	30h	22,00h
Subidas a PRE	52h	100h	38,00h

Figura 1 Tabla con las horas totales del proyecto por proceso

Como se puede observar en la tabla hay algunos procesos que cuentan con 0h pendientes, lo que significa que ese proceso ya ha sido terminado, por lo que la arquitectura de la solución, el perfil peticionario, la integración con SAP y la integración con Ateka ya estarían acabados. También se puede ver que se ha avanzado más con el portal de gestión que con el de teletramitación, contando el de gestión con más de la mitad de las horas ya realizadas.

Algunos de los procesos como se puede ver han llevado más tiempo del esperado o ya han superado el tiempo esperado, contando todavía con algunas horas pendientes como es el caso de la arquitectura de la solución que llevó más de 200 horas más de lo esperado, o el del análisis funcional que ya lleva las 300 horas esperadas, pero aún le faltan otras 100 horas para ser completado. Por lo que podría haber un claro retraso respecto a lo esperado en algunos de los procesos, esto puede ser debido tanto a cambios que hubo por parte del cliente durante el desarrollo que en un principio no se habían avisado y también debido a posibles problemas durante el desarrollo.

2.3 Mi total de horas por proceso

Proceso	Realizado
Portal de Gestión	40h
Portal de Teletramitación	40h
Registra	75h
CSV	30h

Email	25h
SMS	2h
Reuniones daily	10h

Figura 2 Tabla con mis horas por proceso

En este caso lo que más me llevó con mucha diferencia fue la parte de Registra, ya que tiene muchos pasos que ir haciendo, mucha documentación que ir revisando para implementarlo, era mi primera vez implementando un servicio externo en un proyecto similar y en muchos casos había que dejarlo de lado al tener que esperar a que nos dieran de alta en cada parte necesaria y necesitar preguntar muchas veces como solucionar los diferentes problemas que se fueron encontrando durante su implementación. Por otra parte, implementar SMS era muy sencillo al usar lo mismo que email, pero cambiando unas pocas cosas y las reuniones daily llevaron en total también muchas horas, al durar aproximadamente media hora todos los días que se hacen y tener entre 2 y 3 cada semana.

3. Requisitos del sistema

3.1 Requisitos funcionales

3.1.1 Multi idioma

Al ser un proyecto realizado para el Gobierno de Navarra este requiere que esté tanto en castellano como en euskera, por lo que me pidieron que añadiera al portal de Teletramitación la opción de multi idioma, que permite de manera sencilla e intuitiva para el usuario cambiar entre estos idiomas, simplemente pulsando el botón es para español o el botón eu para el euskera. De este modo los usuarios de Teletramitación tienen la opción de ver la página con el idioma que ellos prefieran, viniendo por defecto en castellano.

3.1.2 Registra

Este es un servicio externo del gobierno de Navarra, que nos pidieron implementar para que cuando uno de los usuarios de Teletramitación se apuntara a alguna de las ofertas disponibles, quedara constancia de ello. Para esto cuentan con este servicio que era necesario añadir en el proyecto, concretamente en la carpeta de ExternalServices. Este servicio cuenta con una gran cantidad de clases y métodos, pero nosotros necesitábamos únicamente usar 3 de todos esos métodos para que funcionara correctamente.

3.1.3 PDF

El proyecto requiere de generar PDFs de manera dinámica y personalizada, para registrar las información de las personas que necesitan realizar un apunte. Esto se consigue mediante el uso de la librería iTextSharp, que permite crear PDFs al vuelo y personalizar el contenido mediante código C#.

3.1.4 CSV

CSV es un código seguro de verificación que se usa para comprobar por internet que el documento que estamos viendo es el real y no ha sido modificado. Actualmente este es una API Rest que cuenta con 4 métodos, entre los cuales está el que necesitaba usar que es la FirmaCSV y otros 3 que entre otros sirven para obtener el documento original mediante ese código CSV, otro que sirve para firmar documentos que tengan un tamaño mayor de 4mb de memoria y por último uno que permite actualizar la situación de documento, haciéndolo accesible o no.

En nuestro caso al contar con PDF de un tamaño muy pequeño que nunca llegará a superar siquiera 1 MB de tamaño usaremos la firma PAdES que permite firmar un archivo con un máximo de 4MB de tamaño, mientras que la otra firma mediante XAdES no tiene límite alguno.

3.1.5 Almacenar apuntes en la base de datos

Es necesaria la creación de una tabla en la base de datos que sea apunte y de ese modo poder almacenar los apuntes creados, junto con su código CSV correspondiente que se haya usado para firmar el PDF. De este modo hace falta hacer una migración para que la tabla se cree en la base de datos mediante Entity Framework.

3.1.6 Envío de emails y SMSs

Este es un servicio del Gobierno de Navarra que permite necesita ser implementado en el proyecto, para poder realizar envío de emails y SMSs personalizados en caso de ser necesarios, en nuestro caso para avisar a los usuarios de ciertos eventos importantes.

3.2 Requisitos no funcionales

3.2.1 Usabilidad

Uno de los requisitos no funcionales más importantes es el de la usabilidad, que permite que la aplicación sea más sencilla o intuitiva para los usuarios finales de la web. En mi caso tuve que modificar algunos de los botones para que todos usaran los mismo nombres, de modo que fuese más intuitivo para los usuarios o modificaciones visuales simples, así como añadir una página de inicio al portal de teletramitación que fuese más atractivo.

3.3 Diagrama de casos de uso

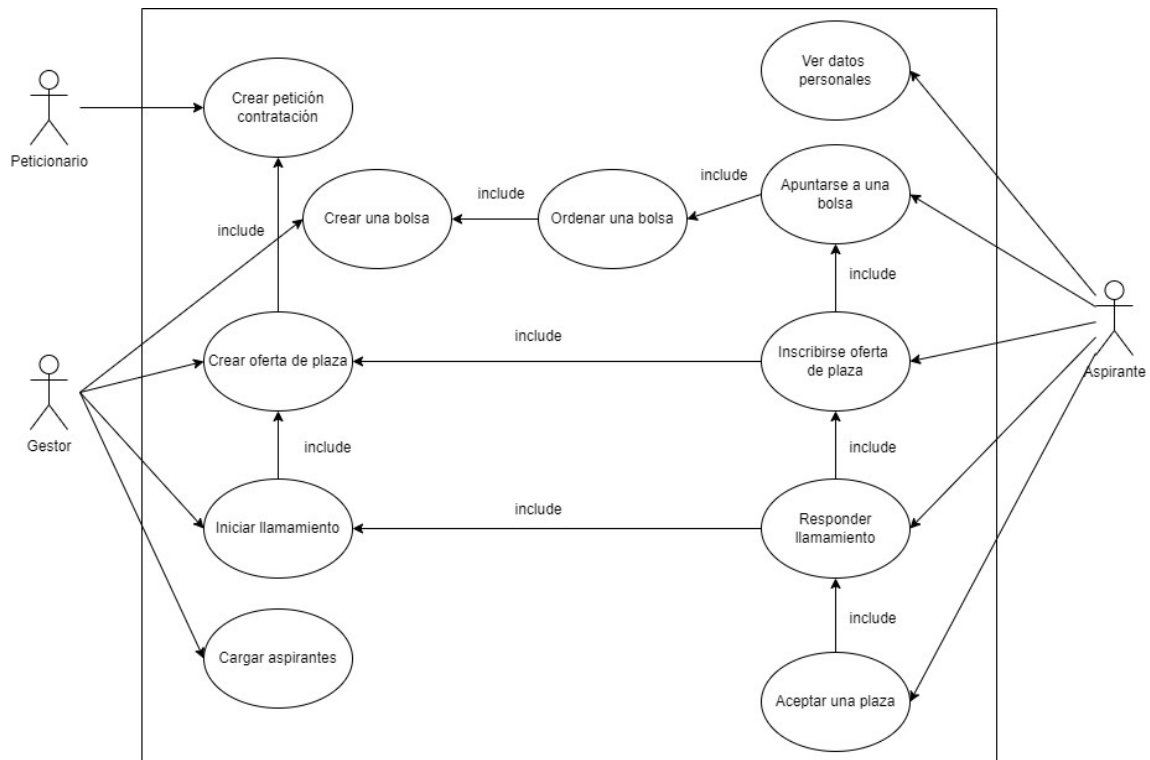


Figura 3 Diagrama de casos de uso

3.4 Requisitos Software

3.4.1 Tecnologías

Para el desarrollo de este proyecto se ha hecho uso de varias tecnologías, como marco de trabajo se ha empleado el uso de .Net Core que proporciona una base sólida para el backend del sistema, permitiendo una arquitectura eficiente. Angular destaca como framework de desarrollo frontend con TypeScript, permitiendo el desarrollo de interfaces de usuario dinámicas. Entity Framework como ORM (Mapeador de Relación-Objeto), para simplificar el acceso a datos y la manipulación de la base de datos. Estas tecnologías combinadas ofrecen un entorno de desarrollo moderno que permite el desarrollo del sistema de contratación temporal para Justicia Navarra.

3.4.1.1 .Net

Es una plataforma de aplicaciones gratuita y de código abierto respaldada por Microsoft para compilar muchos tipos de aplicaciones diferentes. C# es el lenguaje de programación para .Net, este está fuertemente tipado y tiene seguridad de tipos, siendo el lenguaje más comúnmente usado con esta plataforma, pero también es compatible con otros lenguajes como Visual Basic.NET, F# y los más recientes Python Y JavaScript.

.Net inicialmente fue asociado con el sistema Windows, pero tras la introducción de .Net Core paso a ser multiplataforma, contando con compatibilidad con varios sistemas operativos, concretamente IOs, Android, Linux y Windows. Este cuenta con el administrador de paquetes NuGet que cuenta con más de 300.000 paquetes. Este administrador permite gestionar las dependencias de los proyectos y permite también a los desarrolladores crear, compartir y consumir paquetes de manera muy eficiente.

Permite la compilación de muchos tipos de aplicaciones, como por ejemplo aplicaciones móvil, de escritorio, de aprendizaje automático e incluso para desarrollar juegos. Cuenta con una gran comunidad y fomenta el desarrollo abierto y la colaboración en torno a su ecosistema, cuenta con más de 100.000 contribuidores y más de 3.700 empresas que contribuyen a la plataforma.

Se puede compilar de varias maneras, con la CLI de .Net, Visual Studio y Visual Studio Code que son los IDE de Microsoft, además de otras herramientas. También es rápido, ofreciendo mejores tiempos de respuesta y requiriendo menos potencia de procesamiento que otros como Java Servlet o Node.js.

En resumen, .Net es una plataforma versátil y robusta que ofrece una amplia gama de herramientas y tecnologías para el desarrollo de aplicaciones de diferentes tipos, web y móviles entre otros y cuenta con una gran comunidad que está en constante crecimiento y que puede ser muy atractivo al contar con código abierto y el apoyo y soporte de Windows siendo esta una de las empresas más importantes del mundo. [1]

3.4.1.2 .Net core

.Net core es la plataforma de desarrollo de Microsoft más moderna y de código abierto. Esta también es multiplataforma y de alto rendimiento para la creación de aplicaciones. Esta arquitectura modular permite se pueda incluir lo necesario de las aplicaciones que se quieran desarrollar, haciendo que tenga un menor peso a la hora de desplegarlas y cuentan con una manera más sencilla de actualización a través de los paquetes NuGet. Este es multiplataforma, por lo que no depende de cosas específicas de cada sistema operativo, como si sucedía con su predecesor el .Net framework, que solo se podía usar con el sistema operativo Windows. Además, .Net framework se actualiza a través de Windows Update, mientras que .Net core se puede actualizar fácilmente mediante el sistema de paquetes NuGet.

Las principales características de .Net core son:

- Es multiplataforma, pudiendo usarse tanto si se cuenta con un sistema operativo macOS, Windows o Linux y además, cuenta con soporte para su uso con contenedores Docker.
- Cuenta con asincronía con el uso de `async` y `await`, que es una de las principales características de primera clase del lenguaje de programación C#.
- Alto rendimiento, fue diseñado desde cero, dándole una gran importancia a esta característica.

Las ventajas de usar .Net core en comparación a otras tecnologías de desarrollo software son varias, la primera sería el hecho de que es multiplataforma, permitiendo de este modo que se ejecute en una gran variedad de sistemas operativos, proporcionando una gran flexibilidad y alcance en el despliegue de las aplicaciones que lo emplean. También cuenta con un gran rendimiento, ya que fue diseñado desde cero con un enfoque en el rendimiento, ofreciendo un arranque más rápido y un menor uso de memoria que anteriores versiones de .Net.

Las desventajas de usar .Net core pueden ser por ejemplo que es relativamente nuevo, por lo que puede haber menos recursos e información sobre esta tecnología que otras que llevan más años, además de que puede que al tener menos años que otras no tenga compatibilidad con demasiadas bibliotecas o herramientas de terceros y por último, puede que versiones anteriores de proyectos que empleen .Net requieran de modificaciones para ser migradas a .Net core.

En resumen, a pesar de que .Net core cuenta con numerosas ventajas respecto al rendimiento y desarrollo moderno de aplicaciones, así como el hecho de ser multiplataforma, también puede haber ciertos retos asociados a su adopción, puesto que es relativamente nueva y puede no ser compatible con herramientas y bibliotecas de terceros. [2]

3.4.1.3 ASP.Net core

Este es un marco de trabajo que permite la creación de aplicaciones y servicios web con .Net y C#. Este es desarrollado por Microsoft, siendo gratis, multiplataforma (compatible con macOS, Windows y Linux) y de código abierto. ASP.Net amplía la plataforma .Net, con herramientas y bibliotecas específicas para compilar aplicaciones web.

Sus principales características son que es multiplataforma, de código abierto, ligero y modular, enfocado en el rendimiento, cuenta con soporte para contenedores y ofrece una API de alto rendimiento. Todas estas características son también las ventajas con las que cuenta.

Las posibles desventajas son la compatibilidad con bibliotecas existentes, que a pesar de que es compatible con muchas herramientas y bibliotecas, no pueden ser totalmente compatibles por diferencias en la arquitectura y la API. La otra desventaja es la curva de aprendizaje, pudiendo ser difícil adaptarse a esta tecnología para aquellos desarrolladores que no estén familiarizados con C# o el propio ecosistema de .Net.

En resumen, es un marco de trabajo que ofrece multitud de ventajas y que si se cuenta con cierta experiencia puede ser muy buena elección a la hora de emplearse en proyectos ligeros, que busquen un alto rendimiento para la construcción de aplicaciones web modernas y escalables. [3]

3.4.1.4 TypeScript

TypeScript es un lenguaje de programación de código abierto desarrollado por Microsoft. TypeScript ofrece todas las características de JavaScript y una capa adicional,

el sistema de tipos de TypeScript. El principal beneficio de TypeScript es que puede resaltar un comportamiento inesperado en su código, reduciendo de ese modo el número de posibles errores. Es usado para el desarrollo de aplicaciones JavaScript del lado del cliente o del servidor. Además, que cualquier código escrito en JavaScript debería funcionar en TypeScript.

Las ventajas de TypeScript son entre otras, que cuenta con compatibilidad con JavaScript, facilitando su adopción en proyectos existentes, así como minimizar los errores de tipo durante la fase de desarrollo al ser capaz de detectarlos e integrar herramientas modernas de desarrollo y una gran cantidad de bibliotecas y marcos de trabajo.

Las desventajas pueden ser la curva de aprendizaje inicial, especialmente en desarrolladores web nuevos o que sean nuevos con JavaScript o TypeScript. En proyectos pequeños puede aumentar el tiempo de desarrollo al necesitar un proceso de compilación adicional a diferencia de JavaScript y aumentar la dificultad al ser necesario definir tipos adicionales.

En resumen, es un lenguaje de programación que ofrece todas las características de JavaScript, mejorando la seguridad y claridad del código, permitiendo una menor cantidad de errores durante el desarrollo pero que puede no ser adecuado para proyectos pequeños, en los que complicaría su desarrollo. [4]

3.4.1.5 Angular

Angular es una plataforma de desarrollo construida en TypeScript, como plataforma incluye un framework basado en componentes para construir aplicaciones web escalables. Una colección de bibliotecas integradas que cubren una gran variedad de características, como el manejo de formularios, comunicación cliente servidor y más. Además, cuenta con una variedad de herramientas de desarrollo para ayudar en el desarrollo, construcción, test y actualización del código.

Con Angular tienes la ventaja de una plataforma que se puede escalar desde un proyecto desarrollado por una sola persona, hasta una desarrollada por una gran empresa. Lo mejor es que el ecosistema de Angular costa de alrededor de 1.7 millones de desarrolladores, creadores de bibliotecas y creadores de contenido. Angular cuenta con ciertos requisitos, los cuáles son muy comunes en frameworks modernos, esperando que los desarrolladores estén familiarizados con HTML, CSS y JavaScript, también es recomendado estar familiarizado con conceptos como las clases de JavaScript y los fundamentos de TypeScript. [5]

3.4.1.6 Entity framework

Es marco de trabajo de mapeo objeto-relacional (ORM) desarrollado por Microsoft para el ecosistema .Net. Su principal propósito es el de facilitar el acceso y manipulación de datos almacenados en una base de datos relacional utilizando un enfoque orientado a objetos. Permitiendo a los desarrolladores usar la base de datos utilizando objetos y clases en lugar de consultas SQL directas.

Sus características son el mapeo de objetos relacionales (ORM), mapeando las clases de objetos del modelo de la aplicación a las tablas de la base de datos. Es compatible con diferentes proveedores de bases de datos como MySQL, SQLite y otros más. Cuenta con un mecanismo de caché que permite mejorar el rendimiento de la aplicación al almacenar en caché resultados de consultas frecuentes. Y permite realizar consultas a la base de datos mediante una característica de C# que tiene una sintaxis similar a SQL, pero integrado en el propio lenguaje.

Las ventajas son que aumenta la productividad al simplificar el acceso a los datos de la base de datos y, además, reduce la cantidad de código necesario para poder interactuar con los datos. De este modo también impide tener ciertas vulnerabilidades como pueden ser las inyecciones SQL al generar las consultas parametrizadas de manera automática. Entity Framework está integrado con Visual Studio, por lo que facilita el desarrollo y la depuración de aplicaciones. Por último, permite trabajar con objetos y clases en lugar de tener que hacerlo mediante consultas SQL, lo cual facilita mucho el mantenimiento y refactorización del código.

Algunas de sus desventajas pueden ser que puede estar algo limitado a la hora de realizar consultas que requieran de una mayor personalización o mayor control con la base de datos en comparación con las consultas SQL. También puede llevar tiempo aprender a dominarlo, especialmente para aquellos desarrolladores que sean nuevos en ORM.

En resumen, es un marco de trabajo creado para simplificar el acceso y manipulación de los datos de la base de datos, aumentando su seguridad. Requiriendo de algo de tiempo de aprendizaje en caso de no estar acostumbrado y pudiendo ser no demasiado flexible a la hora de realizar algunas consultas más específicas. [6]

3.4.1.7 Dapper

Dapper es un micro-ORM (mapeo de objeto relacional), una herramienta que permite convertir los objetos en un formato propicio para ser guardados en base de datos sin necesidad de emplear consultas SQL, haciendo esto mediante los objetos ya creados. Las acciones permitidas por Dapper son las de crear, leer, actualizar y borrar (CRUD). Este fue desarrollado por el equipo de Stack Overflow.

Una de las principales características es que es muy ligero, siendo muy recomendado para facilitar la programación de los desarrolladores. Otra es la velocidad, ya que se consigue escribir la consulta SQL de manera manual, por lo que es escrito de manera más rápida que con Entity Framework. Además, es bastante fácil de usar al estar enfocado para desarrolladores. Este funciona con una gran cantidad de bases de datos entre ellas MySQL, Oracle o SQL server. Es fácil de usar por medio de mapeo automático de los objetos C#. [7]

3.4.1.8 Comparación entre Entity Framework y Dapper

Ambas herramientas son ORM que simplifican el acceso y manipulación de datos de la base de datos en el ecosistema .Net. También son compatibles con una gran cantidad de bases de datos, haciendo que sean adecuadas para mucho proyecto y escenarios

diferentes. Ambos son utilizados por muchos desarrolladores y cuentan con una gran comunidad y recursos en línea.

En resumen, Dapper es bastante ligero y veloz, contando con bastante flexibilidad y un fácil uso para aquellos familiarizados con SQL. Por otro lado, Entity Framework ofrece más abstracción y productividad y una pequeña curva de aprendizaje. Ambos se pueden usar en Visual Studio, contando con integración de Entity Framework, pero siendo también sencillo instalar Dapper mediante el gestor de paquetes de Visual Studio llamado NuGet. La elección entre estas herramientas dependerá de cuales sean las necesidades específicas del proyecto que se vaya a realizar y el conocimiento previo que se tenga de estas, así como el rendimiento de la base de datos.

3.4.1.9 CoreUI

CoreUI es un framework de interfaz de usuario de código abierto que cuenta con un conjunto completo de componentes y estilos predefinidos para la construcción rápida de interfaces de usuario modernas y atractivas en aplicaciones web. Está diseñado para ser configurable y adaptable a una gran variedad de proyectos, integrándose fácilmente con otros frameworks populares como Bootstrap, Angular, React entre otros.

Características:

- **Diseño responsivo:** Los componentes están diseñados para adaptarse y funcionar correctamente en una gran variedad de dispositivos y tamaños de pantalla.
- **Integración con frameworks populares:** Se integra fácilmente con frameworks de desarrollo web populares como Angular, React y Bootstrap, lo que facilita su incorporación en proyectos existentes o en nuevos proyectos.
- **Componentes personalizables:** Ofrece una amplia gama de componentes de interfaz de usuario como botones, formularios y tablas entre otros, que se pueden personalizar fácilmente para que se ajusten a las necesidades específicas de cada proyecto.

La gran mayoría de sus ventajas tienen que ver con sus características, por ejemplo, la compatibilidad con otros frameworks, haciéndolo muy versátil a la hora de integrarlo en los proyectos. La personalización que permite que los desarrolladores adapten fácilmente el aspecto y la funcionalidad de las interfaces de usuario de cada proyecto. Otra ventaja es que cuenta con componentes y estilos predefinidos, que facilitan el desarrollo de las interfaces de usuario y por último cuenta con una documentación muy completa, contando con ejemplos que facilitan entender cómo se usa.

Algunas de las desventajas es que los paquetes dependiendo de las características que se utilicen pueden tener un gran tamaño, lo que podría afectar al rendimiento de la aplicación, en especial en los dispositivos con recursos limitados. Otra desventaja es la dependencia de frameworks de terceros, en este caso con Bootstrap, por lo que si hay cambios en Bootstrap podría necesitar ciertos ajustes de código de la aplicación. Además, requiere de algo de tiempo para familiarizarse con sus características, por lo que cuenta con una pequeña curva de aprendizaje inicial.

En resumen, CoreUI cuenta con varias ventajas y desventajas, que habría que valorar antes de implementar en un proyecto, pero para aplicaciones web cuenta con una mayor cantidad de ventajas, por lo que puede ser muy útil para crear una interfaz gráfica de una manera rápida, versátil y personalizada.

3.4.2 Herramientas

3.4.2.1 Visual Studio

Visual Studio es un IDE de desarrollo que permite editar, depurar y compilar código. También incluye compiladores, herramientas de completado de código, diseñadores gráficos y más funciones para mejorar el desarrollo de software. Un IDE es un entorno de desarrollo integrado, que de manera resumida es un kit de herramientas que un programador usa para escribir, compilar y gestionar código de manera eficiente. En este caso es un IDE desarrollado por Microsoft. Visual Studio permite el desarrollo software con diversos lenguajes de programación, incluyendo C#, C++, JavaScript, TypeScript, Python y otros muchos.

Cuenta con una gran comunidad de usuarios y una gran cantidad de documentación en línea, así como soporte de Microsoft y recursos de aprendizaje. En resumen, es un IDE muy completo que ofrece una gran cantidad de herramientas y características para mejorar el desarrollo de software. [8]

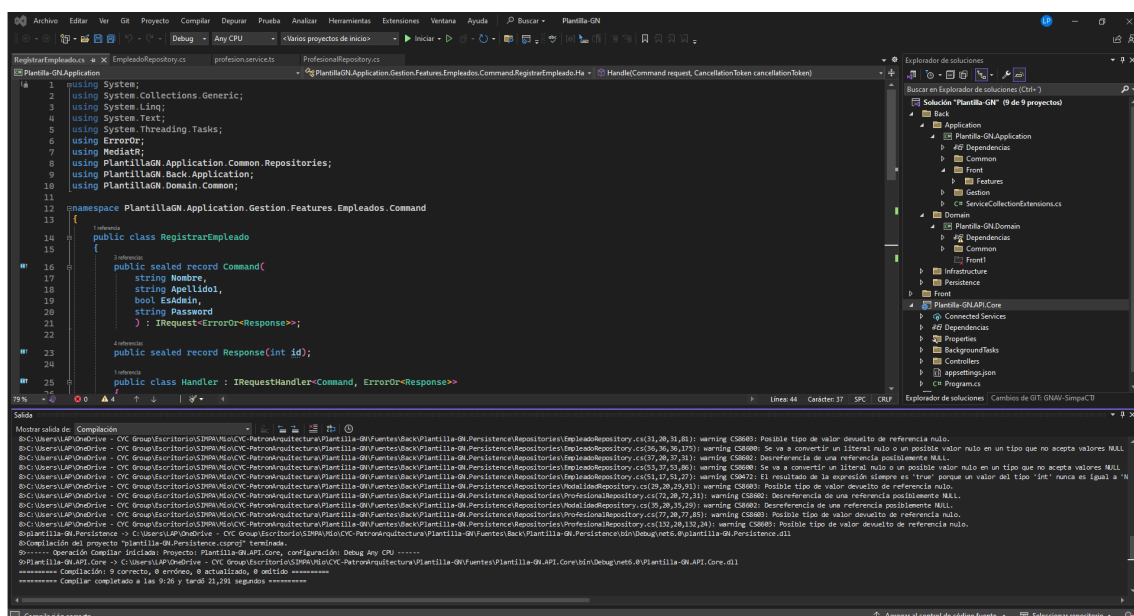


Figura 4 IDE visual studio

3.4.2.2 SQL server management studio

Es un entorno de desarrollo integrado para administrar cualquier infraestructura de SQL. Se utiliza para acceder a todos los componentes de SQL Server y SQL Database, así como para configurarlos, administrarlos y desarrollarlos. Este ha sido desarrollado por

Microsoft durante años siendo un programa de administración de servidores y bases de datos. Este no es multiplataforma, solo se puede ejecutar en Windows, pero no importa el procesador con el que se cuente (AMD o Intel), como alternativa para otras plataformas existe Azure Data Studio que si permite ejecutarlo en macOS, Linux y Windows.

Algunas de sus características son la posibilidad de crear y modificar bases de datos, agregar y modificar objetos de la base de datos, incluyendo tablas y vistas. También sirve para importar y exportar datos, así como implementar bases de datos. Cuenta con varios componentes que permiten, por ejemplo, usar herramientas de diseño visual, para compilar consultas, tablas y bases de datos de diagrama. [9]

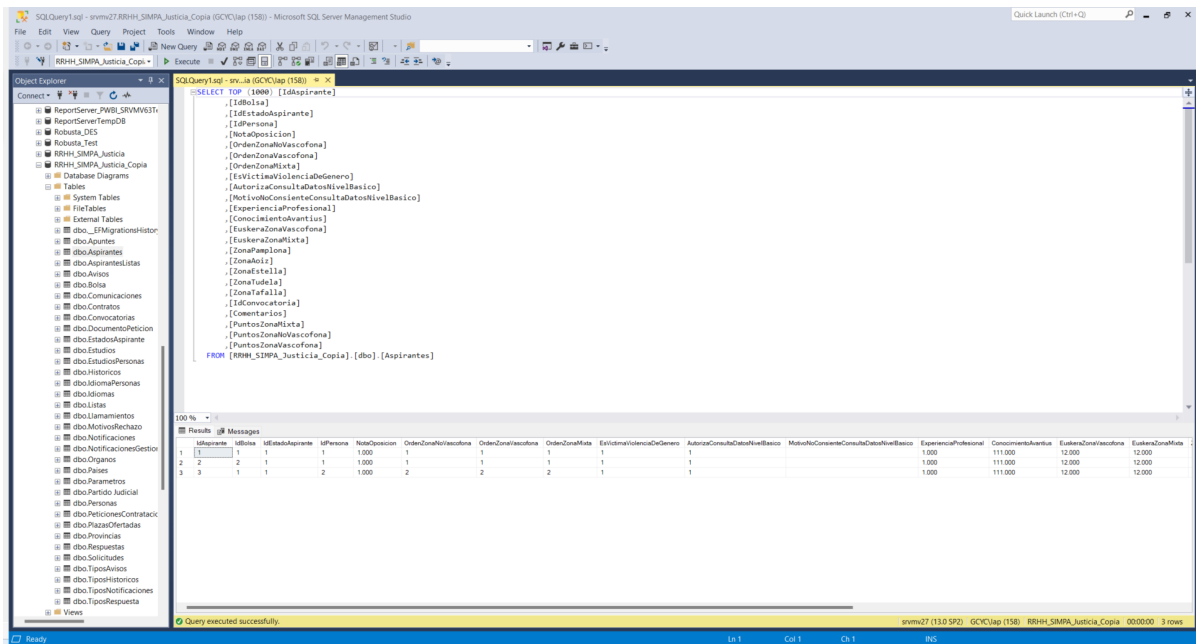


Figura 5 IDE SQL server management studio

3.4.2.3 Swagger

Es un potente pero fácil de usar conjunto de herramientas para desarrolladores de Api, tanto para equipos como para individuos, que permite el desarrollo en todo el ciclo de vida de la API, desde el diseño y la documentación hasta la prueba y la implementación. Swagger está formado por herramientas de código abierto, gratuitas y disponibles comercialmente, que permiten a cualquier persona, desde ingenieros hasta gerentes de producto, de construir Apis. Este fue desarrollado por SmartBear Software. Líder en herramientas de calidad de software para equipos.

Características que destacar:

- Interfaz de usuario intuitiva: Swagger proporciona una interfaz intuitiva y fácil de usar, que permite a los desarrolladores visualizar, probar e interactuar con las APIs de forma dinámica. Esta muestra de manera automática la documentación, facilitando a los desarrolladores entender rápidamente cómo utilizar la API.
- Pruebas de APIs integradas: Incluye herramientas que permiten probar las APIs directamente desde su interfaz de usuario, Swagger UI. De este modo los

desarrolladores pueden enviar solicitudes HTTP, ver las respuestas y validar el comportamiento de la API, facilitando el desarrollo.

- OpenAPI Specification: Utiliza la especificación de OpenAPI para describir la estructura y el comportamiento de las APIs de forma estándar y legible para humanos y máquinas.

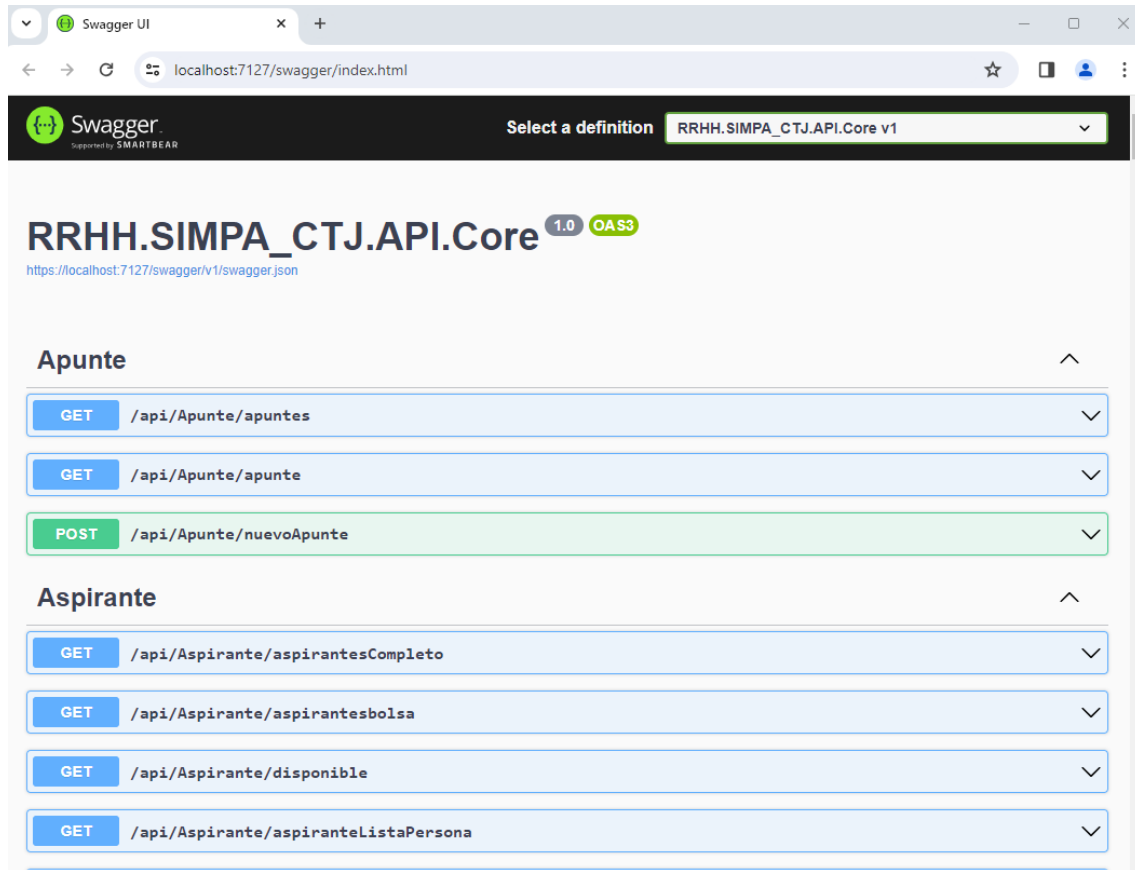


Figura 6 Swagger

4. Análisis y diseño

4.1 Base de datos

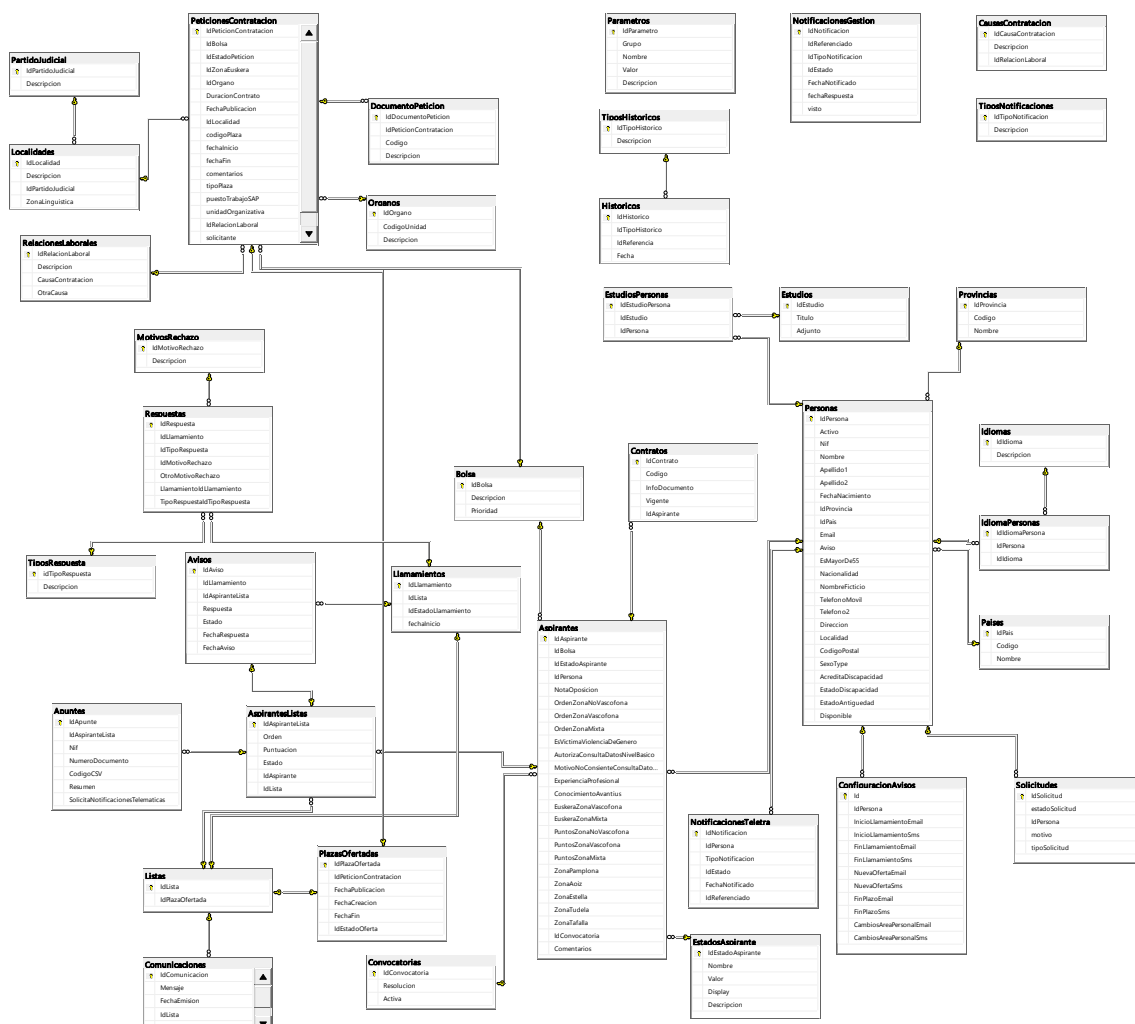


Figura 7 Tablas de la base de datos

Estas tablas son las que se usan en el proyecto, pero las que he creado yo son la tabla Apuntes y ConfiguracionAvisos.

La tabla Apunte es usada para almacenar en la base de datos todos los apuntes que se generan al apuntarse algún aspirante a alguna de las ofertas disponibles. Por esa razón está relacionada con la tabla AspirantesListas, quedando relacionada con el Aspirante que se apuntó y a que Oferta.

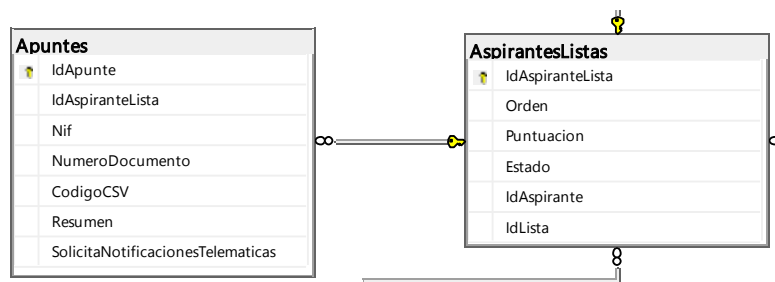


Figura 8 Tablas Apuntes y AspirantesListas

También he creado la tabla ConfiguracionAvisos que es necesaria para saber cómo desea recibir los avisos cada persona. En esta tabla se almacena tanto el Id de la persona, como 10 booleanos que sirven para saber si desea recibir o no los avisos por email o SMS. De este modo si al iniciar un llamamiento tiene activados los avisos por email, pero no los que son por SMS, solo recibiría el aviso por email. Esto se puede configurar tanto para inicio como fin de llamamiento, al haber una nueva oferta, cuando se termine el plazo de una oferta y cuando haya algún cambio en sus datos personales por parte de un gestor.

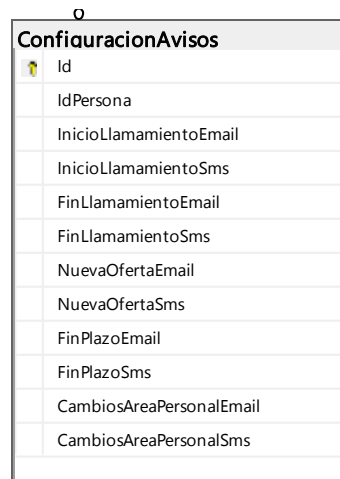


Diagrama de la tabla ConfiguracionAvisos. La tabla tiene un encabezado con el título 'ConfiguracionAvisos' y un ícono de llave. El cuerpo de la tabla contiene 13 filas, cada una con un campo de texto. El primer campo es 'Id', que tiene un ícono de llave a su izquierda. Los campos restantes son: 'IdPersona', 'InicioLlamamientoEmail', 'InicioLlamamientoSms', 'FinLlamamientoEmail', 'FinLlamamientoSms', 'NuevaOfertaEmail', 'NuevaOfertaSms', 'FinPlazoEmail', 'FinPlazoSms', 'CambiosAreaPersonalEmail', y 'CambiosAreaPersonalSms'.

ConfiguracionAvisos	
Id	
IdPersona	
InicioLlamamientoEmail	
InicioLlamamientoSms	
FinLlamamientoEmail	
FinLlamamientoSms	
NuevaOfertaEmail	
NuevaOfertaSms	
FinPlazoEmail	
FinPlazoSms	
CambiosAreaPersonalEmail	
CambiosAreaPersonalSms	

Figura 9 Tabla ConfiguracionAvisos

4.2 Diagramas

4.2.1 Diagrama de clases

En el diagrama de clases me voy a centrar en las 2 clases que he creado yo, que en este caso son la clase Apunte y la clase ConfiguracionAviso. La tabla Apunte se usa para almacenar los apuntes de cada aspirante en una lista en concreto, por lo que está relacionada con la clase AspirantesListas. Por otro lado, la clase ConfiguracionAviso se usa para guardar la configuración de cada persona de todos los avisos, que en este caso son 10 avisos diferentes, al ser 5 avisos de cada tipo y existir la posibilidad de recibir avisos por email o SMS, según se tenga configurado.

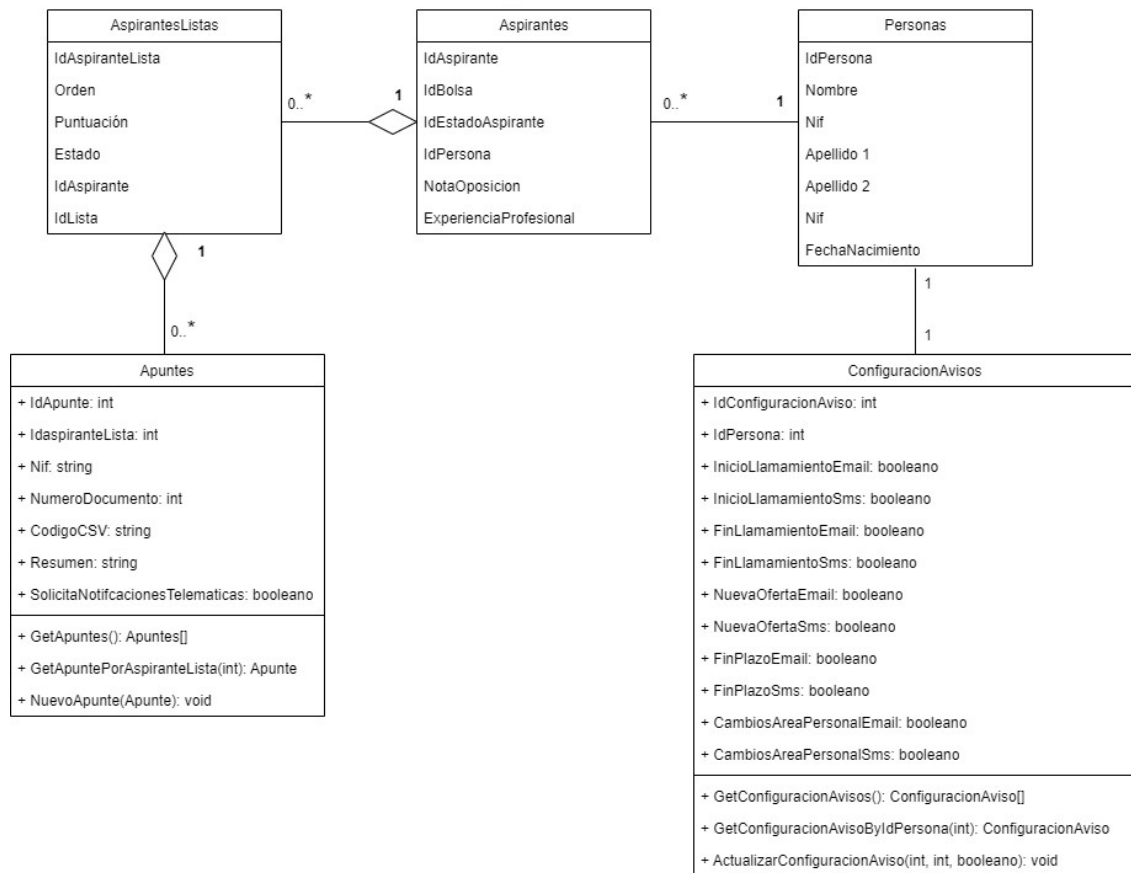


Figura 10 Diagrama de clases

4.2.2 Diagramas de flujo

Creación y ordenación de una bolsa

En este proceso intervienen el perfil Gestor que es quién la crea y la propia aplicación que es quien de manera automática recupera el orden vigente.

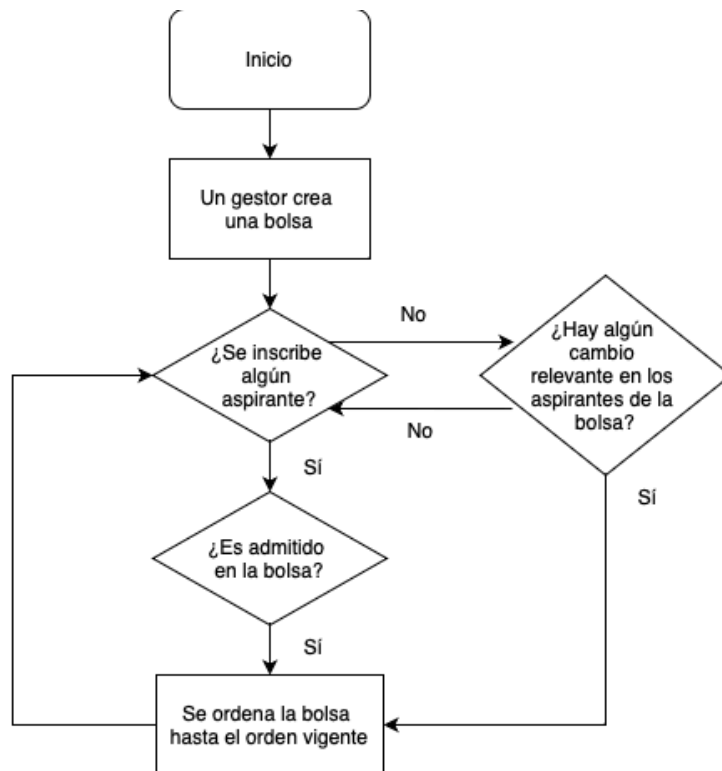


Figura 11 Diagrama de flujo de la creación y ordenación de una bolsa

Transformación de una petición de contratación hasta el llamamiento

En este flujo intervendrá de manera directa el perfil peticionario y el perfil gestor. Se puede ver la evolución de la petición de contratación en una oferta de plaza para después construir una lista sobre la que se inicia un llamamiento que termina con un aspirante seleccionado.

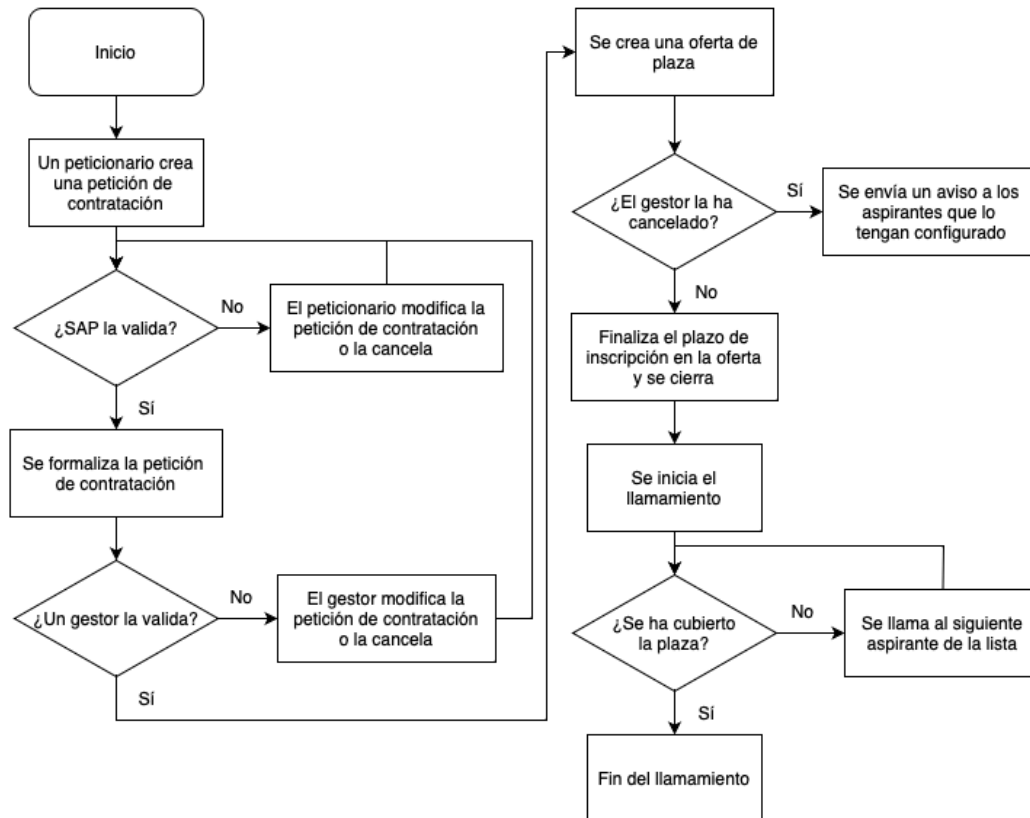


Figura 12 Diagrama de flujo de la transformación de petición de contratación hasta el llamamiento

Recorrido de los aspirantes por la aplicación

Siendo el perfil aspirante el protagonista, se recorren desde el acceso al portal de teletramitación de el mismo pasando por todas las acciones que puede realizar dentro del proceso de contratación temporal.

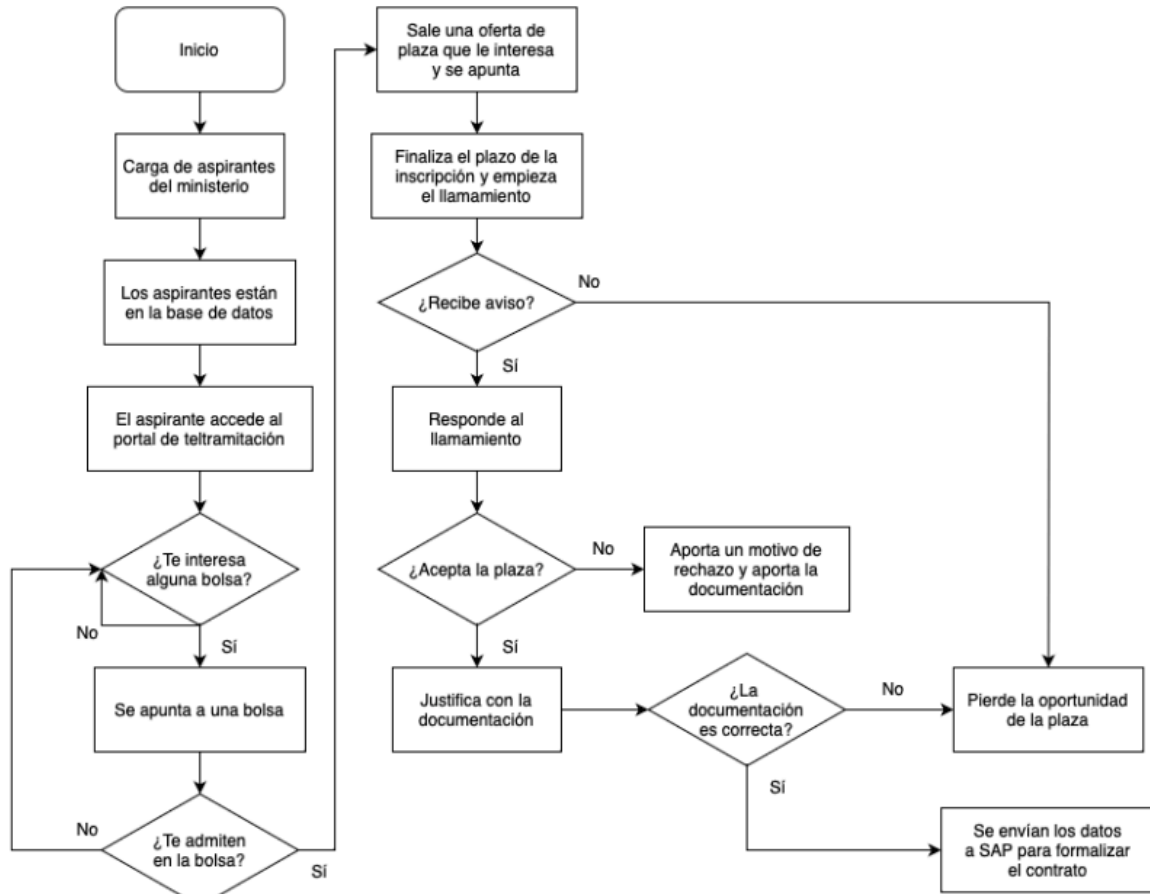


Figura 13 Diagrama de flujo del recorrido de los aspirantes por la aplicación

4.3 Vertical slice

Se centra en la implementación de características de manera aislada, dividiendo la aplicación de manera vertical. De este modo en cada separación vertical incluye los componentes necesarios para implementar una funcionalidad específica.

4.4 Patrón CQRS (Command query responsibility segregation)

Este patrón separa las operaciones de lectura (queries) y escritura (command) en dos modelos diferentes, de manera que optimiza cada uno para sus responsabilidades específicas. Este trae muchas ventajas, ya que permite aumentar la escalabilidad de manera independiente

4.5 Componentes

4.5.1 Proyectos CORE

Arquitectura con API Backend

Independientemente del número de frontales que tenga la aplicación, todos serán gestionados por la siguiente estructura en ejecución de manera única.

Proyecto API.CORE

Este proyecto gestiona la configuración de las tareas principales de backend. Debe tener dependencias a los proyectos de librerías de clases de backend.

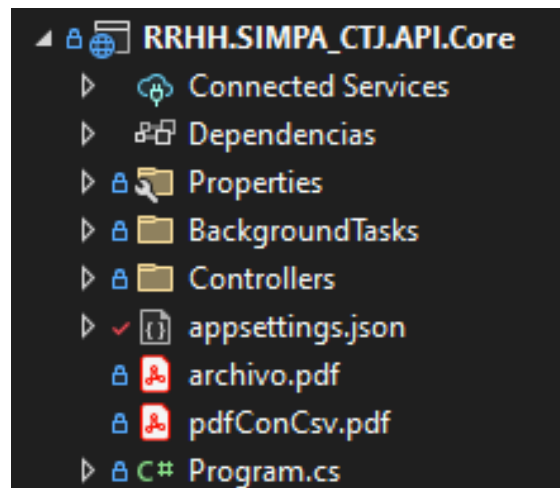


Figura 14 Proyecto API.CORE

Elementos clave:

- Program.cs: Configuración de servicios generales de la aplicación como: controladores, autenticación, loggers, CORS, Swagger, Hangfire, etc.
- Appsettings.json: Almacenar configuraciones generales de la aplicación, por ejemplo, los strings de conexión a BBDD (uno por entorno).
- Launchsettings.json: Almacena configuraciones de ejecución, por ejemplo, las URL.
- Carpeta “Controllers”: Carpeta donde se encuentran los controladores.
- Carpeta “BackgroundTasks” *: Carpeta para almacenar las clases con los métodos que se ejecutan en segundo plano de manera independiente o temporal.
- Carpeta “Services” *: Carpeta para almacenar servicios de gestión propios, por ejemplo, controles de acceso personalizados, utilizados por el “program.cs” para proteger las llamadas a los controladores.

Proyecto GATEWAY *:

Proyecto que actúa como intermediario o puerta de enlace entre los servicios de una API y los clientes que la usan. Gestiona las solicitudes de los clientes, realiza la lógica

de autorización y autenticación si es necesario y luego redirige las solicitudes a los servicios de la API correspondientes.

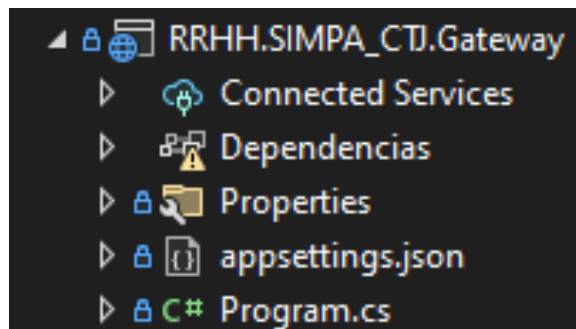


Figura 15 Proyecto GATEWAY

Elementos clave:

- Program.cs: configuración de los elementos utilizados para enrutar y redireccionar, como YARP u otros Proxy.
- Launchsettings.json: Gestiona la configuración de direcciones de despliegue de estos servicios.
- Appsettings.json: Configura otros aspectos de estos servicios, como políticas de aceptación de peticiones (uno por entorno).

Arquitectura sin API Backend

Para cada frontal tendremos un proyecto general que correrá de manera independiente. Por lo que cada proyecto contará con sus propios archivos de configuración y de ejecución.

Elementos clave:

- Program.cs: Configuración de servicios generales de la aplicación como: controladores, autenticación, loggers, CORS, Swagger, Hangfire, etc. Además, se establece un mapeo para lanzar adecuadamente el proyecto frontal correspondiente.
- Carpeta “ClientApp”: Carpeta que almacena los ficheros que componen el frontal correspondiente.
- Appsettings.json: Almacenar configuraciones generales de la aplicación, por ejemplo, los strings de conexión a BBDD (uno por entorno).
- Launchsettings.json: Almacena configuraciones de ejecución, por ejemplo, las URL.
- Carpeta “Controlllers”: Carpeta donde se encuentran los controladores.

- Carpeta “BackgroundTasks” *: Carpeta para almacenar las clases con los métodos que se ejecutan en segundo plano de manera independiente o temporal.
- Carpeta “Services” *: Carpeta para almacenar servicios de gestión propios, por ejemplo, controles de acceso.

4.5.2 Proyectos Backend, librerías y servicios

Proyecto Application

El proyecto Application se encarga de definir las interfaces y componentes generales que encapsulan la lógica de negocio de la aplicación, con clases como interfaces de servicios, repositorios o commands y queries.

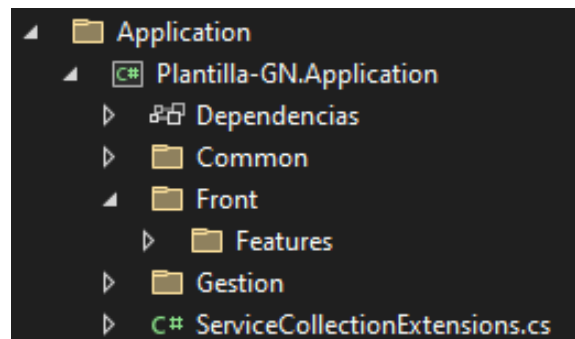


Figura 16 Proyecto Application

Elementos clave:

- ServiceCollectionExtensions.cs: La clase ServiceCollection se utiliza para registrar y configurar los servicios globales que se utilizarán en toda la aplicación Core.
- Carpeta “Common”: Carpeta que almacena elementos comunes a todos los frontales, se suelen dividir por carpetas según su tipo de funcionalidad: servicios, repositorios, etc.
- Carpetas “Front/Features”: Carpeta que almacena clases con funcionalidades específicas para cada frontal, como commands y queries específicos.

Proyecto Infrastructure

Este proyecto se encarga de la implementación de componentes referidos a la conexión y gestión de servicios externos.

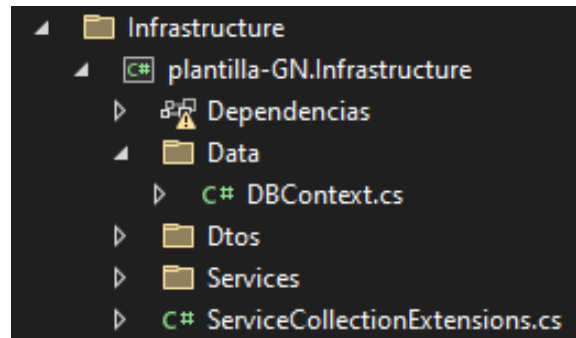


Figura 17 Proyecto Infrastructure

Elementos clave:

- ServiceCollectionExtensions.cs: La clase ServiceCollection se utiliza para registrar y configurar los servicios externos que se utilizarán en toda la aplicación Core.
- Carpeta “Data”: Carpeta que almacena la implementación de elementos para la conexión con los servicios, por ejemplo, el DbContext.
- Carpeta “Dtos”: Carpeta que almacena las entidades utilizadas por estos servicios externos.
- Carpeta “Services”: Recoge en sus clases la ejecución de las funcionalidades propias de cada servicio.

Proyecto Persistence

El proyecto Persistence contiene los elementos de configuración para el ORM y la conexión a base de datos o servicios locales.

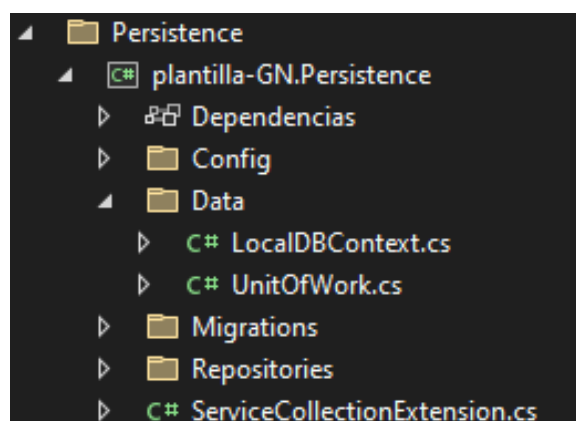


Figura 18 Proyecto Persistence

Elementos clave:

- ServiceCollectionExtensions.cs: La clase ServiceCollection se utiliza para registrar y configurar los repositorios o elementos usados por servicios locales, como el modelo de datos.

- Carpeta “Data”: Almacena la implementación de elementos de conexión y configuración de dichos servicios locales, como el DbContext o la implementación de interfaces como UnitOfWork.
- Carpeta “Config”: Almacena los ficheros de configuración de entidades correspondientes al modelo de base de datos establecido en el proyecto Domain.
- Carpeta “Repositories”: Almacena la implementación de los repositorios establecidos en el proyecto Application.

Proyecto Domain

El proyecto Domain actúa como librería de clases para la capa de dominio, conteniendo las clases y estructuras que representan los objetos de la aplicación y serán utilizados por procesos internos. En el caso de uso de ORMs, como entity Framework, estas clases serían mapeadas directamente a tablas en una base de datos. Es importante no confundir con los elementos de la carpeta Dtos, los cuales van relacionados con servicios externos.

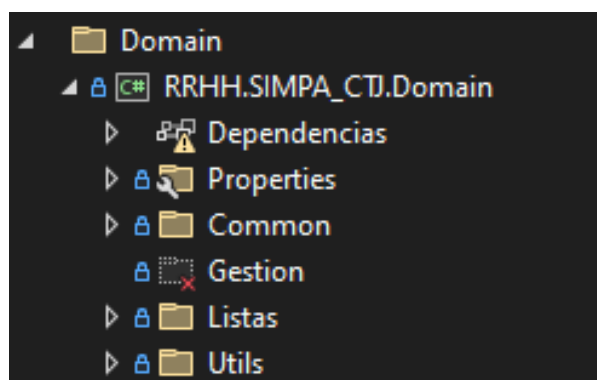


Figura 19 Proyecto Domain

Elementos clave:

- Carpeta “Common”: Carpeta que se utiliza para almacenar las clases del dominio utilizadas de manera común.
- Carpeta “nombre_del_frontal” *: Una carpeta para cada frontal que presente la aplicación, almacena las clases específicas del correspondiente frontal.

4.5.3 Proyectos Frontend

Los proyectos de Frontend se asientan de una manera u otra en función de la arquitectura Core utilizada. Las soluciones con API.CORE tienen los elementos de cada frontal en un proyecto separado del Core, que se encarga solamente de almacenarlo y lanzarlo con los ajustes correspondientes.

Las soluciones sin API.CORE tienen un Core de backend que se ejecuta de manera independiente para cada frontal, con sus propias configuraciones, como ya hemos visto en apartados anteriores.

Los proyectos de Front no tienen por qué estar predeterminados por una arquitectura, se puede usar cualquier tipo de Framework. En este caso, en la plantilla hemos incluido dos frontales base, uno con Angular y otro, también con angular, pero usando COREUI.

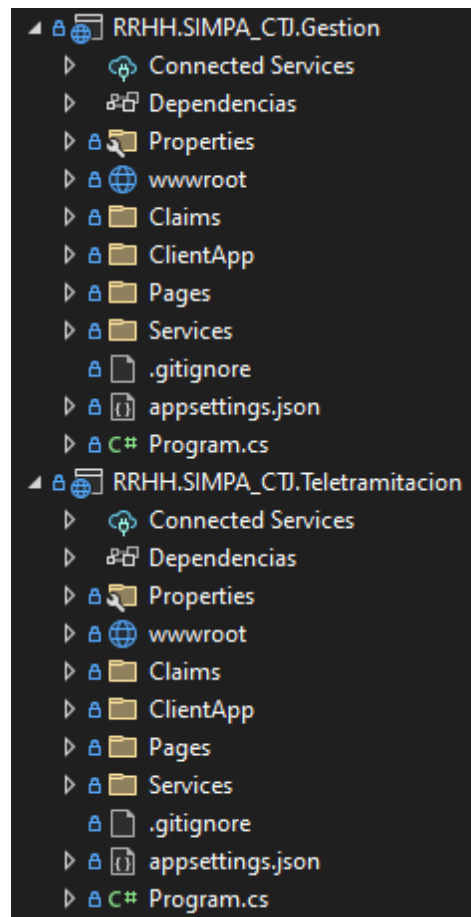


Figura 20 Proyectos Frontend

4.6 Modificaciones del proyecto

4.6.1 Multi idioma

Primero hay que crear la carpeta “i18n” en el directorio “ClientApp\src\assets” del proyecto frontend de teletramitación. Dentro de esta carpeta habrá que crear los archivos JSON que serán los idiomas que vayan a utilizarse en la aplicación.

En el caso de mi proyecto cuenta con dos idiomas:

- es.json (español)
- eu.json (euskera)

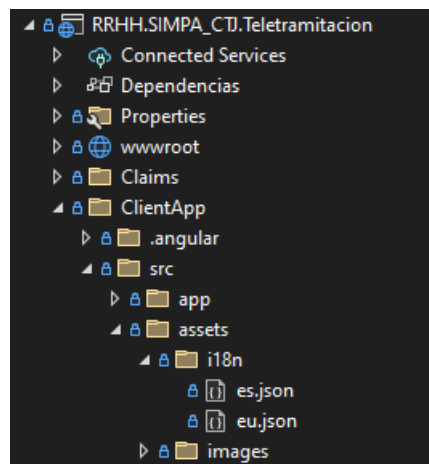


Figura 21 Carpeta i18n con los json de idiomas

También hace falta realizar cambios en el archivo “app.module.ts”, dónde habrá que importar algunas clases de los paquetes anteriormente instalados, además de alguno de Angular.

```
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { TranslateLoader, TranslateModule } from '@ngx-translate/core';
import { TranslateHttpLoader } from '@ngx-translate/http-loader';
```

```
TranslateModule.forRoot({
  loader: {
    provide: TranslateLoader,
    useFactory: (http: HttpClient) => new TranslateHttpLoader(http, './assets/i18n/', '.json'),
    deps: [HttpClient]
  }
})
```

Figura 22 Archivo app.module.ts

4.6.1 Registra

Apunte:

```
public struct Apunte
{
    public string Tipo;
    public int IdAnyo;
    public int IdDocumento;
    public int IdAplicacion;
    public int IdTipoDoc;
    public int IdUdEntrada;
    public int IdUdSalida;
    public string FecEntrada;
    public string FecLegal;
```



```

        public string Resumen;
        public string Descripcion;
        public CIFNIF Solicitante;
        public Domicilio DomSolicitante;
        public CIFNIF Presentador;
        public Domicilio DomPresentador;
        public string NotificacionTelematica;
        public string FecEnvio;
        public int IdUdDestino;
        public CIFNIF Destinatario;
        public Domicilio DomDestinatario;
        public int IdTipoEnvio;
    }

```

Este era necesario porque primero de todo debía hacer el alta de un apunte en el servicio de Registra. Hay algunos campos que no son obligatorios, como Presentador y en nuestro caso no puede presentar alguien un apunte por otra persona, por lo que ese campo estará siempre vacío. El método empleado para dar de alta un apunte será `Alta_Apunte_Telematico`, al cual hay que pasarle como parámetro el Apunte que se desea dar de alta.

Alta_Apunte_Telematico

Funcionalidad

Añade un nuevo documento al registro de entrada o de salida dependiendo del valor indicado en el parámetro “Tipo” (“E” o “S”)

Parámetros de entrada del método:

- **Apunte (struct [Apunte](#)):** Información del apunte que queremos dar de alta.

Valor devuelto por el método ([int](#)):

- **-1: Proceso cancelado al no cumplir todos los criterios de validación o producirse un error.**
- El número del documento de entrada o de salida dado de alta si la operación se ha realizado correctamente.

Para poder adjuntar un archivo hay que pasarle tanto el archivo que será adjuntado que se le pasa como bytes, como el nombre, descripción fecha o el `IdAplicacion`, que es propio de cada proyecto desarrollado para el gobierno de Navarra que use el servicio de Registra.

Adjuntar_Archivo

Funcionalidad

Método encargado de agregar un archivo al SAR y actualiza sus enlaces en la base de datos de registra.

Parámetros de entrada del método:

- **Tipo ([string](#)).** “E” entrada, “S” Salida. **Obligatorio.**
- **IdAplicacion ([int](#)).** Identificador de la aplicación que realiza el alta. **Obligatorio.**
- **IdAnyo ([int](#)).** Identificador del año al que pertenece el apunte registral. **Obligatorio,**

- **IdDocumento** ([int](#)). Número de documento (apunte registral). **Obligatorio**.
- **IdUnidadEntrada** ([int](#)). Identificador de la Unidad de entrada. **Obligatorio**.
- **Solicitante** ([string](#)). Nif del Solicitante relacionado con el apunte. **Opcional**.
- **Nombre** ([string](#)). Nombre del Archivo. **Obligatorio**.
- **Descripcion** ([string](#)). Descripción del Archivo. **Obligatorio**.
- **Metadatos** ([string](#)). Metadatos del documento. **Obligatorio**.
- **Instancia** ([int](#)). Valor de la instancia. En caso de no depender de ninguna instancia, asignar 0. **Obligatorio**.
- **ficheroBytes** ([byte\[\]](#)). Contenido del Archivo a agregar. **Obligatorio**.

Valor devuelto por el método: [int](#)

- El número del fichero creado.
- 0. Se ha almacenado el fichero en SAR pero ha fallado al guardar el proceso en Registra
- -1: No se ha guardado en SAR ni en Registra

RGE_Actualizar_Hoja_Ruta_UN

Funcionalidad

Actualiza el campo Fecha de Recepción de un Registro de Entrada de Registra para un año y documento específico pasados como parámetros. De esta manera el documento queda recepcionado automáticamente en Registra.

Parámetros de entrada del método:

- **IdAño**: Año del documento de 4 cifras ([short](#)).
- **IdDoc**: Número del documento de entrada ([int](#)).
- **Fecha_Envio**. Fecha en la que se va a recepcionar el documento ([string\(10\)](#)).

Valor devuelto por el método: ([int](#))

- **0**. Actualización correcta.
- **-1**. Actualización no terminada.

RGE_Anular_Instancia

Funcionalidad

Actualiza el campo Fecha de Recepción de un Registro de Entrada de Registra para un año y documento específico pasados como parámetros. De esta manera el documento queda recepcionado automáticamente en Registra.

Parámetros de entrada del método:

- **IdAño**: Año del documento de 4 cifras ([short](#)).
- **IdDoc**: Número del documento de entrada ([int](#)).
- **Fecha_Envio**. Fecha en la que se va a recepcionar el documento ([string\(10\)](#)).

Valor devuelto por el método: ([int](#))

- **0**. Actualización correcta.
- **-1**. Actualización no terminada.
-

Todos estos son los métodos que necesite usar para hacer el registro en Registra, así como alguna clase más como es CIFNIF que cuenta con todos los datos de la persona que solicita hacer el registro y la clase Domicilio, que cuenta con varios datos sobre el domicilio de la persona que ha solicitado el Apunte, para que así quede constancia de todo.

4.6.2 Tabla apuntes

En este proyecto para poder generar una tabla nueva hace falta generar varios archivos y después de estar todos creados hacer una migración, haciendo que se genere la tabla nueva. De este modo primero tenía que crear un archivo en la carpeta Domain, llamado “Apunte.cs” que era la clase Apunte, que contaba con varios campos.

```
public partial class Apunte
{
    [Key]
    4 referencias | Luis Azcoiti Peña, Hace 4 días | 1 autor, 1 cambio
    public int IdApunte { get; private set; }
    8 referencias | Luis Azcoiti Peña, Hace 4 días | 1 autor, 1 cambio
    public int IdAspiranteLista { get; private set; }
    6 referencias | Luis Azcoiti Peña, Hace 4 días | 1 autor, 1 cambio
    public string Nif { get; private set; }
    5 referencias | Luis Azcoiti Peña, Hace 4 días | 1 autor, 1 cambio
    public int NumeroDocumento { get; private set; }
    5 referencias | Luis Azcoiti Peña, Hace 4 días | 1 autor, 1 cambio
    public stringCodigoCSV { get; private set; }
    5 referencias | Luis Azcoiti Peña, Hace 4 días | 1 autor, 1 cambio
    public string Resumen { get; private set; }
    5 referencias | Luis Azcoiti Peña, Hace 4 días | 1 autor, 1 cambio
    public bool SolicitaNotificacionesTelematicas { get; private set; }

    3 referencias | Luis Azcoiti Peña, Hace 4 días | 1 autor, 1 cambio
    public virtual AspiranteLista? AspiranteLista { get; private set; }
```

Figura 23 Campos de la clase Apunte

Como podemos ver en la imagen contiene también un IdAspiranteLista, para estar relacionado así con el aspirante que se haya apuntado a una lista en concreto. También cuenta con un constructor con todos los campos, menos IdApunte, que se genera de manera automática siendo un número mayor que el último introducido a la tabla. Contando también con un método para instanciar de manera estática un objeto “Apunte”.

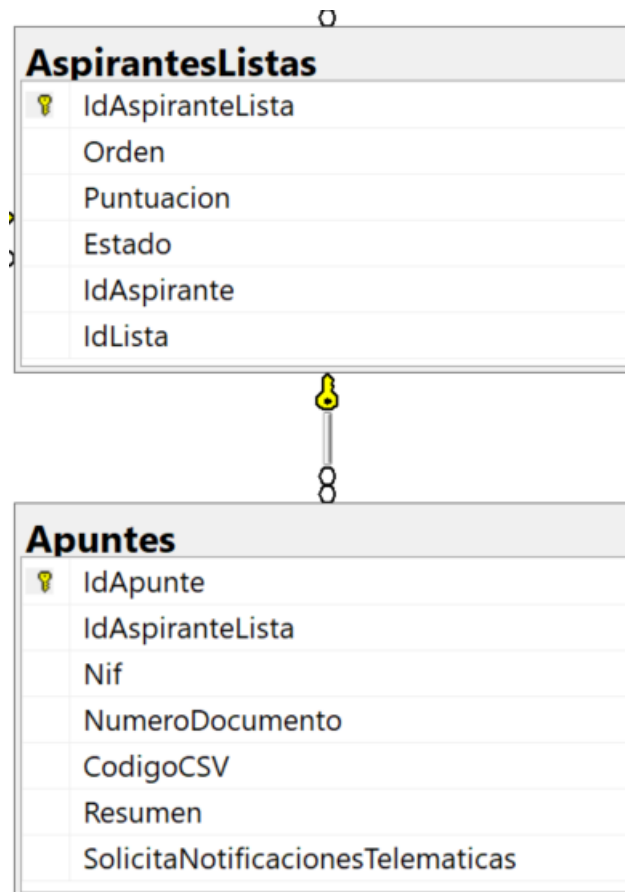


Figura 24 Tabla apuntes generada durante la migración

5. Implementación

5.1 Multi idioma

Para llevar a cabo esta implementación, primero hay que instalar 2 paquetes de ngx-translate. Estos paquetes se instalan fácilmente abriendo un terminal en el directorio del proyecto y usando estos comandos:

- npm i @ngx-translate/core -save
- npm i @ngx-translate/http-loader -save

Una vez instalado hay que crear la carpeta “i18n” en el directorio “ClientApp\src\assets” del proyecto frontend de teletramitación. Dentro de esta carpeta habrá que crear los archivos JSON que serán los idiomas que vayan a utilizarse en la aplicación. En nuestro caso “es.json” para el español y “eu.json” para el euskera.

Lo primero para poder usarlo será en el TypeScript del componente del proyecto Frontend teletramitación, hay que importar el servicio de multi idioma e inyectarlo en el constructor. Esto se ve fácilmente con la siguiente imagen:

```

import { ChangeDetectorRef, Component, OnInit } from '@angular/core';
import { Title } from '@angular/platform-browser';
import { NavigationEnd, Router } from '@angular/router';
import { TranslateService } from '@ngx-translate/core';
import { IconSetService } from '@coreui/icons-angular';

import { iconSubset } from '../shared/icons/icon-subset';
import { LoadingService, ProfileService } from '../data';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent implements OnInit {
  title = 'Contratación temporal justicia';
  cargando?: boolean;

  constructor(
    private router: Router,
    private titleService: Title,
    private iconSetService: IconSetService,
    private readonly translate: TranslateService,
    private loadingService: LoadingService,
    private profileService: ProfileService,
    private cdr: ChangeDetectorRef
  ) {
    this.translate.addLangs(['es', 'eu']);
    this.translate.setDefaultLang('es');
    this.translate.use('es');
  }
}

```

Figura 25 Constructor portal Teletramitación

Aprovechando la imagen de arriba estos son los pasos que hice para poder usar el multi idioma:

- Ahora hay que agregar los idiomas que se van a utilizar con “this.translate.addLangs(['es', 'eu']);”. De este modo “es” se usará para el es.json y “eu” para el eu.json.
- Marcar un idioma por defecto: “this.translate.setDefaultLang('es');”
- Hace falta tener un control para poder seleccionar el idioma que seleccione el usuario. Ese control estará en un componente que deberá asignar el lenguaje seleccionado mediante la siguiente instrucción con el idioma que corresponda: “this.translate.use('es');”.

Por último, pondré un ejemplo de cómo se usa el multi idioma desde el HTML:

```

<table>
<tr>
<td valign="top">
<p class="numero-tabla"><strong>01</strong></p>
<hr class="linea-horizontal">
<h2>
<a class="linkTituloOpcion" routerLink="/portal/listas">
{{'menu.MisListas'|translate}}
</a>
</h2>
</td>
<td valign="top">
<p class="numero-tabla"><strong>02</strong></p>
<hr class="linea-horizontal">
<h2>
<a class="linkTituloOpcion" routerLink="/portal/ofertas">
{{'menu.Ofertas'|translate}}
</a>
</h2>
</td>
</tr>
</table>

```

Figura 26 Archivo home.component.html

```

{
  "general": {
    "Volver": "Volver"
  },
  "menu": {
    "MisListas": "Mis listas",
    "Ofertas": "Ofertas",
    "Bolsas": "Bolsas",
    "Notificaciones": "Notificaciones",
    "AreaPersonal": "Área personal",
    "FichaPersonal": "Ficha personal",
    "Documentacion": "Documentación",
    "EditarPerfil": "Editar perfil",
    "CerrarSesion": "Cerrar sesión"
  },
}

```

Figura 27 Archivo es.json

```

{
  "general": {
    "Volver": "Volver EU"
  },
  "menu": {
    "MisListas": "Mis listas EU",
    "Ofertas": "Ofertas EU",
    "Bolsas": "Bolsas EU",
    "Notificaciones": "Notificaciones EU",
    "AreaPersonal": "Área personal EU",
    "FichaPersonal": "Ficha personal EU",
    "Documentacion": "Documentación EU",
    "EditarPerfil": "Editar perfil EU",
    "CerrarSesion": "Cerrar sesión EU"
  },
}

```

Figura 28 Archivo eu.json



Figura 29 Menú home de teletramitación en español



Figura 30 Menú home de teletramitación en euskera

5.2 Registra

Para poder utilizar este servicio lo primero que tuve que hacer fue incluirlo en los Services de la carpeta ExternalServices, de modo que se generaba de manera automática un archivo con todas las clases y métodos de Registra. Después creé varios archivos que serían los que llamarían a ese servicio para así poder usarlos en el proyecto.

Aquí voy a mostrar varios ejemplos de cómo utilizar todos los métodos mencionados anteriormente en el apartado de análisis y diseño:

5.2.1 Alta_Apunte_Telematico

```
Apunte _Apunte = new Apunte();
_Apunte.Tipo = "E";
_Apunte.IdAnyo = System.DateTime.Today.Year;
_Apunte.IdAplicacion = 2;
```

```

_Apunte.IdUdEntrada = 32000;
_Apunte.IdTipoDoc = 108;
_Apunte.FecEntrada = "29/10/2014";
_Apunte.FecLegal = "29/10/2014";
_Apunte.Resumen = "Prueba Alta Telemática Nuevo método";
_Apunte.Solicitante = CIFNIF_Persona;
_Apunte.DomSolicitante = DomSolicitante;
_Apunte.NotificacionTelematica = "0";
_Apunte.FecEnvio = "29/10/2014";
_Apunte.IdUdDestino = 10003635;
int result = SrvRegistra.Alta_Apunte_Telematico(_Apunte

```

```

public ResultadoAltaApunte RegistrarApunte(DatosAltaApunte datosAltaApunte)
{
    DateTime fechaEntrada = DateTime.Today;
    var requestBody = GetAltaApunteRequest(datosAltaApunte, fechaEntrada.Year, fechaEntrada);
    var resultado = _wsRegistra.Alta_Apunte_TelematicoAsync(requestBody);
    return new ResultadoAltaApunte
    {
        NumeroDocumento = resultado.Result.Body.Alta_Apunte_TelematicoResult,
        EsValido = resultado.Result.Body.Alta_Apunte_TelematicoResult != _configuration.CodigoError
    };
}

```

Figura 31 Método RegistrarApunte

5.2.2 Adjuntar_Archivo

Ejemplo de llamada 1 (c#):

Registra SrvRegistra = new Registra();

string Metadatos =

```

"<ADI><documento><fechaAdministrativa>09/01/2015</fechaAdministrativa>
<organo>10003127</organo><titulo></titulo><tipoDocumento>1</tipoDocume
nto><descripcion>Descripción
archivo</descripcion><idioma></idioma><nivelLOPD>1</nivelLOPD><acceso>1</acces
o><estado>EE01</estado><firmado>0</firmado></documento><firmas></audit
oria><usuario>REGISTRA</usuario></auditoria><infoValidacion/><docFinal
>true</docFinal><publico>true</publico><convertir>false</convertir></A
DI>"

```

HttpPostedFile fichero = fuUpload.PostedFile;

int longitud = fichero.ContentLength;

byte[] datos = new byte[longitud];

fichero.InputStream.Read(datos, 0, longitud);

```

int _Fichero = SrvRegistra.Adjuntar_Archivo("E", 1, 2015, 2, 10003127, "",
fuUpload.FileName, "Descripción", Metadatos, 1, datos);

```



```

public ResultadoAdjuntarArchivo AdjuntarArchivo(DatosAdjuntarArchivo datosAdjuntarArchivo)
{
    var request = new Adjuntar_ArchivoRequest
    {
        Body = new Adjuntar_ArchivoRequestBody
        {
            Tipo = "E",
            IdAplicacion = 0,
            IdAnyo = datosAdjuntarArchivo.FechaRegistro.Year,
            IdDocumento = datosAdjuntarArchivo.NumeroApunte,
            IdUnidad = 0,
            Nombre = datosAdjuntarArchivo.NombreArchivo,
            Descripcion = datosAdjuntarArchivo.DescripcionArchivo,
            Solicitante = datosAdjuntarArchivo.NIFSolicitante,
            Metadatos = GetMetadata(datosAdjuntarArchivo),
            Instancia = 1,
            ficheroBytes = datosAdjuntarArchivo.Contenido
        }
    };
    var respuesta = _wsRegistra.Adjuntar_ArchivoAsync(request);

    return new ResultadoAdjuntarArchivo
    {
        NumeroFicheroCreado = respuesta.Result.Body.Adjuntar_ArchivoResult,
        EsValido = respuesta.Result.Body.Adjuntar_ArchivoResult > 0
    };
}

```

Figura 32 Método AdjuntarArchivo

5.2.3 RGE_Actualizar_Hoja_Ruta_UN

Ejemplo de llamada (c#):

RGE_Actualizar_RE_Hoja_Ruta_UN (2013,1587,"27/11/2013");

```

public bool ActualizarHojaRuta(DatosActualizarHojaRuta datosActualizarHojaRuta)
{
    string fechaFormateada = GetFormattedDate(datosActualizarHojaRuta.FechaEnvio);
    var petition = new RGE_Actualizar_RE_Hoja_Ruta_UNRequest
    {
        Body = new RGE_Actualizar_RE_Hoja_Ruta_UNRequestBody
        {
            IdAño = datosActualizarHojaRuta.IdAño,
            IdDoc = datosActualizarHojaRuta.IdDocumento,
            Fecha_Envio = fechaFormateada
        }
    };

    var respuesta = _wsRegistra.RGE_Actualizar_RE_Hoja_Ruta_UNAsync(petition);
    return respuesta.Result.Body.RGE_Actualizar_RE_Hoja_Ruta_UNResult != _configuration.CodigoError;
}

```

Figura 33 Método ActualizarHojaRuta

5.2.4 RGE_Anular_Instanceia

Ejemplo de llamada (c#):

RGE_Actualizar_RE_Hoja_Ruta_UN (2013,1587,"27/11/2013");

```

public bool AnularInstancia(DatosAnularInstancia datosAnularInstancia)
{
    var petition = new RGE_Anular_InstanciaRequest
    {
        Body = new RGE_Anular_InstanciaRequestBody
        {
            IdAnyo = datosAnularInstancia.IdAnyo,
            IdDoc = datosAnularInstancia.IdDoc,
            Tipo = datosAnularInstancia.Entrada
        }
    };

    var respuesta = _wsRegistra.RGE_Anular_InstanciaAsync(petition);
    return respuesta.Result.Body.RGE_Anular_InstanciaResult != _configuration.CodigoError;
}

```

Figura 34 Método AnularInstancia

5.3 PDF

Para poder utilizar la librería iTextSharp, primero la instalé en el proyecto, usando el administrador de paquetes NuGet de Visual Studio. Una vez instalado le di un formato idéntico al que teníamos de ejemplo, modificando los datos de este en función de nuestras necesidades concretas. En este caso como será una persona quien se inscriba personalmente, no necesitará el apartado de ejemplo que tenía “en representación de” ni el de “documentos aportados” al no necesitar aportar ningún tipo de archivo.

La información más importante de este documento son los datos de la persona que se haya inscrito, como nombre, apellidos, NIF y además, la fecha en la que se apuntó, por lo que ese dato también se genera de manera automática al ser generado el PDF, teniendo constancia de los datos más importantes.

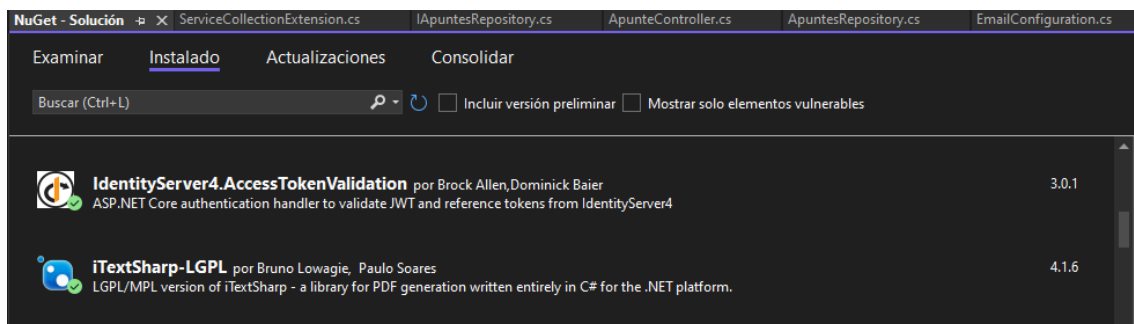


Figura 35 Administrador de paquetes NuGet donde se puede ver el paquete iTextSharp instalado en el proyecto

Instancia General

Presentado por

Nombre y apellidos: Miguel Peña Gorricho	DNI/NIF: 73737373A
Dirección: Mutilva Baja, 4	Código postal: 31006
Localidad: Pamplona	Provincia:
Teléfono: 600800400	

Datos de la solicitud

Destino: GOBIERNO DE NAVARRA, DEPARTAMENTO DE INTERIOR, FUNCIÓN PÚBLICA Y JUSTICIA

Resumen de la solicitud: Miguel Peña Gorricho se ha apuntado a una oferta de plaza

Fecha inscripción: 25/04/2024 13:18:04

Solicitud de notificación telemática

No se ha solicitado respuesta telemática

Figura 36 PDF generado mediante iTextSharp

5.4 CSV

Para poder usar los métodos CSV de la API necesitamos primero obtener un token para autenticar que tenemos permiso para usar este servicio. Para ello tendremos que obtener el token de Ateka, que es el sistema de autenticación del gobierno de Navarra.

```
// 3º Firmar con CSV
var codigo = await _firmaPdfConCsv.FirmaPdf(_obtenerTokenAteka.TokenAteka().Result).ConfigureAwait(false);
if (codigo == null)
{
    var resultadoAnulacion = _anularInstancia.AnularInstanciaMetodo(result.GetValue().NumeroDocumento, servicio);
    if (resultadoAnulacion)
    {
        return new Response(true, "Error al tratar de firmar el csv", result.GetValue().NumeroDocumento);
    }
    return new Response(false, "Error al tratar de firmar el csv", result.GetValue().NumeroDocumento);
}
```

Figura 37 Uso método FirmarPdf con CSV

Este token se le pasa al método FirmarPdf, donde lo primero que se hace es firmar el PDF que se ha generado anteriormente con el CSV y de ese modo se generará un PDF igual, pero con el código CSV en la parte inferior de este, además de que se obtendrá el código CSV generado, almacenándolo en la variable codigo, que más adelante será usado.

Aquí hay un ejemplo de cómo se vería la parte de abajo del PDF tras ser firmado:

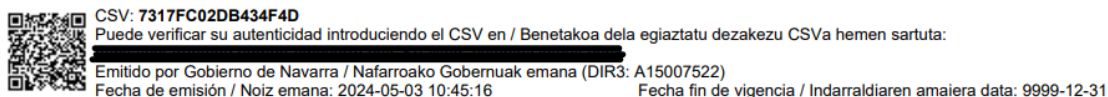


Figura 38 Ejemplo firma CSV en PDF

Una vez se tiene ese CSV se podría buscar tanto con el QR, como con el propio CSV en la página del gobierno de Navarra una copia original de ese archivo, de modo que, aun recibiendo una copia modificada, podamos de manera segura y sencilla comprobar si es o no el original usando esto.

5.5 Almacenar apuntes en la base de datos

Para que a la hora de la migración se genere de manera correcta la tabla hace falta añadir en el proyecto Persistence un archivo configuration en el que se pone que campos son obligatorios, valores por defecto si se quieren usar, si son claves primarias o foráneas algunos de los campos y por último si alguna es foránea como se relaciona con la otra tabla, en este caso siendo con la tabla AspiranteLista mediante el campo IdAspiranteLista.

```
internal class ApunteConfiguration : IEntityTypeConfiguration<Apunte>
{
    0 referencias | Luis Azcóiti Peña, Hace 4 días | 1 autor, 1 cambio
    public void Configure(EntityTypeBuilder<Apunte> builder)
    {
        builder.ToTable("Apuntes");
        builder.HasKey(e => e.IdApunte);
        builder.Property(e => e.IdAspiranteLista).IsRequired();
        builder.Property(e => e.Nif).IsRequired().HasColumnType("varchar(9)");
        builder.Property(e => e.NumeroDocumento).IsRequired();
        builder.Property(e => e.CodigoCSV).IsRequired();
        builder.Property(e => e.SolicitaNotificacionesTelematicas).IsRequired().HasDefaultValue(false);
        builder.Property(e => e.Resumen);

        builder.HasOne(d => d.AspiranteLista).WithMany(p => p.Apuntes)
            .HasForeignKey(d => d.IdAspiranteLista)
            .HasConstraintName("FK_Apunte_Aspirante_IdAspirante");
    }
}
```

Figura 39 Contenido del archivo ApunteConfiguration

En este caso cada apunte puede ser de un único AspiranteLista, mientras que cada AspiranteLista puede tener varios apuntes al poder apuntarse un mismo Aspirante a la misma lista más de una vez.

Con esto sería posible crear la tabla en la base de datos haciendo una migración. Para ello primero hay que poner como proyecto de inicio el proyecto que tiene el string de conexión, en este caso el API.Core:

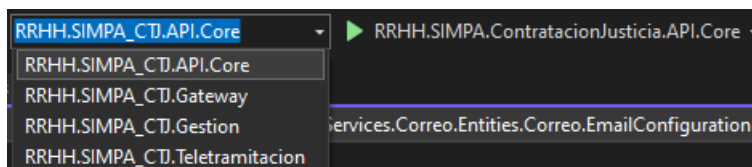


Figura 40 Proyecto de inicio único API.Core

Después hay que abrir una consola de NuGet donde habrá que establecer como proyecto por defecto el que contenga la información del EntityFramework, en este caso Persistence.

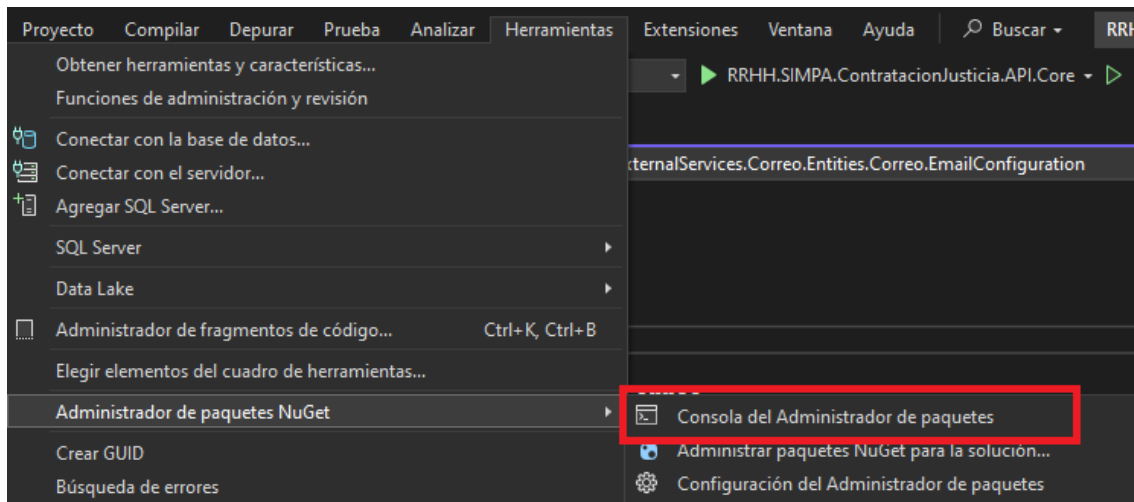


Figura 41 Donde encontrar la consola de administrador de paquetes en visual studio

Una vez hecho esto habrá que usar los siguientes comandos en la consola:

- add-migration nombreMigracion
- update-database -verbose

Mediante este comando se hará la migración y se generará en caso de estar bien los archivos, la nueva tabla con sus campos y relacionada con la otra tabla AspiranteLista.

Después de generar de manera correcta la tabla toca crear los métodos que serán utilizados para añadir a esta los Apuntes generados por el servicio de Registra junto con el código CSV. Para poder añadir Apuntes a esta tabla hace falta crear 3 archivos nuevos. Uno en API.Core que será el controlador que manejará las solicitudes HTTP relacionadas con los apuntes, recibiendo las solicitudes del cliente, procesarlas y devolver una respuesta apropiada. En mi caso he creado ApunteController que cuenta con dos operaciones get, una para obtener todos los apuntes y otra para obtener un único apunte en función del IdAspiranteLista y una post para crear un nuevo apunte cuando alguno de los aspirantes se apunte a una lista.

```
[HttpPost("nuevoApunte", Name = "PostApunte")]
[Consumes("application/json")]
public async Task<ActionResult<NuevoApunte.Response[]>> NuevoApunte([FromQuery] NuevoApunte.Command request)
{
    var errorOrNuevoApunte = await Mediator.Send(request);
    return errorOrNuevoApunte.Match(
        value => Ok(value),
        errors => Problem(errors));
}
```

Figura 42 Operación nuevo apunte controlador

Los otros dos archivos necesarios para esto son ApunteRepository, que es un componente que da acceso a datos que encapsulan la lógica para dar interactuar con la base de datos y realizar operaciones CRUD en la tabla apunte, por otro lado, tenemos el

archivo `IApunteRepository` que es una interfaz permite que el controlador interactúe con el repositorio a través de ella. Estos dos archivos habrá que añadirlos a los servicios metiéndolos en el archivo `ServiceCollectionExtensions.cs` añadiendo:

- `services.AddTransient<IApunteRepository, ApunteRepository>();`

```
public void NuevoApunte(Apunte apunte)
{
    if (apunte != null)
    {
        _context.Apuntes.Add(apunte);
    }
}
```

Figura 43 Método nuevo apunte del archivo `ApunteRepository`

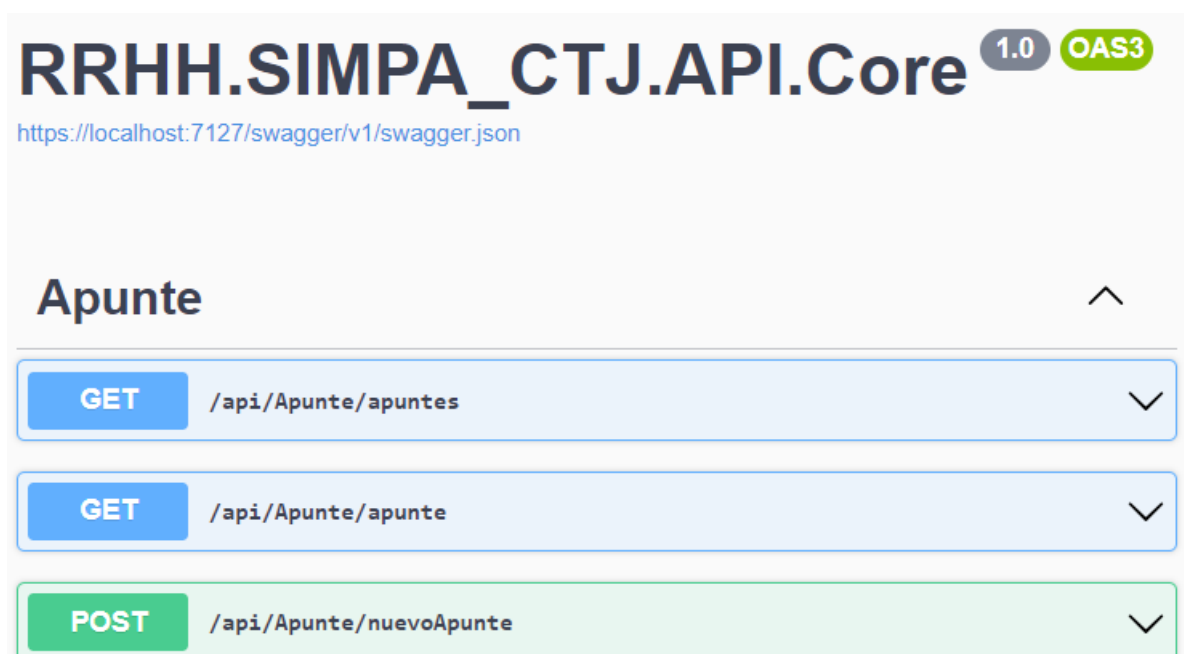


Figura 44 Operaciones que se pueden realizar de Apunte en el Swagger

5.6 Proceso de registro en Registra

Una vez que funcionaba todo lo anterior había que seguir un orden para hacerlo todo de manera correcta. Por lo que el primer paso era el de dar de alta el apunte en Registra. Este paso es el más importante y en caso de que hubiese algún fallo durante el resto de los procesos habría que anular la instancia de este apunte, de modo que sea una transacción distribuida. En este paso se le pasan unos datos de alta que son por ejemplo el nombre, apellidos, DNI entre otros, de la persona que se ha inscrito y en caso de ir bien devuelve el número del documento generado, siendo -1 en caso de haber fallado algo. Este número es muy importante, ya que será necesario en los siguientes pasos, como por ejemplo si algo falla y hay que anular esa instancia o para los otros métodos de Registra que se usarán más adelante.

El siguiente paso es el de generar el PDF, el cual se genera mediante la librería iTextSharp, anteriormente explicada con algunos de los campos que se usaron para generar el apunte en Registra, así como algún otro valor predefinido como la fecha en la que se creó o el departamento del gobierno de Navarra. Este PDF se guarda de manera temporal, ya que se usará para el siguiente paso.

El tercer paso es el de la firma del PDF con CSV, este paso requiere de un token que se obtiene de Ateka para poder autenticar que está siendo usado por nuestro proyecto en concreto. De este modo se generará un nuevo PDF con el CSV al final de este.

El cuarto paso es el de adjuntar el archivo generado en Registra, usando el PDF que ha sido firmado y los datos del apunte, incluyendo los datos personales del solicitante y el número de documento del primer paso.

El quinto paso es el de almacenar en la base de datos del proyecto los datos del apunte, así como el código CSV generado anteriormente en el paso 3. De este modo también se guarda en nuestra propia base de datos esta información.

Y el último paso sería el de actualizar la hoja de ruta, tras el cual si todo ha ido bien ya habría acabado todo el proceso de registrar un apunte en Registra. En caso de que hubiese habido algún error en algunos de los pasos anteriores se habría hecho la anulación de instancia del apunte.

6. Valoración y conclusión

6.1 Valoración

Durante el tiempo que he formado parte del desarrollo de este proyecto he aprendido cómo funciona realmente el desarrollo de un proyecto informático desde la parte de requisitos hasta un desarrollo bastante avanzado.

Además, aprendí a utilizar nuevos lenguajes y frameworks que no había usado antes, como es el caso de C# o Angular, que son lenguajes modernos que se usan cada vez más en esta industria y son de gran interés a la hora de realizar proyectos futuros de páginas web.

Al principio realice 2 proyectos pequeños para aprender antes de meterme con el proyecto que tenían en desarrollo, para que una vez que me metieran supiera lo que hacía y así experimentar de manera segura. En todo momento tuve ayuda por parte de los otros integrantes del equipo y gente con mucha más experiencia que supo ayudarme para llegar a conseguir los objetivos de aprendizaje y finalmente desarrollar por mi cuenta algunas funcionalidades extras en este proyecto web.

Estos proyectos fueron tanto una página web de recursos humanos, donde por una parte se podían gestionar a los empleados, como por otra parte los empleados ver sus datos personales y modificar algunos de ellos. La otra era una que permitía la búsqueda de deportistas profesionales en función de ciertos parámetros que se introducían en un

formulario. Estos 2 proyectos tenían una arquitectura similar a la del proyecto que estaban desarrollando y usaba las mismas tecnologías, lo que me permitió después trabajar sin problema alguno en el proyecto de contratación temporal de justicia.

Lo que más hice en este proyecto fue añadir integraciones con servicios de Gobierno de Navarra, como fue el caso con Registra, CSV, envío de Mails y SMSs. La primera vez fue bastante complicado, al no haber hecho nada similar antes, pero gracias a que se añaden de manera similar y la ayuda tanto de mis compañeros de trabajo, como la utilización de ellos en anteriores proyectos facilitó la labor tanto de investigación de estos servicios, como la integración.

6.2 Conclusión

Este proyecto se llevó a cabo, ya que el gobierno de Navarra hacía hasta ese momento todo de manera muy anticuada, por lo que contaba con varios aspectos negativos. Los procesos eran manuales, la ordenación y el mantenimiento de las listas se realizaba de manera manual, lo que requiere mucho tiempo y puede llevar a error humano, el llamamiento se realizaba uno por uno a cada candidato utilizando correo electrónico y llamadas telefónicas siendo ineficiente y consumiendo mucho tiempo.

Por eso al crear esta web se automatizaba esto aprovechando la tecnología para automatizar gran parte del proceso de contratación temporal, ahorrando mucho tiempo y reduciendo la posibilidad de errores humanos. Además, cuenta con una trazabilidad que implementa el sistema que registra todas las acciones realizadas en la herramienta, permitiendo acreditar las acciones realizadas en cualquier momento.

6.3 Líneas futuras

El desarrollo de este proyecto se estima en unas 5000 horas y llevando 2500 prácticamente todas las funcionalidades que requiere el portal de gestión estarían casi terminadas por completo, por lo que casi todo lo que faltaría son funcionalidades del portal de teletramitación y la salida a producción del proyecto una vez finalizado el desarrollo. Por lo que antes se deberá realizar pruebas y asegurarse de que el sistema es seguro para así salir a producción para su uso por parte del público.

7. Bibliografía

7.1 Técnicas:

[1] ".Net", Microsoft, [Online]. Disponible en: <https://dotnet.microsoft.com/es-es/learn/dotnet/what-is-dotnet>
https://learn.microsoft.com/es-es/dotnet/core/introduction?WT.mc_id=dotnet-35129-website

<https://dotnet.microsoft.com/es-es/platform/why-choose-dotnet>

[2] “.Net core”, OpenWebinars, [Online]. Disponible en:

<https://openwebinars.net/blog/que-es-net-core/>

[3] “ASP.Net core”, Microsoft, [Online]. Disponible en: [https://dotnet.microsoft.com/es-](https://dotnet.microsoft.com/es-es/learn/aspnet/what-is-aspnet)

[es/learn/aspnet/what-is-aspnet](https://dotnet.microsoft.com/es-es/learn/aspnet/what-is-aspnet)

[4] “TypeScript”, Typescript Org, [Online]. Disponible en:

<https://www.typescriptlang.org>

[5] “Angular”, Angular IO, [Online]. Disponible en: [https://angular.io/guide/what-is-](https://angular.io/guide/what-is-angular)

[angular](https://angular.io/guide/what-is-angular)

[6] “Entity framework”, Microsoft, [Online]. Disponible en:

<https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/ef/overview>

[7] “Dapper”, Render2Web, [Online]. Disponible en: [https://render2web.com/net-](https://render2web.com/net-6/dapper/que-es-dapper/)

[6/dapper/que-es-dapper/](https://render2web.com/net-6/dapper/que-es-dapper/)

7.2 Herramientas:

[8] “Visual Studio 2022”, Microsoft, [Online]. Disponible en:

<https://visualstudio.microsoft.com/es/vs/>

[9] “SQL Server Management Studio”, Microsoft, [Online]. Disponible en:

[https://learn.microsoft.com/es-es/sql/ssms/sql-server-management-studio-](https://learn.microsoft.com/es-es/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16)
[ssms?view=sql-server-ver16](https://learn.microsoft.com/es-es/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16)