Grado en Ingeniería Informática

Trabajo de Final de Grado

Adaptación tecnológica Add-In de MS Dynamics NAV a MS Dynamics 365 Business Central

Autor:
Céline Pérez Miravete

Supervisor:
Luis Rius Gumbau
Tutor académico:
Carlos Granell Canut

Fecha de lectura: 13 de Julio de 2022
Curso académico 2021/2022

Resumen

Este documento recoge el Proyecto de Final de Grado realizado en *Laberit Sistemas S.L* durante la estancia en prácticas, y describe el proceso de migración o adaptación tecnológica entre dos ERP (*Enterprise Resource Planning*) bien conocidos: MS Dynamics NAV y MS Dynamics 365 Business Central.

El proyecto tiene como objetivo adaptar los desarrollos externos de MS Dynamics NAV, que se encuentran en las librerías Add-In, a las nuevas tecnologías de MS Dynamics 365 Business Central mediante la creación de un servicio web API REST con las funciones extraídas de MS Dynamics NAV. Los servicios web propuestos se adhieren a las directrices funcionales y tecnológicas de la nueva versión del ERP MS Dynamics 365 Business Central.

Palabras clave

MS Dynamics NAV, MS Dynamics 365 Business Central, ERP, Add-In, API REST, servicio web.

Keywords

MS Dynamics NAV, MS Dynamics 365 Business Central, ERP, Add-In, API REST, web service.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El proyecto se realiza en *Laberit Sistemas SL* [13], una empresa dedicada a la consultoría, desarrollo y mantenimiento de sistemas informáticos, que ofrece soluciones para la transformación digital, estrategia tecnológica, ingeniería de sistemas y outsourcing en entornos sanitarios, sector público, industria, puertos y automoción.

La estancia en prácticas y por ende el proyecto se desarrolla en el sector de puertos. Las autoridades portuarias [13] tienen necesidades específicas a la hora de abordar la transformación digital para la modernización de procesos clave de producción y gestión. Para este sector, el uso de aplicaciones ERP o *Enterprise Resource Planning* es bastante usual para la gestión integrada de las actividades diarias de las empresas.

Un ERP o *Enterprise Resource Planning* [6] es un tipo de software que se utiliza para gestionar las actividades empresariales diarias, como la contabilidad, el aprovisionamiento, la gestión de proyectos, la gestión de riesgos, el cumplimiento y las operaciones de la cadena de suministro. Una solución de ERP completa también incluye herramientas de gestión del rendimiento empresarial, que ayudan a planificar, presupuestar, predecir y notificar los resultados financieros de una organización.

En este contexto, la principal motivación del proyecto es la necesidad de llevar a cabo la adaptación tecnológica de un sistema obsoleto a una versión más actualizada. *Laberit Sistemas SL* trabajaba y daba servicio a sus clientes con el ERP MS Dynamics NAV [10], el cual se va a reemplazar prograsivamente por la versión mejorada de este ERP denominada MS Dynamics 365 Business Central [9]. Por lo tanto, la necesidad de este proyecto gira en torno a la adaptación tecnológica de ciertos componentes clave, como los Add-In que se explican más adelante, de MS Dynamics NAV a MS Dynamics 365 Business Central.

Como existen dos proyectos de estancia en prácticas en este mismo contexto y empresa, la Figura 1.1 muestra un esquema de integración para ilustrar el alcance de cada uno de los proyectos. El proyecto de Nieves Cubedo (azul) se enfoca a la migración tecnológica de los

servicios web que ya se encuentran autopublicados en MS Dynamics NAV para así transformarlos en APIs (SOAP, ODATA, REST), incluyendo el rediseño en MS Dynamics 365 Business Central y la actualización de la documentación de los mismos. En cambio, el proyecto de Céline Pérez, descrito en esta memoria, se enmarca en la transformación de las librerías Add-In y dll de MS Dynamics NAV a MS Dynamics 365 Business Central.

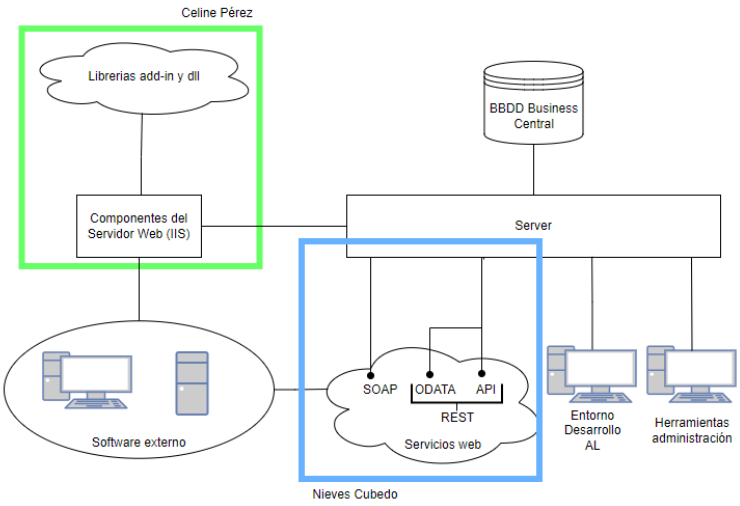

Figura 1.1: Esquema de integración de los dos proyectos.

1.2. Objetivos del proyecto

El objetivo principal del proyecto es realizar un rediseño tecnológico de las librerías Add-In de MS Dynamics NAV. Cuando hablamos de rediseño tecnológico, se trata de un nuevo diseño a través de un servicio web API REST. Las librerías Add-In de MS Dynamics NAV se transforman en un servicio web de tipo API REST en el contexto de MS Dynamics 365 Business Central.

Para poder entender el proyecto es necesario saber que son las librerías Add-In, éstas son un módulo externo de software que añade una funcionalidad de una manera más fácil y además puede añadir conectividad extra a los programas.

Un servicio web encapsula una funcionalidad (por ejemplo, una tarea concreta, acceso a una base de dato remota, etc.) mediante tecnologías basadas en un conjunto de protocolos y estándares de la Web que permiten el intercambio y acceso a datos a través de la Web.

En este caso se utiliza un servicio web de tipo API REST, que se especifica en base a la identificación de recursos relevantes y las operaciones que se pueden realizar sobre éstos. Estas operaciones vienen delimitadas por el protocolo HTTP, el cual describe los estados de los recursos que se transfieren a través de la red ente un cliente y el servicio (servidor).

En cuanto a las API [1], se trata de conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar software de las aplicaciones. Para este proyecto, se ajusta a los límites de la arquitectura REST lo que significa que se ajusta a los principios de diseño de la arquitectura

REST para la comunicación cliente-servidor.

El objetivo principal se puede desglosar en los siguientes subobjetivos:

- Comprensión de la funcionalidad de las librerías Add-In.
- Comprensión de la llamada y la ejecución de esas librerías Add-In dentro de MS Dynamics Business Central.
- Extracción de las funcionalidades de MS dynamics NAV.
- Creación y desarrollo del nuevo servicio web API REST.
- Pruebas del nuevo servicio web API REST.
- Consumo del servicio web API REST a través MS Dynamics 365 Business Central .

1.2.1. Alcance Funcional

Desde el punto de vista funcional, este rediseño funcional y tecnológico debe adaptar los desarrollos externos, los cuales se encuentran en funcionamiento en MS Dynamics NAV y adaptarlos a la nueva plataforma MS Dynamics Business Central. En particular, implica la transformación de las librerías dinámicas dll que se están utilizando de forma directa en MS Dynamics NAV, para que estas se comiencen a usar de manera indirecta a través de un servicio web API REST.

1.2.2. Alcance Organizativo

Desde el punto de vista organizativo, ya que el objetivo de la migración es crear un servicio web API REST donde vamos a encontrar los desarrollos externos de MS Dynamics NAV, todos los departamentos que antes usaban los Add-In en MS Dynamics NAV ahora van a usar el servicio web API REST en MS Dynamics 365 Business Central. Por lo tanto, los usuarios a quienes va dirigido el proyecto, a parte del propio ERP que consume el servicio web, son todos los departamentos que antes ya utilizaban los Add-In en cuestión para sus tareas empresariales.

1.2.3. Alcance Informático

Desde el punto de vista informático, el nuevo servicio web interactuará principalmente con otros servicios y/o sistemas que necesiten la funcionalidad del servicio web. Además, se debe conseguir que tanto el funcionamiento principal como las propiedades de las librerías Add-In, que utilizan bibliotecas de enlaces dinámicos (dll), facilitan la transparencia de redes y el desarrollo de aplicaciones de valor añadido.

1.3. Descripción del proyecto

La base sobre la que el proyecto se fundamenta son las librerías Add-In. Este tipo especial de librerías son un módulo externo de software que facilita añadir una funcionalidad y una conectividad extra a un programa o aplicación. Las librerías Add-In se encuentran incluidas en MS Dynamics NAV y funcionan a través de las biblioteca de enlaces (dll), que son archivos con código ejecutable que se cargan bajo demanda en un software, en nuestro caso se trata de MS Dynamincs NAV, y se encuentran programadas en .Net.

El propósito de este proyecto es la realización de un rediseño tecnológico, que parte de las librerías Add-In que funcionan a través de las bibliotecas de enlaces (dll) MS Dynamincs NAV, para transformarlas en un servicio web API REST. Una vez la migración esté completa, es decir, cuando el servicio web API REST esté implementado y activo en el contexto de MS Dynamics 365 Business Central, las librerías Add-In originales deben de dejar de usarse de forma directa por MS Dynamincs NAV para finalizar el proceso de rediseño.

1.3.1. Tecnologías empleadas

A continuación se incluye una breve descripción de las tecnologías utilizadas a lo largo del desarrollo del proyecto.

Microsoft Dynamics Nav

Microsoft Dynamics Nav [10], o Navision, es básicamente un software de gestión o ERP (Enterprise Resource Planning) que facilita el manejo de diferentes tareas administrativas dentro de la empresa. Permite integrar procesos de diferentes departamentos, como pueden ser la contabilidad, gestión de inventario, compras. Navision ERP funciona mediante módulos válidos para cualquier tarea dentro de la empresa. Por ejemplo, en gestión financiera lleva el control de la contabilidad, los movimientos bancarios y la tesorería.

Este software es necesario para el proyecto ya que desde este ERP se obtienen los Add-In para poder realizar la migración. Se ha instalado una versión básica de escritorio (versión 2013) y la licencia de uso ha sido proporcionada por Laberit Sistemas SL. La aplicación cuenta con varios módulos y una funcionalidad base, la cual es más que suficiente para poder realizar la extracción y comprensión de los Add-In.

Microsoft Dynamics 365 Business Central

Microsoft Dynamics 365 Business Central, o Business Central [9] forma parte de la familia de software de sistemas de planificación de recursos empresariales creado por Microsoft. Más concretamente es el sucesor de Microsoft Dynamics NAV y que basa parte de su desarrollo en el código base del mismo.

Business Central es fundamental para optimizar la lógica del negocio, los tiempos de duración de cualquier proceso y minimizar los costes de cualquier gestión. Business Central conecta a toda la empresa, ya que todos los datos se encuentran en una misma plataforma y por ende toda la plantilla tendrá acceso a ellos. De esta manera, los proyectos de una empresa se gestionan de forma más rápida y todo el personal está al tanto.

Hay diferente formas de adquirir Business Central. Una de ellas es *OnPremise*, que significa que los datos se encuentran almacenados localmente. Otra es la versión *OnCloud*, diseñada en un entorno en la nube, tanto en el propio de Microsoft o en otros sistemas de hosting y almacenamiento en la nube.

Para la realización del proyecto se ha optado por una versión *OnPremise*, que ha sido proporcionado por la empresa a y se ha instalado través de PowerShell en versión administrador.

Visual Studio Code

Visual Studio Code [2] es un editor de código simplificado con soporte para operaciones de desarrollo como la depuración, la ejecución de tareas y el control de versiones. Su objetivo es proporcionar sólo las herramientas que un desarrollador necesita para un ciclo rápido de código-construcción-depuración. La licencia es gratuita para uso privado o comercial.

En el proyecto hemos usado este editor de desarrollo, ya que para realizar la programación en C/AL resulta muy intuitivo y soporta todas las extensiones necesarias, como la integración con Docker y herramientas de depuración para ir probando la funcionalidad que se va desarrollando a lo largo de todo el proyecto.

Visual Studio 2022 (Community Edition)

Visual Studio [5] es un entorno de desarrollo integrado de Microsoft creado para facilitar las tareas del programador y diseñado para crear aplicaciones web. Incluye un editor de texto que soporta diferentes lenguajes de programación tales como C++, C Sharp. Es un IDE gratuito con todas las funciones para estudiantes, colaboradores de código abierto y desarrolladores individuales.

Para la realización del proyecto hemos usado este editor para la implementación completa del servicio API REST.

Swagger

Swagger[15] es una especificación abierta para definir servicios API REST. Un documento Swagger es el equivalente de API REST de un documento WSDL para un servicio web basado en SOAP.

El documento Swagger[15] especifica la lista de recursos disponibles en la API REST y

las operaciones a las que se puede llamar en estos recursos. El documento Swagger especifica también la lista de parámetros de una operación, que incluye el nombre y tipo de los parámetros, si los parámetros son necesarios u opcionales, e información sobre los valores aceptables para estos parámetros. Además, el documento Swagger puede incluir un esquema JSON adicional que describa la estructura del cuerpo de la solicitud que se envía a una operación en una API REST, así como la estructura de datos de las respuestas devueltas de una operación.

En el proyecto ha sido vital el uso de Swagger para poder definir el API REST de la mejor manera, y poder realizar pruebas para asegurar saber si estaba realizado correctamente.

SOAPUI

SoapUI [14] es una herramienta de pruebas de automatización funcional multiplataforma. SoapUI es una herramienta gratuita y de código abierto y ha sido diseñada para ayudar a probar APIs como interfaces SOAP y REST para asegurar la interoperabilidad de diferentes aplicaciones.

En el proyecto nos ha ayudado para realizar más pruebas entorno a la API, y además nos hemos asegurado de que el proyecto de la API estaba bien realizado y se encontraba en la URL correcta.

Microsoft Project

Microsoft Project (MSP)[12] es la herramienta ideal para planificar y controlar proyectos. Este software de Microsoft fue lanzado en 1988 y desde entonces no para de evolucionar y mejorar. Esta herramienta favorece la productividad entre el equipo de trabajo y reduce el tiempo para crear calendarios y organizar tareas. Con Microsoft Project se evitan retrasos y asegurar que los proyectos se entreguen a tiempo y dentro de lo previsto.

Con este software hemos creado el diagrama de Gantt para realizar una planificación temporal de los diferentes hitos y tareas del proyecto en la plazo estipulado.

Git

Es una herramienta de control de versiones de código de forma distribuida, es muy potente y no depende de un repositorio central. Git [8] se caracteriza por un sistema basado en ramas, destinadas a provocar proyectos divergentes a partir de un proyecto principal, para hacer experimentos o para probar nuevas funcionalidades. Las ramas pueden tener una línea de progreso diferente de la rama principal donde está el core de nuestro desarrollo. En algún momento podemos llegar a probar algunas de esas mejoras o cambios en el código y hacer una fusión a nuestro proyecto principal, ya que todo esto lo maneja Git de una forma muy eficiente.

Gracias al control de versiones que nos proporciona Git, a través del sistema basado en ramas hemos podido crear varias ramas, siempre dejando el código que daba buenos resultados

en la rama principal, y haciendo diferentes pruebas con las otras ramas para así poder realizar todos los cambios necesarios sin tener que tocar la rama principal.

1.4. Estructura de la memoria

En esta parte de la memoria podemos encontrar al detalle como se encuentra estructurada.

En el *Capítulo 2* podemos encontrar la *Planificación del proyecto* en la cual definimos cual es la metodología que se ha usado en el desarrollo del proyecto, así como la planificación teniendo en cuenta la metodología escogida, la estimación de recursos y costes del proyecto y por último el seguimiento del proyecto.

En el *Capítulo 3* podemos encontrar el *Análisis y diseño del sistema* en el cual entramos más en detalle sobre estos dos puntos anteriormente nombrados.

En el *Capítulo 4* podemos encontrar la *Implementación y las pruebas*, se detallan los procedimientos sobre la implementación en Microsoft Dynamics 365 Business Central.

En el *Capítulo 5* podemos encontrar la *Conclusión*, donde se presentan varios ámbitos, el formativo, el profesional y el personal.

Capítulo 2

Planificación del proyecto

2.1. Metodología

La metodología que se ha utilizado para desarrollar el proyecto ha sido la metodología predictiva. Cuando usamos este tipo de metodología, nos referimos en particular a la metodología en cascada [16], donde las tareas del proyecto se van realizando de manera secuencial y se puede, si fuera necesario, retroceder a alguna de ella. Al final de cada una de las fases, las cuales están compuestas por tareas, se debe realizar una revisión.

Las fases que componen el proyecto son las siguientes:

- Formación básica: Es necesaria una formación básica tanto en contabilidad, como finanzas, como en el propio ERP, para poder conocer cómo funciona en profundidad y además conocer los diferentes circuitos que se pueden realizar. Cuando hablamos de circuitos nos referimos a los diferentes pasos que hay que dar para realizar una venta o una compra dentro del ERP. Asimismo es necesario conocer la funcionalidad que el ERP nos aporta para poder llevar a cabo las tareas encomendadas en el proyecto.

- Planificación: En esta parte el objetivo es realizar un estudio de las bases del proyecto, definir los objetivos, el alcance, redactar la propuesta técnica y realizar el diagrama de Gantt del proyecto.

- Análisis: En cuanto a esta fase, la meta es lograr unos conocimientos sobre todo lo que se encuentra en MS dynamics NAV y MS Dynamics 365 Business Central, para poder lograr el rediseño y transformación a un servicio web API REST.

- Diseño e implementación: Diseño y creación en lenguaje C/AL de las páginas finales que se van a mostrar en MS Dynamics 365 Business Central y del servicio web API REST.

- Pruebas: Realización de las pruebas tanto de bajo como de alto nivel.

2.2. Planificación

Teniendo en cuenta que la metodología escogida ha sido la metodología predictiva, es decir el modelo en cascada, la realización de la planificación temporal del proyecto se llevó a cabo con la herramienta Microsoft Project, donde se detallaron las diferentes fases del proyecto. Las figuras 2.1 y 2.2 muestran la planificación temporal del proyecto, incluyendo las fases, tareas contenidas y duración temporal de cada tarea.

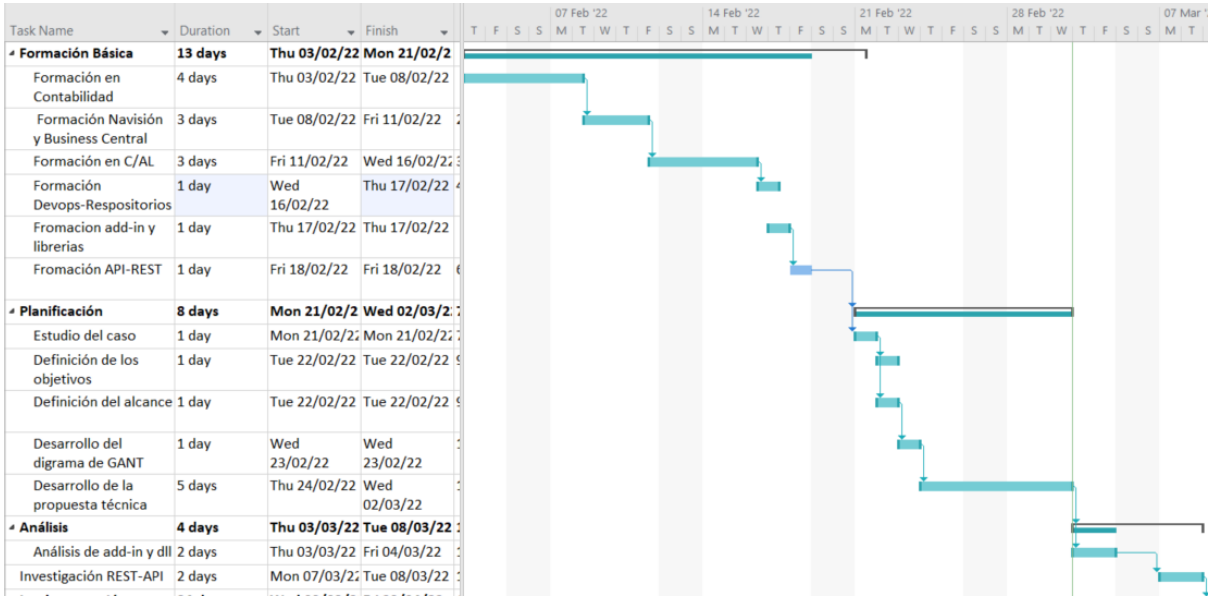

Figura 2.1: Planificación.

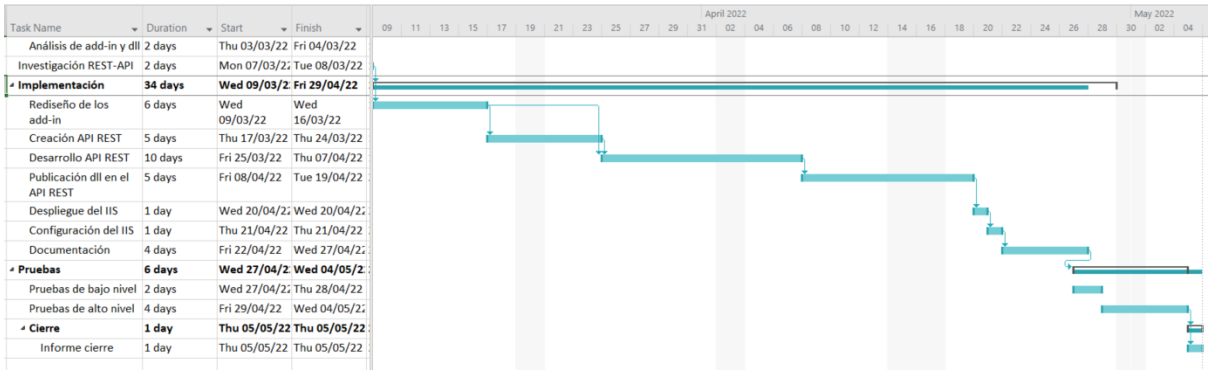

Figura 2.2: Continuación de la planificación.

2.3. Gestión de riesgos

En esta sección vamos a detectar los riegos que pueden ocurrir y a analizarlos. Este apartado es uno de los más importantes ya que la previa detección de riesgos puede ayudarnos a garantizar el éxito del proyecto.

La gestión de riesgos está administrada por los desarrolladores ya que éstos poseen el conocimiento necesario como para poder solventar los problemas que puedan surgir.

2.3.1. Identificación de riesgos

En este apartado vamos a conocer los diferentes riesgos que puedan surgir durante la realización de proyecto de prácticas. A continuación se indican los riesgos que hemos encontrado en el proyecto

- R01: Desconocimiento base del funcionamiento de los ERP.
- R02: Falta de experiencia en tareas de implementación.
- R03: Cambios en los requisitos.
- R04: Estimación inadecuada del tiempo de ejecución.

2.3.2. Análisis de riesgos

En este apartado haremos un análisis de los diferentes riesgos y se va a detallar la descripción, la importancia de cada riesgo, y qué o quién lo origina.

- R01: Desconocimiento base del funcionamiento de los ERP. No se tiene experiencia previa trabajando con ERP. La importancia es alta. Lo origina el alumno por falta de experiencia.

- R02: Falta de experiencia en tareas de implementación. Se tiene experiencia pero no en está área en concreto (implementación en ERP) por lo que es un riesgo que hay que asumir y tener en cuenta. Importancia alta. Lo origina el alumno por falta de experiencia

- R03: Cambios en los requisitos. Los cambios pueden traer des-estabilidad a la hora de el desarrollo del proyecto, ya que éstos sirven de guía para el desarrollo de todo el proyecto. Importancia alta. Importancia alta. Lo originan los cambios imprevistos y de última hora para añadir o quitar funcionalidades.

- R04: Estimación inadecuada del tiempo de ejecución. Este riesgo puede llevar problemas a lo largo del proyecto ya que se irían acumulando tareas ya que la estimación es inadecuada. Importancia alta. Lo origina la mala comprensión de las técnicas de planificación.

2.4. Estimación de recursos y costes del proyecto

Este apartado identifica y describe los recursos necesarios, así como su coste, para el desarrollo del proyecto.

Comenzaremos por enumerar los diferentes materiales e infraestructura necesaria para la realización del proyecto, incluyendo también personal necesario (programador Junior), todo en función del periodo de tiempo en el que se ha llevado a cabo el proyecto. El total de horas del proyecto en la empresa son 300 horas, por lo que toda la estimación se basará en torno a esa duración.

En primer lugar, contamos con un ordenador portátil HP 15S-FQ4079NS 15.6", con las siguientes prestaciones:

- Procesador. Intel Core i5-1155G7.
- RAM: 8 GB.
- Disco: 512 GB SSD.
- Gráfica: Intel Iris Xe Graphics.

También contamos con otros accesorios:

- Monitor LG 22M47VQ-P con pantalla IPS LED 16:9.
- Ratón logitech M185.
- Teclado logitech k270.

Por otra parte, hay que contemplar la licencia que nos ha aportado *Laberit Sistemas S.L*, para poder hacer uso tanto de MS Dynamics NAV como de MS Dynamics 365 Business Central. La licencia de MS Dynamics NAV se encuentra por 58 euros, usuario por mes, si hablamos de un licencia *essential*, y también se encuentra por 84 euros, usuario al mes, si es un licencia *premium*. En nuestro caso hemos utilizado la versión *essential*. Por otro lado, la licencia de MS Dynamics 365 Business Central, disponible también en las dos versiones anteriores, con un coste de 59 euros usuario por mes (*essential)* y 84,30 euros usuario por mes (*premium*).

En cuanto a los recursos humanos, debemos añadir la labor de analista, el cual desarrolla tareas de recogida de datos, análisis de los mismos, documentación posterior, con un coste de 20 euros la hora. Por otra parte, debemos añadir la labor del programador, el cual se encarga básicamente de realizar la programación al completo. Al tratarse de un programador junior, se contempla un coste de 10 euros la hora. Finalmente, no nos podemos olvidar del supervisor del programador, el cual podríamos estimar en un coste de 15 euros la hora.

Con toda esta información podemos realizar la siguiente estimación del coste total del proyecto, resumida en los siguientes Cuadros 2.1 y 2.2.

Estimación del proyecto			
Recursos	Unidades	Coste	Subtotal
Portátil HP	1	599,99€	599,99€
Monitor	1	119,99€	119,99€
Teclado	1	28,50€	28,50€
Ratón	1	10,74€	10,74€
TOTAL MATERIALES			**759,22€**

Cuadro 2.1: Materiales informáticos

Estimación del proyecto			
Recursos	Cantidad Horas	Coste	Subtotal
Licencia NAV	300 horas	58€	174€
Licencia BC	300 horas	59€	177€
Analista	300 horas	20€	6000€
Programador Junior	300 horas	10€	3000€
Supervisor del programador	300 horas	15€	4500€
TOTAL INFRAESTRUCTURA Y PERSONAL			**13.851€**

Cuadro 2.2: Infraestructura y personal

El total de la estimación del proyecto, es la suma de las dos cantidades totales (Cuadros 2.1 y 2.2) es de **14.610,22€**

2.5. Gestión de recursos humanos

En todos los proyectos hay que definir de manera exacta cual es el rol de cada una de las personas implicadas para evitar conflictos. Los roles definidos son:

- Gerente: Se encarga de dirigir el proyecto y dictaminar las diferentes tareas que se van a llevar a cabo a largo plazo, y también es quien va a programar las reuniones para poder realizar un seguimiento del proyecto.
- Supervisor: Persona que se encarga de delegar y repartir los diferentes hitos y tareas a los programadores, además también es el encargado de controlar a los programadores.
- Programador: Persona que se encarga de realizar las tareas que le ha encomendado el supervisor.

En el siguiente Cuadro 2.3 podemos ver con más claridad quien es el personal que desempeñará cada rol.

Gestión de recursos humanos	
Roles	Personal
Gerente	Luis Rius Gumbau
Supervisor	Luis Rius Gumbau
Programador	Celine Pérez Miravete

Cuadro 2.3: Roles del personal

2.6. Seguimiento del proyecto

Las principales herramientas de seguimiento del proyecto han sido las reuniones semanales, reuniones con el supervisor y los informes quincenales.

Las reuniones semanales se realizaban con la intención de hacer trabajo nuevo todas las semanas y corregir el trabajo anterior si había fallos, o trasladar las dudas a los supervisores, en cuanto al trabajo realizado durante la semana. También se comentaban los avances logrados durante la semana. Si antes de la siguiente reunión semanal surgían dudas, se realizaba una reunión con las dudas tanto con el supervisor como con los compañeros.

Con respecto a los informes quincenales, éstos tienen como principal cometido llevar un control más a largo plazo e informar al tutor de prácticas. Todos y cada uno de los informes han sido elaborados y liberados puntualmente, recibiendo las oportunas correcciones por parte del tutor.

Por último las reuniones mensuales con el supervisor de la empresa, en las que se mostraban los diferentes avances del proyecto y se hacían las pertinentes recomendaciones.

Por lo tanto, el seguimiento del proyecto es una labor que se ha llevado a cabo durante toda la realización del mismo, y que comenzó a la vez que el proyecto.

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

En este capítulo se explica la fase del análisis que es posiblemente una de las fases más importantes del proyecto, ya que especifica los requisitos, la funcionalidad a desarrollar y las diferentes técnicas y herramientas que se han usado a lo largo del proyecto. Previamente se ha realizado un estudio y formación de la herramienta MS Dynamics 365 Business Central, como parte de la fase de formación básica.

La tarea de análisis arranca desde el principio del proyecto, con la propuesta técnica, describiendo y concretando los objetivos con el supervisor. También con las diferentes reuniones semanales en las cuales definíamos poco a poco las características que iba a tener el proyecto.

Para poder realizar un correcto análisis es necesario conocer la herramienta con la que vamos a trabajar, por lo que es necesario un proceso de formación. Debemos tener una noción básica tanto en MS Dynamics 365 Business Central, que actúa de cliente, éste consumirá el servicio web API REST. Además debemos tener una noción de MS Dynamics NAV, ya que es desde donde se va a sacar la funcionalidad de los Add-In.

Se ha recibido formación en cuanto a contabilidad y finanzas en torno a MS Dynamics 365 Business Central, para poder conocer las funcionalidades, además del entorno de diseño en Microsoft Dynamics NAV. Se han llevado a cabo ejercicios en el lenguaje C/AL que es el lenguaje en el que el ERP está desarrollado, así como informes y gestión de requisitos a través de la herramienta Azure DevOps Server y formación básica sobre servicios web y tecnologías asociadas.

Debemos sumar a lo anterior, la revisión del sistema en funcionamiento sobre el cual se debe tener conocimiento y control. En este caso los requisitos que necesitamos son los Add-In que se van a utilizar para poder hacer la migración y poder convertirlo en un servicio web. Una vez reconocemos y sabemos donde se encuentran los Add-In, podemos decir que la parte de toma de requisitos se da por finalizada.

3.1.1. Definición de requisitos

Como se ha comentado anteriormente, el proyecto se basa en la migración de los Add-in en un servicio web API REST. En este contexto, los requisitos que se han definido para la consecución de dicho servicio web API REST son los siguientes:

- El servicio debe poder realizar la mismos cálculos que el Add-in original.
- El resultado del servicio web REST API debe coincidir con el resultado de la ejecución de los Add-ins.
- El servicio debe desplegarse sobre Business Central.
- El servicio debe poder ser accesible y consultable desde cualquier cliente software externo.

3.2. Diseño de la arquitectura del sistema

La arquitectura del sistema en la que vamos a realizar el proyecto, presenta una base sólida debido a que el ERP sobre el que vamos a trabajar ya muestra una arquitectura robusta. Tanto el ERP Microsoft Dynamics NAV, del cual vamos a extraer los Add-In, como MS Dynamics 365 Business Central del cual vamos a consumir el servicio web API REST, muestran una arquitectura de calidad y que permiten una buena interacción.

En primer lugar, Microsoft Dynamics NAV es el ERP base del cual vamos a extraer las funcionalidades básicas para poder realizar la migración. Este ERP es la versión antigua, la cual está comenzando a estar obsoleta y necesitamos hacer que toda la funcionalidad que se encuentra desarrollada a través de sus librerías Add-In se pueda integrar en el nuevo y mejorado ERP.

Para poder realizar esta integración, se requiere en primer lugar un análisis exhaustivo de los Add-In para poder saber que funcionalidades siguen teniendo un uso habitual en MS Dynamics 365 Business Central.

En segundo lugar, tenemos el nuevo y mejorado ERP, MS Dynamics 365 Business Central, vamos a hacer que llame al API externo, y en el que se encuentran los Add-In con la funcionalidades que se van a usar en el ERP.

Para poder hacer esto posible contamos con una página que muestra los datos que se extraen de los Add-In que se denomina “PREFIX NumeroyCad” y otra en la que hacemos una llamada al API y que se denomina “PREFIX DemoApiPage”. Estas páginas están controladas a través de CodeUnits, las cuales se encargan de agrupar diferentes procedimientos en un objeto, donde se debe de desarrollar el código para que pueda aparecer en las páginas.

En la Figura 3.1, se puede observar el estado en el que podíamos encontrar el sistema antes de la migración y el estado del sistema después de la migración.

La página “PREFIX NumeroyCad” la podemos encontrar en la primera parte de la Figura 3.1, es decir en el “Antes” y la página “PREFIX DemoApiPage” la podemos encontrar en la segunda parte de la Figura 3.1, es decir una vez la migración se ha completado. Se accede al servicio web a través de la página “PREFIX DemoApiPage”.

Puede parecer a simple vista parecido y pensar que no es necesario crear un servicio web, pero los Add-In ya no son útiles y se están quedando obsoletos, por eso es necesaria esta migración, y la forma menos laboriosa de transformar los Add-In es a través de un servicio web API.

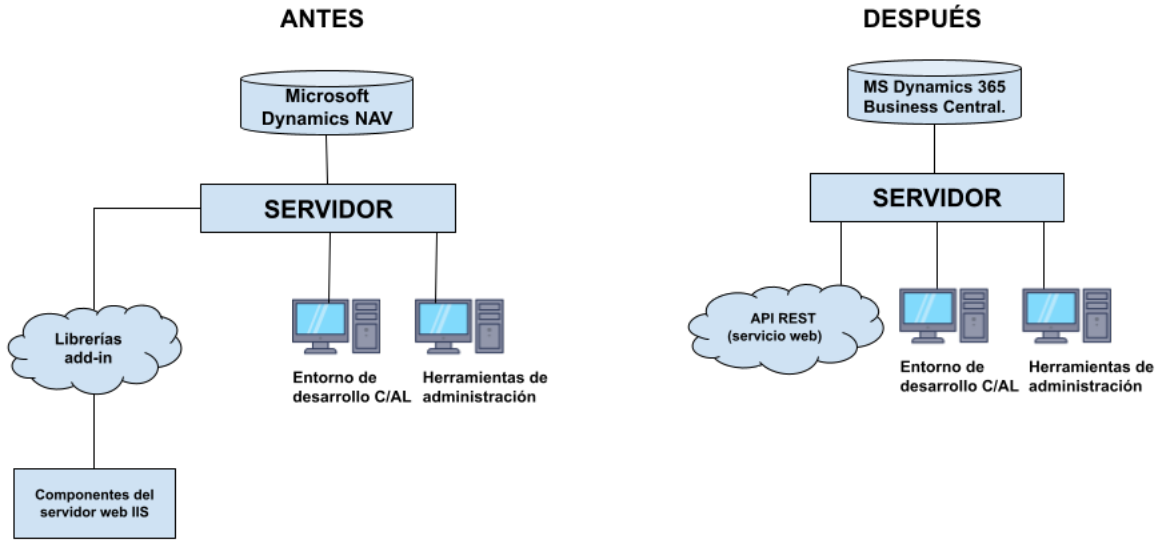

Figura 3.1: Antes y después de la migración.

3.3. Diseño de la interfaz

El diseño de una buena interfaz gráfica es esencial tanto como para el usuario como para el programador. Para el programador es fundamental para poder ver como el trabajo realizado va tomando forma y comprobar como sería la experiencia del usuario. Por parte del usuario también es imprescindible ya que una interfaz con un diseño visualmente atractivo, simple de usar e intuitivo, asegura una buena respuesta por parte del usuario. Un buen diseño de la interfaz es lo que diferencia un buen producto de software de uno que no lo es.

La interfaz que se utiliza para todo el proyecto es la que nos proporciona MS Dynamics 365 Business Central, que proporciona las siguientes cualidades:

- Uso de la interfaz en diversas plataformas ya que está completamente equipada.
- Interfaz de usuario personalizable, con diferentes fuentes, vistas y funciones.
- Evita la sobrecarga de información y la carga memorística.

El diseño que se ha desarrollado en MS Dynamics 365 Business Central se ha dividido en tres páginas. Por un lado podemos encontrar la primera página y principal en la que realizamos la extracción de los Add-In que se encuentran en Microsoft Dynamics NAV y pasamos a añadirlos a MS Dynamics 365 Business Central, tal como muestra la Figura 3.2.

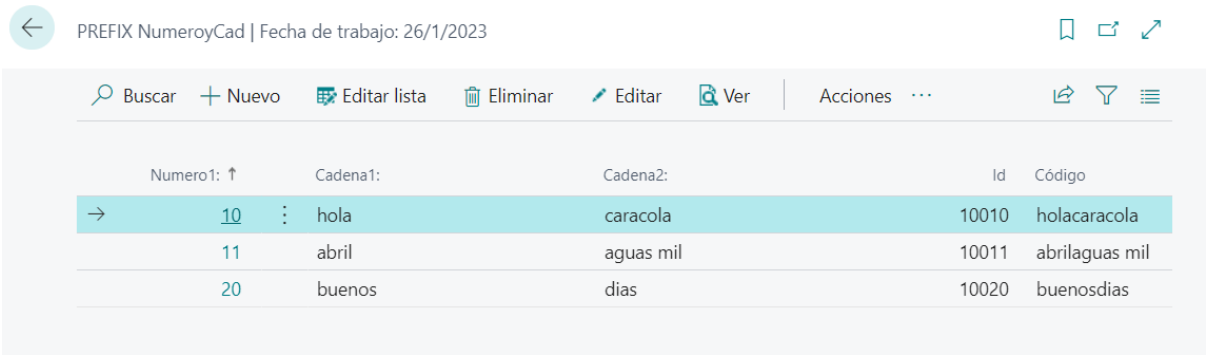

Figura 3.2: Diseño de la extracción de los Add-In.

Por otro lado, la segunda página en la Figura 3.3 permite la introducción de los valores de las parámetros de entrada de los Add-In, para que aparezca el resultado de la ejecución de los Add-In en la página principal (Figura 3.2). Estas dos páginas están relacionadas ya que sin una introducción de campos (al pulsar "Nuevo" en la Figura 3.2 nos redirige a la segunda página mostrada en la Figura 3.3) no se puede hacer una llamada a los Add-In.

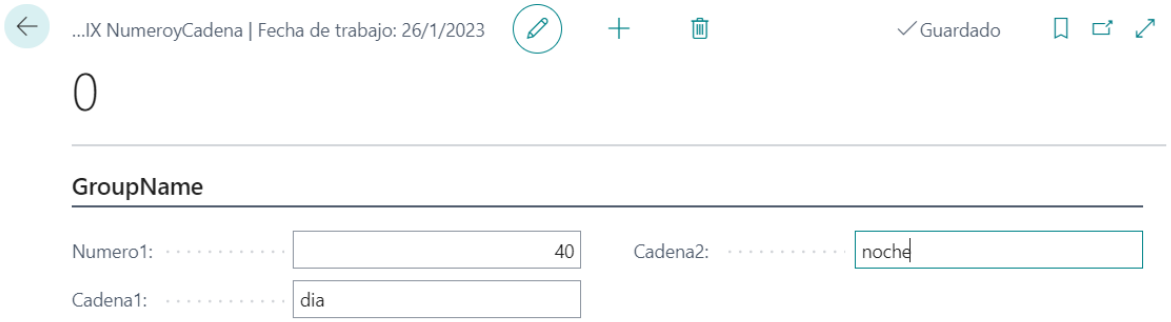

Figura 3.3: Campos a rellenar.

La tercera y última página del diseño se encarga de realizar una llamada a la API externa y se consume a través de MS Dynamics 365 Business Central. Como podemos observar en la Figura 3.4, vemos la llamada a la API externa y en cuanto la llamada se ha realizado con éxito aparece un mensaje que nos indica el atributo *Id* y el atributo *Code*, éstos se integran en MS Dynamics 365 Business Central.

La relación que existe entre las Figuras 3.2 y 3.4 compara la situación inicial, antes de la migración, utilizando Microsoft Dynamics NAV (Figura 3.2) y la nueva situación tras la migración (Figura 3.4) a través de un servicio web desplegado en MS Dynamics 365 Business Central.

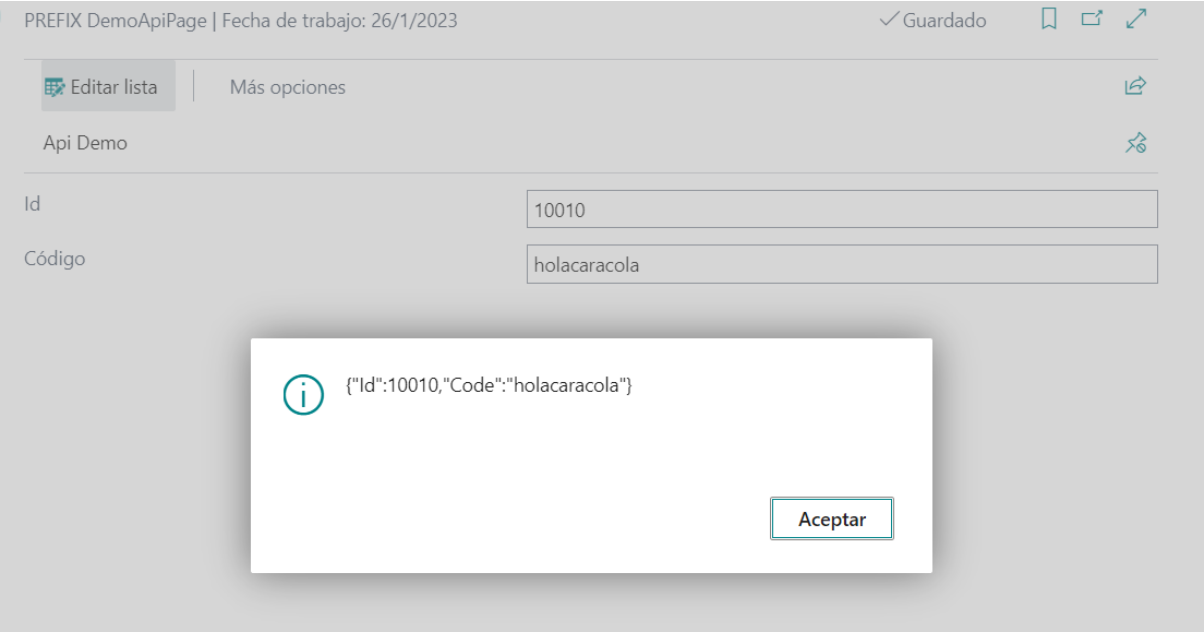

Figura 3.4: Página de la API.

En este capítulo se ha introducido brevemente el proceso de migración, cubriendo el análisis, diseño del sistema y de las interfaces de usuario. El siguiente capítulo profundiza en la implementación del proceso de migración de forma más extensa.

Capítulo 4

Implementación y pruebas

En este capítulo se va a explicar el trabajo de programación realizado, detallando cada parte del proyecto, así como las diferentes decisiones tomadas a lo largo del mismo. Asimismo se explican la instalación de los ERP utilizados y se detallan las pruebas de verificación y validación realizadas.

4.1. Instalación de los ERP

Para poder realizar todo el proyecto es necesaria la correcta instalación de los dos ERP. En particular, la primera parte del proyecto en la cual hacemos la extracción de los Add-In, se requiere de Microsoft Dynamics NAV, mientras que en la segunda parte, las mejoras introducidas para consumir el API REST requieren de MS Dynamics 365 Business Central.

4.1.1. Instalación de Microsoft Dynamics NAV

En primer lugar para la instalación de Microsoft Dynamics NAV se ha seguido la guía de instalación que la empresa nos ha otorgado, es bastante simple, solo hay que seguir los pasos del documento.

El primer paso es descargarse el link para obtener la versión de la CRONUS española, una vez esta versión está instalada simplemente descomprimimos el archivo y seleccionamos el "setup.exe" y elegimos la opción de programadores y dejamos que se instale.

Una vez esta parte está instalada hay que configurar los puertos donde vamos a levantar el ERP para que no se produzcan colisiones con los puertos que vienen predefinidos por si ya los tenemos en uso.

Después hay que crear una instancia para la base de datos. Como en nuestro caso no teníamos ninguna, el propio ERP instala una de base de datos por defecto SQL Server versión 2017.

La interacción con la base de datos se puede realizar mediante la interfaz gráfica o mediante comandos en la consola. La instancia creada se ha denominado *bc13*.

También hay que tener en cuenta la licencia, que ha sido proporcionada por parte de la empresa y que era válida durante los meses en los que se ha desarrollado el proyecto.

Por último, queda la instalación del "AL development environment", que es el entorno de programación con el que el ERP trabaja y funciona. Además, la configuración completa del ERP requiere de la instalación de la herramienta de administrador del servidor, el propio servidor y sus componentes y de la integración de la base de datos SQL Server en el ERP.

4.1.2. Instalación de MS Dynamics 365 Business Central

Para realizar la instalación de este ERP el proceso ha sido un poco diferente ya que vamos a trabajar con una versión *On premise*. En este caso se requiere la instalación de *Docker*, que ha sido proporcionado por la empresa, y se instala con un script (también proporcionado por la empresa) a través de PowerShell en su versión de administrador.

Un *Docker* [4] es un proyecto de código abierto que nos ayuda a automatizar el despliegue de aplicaciones dentro de contenedores. Los contenedores [3] tienen como finalidad agrupar y aislar entre sí aplicaciones que se ejecutan sobre un mismo sistema operativo. En nuestro caso solo contamos con uno que hemos denominado *bc19v2* y que podemos controlar a través de una página que se denomina "portainer.io". Desde esta página podemos saber si el contenedor está o no está activo y monitorizar el estado del *Docker* por completo.

Una vez la instalación del *Docker* se ha completado, simplemente hay que poner las credenciales que han sido proporcionadas por la empresa e introducir la licencia que también han sido proporcionada por la empresa.

Cuando esta parte se ha completado las imágenes del *Docker* se descargan en el ordenador y ya podemos acceder al ERP a través de internet con el enlace que se crea durante el proceso de instalación, en nuestro caso se trata de *https://bc19v2*.

4.2. Preparación del entorno de desarrollo

Para que el desarrollo se realice con la mayor fluidez posible vamos a necesitar diferentes herramientas que ya hemos mencionado en el Capítulo 1, en el apartado 1.3.1 de tecnologías empleadas.

En primer lugar vamos a necesitar Visual Studio Code, con las siguientes extensiones [7] para que el entorno de desarrollo en el cual trabaja el ERP MS Dynamics 365 Business Central pueda estar cubierto al completo.

- AL Extension Pack: Se trata de una extensión que une todo lo necesario para poder desarrollar en C/AL.

- AL Object Designer: Con esta extensión obtenemos un listado de los objetos que tenemos cuando descargamos los símbolos del sistema. Devolviendo así una capacidad de visión general de todas las tablas, paginas y codeunits que hay.

- Waldo's CRS AL Language Extension: Es una extensión que facilita la programación en C/AL, proporciona diferentes herramientas para poder renombrar, organizar y además nos ofrece diferentes atajos.

En segundo lugar, vamos a usar Visual Studio para realizar el servicio web API REST. En Visual Studio vamos a tener que usar los Add-In que hemos sacado de Microsoft Dynamics NAV para poder llevar a cabo y construir el API REST.

Finalmente, tenemos dos herramientas que sirven para probar el servicio web, Swagger y SoaupUI. Estas dos herramientas hacen la misma función pero de distinta manera. Las dos son de gran ayuda para el desarrollo del proyecto.

4.3. Detalles de la implementación

En esta sección contaremos en profundidad el desarrollo del proyecto, dividiéndolo en fases para que se pueda entender de la mejor manera posible.

4.3.1. Desarrollo de los Add-In

Para la realización de esta parte tenemos que hacer un análisis de los Add-In que se encuentran, como en otras ocasiones hemos mencionado, en MS Dynamics NAV. En nuestro caso y para poder simplificar la explicación y que sea todo más entendible, hemos escogido un Add-In simple, que devuelve un *Id* que se le suma 10000 al número de entrada y un *Code* que devuelve la concatenación de dos cadenas de texto de entrada (Figura 4.1).

```
public Resultado GetResultado(int numero, string cadena1,string cadena2)
{
    // Debug, informamos objeto y lo devolvemos
    log.Debug(string.Format("Numero: {0}, cadena1: {1}, cadena2: {2}", numero, cadena1, cadena2));
    Resultado resultado = new Resultado();
    resultado.Id = 10000 + numero;
    resultado.Code = cadena1 + cadena2;
    return resultado;
}
```

Figura 4.1: Add-In.

Una vez que tenemos claro cuales son las funcionalidades del Add-In, el siguiente paso consiste en realizar el desarrollo en *C/AL* para que veamos como se consume desde Business Central.

Para consumir el Add-In debemos crearnos un proyecto en Visual Studio Code y abrirlo en forma de *workspace*, es decir, en forma de espacio de trabajo personal. En este proyecto, como se muestra en la Figura 4.2, debemos crear una carpeta para guardar los Add-In que se debe de llamar '.netpackages'. En ella creamos una subcarpeta ('demoaddin' en nuestro caso) para poder incluir los Add-In que queramos en este momento.

Si en un futuro quisiéramos agregar más Add-In deberíamos crear nuevas subcarpetas para que cada Add-In estuviera organizado de forma correcta y así se evitan posibles errores o confusiones entre los Add-In.

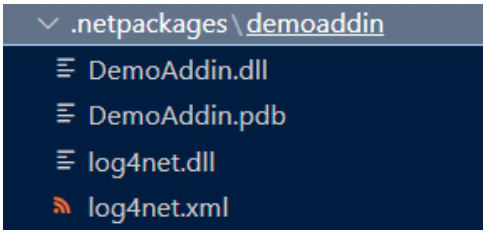

Figura 4.2: Ejemplo de carpetas para guardar los Add-In en Visual Studio Code.

Para que el compilador reconozca la ubicación de los Add-In debemos crear una carpeta en nuestro ordenador personal donde se encuentren todos los Add-In que vamos a utilizar, como se muestra en la Figura 4.3.

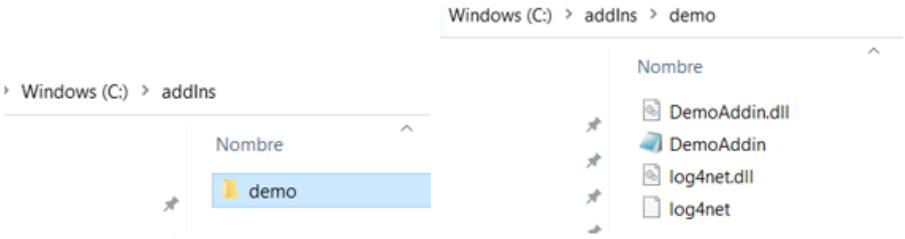

Figura 4.3: Ejemplo de carpetas para guardar los Add-In en el ordenador.

Una vez guardados los Add-In en nuestro ordenador, debemos añadir la ruta también en Visual Studio Code en la carpeta que contiene el ensamblado a la propiedad "al.assemblyProbingPaths" (Figura 4.4). Se accede a esta propiedad abriendo la paleta de comandos Ctrl+Shift+P en Visual Studio Code, buscando por "Configuración de usuario" o "Configuración del espacio de trabajo" y luego especificando la ruta en "al.assemblyProbingPathsconfiguración".

```
"al.assemblyProbingPaths":[
    "./.netpackages",
    "C:/Windows/assembly/",
    "C:/addIns/demo"
]
```

Figura 4.4: Ruta en Visual Studio Code.

Anteriormente hemos introducido los Add-In en Visual Studio Code a través de la ruta que contiene el ensamblado en "al.assemblyProbingPaths" (Figura 4.4) y además los hemos introducido en nuestro ordenador, creando carpetas como hemos mostrado (Figura 4.3).

El último sitio donde tenemos que introducir los Add-In es en el script que se usa para instalar Microsoft Dynamics 365 Business Central. En este script encontramos un apartado que se denomina "addinsFolder" y ahí es donde debemos poner la ruta, esta ruta debe especificar donde se encuentran los Add-In en nuestro ordenador.

Una vez realizada toda esta parte debemos tener en cuenta que vamos a tratar con Add-In y estos son librerías "dll" lo que significa que están programados en .NET. En otros términos, solo podemos trabajar en .NET si está disponible en las instalaciones del Docker. Si se desea utilizar esta funcionalidad, se debe configurar "target: OnPrem" en el fichero app.json.

Cuando hayamos finalizado la configuración tanto en Visual Studio Code como en nuestro ordenador, llega la parte de hacer las CodeUnits para poder mostrar la información en las páginas de MS Dynamics 365 Business Central.

En primer lugar hay que comenzar a declarar los paquetes DotNet para poder llamar a los Add-In y poder hacer uso de los métodos. Vamos a crear un archivo que se llame DotNet y vamos a introducir el contenido que se muestra en la Figura 4.5:

```
dotnet
{
    assembly(DemoAddin)
    {
        1 reference
        type(DemoAddin.Functions; func) { }
        type(DemoAddin.Resultado; res) { }
    }
    assembly(mscorlib)
    {
        type("System.String"; TestString) { }
    }
}
```

Figura 4.5: Paquetes DotNet.

Para poder llamar a las funciones que se encuentran dentro del Add-In debemos llamar primero a las clases y darles un nombre para poder trabajar en el resto del código de una forma

sencilla, sin tener que llamar al Add-In y a la clase cada vez que queramos usar algún método de la misma.

En la Figura 4.5 se puede ver como hemos realizado la llamada al Add-In y a la clase, a continuación encontramos una breve explicación:

- **DemoAddin.Functions; func** : De esta manera estamos llamando al Add-In "DemoAddin" y dentro del Add-In llamamos a la clase "Functions" y le asignamos el nombre "func".
- **DemoAddin.Resultado; res** : De esta manera estamos llamando al Add-In "DemoAddin" y dentro del Add-In llamamos a la clase "Resultado" y le asignamos el nombre "res".

Después de haber nombrado las funciones que vamos a utilizar, podemos pasar a crear la CodeUnit y la página que vamos a usar para poder llamar al método y que se haga visible en MS Dynamics 365 Business Central, como podemos observar en la Figura 4.6. A continuación, deberíamos llamar al método que hemos creado en la CodeUnit en una página a través de un método Action (Figura 4.7).

Una vez realizado todo esto, podemos ver el resultado en MS Dynamics 365 Business Central, nos damos cuenta de que suma 10000 al número que se le pasa y concatena las dos cadenas de entrada. La Figura 4.8 muestra el resultado que esperábamos para las variables *Id* y *Código* del Add-In definido en la Figura 4.1.

```
codeunit 50301 "PREFIX Temp1"
{
    1 reference
    procedure dotnet()
    var
        data: Record "PREFIX NumeroyCadena";
        funcion: DotNet func;
        resultado: DotNet res;

    begin
        funcion := funcion.Functions();
        resultado := resultado.Resultado();

        resultado := funcion.GetResultado(data.Numero1, data.Cadena1, data.Cadena2);

    end;

}
```

Figura 4.6: CodeUnit de los Add-In.

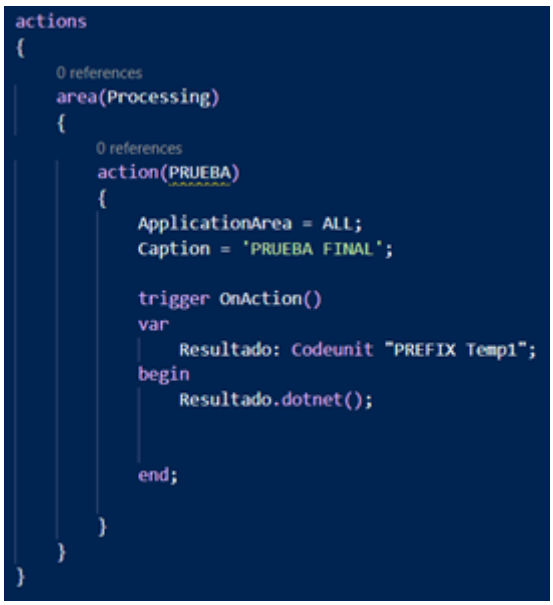

Figura 4.7: Las acciones de la página.

Numero1:	Cadena1:	Cadena2:	Id	Código
10	hola	caracola	10010	holacaracola
→ 11	abril	aguas mil	10011	abrilaguas mil

Figura 4.8: Resultado del Add-In.

4.3.2. Creación y desarrollo del API REST

Para la correcta creación y desarrollo del API hemos instalado Visual Studio 2022, *Versión 17.1*. Una vez descargado e instalado debemos crear un proyecto de tipo *Aplicación Web de ASP.NET Core Model-Vista-Controlado*, que define una plantilla de proyecto para crear una aplicación ASP.NET Core con controladores y vistas de ASP.NET Core usando el patrón Modelo-Vista-Controlador (MVC) con ejemplos incluidos. Este tipo de proyectos se puede usar para la creación de servicios REST.

En el proyecto, dividiremos y desarrollaremos las diferentes partes del Add-In según el patrón MVC. Luego, pasaremos a usar la herramienta Swagger (ver capítulo 1) y haremos la primera prueba de integración en el proyecto creado en MS Dynamics 365 Business Central.

Antes de hablar del patrón MVC y concretar su estructura y desarrollo, debemos tener en cuenta que debemos insertar de la manera correcta los Add-In en este proyecto. Para ello, cuando creamos proyectos en Visual Studio, éstos se dividen en soluciones. Podemos encontrar las siguientes soluciones; primera solución es el DemoAddIn, después encontramos DemoAPIREST y por último tenemos la solución DemoTest.

Para que la creación y desarrollo del API sea entendible pasaremos a explicar las tres soluciones que se han desarrollado.

DemoAddIn

En esta primera solución, debemos incluir los Add-In y los componentes necesarios para que funcionen de manera correcta. Tal como se añadieron los Add-In en Visual Studio Code (Figura 4.2), en esta solución hay que añadirlos para que Visual Studio pueda trabajar con los Add-In sin ningún tipo de problema.

Como podemos observar en la Figura 4.9, nos damos cuenta que el Add-In se encuentra dentro de la solución, al igual que hicimos en Visual Studio Code al añadirlo en el .netpackages. En este caso lo hemos añadido en la carpeta *bin* y dentro de la carpeta *Debug*.

Una vez hemos realizado estos pasos, estamos seguros que vamos a poder usar los Add-In y por lo tanto vamos a poder realizar el desarrollo de la API. En la Figura 4.9, podemos observar dos archivos realizados en el lenguaje de programación *C#* (Functions.cs y Resultado.cs) dentro de la solución DemoAddIn. Estos dos archivos son los resultados de los Add-In, es decir, en estos dos archivos podemos ver el desarrollo al completo de los Add-In.

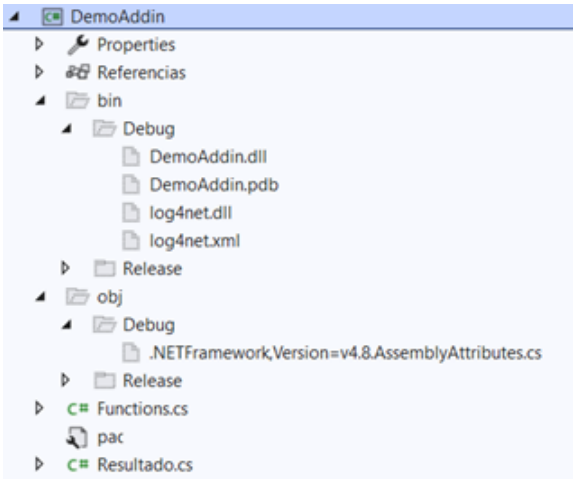

Figura 4.9: Solución DemoAddIn con los Add-In insertados correctamente.

DemoAPIREST

En la segunda solución podemos encontrar la construcción y desarrollo del API. Como se trata de una solución nueva vamos a tener que agregar los Add-In al apartado de referencias, para que se pueda realizar una conexión con las librerías Add-In, tal como se observa en la Figura 4.10.

Además, en este apartado es donde encontramos el patrón Modelo-Vista-Controlado (MVC). En esencia, el patrón MVC [11] se utiliza en la arquitectura de software, y separa los datos y la lógica de negocio de una aplicación de su representación y es el módulo encargado de gestionar los eventos.

Un de las ventajas del Modelo-Vista-Controlador, es que se ha desacoplado el modelo de datos y su visualización. Esto implica que se puede cambiar la vista sin cambiar el modelo, se

Figura 4.10: Introducción del Add-In al apartado de referencias.

puede tener más de una vista haciendo referencia al mismo modelo y se pude cambiar el modelo sin necesidad de tocar la vista.

El modelo solo conoce a la vista, la vista conoce al controlador y al modelo y el controlador conoce a la vista y al modelo. Entre las tres partes del patrón existen referencias cruzadas. La Figura 4.11 muestra la interconexión entre las tres partes del patrón MVC.

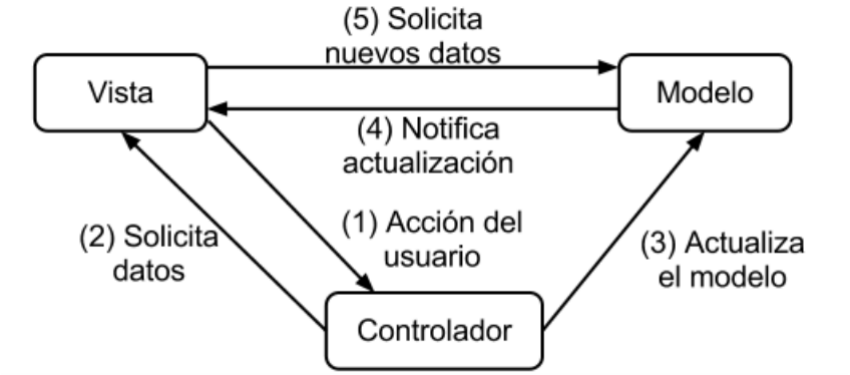

Figura 4.11: Modelo-Vista-Controlador (según [11]).

Ahora sí que podemos pasar a explicar como se han realizado las tres partes del patrón en torno a nuestro proyecto.

En primer lugar encontramos el *Modelo* [11], el cual define qué datos debe contener la aplicación. Además contiene todas las clases responsables de mantener y gestionar los datos de nuestro proyecto. Si el estado de estos datos cambia, el modelo notificará a la *Vista*, para que ésta pueda cambiar según sea necesario.

Como hemos mencionado anteriormente, cuando se crean este tipo de proyectos se crean clases de ejemplo, para que el programador tenga un ejemplo de como realizar el desarrollo. En este caso las clases DemoClass y DemoSubClass son las clases que nos proporciona Visual Studio como ejemplo de *Modelo*. La clase Entrada, que completa el *Modelo*, es la que hemos creado y contiene los parámetros que vamos a usar en la API. La Figura 4.12 muestra la carpeta asociada al *Modelo*.

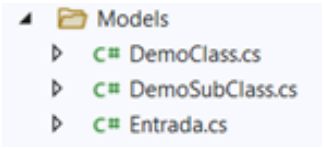

Figura 4.12: Estructura del Modelo.

A continuación pasamos a explicar la parte de la *Vista* [11], la cual define cómo se deben mostrar los datos de la aplicación. En nuestro caso hemos usado la vista que venía predefinida en Visual Studio ya que no nos interesa demasiado la estética sino la funcionalidad, es decir, que los Add-In se puedan convertir en una API. En la siguiente Figura 4.13 podemos observar la estructura de esta parte del patrón.

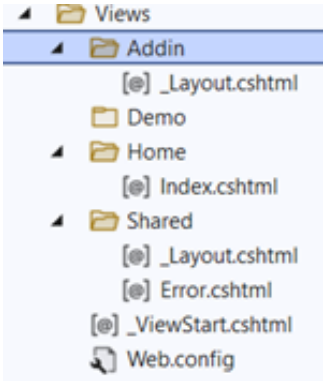

Figura 4.13: Estructura de la Vista.

Por último, dentro del patrón MVC, tenemos que implementar el *Controlador*[11], que se encarga de la lógica que actualiza el modelo y/o vista según las necesidades del proyecto. Para el *Controlador* también encontramos ficheros de ejemplo que Visual Studio nos ofrece. Como podemos observar en la figura 4.14 la estructura de la carpeta Controlador contiene varios ficheros. El fichero correspondiente al controlador es *AddinController.cs*; los otros tres son ejemplos.

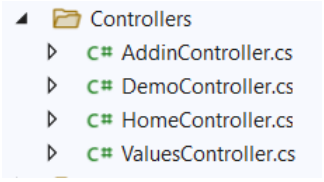

Figura 4.14: Estructura del Controlador.

En el fichero *AddinController.cs* hemos creado un método tipo *POST*. De los cuatro tipos de métodos soportados por HTTP y que se usan en servicios web API REST, este método es el que más nos conviene:

- PUT: Éste método utiliza para modificar un recurso ya existente y en nuestro caso no es necesario hacer una modificación, ya que no se puede hacer una modificación a algo que no existe.

- GET: Éste método se utiliza para consultar información de un recurso y no necesitamos consultar información sino crearla.

- DELETE: Éste método se utiliza para borrar un recurso y tampoco necesitamos borrar ningún recurso.

- POST: Éste método sirve para crear un recurso nuevo, que es lo que nos interesa ya que necesitamos crear ese recurso para poder hacer uso del Add-In como si no existieran anteriormente, pero en este caso usarlos en forma de API REST.

Para realizar un *POST* hay que realizar llamadas a los Add-In que hemos insertado anteriormente en las referencias y así podemos crearnos el objeto adecuado.

DemoTest

En esta última solución encontramos las pruebas que se han realizado sobre el Add-In para verificar que funcionan correctamente. Además también hemos añadido las pruebas del API REST a través de la herramienta Swagger.

La primera parte consiste en la creación de pruebas muy simples para asegurarnos que la funcionalidad del Add-In es la correcta.

La segunda parte de esta solución es la prueba del API REST con la herramienta Swagger. Para ello hemos tenido que configurar la herramienta en nuestra solución y agregar una URL para que la API se levantara en internet, siempre y cuando el proyecto estuviera en ejecución.

Un documento Swagger es el equivalente de API REST de un documento WSDL (*Web Services Description Language* [17]) para la descripción de servicios web. El documento Swagger especifica la lista de recursos disponibles en la API REST y las operaciones a las que se puede llamar en estos recursos.

Para configurar el Swagger hay que crear el fichero *SwaggerConfig.cs* en la carpeta correspondiente, como se muestra en la Figura 4.15.

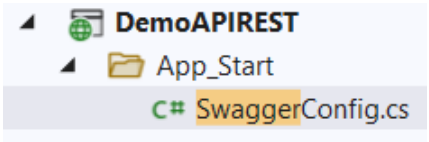

Figura 4.15: Carpeta donde añadir la configuración del Swagger.

En ese fichero se encuentra la configuración de la herramienta. No es necesario crear la configuración desde cero porque existen plantillas disponible públicamente que solo requieren de ligeros cambios para que funcione en nuestra solución. La última línea de la Figura 4.16 es la que se debe adaptar al proyecto en cuestión.

```
namespace DemoAPIREST
{
    2 referencias
    public class SwaggerConfig
    {
        0 referencias
        public static void Register()
        {
            var thisAssembly = typeof(SwaggerConfig).Assembly;

            GlobalConfiguration.Configuration
                .EnableSwagger(c =>
                    {
                        // By default, the service root url is inferred from the request used to access the docs.
                        // However, there may be situations (e.g. proxy and load-balanced environments) where this does not
                        // resolve correctly. You can workaround this by providing your own code to determine the root URL.
                        //
                        //c.RootUrl(req => GetRootUrlFromAppConfig());

                        // If schemes are not explicitly provided in a Swagger 2.0 document, then the scheme used to access
                        // the docs is taken as the default. If your API supports multiple schemes and you want to be explici
                        // about them, you can use the "Schemes" option as shown below.
                        //
                        //c.Schemes(new[] { "http", "https" });

                        // Use "SingleApiVersion" to describe a single version API. Swagger 2.0 includes an "Info" object to
                        // hold additional metadata for an API. Version and title are required but you can also provide
                        // additional fields by chaining methods off SingleApiVersion.
                        //
                        c.SingleApiVersion("v1", "DemoAPIREST");
```

Figura 4.16: Configuración Swagger.

A continuación, para realizar la configuración de la URL, es necesario crear dos ficheros como se muestra en la Figura 4.17, se trata de los ficheros “WebApiConfig” y “RouteConfig”.

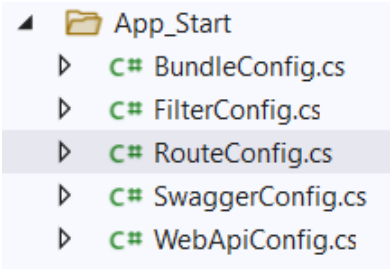

Figura 4.17: Configuración URL.

En los ficheros que hemos creado hay que realizar lo que se muestra en las Figuras 4.18 y 4.19.Estos dos ficheros no son necesarios programarlos desde cero ya que podemos encontrarlos en la página oficial de la herramienta (Swagger), solo es necesario modificarlo para que funcione en nuestro código.

Las modificaciones que hay que llevar a cabo, para poder probar la API con la herramienta (Swagger) son las siguientes;

- En la Figura 4.18, debemos cambiar los atributos “name”, “routeTemplate” y “dafaults”.
- En la Figura 4.19, debemos cambiar los atributos “name”, “url”, “defaults”.

Una vez hemos realizado todos los pasos, pasamos a realizar la prueba en la herramienta que hemos configurado. En la Figura 4.20 podemos observar como la respuesta a la llamada del

```
namespace DemoAPIREST
{
    1 referencia
    public static class WebApiConfig
    {
        1 referencia
        public static void Register(HttpConfiguration config)
        {
            // Web API configuration and services

            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

Figura 4.18: La ruta de la Web API.

```
namespace DemoAPIREST
{
    1 referencia
    public class RouteConfig
    {
        1 referencia
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```

Figura 4.19: Mapa de creación de la URL.

método *POST* da como resultado el código de estado *Status 200* lo que significa que la petición que acabamos de hacer ha sido entendida, enviada y recibida, además encontramos la entrada de los parámetros y una explicación de como se deben escribir.

En primer lugar en la Figura 4.21, podemos observar una instrucción, esa es la instrucción que nos ayuda si queremos hacer la comprobación del servicio web por línea de comando. Además podemos encontrar la URL donde se lanza la API. A continuación observamos el cuerpo de la API (*Response Body*), en el que encontramos el *Id* y el *Code*. Debajo vemos el código de respuesta de la API (*Response Code*) y por último las cabeceras necesarias de la API que hemos creado (*Response Headers*).

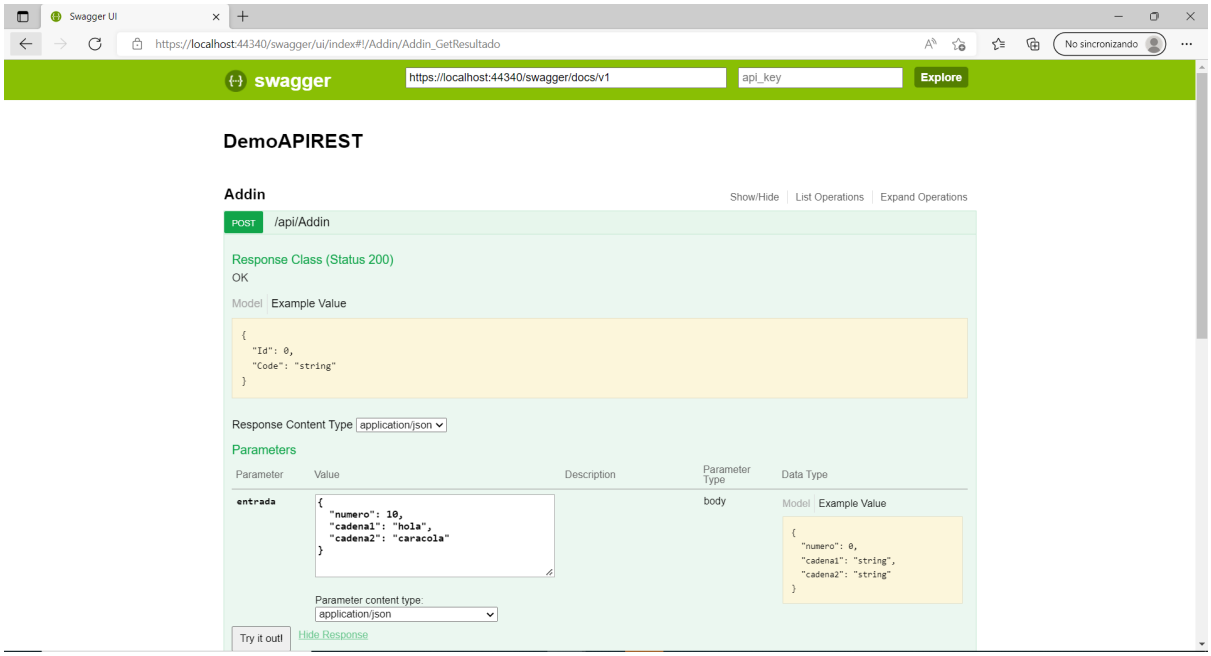

Figura 4.20: Prueba del API en Swagger.

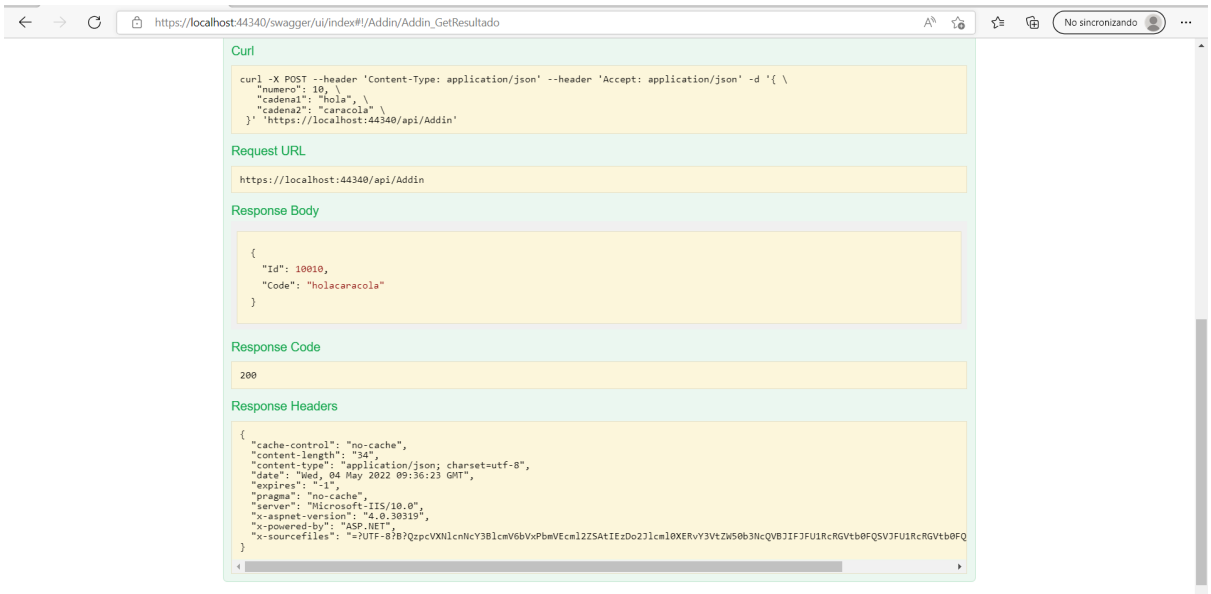

Figura 4.21: Continuación de la prueba del API en Swagger

4.3.3. Consumo del API REST desde MS Dynamics 365 Business Central

Para poder realizar esta parte debemos volver a Visual Studio Code para realizar una llamada al API que acabamos de crear, a través del protocolo HTTP y que sabemos que funciona gracias a la herramienta que hemos utilizado anteriormente (Swagger). Para ello, es necesario crear una nueva página en el lenguaje de programación C/AL y una nueva CodeUnit para poder realizar una llamada a la API.

En primer lugar tenemos que crear la CodeUnit que hemos denominado *PREFIX APICode*. En esta CodeUnit vamos a realizar lo que corresponde a la llamada a la API que hemos creado. Para ello necesitamos crear un método denominado *apiPost* dentro de la CodeUnit para que luego se puede llamar desde la página. En este método vamos a realizar una petición HTTP con los siguientes atributos:

- **Client: HttpClient.** Este atributo es necesario porque es el cliente de los métodos REST, en nuestro caso se trata de un POST.
- **ResponseMessage : HttpResponseMessage.** En este atributo se guarda la respuesta de la API.
- **Url: Text.** Se trata de la URL que hemos creado en el apartado anterior para que pueda hacer conexión.
- **Content: HttpContent.** El contenido de la API, es decir, es la clase base que representa el cuerpo de la entidad HTTP, en nuestro caso se insertan las cabeceras y el contenido del cuerpo de la API.
- **Headers: HttpHeaders.** Es donde se crean las cabeceras.
- **JBody: JsonObject.** El elemento donde se guarda el cuerpo, y se guarda como un objeto tipo Json, es un formato estándar de datos.
- **Jtoken: JsonToken.** Se usa para enviar el resultado final a los parámetros que sea adecuado, es decir guardamos en el Token el resultado del *id* y del *code*.
- **idapi: Text.** Variable de clase que se usa para trabajar en la página.
- **codeapi: Text.** Variable de clase que se usa para trabajar en la página.

Una vez se han creado los parámetros pertinentes se pasa a invocar la llamada. Para poder mostrar los resultados por MS Dynamics 365 Business Central debemos crearnos una página que llame al método que hemos creado en la CodeUnit y observar los resultados de la ejecución. En la Figura 4.22 se hace la llamada a la API que es el mensaje que nos sale con el texto *Aceptar* y después las variables globales obtienen el resultado de la llamada como podemos observar en la figura 4.23.

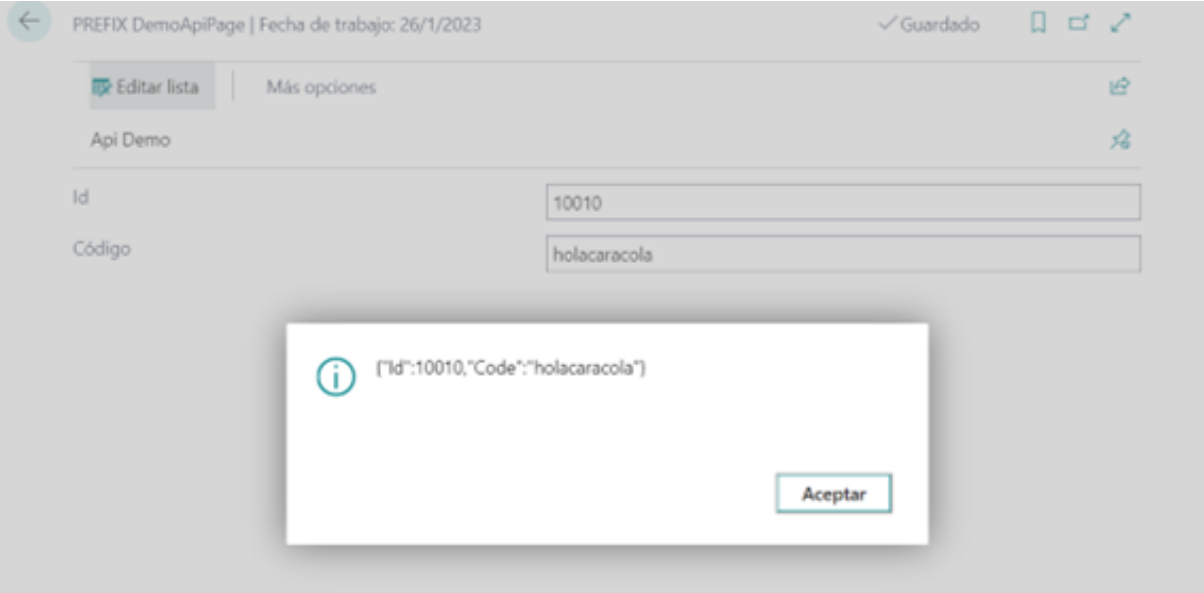

Figura 4.22: Resultado del API en MS Dynamics 365 Business Central.

Figura 4.23: Los atributos de la API.

4.4. Verificación y Validación

En cada parte del desarrollo del proyecto se han realizado labores de verificación y validación.

Con respecto a la parte de verificación, se han ido realizando pruebas cada semana para asegurarnos de que el alcance del proyecto se cumplía. En las reuniones semanales se ha comprobado los requisitos que se pedían. En las diferentes fases del proyecto se han especificado diferentes tipos de pruebas para ir controlando el funcionamiento correcto de las diferentes partes. Cada error que se encontraba durante la realización de ambas partes (primera parte Add-In, y la segunda parte el servicio web API REST), se estudiaba, se buscaban diferentes soluciones, se comentaba con el supervisor y se llegaba a un acuerdo para arreglarlo.

Con respecto a la parte de validación, se ha realizado a través de las CodeUnit que se han creado anteriormente. Además ha sido necesario crear una nueva página en la que se muestren los resultados de la Add-In y servicios web API REST para validar fácilmente si el resultado es el correcto, es decir, si la migración se ha llevado a cabo correctamente.

En la página creada para la comprobación final se encuentran el *Id* y el *Code* del Add-In (Figura 4.24). Asignamos valores a estas dos variables y realizamos una llamada al API (Figura 4.25). Finalmente, encontramos un botón que nos indica si ambos resultado son iguales o no (Figura 4.26).

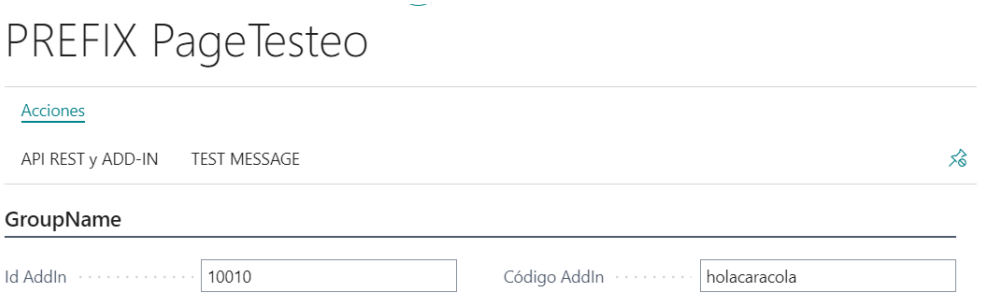

Figura 4.24: Resultado del Add-In.

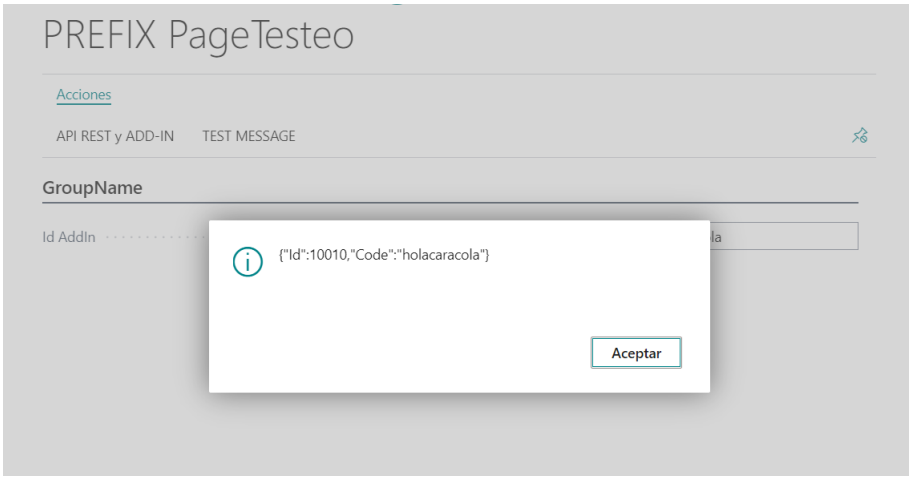

Figura 4.25: Resultado del API.

Figura 4.26: Comprobación final.

Capítulo 5

Conclusiones

En este apartado final se presentan los resultados obtenidos durante el desarrollo del proyecto y la valoración a nivel formativo, profesional y personal.

5.1. Ámbito formativo

Respecto al ámbito formativo, ha sido una muy buena oportunidad para aprender nuevas tecnologías que no conocía, como se puede tratar de la programación en el lenguaje *C/AL*, como trabajar con un *ERP* o programar en el lenguaje de programación *.NET* para poder trabajar con los Add-In.

Es importante añadir que la base de programación que se aprende en la universidad, me ha sido de gran ayuda para poder de manera más simple, adaptarme a los nuevos lenguajes con los que la empresa trabaja.

El objetivo principal del proyecto que se ha descrito al inicio de la memoria se ha logrado de manera satisfactoria, se ha conseguido realizar la migración en el tiempo establecido con éxito.

5.2. Ámbito profesional

En relación al ámbito profesional, la realización de la prácticas en *Laberit Sistemas S.L*, es una gran oportunidad en el ámbito profesional ya que en muchas empresas utilizan los ERP como sistema de recursos empresariales, y es muy interesante tener un conocimiento básico sobre los ERP.

El desarrollo de este proyecto me ha permitido aprender como es trabajar en una empresa tecnológica, asimismo me ha enseñado a adaptarme al ritmo de trabajo que hay que llevar en las empresas y poder ver como se desarrollan y llevan a cabo diferentes proyectos.

5.3. Ámbito personal

En cuanto al ámbito personal, estoy muy agradecida de poder haber realizado la estancia de prácticas y por ende el proyecto en *Laberit Sistemas S.L.* La empresa se ha esforzado siempre por integrarme en el ambiente de trabajo tanto a nivel personal como profesional. Se han preocupado desde el inicio de la estancia en que estuviera al tanto de todo y que entendiera todo lo que debía realizar.

Ha sido muy agradable que, por parte de la empresa se realizara un seguimiento del proyecto y que se realizaran reuniones para verificar que todo funcionaba correctamente. Es una grata sorpresa contar con una empresa que se preocupa, te ayuda y además te empuja a realizar el trabajo de la mejor forma posible.

Por otra parte, el desarrollo del proyecto también me ha servido como punto de partida para mi carrera profesional, ya que he adquirido diferentes habilidades y competencias, que pueden ser útiles en un futuro.

Bibliografía

[1] API. Página web sobre el funcionamiento de las API. https://www.redhat.com/es/topics/api/what-is-a-rest-api, 2022.

[2] Visual Studio Code. Página web de visual studio sobre Visual Studio Code. https://code.visualstudio.com/, 2022.

[3] Contenedor. Página web sobre Contenedores. https://www.icot.es/introduccion-a-los-containers/, 2022.

[4] Docker. Página web de wikipedia sobre Dockers. https://es.wikipedia.org/wiki/Docker_(software), 2022.

[5] Visual Studio Community Edition. Página web de visual studio sobre Visual Studio Community. https://visualstudio.microsoft.com/es/vs/, 2022.

[6] ERP. Página web sobre ERP. https://www.oracle.com/co/erp/what-is-erp/, 2022.

[7] Extensiones. Página web de visual studio marketplace sobre las Extensiones. https://marketplace.visualstudio.com/items?itemName=waldo.al-extension-pack, 2022.

[8] Git. Página web de openwebinars sobre el funcionamiento de Git. https://openwebinars.net/blog/que-es-git-y-para-que-sirve/, 2022.

[9] Microsoft. Página web de Microsoft Dynamics 365 Business Central. https://dynamics.microsoft.com/es-es/business-central/overview/, 2022.

[10] Microsoft. Página web de Microsoft Dynamics NAV. https://dynamics.microsoft.com/es-es/nav-erp/, 2022.

[11] MVC. Página web del aula virtual de la asignatura programación avanzada ei1017 sobre el Patron MVC. https://www3.uji.es/~belfern/Docencia/Presentaciones/ProgramacionAvanzada/Tema3-JavaFX/programacionDirigidaEventos.html#23, 2022.

[12] Microsoft Project. Página web de aglaia sobre Microsoft Project. https://aglaia.es/blog/office-365/que-es-microsoft-project/, 2022.

[13] Laberit Sistemas. Página web de Laberit Sistemas SL. https://www.laberit.com/, 2022.

[14] SoapUI. Página web de the economics time sobre SoapUI. https://economictimes.indiatimes.com/definition/soapui, 2022.

[15] Swagger. Página webde ibm integration bus sobre Swagger. https://www.ibm.com/docs/es/integration-bus/10.0?topic=apis-swagger, 2022.

[16] Wikipedia. Página web de wikipedia sobre Desarrollo en cascada. https://es.wikipedia.org/wiki/Desarrollo_en_cascada, 2022.

[17] WSDL. Página web de ibm describiendo el WSDL. https://www.ibm.com/docs/es/rsas/7.5.0?topic=standards-web-services-description-language-wsdl, 2022.