Grado en Ingeniería Informática

Trabajo de Final de Grado

Adaptación tecnológica de Servicios Web de MS Dynamics NAV a MS Dynamics 365 Business Central

Autor:
Nieves Cubedo Hurtado

Supervisor:
Luis Rius Gumbau
Tutor académico:
Carlos Granell Canut

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

Resumen

En este documento se detalla el desarrollo realizado durante la estancia en prácticas en la empresa Lãberit S.L.

El proyecto consiste en realizar una migración de servicios web SOAP a servicios web OData y API REST en Microsoft Business Central. Para ello, se ha creado un servicio SOAP sencillo que servirá de ejemplo, se han creado sus servicios equivalentes REST, se han consumido dichos servicios en distintos lenguajes y se ha documentado el proceso completo. El objetivo es realizar un manual con ejemplos de cómo se realiza una migración de este tipo. Esta migración es de vital importancia, pues Microsoft va a descatalogar los servicios SOAP de Business Central, por lo que va a dejar de darles soporte y, por tanto, lo mejor es buscar otras alternativas.

Palabras clave

Servicio web, Microsoft Business Central, petición HTTP

Keywords

Web service, Microsoft Business Central, HTTP request

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

Lãberit [6] surge en mayo de 2021, fruto de la fusión de ocho empresas del grupo Alfatec. Esta nueva sociedad abarca casi todas las actividades del sector de Nuevas Tecnologías de la Información y Comunicación. Una de sus unidades de negocio es la de Servicios para Autoridades Portuarias, donde acumulan más de 15 años de experiencia. Son expertos en la implantación de Microsoft Dynamics 365 Business Central y Microsoft Dynamics NAV en puertos.

Microsoft Dynamics NAV [10], también conocido como Navision, es un software de gestión de recursos empresariales o ERP (Enterprise Resource Planning). Este software permite integrar y automatizar distintos procesos de negocio dentro de una empresa. En abril de 2018, Microsoft lanza al mercado al sucesor de Navision, Microsoft Dynamics 365 Business Central [9], también conocido como Business Central. Este software es una evolución tecnológica de Navision, pero con sus mismas funcionalidades. Por ello, Navision ha quedado obsoleto y tiende a desaparecer.

La principal motivación para desarrollar este proyecto surge de la necesidad de migrar todas las soluciones existentes en Navision a Business Central dada la situación de obsolescencia en la que se encuentra dicho programa. En concreto, este proyecto se centra en migrar los servicios web de tipo SOAP (*Simple Object Access Protocol*) a servicios web de tipo REST (*REpresentational State Transfer*) y redactar documentación que facilite esta tarea de migración a otros desarrolladores de la empresa. Si bien es cierto que los servicios web SOAP pueden ser usados en Business Central, Microsoft ya ha advertido de que van a ser descatalogados [17], pues realizan el intercambio de datos mediante XML (*eXtensible Markup Language*) y cada vez está más en desuso.

1.2. Objetivos del proyecto

El objetivo principal del proyecto es actualizar los servicios web SOAP de Business Central transformándolos a OData (Open Data Protocol) o API REST. Esto es necesario porque Busi-

ness Central va a dejar de utilizar este tipo de servicios en el futuro, por lo que es imprescindible anticiparse al cambio.

El objetivo principal se puede desglosar en los siguientes subobjetivos:

- Crear un servicio SOAP
- Transformar un servicio SOAP en API REST y OData
- Documentar cómo transformar un servicio SOAP en REST en Business Central
- Documentar cómo consumir un servicio OData usando diferentes tecnologías
- Documentar cómo consumir un servicio API REST usando diferentes tecnologías

1.2.1. Alcance

Para poder definir de forma más clara el proyecto, es necesario definir el alcance. El cual vamos a dividir en tres categorías: alcance funcional, organizativo e informático.

- Alcance funcional: El proyecto se basa en transformar un servicio web SOAP en un servicio REST, documentando tanto el proceso de migración como el modo en que consumirá tras el cambio de tecnologías. Por este motivo, la funcionalidad es la misma y no se va a ver afectada. Vamos a realizar un cambio de arquitectura manteniendo la funcionalidad original.

- Alcance organizativo: El servicio debe poder interactuar con cualquier empresa o software externo que necesite conectar con nuestra solución. Al tratarse de un servicio web, los usuarios no son entes físicos sino otros servicios o software.

- Alcance informático: Por un lado, el servicio web se comunica con Microsoft Business Central. Por otro lado, se comunica con el software de cualquier empresa que necesite conectar con nuestra solución. Este software puede estar desarrollado en diferentes tecnologías, por ejemplo, Java o .Net. Podemos ver un diagrama de esta comunicación en la figura 1.1.

1.3. Descripción del proyecto

El proyecto se centra en replantear funcional y tecnológicamente los servicios web a las nuevas directrices de las nuevas versiones del producto. Además, será necesario realizar los tests unitarios pertinentes y ejecutarlos. Actualmente, estos servicios web usan tecnología SOAP con XML, la cual está cada vez más en desuso. De hecho, Business Central la va a descatalogar,

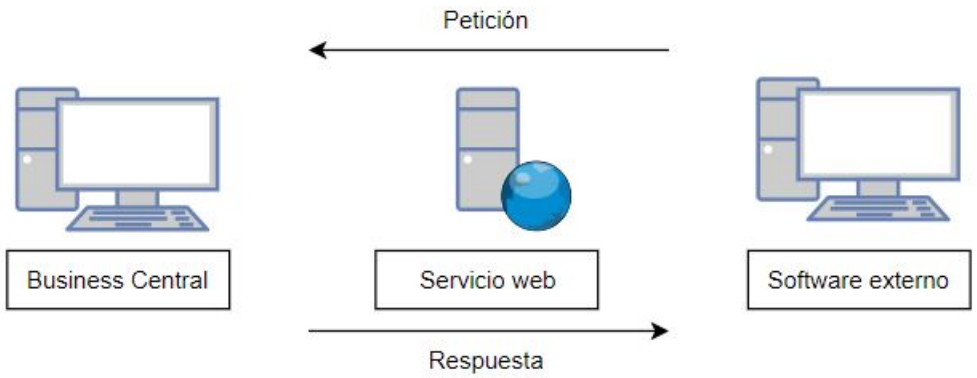


Figura 1.1: Diagrama de comunicación de un servicio web

por lo que debe ser sustituida por REST. Además, será necesario consumir dichos servicios utilizando diferentes tecnologías, en concreto usaremos: Java, .NET y una extensión de Visual Studio Code llamada REST Client. Finalmente, se realizará documentación técnica que describa todo el proceso realizado.

En resumen, se trata de un proyecto de investigación donde se van a crear un par de servicios SOAP sencillos a modo de ejemplo y se van a migrar a API REST y OData. A su vez, se investigará y ejemplificará como consumir dichos servicios REST usando diferentes tecnologías. De igual manera, se debe elaborar documentación técnica que especifique cómo realizar la migración y como consumir los servicios publicados con distintos lenguajes de programación. Esta documentación será usada por otros desarrolladores de la empresa a modo de guía o manual a la hora de realizar futuras migraciones.

Debido a que hay otra alumna realizando un proyecto sobre servicios web en Business Central en la misma empresa, en la figura 1.2 se muestra un diagrama donde se aprecia la diferencia entre ambos proyectos, siendo el cuadrado azul en lo que se va a centrar este proyecto. Resumidamente, este proyecto se centra en transformar los servicios web SOAP publicados en Business Central en servicios REST y ver como se consumen usando software externo. En cambio, el proyecto realizado por la otra alumna se centra en transformar los servicios add-in y dll publicados mediante software externo en servicios REST y ver como se consumen desde Business Central.

1.3.1. Tecnologías y herramientas utilizadas

Para elaborar este proyecto se han utilizado diferentes tecnologías y herramientas, algunas de ellas son:

- Visual Studio Code (VSC)[1]: Potente editor de código desarrollado por Microsoft. Es compatible con múltiples lenguajes de programación, entre ellos, el lenguaje de Business Central, AL. Con él, se han desarrollado los servicios web en lenguaje AL.
- Microsoft Visual Studio (VS)[22]: Entorno de desarrollo integrado con el cual se ha realizado la implementación del código `C#` y .Net con el que se han creado diversos clientes para consumir los servicios web.

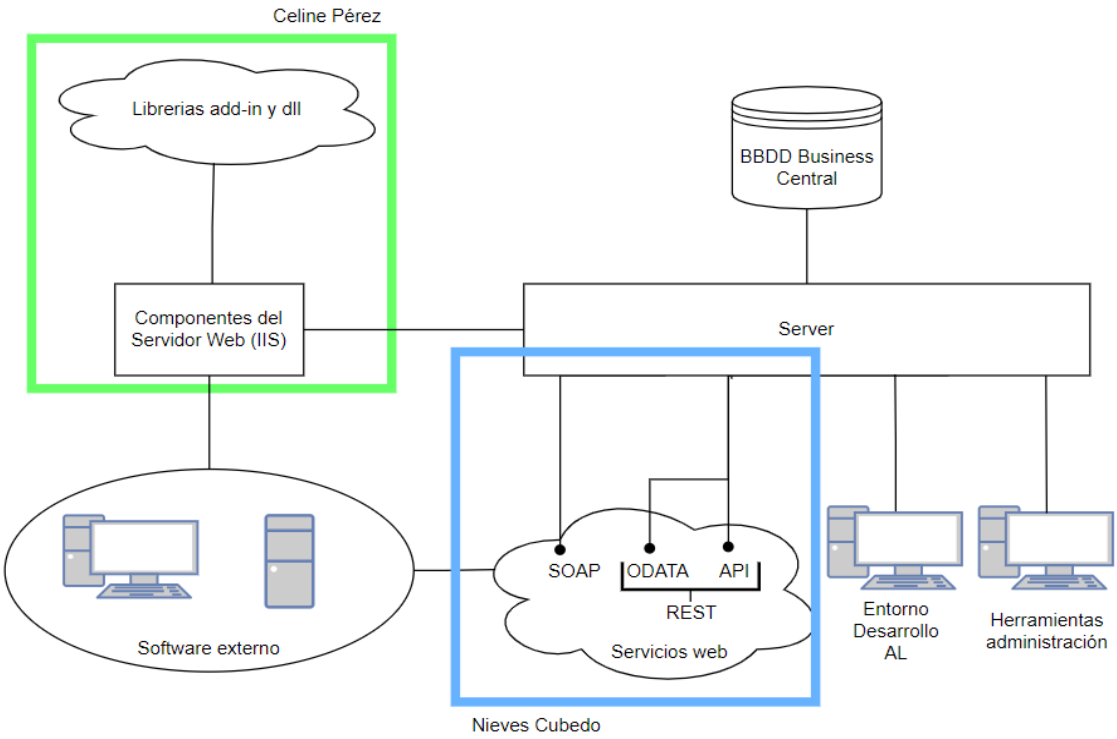


Figura 1.2: Diagrama de servicios web en Business Central

- IntelliJ IDEA[5]: Entorno de desarrollo integrado diseñado por JetBrains donde se ha elaborado el código Java con el cual se han creado clientes que consumen los servicios web.
- Azure Devops Server[12]: Herramienta de control de versiones, gestión de proyectos, gestión de requisitos, entre otros, desarrollada por Microsoft. Se ha usado para el control de versiones del software desarrollado.
- SoapUI[21]: Herramienta para la realización de pruebas a servicios web SOAP y REST.
- Postman[19]: Plataforma API para desarrolladores que permite diseñar, construir, testear e iterar sobre APIs. Se ha utilizado para realizar pruebas a servicios web.
- Microsoft Dynamics 365 Business Central (BC)[13][9]: Sistema de gestión empresarial desarrollado por Microsoft. Permite gestionar los apartados de finanzas, ventas, marketing, servicios, comercio, entre otros. Se ha utilizado para publicar los servicios web, así como para acceder a las interfaces de los servicios de tipo página SOAP y OData.
- Google Chrome[4] y Microsoft Edge[14]: Navegadores web donde se ha lanzado nuestra instancia local de Business Central.
- Magic Draw[23]: Herramienta CASE con la que se han diseñado tanto el diagrama de casos de uso como el diagrama de clases.
- Microsoft Project[15]: Herramienta de gestión de proyectos desarrollada por Microsoft. Se ha empleado para realizar la planificación del proyecto.

- OData Connected Service[20]: Extensión de Visual Studio que genera una clase en lenguaje `C#` que permite interaccionar con un servicio web REST específico.
- REST Client[7]: Extensión para Visual Studio Code que permite enviar peticiones HTTP (Hypertext Transfer Protocol) y visualizar la respuesta en VSC directamente.
- Lenguaje AL[11][8]: Lenguaje de programación de Business Central a partir de la versión 14. Se usa principalmente para consultar, insertar, modificar y eliminar registros en una base de datos de Business Central, además de manejar objetos, variables y funciones propias de Business Central. Su predecesor es el lenguaje C/AL. El lenguaje AL se usará para desarrollar nuestra solución.
- Base de datos local de Business Central: Almacén de datos estructurados que están relacionados por el contexto. Se ha utilizado para realizar distintas pruebas.

1.4. Estructura de la memoria

Esta memoria se divide en una serie de capítulos para organizar el contenido. En este apartado se describe brevemente el contenido de cada uno:

En el capítulo 2 se muestra la planificación del proyecto, así como la metodología que se ha seguido y una estimación de los costes y recursos utilizados en el proyecto.

En el capítulo 3, se expone el análisis previo realizado, el cual incluye: una comparativa entre los distintos tipos de servicios web, la definición de requisitos, un diagrama de casos de uso y un diagrama de clases. Además, en este capítulo se describe el diseño de la arquitectura del sistema y de la interfaz.

En el capítulo 4, se detalla el trabajo realizado para implementar el proyecto y las pruebas que se han realizado para certificar que es correcto.

En el capítulo 5, se exponen las conclusiones finales que se han extraído sobre el desarrollo del proyecto.

Capítulo 2

Planificación del proyecto

2.1. Metodología

La metodología escogida para el desarrollo de este proyecto es la metodología predictiva, en concreto el desarrollo en cascada [25]. Esta metodología divide el proyecto en etapas que se realizan de forma secuencial, es decir, al terminar una comienza la siguiente. Para que esto sea posible, es necesario tener claros los objetivos y requisitos del proyecto de antemano, realizando un análisis previo exhaustivo a fin de evitar contratiempos. Por tanto, se ha realizado un análisis previo y se ha dividido el proyecto en una serie de etapas y tareas que se realizarán de manera secuencial.

Se ha escogido esta metodología por distintos motivos. En primer lugar, porque gracias a su simplicidad es fácil de implementar. En segundo lugar, es más sencillo ejecutar las tareas de forma secuencial que en paralelo al realizar el proyecto una única persona. Además, esta metodología permite ser más preciso a la hora de estimar los costes y la duración del proyecto dado que se planifica meticulosamente desde el principio, lo que permite que no suelan haber contratiempos que no estén previstos. Finalmente, se ha elegido porque el proyecto no presenta mucha incertidumbre, por lo que no se van a realizar muchos cambios sobre el mismo.

Durante el desarrollo del proyecto, se han realizado reuniones semanales para evaluar el estado del mismo y evaluar si se puede pasar a la siguiente fase o tarea. Igualmente, si durante el desarrollo del mismo ha surgido algún problema o duda, se han podido concertar reuniones adicionales en cualquier momento.

2.2. Planificación

Para planificar el proyecto, en primer lugar, se ha elaborado un listado con las etapas que lo van a definir, lo cual nos ayudará a saber en que orden hay que ejecutar cada tarea.

El proyecto comienza con una fase de formación básica en la que se imparten nociones básicas

sobre contabilidad, el entorno de Business Central y Navision, servicios web y el lenguaje de programación de Business Central, AL. Una vez adquiridos los conocimientos mínimos, se inicia la fase de planificación del proyecto. En ella se realiza un estudio del caso que permite definir los requisitos, el alcance y los objetivos. Con toda esta información se desarrolla la propuesta técnica. Tras todo esto, comienza el análisis y el diseño del proyecto donde se diseña el diagrama de casos de uso y el de clases, y se realiza una comparativa entre los servicios SOAP, OData y REST. A continuación, se realiza la implementación del proyecto donde se crea un objeto SOAP sencillo a modo de ejemplo. Este es transformado en OData y API REST mientras se documenta todo el proceso y se plantean ejemplos de cómo consumir los servicios creados en REST. Finalmente, se comprueba el correcto funcionamiento de los servicios web elaborando test unitarios en java y comparando los resultados de los servicios SOAP con los servicios REST.

A continuación, se muestra el listado de etapas del proyecto:

- Formación básica
  - Formación en contabilidad, compras, ventas, cobros y pagos
  - Formación en Business Central y Navision
  - Formación C/AL
  - Formación en Servicios Web
- Planificación
  - Estudio del caso
  - Definir requisitos
  - Definición del alcance y los objetivos
  - Desarrollo de la Propuesta Técnica
- Análisis y Diseño
  - Diseño diagrama de clases
  - Comparativa SOAP, OData Y API REST
- Implementación
  - Crear objeto SOAP
  - Convertir servicio web SOAP en OData
  - Convertir servicio web SOAP en API REST
  - Documentar como consumir el servicio usando distintas herramientas
  - Plantear ejemplos de como consumir servicio web
- Pruebas
  - Pruebas de bajo nivel
  - Pruebas de alto nivel

En segundo lugar, se ha usado la herramienta Microsoft Project para elaborar un diagrama de Gantt en el que se han especificado las fases y tareas de las que consta el proyecto, así como la duración estimada del proyecto para así poder estimar las fechas de inicio y fin de cada fase. En las figuras 2.1, 2.2 se muestra el diagrama de Gantt creado.

	Nombre de tarea	Duració	Comienzo	Fin
1	**Proyecto Servicios Web**	**60 días**	**jue 03/02/22**	**mar 03/05/22**
2	**Formación básica**	**12 días**	**jue 03/02/22**	**vie 18/02/22**
3	Formación en contabilidad, compras, ventas, cobros y pagos	2 días	jue 03/02/22	vie 04/02/22
4	Formación en Business Central y Navision	3 días	lun 07/02/22	mié 09/02/22
5	Formación C/AL	5 días	jue 10/02/22	mié 16/02/22
6	Formación en Servicios Web	2 días	jue 17/02/22	vie 18/02/22
7	**Planificación**	**6 días**	**lun 21/02/22**	**lun 28/02/22**
8	Estudio del caso	1 día	lun 21/02/22	lun 21/02/22
9	Definir requisitos	1 día	mar 22/02/22	mar 22/02/22
10	Definición del alcance y los objetivos	2 días	mié 23/02/22	jue 24/02/22
11	Desarrollo de la Propuesta Técnica	4 días	mié 23/02/22	lun 28/02/22
12	**Análisis y Diseño**	**4 días**	**mar 01/03/22**	**vie 04/03/22**
13	Diseño diagrama de clases	2 días	mar 01/03/22	mié 02/03/22
14	Comparativa SOAP, ODATA Y API REST	2 días	jue 03/03/22	vie 04/03/22
15	**Implementación**	**27 días**	**lun 07/03/22**	**mié 13/04/22**
16	Crear objeto SOAP	4 días	lun 07/03/22	jue 10/03/22
17	Convertir servicio web SOAP en ODATA	3 días	vie 11/03/22	mar 15/03/22
18	Convertir servicio web SOAP en API REST	9 días	mié 16/03/22	mar 29/03/22
19	Documentar como consumir el servicio usando distintas herramientas	4 días	mié 30/03/22	lun 04/04/22
20	Plantear ejemplos de como consumir servicio web	11 días	mié 30/03/22	mié 13/04/22
21	**Pruebas**	**11 días**	**mar 19/04/22**	**mar 03/05/22**
22	Pruebas de bajo nivel	5 días	mar 19/04/22	lun 25/04/22
23	Pruebas de alto nivel	6 días	mar 26/04/22	mar 03/05/22
24	Cierre del proyecto	0 días	mar 03/05/22	mar 03/05/22

Figura 2.1: Planificación del proyecto

2.3. Gestión de riesgos

La gestión de riesgos es una pieza fundamental en la planificación del proyecto, puesto que ayuda a prevenir o disminuir el impacto de ciertos problemas que pueden surgir a lo largo del proyecto. Por supuesto, estos riesgos pueden no darse en ningún momento o bien surgir algún problema no previsto, pero este análisis previo ayuda a reducir la incertidumbre del proyecto al prevenir o controlar los imprevistos que puedan surgir.

En primer lugar, se identifican los riesgos que pueden surgir y afectar al proyecto, indicando también el tipo al que pertenecen. Esto nos ayuda a saber el posible origen de este riesgo o a que áreas va a afectar. Siendo el riesgo de entorno procedente de causas ajenas al proyecto, el riesgo de producto afecta directamente a la calidad del producto o a su coste temporal y el riesgo de proyecto afecta al coste o planificación del proyecto. Se han identificado los riesgos del proyecto en el cuadro 2.1.

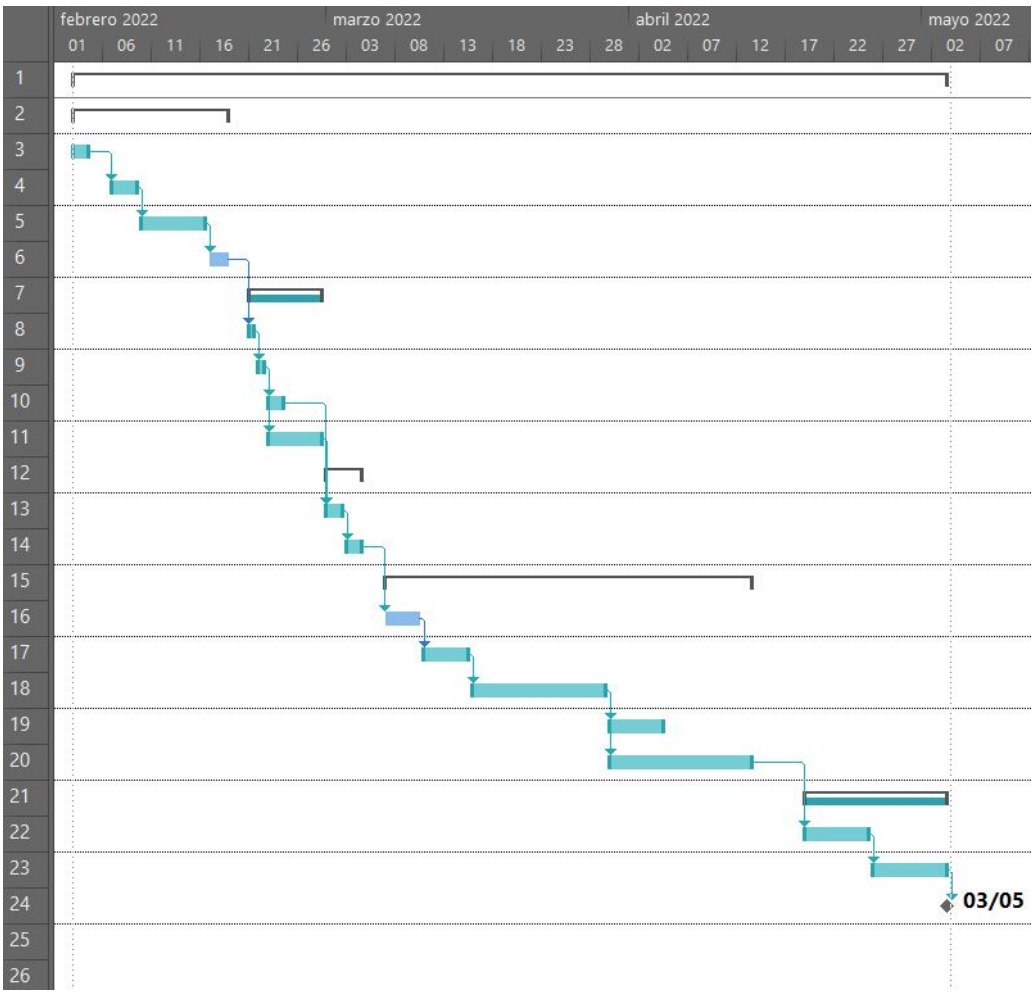


Figura 2.2: Planificación del proyecto (continuación)

ID	**Descripción del riesgo**	**Tipo de riesgo**
R01	Requisitos poco claros o mal definidos	Riesgo de producto
R02	Falta de conocimientos o experiencia en las tecnologías usadas	Riesgo del producto
R03	Falta de experiencia en tareas de planificación de proyectos	Riesgo del proyecto
R04	Baja temporal de un miembro del equipo	Riesgo del proyecto
R05	Pandemia	Riesgo de entorno

Cuadro 2.1: Identificación del riesgo

A continuación, se procede a realizar un análisis de los riesgos definidos a fin de evaluar el peligro que suponen para el proyecto. En el cuadro 2.2, se describe cada riesgo indicando: la magnitud con la que afectará al proyecto, es decir, el peligro que supone para el mismo; la descripción del riesgo; el impacto que supondría si se produjera; y los indicadores que pueden ayudar a tomar acciones de prevención o corrección al ver que el riesgo se está produciendo.

ID	Análisis del riesgo
R01	**Magnitud** Variable. Baja durante las primeras fases iniciales, alta si el proyecto está muy avanzado. **Descripción** El análisis de requisitos realizado es insuficiente, erróneo o no se ajusta a las demandas de los clientes. **Impacto** Incorporar o modificar requisitos suele suponer un retraso. Si es durante las fases iniciales solo habrá que modificar el análisis y el diseño del proyecto, pero si ya está muy avanzado habrá que modificar también parte del desarrollo y la documentación. **Indicadores** El cliente no es capaz de especificar qué es lo que necesita de la solución.
R02	**Magnitud** Media durante las primeras fases iniciales, alta si el proyecto está muy avanzado. **Descripción** Algún miembro del equipo tiene dificultades para cumplir sus objetivos puesto que carece de los conocimientos necesarios en las herramientas utilizadas. **Impacto** Puede provocar retrasos o reducir la calidad del proyecto. **Indicadores** No se cumplen los plazos de entrega respecto de la planificación o las entregan se realizan pero incompletas o con numerosos errores.
R03	**Magnitud** Variable. Baja durante las primeras fases iniciales, alta si el proyecto está muy avanzado. **Descripción** Falta de experiencia a la hora de planificar tareas para el desarrollo de software. **Impacto** Una buena planificación es clave para el correcto desarrollo del proyecto, realizarla de forma incorrecta puede llevar a grandes retrasos. Utilizar herramientas y documentación de planificación de tareas o contar con un profesional con experiencia puede reducir estos errores de planificación. **Indicadores** Discrepancias entre la planificación y el avance real del proyecto.
R04	**Magnitud** Alta **Descripción** Un miembro del equipo no puede continuar trabajando durante un período corto de tiempo, lo más probable es que sea por enfermedad dada la situación de la pandemia actual. **Impacto** La falta de personal puede provocar pérdidas de calidad o retrasos en el proyecto respecto a la planificación inicial. **Indicadores** Ninguno. Es muy difícil de predecir dado que es ajeno al proyecto.

ID	Análisis del riesgo
R05	**Magnitud** Media **Descripción** La pandemia de coronavirus afecta a toda la población mundial. Puede llevar a confinamientos que impidan que los trabajadores se desplacen a sus puestos de trabajo o incluso a bajas laborales. **Impacto** El impacto puede variar mucho. En el mejor de los casos, si se imponen confinamientos, los miembros del equipo pueden continuar teletrabajando desde sus casas. En el peor de los casos, puede provocar bajas laborales y provocar retrasos o perdidas en la calidad del producto. **Indicadores** Ninguno o muy pocos. Es muy difícil de predecir. Un indicador podría ser un repunte en la incidencia acumulada por casos covid en los últimos días.

Cuadro 2.2: Análisis del riesgo

Finalmente, se elabora un plan de prevención o acción y otro de corrección o contingencia. En el plan de prevención o acción se especifican las medidas a tomar para prevenir que un riesgo surja y pase a ser un problema, evitando así que se produzca o reduciendo sus consecuencias. En cambio, el plan de corrección o contingencia se lleva a cabo una vez el riesgo ya se ha llevado a cabo y es un problema. Este plan, permitiría contrarrestar o reducir las consecuencias del riesgo al mínimo posible. Podemos contemplar estas medidas en el cuadro 2.3.

ID	Plan de Prevención/Acción	Plan de Corrección/Contingencia
R01	Realizar diversas entrevistas y hacer gran cantidad de preguntas a los usuarios para asegurarse de entender bien los requisitos de datos. Además, realizar un análisis exhaustivo del proyecto.	Si se detecta en las fases iniciales se pueden modificar requisitos y el análisis. En cambio, si el proyecto está muy avanzado, habrá que valorar si los cambios son muy importantes o si van a suponer un gran beneficio y se pueden ajustar a la planificación sin suponer grandes retrasos.
R02	Realizar cursos de formación en las tecnologías y herramientas usadas durante el proyecto.	Solicitar ayuda de un experto sobre cómo lidiar con las dificultades que está teniendo.
R03	Realizar una planificación con cierto margen entre tareas por si surgieran contratiempos. Utilizar herramientas y documentación de planificación de tareas y consultar al tutor o con algún profesional para reducir errores.	Comparar el avance real del proyecto con la planificación frecuentemente con el fin de poder corregir a tiempo las pequeñas desviaciones que puedan surgir.
R04	Tratar de cumplir los objetivos del proyecto antes de la fecha estimada en la planificación para que si hay alguna baja no conlleve a un retraso importante.	Intentar ajustar la planificación o eliminar requisitos poco importantes para poder cumplir los plazos.

ID	Plan de Prevención/Acción	Plan de Corrección/Contingencia
R05	Seguir las recomendaciones o obligaciones impuestas por las autoridades regionales. Fomentar el uso de la mascarilla en lugares cerrados, mantener una buena ventilación e higiene de manos, así como mantener la distancia interpersonal.	Ofrecer teletrabajo a las personas confinadas.

Cuadro 2.3: Acciones de Prevención y Corrección

2.4. Estimación de recursos y costes del proyecto

En esta sección se van a estimar los costes del proyecto como si se tratase de un proyecto profesional, incluyendo los costes referentes a recursos humano y los costes de hardware y software.

En primer lugar, se van a estimar los costes de Recursos Humanos que, para este proyecto, se refieren solo a los de un trabajador. Para ello, partimos de que se ha contratado a un desarrollador por 18000€ al año y hace jornadas de 8 horas al día. Asumimos también que las pagas extra están prorrateadas, es decir, repartidas en las nóminas mensuales. En este proyecto, las jornadas son de 5 horas al día, por tanto, el primer paso es calcular su sueldo proporcional para la jornada del proyecto. Para ello, se divide el sueldo anual del trabajador entre el número de horas de su jornada y se multiplica por las horas que va a realizar durante su jornada laboral en el proyecto: 18000€$\times$5 $\div$ 8 = 11250€. Esta cantidad es anual, pero el proyecto tiene una duración de 3 meses. Por tanto, para calcular el sueldo proporcional, se divide la cantidad anteriormente calculada entre los doce meses del año y se multiplica por el número de meses que se van a trabajar, en este caso 3, y este será el coste de Recursos Humanos de nuestro proyecto que podemos ver en el cuadro 2.4.

Sueldo anual (8h)	Sueldo anual (5h)	Sueldo mensual (5h)	Sueldo 3 meses (5h)
18000€	11250€	938,5€	**2812,5€**

Cuadro 2.4: Costes Recursos Humanos del proyecto

En segundo lugar, tenemos los costes de Hardware. Se asume que la vida útil de los equipos será de unos 5 años, lo que equivale a 60 meses, dado que la duración del proyecto es de 3 meses, se debe calcular el coste aproximado mensual y multiplicarlo por el número de meses del proyecto. Este coste se detalla en el cuadro 2.5.

Coste mensual = Coste $\div$ 60
Coste proporcional = Coste mensual $\times$ duración del proyecto en meses

En tercer lugar, se encuentran los costes de Software. Nos basta con licencias gratuitas para poder usar la mayor parte de los programas, pues la solución desarrollada no se va a comercializar y se va a usar como material didáctico dentro de la organización. Las licencias que sí necesitaremos serán las de Dynamics 365 Business Central Essentials por 59€ al mes por

Hardware	Coste	Coste mensual	Coste proporcional
Ordenador portátil	500€	41,67€	125€
Teclado	15€	1,25€	3,75€
Ratón	10€	0,83€	2,5€
Auriculares	25€	2,083€	6,25€
Monitor	150€	12,5€	37,5€
Total			**175€**

Cuadro 2.5: Costes del Hardware del proyecto

usuario y Microsoft Office 365 Empresa Estándar por 10,5€ al mes por usuario. En el cuadro 2.6 podemos ver en detalle los costes del Software del proyecto.

Software	Coste mensual	Coste proporcional
Dynamics 365 Business Central Essentials	59€	177€
Microsoft 365 Empresa Estándar	10,5€	31,5€
SoapUI	0€	0€
Postman	0€	0€
Visual Studio Community	0€	0€
Visual Studio Code	0€	0€
IntelliJ	0€	0€
Azure DevOps	0€	0€
Total		**208,5€**

Cuadro 2.6: Costes del Software del proyecto

Finalmente, los costes totales serán la suma de los costes de Recursos Humanos, Hardware y Software más un 20 % para gastos indirectos (Cuadro 2.7).

Coste	Total
Coste Recursos Humanos	2812,5€
Coste Hardware	175€
Coste Software	208,5€
Total	**3196€**
Total con gastos indirectos	**3835,2€**

Cuadro 2.7: Coste total del proyecto

2.5. Seguimiento del proyecto

Este proyecto utiliza la metodología predictiva. A lo largo de su desarrollo, se han realizado reuniones semanales con el responsable de la empresa para ver el progreso de este y evaluar si se puede avanzar a la siguiente fase o tarea. Si no se puede avanzar, se debe revisar el proyecto hasta que esté apto para continuar.

También se han realizado informes quincenales donde se detallan las tareas que se han realizado a lo largo de la quincena y las que se prevé realizar en la próxima quincena.

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

En este apartado se definen los pasos que se han seguido para realizar el análisis del proyecto. En primer lugar, se han definido los requisitos que deben cumplir los servicios web desarrollados. El análisis de requisitos es una de las partes fundamentales de un buen análisis del sistema, pues nos permite ver qué funcionalidades debe cumplir el software desarrollado. Gracias a estos requisitos, se ha realizado el diagrama de casos de uso con sus debidas especificaciones. El diagrama de casos de uso nos permite observar la relación entre los diversos actores y el sistema. Mientras que en la especificación del caso de uso, se puede ver un paso a paso de esta interacción, lo cual nos permite ver qué partes tienen mayor importancia.

3.1.1. Comparativa SOAP, OData y API REST

Un servicio web [16] [24] es un software que usa un conjunto de protocolos y estándares para facilitar el intercambio de información y funcionalidad entre aplicaciones. Permite el intercambio de peticiones y respuestas entre diferentes sistemas a través de internet, independientemente del lenguaje en el que fueron desarrollados. En Business Central contamos con tres tipos de servicios web: SOAP, OData y API REST.

El protocolo SOAP[2], siglas de Simple Object Access Protocol, permite que dos objetos de diferentes aplicaciones se comuniquen mediante lenguaje XML. Gracias al uso del XML permite invocar funciones en muchos lenguajes diferentes, lo que le confiere una gran interoperabilidad. A su vez, es más seguro que otros servicios web, pero es más lento que los servicios REST y al estar tan estandarizado es poco flexible. Para usarlo en Business Central basta con crear el objeto que se necesita publicar como servicio web, añadirlo a Business Central y publicarlo como servicio web: un objeto Page para leer o modificar la base de datos, o un objeto Codeunit si se necesita mayor flexibilidad, pues permite exponer funciones propias como operaciones. Cabe destacar que Microsoft ha anunciado que va a descatalogar este tipo de servicios de Business Central, por lo que ya no son una opción viable.

Por otro lado, REST (REpresentational State Transfer)[3] es un estilo de arquitectura software que define los estándares para intercambiar información entre dos sistemas mediante el protocolo HTTP. REST es guiado por 6 principios de ingeniería del software que son: arquitectura Cliente-Servidor, habilitación y uso de la caché, ausencia de estado, sistema en capas, interfaz uniforme y código bajo demanda.

En cuanto a los servicios API REST, cualquier API (Application Programming Interfaces) que cumpla los principios anteriores es API REST. Estos servicios permiten que los servidores y los clientes que los consuman estén desarrollados en cualquier lenguaje siempre que envíen las peticiones en un formato adecuado, en el caso de Business Central en JSON(JavaScript Object Notation). Esto le otorga mucha flexibilidad, requiere un menor ancho de banda que XML y facilita su implementación. Además, debido a la posibilidad de cachear las cabeceras de las llamadas HTTP es más ligero y eficiente, pero puede suponer un problema de seguridad tener expuestas las URIs. De momento en Business Central solo se pueden publicar objetos Page y Query como API REST, por lo que será necesario complementarlo con servicios OData si se necesita publicar funcionalidad a través de un objeto Codeunit. Para publicarlos basta con crear el objeto API PAGE o API QUERY y subirlo a Business Central, será accesible sin necesidad de publicarlo manualmente.

Por lo referente al protocolo OData[18], siglas de Open Data Protocol, se trata de un protocolo abierto que facilita la creación y uso de APIs REST. Permite que los clientes web puedan consultar y editar datos dentro de las redes corporativas y a través de internet mediante el intercambio de mensajes HTTP. OData es un estándar de buenas prácticas para crear y usar servicios API REST. Por tanto, debe seguir los principios REST. Al igual que los servicios API REST, sus principales características son su flexibilidad, ligereza y facilidad de implementación, sumado a que este protocolo simplifica el proceso al añadir cierta estandarización. Business Central admite los formatos JSON y AtomPub[26]. Se implementa de forma similar a los servicios SOAP. En primer lugar se crea el objeto ya sea Page, Codeunit o Query y se sube a Business Central para más tarde publicarlo como servicio web.

3.1.2. Definición de requisitos

Como se ha comentado en puntos anteriores, el proyecto consiste en realizar una transformación tecnológica de unos servicios SOAP a unos servicios REST, documentando el proceso y realizando pruebas para comprobar su funcionamiento. Por tanto, los requisitos que se han definido para dichos servicios son los siguientes:

- El servicio debe poder calcular la suma de dos números enteros y/o guardar el resultado.
- El servicio debe poder calcular el producto de dos números enteros y/o guardar el resultado.
- El servicio debe poder calcular el resultado de concatenar dos cadenas y/o guardar el resultado.
- El servicio debe poder calcular el resultado de cambiar los espacios en blanco de una cadena por el símbolo (_) y/o guardar el resultado.

- El servicio de tipo página debe poder leer, crear, modificar y eliminar un objeto Dato.
- El resultado de los servicios REST siempre debe coincidir con el resultado del servicio SOAP.
- El servicio debe poder conectar Business Central con software externo.

3.1.3. Casos de Uso

El diagrama de casos de uso permite ver como interactúan el sistema y los usuarios o actores. A partir de los requisitos definidos en el apartado anterior, se ha generado un diagrama de casos de uso del sistema que podemos observar en la figura 3.1 junto con las especificaciones de dichos casos de uso en los cuadros 3.1-3.8. En este proyecto se tienen dos actores principales los cuales pueden realizar las mismas funciones: Business Central y un aplicativo externo.

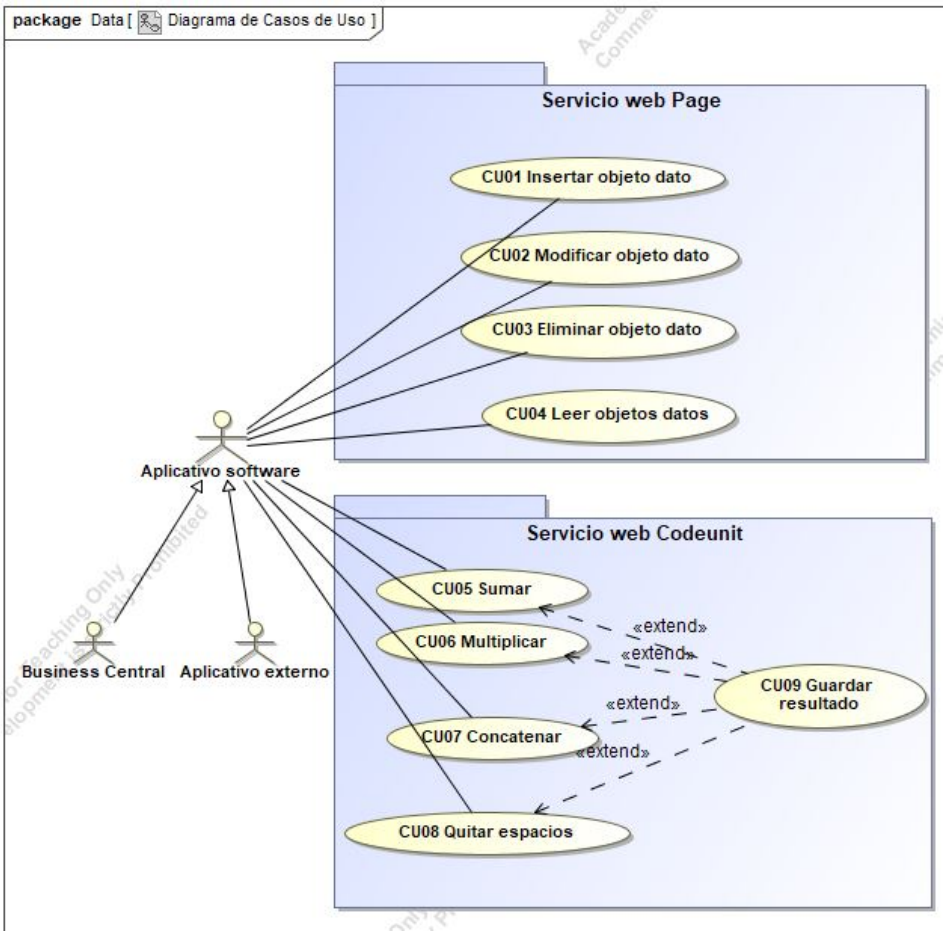


Figura 3.1: Diagrama de Casos de Uso

Business Central no solo publica el servicio web, también es capaz de usarlo para realizar operaciones. De hecho, Business Central permite usar nativamente todos los servicios web creados excepto los de tipo API REST, pues para acceder a ellos es necesario realizar una petición HTTP como si se tratase de un servicio externo. Por otro lado, el aplicativo externo es el software desde el cual se quiere interactuar con Business Central.

Especificación del caso de uso	
Código	CU01
Nombre	Insertar objeto dato
Versión	1.0
Autor	Nieves Cubedo Hurtado
Fuente	Unidad de Negocio BC NAV Puertos Castellón
Descripción	El servicio debe permitir al usuario crear un nuevo dato.
Alcance	Desde que se recoge la información hasta que se informa al usuario.
Nivel	Tarea principal
Actor principal	Business Central y Aplicativo externo
Actores secundarios	
Relaciones	
Precondición	El usuario debe estar dado de alta en el sistema.
Condición fin con éxito	Se ha creado un dato nuevo y se ha añadido a la lista de datos.
Condición fin con fracaso	No se ha podido crear un dato nuevo y no se ha añadido a la lista de datos.
Trigger	El servicio recibe una petición de inserción.
Secuencia normal	**Acción**
1	El usuario redacta la petición de inserción, indicando la autenticación necesaria.
2	Se envía la petición al servicio.
3	El servicio procesa la petición y crea un nuevo dato.
4	El servicio envía una respuesta al usuario indicando que se ha realizado con éxito y devolviendo la información del nuevo dato creado.
Excepción <acción 3>	**Excepción**
1	El servicio no crea el dato nuevo.
2	El servicio envía una respuesta indicando el código de error.
Frecuencia esperada	50 veces al día
Importancia	Muy necesario
Prioridad	Corto plazo
Comentarios	

Cuadro 3.1: Especificación del caso de uso CU01

Especificación del caso de uso	
Código	CU02
Nombre	Modificar objeto dato
Versión	1.0
Autor	Nieves Cubedo Hurtado
Fuente	Unidad de Negocio BC NAV Puertos Castellón
Descripción	El servicio debe permitir al usuario modificar un dato.
Alcance	Desde que se recoge la información hasta que se informa al usuario.
Nivel	Tarea principal
Actor principal	Business Central y Aplicativo externo
Actores secundarios	
Relaciones	CU01
Precondición	El usuario debe estar dado de alta en el sistema y debe existir el dato que se pretende modificar.
Condición fin con éxito	Se ha modificado el dato y se han guardado los cambios.
Condición fin con fracaso	No se ha podido modificar el dato y no se han guardado los cambios.
Trigger	El servicio recibe una petición de modificación.
Secuencia normal	**Acción**
1	El usuario redacta la petición de modificación, indicando la autenticación necesaria.
2	Se envía la petición al servicio.
3	El servicio procesa la petición y modifica el dato.
4	El servicio envía una respuesta al usuario indicando que se ha realizado con éxito y devolviendo la información actualizada del dato.
Excepción <acción 3>	**Excepción**
1	El servicio no modifica el dato.
2	El servicio envía una respuesta indicando el código de error.
Frecuencia esperada	10 veces al día
Importancia	Muy necesario
Prioridad	Corto plazo
Comentarios	

Cuadro 3.2: Especificación del caso de uso CU02

Especificación del caso de uso	
Código	CU03
Nombre	Eliminar objeto dato
Versión	1.0
Autor	Nieves Cubedo Hurtado
Fuente	Unidad de Negocio BC NAV Puertos Castellón
Descripción	El servicio debe permitir al usuario eliminar un dato.
Alcance	Desde que se recoge la información hasta que se informa al usuario.
Nivel	Tarea principal
Actor principal	Business Central y Aplicativo externo
Actores secundarios	
Relaciones	CU01
Precondición	El usuario debe estar dado de alta en el sistema y debe existir en el sistema el dato que se pretende eliminar.
Condición fin con éxito	Se ha eliminado el dato del sistema.
Condición fin con fracaso	No se ha podido eliminar el dato.
Trigger	El servicio recibe una petición de eliminación.
Secuencia normal	**Acción**
1	El usuario redacta la petición de eliminación, indicando la autenticación necesaria.
2	Se envía la petición al servicio.
3	El servicio procesa la petición y elimina el dato.
4	El servicio envía una respuesta al usuario indicando que se ha realizado con éxito.
Excepción <acción 3>	**Excepción**
1	El servicio no elimina el dato.
2	El servicio envía una respuesta indicando el código de error.
Frecuencia esperada	10 veces al día
Importancia	Muy necesario
Prioridad	Corto plazo
Comentarios	

Cuadro 3.3: Especificación del caso de uso CU03

Especificación del caso de uso	
Código	CU04
Nombre	Leer objetos datos
Versión	1.0
Autor	Nieves Cubedo Hurtado
Fuente	Unidad de Negocio BC NAV Puertos Castellón
Descripción	El servicio debe permitir al usuario consultar los datos disponibles.
Alcance	Desde que se recoge la información hasta que se informa al usuario.
Nivel	Tarea principal
Actor principal	Business Central y Aplicativo externo
Actores secundarios	
Relaciones	
Precondición	El usuario debe estar dado de alta en el sistema.
Condición fin con éxito	Los datos se muestran.
Condición fin con fracaso	Los datos no se muestran.
Trigger	El servicio recibe una petición de lectura.
Secuencia normal	**Acción**
1	El usuario redacta la petición de lectura, indicando la autenticación necesaria y si hay que aplicar algún filtro a los datos.
2	Se envía la petición al servicio.
3	El servicio procesa la petición.
4	El servicio envía una respuesta al usuario con los datos que cumplen los requisitos.
Excepción <acción 4>	**Excepción**
1	El servicio no muestra datos.
Frecuencia esperada	20 veces al día
Importancia	Muy necesario
Prioridad	Corto plazo
Comentarios	

Cuadro 3.4: Especificación del caso de uso CU04

Especificación del caso de uso	
Código	CU05
Nombre	Sumar
Versión	1.0
Autor	Nieves Cubedo Hurtado
Fuente	Unidad de Negocio BC NAV Puertos Castellón
Descripción	El servicio debe permitir al usuario sumar dos números enteros.
Alcance	Desde que se recoge la información hasta que se informa al usuario.
Nivel	Tarea principal
Actor principal	Business Central y Aplicativo externo
Actores secundarios	
Relaciones	
Precondición	El usuario debe estar dado de alta en el sistema.
Condición fin con éxito	Se muestra el resultado de la suma.
Condición fin con fracaso	No se muestra el resultado de la suma.
Trigger	El servicio recibe una petición de suma de dos enteros.
Secuencia normal	**Acción**
1	El usuario redacta la petición de suma de dos enteros, indicando la autenticación necesaria.
2	Se envía la petición al servicio.
3	El servicio procesa la petición, calcula el resultado y lo guarda (CU09) junto a los dos enteros en un nuevo dato.
4	El servicio envía una respuesta al usuario con el resultado de la suma.
Excepción <acción 3>	**Excepción**
1	El servicio no realiza la suma y no guarda la información.
2	El servicio envía una respuesta indicando el código de error.
Frecuencia esperada	10 veces al día
Importancia	Muy necesario
Prioridad	Corto plazo
Comentarios	

Cuadro 3.5: Especificación del caso de uso CU05

Especificación del caso de uso	
Código	CU06
Nombre	Multiplicar
Versión	1.0
Autor	Nieves Cubedo Hurtado
Fuente	Unidad de Negocio BC NAV Puertos Castellón
Descripción	El servicio debe permitir al usuario multiplicar dos números enteros.
Alcance	Desde que se recoge la información hasta que se informa al usuario.
Nivel	Tarea principal
Actor principal	Business Central y Aplicativo externo
Actores secundarios	
Relaciones	
Precondición	El usuario debe estar dado de alta en el sistema.
Condición fin con éxito	Se muestra el resultado del producto.
Condición fin con fracaso	No se muestra el resultado del producto.
Trigger	El servicio recibe una petición de producto de dos enteros.
Secuencia normal	**Acción**
1	El usuario redacta la petición de producto de dos enteros, indicando la autenticación necesaria.
2	Se envía la petición al servicio.
3	El servicio procesa la petición, calcula el resultado y lo guarda (CU09) junto a los dos enteros en un nuevo dato.
4	El servicio envía una respuesta al usuario con el resultado del producto.
Excepción <acción 3>	**Excepción**
1	El servicio no realiza el producto y no guarda la información.
2	El servicio envía una respuesta indicando el código de error.
Frecuencia esperada	10 veces al día
Importancia	Muy necesario
Prioridad	Corto plazo
Comentarios	

Cuadro 3.6: Especificación del caso de uso CU06

Especificación del caso de uso	
Código	CU07
Nombre	Concatenar
Versión	1.0
Autor	Nieves Cubedo Hurtado
Fuente	Unidad de Negocio BC NAV Puertos Castellón
Descripción	El servicio debe permitir al usuario concatenar dos cadenas de texto.
Alcance	Desde que se recoge la información hasta que se informa al usuario.
Nivel	Tarea principal
Actor principal	Business Central y Aplicativo externo
Actores secundarios	
Relaciones	
Precondición	El usuario debe estar dado de alta en el sistema.
Condición fin con éxito	Se muestra el resultado de la concatenación.
Condición fin con fracaso	No se muestra el resultado de la concatenación.
Trigger	El servicio recibe una petición de concatenación de dos cadenas de texto.
Secuencia normal	**Acción**
1	El usuario redacta la petición de concatenación de dos cadenas, indicando la autenticación necesaria.
2	Se envía la petición al servicio.
3	El servicio procesa la petición, calcula el resultado y lo guarda (CU09) junto a las dos cadenas en un nuevo dato.
4	El servicio envía una respuesta al usuario con el resultado de la concatenación.
Excepción <acción 3>	**Excepción**
1	El servicio no realiza la concatenación y no guarda la información.
2	El servicio envía una respuesta indicando el código de error.
Frecuencia esperada	10 veces al día
Importancia	Muy necesario
Prioridad	Corto plazo
Comentarios	

Cuadro 3.7: Especificación del caso de uso CU07

Especificación del caso de uso	
Código	CU08
Nombre	Quitar espacios
Versión	1.0
Autor	Nieves Cubedo Hurtado
Fuente	Unidad de Negocio BC NAV Puertos Castellón
Descripción	El servicio debe permitir al usuario cambiar los espacios en blanco por barras bajas de una cadena.
Alcance	Desde que se recoge la información hasta que se informa al usuario.
Nivel	Tarea principal
Actor principal	Business Central y Aplicativo externo
Actores secundarios	
Relaciones	
Precondición	El usuario debe estar dado de alta en el sistema.
Condición fin con éxito	Se muestra el resultado de la operación.
Condición fin con fracaso	No se muestra el resultado de la operación.
Trigger	El servicio recibe una petición del método quitarEspacios, el cual cambia los espacios en blanco por barras bajas de una cadena.
Secuencia normal	**Acción**
1	El usuario redacta la petición de la operación, indicando la autenticación necesaria.
2	Se envía la petición al servicio.
3	El servicio procesa la petición, calcula el resultado y lo guarda (CU09) junto a la cadena en un nuevo dato.
4	El servicio envía una respuesta al usuario con el resultado de la operación.
Excepción <acción 3>	**Excepción**
1	El servicio no realiza la operación y no guarda la información.
2	El servicio envía una respuesta indicando el código de error.
Frecuencia esperada	10 veces al día
Importancia	Muy necesario
Prioridad	Corto plazo
Comentarios	

Cuadro 3.8: Especificación del caso de uso CU08

3.1.4. Diagrama de clases

Un diagrama de clases muestra la estructura del sistema a través de las clases que lo componen y como se relacionan entre sí, indicando también, qué atributos y operaciones poseen dichas clases.

Por un lado, el sistema tiene una parte en la que se deben crear los servicios web SOAP y sus equivalentes OData y API REST. Este trabajo de desarrollo se ha elaborado usando la herramienta Visual Studio Code. En la figura 3.2 se puede ver el diagrama de clases diseñado para la creación de los servicios web. En el diagrama se aprecian cuatro clases relacionadas entre sí, y una sin relacionar. Las clases relacionadas representan los servicios OData y SOAP, pues al publicar un objeto en Business Central como servicio web se publica en ambos protocolos. La clase sin relacionar correspone al equivalente API PAGE para las clases Datos y ListaDatos.

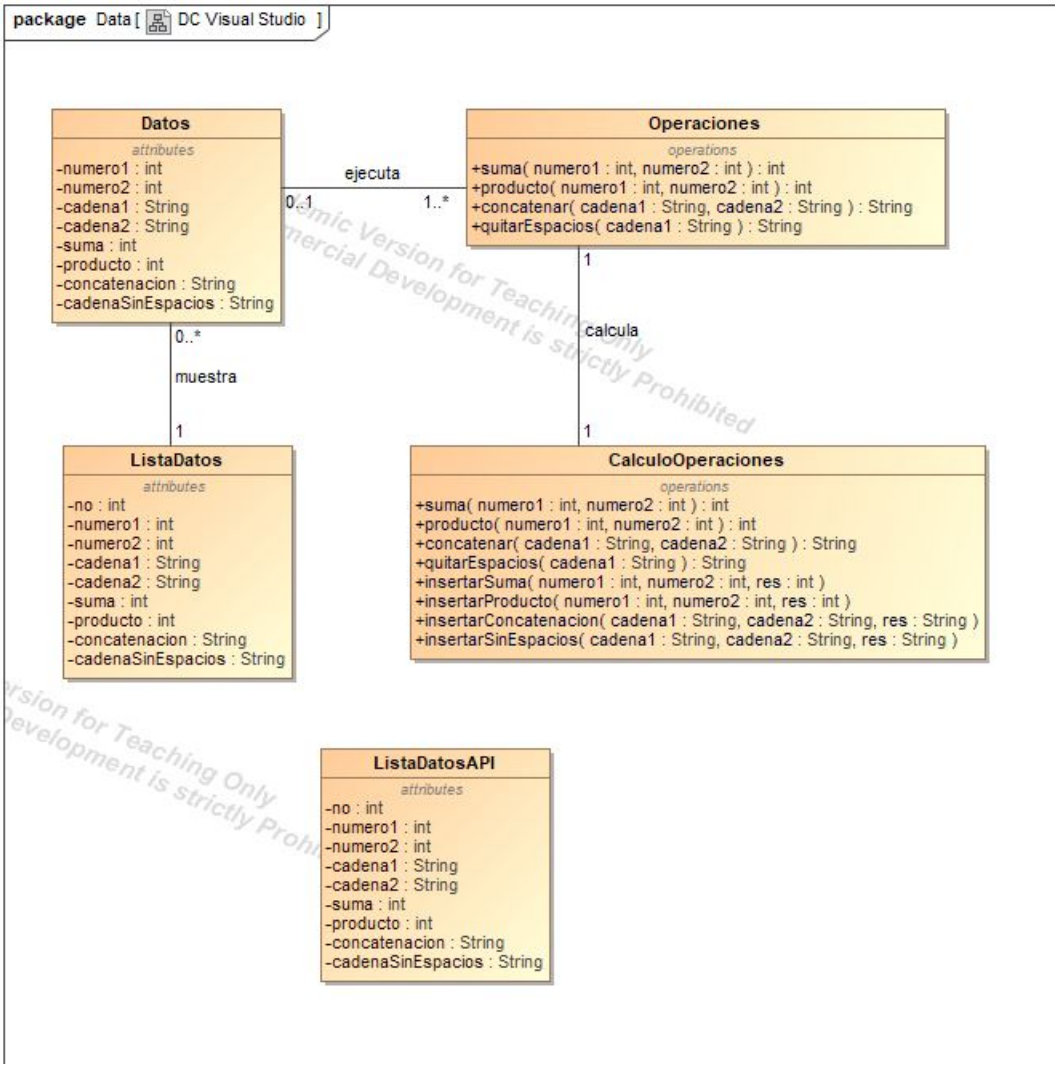


Figura 3.2: Diagrama de clases en Visual Studio Code

A continuación, se definen brevemente las clases del diagrama:

- **Datos:** Esta clase se utiliza para almacenar un nuevo objeto o eliminarlo o modificarlo una vez creado. Contiene el identificador, aunque está oculto; dos enteros y dos cadenas con los que almacenar los datos iniciales con los que se realizan los cálculos; dos enteros donde guardar el resultado de la suma y el producto respectivamente; y dos cadenas para guardar el resultado de la concatenación y la transformación de espacios en blanco a barras bajas en una cadena. El identificador se asigna automáticamente y no es modificable. Además, permite realizar diversos cálculos a través de acciones usando los métodos de la clase Operaciones.

- **ListaDatos:** Esta clase permite visualizar el listado de datos guardados en el sistema. Los atributos que contiene son los mismos que la clase Datos más el identificador, pues al tratarse de un listado estos serán los atributos que muestre.

- **Operaciones:** Esta clase consta de 4 operaciones sencillas que llaman a otra clase llamada CalculoOperaciones para realizar los cálculos y guardar los datos en la tabla:

  - Suma: Recibe dos enteros, devuelve su suma y guarda los datos creando un nuevo objeto Datos.
  - Producto: Recibe dos enteros, devuelve su producto y guarda los datos creando un nuevo objeto Datos.
  - Concatenar: Recibe dos cadenas, devuelve una cadena con el resultado de concatenar las dos cadenas de entrada y guarda los datos creando un nuevo objeto Datos.
  - QuitarEspacios: Recibe una cadena, devuelve una cadena idéntica donde se sustituyen los espacios en blanco por barras bajas (_) y guarda los datos creando un nuevo objeto Datos.

- **CalculoOperaciones:** Esta clase realiza todos los cálculos de las operaciones y guarda los datos de dichas operaciones en la base de datos de Business Central. Las operaciones que realiza son las siguientes:

  - Suma: Recibe dos enteros y devuelve su suma.
  - Producto: Recibe dos enteros y devuelve su producto.
  - Concatenar: Recibe dos cadenas y devuelve una cadena con el resultado de concatenar las dos cadenas de entrada.
  - QuitarEspacios: Recibe una cadena y devuelve una cadena idéntica donde se sustituyen los espacios en blanco por barras bajas (_).
  - insertarSuma: Recibe tres enteros, dos de ellos son los datos iniciales y el tercero es el resultado de su suma, y los guarda creando un nuevo objeto Datos.
  - insertarProducto: Recibe tres enteros, dos de ellos son los datos iniciales y el tercero es el resultado de su producto, y los guarda creando un nuevo objeto Datos.
  - insertarConcatenacion: Recibe tres cadenas, dos de ellas son los datos iniciales y la tercera es el resultado de su concatenación, y las guarda creando un nuevo objeto Datos.
  - insertarSinEspacios: Recibe dos cadenas, una de ellas es el dato inicial y la segunda es el resultado de cambiar los espacios en blanco por barras bajas, y las guarda creando un nuevo objeto Datos.

- **ListaDatosAPI:** Clase equivalente al servicio Datos o listaDatos pero para servicios API REST. Presenta los mismos atributos que listaDatos. No es necesario crear un listado de datos o una ficha de datos pues este tipo de clases no tiene interfaz y el servicio sería idéntico. A diferencia de la clase Datos, al no tener interfaz no puede realizar acciones que no estén automatizadas.

Por otro lado, se ha diseñado otro Diagrama de clases, visible en la figura 3.3, que muestra la estructura que tiene el sistema para consumir servicios web desde Java. Esta parte del desarrollo se ha elaborado usando el editor de código IntelliJ IDEA. En este caso, todas las operaciones se realizan en la clase Main y el resto de clases se usan para definir los objetos que se necesitan.

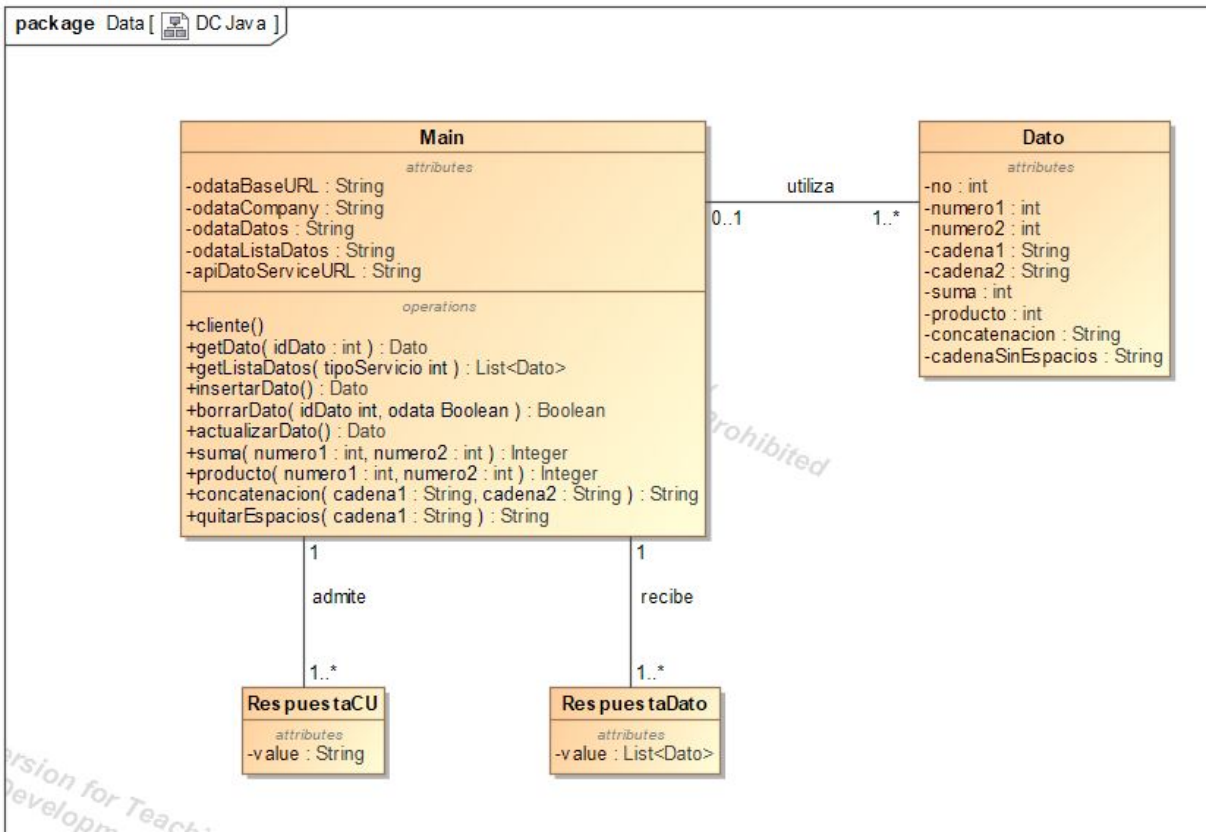


Figura 3.3: Diagrama de clases en Java

Procedemos pues a definir las clases del diagrama:

- **Main:** Esta clase se encarga de realizar todas las operaciones y peticiones a servicios web de Business Central desde Java. Para ello, contiene los enlaces que necesita almacenados en fragmentos, pues algunos enlaces no necesitan todos los fragmentos. Las operaciones de las que dispone son las siguientes:
  - cliente: Contiene la definición del cliente de la petición HTTP.
  - getDato: Realiza una petición a los servicios web con el fin de obtener un objeto Dato concreto a partir de un identificador dado.

- getListaDatos: Realiza una petición a los servicios web con el fin de obtener el listado de objetos Dato almacenado en Business Central.
- insertarDato: Dados los atributos de un objeto dato, se envía una petición al servicio web correspondiente para insertar dicho dato en el sistema.
- borrarDato: Dado un identificador, realiza una petición de borrado para que el objeto dato con dicho identificador sea eliminado.
- actualizarDato: Dados los atributos de un objeto dato, se envía una petición al servicio web correspondiente para modificar dicho dato en el sistema.
- suma: Dados dos números enteros, realiza una petición al servicio web correspondiente para obtener el resultado de dicha suma. Los datos y el resultado se guardan en el sistema.
- producto: Dados dos números enteros, realiza una petición al servicio web correspondiente para obtener el resultado de dicho producto. Los datos y el resultado se guardan en el sistema.
- concatenación: Dadas dos cadenas, realiza una petición al servicio web correspondiente para obtener el resultado de dicha concatenación. Los datos y el resultado se guardan en el sistema.
- quitarEspacios: Dada una cadena, realiza una petición al servicio web correspondiente para obtener el resultado de cambiar los espacios en blanco por barras bajas. Los datos y el resultado se guardan en el sistema.

- **Dato:** Esta clase define la estructura de un objeto Dato, de esta forma, cuando se realice una petición a los servicios web y se obtengan objetos datos completos se pueden crear directamente de forma sencilla.
- **RespuestaCU:** Esta clase facilita recoger la respuesta de una petición a una operación de cálculo en Business Central.
- **RespuestaDato:** Esta clase facilita recoger la respuesta de una petición para obtener la lista de Datos almacenada en el sistema.

No se han realizado diagramas para la consumición de los servicios web usando .Net y la extensión REST Client de Visual Studio Code, puesto que el desarrollo solo consta de una clase donde se realizan todas las operaciones. Además, se complementa con el uso de extensiones, por lo que los diagramas son demasiado sencillos.

3.2. Diseño de la arquitectura del sistema

En este punto se van a describir las distintas secciones que forman parte de este proyecto y como se relacionan entre sí. Por un lado, imaginemos que tenemos un cliente que cuenta con una solución de Business Central para llevar la gestión de su empresa. Este cliente cuenta con una serie de servicios web de tipo SOAP que le permiten compartir toda la información que necesita con otros aplicativos externos de una forma robusta y segura. Por otro lado, estos aplicativos externos pueden estar desarrollados en cualquier lenguaje y necesitan conectarse a los servicios web de nuestro cliente para obtener cierta información. Puesto que Microsoft ha anunciado que

los servicios SOAP van a ser eliminados de Business Central, es necesario realizar una migración de estos a REST, ya sea OData o API REST. Por tanto, se pasará de utilizar servicios SOAP que utilizan lenguaje XML a servicios OData y API REST que utilizan JSON.

En este proyecto, Business Central es el emisor de información mediante el uso de servicios web, por lo que responde a las peticiones enviadas por los aplicativos externos con el fin de que estos puedan obtener la información que necesiten. El flujo de datos comienza con el aplicativo externo elaborando y enviando una petición HTTP a los servicios web REST de Business Central, se procesa dicha petición, se realizan los cálculos que sean necesarios y se envía la respuesta de vuelta al aplicativo externo. Cabe destacar que este intercambio lo hemos visto representado en la figura 1.1 anteriormente vista.

3.3. Diseño de la interfaz

El diseño de la interfaz de Business Central es elegante, moderno y limpio, sin demasiados elementos decorativos. La paleta de colores elegida es sencilla, en blanco y negro con tonos azules para resaltar los elementos importantes y dar un poco de color. Las ventanas son personalizables, pero en general contienen mucha información sin llegar a sobrecargar la pantalla. En general, la interfaz es muy intuitiva y presenta un estilo minimalista centrado en mostrar la mayor cantidad de información posible sin dejar de lado el diseño. En la figura 3.4 podemos apreciar el menú de inicio de Business Central.

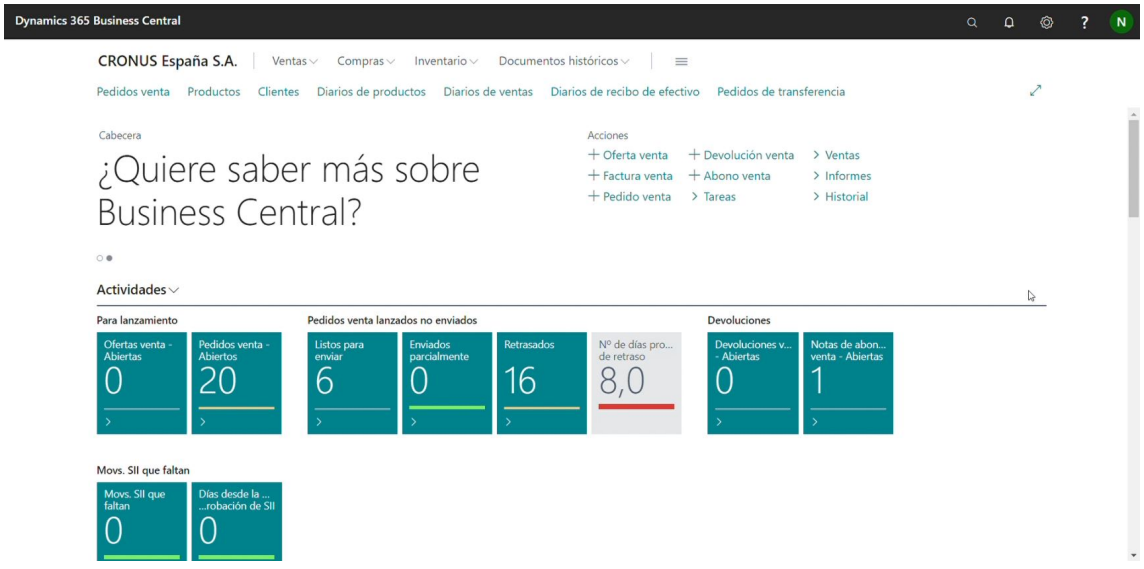


Figura 3.4: Menú de inicio de Business Central

En este proyecto el diseño de la interfaz no tiene una gran relevancia, pues los objetivos se centran en tratar servicios web los cuales no cuentan con interfaz gráfica. A pesar de esto, para poder publicar algunos servicios SOAP y OData, se han tenido que crear dos objetos página los cuales tienen interfaz gráfica y podemos observarlas en las figuras 3.5 y 3.6. Como nos hemos centrado en su funcionalidad como servicios web, no nos hemos interesado en que su interfaz sea

intuitiva o cómoda, se ha usado simplemente la que viene predeterminada. Además, al tratarse de un proyecto de ejemplo para ilustrar como realizar una transformación de un tipo de servicio web a otro, está centrado en demostrar que la implementación funciona correctamente y no en que sea fácil de usar para un usuario común.

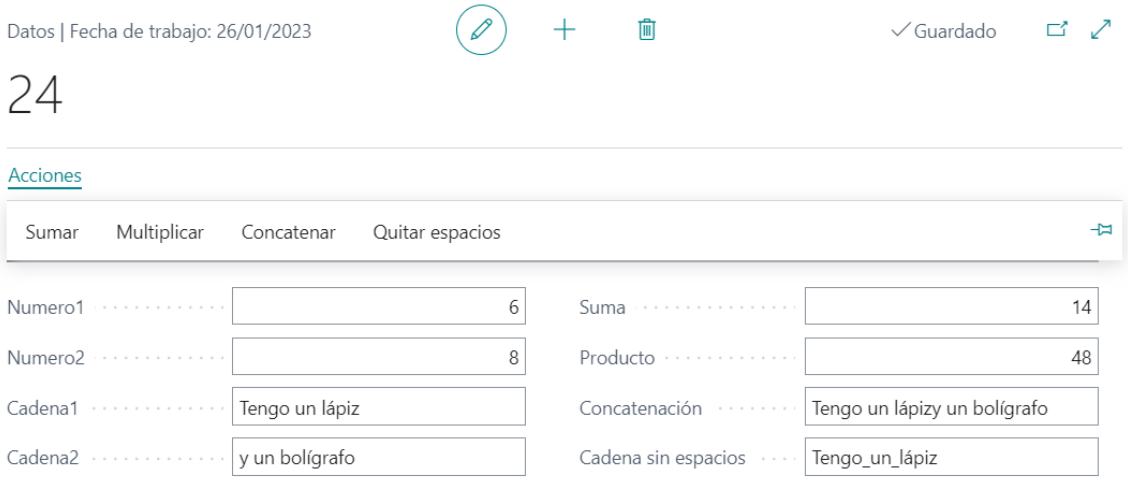
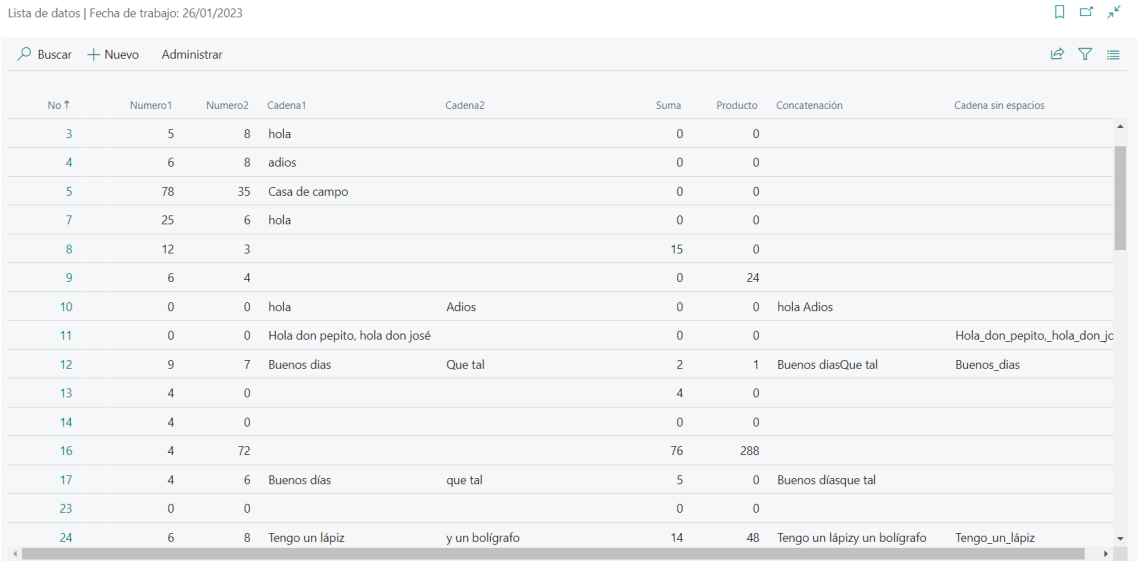


Figura 3.5: Página Datos desde dentro de Business Central



No	Numero1	Numero2	Cadena1	Cadena2	Suma	Producto	Concatenación	Cadena sin espacios
3	5	8	hola		0	0		
4	6	8	adios		0	0		
5	78	35	Casa de campo		0	0		
7	25	6	hola		0	0		
8	12	3			15	0		
9	6	4			0	24		
10	0	0	hola	Adios	0	0	hola Adios	
11	0	0	Hola don pepito, hola don josé		0	0		Hola_don_pepito,_hola_don_jc
12	9	7	Buenos dias	Que tal	2	1	Buenos diasQue tal	Buenos_dias
13	4	0			4	0		
14	4	0			0	0		
16	4	72			76	288		
17	4	6	Buenos días	que tal	5	0	Buenos díasque tal	
23	0	0			0	0		
24	6	8	Tengo un lápiz	y un bolígrafo	14	48	Tengo un lápizy un bolígrafo	Tengo_un_lápiz

Figura 3.6: Página ListaDatos desde dentro de Business Central

Capítulo 4

Implementación y pruebas

4.1. Objetos básicos de Microsoft Dynamics 365 Business Central

Existen distintos tipos de objetos en Business Central que nos facilitan el desarrollo de soluciones. Cada tipo de objeto cuenta con unas características diferentes, por lo que es muy importante conocerlos y saber distinguirlos para así poder aprovechar toda su utilidad. Antes de comenzar a describir la implementación proyecto es necesario aclarar algunos de estos términos referentes a objetos exclusivos de Business Central.

- **Table o tabla:** Estos objetos permiten almacenar y manipular información en Business Central. Las tablas están formadas por los campos, las propiedades, los disparadores y los identificadores.
- **Page o página:** Son una forma de visualizar, modificar, insertar y eliminar información. Tienden a estar conectadas a una o varias tablas de las cuales extraen y manipulan información. Hay diferentes tipos de página dependiendo del comportamiento se necesite que tenga: que actúe como la ficha de un dato, que muestre un listado, entre otros. Tienen tres partes diferenciadas: la primera parte, contiene los atributos y características principales de la página, así como el tipo de página que es y de que tablas está extrayendo los datos; la segunda parte, indica que campos de dichas tablas se van a mostrar; finalmente, la tercera parte está destinada a las acciones.
- **Codeunit:** Son fragmentos de código que permiten crear funciones para que puedan ser utilizadas en distintas partes de la solución sin necesidad de duplicar código.
- **Query:** Consultas que obtienen información de una o varias tablas y la combinan o realizan operaciones con dicha información.
- **Report:** Permiten obtener y mostrar información de la base de datos de Business Central, e incluso resumir y exportar la información en documentos.
- **XMLport:** Son usados para exportar e importar información entre Business Central y un aplicativo externo en formato XML de forma fácil y rápida.

4.2. Detalles de implementación

En este apartado se va a describir detalladamente la implementación del proyecto. Por un lado, se va a explicar el desarrollo de los servicios web. Por otro lado, se va a narrar como consumir dichos servicios para obtener datos de Business Central.

4.2.1. Configuración de Visual Studio Code

Antes de comenzar, es necesario configurar la herramienta Visual Studio Code para que esté conectada a nuestra instancia de Business Central local. En primer lugar se han instalado las siguientes extensiones:

- **AL Extension Pack:** Recopilación de extensiones que facilitan el desarrollo en AL.
- **REST Client:** Extensión que permite enviar peticiones HTTP y visualizar la respuesta en VSC directamente.
- **AL Language Tools:** Extensión que agiliza la creación de ficheros de traducción y el testeo de funciones.
- **Git Graph:** Posibilita la visualización de un gráfico con las ramas y versiones del proyecto. También permite realizar operaciones en nuestra herramienta de control de versiones directamente desde esta extensión, de forma clara y sencilla.

A continuación, se ha creado un nuevo proyecto a partir de una plantilla proporcionada por la empresa. Dicha plantilla contiene algunos ficheros de configuración para facilitar la conexión con Business Central, así como la estructura de las carpetas y ficheros utilizados por la empresa. Pese a esto, es necesario configurar manualmente un par de ficheros para adaptarlos a nuestro caso específico. En el fichero de configuración JSON llamado *app.json* es necesario indicar: la versión de Business Central que se va a utilizar; el atributo *name*, que debe coincidir con el nombre del proyecto; el *target* es el tipo de versión, es decir, si se va a usar una versión local o en la nube; los rangos de los identificadores de los objetos que vamos a crear que deben ser únicos para cada trabajador de la empresa a fin de evitar colisiones; y un número GUID(Globally Unique IDentifier) el cual identifica de forma única nuestra solución. El número GUID se puede obtener gracias a una de las extensiones instaladas, para ello se abre la paleta de comandos y se ejecuta el comando *Create GUID*. Este comando genera un número GUID único y lo guarda en el portapapeles para que se pueda pegar donde sea necesario. El segundo fichero JSON para configurar es *LAUNCH.json*, en este fichero hay que dejar gran parte de las opciones tal y como vienen, excepto: un nombre para la extensión que sea distinguible, pues este nombre será visible desde dentro de Business Central; la versión; el servidor y la instancia del servidor.

Tras terminar de configurar los ficheros, es necesario descargar los símbolos para poder acceder a los objetos base que posee Business Central. Además, esta acción nos ayuda a comprobar si lo hemos configurado correctamente y está conectado con Business Central. Para descargarlos se abre la paleta de comandos y se ejecuta la orden *AL: Download symbols*. Si se ha ejecutado de forma satisfactoria desde la extensión AL Object Designer podremos observar todos los objetos base de nuestro servidor.

4.2.2. Desarrollo de los servicios web

Como se ha explicado en apartados anteriores, el proyecto consiste en crear unos servicios web SOAP, transformarlos en OData y API REST, consumir los servicios y documentar todo el proceso. Por tanto, el primer paso es crear los servicios SOAP.

En primer lugar, se ha creado un objeto tabla donde se almacena toda la información y de donde extraen la información los servicios web. La tabla cuenta con los siguientes campos:

- **no:** Número entero que actúa como identificador de la tabla. Es un valor autoincrementado, de modo que no es necesario calcular cual es el siguiente número cada vez que se crea un nuevo registro. Si al crear un nuevo registro no se especifica este valor, el sistema le asignará automáticamente la cifra inmediatamente posterior tras el último registro creado.
- **numero1:** Número entero donde se guarda el primero de los operandos o un valor entero.
- **numero2:** Número entero donde se almacena el segundo de los operandos o un valor entero.
- **cadena1:** Cadena de texto donde se guarda el primer dato dato inicial de las operaciones con cadenas.
- **cadena2:** Cadena de texto donde se almacena el segundo dato inicial de las operaciones con cadenas.
- **suma:** Número entero donde se guarda el resultado de la suma de numero1 y numero2.
- **producto:** Número entero donde se guarda el resultado del producto de numero1 y numero2.
- **concatenacion:** Cadena de texto donde se almacena el resultado de la concatenación entre cadena1 y cadena2.
- **cadenaSinEspacios:** Cadena de texto donde se almacena el resultado de cambiar los espacios en blanco por barras bajas (_) de la cadena1.

En segundo lugar, se han elaborado dos Codeunit llamadas Operaciones y CalculoOperaciones. En CalculoOperaciones tenemos las cuatro funciones que realizan los distintos cálculos (sumar, multiplicar, concatenar y quitar espacios) y cuatro funciones para guardar los resultados de cada una de las cuatro operaciones disponibles. Se ha separado la funcionalidad de calcular y guardar los resultados por si fuera necesario realizar algún cálculo y no guardarlo. La Codeunit Operaciones cuenta con cuatro métodos que llaman a los métodos de CalculoOperaciones de la siguiente forma: por ejemplo, el método suma llama a los métodos suma e insertarSuma de CalculoOperaciones. Operaciones es la Codeunit que ha sido publicada como servicio web. Se ha realizado así para que sea más seguro y porque es mejor que el cliente no pueda ver como se realizan los cálculos.

A continuación, se han desarrollado dos objetos página: una lista llamada ListaDatos y una ficha llamada Datos. La página de tipo ficha tiene los campos: numero1, numero2, cadena1,

cadena2, suma, producto, concatenacion y cadenaSinEspacios. Como se puede observar, son los mismos campos que la tabla menos el identificador, de este modo cuando se cree un nuevo registro, el identificador se asignará automáticamente. También se le ha indicado cuál es su atributo tipo de página, en este caso *Card* y su atributo *Caption*. Para poder acceder a los datos de la tabla es importante indicarle a la página cuál es la tabla fuente. Esta página ficha cuenta con cuatro acciones que permiten ejecutar los métodos de la Codeunit para realizar los cálculos y actualizar así los campos suma, producto concatenación y cadenaSinEspacios respectivamente. En cambio, la lista usa todos los atributos de la tabla, incluido el identificador. También se le ha indicado que tipo de página es, en este caso *List*, cual es la página ficha para cada registro de la lista, la tabla fuente y el atributo *Caption*. Además, se le han añadido algunos atributos extras para poder encontrar la página desde el buscador de Business Central.

Una vez elaborados los objetos, se han de insertar en Business Central. Se abre la paleta de comandos y se ejecuta la orden *AL: Publish without debugging*. Tras esto se puede entrar en Business Central, buscar los objetos y probar su funcionalidad.

De este modo, tras haber desarrollado los objetos necesarios e insertarlos en Business Central, se procedió a publicar los servicios web SOAP y OData. No hay ninguna diferencia a la hora de publicar un servicio web como SOAP o OData, los pasos son idénticos solo cambia la forma de consumirlo. Para comenzar, se accede a la página de servicios web de Business Central, esta página muestra un listado con los servicios web SOAP y OData disponibles, y se pulsa en Nuevo, lo que genera una línea vacía que se puede rellenar. Una vez rellenados los campos, es importante marcar la opción *Published* para que el objeto seleccionado sea publicado como servicio web. Si se ha realizado con éxito, al recargar la página se habrán rellenado los campos de URL (Localizador de Recursos Uniforme) OData y/o SOAP. Solo se pueden publicar como servicios web los objetos página, codeunit y query. Cuando se publica un objeto de tipo Page, se genera tanto un enlace OData como SOAP. En cambio, cuando se publica un objeto Codeunit solo se genera un enlace SOAP. Si es de tipo Query, solo está disponible como OData. Aunque no haya enlace OData para la Codeunit si que se publica como servicio web, su enlace varía en función de a que método se desee invocar por lo que no hay un único enlace para consumir dicho servicio. En la figura 4.1 se puede observar la página Servicios Web de Business Central mientras se publica un servicio web de tipo página.

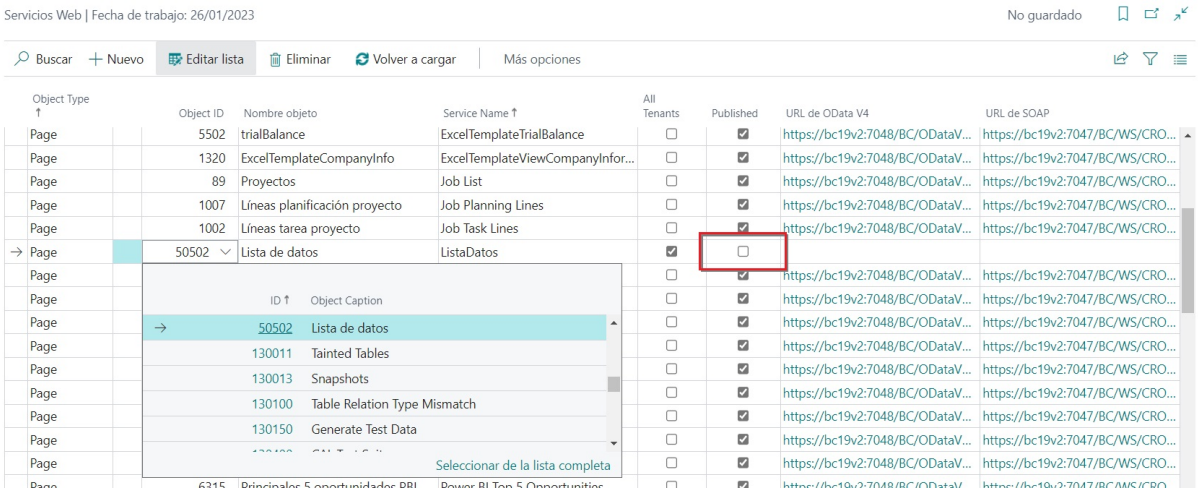


Figura 4.1: Publicación de un servicio web SOAP y OData

Finalmente, los servicios API REST son un poco diferentes. Solo pueden usar objetos página o query y estos no serán accesibles desde Business Central a diferencia de los SOAP y REST. Si se necesitan usar, será necesario consumir los servicios mediante peticiones HTTP como si se tratase de software externo. En nuestro caso, solo hemos podido crear el equivalente de las páginas Datos y ListaDatos, pues al no admitir Codeunit no lo podemos desarrollar. Puesto que este tipo de objetos API no tienen interfaz, el servicio web equivalente de listaDatos y Datos es casi idéntico, solo cambiando en que uno posee identificador y el otro no, por tanto, se ha elaborado solo un servicio web con identificador. Se ha desarrollado pues, una página con los mismos campos que listaDatos pero en este caso es de tipo API y con los atributos necesarios para este tipo de páginas, los cuales pueden verse en la figura 4.2. Los atributos *APIGroup, APIPublisher* y *APIVersion* son importantes pues con ellos se construye la URI con la cual consumir el servicio. Para que sea accesible como servicio web basta con insertar el objeto API en Business Central, no es necesario ningún paso adicional.

```
page 50503 "PREFIX ListaDatosAPI"
{
    APIGroup = 'custom';
    APIPublisher = 'laberit';
    APIVersion = 'v2.0';
    Caption = 'Lista Datos API';
    DelayedInsert = true;
    EntityName = 'listaDatosAPI';
    EntitySetName = 'datos';
    PageType = API;
    SourceTable = "PREFIX Datos";
    CardPageId = "PREFIX Datos";
    ODataKeyFields = SystemId;
    EntityCaption = 'dato';
    EntitySetCaption = 'datos';

    layout
    {
```

Figura 4.2: Publicación de un servicio web SOAP y OData

4.2.3. Endpoints

Un *endpoint* es un enlace URL al cual va a responder un servicio web ya sea para cargar, consumir o mostrar información.

Para consumir un servicio web hay que realizar peticiones HTTP al servidor de Business Central especificando el tipo de petición y el *endpoint* del servicio. Cada servicio presenta un *endpoint* diferente e incluso hay algunos que son variables dependiendo de la tarea que se quiera realizar. Al publicar un servicio web, Business Central genera automáticamente su *endpoint* si se trata de un servicio SOAP o de un servicio de tipo Page OData. Puesto que para este proyecto solo se pide consumir servicios REST, nos centraremos en estos. Para usar servicios web es necesario conocer la estructura de los *endpoints* por si fuera necesario modificar el *endpoint* para realizar una petición diferente. Por todo esto, el primer paso va a ser aprender a construir los *endpoints* de los servicios.

Comenzaremos por los servicios OData, concretamente por las páginas, sus *endopoints* son bastante sencillos y se pueden consultar directamente desde Business Central en la página de servicios web. Su estructura, junto a un ejemplo, es la siguiente:

https://<Servidor>:<PuertoServicioWeb>/<InstanciaServidor>/ODataV4/Company(‘<Nombre de la empresa | IdEmpresa>’)/<Nombre del servicio>

Ejemplo:

`https://bc19v2:7048/BC/ODataV4/Company('CRONUS%20Espa%C3%B1a%20S.A.')/Datos`

A continuación, se va a hablar de los *endpoints* de los objetos codeunit de los servicios OData. Estos son un poco más complejos que los de las páginas puesto que varían en función de a qué método de la codeunit se esté invocando. Su estructura sigue así:

https://<Servidor>:<PuertoServicioWeb>/<InstanciaServidor>/ODataV4/<NombreServicio>_<NombreFuncion>?company=<Nombre de la empresa | IdEmpresa>

Ejemplo:

`https://bc19v2:7048/BC/ODataV4/Operaciones_Suma?company=CRONUS%20Espa%C3%B1a%20S.A.`

Finalmente, solo resta hablar de los *endpoints* de las páginas de tipo API REST. En este caso hay dos tipos los servicios del estándar y los servicios personalizados. Los *endpoints* de los servicios del personalizados dependen de los valores que se asignen a ciertos atributos (*APIGroup, APIPublisher* y *APIVersion*) al crear la página. Estos *endpoints* siguen las siguientes estructuras:

Servicio API estándar:

https://<Servidor>:<PuertoServicioWeb>/<InstanciaServidor>/<APIversion>/companies(<Nombre de la Empresa | IdEmpresa>)/<NombreServicio>

Ejemplo:

`https://bc19v2:7048/BC/api/v2.0/companies(5d3bf4b2-0d57-ec11-bb7e-000d3a21fc0b)/customers`

Servicio API personalizado:

https://<Servidor>:<PuertoServicioWeb>/<InstanciaServidor>/api/<APIpublisher>/<APIgroup>/<APIversion>/<NombreServicio>

Ejemplo: `https://bc19v2:7048/BC/api/laberit/custom/v2.0/datos`

4.2.4. Consumo de los servicios web

Consumir servicios web consiste en realizar peticiones a un servicio web con el fin de extraer, modificar o eliminar información. No hay diferencia a la hora de consumir un servicio OData o API REST, excepto que cambia la estructura del *endpoint*. Solo hay diferencia entre consumir un objeto page o un objeto codeunit.

En el desarrollo de este proyecto se han consumido servicios web REST usando tres lenguajes o herramientas diferentes con el fin de ejemplificar en que podría consistir dicha tarea. A pesar de que cada lenguaje tiene una forma diferente para consumir los servicios, existen algunos elementos comunes: se trata de realizar una petición HTTP utilizando el *endpoint* del servicio a la cual le indicaremos en la cabecera la autenticación a Business Central y el formato de los datos, en el cuerpo le introduciremos los argumentos en caso de ser necesarios. En esta sección veremos como consumir servicios web OData y API REST usando JAVA, .NET y una extensión de Visual Studio Code llamada REST Client.

Consumo de los servicios web usando .Net y C Sharp

En primer lugar, se ha creado un proyecto nuevo en Visual Studio 2019 usando la plantilla Aplicación de consola `C#`. Se ha usado esta versión del programa para poder usar la extensión OData Connected Service, pues esta extensión, a fecha actual cuando se está redactando esta memoria, no soporta versiones más modernas del programa. Después de instalar la extensión, fue necesario configurarla. La extensión nos pide un nombre para los servicios, la dirección que apunte a los metadatos de los servicios web y debemos insertar una cabecera personalizada en la que añadiremos la autentificación de Business Central. El resto de las opciones las dejamos igual y al final le añadimos un nombre al *namespace* y al fichero. Esta extensión genera una clase que facilita el consumo de los servicios web.

En este caso, se ha desarrollado nuestra solución en el método Main. A continuación, se crea una instancia llamada context de la clase generada por la extensión y se le inserta la URI del *endpoint* del servicio web. A esta instancia se le inserta una cabecera con la autentificación a Business Central. El código anteriormente descrito es igual para todos los servicios web, pero hay algunas diferencias dependiendo de si se va a consumir un servicio página o codeunit y del método que se vaya a consumir.

Por un lado, los servicios página permiten realizar operaciones CRUD (Create, Read, Update and Delete), es decir, operaciones de lectura, creado, modificación y eliminación de datos. Si se quiere hacer una petición GET de lectura de datos, usamos la función Execute de la siguiente forma: ≪context.<nombre del servicio>.Execute()≫. El resultado que retorna está función es el listado de datos de la página. Si en cambio se quiere realizar una operación POST de creado de datos, se debe crear un objeto con los atributos deseados, ejecutar la función AddTo<nombre del servicio> pasándole como argumento el objeto creado y ejecutar la función SaveChanges para que los cambios se apliquen a Business Central. Las operaciones PUT de modificación son iguales que las POST excepto porque en vez de crear un objeto nuevo, hay que extraerlo usando una petición GET. Además, la función también es diferente, pues hay que utilizar UpdateObject. Las peticiones DELETE de eliminación se realizan extrayendo el objeto a eliminar y ejecutando

los métodos DeleteObject y SaveChanges.

Por otro lado, los servicios web Codeunit permiten realizar peticiones a funciones propias o personalizadas. Para realizar peticiones POST a una de estas funciones basta con llamar a su función pasándole los parámetros que necesite.

Consumo de los servicios web usando REST Client

Consumir servicios web usando la extensión REST Client de Visual Studio Code es bastante sencillo. Se comienza creando un fichero HTTP. En este fichero se escriben las peticiones directamente, separadas por un salto de línea. Para ejecutar una petición, se selecciona su código y desde la paleta de comandos se ejecuta el comando *Rest Client: Send request*, tras esto se abrirá una pestaña con la respuesta del servicio web a dicha petición. Todas las peticiones HTTP siguen la misma estructura, por ello se va a ejemplificar usando una petición PUT, pues cuenta con todas las características especiales que estas pueden presentar. En la figura 4.3 podemos observar un ejemplo de una petición de modificación de datos.

```
PUT https://bc19v2:7048/BC/ODataV4/Company('CRONUS%20Espa%C3%B1a%20S.A.')/Datos(51) HTTP/1.1
Content-Type: application/json
Authorization: Basic XXXXXXXXXXXXXX
If-Match: *

{
  "numero1": 25,
  "numero2": 6,
  "cadena1": "posa",
  "cadena2": "vasos"
}
```

Figura 4.3: Petición PUT usando la extensión REST Client de Visual Studio Code

La petición comienza indicando el tipo de la petición, en este caso PUT. A continuación, se introduce el *endpoint* del servicio web, en este caso, al final hay un número entre paréntesis que indica el identificador del dato sobre el que se tiene que realizar la modificación. En la cabecera se debe especificar la autentificación a Business en Base64 y como se trata de una petición PUT, es importante añadir el atributo if-Match que en este caso contiene un valor comodín. Los parámetros de la petición, si es que existen, se escriben entre llaves a un salto de línea de distancia de los atributos de la cabecera.

Consumo de los servicios web usando Java

A diferencia de los lenguajes anteriores, en Java no se ha hecho uso de ninguna extensión adicional, solo se ha usado la herramienta Maven para facilitar la importación de las bibliotecas necesarias para desarrollar la solución. La biblioteca que se ha importado utilizando Maven es la biblioteca Jackson. Esta biblioteca entre otras cosas ayuda a transformar un documento JSON o XML en objetos Java y así poder manipular la información más fácilmente.

Para el desarrollo se ha usado el paquete java.net.http para manipular las distintas partes de

una petición HTTP: el cliente, la petición y la respuesta. Al igual que en los lenguajes anteriores hay que realizar peticiones indicando los atributos y parámetros que correspondan a cada tipo de petición. Si se ha necesitado insertar parámetros, se ha usado la clase HashMap para crear un diccionario con los pares clave valor que son los parámetros de la petición y así poderlos insertar de forma sencilla. Cómo se ha indicado anteriormente, se ha usado la biblioteca Jackson para poder transformar de forma sencilla la respuesta de la petición a objetos Java y así poder manipularlos de forma cómoda. Para facilitar la trasnformación, se han creado una serie de clases con tipos de objeto de las respuestas que devuelven los servicios creados, estas son: Dato, RespuestaCU y RespuestaDato.

La clase Dato contiene los mismos atributos que los servicios página de Business Central. La clase RespuestaCU contiene una cadena y se utiliza para obtener las respuestas de los servicios codeunit. Finalmente, la clase RespuestaDato contiene una lista de datos que se utiliza para recoger la respuesta de la petición GET a los servicios página.

La implementación es bastante extensa, pero a grandes rasgos cada petición comienza creando un cliente sin añadirle atributos. A continuación, se crea una petición usando HttpRequest y se le introducen los valores de la cabecera y la URI al servicio web. Tras esto se ejecuta el método send del cliente para enviar la petición, indicándole que la respuesta la queremos como una cadena de texto. Finalmente, transformamos la respuesta en objetos para poder trabajar con ellos.

4.3. Verificación y validación

En este apartado se van a detallar las pruebas realizadas para comprobar el correcto funcionamiento del sistema desarrollado. El obejtivo principal del proyecto es realizar una transformación de un servicio SOAP a REST e investigar cómo se consumen los servicios REST. Por todo esto, se asume que el servicio SOAP es correcto y las pruebas van a consistir principalmente en comparar que los servicios REST generen los mismos resultados que los SOAP.

Durante el desarrollo del sistema se utilizaron las herramientas SoapUI y Postman para lanzar peticiones a los servicios web SOAP y así poder comparar manualmente si la solución desarrollada era similar a la versión SOAP. Asimismo, se usaba la interfaz de la página de ListaDatos para comprobar los cambios en la base de datos del servidor y observar si funcionaba de la forma esperada y los cambios realmente se aplicaban.

Puesto que la parte de consumo de los servicios web se ha elaborado usando distintos lenguajes y herramientas, elaborar test unitarios para todos ellos hubiera sido una tarea monumental y aportaba poco valor al proyecto. Además, de las tres tecnologías utilizadas para consumir servicios durante el proyecto, Java es en la que se tiene más desenvoltura y experiencia. Por ello se ha optado por realizar test unitarios únicamente en Java. Se han elaborado dos baterías de pruebas unitarias: una para los servicios codeunit y otra para los de tipo página.

Ha sido necesario realizar peticiones SOAP equivalentes a las realizadas para REST para poder elaborar las pruebas unitarias y así poder comparar ambos resultados y ver si son iguales. Por supuesto hay algunas diferencias, la forma de realizar las peticiones es algo diferente y la

respuesta se devuelve en XML, pero el contenido debería ser el mismo. Las peticiones se han realizado utilizando el paquete java.net, concretamente mediante la clase HttpURLConnection, observando como construye SoapUI y Postman la petición para poder replicarlo desde Java. El cuerpo de la petición es un fragmento de un documento XML guardado en una cadena y que más tarde se ha transformado en una secuencia de bytes para poder ser insertado en la petición.

El tiempo de ejecución de estas pruebas no depende tanto de los cálculos realizados, si no de las peticiones al servidor realizadas, pues hay muchas variables que influyen en esta velocidad. Por ello se han tratado de reducir el número de peticiones al mínimo posible. Por ejemplo, para el test de obtener la lista de datos, se ha realizado una única petición por cada tipo de servicio, obteniendo el listado de datos y para luego ir comprobando los atributos de cada dato entre sí, en vez de ir realizando peticiones para cada dato. Se han podido desarrollar todos las pruebas con éxito excepto las pruebas para insertar y modificar datos de los servicios página pues generaban el error *Internal Server Error*. Este error se consultó con un experto de la empresa, pero no se consiguió encontrar una solución en el tiempo restante.

Finalmente, en las herramientas para las que no se pudo realizar test unitarios se comprobó el resultado de forma manual igual que durante el desarrollo del proyecto, comparando con los resultados generados por los servicios SOAP utilizando SoapUI y Postman.

Capítulo 5

Conclusiones

En resumen, el proyecto ha consistido en elaborar unos servicios SOAP sencillos, desarrollar los servicios equivalentes a estos en OData y API REST, y consumir dichos servicios en diferentes lenguajes. Todo esto, con el fin de ejemplificar y documentar como se realizaría el proceso de transformación completo de un servicio SOAP a REST.

A nivel formativo, la empresa me proporcionó una formación básica sobre Microsoft Business Central que fue clave para el correcto desarrollo de la solución. He aprendido mucho no solo sobre tecnologías que no había empleado anteriormente como Business Central o los lenguajes AL y `C#`, si no también conceptos sobre de contabilidad y servicios web. De esta forma, este proyecto me ha permitido aplicar los conocimientos aprendidos a lo largo de la carrera en una situación real.

A nivel profesional, me ha parecido una experiencia muy enriquecedora pues he podido ver cómo funciona una empresa de consultoría software desde dentro. He podido observar de primera mano cómo en una empresa de estas características el trabajo en equipo es muy importante y como encuentran soluciones a los problemas que van surgiendo. Ha sido una experiencia muy agradable pues me han tratado como una más y todo el personal ha sido muy atento y amable conmigo.

A nivel personal, esta experiencia me ha ayudado a ganar confianza en mí misma como desarrolladora y ver que con un poco de formación específica me puedo adaptar a cualquier situación que se me presente. Ha sido un gran reto, pero estoy muy contenta de haber tenida esta oportunidad.

En conclusión, el proyecto se ha completado satisfactoriamente, pues se han cumplido todos los objetivos propuestos en el tiempo establecido. El proyecto consta de cinco subobjetivos, de los cuales los dos primeros se pueden ver en los capítulos de análisis e implementación, y los tres últimos se encuentran en la implementación y más detalladamente, en la documentación adjunta en el anexo. Además, me ha parecido una experiencia muy valiosa y ha supuesto un primer contacto con el mundo laboral que me ha dado más seguridad y confianza en mi trabajo.

Bibliografía

[1] Visual Studio Code. Página web de Visual Studio Code. https://code.visualstudio.com/, 2022. [Consulta: 17 de Mayo de 2022].

[2] World Wide Web Consortium. Especificación SOAP. https://www.w3.org/TR/soap/, 2004. [Consulta: 15 de Junio de 2022].

[3] Roy Thomas Fielding. Estilos Arquitectónicos y el Diseño de Arquitecturas de Software basadas en la Red. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000. [Consulta: 16 de Junio de 2022].

[4] Google. Página web de Google Chrome. https://www.google.com/intl/es/chrome/, 2022. [Consulta: 17 de Mayo de 2022].

[5] JetBrains. Página web de IntelliJ IDEA. https://www.jetbrains.com/es-es/idea/, 2022. [Consulta: 17 de Mayo de 2022].

[6] Laberit. Conócenos ¿Qué es Laberit? https://www.laberit.com/conocenos, 2022. [Consulta: 16 de Febrero de 2022].

[7] Huachao Mao. Página web de la extensión Rest Client en la tienda de Visual Studio Code. https://marketplace.visualstudio.com/items?itemName=marketplace.ODataConnectedService, 2022. [Consulta: 17 de Mayo de 2022].

[8] Eduardo Marín. Actualización a Dynamics 365 Business Central 2020. https://www.abd.es/2020/09/actualizacion-a-dynamics-365-business-central-2020/, 2020. [Consulta: 19 de Mayo de 2022].

[9] Microsoft. Información general sobre el primer lanzamiento de versiones de Dynamics 365 Business Central en 2021. https://docs.microsoft.com/es-es/dynamics365-release-plan/2019wave2/dynamics365-business-central/, 2021. [Consulta: 16 de Febrero de 2022].

[10] Microsoft. Información general sobre Microsoft Dynamics NAV. https://dynamics.microsoft.com/es-es/nav-erp/, 2022. [Consulta: 16 de Febrero de 2022].

[11] Microsoft. Programando en al. https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/devenv-programming-in-al, 2022. [Consulta: 19 de Mayo de 2022].

[12] Microsoft. Página web de Azure DevOps Server. https://azure.microsoft.com/es-es/services/devops/server/, 2022. [Consulta: 17 de Mayo de 2022].

[13] Microsoft. Página web de Dynamics 365 Business Central. https://dynamics.microsoft.com/es-es/business-central/overview/, 2022. [Consulta: 17 de Mayo de 2022].

[14] Microsoft. Página web de Microsoft Edge. https://www.microsoft.com/es-es/edge, 2022. [Consulta: 17 de Mayo de 2022].

[15] Microsoft. Página web de Microsoft Project. https://www.microsoft.com/es-es/microsoft-365/project/project-management-software, 2022. [Consulta: 17 de Mayo de 2022].

[16] Microsoft. Servicios Web de Business Central. https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/webservices/web-services, 2022. [Consulta: 1 de marzo de 2022].

[17] Microsoft. Servicios Web SOAP. https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/webservices/soap-web-services, 2022. [Consulta: 16 de Febrero de 2022].

[18] Odata. Página web de OData. https://www.odata.org/, 2022. [Consulta: 16 de Junio de 2022].

[19] Postman. Página web de Postman. https://www.postman.com/, 2022. [Consulta: 17 de Mayo de 2022].

[20] OData Connected Service. Página web de la extensión OData Connected Service en la tienda de Visual Studio. https://marketplace.visualstudio.com/items?itemName=marketplace.ODataConnectedService, 2022. [Consulta: 17 de Mayo de 2022].

[21] SmartBear Software. Página web de SoapUI. https://www.soapui.org/, 2022. [Consulta: 17 de Mayo de 2022].

[22] Microsoft Visual Studio. Página web de Microsoft Visual Studio. https://visualstudio.microsoft.com/es/vs/, 2022. [Consulta: 17 de Mayo de 2022].

[23] Dassault Systèmes. Página web de Magic Draw. https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/, 2022. [Consulta: 17 de Mayo de 2022].

[24] Wikipedia. Página web de wikipedia sobre servicios web. https://es.wikipedia.org/wiki/Servicio_web, 2021. [Consulta: 20 de Mayo de 2022].

[25] Wikipedia. Página web de wikipedia sobre Desarrollo en cascada. https://es.wikipedia.org/wiki/Desarrollo_en_cascada, 2022. [Consulta: 11 de mayo de 2022].

[26] Mark Nottingham y Robert Sayre. Página web de Atom. https://datatracker.ietf.org/doc/rfc4287/, 2005. [Consulta: 16 de Junio de 2022].

Anexo A

En este apartado se adjunta la documentación sobre la migración elaborada para la empresa.

Adaptación tecnológica de Servicios Web de MS Dynamics NAV a MS Dynamics 365 Business Central

Fecha 20 Abril 2022

LABERIT
PEOPLE + INNOVATION + TECHNOLOGY

Índice

1. Introducción

Los servicios web SOAP están cada vez más en desuso. Microsoft advierte de que va a eliminar los servicios SOAP de Business Central en futuras versiones, por lo que recomienda migrar a REST lo antes posible.

En este documento se detalla como transformar los servicios SOAP de Business Central en servicios tanto ODATA V4 como API REST, así como algunos ejemplos de cómo consumirlos desde diferentes lenguajes de programación.

2. Estado inicial: Servicios SOAP

Los servicios web SOAP, ODATA y API REST se han programado en lenguaje “AL” para Business Central (BC) mediante el editor de código Visual Studio Code (VSC). Por simplicidad, en este documento se ha utilizado la autenticación de tipo Basic, pero existe también la autenticación oAuth2 que es un poco más compleja.

Para ejemplificar el proceso de migración de SOAP a REST se ha creado un objeto Codeunit llamado “PREFIX Operaciones” y dos objetos Page: una Page ficha llamada “PREFIX Datos” y una Page lista llamada “PREFIX ListaDatos”. Todos estos objetos usarán la misma tabla: “PREFIX Datos”.

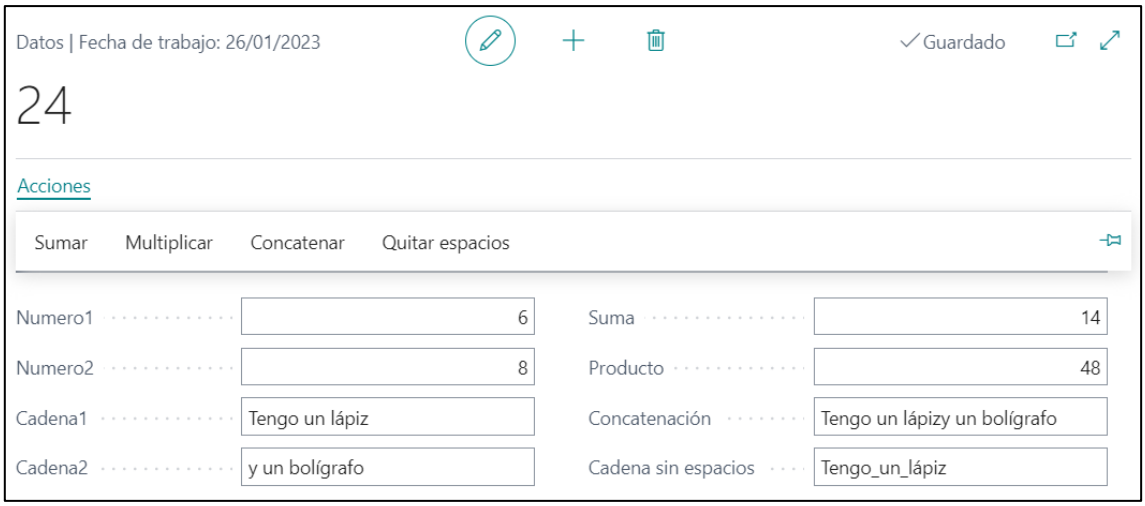
2.1. Codeunit Operaciones

La Codeunit Operaciones consta de 4 métodos sencillos que llaman a otra Codeunit para realizar los cálculos y guardar los datos en la tabla:

- **Suma:** Recibe dos enteros y devuelve su suma.
- **Producto:** Recibe dos enteros y devuelve su producto.
- **Concatenar:** Recibe dos cadenas y devuelve una cadena con el resultado de concatenar las dos cadenas de entrada.
- **QuitarEspacios:** Recibe una cadena y devuelve una cadena idéntica donde se sustituyen los espacios en blanco por barras bajas “_”.

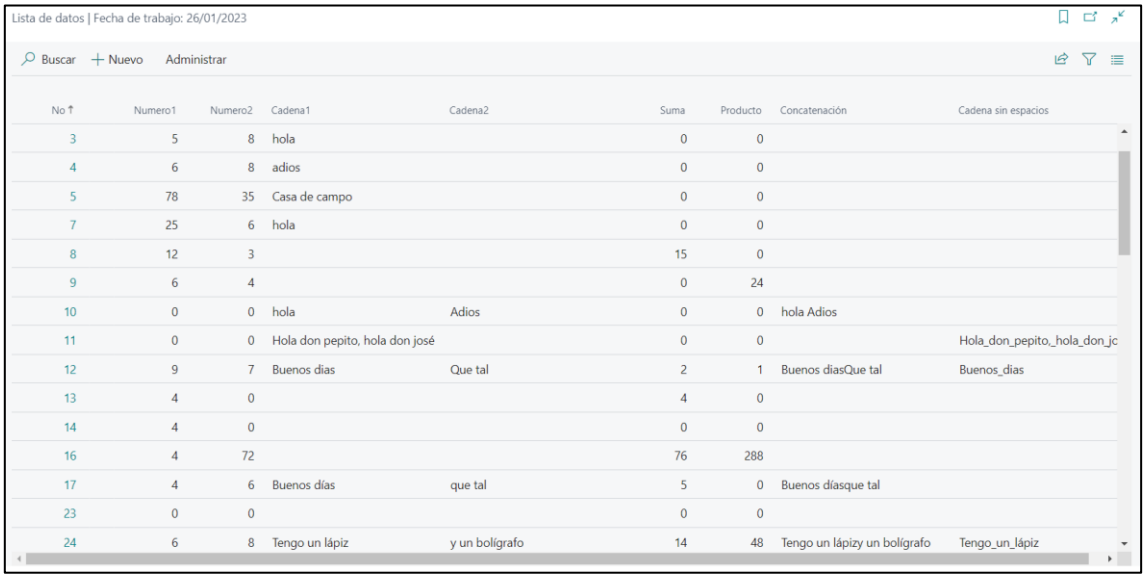
2.2. Ficha de datos

Esta Page de tipo ficha permite añadir nuevos registros a la tabla sin preocuparnos por el identificador, pues se ha definido como un campo autoincrementado en la tabla por lo que se incrementa automáticamente. Esta página permite realizar los cálculos descritos para la Codeunit a través de acciones.



2.3. Lista de datos

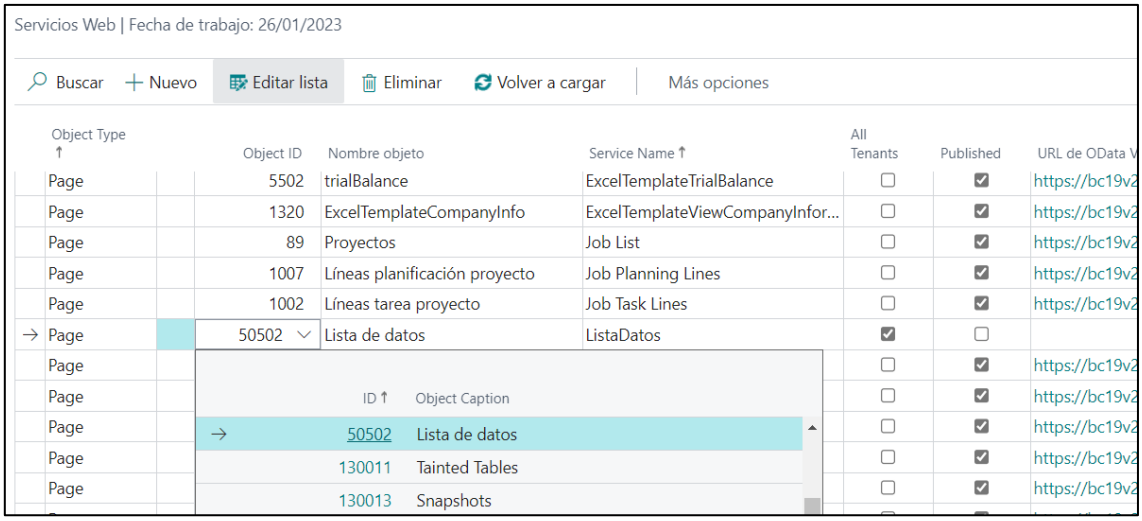
Esta Page de tipo lista muestra el listado de Datos que tiene nuestra tabla. Se ha elegido publicar ambas páginas para ilustrar que el tipo de página no importa, pues como servicio actúan igual. La única diferencia es que en este caso la lista si tiene el identificador de la tabla, además, la lista tiene bloqueada la edición, por lo que su servicio web solo nos permitirá realizar lectura de datos.



No	Numero1	Numero2	Cadena1	Cadena2	Suma	Producto	Concatenación	Cadena sin espacios
3	5	8	hola		0	0		
4	6	8	adios		0	0		
5	78	35	Casa de campo		0	0		
7	25	6	hola		0	0		
8	12	3			15	0		
9	6	4			0	24		
10	0	0	hola	Adios	0	0	hola Adios	
11	0	0	Hola don pepito, hola don josé		0	0		Hola_don_pepito,_hola_don_jc
12	9	7	Buenos dias	Que tal	2	1	Buenos diasQue tal	Buenos_dias
13	4	0			4	0		
14	4	0			0	0		
16	4	72			76	288		
17	4	6	Buenos días	que tal	5	0	Buenos díasque tal	
23	0	0			0	0		
24	6	8	Tengo un lápiz	y un bolígrafo	14	48	Tengo un lápizy un bolígrafo	Tengo_un_lápiz

2.4. Publicar los objetos como servicios

Desde la página de **Servicios Web** en Business Central se pueden añadir nuevos servicios. Para ello, se pulsa en Nuevo, lo cual crea una línea vacía que se debe rellenar. Una vez rellenados los campos Object Type, Object ID, Nombre de objeto y Service Name, se debe marcar la opción Published para que se publique dicho objeto como servicio web. Si se ha realizado con éxito, al recargar la página se habrán rellenado los campos de URL ODATA y SOAP. Cuando se publica un objeto de tipo Page, se genera tanto un enlace ODATA como SOAP. En cambio, cuando se publica un objeto Codeunit solo se genera un enlace SOAP. Si es de tipo Query, solo está disponible como ODATA.



3. Endpoints

Un endpoint es un enlace URL al cual va a responder un servicio web ya sea para cargar, consumir o mostrar información.

Al publicar un servicio web, Business Central genera automáticamente su endpoint si se trata de un servicio SOAP o de un servicio de tipo Page ODATA. Para usar dichos servicios webs es necesario conocer la estructura de los endpoints por si fuera necesario modificar el endpoint para realizar una petición diferente.

3.1. Page ODATA

Uno de los endpoints más sencillos es el de servicios de tipo page ODATA. Este enlace se puede consultar directamente desde BC, en la ventana de Servicios Web. Su estructura, junto a un ejemplo, es la siguiente:

```
https://<Servidor>:<PuertoServicioWeb>/<InstanciaServidor>/ODataV4/
Company('<NombreEmpresa|IdEmpresa>')/<Nombre del servicio>

Ejemplos:
https://bc19v2:7048/BC/ODataV4/Company('CRONUS%20Espa%C3%B1a%20S.A.')
/Datos
https://bc19v2:7048/BC/ODataV4/Company(5d3bf4b2-0d57-ec11-bb7e-
000d3a21fc0b)/Datos
```

3.2. Codeunit ODATA

Business Central no genera un enlace con la dirección del servicio web de la Codeunit al publicar la Codeunit porque el endpoint varía en función de a que método se esté realizando la petición. Su estructura es la siguiente:

```
https://<Servidor>:<PuertoServicioWeb>/<InstanciaServidor>/ODataV4/<Nombre
Servicio>_<NombreFuncion>?company=<NombreEmpresa|IdEmpresa>

Ejemplo:
https://bc19v2:7048/BC/ODataV4/Operaciones_Suma?company=CRONUS%20Espa%C3%B
1a%20S.A.
```

3.3. Page API REST

Al contrario que los servicios SOAP y ODATA, los servicios API REST se publican automáticamente cuando son añadidos a BC. Por tanto, BC no genera un enlace a dichos servicios, por lo que debes construirlo tu mismo. Dependiendo de si se trata de un servicio nativo de BC o uno personalizado su enlace será diferente. Los endpoints API siguen las siguientes estructuras:

```
Servicio API:
https://<Servidor>:<PuertoServicioWeb>/<InstanciaServidor>/<APIversion>/co
mpanies(<NombreEmpresa|IdEmpresa>)/<NombreServicio>

Ejemplo:
https://bc19v2:7048/BC/api/v2.0/companies(5d3bf4b2-0d57-ec11-bb7e-
000d3a21fc0b)/customers

Servicio API personalizado:
https://<Servidor>:<PuertoServicioWeb>/<InstanciaServidor>/api/<APIpublish
er>/<APIgroup>/<APIversion>/<NombreServicio>

Ejemplo:
https://bc19v2:7048/BC/api/laberit/custom/v2.0/datos
```

Los campos API Publisher, API group y API version se definen en el objeto API. Actualmente, no existen objetos Codeunit de tipo API REST.

4. Servicios ODATA V4

Para migrar un objeto publicado como servicio SOAP a ODATA V4 no hay que realizar ningún cambio, pues al publicarlo como SOAP también se publica como ODATA. Lo que si cambia es la forma en que se consume, sobre todo los objetos Codeunit pues Business Central no nos genera ningún enlace ODATA V4 para ellos.

4.1. Consumir ODATA V4 usando la extensión REST CLIENT de VSC

En primer lugar, se debe instalar la extensión REST client en Visual Studio Code. A continuación, se abre un proyecto y se crea un fichero ".http". En este fichero se pueden escribir peticiones HTTP para llamar a los servicios web.

El formato de una petición GET tiene la siguiente forma:

```
GET    https://<Servidor>:<PuertoServicioWeb>/<InstanciaServidor>/ODataV4/
Company('<Nombre de la empresa>')/<Nombre del servicio> HTTP/1.1
Authorization: Basic <usuario:contraseña(en formato Base64)>

Ejemplo:
GET  https://bc19v2:7048/BC/ODataV4/Company('CRONUS%20Espa%C3%B1a%20S.A.')
/Datos HTTP/1.1
Authorization: Basic XXXXXXXX
```

Introducimos la autorización en la cabecera de la petición, en este caso usamos autenticación Basic.

Finalmente, se seleccionan las líneas de la petición y se ejecuta el comando REST CLIENT: SEND REQUEST. Si todo ha ido bien, se mostrará en pantalla la respuesta de la petición en formato JSON.

Para realizar una petición POST a un servicio Page e introducir un registro nuevo en la tabla el formato es similar, pero añadiendo los parámetros necesarios:

```
POST https://bc19v2:7048/BC/ODataV4/Company('CRONUS%20Espa%C3%B1a%20S.A.')
/Datos HTTP/1.1
Content-Type: application/json
Authorization: Basic XXXXXXXX

{
  "Numero1": 25,
  "Numero2": 6,
  "Cadena1": "posa",
  "Cadena2": "vasos"
}
```

La petición PUT, para modificar registros, es similar a la petición POST pero se debe indicar en el endpoint que registro se pretende modificar. Además, se debe añadir una cabecera If-Match:

```
PUT https://bc19v2:7048/BC/ODataV4/Company('CRONUS%20Espa%C3%B1a%20S.A.')/
Datos(51) HTTP/1.1
Content-Type: application/json
Authorization: Basic XXXXXXXXXX
If-Match: *

{
  "numero1": 25,
  "numero2": 6,
  "cadena1": "posa",
  "cadena2": "vasos"
}
```

La petición DELETE es como una petición GET donde se indica que registro se quiere eliminar a través del endpoint.

```
DELETE https://bc19v2:7048/BC/ODataV4/Company('CRONUS%20Espa%C3%B1a%20S.A.
')/Datos(25) HTTP/1.1
Authorization: Basic bmF2OiFOQHYyMDIw
```

Finalmente, para llamar a las funciones de un servicio Codeunit, se debe usar una petición POST con el endpoint de la función a la cual se quiera realizar la petición.

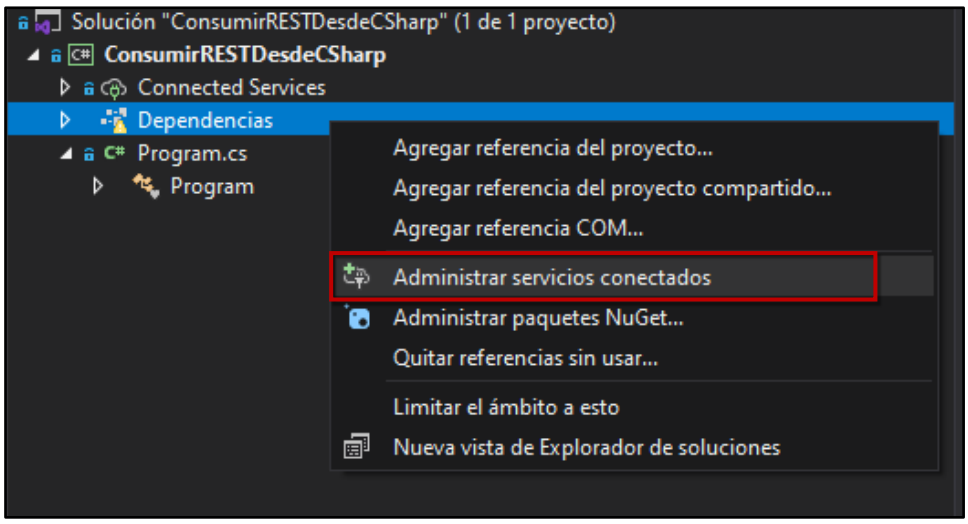
```
POST    https://bc19v2:7048/BC/ODataV4/Operaciones_QuitarEspacios?company=
CRONUS%20Espa%C3%B1a%20S.A. HTTP/1.1
content-type: application/json
Authorization: Basic XXXXXXXXXX

{
    "cadena1": "Hello world!"
}
```

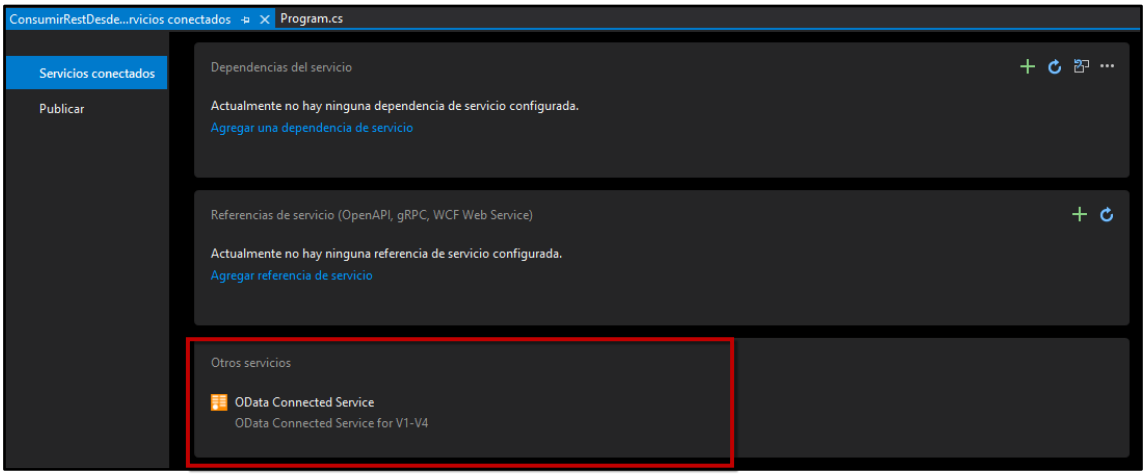
4.2. Consumir ODATA V4 mediante C# y .Net Core en Visual Studio

En primer lugar, se ha creado un nuevo proyecto en Visual Studio 2019 usando la plantilla Aplicación de consola C#. Se ha usado Visual Studio 2019 para poder usar la extensión OData Connected Service pues, cuando se ha redactado este documento, esta extensión aún no soporta versiones más modernas de Visual Studio. Sin embargo, el desarrollador está trabajando en una versión para Visual Studio 2022.

Ahora se procede a configurar los servicios a los que se pretende conectar usando la nueva extensión. Al abrir el proyecto en Visual Studio, se puede ver una carpeta llamada Dependencias, haz clic derecho y selecciona la opción Administrar Servicios Conectados.

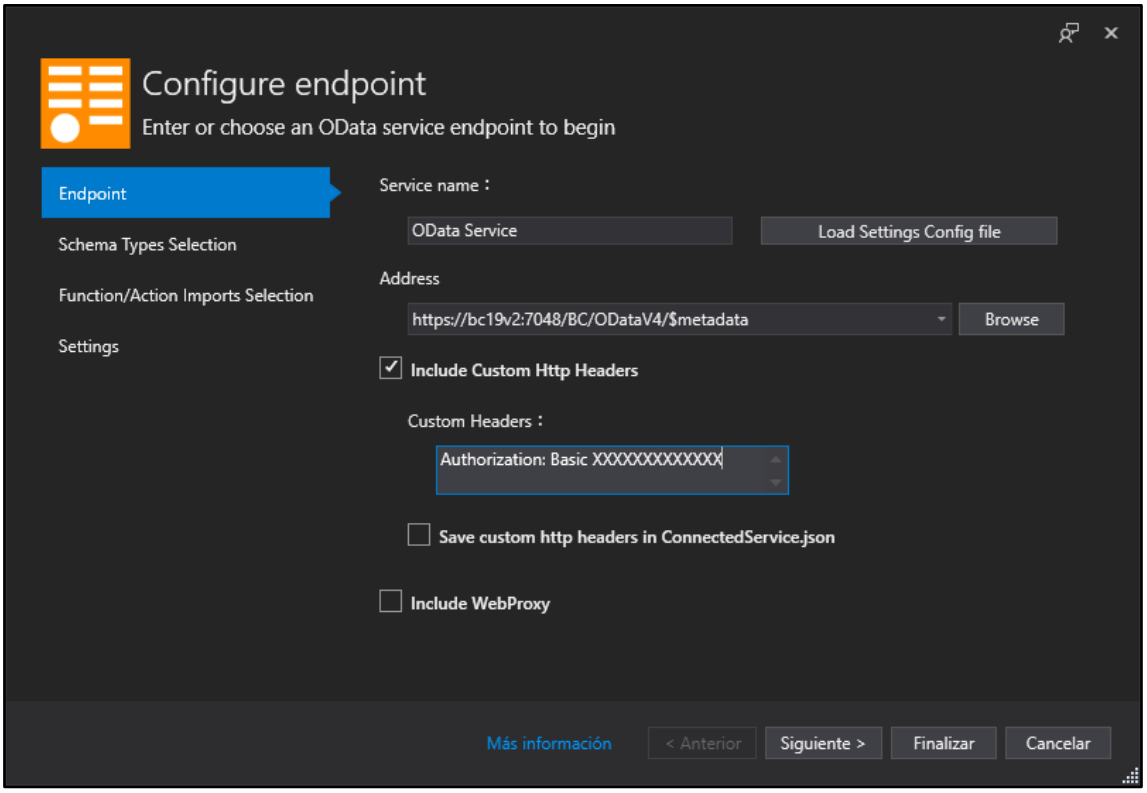


Se abrirá una nueva pestaña y se podrá acceder a la extensión ODATA Connected Service.

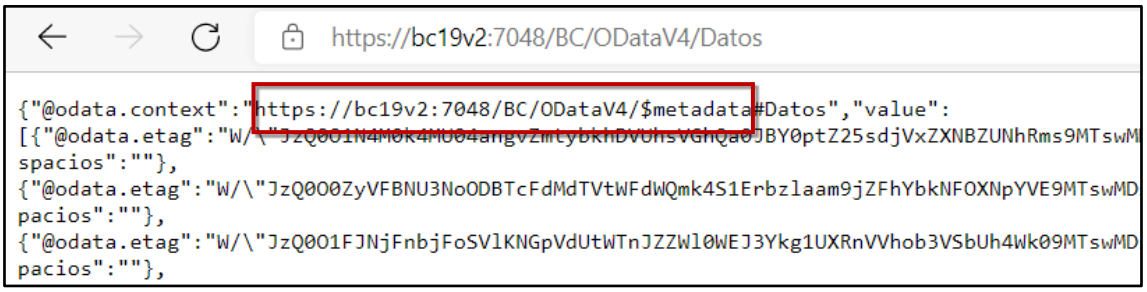


La nueva ventana permite configurar los servicios web. En primer lugar, se configura el endpoint. Se debe indicar:

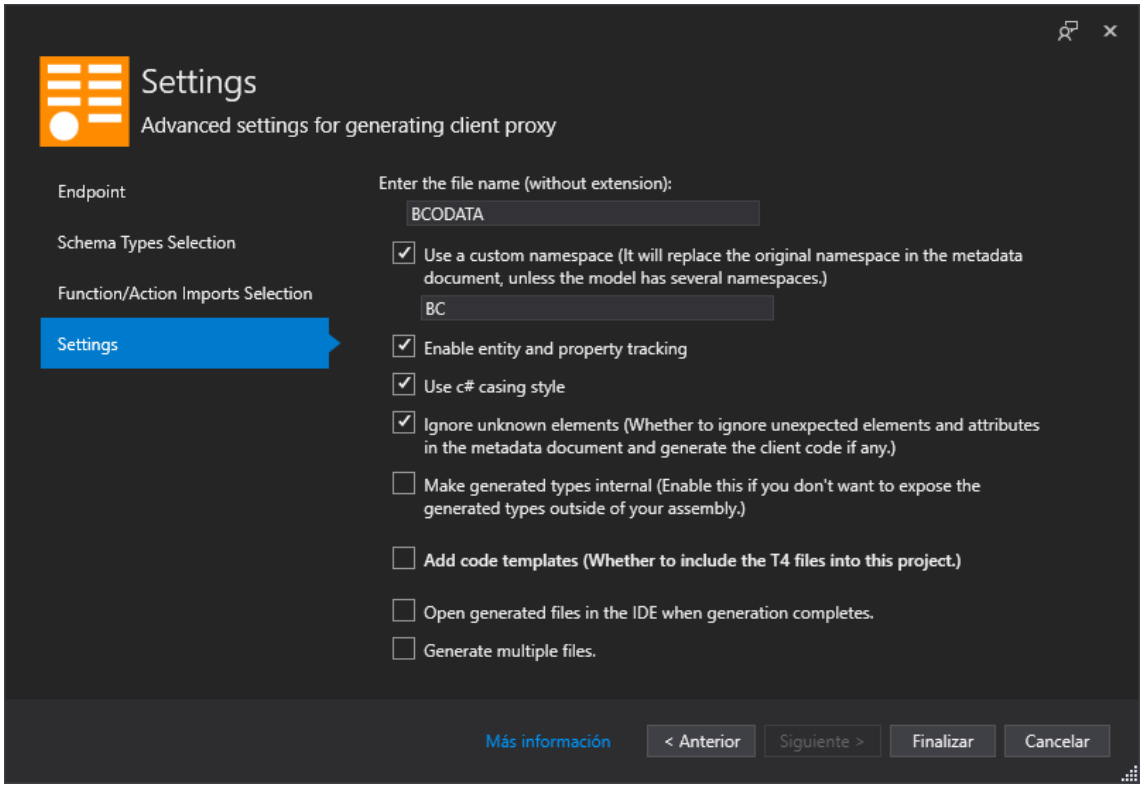
- **Service Name:** Nombre que se le quiera tenga el servicio. Si se van a añadir diferentes endpoints, es importante que tengan nombres de servicio diferentes para que no colisionen.
- **Address:** Dirección donde se encuentra el documento XML con los datos de los servicios.
- **Include Custom Http Headers:** Es importante marcarlo para poder introducir las credenciales de BC y permita usar los servicios. Las credenciales tienen el mismo formato que en el caso anterior en el que consumíamos ODATA con la extensión REST CLIENT en Visual Studio Code: “Authorization: Basic <usuario:contraseña(en formato Base64)>”



Para averiguar la dirección ODATA: se abre BC, se accede al apartado Servicios Web y se abre el enlace de algún servicio web ODATA v4. Se abrirá una pestaña con el servicio en formato JSON. El enlace que se necesita se encuentra en el elemento @odata.context y abarca hasta $metadata.



Tras esto, si la autenticación está bien configurada, la extensión permitirá avanzar a la siguiente ventana. En las ventanas Schema Types Selection y Function/Action Imports Selection se puede seleccionar que servicios se quieren incluir y cuales no. Finalmente, en Settings se puede cambiar el nombre del archivo y en Advanced Settings se puede cambiar el nombre del namespace. Es importante cambiarlos, sobre todo si se van a importar servicios de diferentes endpoints. El resto de las opciones no se van a modificar, así que haz clic en Finalizar.



A continuación, se abre el fichero donde se quieren realizar las peticiones a los servicios, en este caso, se va a usar el método Main. En primer lugar, se declara una variable con la dirección del servicio al cual queremos acceder. A continuación, se declara otra variable donde se va a guardar un objeto namespace NAV que será nuestro context, este se llama NAV porque originalmente en BC el namespace se llama NAV. Finalmente, añadimos al evento BuildingRequest el método Context_BuildingRequest. Con este método se puede modificar la URI si es necesario o añadir cabeceras a la petición para insertar la autenticación, por ejemplo.

```
using System;
namespace ConsumirRESTDesdeCShrap
{
    class Program
    {
        static void Main(string[] args)
        {
            var serviceRoot =
"https://bc19v2:7048/BC/ODataV4/Company('CRONUS%20España%20S.A.')/";
            var context = new BC.NAV(new Uri(serviceRoot));
            context.BuildingRequest += Context_BuildingRequest;
         }

        private static void Context_BuildingRequest(object sender,
Microsoft.OData.Client.BuildingRequestEventArgs e)
        {
            e.Headers.Add("Authorization", "Basic XXXXXXXXXXXXX");
        }
    }
}
```

El código anterior es el mismo para todas las peticiones excepto la dirección, que irá cambiando según a que servicio se llame. Si se quiere realizar una petición GET, solo hay que llamar al método Execute del servicio al que queramos llamar. Si se quiere añadir un filtro al resultado para que devuelva justo los valores que se necesitan, se puede usar el método AddQueryOption. En el ejemplo a continuación, se puede ver como se usan ambos métodos y en este caso, los resultados que recupera los muestra por pantalla:

```
//todos los datos
//var data = context.Datos.Execute();
//solo datos filtrados
var data = context.Datos.AddQueryOption("$filter", "producto gt 0");
//mostrar por pantalla el resultado
foreach (var dato in data)
{
      Console.WriteLine("{0} {1} {2} {3} {4} {5} {6} {7}",
dato.Numero1, dato.Numero2, dato.Cadena1, dato.Cadena2, dato.Suma,
dato.Producto, dato.Concatenacion, dato.CadenaSinEspacios);
}
```

En caso de querer realizar una inserción con una petición POST, se debe crear un objeto del servicio que queramos e introducirle los atributos que sean necesarios. Tras esto se añade al objeto context y se ejecuta el método SaveChanges() para que se envíe la petición, aplicando los cambios en BC.

```
var dato = new BC.Datos();
dato.Numero1 = 7;
dato.Numero2 = 18;
dato.Cadena1 = "Hoy compro fruta";
dato.Cadena2 = "por la mañana";
context.AddToDatos(dato);
var serviceResponse = context.SaveChanges();
foreach (var operationResponse in serviceResponse)
{
```

Por otra parte, para eliminar un registro debemos extraer el registro a eliminar mediante una petición GET. A continuación, se ejecuta el método DeleteObject() introduciendo el objeto a eliminar. Finalmente, se ejecuta el método SaveChanges para guardar los cambios en BC y ejecutar la petición.

```
var data = context.Datos.AddQueryOption("$filter", "no eq 36");
BC.Datos dato = null;
using (IEnumerator<BC.Datos> enumerator = data.GetEnumerator())
{
	if (enumerator.MoveNext()) dato = enumerator.Current;
}
context.DeleteObject(dato);
var serviceResponse = context.SaveChanges();
foreach (var operationResponse in serviceResponse)
{
	Console.WriteLine(operationResponse.StatusCode);
}
```

Si se quiere modificar un registro, en primer lugar, hay que realizar una petición GET y extraer el registro que se quiera modificar. Tras esto, se modifican los campos necesarios, se añade el objeto modificado al objeto context a través del método UpdateObject() y se ejecuta el método SaveChanges() para aplicar los cambios en BC.

```
var data = context.Datos.AddQueryOption("$filter","no eq 3");
BC.Datos dato = null;
using (IEnumerator<BC.Datos> enumerator = data.GetEnumerator())
{
       if (enumerator.MoveNext())  dato = enumerator.Current;
}
dato.Numero1 = 7;
dato.Numero2 = 18;
dato.Cadena1 = "Hoy compro fruta";
dato.Cadena2 = "por la mañana";
dato.Suma = 25;
dato.Producto = 126;
context.UpdateObject(dato);
var serviceResponse = context.SaveChanges();
foreach (var operationResponse in serviceResponse)
{
       Console.WriteLine(operationResponse.StatusCode);
}
```

Por último, para realizar peticiones a los métodos de una Codeunit, solo se tiene que llamar al método, introducirle los parámetros necesarios y llamar al método GetValue() para capturar el resultado.

```
//método quitarEspacios
String data = context.Operaciones_QuitarEspacios("Hello
world").GetValue();
//método producto
//int? data = context.Operaciones_Producto(5, 8).GetValue();
//imprimir resultado
Console.WriteLine("El resultado es: {0}", data);
```

Es necesario cambiar el endpoint del servicio web tal y como se ha visto en apartados anteriores. Para hacerlo se añade al final de la URI la empresa, usando el método Context_BuildingRequest.

```
e.RequestUri = new Uri(e.RequestUri.ToString() +
"?company=CRONUS%20Espa%C3%B1a%20S.A.");
```

4.3. Consumir ODATA V4 mediante Java

Empezamos creando un nuevo proyecto. Para facilitar la importación de las bibliotecas, se ha creado un proyecto Maven, en concreto se ha usado la plantilla: org.apache.maven.archetypes:maven-archetype-quickstart. El sdk escogido ha sido openjdk-17 Oracle OpenJDK version 17.0.2. Esta información es de carácter informativo, las peticiones deberían poder ejecutarse en cualquier proyecto Java.

En primer lugar, importamos la biblioteca Jackson, una forma de hacerlo es añadirla al fichero pom.xml. Esta biblioteca nos va a facilitar transformar el JSON respuesta de los servicios a objetos Java y así poder manipularlos.

```
<dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.13.2.1</version>
```

Se comienza declarando el cliente usando la clase HttpClient. Este cliente será el mismo para todas las peticiones. La dirección del servicio se ha declarado fuera del método que vamos a ejecutar porque no se va a modificar durante la ejecución y varios métodos usaran la misma. Además, la hemos declarado en fragmentos porque los servicios Codeunit y Page usan direcciones un poco diferentes.

```
//ODATA
private static final String odataBaseURL =
"https://bc19v2:7048/BC/ODataV4/";
private static final String odataCompany =
"Company('CRONUS%20Espa%C3%B1a%20S.A.')/";
private static final String odataDatos = "Datos";
private static final String odataListaDatos = "ListaDatos";
//apirest
private static final String apiDatoServiceURL =
"https://bc19v2:7048/BC/api/laberit/custom/v2.0/datos";

public static HttpClient cliente(){
      //----------------CLIENTE-------------------------
      HttpClient cliente = HttpClient.newBuilder()
                  .version(HttpClient.Version.HTTP_1_1)
                  .build();
      return cliente;
}
```

Vamos a ver como se realiza una petición GET al servicio ListaDatos. En primer lugar, se crea la petición GET usando la clase HttpRequest. En la cabecera se introduce la autentificación en el mismo formato visto anteriormente ("Authorization","Basic <usuario:contraseña(en formato Base64)>"). Se introduce también cuál es la URI, se crea a partir del enlace al servicio. Se envía la petición y se recoge la respuesta con la clase HttpResponse. Se debe indicar que devuelva la respuesta como una cadena. Finalmente, solo queda transformar la respuesta a un objeto java para poder trabajar con él. La respuesta se recibe como un JSON en forma de cadena. Hay que adaptar la transformación a objeto Java según el tipo de objeto o de valor que esperemos recibir. En este caso, se va a recibir una lista de objetos Dato. Para facilitar su transformación, se crea una clase respuesta cuyo atributo sea una lista de objetos Dato. Crea un objeto ObjectMapper y configuralo para que no falle si no es capaz de reconocer algún valor, esto es importante porque en el JSON viene más información de la que se va a usar en este momento. Tras esto, se transforma el JSON usando el método readValue del objeto mapper y se guarda el resultado en una variable. Ya se pueden utilizar los datos, para comprobarlo se imprimen por pantalla.

```
//------------------GET--------------------------
url = odataBaseURL + odataCompany + odataListaDatos;
//creamos la petición GET
HttpRequest peticion = HttpRequest.newBuilder()
             .GET().header("Authorization","Basic XXXXXXXXXXX")
             .header("Content-Type","application/json")
             .uri(URI.create(url))
             .build();

//----------------RESPUESTA-GET-------------------------
//enviamos la petición y obtenemos el resultado
HttpResponse<String> respuesta = cliente.send(peticion,
HttpResponse.BodyHandlers.ofString());


//Transformar JSON en objetos
ObjectMapper mapper = new ObjectMapper();
mapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,
false);
RespuestaDato valor = mapper.readValue(respuesta.body(), new
TypeReference<>() {});
valor.getValue().forEach(System.out::println);
```

Para crear un nuevo registro usando una petición POST del servicio Datos se crea un diccionario donde almacenar los parámetros de la petición y se transforma en una cadena. A continuación, se crea una petición HttpRequest, en este caso de tipo POST, y se le inyectan los parámetros. Como en la petición GET, se inserta la URI y en la cabecera se introduce la autentificación, pero en este caso añadimos también el Content-Type para que sepa que tipo de valor está devolviendo. Enviamos la petición usando el método send del cliente y guardamos el resultado en una variable HttpResponse. Igual que en el caso del GET, configuramos el objeto mapper, pero en este caso el resultado es un único objeto Dato, por lo que no necesitamos un objeto respuesta, lo podemos convertir directamente con el mapper.

```
String url = odataBaseURL + odataCompany + odataDatos;
//--------------------POST--------------------------
//También se pueden pasar los parámetros directamente como un string
json
//String message = "{\"cadena1\":\"hola que tal\",\"cadena2\":\"muy
bien\"}";

//creamos un diccionario donde guardaremos los parametros del post
Map<String,String> parametros = new HashMap<>();
parametros.put("numero1","9");
parametros.put("numero2","5");
parametros.put("cadena1","El cielo es ");
parametros.put("cadena2","azul");
//transformamos el diccionario en json para poderlo pasar al post
ObjectMapper mapperMap = new ObjectMapper();
String message = mapperMap.writeValueAsString(parametros);

//construimos la petición POST
HttpRequest peticion = HttpRequest.newBuilder()
             .POST(HttpRequest.BodyPublishers.ofString(message))
             .header("Authorization","Basic XXXXXXXXXXXXXXXXX")
             .header("Content-Type","application/json")
             .uri(URI.create(url))
             .build();

//----------------RESPUESTA-POST--------------------------

//enviamos la petición y obtenemos el resultado
HttpResponse<String> respuesta = cliente.send(peticion,
HttpResponse.BodyHandlers.ofString());
```

```
//Transformar JSON en objetos
ObjectMapper mapper = new ObjectMapper();
mapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,
false);
Dato valor = mapper.readValue(respuesta.body(), new
TypeReference<Dato>() {});
System.out.println(valor);
```

La petición DELETE es similar a la petición GET, pero en la URI se debe añadir el identificador del registro que se quiera eliminar.

```
String url = odataBaseURL + odataCompany + odataDatos + "(" +
Integer.toString(idDato) + ")";
//------------------DELETE--------------------------
//creamos la petición DELETE
HttpRequest peticion = HttpRequest.newBuilder()
              .DELETE().header("Authorization", "Basic XXXXXXXXX")
              .header("Content-Type", "application/json")
              .uri(URI.create(url))
              .build();
//----------------RESPUESTA-DELETE-------------------------
//enviamos la petición y obtenemos el resultado
```

La petición PUT permite actualizar un registro. Esta petición es similar a la petición POST, pero la URI se construye igual que la petición DELETE, indicando el identificador de registro. Además, en la cabecera se debe incluir un parámetro adicional llamado If-Match.

```
String url = odataBaseURL + odataCompany + odataDatos +
"("+Integer.toString(15)+")";
//También se pueden pasar los parámetros directamente como un string
json
//String message = "{\"cadena1\":\"hola que tal\",\"cadena2\":\"muy
bien\"}";
```

```
//creamos un diccionario donde guardaremos los parametros del post
Map<String,String> parametros = new HashMap<>();
parametros.put("numero1", 25);
parametros.put("numero2",2);
parametros.put("cadena1","hola");
parametros.put("cadena2","adios");
parametros.put("suma",27);
parametros.put("producto",50));
parametros.put("concatenacion","holaadios");
parametros.put("cadenaSinEspacios","hola");
//transformamos el diccionario en json para poderlo pasar al put
ObjectMapper mapperMap = new ObjectMapper();
String message = mapperMap.writeValueAsString(parametros);

//construimos la petición PUT
HttpRequest peticion = HttpRequest.newBuilder()
              .PUT(HttpRequest.BodyPublishers.ofString(message))
              .header("Authorization","Basic XXXXXXXXXXXXXX")
              .header("If-Match", "*")
              .header("Content-Type","application/json")
              .uri(URI.create(url))
              .build();

//----------------RESPUESTA-PUT-------------------------
//enviamos la petición y obtenemos el resultado
HttpResponse<String> respuesta = cliente.send(peticion,
HttpResponse.BodyHandlers.ofString());
//System.out.println(respuesta.body());

ObjectMapper mapper = new ObjectMapper();
mapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,
false);
Dato valor = mapper.readValue(respuesta.body(), new
TypeReference<Dato>() {});
```

Finalmente, para llamar a los métodos de la Codeunit se realiza una petición POST, se añaden solo los parámetros necesarios, se ajusta el endpoint según el método que se desee llamar y se ajusta la transformación de JSON a objeto a los tipos que sean necesarios. En el siguiente fragmento de código se devuelve el JSON resultado cómo una cadena, así facilita extraer enteros y cadenas, pues es fácil convertir de cadena a entero.

```
ObjectMapper mapper = new ObjectMapper();
mapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,
false);
RespuestaCU valor = mapper.readValue(respuesta.body(), new
TypeReference<RespuestaCU>() {});
System.out.println("El resultado es: "+valor.getValue());
```

5. Servicios API REST

Los objetos Codeunit no se pueden publicar como servicios API REST, solo como servicios SOAP o ODATA. En cambio, los objetos Page si se pueden publicar como API REST. Por tanto, solo vamos a poder transformar los servicios Page SOAP a API REST. Las páginas de tipo API no son como las páginas SOAP, no se pueden ver o usar desde BC, actúan únicamente como servicios web. Es decir, no pueden ser accedidos directamente a través del lenguaje AL, es necesario realizar una petición como si se tratase de un servicio web externo a BC.

Para crear un servicio de tipo Page API REST, se crea un objeto Page y se indica que el atributo PageType es API. Se deben configurar los atributos básicos tales como SourceTable, Caption, EntityName, entre otros. Es muy importante indicar los atributos APIGroup, APIPublisher y APIVersion para crear la ruta a dicho servicio. Los campos field deberán ser idénticos a los del objeto SOAP. Dado que no se va a ver desde BC, no tiene sentido añadir acciones a la página. Una vez se ha terminado de configurar, se sube a BC. No es necesario realizar ninguna acción más para publicarla como servicio web, dado que BC interpreta que se trata de una página de tipo API y la publica automáticamente.

5.1. Consumir API REST

Para consumir este servicio API REST, se procede exactamente igual que lo visto anteriormente para consumir ODATA, puesto que recibe los parámetros en el mismo formato y devuelve el resultado en un JSON igual que ODATA. La única diferencia es que la estructura del endpoint es distinta.

Para encontrar la ruta al metadata y poder configurar la extensión ODATA Connected Service de C# y .Net, se sigue la siguiente estructura:

```
Servicio API personalizado:
https://<Servidor>:<PuertoServicioWeb>/<InstanciaServidor>/api/<APIpublisher>/<APIgroup>/<APIversion>/$metadata

Ejemplo:
https://bc19v2:7048/BC/api/laberit/custom/v2.0/$metadata
```

6. Test

A lo largo del proyecto, se ha verificado el correcto funcionamiento del código mediante distintas herramientas.

En primer lugar, se han usado los programas SoapUI y Postman para comprobar que los servicios SOAP funcionan correctamente y son accesibles. En segundo lugar, se iba ejecutando el código creado para consumir los servicios ODATA y API REST en diferentes lenguajes y se comparaba visualmente con los resultados obtenidos con SoapUI y Postman para comprobar que fueran correctos. Finalmente, se realizaron diversas pruebas unitarias en Java donde se compara el resultado obtenido por el servicio SOAP con sus equivalentes en ODATA y API REST.

7. Glosario

- **AL:** Lenguaje de programación de Business Central a partir de la versión 14. Se usa principalmente para consultar, insertar y modificar registros en una base de datos de Business Central
- **BC (Business Central):** También llamado Microsoft Dynamics 365 Business Central es sistema de gestión empresarial desarrollado por Microsoft. Permite gestionar los apartados de finanzas, ventas, marketing, servicios, comercio, entre otros.
- **VSC (Visual Studio Code):** Potente editor de código desarrollado por Microsoft compatible con múltiples lenguajes de programación, entre ellos el lenguaje de Business Central, AL.
- **VS (Visual Studio):** Entorno de desarrollo integrado desarrollado por Microsoft compatible con diferentes lenguajes de programación.
- **Codeunit:** Contenedor de código AL para implementar lógica de negocio, organizado en funciones.
- **Page o Página:** Forma principal para mostrar y organizar visualmente datos en Business Central.
- **OData Connected Service:** Extensión de Visual Studio que genera una clase en lenguaje C# que permite interaccionar con un servicio web REST específico.
- **REST CLIENT:** Extensión para Visual Studio Code que permite enviar peticiones HTTP y visualizar la respuesta en VSC directamente.
- **Maven:** Herramienta de gestión e implementación de software que hace posible la implementación de software con dependencias incluidas dentro de la estructura del JAR. Para ello, se deben definir todas las dependencias en un fichero POM.
- **SoapUI:** Herramienta para la realización de pruebas a servicios web SOAP y REST.
- **Postman:** Plataforma API para desarolladores para diseñar, construir, testear e iterar sobre APIs.