# Realización del plan de trabajo de pasantía en la unidad de "Identity and Access Solutions" de la empresa BEXTechnology S.A.

Autor:

Jeffrey Alexander Díaz Pérez

Universidad Nacional Abierta y a Distancia – UNAD

Escuela de Ciencias Básicas, Tecnología e Ingeniería

Ingeniería de sistemas

Junio 2022

# Realización del plan de trabajo de pasantía en la unidad de "Identity and Access Solutions" de la empresa BEXTechnology S.A.

Autor:

Jeffrey Alexander Díaz Pérez

Tutor encargado:

Ing. Jaime Junior Sedano Pinzón

Coordinador pasantía:

Ing. Cristhian Camilo Medina Solano

Trabajo de grado para optar al título de Ingeniero de sistemas

Universidad Nacional Abierta y a Distancia – UNAD

Escuela de Ciencias Básicas, Tecnología e Ingeniería

Ingeniería de sistemas

CEAD Vélez

2022

## Dedicatoria

Dedico este trabajo de sustentación de pasantía a mis padres, quienes, a pesar de las dificultades, siempre me han apoyado y motivado constantemente en este proceso de aprendizaje, este logro se lo debo a ellos, muchas gracias de corazón por todo.

Gracias Mamá y Papá por inculcarme la responsabilidad y superación.

## Agradecimientos

Agradezco de todo corazón a mis padres, ya que de diferentes maneras siempre me apoyaron en este proceso de aprendizaje, sus apoyos constantes me permitieron culminar los estudios profesionales con muy buenos resultados, siempre les estaré muy agradecido por todo.

A Héctor Martin Rodríguez de BEXTechnology S.A. por permitirme realizar la pasantía en esta gran empresa.

A Cristhian Camilo Medina Solano por apoyarme desde un inicio en este proceso de pasantía, un gran profesional, mentor y persona.

A mis tutores de la UNAD que siempre de manera muy amable y profesional me guiaron en este camino del aprendizaje, siempre me aclararon cualquier duda que tuviera, gracias por todas sus enseñanzas.

## Resumen

El presente documento abordará la sustentación del plan de trabajo realizado en cumplimiento de la opción de grado pasantía, se expondrán los resultados y logros conseguidos en este proceso práctico de aprendizaje, se mostrará la aplicación de los conocimientos adquiridos en el proceso formativo del programa de ingeniería de sistemas, se describirán detalladamente las actividades realizadas cada mes mediante modalidad de teletrabajo en la empresa BEXTechnology S.A., se finaliza el documento con algunas reflexiones y conclusiones finales.

**Palabras clave:** realización, plan de trabajo, pasantía, unidad, identity, access solutions, empresa, BEXTechnology S.A., programación, C#, Entity Framework Core, Web APIs RESTful, ASP.NET Core 6, servicios REST, métodos HTTP, JSON, JWT, Identity Server, casos de uso, ingeniería, sistemas

## Abstract

The present document will approach the sustentation of the plan of work realized in fulfillment of the option of degree internship, the results and achievements obtained in this practical process of learning will be exposed, the application of the knowledge acquired in the formative process of the program of engineering of systems will be shown, the activities realized every month by means of modality of telework in the company BEXTechnology S.A. will be described in detail, the document finishes the document with some reflections and final conclusions.

**Keywords:** realization, work plan, internship, internship, unit, identity, access solutions, company, BEXTechnology S.A., programming, C#, Entity Framework Core, RESTful Web APIs, ASP.NET Core 6, REST services, HTTP methods, JSON, JWT, Identity Server, use cases, engineering, systems, systems

# Tabla de Contenido

Resumen ..... 5

Abstract ..... 6

Introducción ..... 14

Justificación ..... 15

Objetivos ..... 17

- Objetivo general ..... 17
- Objetivos específicos ..... 17

Presentación de la Empresa ..... 18

- Visión ..... 19
- Propósito ..... 19

Metodología ..... 20

Descripción detallada del plan de trabajo ..... 21

- Cronograma ..... 24
- Resultados esperados ..... 25
- Resultados obtenidos ..... 26

Descripción de la pasantía ..... 27

Desarrollo del plan de trabajo de la pasantía ..... 28

Modernización del software multiplataforma “Bext Self Reset” ..... 171

Reflexiones finales ..... 174

Conclusiones ..... 175

Bibliografía ..... 176

## Lista de tablas y figuras

Tabla 1. Cronograma plan de trabajo desarrollado 24

Tabla 2. Resultados y productos esperados 25

Tabla 3. Resultados y productos obtenidos 26

Figura 1. Clase Modelo/Entidad del desarrollo del caso de uso de Personalización BSR 30

Figura 2. Método Post() para crear, en la clase de servicio TemplateService 31

Figura 3. Método Put() para modificar, en la clase de servicio TemplateService 32

Figura 4. Clase Modelo/Entidad del desarrollo del caso de uso de Plantillas de Notificación 33

Figura 5. Método Post() para crear, en la clase de servicio NotificationTemplateService 34

Figura 6. Método Put() para modificar, en la clase de servicio NotificationTemplateService 35

Figura 7. Servicios implementados para el desarrollo de Personalización BSR 36

Figura 8. Configurar una petición para crear una plantilla desde Swagger UI 37

Figura 9. Respuesta que muestra Swagger UI al crearse la plantilla de personalización 38

Figura 10. Configurar una petición para modificar una plantilla desde Swagger UI 39

Figura 11. Respuesta que muestra Swagger UI al modificarse la plantilla de personalización 40

Figura 12. Servicios implementados para el desarrollo de Plantillas de Notificaciones 41

Figura 13. Configurando una plantilla de notificación a crear desde Swagger UI 42

Figura 14. Respuesta que muestra Swagger UI al crearse la plantilla de notificación 43

Figura 15. Configurando una plantilla de notificación a modificar desde Swagger UI 44

Figura 16. Respuesta que muestra Swagger UI al modificarse la plantilla de notificación 45

Figura 17. Diseño de la tabla Templates y relación con la tabla Clients 46

Figura 18. Diseño de las tablas NotificationTemplate y NotificationTemplates_Domains 47

Figura 19. Actualización de las tablas Clients, FileTypes y Languages 48

Figura 20. Actualización de las tablas AuthenticationFactors y Notifications 49

Figura 21. Actualización de las tablas TermsConditions, Domains y EncryptTypes 50

Figura 22. Flujo normal – Agregar dominio 51

Figura 23. Flujo normal – Editar dominios 52

Figura 24. Diagrama BPMN – Agregar dominio 53

Figura 25. Diagrama BPMN – Editar dominio 54

Figura 26. Clase Entidad del desarrollo del caso de uso de Categoría de Tipos de Archivo 56

Figura 27. Métodos Get() y Post(), en la clase de servicio FileTypeService 57

Figura 28. Método Put() para modificar, en la clase de servicio FileTypeService 58

Figura 29. Clase Modelo/Entidad del desarrollo del caso de uso de Jornadas laborares 59

Figura 30. Método Post() para crear, en la clase de servicio CalendarService 60

Figura 31. Método Put() para modificar, en la clase de servicio CalendarService 61

Figura 32. Servicios implementados para el desarrollo de Categoría Tipos de Archivo 62

Figura 33. Configurando una categoría de tipos de archivo a crear desde Swagger UI 63

Figura 34. Respuesta que muestra Swagger UI al crearse la categoría de tipos de archivo 64

Figura 35. Configurando una categoría de tipos de archivo a modificar desde Swagger UI 65

Figura 36. Respuesta que muestra Swagger UI al modificar la categoría de tipos de archivo 66

Figura 37. Servicios implementados para el desarrollo de Jornadas Laborales 67

Figura 38. Configurando una jornada laboral a crear desde Swagger UI 68

Figura 39. Respuesta que muestra Swagger UI al crearse la jornada laboral 69

Figura 40. Configurando una jornada laboral a modificar desde Swagger UI 70

Figura 41. Respuesta que muestra Swagger UI al modificarse la jornada laboral 71

Figura 42. Diseño la tabla FileTypes y relación con la tabla Files 72

Figura 43. Diseño de las tablas Calendars, Calendars_WeekDays y NonWorkingdays 73

Figura 44. Flujo normal – Agregar esquema 74

Figura 45. Flujo normal – Modificar esquema 75

Figura 46. Diagrama BPMN flujo básico – Nuevo esquema 76

Figura 47. Diagrama BPMN flujo básico – Modificar esquema 77

Figura 48. Clase Entidad del desarrollo de Asignación de Roles a Cuenta de Usuario 79

Figura 49. Método Post() para crear, en la clase de servicio Domail_RoleService 80

Figura 50. Método Put() para modificar, en la clase de servicio Domail_RoleService 81

Figura 51. Clase Modelo/Entidad del desarrollo del caso de uso de Flujo de notificaciones 82

Figura 52. Método Post() para crear, en la clase de servicio NotificationFlowService 83

Figura 53. Método Put() para modificar, en la clase de servicio NotificationFlowService 84

Figura 54. Servicios implementados para el desarrollo de Asignación de Roles a Usuarios 85

Figura 55. Configurando una asignación de rol por dominio a crear desde Swagger UI 86

Figura 56. Respuesta que muestra Swagger UI al asignarse un rol por dominio 87

Figura 57. Configurando una asignación de rol por dominio a modificar desde Swagger UI 88

Figura 58. Respuesta que muestra Swagger UI al modificarse un rol por dominio 89

Figura 59. Servicios implementados para el desarrollo de Flujo de Notificaciones 90

Figura 60. Configurando un flujo de notificación a crear desde Swagger UI 91

Figura 61. Respuesta que muestra Swagger UI al crearse un flujo de notificación 92

Figura 62. Configurando un flujo de notificación a modificar desde Swagger UI 93

Figura 63. Respuesta que muestra Swagger UI al modificarse un flujo de notificación 94

Figura 64. Diseño de las tablas Domains_Roles y Users_Roles con relaciones 95

Figura 65. Diseño de las tablas NotificationsFlow y NotificationsFlow_Users_Roles 96

Figura 66. Configurar un proveedor de identidad en servidor de identidad WSO2 97

Figura 67. Configurar un proveedor de servicio en servidor de identidad WSO2 98

Figura 68. Configuración en detalle del proveedor de servicio 99

Figura 69. Prueba en Software Insomnia para obtener todos los proveedores de servicio 100

Figura 70. Prueba en Software Insomnia para consultar proveedor de servicio específico 101

Figura 71. Generar token de acceso (JWT) en software Insomnia para consumir servicios 102

Figura 72. Método Get() que retorna atributos editables en clase de servicio AttributeService 104

Figura 73. Método Patch() que actualiza atributos editables en clase de servicio UserService 105

Figura 74. Método Patch() que consulta políticas de seguridad por dominio 106

Figura 75. Se validan algunas políticas de seguridad para el cambio de contraseña 107

Figura 76. Servicio implementado para el desarrollo de Consultar y actualizar datos 108

Figura 77. Consultar los atributos editables del usuario en Swagger UI 109

Figura 78. Respuesta que muestra Swagger UI al consultar atributos editables del usuario 110

Figura 79. Configurando los atributos editables del usuario a actualizar en Swagger UI 111

Figura 80. Respuesta que muestra Swagger UI al actualizarse los atributos editables 112

Figura 81. Servicio en el que se implementó el desarrollo de Cambiar contraseña 113

Figura 82. Petición para actualizar la contraseña de la cuenta de usuario en Swagger UI 114

Figura 83. Respuesta que muestra Swagger UI al actualizarse la contraseña de la cuenta 115

Figura 84. Clase Entidad del desarrollo del caso de uso de configuración de excepciones 117

Figura 85. Método Get() que retorna todas las excepciones en clase ExceptionService 118

Figura 86. Método Put() para modificar, en la clase de servicio ExceptionService 119

Figura 87. Función ResetExceptions() para restablecer las excepciones a idiomas base 120

Figura 88. Clase Entidad del desarrollo del caso de uso de administración de esquemas 121

Figura 89. Método Post() para crear, en la clase de servicio SchemaService 122

Figura 90. Método Put() para modificar, en la clase de servicio SchemaService 123

Figura 91. Servicios implementados para el desarrollo de Configuración de Excepciones 124

Figura 92. Configurando un mensaje de error/excepción a modificar desde Swagger UI 125

Figura 93. Respuesta que muestra Swagger UI al modificarse un mensaje de excepción 126

Figura 94. Petición para restablecer los mensajes de error/excepción a idiomas base 127

Figura 95. Respuesta que muestra Swagger UI al restablecerse a los idiomas base 128

Figura 96. Servicios implementados para el desarrollo de Administración de Esquemas 129

Figura 97. Petición para configurar un esquema a crear desde Swagger UI 130

Figura 98. Respuesta que muestra Swagger UI al crearse una configuración de esquema 131

Figura 99. Configurando un esquema a modificar desde Swagger UI 132

Figura 100. Respuesta que muestra Swagger UI al modificarse un esquema 133

Figura 101. Añadir nueva tubería en Azure DevOps (CI/CD) 134

Figura 102. Seleccionar proyecto, repositorio y rama de compilación (CI/CD) 135

Figura 103. Configuración general de la nueva tubería (CI/CD) 136

Figura 104. Configuración del artefacto (CI/CD) 137

Figura 105. La nueva tubería al ejecutarse automáticamente (CI/CD) 138

Figura 106. Configurando un nuevo proceso de liberación (CI/CD) 139

Figura 107. Configurando condiciones previas al despliegue (CI/CD) 140

Figura 108. Habilitar la integración continua (CI/CD) 141

Figura 109. La entrega continua en ejecución (CI/CD) 142

Figura 110. La entrega continúa realizada correctamente (CI/CD) 143

Figura 111. Acceder a la Web API desde Internet (CI/CD) 144

Figura 112. Clase Entidad del desarrollo del caso de uso de Configurar Canales de Envío 147

Figura 113. Método Post() para crear, en la clase de servicio NotificationChannelService 148

Figura 114. Método Put() para modificar, en la clase de servicio NotificationChannelService 149

Figura 115. Clase Entidad del desarrollo del caso de uso de asociar preguntas de seguridad 150

Figura 116. Método Get() que retorna todas las preguntas de seguridad de los usuarios 151

Figura 117. Método Post() para asociar múltiples preguntas de seguridad a un usuario 152

Figura 118. Método Post() para asociar múltiples factores de autenticación a un dominio 153

Figura 119. Servicios implementados para el desarrollo de Configuración Canales de Envío 154

Figura 120. Petición para configurar un canal de notificación a crear desde Swagger UI 155

Figura 121. Respuesta que muestra Swagger UI al crearse un canal de notificación 156

Figura 122. Configurando un canal de notificación a modificar desde Swagger UI 157

Figura 123. Respuesta que muestra Swagger UI al modificarse un canal de notificación 158

Figura 124. Servicios implementados para el desarrollo de Asociar preguntas de seguridad 159

Figura 125. Petición para obtener las preguntas de seguridad asociadas con su respuesta 160

Figura 126. Respuesta que muestra Swagger UI al obtener preguntas de seguridad asociadas 161

Figura 127. Asociar múltiples preguntas de seguridad con su respuesta a un usuario 162

Figura 128. Respuesta que muestra Swagger UI al asociar múltiples preguntas de seguridad 163

Figura 129. Crear y modificar preguntas de seguridad con su respuesta a un usuario 164

Figura 130. Respuesta que muestra Swagger UI al crear y modificar preguntas de seguridad 165

Figura 131. Servicios implementados para el desarrollo de Configuración de Desafíos 166

Figura 132. Configurar múltiples factores de autenticación por dominio desde Swagger UI 167

Figura 133. Respuesta que muestra Swagger UI al configurar múltiples desafíos por dominio 168

Figura 134. Asociar y modificar múltiples factores de autenticación por dominio 169

Figura 135. Respuesta que muestra Swagger UI al asociar y modificar múltiples desafíos 170

## Introducción

En la universidad se adquieren las diferentes bases de conocimientos relacionados al área de formación, se nos prepara para encontrar la mejor solución a problemas y necesidades del mundo real que encontraremos al momento de entrar al mercado laboral, para ello, haciendo buen uso de las diferentes herramientas y recursos académicos que se ponen a nuestra disposición, permitiendo adquirir y fortalecer las habilidades requeridas para abordar de la mejor manera y en un futuro cercano; aquellos desafíos que se presenten en el mundo laboral relacionados al campo de formación. Para cumplir con el propósito de estar mejor preparados, la universidad nos ofrece la opción de realizar un proceso de aprendizaje más práctico y real llamado pasantía, como opción de grado, la cual realicé y describiré en este trabajo.

En el presente informe final de pasantía se exponen las diferentes actividades prácticas realizadas cada mes, logros, nuevas habilidades y conocimientos adquiridos, algunas dificultades presentadas y experiencias personales de realizar este proceso de aprendizaje.

La pasantía se realizó en la empresa BEXTechnology S.A. en convenio con la Universidad Nacional Abierta y a Distancia (UNAD), lo realizado en la pasantía fue lo acordado entre BEXTechnology S.A., la UNAD y el estudiante en el plan de trabajo aprobado el 27 de septiembre del 2021 por el Comité Curricular de la Cadena de Formación en Sistemas, la pasantía inicio el día 6 del mes de octubre del año 2021 y finalizó el 6 de abril de 2022, en el transcurso de veintiséis (26) semanas.

## Justificación

Una de las principales especialidades que abarca la ingeniería de sistemas es el conjunto de actividades y buenas prácticas enfocadas a desarrollar productos de software con la mayor calidad posible. Esto, debido a que el software tiende a ser cada vez más amplio y complejo a como se desarrollaba anteriormente, ya que con el pasar del tiempo los requerimientos cambian para el sistema y aumentan en cuanto a funcionalidad, seguridad, fiabilidad, usabilidad y mantenibilidad.

Esto ha incentivado a que los diversos procesos y metodologías de desarrollo de software hayan mejorado continuamente en el transcurso de las últimas cinco décadas, permitiendo actualmente desarrollar soluciones de software que satisfacen las necesidades presentes e incluso futuras del cliente.

Decidí escoger pasantía como opción de grado, ya que considero de gran importancia conocer y poner en práctica metodologías como Scrum y Kanban aplicadas a proyectos de desarrollo de software, las cuales permiten organizar adecuadamente desde un inicio, las diferentes actividades de desarrollo, permitiendo generar un producto final con calidad, además, de aprender la forma en que se aplican las últimas tendencias en desarrollo de las diferentes funcionalidades que llegare a tener una solución de software realizada mediante arquitecturas limpias (Clean Architecture), en este caso, mediante la creación de diferentes servicios RestFul de una web API, los cuales pueden ser consumidos/ejecutados por aplicaciones de diversos tipos, por ejemplo, aplicaciones de escritorio, web, móviles, Smart TV, etc.

Antes de realizar este proceso de pasantía, no tenía ninguna experiencia práctica con metodologías como Scrum y Kanban, ni la forma de cómo desarrollar diversos servicios RestFul para una web API mediante arquitecturas limpias (Clean Architecture), saber que adquiriría estos

conocimientos de forma práctica, los cuales actualmente son un estándar para el desarrollo de software multiplataforma, me motivó mucho a realizar pasantía como opción de grado. Además, porque es algo recomendado por empresas y profesionales del área de sistemas, adquirir nuevas habilidades y conocimientos que demanda el mercado laboral actual en herramientas de trabajo específicas, fue un factor considerable a tener en cuenta. Por estos motivos considero que la pasantía es la opción de grado más conveniente al terminar el pregrado e iniciar en el mundo laboral.

Durante la realización de la pasantía se trabajó en la nueva versión del software multiplataforma “Bext Self Reset”, el cual permite la autogestión de contraseña de usuarios finales. Este aplicativo estará en la capacidad de poder conectarse a múltiples repositorios de usuarios de tipo Azure AD, Ldap, Web service, por lo general, directorios activos, con la finalidad principal de poder cambiar la contraseña y actualizar la información de contacto de la cuenta de usuario cuando así lo desee el usuario final. El desarrollo se divide en tres secciones principales: desarrollo de funcionalidades administrativas (Web), desarrollo de funcionalidades de usuario final (Web) y desarrollo de funcionalidades a nivel de aplicación móvil.

La nueva versión de la herramienta multiplataforma “Bext Self Reset”, aún en desarrollo, resolverá muchas falencias y errores que presenta la versión actual y original, en cuanto a funcionalidad, rendimiento, seguridad y usabilidad. Será una nueva aplicación que, en conjunto, tendrá una mayor calidad final para los clientes.

# Objetivos

## Objetivo general

Generar nuevas funcionalidades a implementar en el software multiplataforma “Bext Self Reset”, el cual facilitara el proceso seguro de gestión y restablecimiento de contraseñas de cuentas de usuario vinculadas al directorio activo de los diferentes clientes; mediante la documentación de casos de uso, programación orientada a objetos en lenguaje C#, diseño y gestión de bases de datos y pruebas de servicios RESTful durante el último trimestre del año 2021 y primer trimestre del año 2022 en la empresa BEXTechnology S.A.

## Objetivos específicos

Registrar en documentos de casos de uso el comportamiento que tendrán algunos componentes funcionales del software multiplataforma, reglas de negocio y la interacción existente entre los diferentes actores (usuarios finales, administradores y sistema).

Identificar las diferentes reglas de negocio necesarias a aplicar en el desarrollo del software multiplataforma “Bext Self Reset”, para la correcta elaboración del producto según las necesidades de la empresa.

Producir el diseño de nuevas tablas y relaciones necesarias en base de datos para cada desarrollo back-end a implementar de los casos de uso asignados.

Aplicar las funcionalidades requeridas para cada caso de uso mediante la implementación de las reglas de negocio en el desarrollo back-end, haciendo uso de arquitecturas limpias (Clean Architecture) para añadir una mayor calidad final al producto, facilitando las actividades de mantenibilidad y extensibilidad a largo plazo.

Verificar la funcionalidad y aplicación de las reglas de negocio para cada caso de uso desarrollado, mediante pruebas hechas a los diferentes servicios RESTFul implementados.

## Presentación de la Empresa

**BEXTechnology S.A.**

Es una empresa que ofrece diversos servicios de consultoría tecnológica. Se enfoca en dar respuesta rápida y eficiente a las necesidades tecnológicas de las empresas y clientes. Se especializa en brindar diversas soluciones de computación en la nube (Cloud) mediante las plataformas de Azure, AWS y Google Cloud.

Se describe en el sitio web de Linkedin como:

> BEXTechnology es una empresa de servicios profesionales en tecnología. Cuenta con un equipo especializado en múltiples soluciones con la capacidad de dar respuesta eficiente y efectiva a las necesidades de las empresas, de incorporar tecnología que mejore sus procesos de negocio.
>
> Hoy en día somos Microsoft Gold Certified Partner, brindando servicios de alto impacto y valor para su organización. (BEXTechnology S.A, 2022)

Dispone de aplicaciones y servicios propios, orientados a solucionar las necesidades de gestión de identidades y acceso que presenten las empresas e instituciones. Brinda servicios de análisis de datos, ciberseguridad, transformación digital, soporte a las tecnologías de información, estrategias y operaciones en la nube, soluciones de desarrollo de software, SharePoint.

**Visión**

"Ofrecer la mejor experiencia a los clientes que deciden apropiar tecnologías de información en su implementación y uso, beneficiando sus negocios a nivel operativo, financiero y de mercadeo" (BEXTechnology S.A., 2022).

**Propósito**

Ser la compañía más reconocida del sector tecnológico Latinoamericano, con nivel de competencia mundial, reconocidos por nuestra capacidad de creación de valor para nuestros clientes, ofreciendo soluciones efectivas en productos y servicios innovadores y de calidad, con un equipo que se caracteriza por tener la mejor calidad humana y profesional. (BEXTechnology S.A., 2022)

## Metodología

Se aplicó el enfoque cualitativo al desarrollo del plan de trabajo de pasantía, dado que los datos obtenibles del contexto de la situación de estudio mediante la aplicación de métodos de investigación; eran interpretaciones, experiencias y descripciones de los participantes (clientes).

Al no disponer de la facilidad y abundancia de datos cuantificables a obtener de los diferentes participantes (clientes), se decidió escoger el enfoque cualitativo.

La metodología cualitativa es un conjunto de técnicas interpretativas con las que se busca describir y explicar un fenómeno o suceso para lograr entenderlo mejor, sin la obligación de cuantificarlo. (Grijalva & Mancheno, 2012, pág. 1)

El método de investigación usado para la recolección de datos fue la observación participante y no participante, consiguiendo datos verbales y experiencias de usuario al interactuar con la antigua versión del software multiplataforma “Bext Self Reset” que utilizan actualmente algunos clientes.

## Descripción detallada del plan de trabajo

Durante el desarrollo de la pasantía en la empresa BEXTechnology S.A., se realizarán las diferentes actividades de desarrollo de un nuevo producto de software llamado “Bext Self Reset”, el cual tiene por funcionalidad principal el poder conectarse a diferentes repositorios de usuarios para cambiar la contraseña de cuentas vinculadas al directorio activo de una empresa u organización. Este cambio de contraseña se verá reflejado en todas las aplicaciones vinculadas a la cuenta del usuario en el directorio activo. Para apoyar el desarrollo de este producto se empleará un total de veintiséis (26) semanas, realizando las siguientes actividades por mes:

1. Primer mes (octubre 2021):
    - Semana 1, 2, 3 y 4: desarrollo Back-end de los distintos servicios HTTP (Get, Post, Put, Delete) de una REST API Web 2.0 con .NET Core en C#.
    - Semana 1 y 2: documentación de casos de uso del software multiplataforma a desarrollar “Bext Self Reset”.
    - Semana 1, 2, 3 y 4:  probar el correcto funcionamiento de los distintos servicios HTTP (Get, Post, Put, Delete) de la API Web 2.0 desde Swagger UI y Postman.
    - Semana 4: actualización del documento ‘[BSR] Diccionario Base de Datos.docx’, el cual lleva el registro de todas las tablas que tiene la base de datos del aplicativo web.
    - Semana 1 y 3: análisis y diseño de bases de datos en motor SQL Server para el aplicativo multiplataforma a desarrollar.

2. Segundo mes (noviembre 2021):
    - Semana 1, 2, 3 y 4: desarrollo Back-end de los distintos servicios HTTP (Get, Post, Put, Delete) de una REST API Web 2.0 con .NET Core en C#.
    - Semana 1 y 2: documentación de casos de uso del software multiplataforma a desarrollar “Bext Self Reset”.
    - Semana 1, 2, 3 y 4: probar el correcto funcionamiento de los distintos servicios HTTP (Get, Post, Put, Delete) de la API Web 2.0 desde Swagger UI y Postman.
    - Semana 1 y 3: análisis y diseño de bases de datos en motor SQL Server para el aplicativo multiplataforma a desarrollar.
3. Tercer mes (diciembre 2021):
    - Semana 1, 2, 3 y 4: desarrollo Back-end de los distintos servicios HTTP (Get, Post, Put, Delete) de una REST API Web 2.0 con .NET Core en C#.
    - Semana 1 y 2: documentación de casos de uso del software multiplataforma a desarrollar “Bext Self Reset”.
    - Semana 1, 2, 3 y 4: probar el correcto funcionamiento de los distintos servicios HTTP (Get, Post, Put, Delete) de la API Web 2.0 desde Swagger UI y Postman.
    - Semana 1 y 3: análisis y diseño de bases de datos en motor SQL Server para el aplicativo multiplataforma a desarrollar.
4. Cuarto mes (enero 2022):
    - Semana 1, 2, 3 y 4: desarrollo Back-end de los distintos servicios HTTP (Get, Post, Put, Delete) de una REST API Web 2.0 con .NET Core en C#.

- Semana 1, 2, 3 y 4: probar el correcto funcionamiento de los distintos servicios HTTP (Get, Post, Put, Delete) de la API Web 2.0 desde Swagger UI y Postman.

5. Quinto mes (febrero 2022):
    - Semana 1, 2 y 3: desarrollo Back-end de los distintos servicios HTTP (Get, Post, Put, Delete) de una REST API Web 2.0 con .NET Core en C#.
    - Semana 1, 2 y 3: probar el correcto funcionamiento de los distintos servicios HTTP (Get, Post, Put, Delete) de la API Web 2.0 desde Swagger UI y Postman.
    - Semana 4: despliegue de soluciones para ambientes QA y de producción en la nube Microsoft Azure, implementación y configuración de servidores de identidad.
6. Sexto mes (marzo 2022):
    - Semana 1: implementación y configuración de servidores de identidad, despliegue de soluciones para ambientes QA y de producción en la nube Microsoft Azure.
    - Semana 2: desarrollo Front-end (React.js).
    - Semana 3 y 4: desarrollo Back-end de los distintos servicios HTTP (Get, Post, Put, Delete) de una REST API Web 2.0 con .NET Core en C#.
    - Semana 4: actualización el documento ‘[BSR] Diccionario Base de Datos.docx’, el cual lleva el registro de todas las tablas que tiene la base de datos del aplicativo web.

**Cronograma**

**Tabla 1.**

*Cronograma plan de trabajo desarrollado*.

| Actividad | Mes 1 | Mes 2 | Mes 3 | Mes 4 | Mes 5 | Mes 6 |
|---|---|---|---|---|---|---|
| Desarrollo Back-end de los distintos servicios HTTP (Get, Post, Put, Delete) de una REST API Web 2.0 con .NET Core en C# | X | X | X | X | X | X |
| Documentación de casos de uso del software multiplataforma a desarrollar “Bext Self Reset” | X | X | X | | | |
| Desarrollo Front-end (React.js) | | | | | | X |
| Probar el correcto funcionamiento de los distintos servicios HTTP (Get, Post, Put, Delete) de la API Web 2.0 desde Swagger UI y Postman | X | X | X | X | X | |
| Actualización el documento ‘[BSR] Diccionario Base de Datos.docx’, el cual lleva el registro de todas las tablas que tiene la base de datos del aplicativo web | X | | | | | X |
| Implementación y configuración de servidores de identidad | | | | | X | X |
| Análisis y diseño de bases de datos en motor SQL Server para el aplicativo multiplataforma a desarrollar | X | X | X | | | |
| Despliegue de soluciones para ambientes QA y de producción en la nube Microsoft Azure | | | | | X | X |

Fuente: el autor.

**Resultados esperados**

**Tabla 2.**

*Resultados y productos esperados*.

| Resultado/producto esperado | Indicador |
|---|---|
| Casos de uso implementados en desarrollo back-end | - Número de casos de uso implementados en back-end / número de casos de uso asignados<br>- Porcentaje de cumplimiento |
| Casos de uso implementados en el desarrollo Front-end (React.js) | - Número de casos de uso implementados en front-end / número de casos de uso asignados<br>- Porcentaje de cumplimiento |
| Casos de uso documentados / creados | - Número de casos de uso documentados / número de casos de uso asignados<br>- Porcentaje de cumplimiento |
| Configurar canalización para la integración continua y entrega continua (CI/CD) en Azure DevOps | - Número de canalizaciones de integración continua y entrega continúa implementadas / número de canalizaciones de integración continua y entrega continua asignadas<br>- Porcentaje de cumplimiento |
| Diseño de bases de datos | - Número de tablas implementadas en base de datos / número de tablas diseñadas<br>- Porcentaje de cumplimiento |
| Pruebas del correcto funcionamiento de los servicios RESTful mediante Swagger UI | - Número de servicios RESTful probados y validados / número de servicios RESTful implementados<br>- Porcentaje de cumplimiento |
| Implementación y configuración de servidores de identidad | - Número de configuraciones de proveedores de identidad y proveedores de servicios implementados / número de configuraciones de proveedores de identidad y proveedores de servicios asignados<br>- Porcentaje de cumplimiento |

Fuente: el autor.

**Resultados obtenidos**

**Tabla 3.**

*Resultados y productos obtenidos.*

| Resultado/producto esperado | Resultados de indicadores |
|---|---|
| Casos de uso implementados en desarrollo back-end | - 13 casos de uso implementados en back-end / 13 casos de uso asignados<br>- Cumplimiento del 100% |
| Casos de uso implementados en el desarrollo Front-end (React.js) | - 0 casos de uso implementados en front-end / 0 casos de uso asignados<br>- Cumplimiento del 100% |
| Casos de uso documentados / creados | - 2 casos de uso documentados / 2 casos de uso asignados<br>- Cumplimiento del 100% |
| Configurar canalización para la integración continua y entrega continua (CI/CD) en Azure DevOps | - 1 canalización de integración continua y entrega continua (CI/CD) implementada / 1 canalización de integración continua y entrega continua (CI/CD) asignada<br>- Cumplimiento del 100% |
| Diseño de bases de datos | - 11 tablas implementadas en base de datos / 11 tablas diseñadas<br>- Cumplimiento del 100% |
| Pruebas del correcto funcionamiento de los servicios RESTful mediante Swagger UI | - 64 servicios/endpoints RESTful probados y validados correctamente / 64 servicios/endpoints RESTful implementados<br>- Cumplimiento del 100% |
| Implementación y configuración de servidores de identidad | - 1 configuración de proveedor de identidad y 1 configuración de proveedor de servicios implementados / 1 configuración de proveedor de identidad y 1 configuración de proveedor de servicios asignados<br>- Cumplimiento del 100% |

Fuente: el autor.

## Descripción de la pasantía

Se realizó la pasantía en la unidad de “Identity and Access Solutions” de la empresa BEXTechnology S.A., con el fin de aplicar en un ambiente real del mundo laboral, los conocimientos adquiridos durante el proceso formativo del programa de ingeniería de sistemas relacionados al desarrollo de software.

Se estudió de manera rápida, el uso del lenguaje de programación C#, necesario para el desarrollo de software en la plataforma .NET. Se revisó el estado inicial del desarrollo del proyecto “Bext Self Reset” en cuanto a funcionalidad y nuevas características que tendrá a futuro, respecto a la versión anterior.

Se inició aplicando las diferentes reglas de negocio al desarrollo back-end del producto que solicitan los diferentes documentos de casos de uso que me fueron asignados, diseñando algunas nuevas tablas a implementar en la base de datos requeridas para algunos desarrollos, probando la correcta funcionalidad y cumplimiento de las reglas de negocio en los diferentes servicios RESTful implementados para cada desarrollo back-end de los casos de uso, implementando la canalización de integración y entrega continua desde Azure DevOps para que cada desarrollo que se complete en back-end mediante solicitud de “Pull Request” sea funcional y comprobable en un nuevo sitio web de pruebas QA.

Se documentó la funcionalidad, procedimiento e interacción de los actores involucrados en algunos casos de uso próximos a desarrollar, también algunas pruebas para la implementación y configuración de servidores de identidad y la actualización de un documento de guía que describe en detalle las tablas y relaciones agregadas a la base de datos del producto.

## Desarrollo del plan de trabajo de la pasantía

El cumplimiento de las diferentes actividades efectuadas en el transcurso de la pasantía, según lo acordado en el cronograma y objetivos planeados, se realizó en productos finalizados y entregados cada mes, los cuales se detallan a continuación:

**1.** En el primer mes (6 de octubre del 2021 a 5 de noviembre del 2021) se hizo:

- Participación de varias reuniones durante la primera semana con el coordinador de pasantía, Cristhian Camilo Medina Solano, quien me dio varias indicaciones relacionadas al desarrollo del proyecto “Bext Self Reset”, tales como: funcionalidad deseada en la nueva versión del producto, herramientas y recursos de software necesarios para el desarrollo, buenas prácticas de desarrollo de software, explicación y aplicación de arquitecturas limpias (Clean Architecture) al desarrollo.
- El estudio de diferentes conceptos necesarios para el desarrollo de software mediante el lenguaje de programación C#.
- El estudio y desarrollo del caso de uso [CU_19] Personalización BSR, añadiendo las reglas de negocio y funcionalidades necesarias en el código, para permitir que un usuario administrador del sistema pueda gestionar la configuración del cambio de apariencia y fuentes de la aplicación que tendrá el gestor de contraseñas BSR. El usuario administrador establecerá los distintos valores que tendrán los elementos del sistema, tales como imagen Logo Banner, Imagen Background, colores, tamaños, y fuentes que tendrán los botones y textos que se muestren en el sistema.

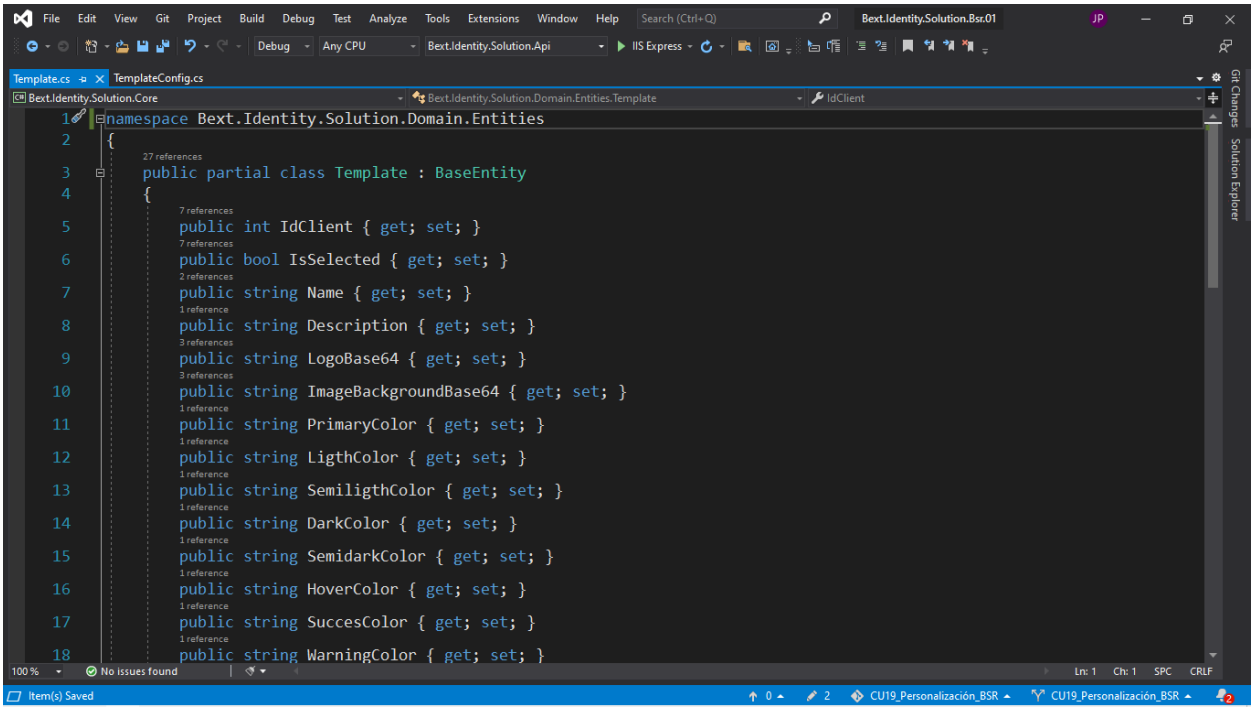
- El modelado de la tabla Templates, con cada uno de sus campos/ atributos y la relación que tendrá con la tabla Clients, en el software Día.
- El estudio y desarrollo del caso de uso [CC25] Administración Plantillas de Notificaciones, añadiendo las reglas de negocio y funcionalidades necesarias en el código, para permitir que un usuario administrador del sistema pueda gestionar la configuración de las Plantillas de Notificaciones que tendrá el Gestor de Contraseñas BSR. Para ello, el administrador seleccionará los dominios a los que se aplicará la plantilla seleccionada, el idioma de la plantilla, tipo de plantilla (SMS, email, push) y el asunto de la notificación.
- El modelo de la tabla NotificationTemplates_Domains, NotificationTemplates y Notification, con cada uno de sus campos/ atributos y las relaciones que tendrá con otras tablas, en el software Día.
- La actualización del documento [BSR] Diccionario Base de Datos.docx, el cual lleva el registro de todas las tablas que tiene la base de datos para el aplicativo multiplataforma.
- La documentación del caso de uso [CC17-N°2] Administración de Dominio.

A continuación, se comparten algunas vistas que detallan las actividades realizadas durante este mes:

- Desarrollo back-end de los casos de uso

**Figura 1.**

*Clase Modelo/Entidad del desarrollo del caso de uso de Personalización BSR*




Fuente: el autor.

**Descripción:** Se muestran cada una de las propiedades requeridas de la clase Template para el desarrollo del caso de uso de Personalización BSR. La propiedad IdClient es una llave foránea que se deriva de la clase modelo/entidad Client, la cual tiene una relación de uno a muchos hacia plantillas. Algunas propiedades representan las imágenes en formato base64, otras propiedades representan los colores en códigos hexadecimales que se guardaran para cada registro en la base de datos

**Figura 2.**

*Método Post() para crear, en la clase de servicio TemplateService*

```
2 references
public async Task<Template> Post(Template entity)
{
    entity = SetAuditFields(entity);
    string[] permittedExtensions = { ".svg", ".webp", ".png" };
    int width = 0, height = 0;
    MagickFormat format;

    using (var image = new MagickImage(Convert.FromBase64String(entity.LogoBase64)))
    {
        width = image.BaseWidth;
        height = image.BaseHeight;
        format = image.Format;
    }
    var extension = Path.GetExtension("." + format).ToLowerInvariant();

    if (!string.IsNullOrEmpty(extension) && permittedExtensions.Contains(extension))
    {
        if (width <= 560 && height <= 220)
        { } else throw new BadRequestBusinessException("¡Error! - El ancho de la imagen logo debe ser menor o igual
    } else throw new BadRequestBusinessException("¡Error! - El archivo no tiene alguna de las extensiones permitida

    using (var image = new MagickImage(Convert.FromBase64String(entity.ImageBackgroundBase64)))
    {
        width = image.BaseWidth;
        height = image.BaseHeight;
        format = image.Format;
    }
    extension = Path.GetExtension("." + format).ToLowerInvariant();

    if (!string.IsNullOrEmpty(extension) && permittedExtensions.Contains(extension))
    {
        if (width <= 1920 && height <= 1080)
        { } else throw new BadRequestBusinessException("¡Error! - El ancho de la imagen de fondo debe ser menor o i
    } else throw new BadRequestBusinessException("¡Error! - El archivo no tiene alguna de las extensiones permitida

    if (entity.IsSelected == true)  // Si IsSelected = true, las demás plantillas del cliente serán IsSelected = fa
    {
        var entitys = _unitAdmin.templateRepository.GetIsSelectedByClient(entity.IdClient).ToList();

        foreach (var entidad in entitys)
        {
            if (entidad != null)
            {
                entidad.IsSelected = false;
                entidad.IdUpdateUser = _userLogged.userId;
```

Fuente: el autor.

**Descripción:** El método Post() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta creación de plantillas de personalización en el sistema. Los formatos base64 que se reciben para cada imagen se transforman a imágenes para validar tamaños máximos permitidos para la imagen logo e imagen de fondo, formatos permitidos y si la plantilla quedara seleccionada para mostrarse/aplicarse en el sistema.

**Figura 3.**

*Método Put() para modificar, en la clase de servicio TemplateService*

```
2 references
public async Task<Template> Put(Template entity)
{
    entity = SetAuditFields(entity);
    string[] permittedExtensions = { ".svg", ".webp", ".png" };
    int width = 0, height = 0;
    MagickFormat format;

    if (await _unitAdmin.templateRepository.GetById(entity.Id) != null)  // Valida que exista la Plantilla con el I
    {
        using (var image = new MagickImage(Convert.FromBase64String(entity.LogoBase64)))
        {
            width = image.BaseWidth;
            height = image.BaseHeight;
            format = image.Format;
        }
        var extension = Path.GetExtension("." + format).ToLowerInvariant();

        if (!string.IsNullOrEmpty(extension) && permittedExtensions.Contains(extension))
        {
            if (width <= 560 && height <= 220)
            { } else throw new BadRequestBusinessException("¡Error! - El ancho de la imagen logo debe ser menor o i
        } else throw new BadRequestBusinessException("¡Error! - El archivo no tiene alguna de las extensiones permi

        using (var image = new MagickImage(Convert.FromBase64String(entity.ImageBackgroundBase64)))
        {
            width = image.BaseWidth;
            height = image.BaseHeight;
            format = image.Format;
        }
        extension = Path.GetExtension("." + format).ToLowerInvariant();

        if (!string.IsNullOrEmpty(extension) && permittedExtensions.Contains(extension))
        {
            if (width <= 1920 && height <= 1080)
            { } else throw new BadRequestBusinessException("¡Error! - El ancho de la imagen de fondo debe ser menor
        } else throw new BadRequestBusinessException("¡Error! - El archivo no tiene alguna de las extensiones permi

        if (entity.IsSelected == true)  // Si IsSelected = true, las demás plantillas de
        {
            var entitys = _unitAdmin.templateRepository.GetCustomerTemplate(entity.Id, entity.IdClient).ToList();

            foreach (var entidad in entitys)
            {
```

Fuente: el autor.

**Descripción:** El método Put() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta modificación de plantillas de personalización en el sistema. Los formatos base64 que se reciben para cada imagen se transforman a imágenes para validar tamaños máximos permitidos para la imagen logo e imagen de fondo, formatos permitidos y si la plantilla quedara seleccionada para mostrarse/aplicarse en el sistema.

**Figura 4.**

*Clase Modelo/Entidad del desarrollo del caso de uso de Plantillas de Notificación*

```
using Bext.Identity.Solution.Domain.Entities;
using System.Collections.Generic;

namespace Bext.Identity.Solution.Core.Entities
{
    29 references
    public class NotificationTemplate : BaseEntity
    {
        0 references
        public NotificationTemplate()
        {
            Notificationtemplate_domain = new HashSet<NotificationTemplate_Domain>();
        }

        9 references
        public int IdNotification { get; set; }
        8 references
        public int IdLanguage { get; set; }
        1 reference
        public string Subject { get; set; }
        4 references
        public string Template { get; set; }
        2 references
        public bool IsHtml { get; set; }

        1 reference
        public virtual Notification IdNotificationNavigation { get; set; }
        7 references
        public virtual ICollection<NotificationTemplate_Domain> Notificationtemplate_domain { get; set; }
    }
}
```

Fuente: el autor.

**Descripción:** Se muestran cada una de las propiedades requeridas de la clase NotificationTemplate para el desarrollo del caso de uso de configuración de plantillas de notificaciones. La propiedad IdNotification es una llave foránea que se deriva de la clase modelo/entidad Notificatión, la cual tiene una relación de uno a muchos hacia plantillas de notificaciones. Una propiedad es para el asunto de la plantilla, otra para el texto de la plantilla y si es en formato HTML.

Esta clase NotificationTemplate tiene una relación de uno a muchas plantillas de notificación por dominio (NotificationTemplate_Domain).

**Figura 5.**

*Método Post() para crear, en la clase de servicio NotificationTemplateService*

```
2 references
public async Task<NotificationTemplate> Post(NotificationTemplate entity)
{
    entity = SetAuditFields(entity);

    // Se valida que se ingrese al menos un dominio
    if(entity.Notificationtemplate_domain.Count() == 0)
        throw new NotFoundBusinessException("Debe ingresar al menos un Dominio");

    NotificationTemplate newNotificationTemplates = entity;
    ICollection<NotificationTemplate_Domain> values = entity.Notificationtemplate_domain;
    newNotificationTemplates.Notificationtemplate_domain = new List<NotificationTemplate_Domain>();

    var domains = values.Select(x => x.IdDomain).ToList();

    // Se valida si hay dominios repetidos a agregar
    if (domains.GroupBy(n => n).Any(c => c.Count() > 1))
        throw new NotFoundBusinessException("Los dominios asociados a una Plantilla de notificación deben ser diferentes

// Se obtienen las plantillas de notificación por notificación e idioma para validar si ya existe exactamente la plantil
IEnumerable<NotificationTemplate> notificationTemplate = _unitAdmin.notificationTemplateRepository.GetNotificationTempla
var notificationIsActive = notificationTemplate.Select(x => x.IdNotificationNavigation.Active).FirstOrDefault();

if (notificationIsActive == false)
    throw new NotFoundBusinessException("No existe la notificación de la Plantilla de Notificación a crear");

if (notificationTemplate.Count() >= 1)
{
    foreach (var notificationtemplate in notificationTemplate)
    {
        var listDomains = notificationtemplate.Notificationtemplate_domain.Select(x => x.IdDomain).ToList();
        var areEquivalent = (domains.Count == listDomains.Count) && !domains.Except(listDomains).Any();

        if (areEquivalent)
            throw new NotFoundBusinessException("No se puede agregar otra Plantilla de Notificación con la misma notific
    }
}

foreach (var notificationtemplate_domain in values)
{
    notificationtemplate_domain.CreationDate = entity.CreationDate;
    notificationtemplate_domain.CreationMachine = entity.CreationMachine;
    notificationtemplate_domain.IdCreatorUser = entity.IdCreatorUser;
```

Fuente: el autor.

**Descripción:** El método Post() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta creación de plantillas de notificación por dominio en el sistema. Se valida que se ingrese al menos un dominio a asociar, que los dominios sean distintos y que no queden plantillas de notificación repetidas en base de datos.

Se guardan registros en la tabla NotificationTemplate y NotificationTemplate_Domain en base de datos.

**Figura 6.**

*Método Put() para modificar, en la clase de servicio NotificationTemplateService*

```
2 references
public async Task<NotificationTemplate> Put(NotificationTemplate entity)
{
    entity = SetAuditFields(entity);
    if (entity.Id > 0)  // Valida que el Id de la Plantilla de Notificación sea mayor a 0
    {
        // Se obtienen todos los datos de la Plantilla de Notificación a modificar
        NotificationTemplate notificationTemplate = _unitAdmin.notificationTemplateRepository.GetNotificationTemplate(en
        IEnumerable<NotificationTemplate_Domain> notificationtemplates_domains = notificationTemplate.Notificationtempla

        // Dominios originales asociados a la plantilla de notificación a modificar
        var originalDomains = notificationtemplates_domains.Select(x => x.IdDomain).ToList();

        // Valida que exista la Plantilla de Notificación con el Id indicado
        if (notificationTemplate != null && notificationTemplate.Active == true)
{
    NotificationTemplate newNotificationTemplates = entity;
    ICollection<NotificationTemplate_Domain> values = entity.Notificationtemplate_domain;
    newNotificationTemplates.Notificationtemplate_domain = new List<NotificationTemplate_Domain>();

    // Se separa por dominios a agregar, editar y por registros
    var finalDomains = new List<int>();
    var domainsToAdd = values.Where(x => x.Id == 0).Select(x => x.IdDomain).ToList();
    var toAdd = values.Where(x => x.Id == 0).ToList();
    var domainsToEdit = values.Where(x => x.Id > 0).Select(x => x.IdDomain).ToList();
    var toEdit = values.Where(x => x.Id > 0).ToList();
    finalDomains = domainsToAdd;

    // Se valida si los dominios finales quedarían repetidos en la Base de Datos
    foreach (var notificationtemplate_domain in notificationtemplates_domains)
    {
        var selectRecord = toEdit.Where(x => x.Id == notificationtemplate_domain.Id).Select(x => x.IdDomain).ToList();
        if(selectRecord.Count() == 1)
            finalDomains.Add(selectRecord.FirstOrDefault());
        else
            finalDomains.Add(notificationtemplate_domain.IdDomain);
    }

    if (finalDomains.GroupBy(n => n).Any(c => c.Count() > 1))
        throw new NotFoundBusinessException("Los dominios asociados a una Plantilla de notificación deben ser diferentes

// Se obtienen las plantillas de notificación por notificación e idioma diferentes a la actual para validar si alguna co
IEnumerable<NotificationTemplate> notificationTemplates = _unitAdmin.notificationTemplateRepository.NotificationTemplate
var notificationIsActive = notificationTemplates.Select(x => x.IdNotificationNavigation.Active).FirstOrDefault();

if (notificationIsActive == false)
    throw new NotFoundBusinessException("No existe la notificación de la Plantilla de Notificación a crear");
```

Fuente: el autor.

**Descripción:** El método Put() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta modificación de plantillas de notificación por dominio en el sistema. Se valida que se ingrese al menos un dominio a asociar, que los dominios sean distintos y que no queden plantillas de notificación repetidas en base de datos.

Se guardan registros en la tabla NotificationTemplate y NotificationTemplate_Domain en base de datos.

- Pruebas de los servicios implementados en desarrollo back-end

El desarrollo de la API RESTFul hace uso del estándar OpenAPI, ampliamente utilizado en la industria para procesos de describir, desarrollar, probar y documentar API compatibles con REST. En general, permite un mejor flujo del desarrollo, documentación y pruebas de la API, mejorando la coordinación del desarrollo entre equipos back-end y front-end.

**Figura 7.**

*Servicios implementados para el desarrollo de Personalización BSR*

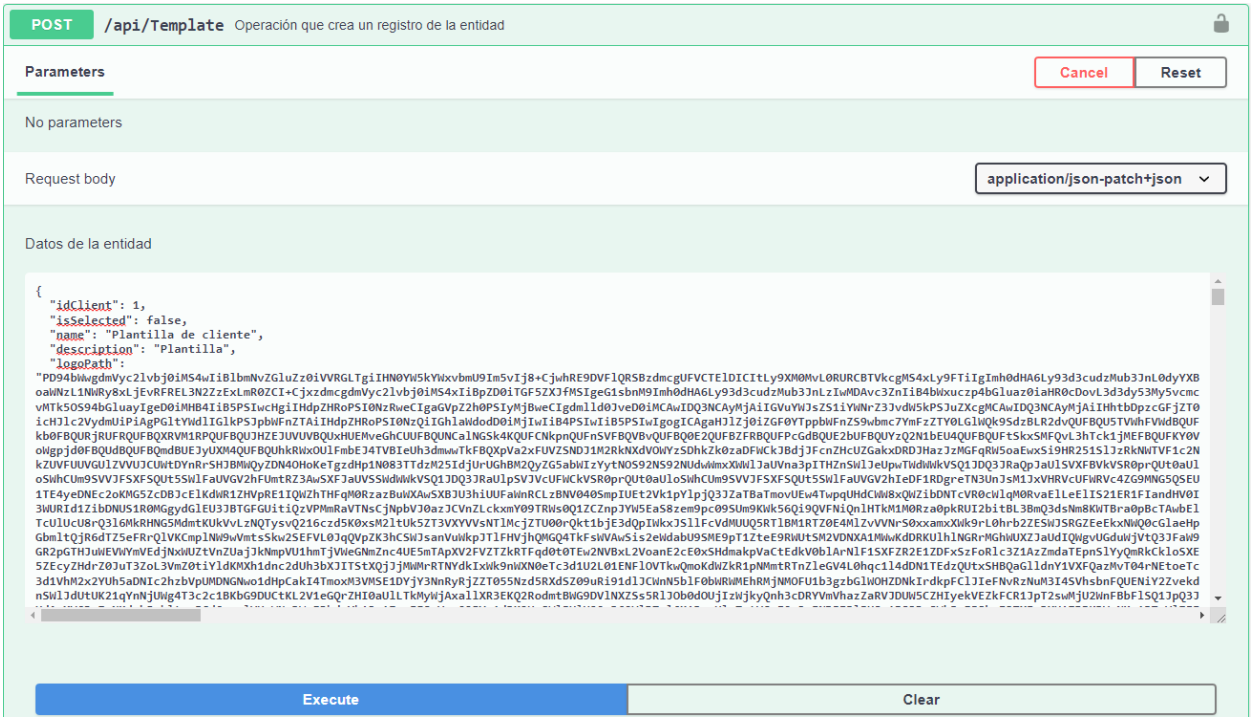
Fuente: el autor.

**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestran los servicios implementados para crear (Post), modificar (Put), eliminar (Delete) y obtener (Get) objetos de plantillas, según las reglas de negocio, relacionados al desarrollo del caso de uso de Personalización BSR.

Mediante las plantillas, los clientes personalizan los aspectos visuales del aplicativo, adecuándolo a la imagen corporativa u organización.

**Figura 8.**

*Configurar una petición para crear una plantilla desde Swagger UI*

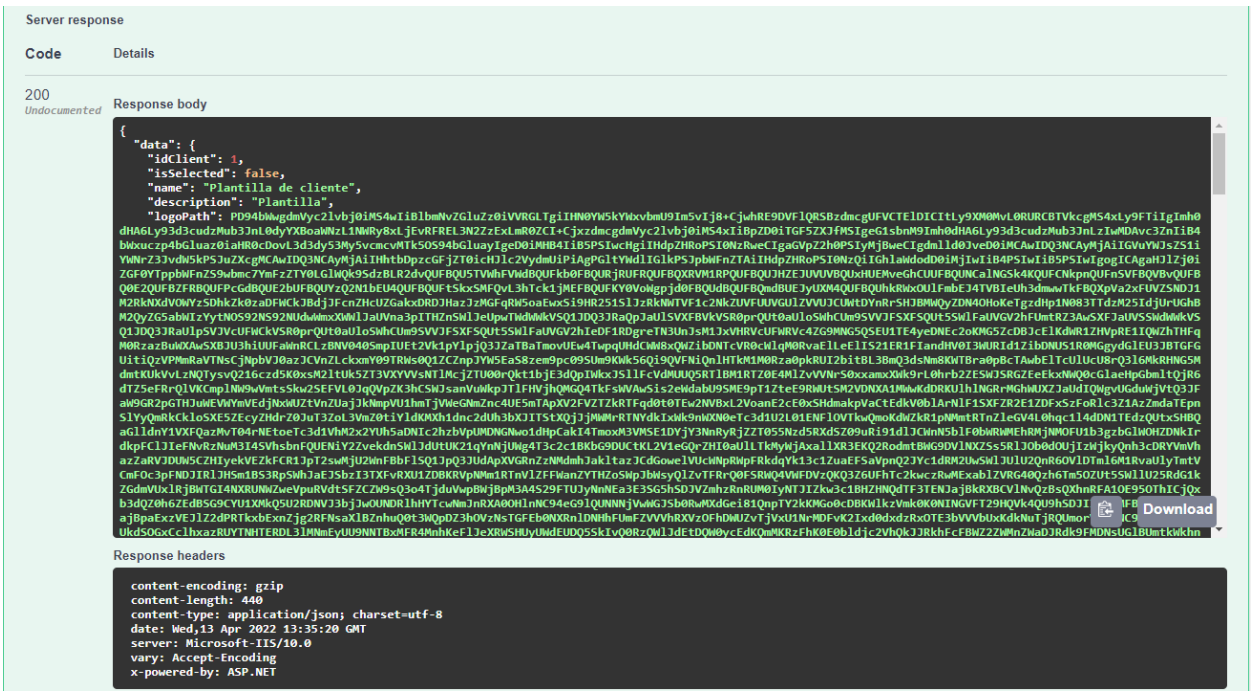



Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se envía la petición y se crea una plantilla de personalización, la cual se guarda en base de datos. Desde el front-end se envía al back-end dos imágenes en formato base64, las cuales se convertirán a imágenes para validar las diferentes reglas de negocio.

**Figura 9.**

*Respuesta que muestra Swagger UI al crearse la plantilla de personalización*

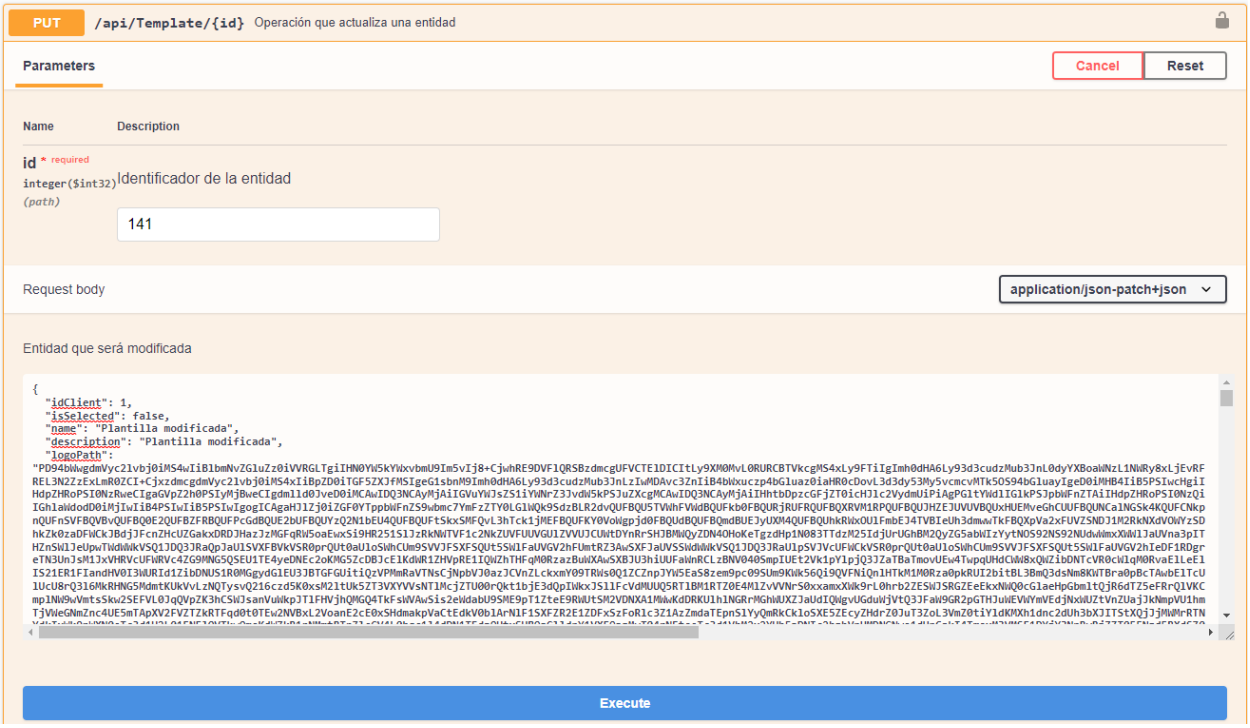



Fuente: el autor.

**Descripción:** Desde el back-end se retorna la plantilla de personalización creada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 10.**

*Configurar una petición para modificar una plantilla desde Swagger UI*

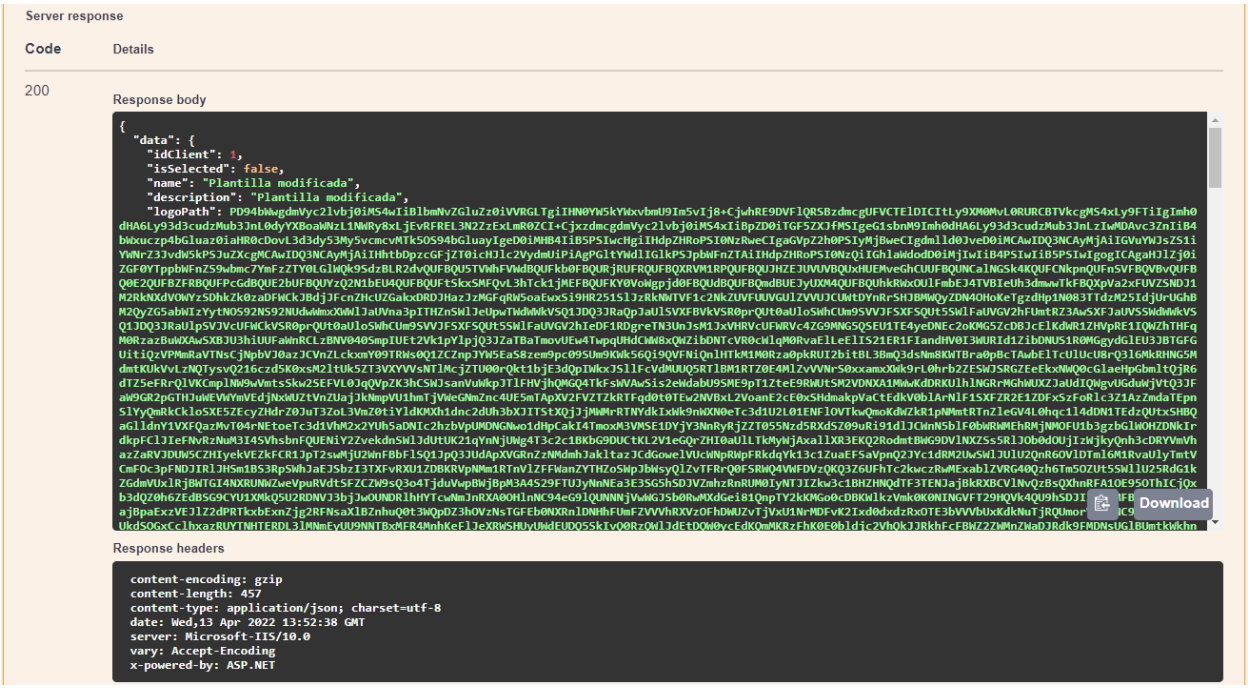



Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se indica el identificador del registro a modificar, se envía la petición y se actualiza la plantilla de personalización en base de datos. Desde el front-end se envía al back-end dos imágenes en formato base64, las cuales se convertirán a imágenes para validar las diferentes reglas de negocio.

**Figura 11.**

*Respuesta que muestra Swagger UI al modificarse la plantilla de personalización*

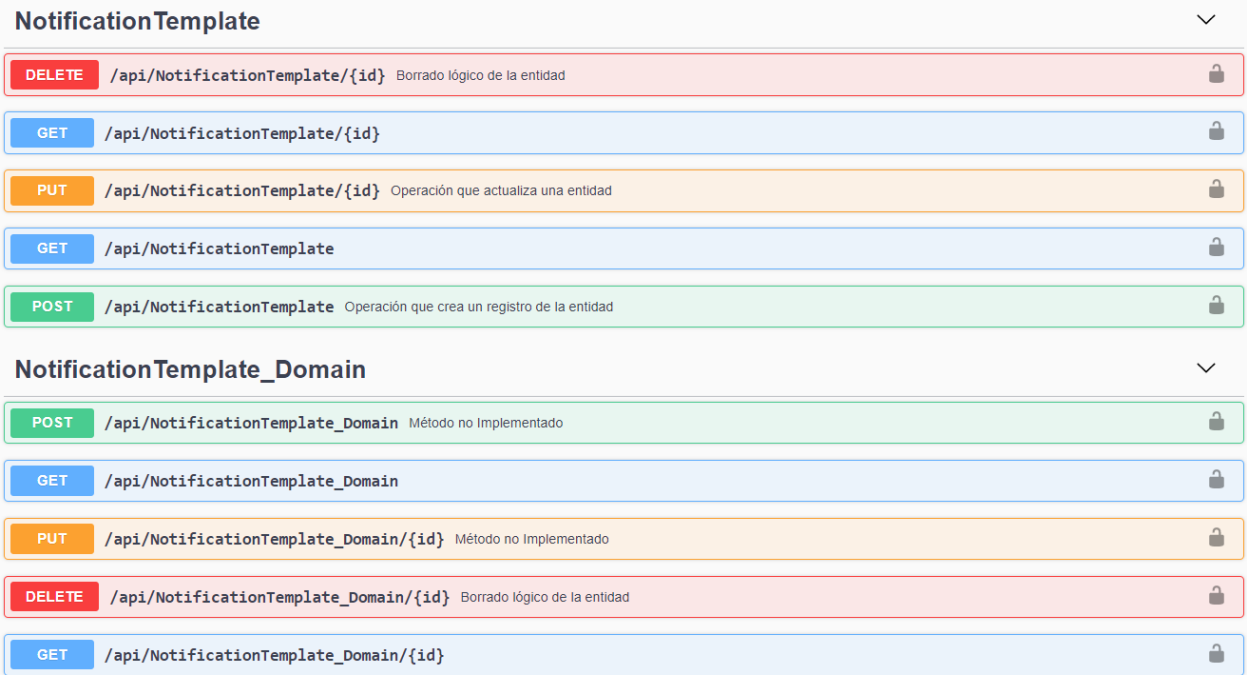



Fuente: el autor.

**Descripción:** Desde el back-end se retorna la plantilla de personalización modificada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 12.**

*Servicios implementados para el desarrollo de Plantillas de Notificaciones*



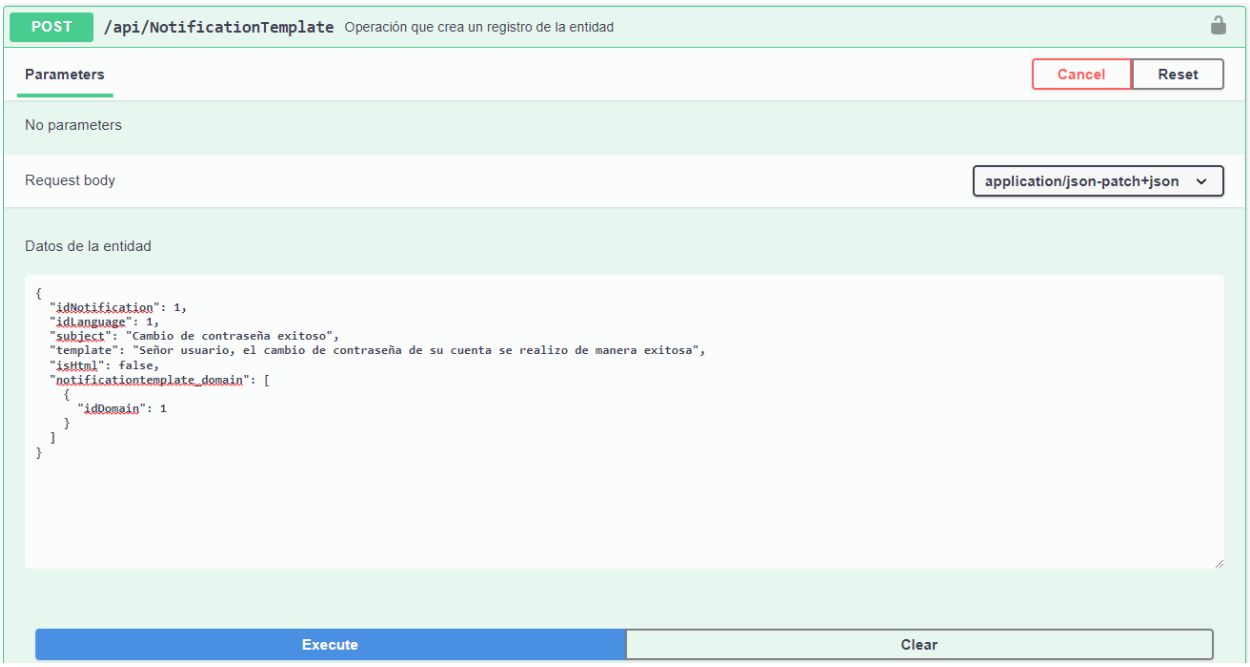

Fuente: el autor.

**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestran los servicios implementados para crear (Post), modificar (Put), eliminar (Delete) y obtener (Get) objetos de plantillas de notificación por dominio, según las reglas de negocio, relacionados al desarrollo del caso de uso de Administración de Plantillas de Notificaciones.

Las plantillas de notificación se usarán en notificaciones que se enviarán mediante correo electrónico, mensaje de texto o notificación push a los usuarios básicos y administradores cuando se cumplan ciertas acciones, por ejemplo: un usuario básico intenta acceder al sistema en horarios no permitidos.

**Figura 13.**

*Configurando una plantilla de notificación a crear desde Swagger UI*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se envía la petición y se guarda la plantilla de notificación por dominio en base de datos.

**Figura 14.**

*Respuesta que muestra Swagger UI al crearse la plantilla de notificación*

Server response

Code | Details

200 *Undocumented*

Response body


```
{
  "data": {
    "idNotification": 1,
    "idLanguage": 1,
    "subject": "Cambio de contraseña exitoso",
    "template": "Señor usuario, el cambio de contraseña de su cuenta se realizo de manera exitosa",
    "isHtml": false,
    "creatingDate": "2022-04-13T08:16:49.7370353-05:00",
    "notificationtemplate_domain": [
      {
        "idNotificationTemplate": 41,
        "idDomain": 1,
        "id": 3
      }
    ],
    "id": 41
  }
}
```


Download

Response headers

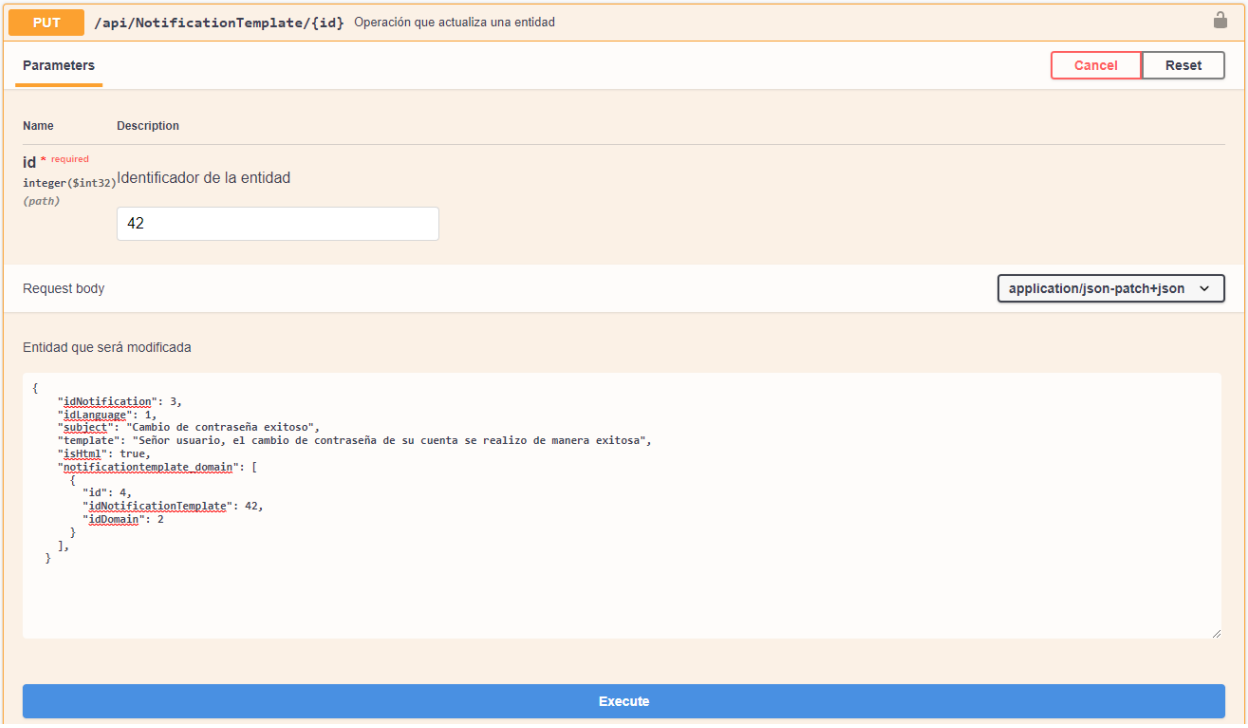
```
content-encoding: gzip
content-length: 367
content-type: application/json; charset=utf-8
date: Wed,13 Apr 2022 13:59:55 GMT
server: Microsoft-IIS/10.0
vary: Accept-Encoding
x-powered-by: ASP.NET
```

Fuente: el autor.

**Descripción:** Desde el back-end se retorna la plantilla de notificación por dominio creada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 15.**

*Configurando una plantilla de notificación a modificar desde Swagger UI*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se indica el identificador del registro a modificar, se envía la petición y se actualiza la plantilla de notificación por dominio en base de datos.

**Figura 16.**

*Respuesta que muestra Swagger UI al modificarse la plantilla de notificación*

Server response

Code | Details

200

Response body


```
{
  "data": {
    "idNotification": 3,
    "idLanguage": 1,
    "subject": "Cambio de contraseña exitoso",
    "template": "Señor usuario, el cambio de contraseña de su cuenta se realizo de manera exitosa",
    "isHtml": true,
    "creatingDate": "2022-04-13T08:55:43.543",
    "updatingDate": "2022-04-13T08:59:54.8310506-05:00",
    "notificationtemplate_domain": [
      {
        "idNotificationTemplate": 42,
        "idDomain": 2,
        "id": 4
      }
    ],
    "id": 42
  }
}
```


Download

Response headers

```
content-encoding: gzip
content-length: 367
content-type: application/json; charset=utf-8
date: Wed,13 Apr 2022 13:59:55 GMT
server: Microsoft-IIS/10.0
vary: Accept-Encoding
x-powered-by: ASP.NET
```

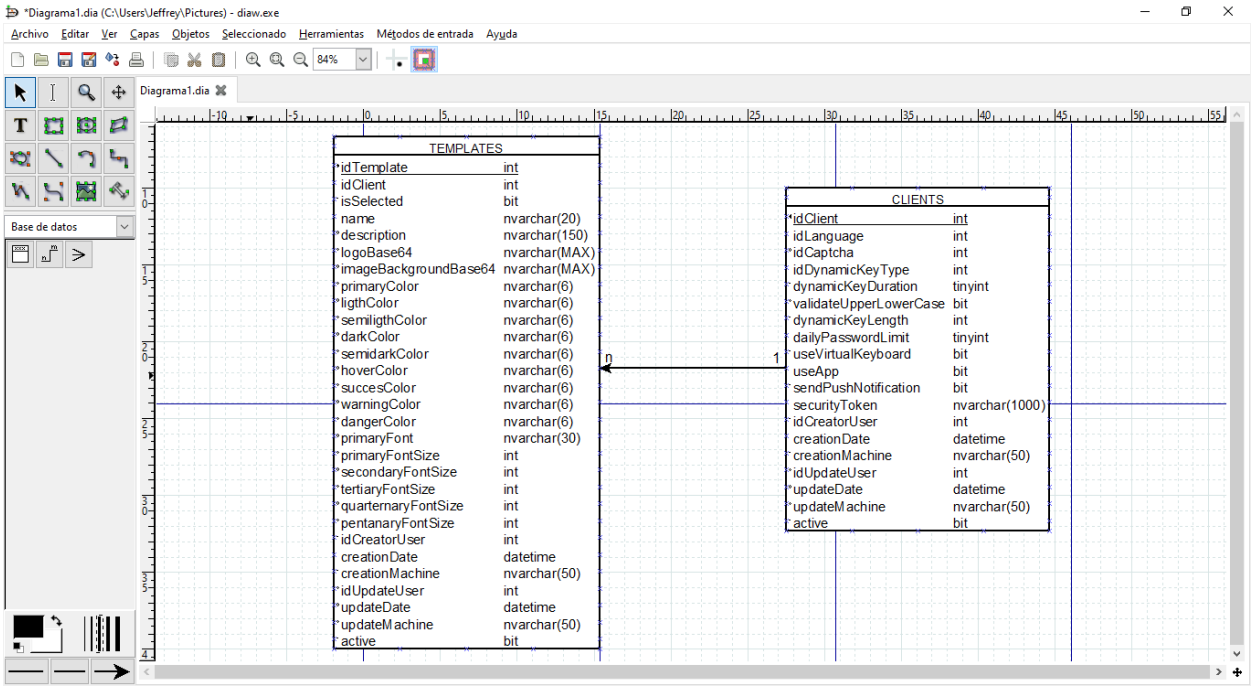
Fuente: el autor.

**Descripción:** Desde el back-end se retorna la plantilla de notificación por dominio modificada correctamente en base de datos, mostrándose en Swagger UI.

- Análisis y diseño de bases de datos en motor SQL Server

**Figura 17.**

*Diseño de la tabla Templates y relación con la tabla Clients*



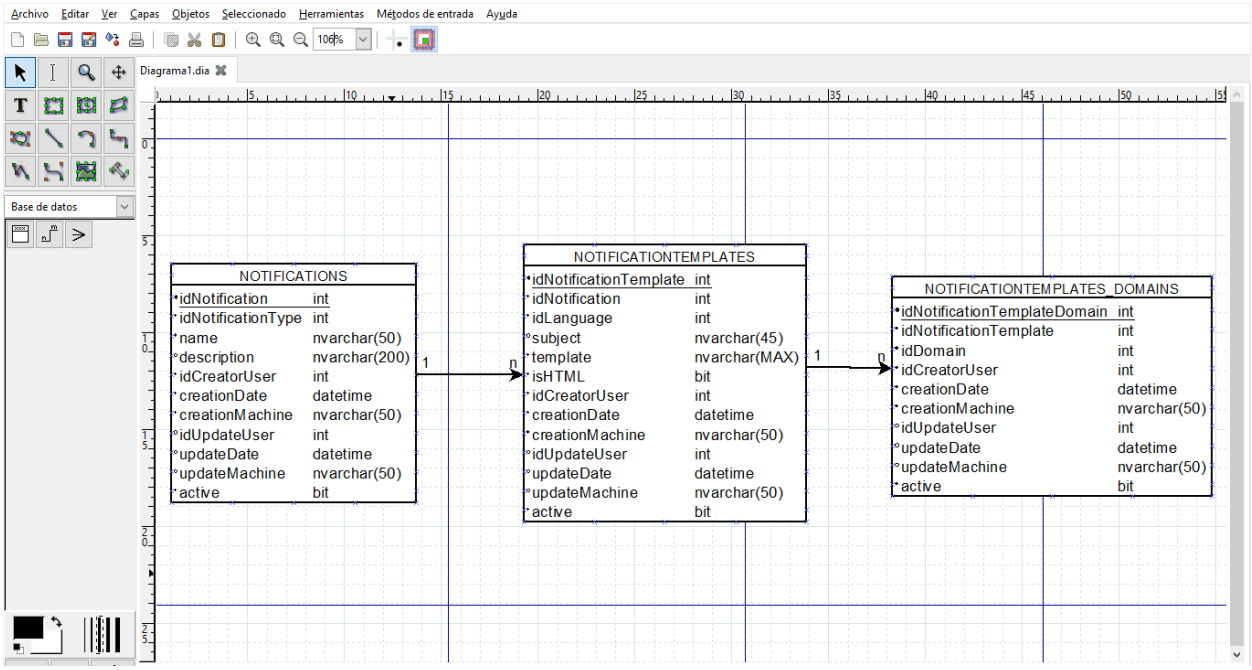

Fuente: el autor.

**Descripción:** Se hizo el modelado de la tabla Templates con cada uno de sus atributos y tipo de dato, se añadió la relación de un (1) Cliente a muchas (n) plantillas.

Fue necesario para el desarrollo back-end del caso de uso de Personalización BSR.

**Figura 18.**

*Diseño de las tablas NotificationTemplate y NotificationTemplates_Domains*




Fuente: el autor.

**Descripción:** Se hizo el modelado de la tabla NotificationTemplate y NotificationTemplates_Domains con cada uno de sus atributos y tipo de dato, se añadió la relación de una (1) notificación a muchas (n) plantillas de notificación y la relación de una (1) plantilla de notificación a muchas (n) plantillas de notificación por dominio.

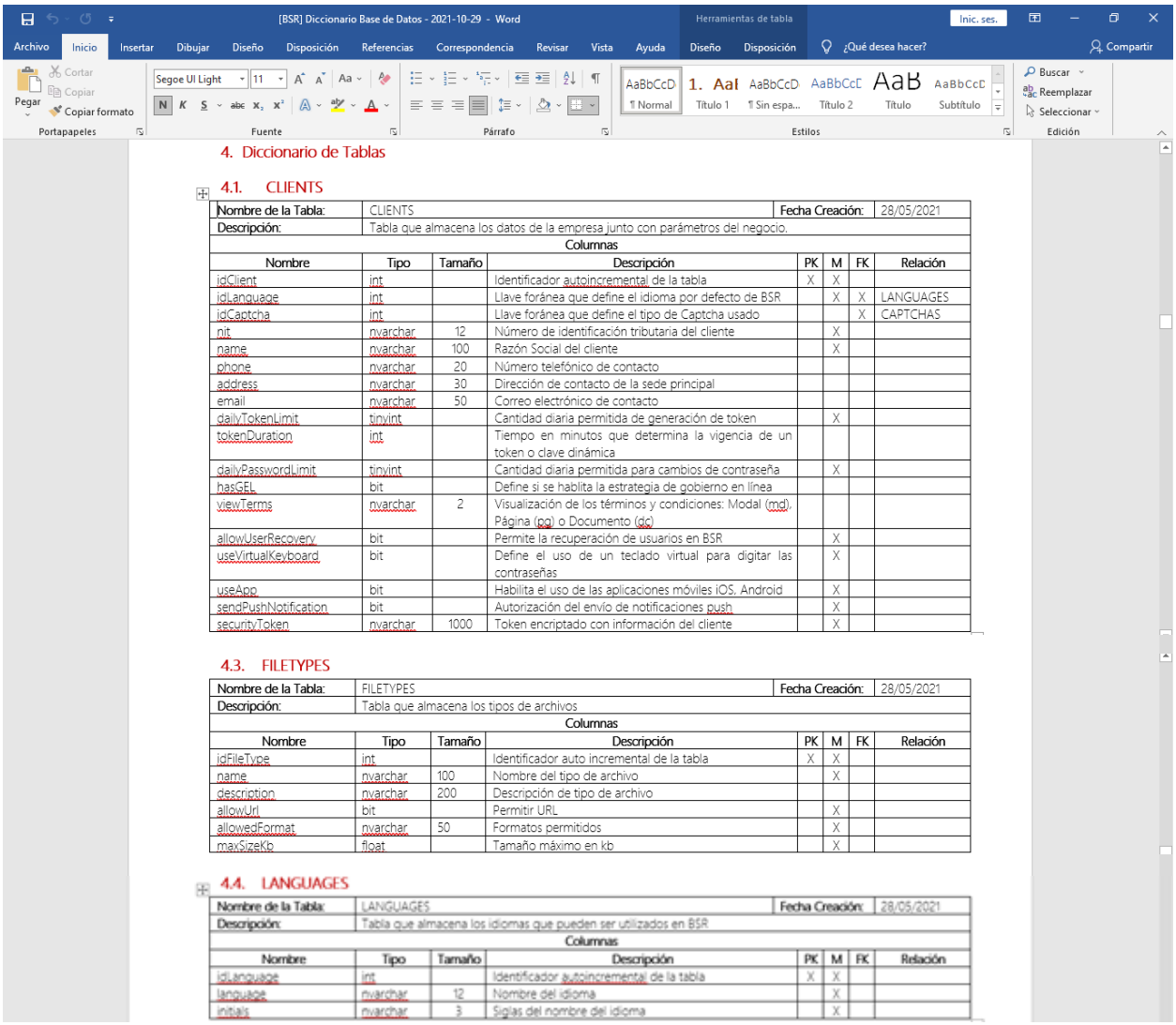
Fue necesario para el desarrollo back-end del caso de uso de Administración Plantillas de Notificaciones.

- Actualización del documento ‘[BSR] Diccionario Base de Datos.docx’

El documento “[BSR] Diccionario Base de Datos” contiene el detalle de todas las tablas implementadas en la base de datos.

**Figura 19.**

*Actualización de las tablas Clients, FileTypes y Languages*



4. Diccionario de Tablas

4.1. CLIENTS

| Nombre de la Tabla: | CLIENTS | | | | | | Fecha Creación: | 28/05/2021 |
|---|---|---|---|---|---|---|---|---|
| Descripción: | Tabla que almacena los datos de la empresa junto con parámetros del negocio. | | | | | | | |
| Columnas | | | | | | | | |
| Nombre | Tipo | Tamaño | Descripción | PK | M | FK | Relación | |
| idClient | int | | Identificador autoincremental de la tabla | X | X | | | |
| idLanguage | int | | Llave foránea que define el idioma por defecto de BSR | | X | X | LANGUAGES | |
| idCaptcha | int | | Llave foránea que define el tipo de Captcha usado | | | X | CAPTCHAS | |
| nit | nvarchar | 12 | Número de identificación tributaria del cliente | | X | | | |
| name | nvarchar | 100 | Razón Social del cliente | | X | | | |
| phone | nvarchar | 20 | Número telefónico de contacto | | | | | |
| address | nvarchar | 30 | Dirección de contacto de la sede principal | | | | | |
| email | nvarchar | 50 | Correo electrónico de contacto | | | | | |
| dailyTokenLimit | tinyint | | Cantidad diaria permitida de generación de token | | X | | | |
| tokenDuration | int | | Tiempo en minutos que determina la vigencia de un token o clave dinámica | | | | | |
| dailyPasswordLimit | tinyint | | Cantidad diaria permitida para cambios de contraseña | | X | | | |
| hasGEL | bit | | Define si se habilita la estrategia de gobierno en línea | | | | | |
| viewTerms | nvarchar | 2 | Visualización de los términos y condiciones: Modal (md), Página (pg) o Documento (dc) | | | | | |
| allowUserRecovery | bit | | Permite la recuperación de usuarios en BSR | | X | | | |
| useVirtualKeyboard | bit | | Define el uso de un teclado virtual para digitar las contraseñas | | X | | | |
| useApp | bit | | Habilita el uso de las aplicaciones móviles iOS, Android | | X | | | |
| sendPushNotification | bit | | Autorización del envío de notificaciones push | | X | | | |
| securityToken | nvarchar | 1000 | Token encriptado con información del cliente | | X | | | |

4.3. FILETYPES

| Nombre de la Tabla: | FILETYPES | | | | | | Fecha Creación: | 28/05/2021 |
|---|---|---|---|---|---|---|---|---|
| Descripción: | Tabla que almacena los tipos de archivos | | | | | | | |
| Columnas | | | | | | | | |
| Nombre | Tipo | Tamaño | Descripción | PK | M | FK | Relación | |
| idFileType | int | | Identificador auto incremental de la tabla | X | X | | | |
| name | nvarchar | 100 | Nombre del tipo de archivo | | X | | | |
| description | nvarchar | 200 | Descripción de tipo de archivo | | | | | |
| allowUrl | bit | | Permitir URL | | X | | | |
| allowedFormat | nvarchar | 50 | Formatos permitidos | | X | | | |
| maxSizeKb | float | | Tamaño máximo en kb | | X | | | |

4.4. LANGUAGES

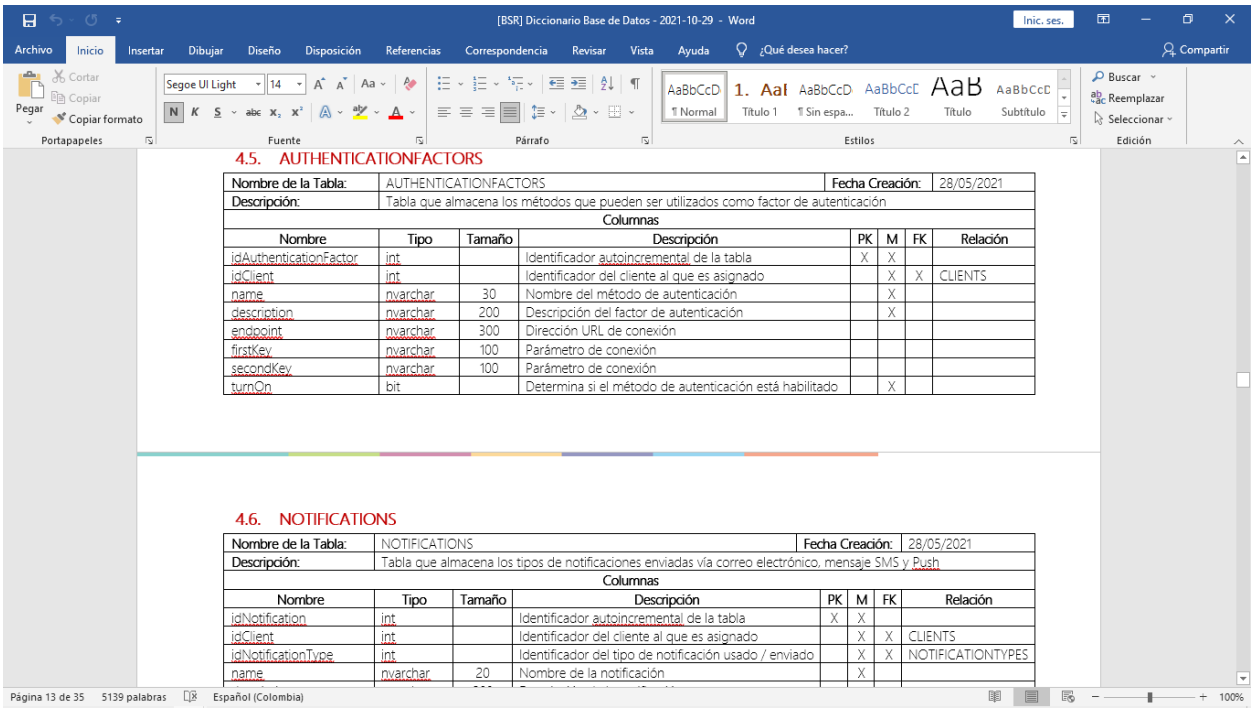
| Nombre de la Tabla: | LANGUAGES | | | | | | Fecha Creación: | 28/05/2021 |
|---|---|---|---|---|---|---|---|---|
| Descripción: | Tabla que almacena los idiomas que pueden ser utilizados en BSR | | | | | | | |
| Columnas | | | | | | | | |
| Nombre | Tipo | Tamaño | Descripción | PK | M | FK | Relación | |
| idLanguage | int | | Identificador autoincremental de la tabla | X | X | | | |
| language | nvarchar | 12 | Nombre del idioma | | X | | | |
| initials | nvarchar | 3 | Siglas del nombre del idioma | | X | | | |

Fuente: el autor.

**Descripción:** Se revisaron las tablas Clients, FileTypes y Languages en base de datos, añadiendo nuevos atributos, tipo de dato, descripción, relaciones y si el valor es opcional (nulo).

**Figura 20.**

*Actualización de las tablas AuthenticationFactors y Notifications*



4.5. AUTHENTICATIONFACTORS

| Nombre de la Tabla: | AUTHENTICATIONFACTORS | | | Fecha Creación: | 28/05/2021 | | |
|---|---|---|---|---|---|---|---|
| Descripción: | Tabla que almacena los métodos que pueden ser utilizados como factor de autenticación | | | | | | |
| Columnas | | | | | | | |
| Nombre | Tipo | Tamaño | Descripción | PK | M | FK | Relación |
| idAuthenticationFactor | int | | Identificador autoincremental de la tabla | X | X | | |
| idClient | int | | Identificador del cliente al que es asignado | | X | X | CLIENTS |
| name | nvarchar | 30 | Nombre del método de autenticación | | X | | |
| description | nvarchar | 200 | Descripción del factor de autenticación | | X | | |
| endpoint | nvarchar | 300 | Dirección URL de conexión | | | | |
| firstKey | nvarchar | 100 | Parámetro de conexión | | | | |
| secondKey | nvarchar | 100 | Parámetro de conexión | | | | |
| turnOn | bit | | Determina si el método de autenticación está habilitado | | X | | |

4.6. NOTIFICATIONS

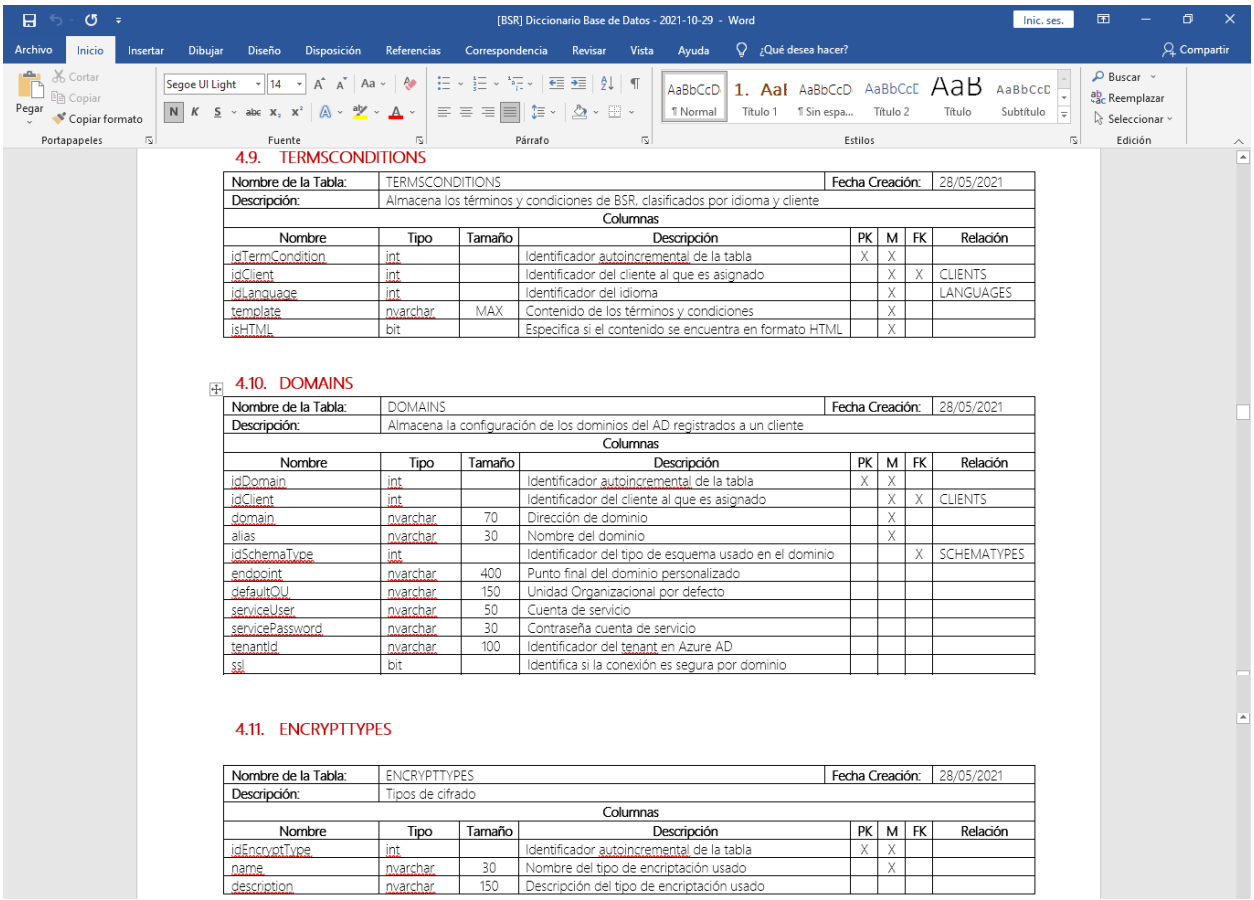
| Nombre de la Tabla: | NOTIFICATIONS | | | Fecha Creación: | 28/05/2021 | | |
|---|---|---|---|---|---|---|---|
| Descripción: | Tabla que almacena los tipos de notificaciones enviadas vía correo electrónico, mensaje SMS y Push | | | | | | |
| Columnas | | | | | | | |
| Nombre | Tipo | Tamaño | Descripción | PK | M | FK | Relación |
| idNotification | int | | Identificador autoincremental de la tabla | X | X | | |
| idClient | int | | Identificador del cliente al que es asignado | | X | X | CLIENTS |
| idNotificationType | int | | Identificador del tipo de notificación usado / enviado | | X | X | NOTIFICATIONTYPES |
| name | nvarchar | 20 | Nombre de la notificación | | X | | |

Fuente: el autor.

**Descripción:** Se revisaron las tablas *AuthenticationFactors y Notifications* en base de datos, añadiendo al documento nuevos atributos faltantes, tipo de dato, descripción, relaciones y si el valor es obligatorio (no nulo).

**Figura 21**

*Actualización de las tablas TermsConditions, Domains y EncryptTypes*



4.9. TERMSCONDITIONS

| Nombre de la Tabla: | TERMSCONDITIONS | | | | | | Fecha Creación: | 28/05/2021 |
|---|---|---|---|---|---|---|---|---|
| Descripción: | Almacena los términos y condiciones de BSR, clasificados por idioma y cliente | | | | | | | |
| Columnas | | | | | | | | |
| Nombre | Tipo | Tamaño | Descripción | PK | M | FK | Relación | |
| idTermCondition | int | | Identificador autoincremental de la tabla | X | X | | | |
| idClient | int | | Identificador del cliente al que es asignado | | X | X | CLIENTS | |
| idLanguage | int | | Identificador del idioma | | X | | LANGUAGES | |
| template | nvarchar | MAX | Contenido de los términos y condiciones | | X | | | |
| isHTML | bit | | Especifica si el contenido se encuentra en formato HTML | | X | | | |

4.10. DOMAINS

| Nombre de la Tabla: | DOMAINS | | | | | | Fecha Creación: | 28/05/2021 |
|---|---|---|---|---|---|---|---|---|
| Descripción: | Almacena la configuración de los dominios del AD registrados a un cliente | | | | | | | |
| Columnas | | | | | | | | |
| Nombre | Tipo | Tamaño | Descripción | PK | M | FK | Relación | |
| idDomain | int | | Identificador autoincremental de la tabla | X | X | | | |
| idClient | int | | Identificador del cliente al que es asignado | | X | X | CLIENTS | |
| domain | nvarchar | 70 | Dirección de dominio | | X | | | |
| alias | nvarchar | 30 | Nombre del dominio | | X | | | |
| idSchemaType | int | | Identificador del tipo de esquema usado en el dominio | | | X | SCHEMATYPES | |
| endpoint | nvarchar | 400 | Punto final del dominio personalizado | | | | | |
| defaultOU | nvarchar | 150 | Unidad Organizacional por defecto | | | | | |
| serviceUser | nvarchar | 50 | Cuenta de servicio | | | | | |
| servicePassword | nvarchar | 30 | Contraseña cuenta de servicio | | | | | |
| tenantId | nvarchar | 100 | Identificador del tenant en Azure AD | | | | | |
| ssl | bit | | Identifica si la conexión es segura por dominio | | | | | |

4.11. ENCRYPTTYPES

| Nombre de la Tabla: | ENCRYPTTYPES | | | | | | Fecha Creación: | 28/05/2021 |
|---|---|---|---|---|---|---|---|---|
| Descripción: | Tipos de cifrado | | | | | | | |
| Columnas | | | | | | | | |
| Nombre | Tipo | Tamaño | Descripción | PK | M | FK | Relación | |
| idEncryptType | int | | Identificador autoincremental de la tabla | X | X | | | |
| name | nvarchar | 30 | Nombre del tipo de encriptación usado | | X | | | |
| description | nvarchar | 150 | Descripción del tipo de encriptación usado | | | | | |

Fuente: el autor.

**Descripción:** Se revisaron las tablas *TermsConditions, Domains y EncryptTypes* en base de datos, añadiendo al documento nuevos atributos faltantes, tipo de dato, descripción, relaciones y si el valor es obligatorio (no nulo).

- Documentación de casos de uso

Se realizó la documentación de un caso de uso que describe la funcionalidad y comportamiento del sistema ante la interacción de algunos usuarios y actores principales.

**Figura 22.**

*Flujo normal – Agregar dominio*

1. Alcance

El presente documento describe el proceso de administración de dominios, definiendo los flujos, reglas que deben ser desarrollados para realizar las operaciones de la administración y configuración de dominios de la aplicación para el módulo de administración de BSR.

2. Caso de uso

| | |
|---|---|
| Nombre del caso de uso: | CU_17 Administración de dominio |
| Actores: | • Usuario Administrador: Usuario interno de BSR con rol de Administrador.<br>• Sistema: Producto Bext Self Reset que provee las funcionalidades descritas en la arquitectura.<br>• Usuario con permisos asignados en el sistema. |
| Descripción: | Este caso de uso indica la Configuración y Administración Dominios. |
| FLUJO NORMAL – AGREGAR DOMINIO | |
| Precondiciones: | • Se requiere que el usuario exista en el sistema (Proveedor de identidad) con permisos de Administración en BSR.<br>• Se debe tener un usuario super administrador para realizar las primeras configuraciones de la herramienta.<br>• La plataforma debe solicitar configurar al menos un cliente y un dominio de forma automática cuando se identifique que no se tiene dominios configurados en BSR. |
| Paso | Acción |
| 1 | El usuario ingresa a BSR (Ver: CU 101 _Inicio de sesión) |
| 2 | El usuario da clic en el botón **"Administración"** |
| 3 | El sistema carga en pantalla la Administración y sus diferentes módulos |
| 4 | El usuario da clic en el módulo de **"Administración de Dominio"** |
| 5 | El sistema carga la información de los dominios previamente configurados. |
| 6 | El usuario da clic en el botón **"Nuevo Dominio"** |
| 7 | El sistema solicita seleccionar un tipo de dominio a configurar (lista desplegable):<br>• LDAP – Genérico.<br>• Web servicie.<br>• Azure AD |

Fuente: el autor.

**Descripción:** Describe todos los pasos que un usuario administrador debe realizar en orden para agregar una nueva configuración de dominio en el sistema y reglas de negocio.

**Figura 23.**

*Flujo normal – Editar dominios*

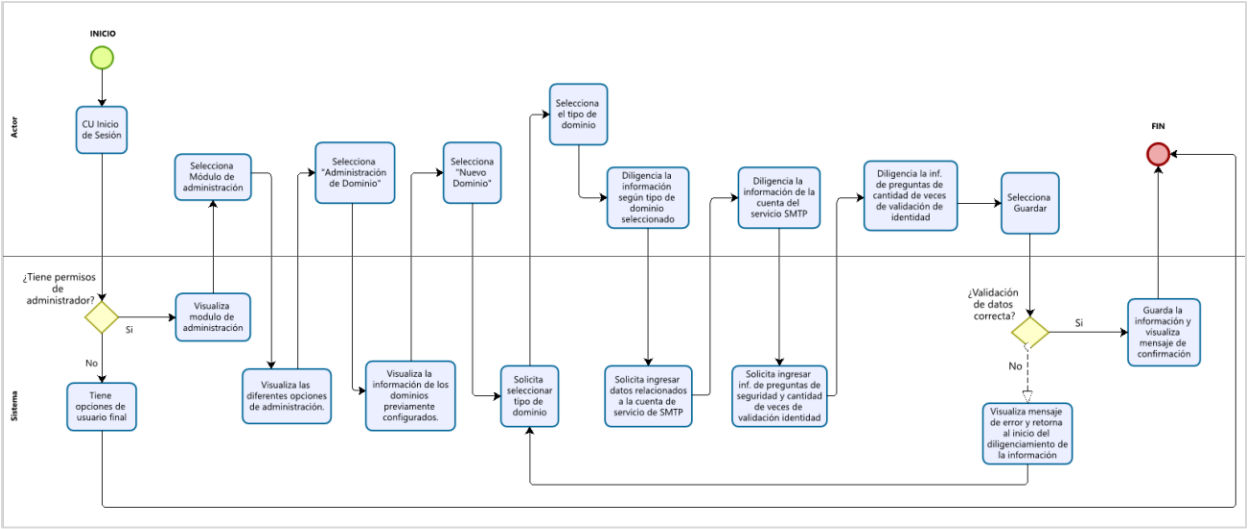
| FLUJO NORMAL – EDITAR DOMINIOS | |
|---|---|
| **Precondiciones:** | El sistema deberá tener registrado previamente mínimo un dominio. |
| Paso | Acción |
| 1 | El usuario ingresa a BSR (Ver: CU 101_Inicio de sesión). |
| 2 | El usuario dará clic en el botón **"Administración"**. |
| 3 | El sistema carga en pantalla la Administración y los diferentes módulos. |
| 4 | El usuario da clic en el módulo de **"Administración de Dominio".** |
| 5 | El sistema carga la información de los N dominios previamente configurados. |
| 6 | El usuario selecciona el dominio a editar. |
| 7 | El usuario da clic en el botón **"Editar".** |
| 8 | El sistema visualiza:<br>• Información del dominio seleccionado.<br>• Información de la **cuenta de servicio SMTP**<br>• Información referente a preguntas de seguridad y cantidad máxima permitida para validación de identidad. |
| 7 | El usuario edita y actualiza los campos que considere necesarios propios del dominio seleccionado, datos de la cuenta de servicio SMTP, información relacionada con las preguntas de seguridad y la cantidad máxima permitida para la validación de identidad de un usuario. |
| 8 | El usuario da clic en **"Actualizar".** |
| 9 | El sistema valida los datos ingresados:<br>• Si son válidos, el sistema actualiza la información en pantalla y en la base de datos, guarda cambios y visualiza mensaje de confirmación de la actualización.<br>• Si no son válidos, el sistema visualiza mensaje de error y retorna al paso 8 del flujo básico. |
| **Postcondición:** | El sistema visualizará el dominio que ha sido editado. |
| **Casos de uso relacionados:** | CU 101_Inicio de sesión; CU Configuración de cliente; CU28 Configuración de Desafíos. |
| **Reglas de Negocio:** | Para cualquier navegación se debe generar un tipo de modificación en la tabla de modificación y se debe ver reflejado en los reportes según el usuario que hizo la modificación. |

Fuente: el autor.

**Descripción:** Describe todos los pasos que un usuario administrador debe realizar en orden para editar una configuración de dominio en el sistema, incluyendo reglas de negocio a tener en cuenta para el desarrollo back-end y front-end.

**Figura 24.**

*Diagrama BPMN – Agregar dominio*



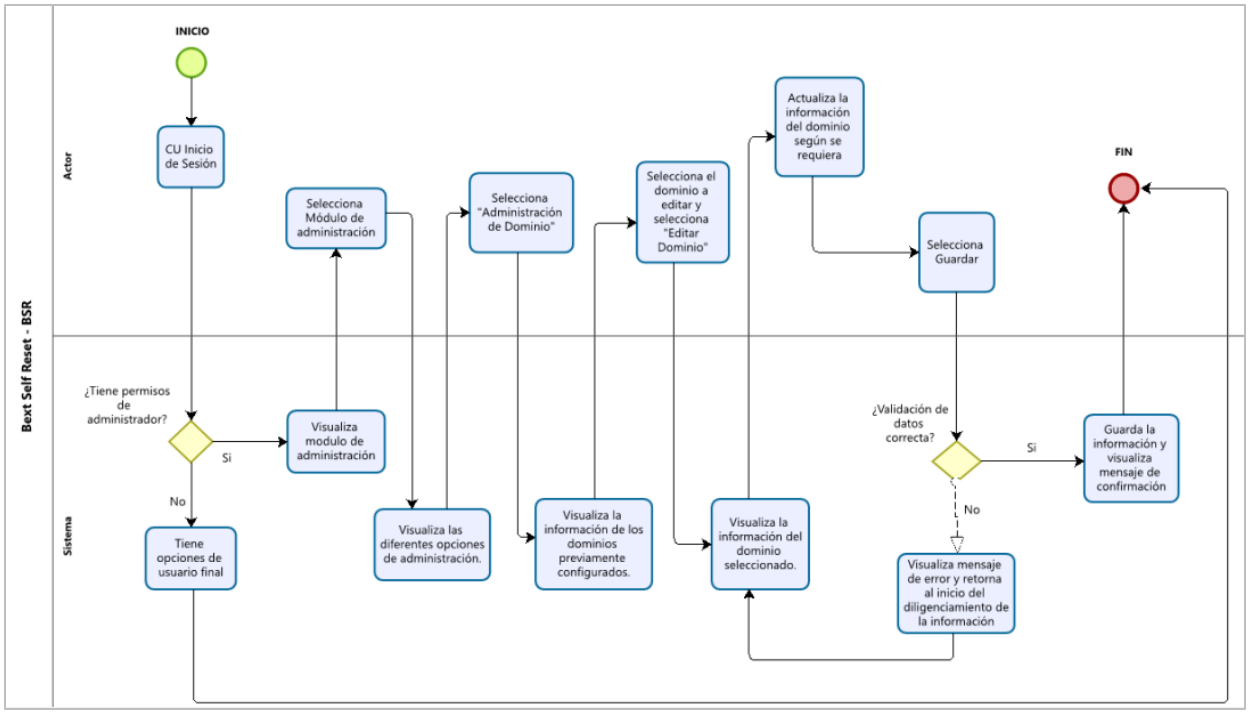

Fuente: el autor.

**Descripción:** Un diagrama BPMN es un estándar para modelar procesos de negocio empresariales. Mediante un diagrama de flujo se modelan procesos de inicio a fin con el objetivo de representar flujos de trabajo de manera sencilla y entendible para todos los involucrados en el negocio.

En este diagrama BPMN se representan los diferentes procesos de negocio a realizar de manera ordenada para agregar una nueva configuración de dominio desde el sistema “Bext Self Reset”.

**Figura 25.**

*Diagrama BPMN – Editar dominio*




Fuente: el autor.

**Descripción:** Un diagrama BPMN es un estándar para modelar procesos de negocio empresariales. Mediante un diagrama de flujo se modelan procesos de inicio a fin con el objetivo de representar flujos de trabajo de manera sencilla y entendible para todos los involucrados en el negocio.

En este diagrama BPMN se representan los diferentes procesos de negocio a realizar de manera ordenada para editar una configuración existente de dominio desde el sistema "Bext Self Reset".

**2.** En el segundo mes (6 de noviembre del 2021 a 5 de diciembre del 2021) se hizo:

- El estudio y desarrollo del caso de uso [CU29] Categoría Tipos de Archivo, añadiendo las reglas de negocio y funcionalidades necesarias en el código para permitir que un usuario administrador del sistema pueda gestionar la configuración de tipos de archivos permitidos para el módulo/pantalla de carga de manuales y documentos, especificando el tamaño, formato y nombre del tipo de documento (Manual de usuario final, manual de administración, otro, etc.). El administrador podrá definir archivos de tipo .pdf, o definir si es un archivo al que se accede mediante un link/enlace.
- El modelado de la tabla FileTypes, con cada uno de sus campos/atributos y la relación que tendrá con la tabla Files, en el software Día.
- El estudio y desarrollo del caso de uso [CU27] Jornadas Laborales, añadiendo las reglas de negocio y funcionalidades necesarias en el código para permitir que un usuario administrador del sistema pueda gestionar la configuración de jornadas laborales que pudiesen existir al interior de una organización, siendo posible definir días laborales con sus horarios y días no laborales de acuerdo con las necesidades de la organización.
- El modelado de la tabla Calendars, Calendars_WeekDays y NonWorkingdays, con cada uno de sus campos/atributos y las relaciones que tendrá con las tablas relacionadas, en el software Día.
- La documentación del caso de uso [CC31] Administración de Esquemas.

A continuación, se comparten algunas vistas que detallan las actividades realizadas durante este mes:

- Desarrollo back-end de los casos de uso

**Figura 26.**

*Clase Entidad del desarrollo del caso de uso de Categoría de Tipos de Archivo*

```
using Bext.Identity.Solution.Domain.Entities;
using System.Collections.Generic;

namespace Bext.Identity.Solution.Core.Entities
{
    21 references
    public partial class FileType : BaseEntity
    {
        0 references
        public FileType()
        {
            Files = new HashSet<File>();
        }

        9 references
        public string Name { get; set; }
        1 reference
        public string Description { get; set; }
        1 reference
        public bool AllowUrl { get; set; }
        5 references
        public string? AllowedFormat { get; set; }
        2 references
        public float MaxSizeKb { get; set; }

        2 references
        public virtual ICollection<File> Files { get; set; }
    }
}
```

Fuente: el autor.

**Descripción:** Se muestran cada una de las propiedades requeridas de la clase FileType para el desarrollo del caso de configuración de categorías de Tipos de Archivo. Tiene propiedades para establecer el nombre y descripción del tipo de archivo, si el archivo se accede mediante URL, el formato permitido y el tamaño máximo en Kilobytes.

Esta clase FileType tiene una relación de uno a muchos archivos (Files).

**Figura 27.**

*Métodos Get() y Post(), en la clase de servicio FileTypeService*

```
3 references
public async Task<FileType> Get(int id)
{//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
    if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.DescargarDocumento).FirstOrD
        throw new UnauthorizedBusinessException("El usuario no puede cargar documentos");
    //
    FileType fileType = await _unitAdmin.fileTypeRepository.GetById(id);

    if (fileType == null)
        throw new NotFoundBusinessException("El tipo de archivo no existe");
    return fileType;
}

2 references
public async Task<FileType> Post(FileType entity)
{
    entity = SetAuditFields(entity);
    var exists = _unitAdmin.fileTypeRepository.ExistsByName(entity.Name);

    if (exists)
        throw new NotFoundBusinessException("Ya existe un tipo de archivo con el nombre '" + entity.Name + "'")

    await _unitAdmin.fileTypeRepository.Add(entity);
    await _unitAdmin.SaveChangesAsync();
    return entity;
}

2 references
```

Fuente: el autor.

**Descripción:** El método Get obtiene y retorna una categoría de tipo de archivo en base al identificador de llave primaria indicado.

El método Post() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta creación de categorías de tipos de archivo en el sistema. Se valida que no queden categorías de tipos de archivo repetidas por nombre en la base de datos.

**Figura 28.**

*Método Put() para modificar, en la clase de servicio FileTypeService*

```
2 references
public async Task<FileType> Put(FileType entity)
{
    entity = SetAuditFields(entity);
    var record = await _unitAdmin.fileTypeRepository.GetById(entity.Id);

    if (record != null && record.Active == true)
    {
        if (entity.Id > 0)  // Valida que el Id sea mayor a 0
        {
            var exists = _unitAdmin.fileTypeRepository.ExistsByIdAndName(entity.Id, entity.Name);

            if (exists)
                throw new NotFoundBusinessException("Ya existe un tipo de archivo con el nombre '" + entity.Name +
        } else throw new NotFoundBusinessException("El 'Id' debe ser mayor a 0");
    } else throw new NotFoundBusinessException("No existe el 'Id' del tipo de archivo o no está activo");

    await _unitAdmin.fileTypeRepository.UpdateAsync(entity);
    await _unitAdmin.SaveChangesAsync();
    return entity;
}
```

Fuente: el autor.

**Descripción:** El método Put() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta modificación de categorías de tipos de archivos en el sistema. Se valida que la categoría de tipos de archivos a modificar exista y este activa en base de datos y que el nombre de la misma no se repita con otra configuración de tipo de archivo en base de datos.

**Figura 29.**

*Clase Modelo/Entidad del desarrollo del caso de uso de Jornadas laborares*

```
using Bext.Identity.Solution.Domain.Entities;
using System.Collections.Generic;

namespace Bext.Identity.Solution.Core.Entities
{
    30 references
    public class Calendar : BaseEntity
    {
        0 references
        public Calendar()
        {
            Roles = new HashSet<Role>();
            NonWorkingDay = new HashSet<NonWorkingDay>();
            Calendar_WeekDay = new HashSet<Calendar_WeekDay>();
        }

        7 references
        public int IdClient { get; set; }
        9 references
        public string Name { get; set; }

        1 reference
        public virtual Client IdClientNavigation { get; set; }
        2 references
        public virtual ICollection<Role> Roles { get; set; }
        14 references
        public virtual ICollection<NonWorkingDay> NonWorkingDay { get; set; }
        13 references
        public virtual ICollection<Calendar_WeekDay> Calendar_WeekDay { get; set; }
    }
}
```

Fuente: el autor.

**Descripción:** Se muestran cada una de las propiedades requeridas de la clase Calendar para el desarrollo del caso de configuración de jornadas laborales. Tiene propiedades para indicar el nombre del calendario y el cliente al cual se va a asociar.

Esta clase Calendar tiene una relación de uno a muchos roles (Role), una relación de uno a muchos días no laborales (NonWorkingDay), una relación de uno a muchos días laborales (Calendar_WeekDay).

**Figura 30.**

*Método Post() para crear, en la clase de servicio CalendarService*

```
71  public async Task<Calendar> Post(Calendar entity)
72  {
73      if (_unitAdmin.calendarRepository.GetAll().Where(x => x.Name.ToLower() == entity.Name.ToLower() && x.Active == t
74          throw new NotFoundBusinessException("Ya existe uno o más Calendarios con el mismo nombre: " + entity.Name);
75
76      entity = SetAuditFields(entity);
77      bool exists = false; int secondCounter = 0, thirdCounter, counterFour, weekDay = 0;
78      var idCalendars = new List<int> { }; var day = new List<int> { }; var start = new List<TimeSpan?> { }; var end =
79
80      foreach (NonWorkingDay value in entity.NonWorkingDay.OrderBy(x => x.NonWorkingDate))
81      {
82          if (!(non_WorkingDays.Contains(value.NonWorkingDate)))
83              non_WorkingDays.Add(value.NonWorkingDate);
84          else
85              throw new NotFoundBusinessException("Los días no laborales deben ser únicos, no repetidos");
86      }
87
88      // Se organiza ascendentemente los horarios, primero por hora inicial, luego por día y se verifica si hay máximo
89      foreach (Calendar_WeekDay record in entity.Calendar_WeekDay.OrderBy(x => x.StartTime).OrderBy(x => x.IdWeekDay))
90      {
91          exists = true;
92          if (record.IdWeekDay == weekDay)
93              secondCounter++;
94          else
95          {
96              weekDay = record.IdWeekDay;
97              secondCounter = 1;
98          }
99      if (secondCounter > 3)
100         throw new NotFoundBusinessException("Solamente se puede elegir como máximo tres horarios para cada día");
101
102     day.Add(record.IdWeekDay);
103     start.Add(record.StartTime);
104     end.Add(record.EndTime);
105 }
106
107 if (exists == false)
108     throw new NotFoundBusinessException("Se debe asignar por lo menos un día laboral y una franja horaria al horario
109
110 // Organizados los horarios se verifica si hay franjas horarias iguales por día y si se cruzan las franjas horarias
111 for (int dayCounter = 1; dayCounter < day.Count(); dayCounter++)
112 {
113     var firstDay = day.ElementAt(dayCounter - 1);
114     var firstInitialHour = start.ElementAt(dayCounter - 1);
115     var firstFinalHour = end.ElementAt(dayCounter - 1);
116     var secondDay = day.ElementAt(dayCounter);
117     var secondInitialHour = start.ElementAt(dayCounter);
118     var finalSecondHour = end.ElementAt(dayCounter);
119
120     bool isTimeBetween = firstDay == secondDay && secondInitialHour > firstInitialHour && secondInitialHour < firstF
121                          || firstDay == secondDay && secondInitialHour > firstInitialHour && secondInitialHour
122                          || firstDay == secondDay && secondInitialHour > firstInitialHour && secondInitialHour
123
124     if (firstDay == secondDay && firstInitialHour == secondInitialHour && firstFinalHour == finalSecondHour)
125         throw new NotFoundBusinessException("Franja horaria del dia repetida: 1) Dia: " + firstDay + "  Hora Inicial
126
```

Fuente: el autor.

**Descripción:** El método Post() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta creación de jornadas laborales en el sistema. Se valida que no queden calendarios repetidos por nombre en la base de datos, que no exista previamente un calendario igual al que se va a crear y franjas horarias correctas por días laborales.

**Figura 31.**

*Método Put() para modificar, en la clase de servicio CalendarService*

```
196     public async Task<Calendar> Put(Calendar entity)
197     {
198         if (entity.Id > 1)  // Valida que el Id sea mayor a 1
199         {
200             var record = await _unitAdmin.calendarRepository.GetById(entity.Id);
201             if (record != null && record.Active == true)  // Valida que el Id del Calendario exista y este activo (true
202             {
203                 if (_unitAdmin.calendarRepository.GetAll().Where(x => x.Id != entity.Id && x.Name.ToLower() == entity.N
204                     throw new NotFoundBusinessException("Ya existe uno o más Calendarios con el mismo nombre: " + entit
205
206                 entity = SetAuditFields(entity);
207                 int secondCounter = 0, thirdCounter, counterFour, weekDay = 0;
208                 var idCalendars = new List<int> { }; var day = new List<int> { }; var start = new List<TimeSpan?> { };
209
210                 foreach (NonWorkingDay value in entity.NonWorkingDay.OrderBy(x => x.NonWorkingDate))
211     {
212         NonWorkingDay exist = _unitAdmin.nonWorkingDayRepository.GetNonWorkingDay(entity.Id, value.NonWorkingDate);
213         if (exist != null && exist.NonWorkingDate == value.NonWorkingDate && exist.Id != value.Id && value.Id > 0)
214             throw new NotFoundBusinessException("El día no laboral '" + value.NonWorkingDate + "' ya se encuentra asigna
215
216         if (exist != null && exist.NonWorkingDate == value.NonWorkingDate && exist.Id != value.Id && value.Id == 0 || no
217             throw new NotFoundBusinessException("Los días no laborales deben ser únicos, no repetidos");
218         non_WorkingDays.Add(value.NonWorkingDate);
219     }
220
221     // Se verifica que los Id de los Calendar_WeekDays a modificar existan en la BD y los almacena en una lista
222     foreach (Calendar_WeekDay item in entity.Calendar_WeekDay.OrderBy(x => x.StartTime).OrderBy(x => x.IdWeekDay))
223     {
224         secondValues.Add(item);
225         if (item.Id > 0)
226         {
227             Calendar_WeekDay calendar_WeekDay = _unitAdmin.calendarWeekDayRepository.GetByCalendarWeekDay(item.Id, item.
228             if (calendar_WeekDay != null)
229                 thirdValues.Add(item.Id);
230             else
231                 throw new NotFoundBusinessException("No existe el Id " + item.Id + "a modificar");
232         }
233     }
234
235     // La nueva lista de registros a añadir, modificar y los de la BD por idCalendar diferentes a los que se van a modif
236     var records = _unitAdmin.calendarWeekDayRepository.GetAll().Where(x => x.IdCalendar == entity.Id && x.Active == true
237     foreach (Calendar_WeekDay item in records.OrderBy(x => x.StartTime).OrderBy(x => x.IdWeekDay))
238     {
239         if (!(thirdValues.Contains(item.Id)) && !(secondValues.Contains(item)))
240             secondValues.Add(item);
241     }
```

Fuente: el autor.

**Descripción:** El método Put() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta modificación de jornadas laborales en el sistema. Se valida que no queden calendarios repetidos por nombre en la base de datos, que no exista previamente un calendario igual al que se va a modificar, franjas horarias validas por días laborales y días no laborales. Las jornadas laborales definen los horarios de ingreso al sistema.

- Pruebas de los servicios implementados en desarrollo back-end

**Figura 32.**

*Servicios implementados para el desarrollo de Categoría Tipos de Archivo*

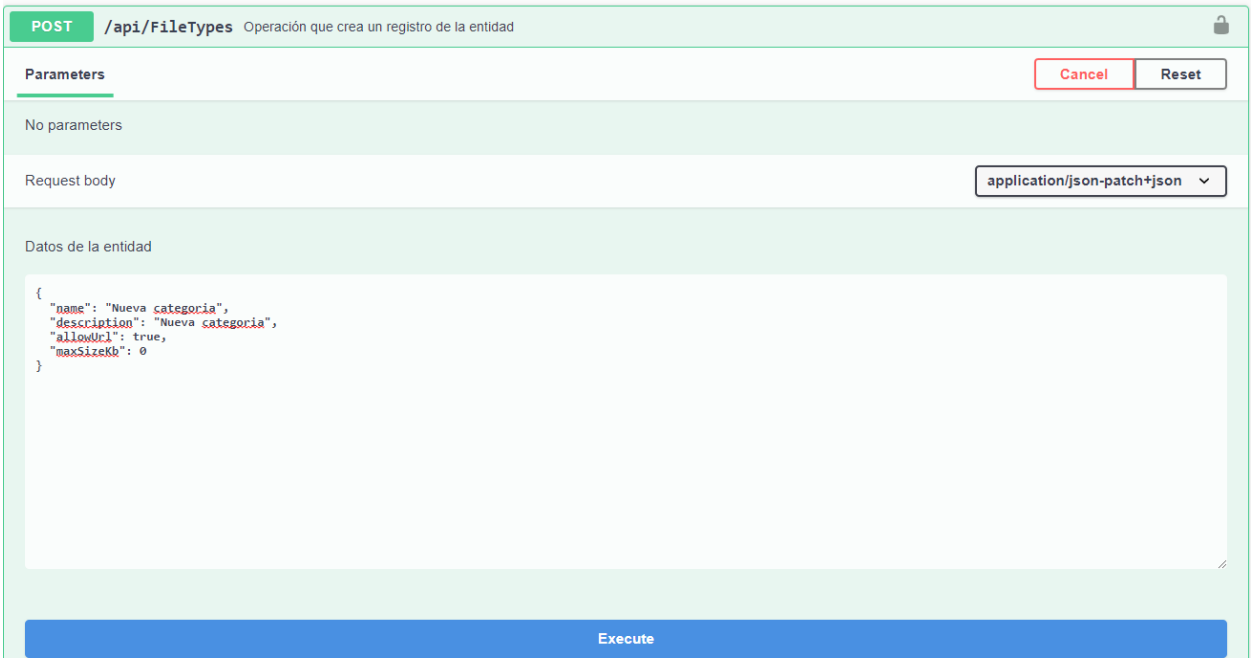
Fuente: el autor.

**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestran los servicios implementados para crear (Post), modificar (Put), eliminar (Delete) y obtener (Get) objetos de tipos de archivos, según las reglas de negocio, relacionados al desarrollo del caso de uso de Categorías de Tipos de Archivo.

Los tipos de archivo permitirán definir si se accede a los archivos mediante URL, el tamaño en KB, el formato permitido.

**Figura 33.**

*Configurando una categoría de tipos de archivo a crear desde Swagger UI*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se envía la petición y se guarda la categoría de tipos de archivo en base de datos.

**Figura 34.**

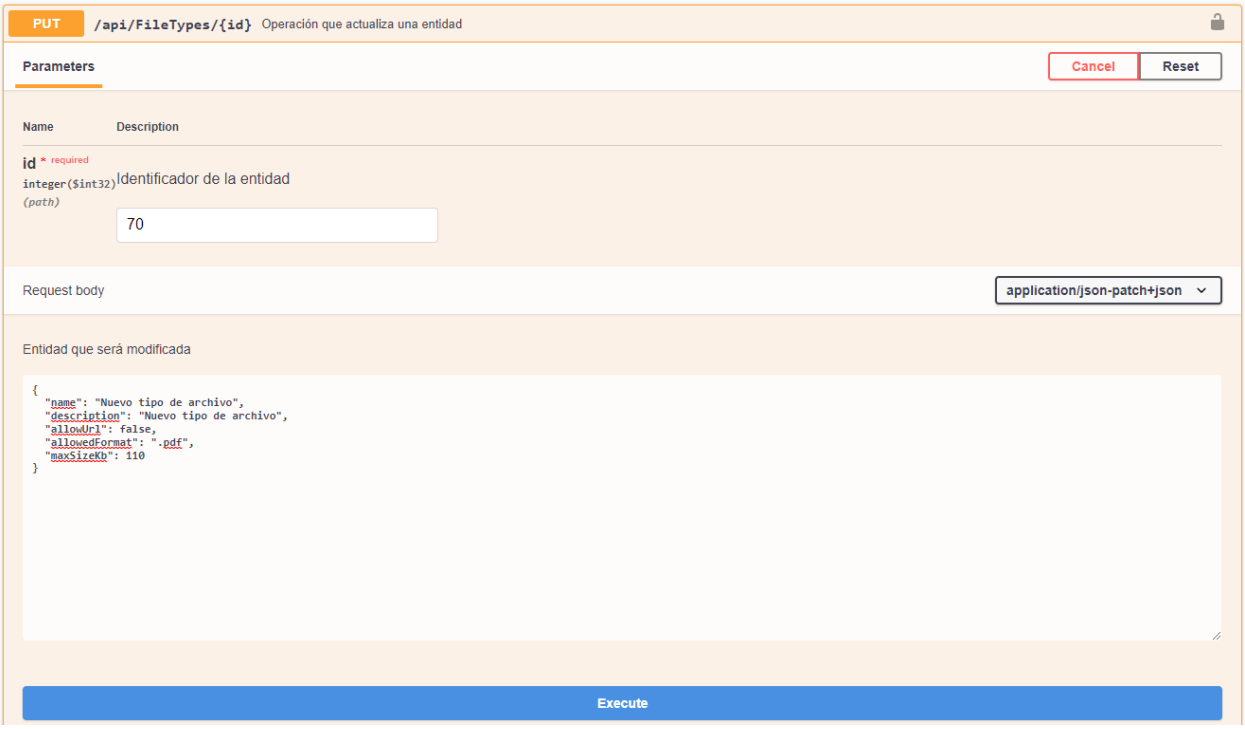
*Respuesta que muestra Swagger UI al crearse la categoría de tipos de archivo*

Fuente: el autor.

**Descripción:** Desde el back-end se retorna la categoría de tipos de archivo creada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 35.**

*Configurando una categoría de tipos de archivo a modificar desde Swagger UI*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se indica el identificador del registro a modificar, se envía la petición y se actualiza la categoría de tipos de archivo en base de datos.

**Figura 36.**

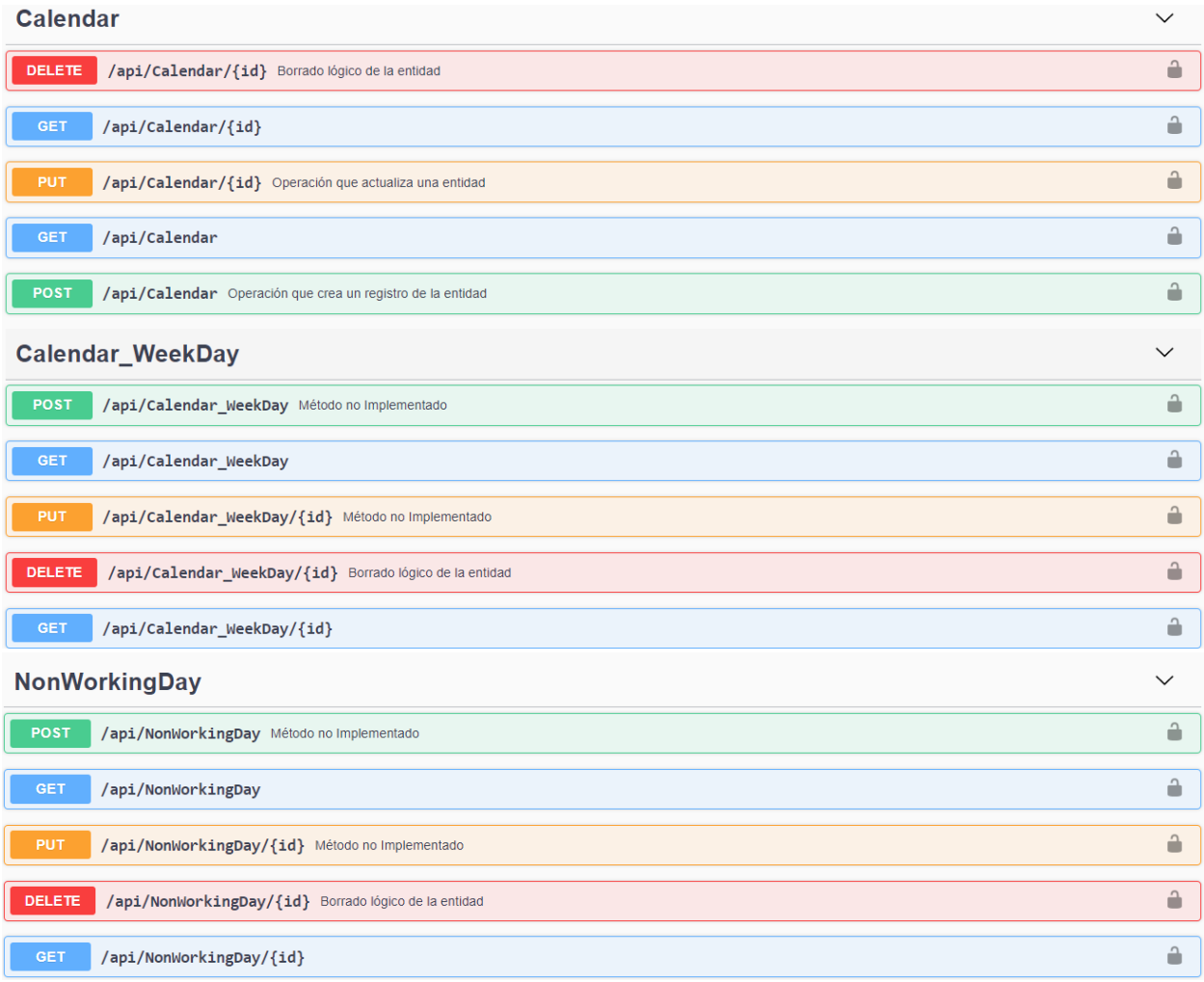
*Respuesta que muestra Swagger UI al modificar la categoría de tipos de archivo*

Fuente: el autor.

**Descripción:** Desde el back-end se retorna la categoría de tipos de archivo modificada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 37.**

*Servicios implementados para el desarrollo de Jornadas Laborales*



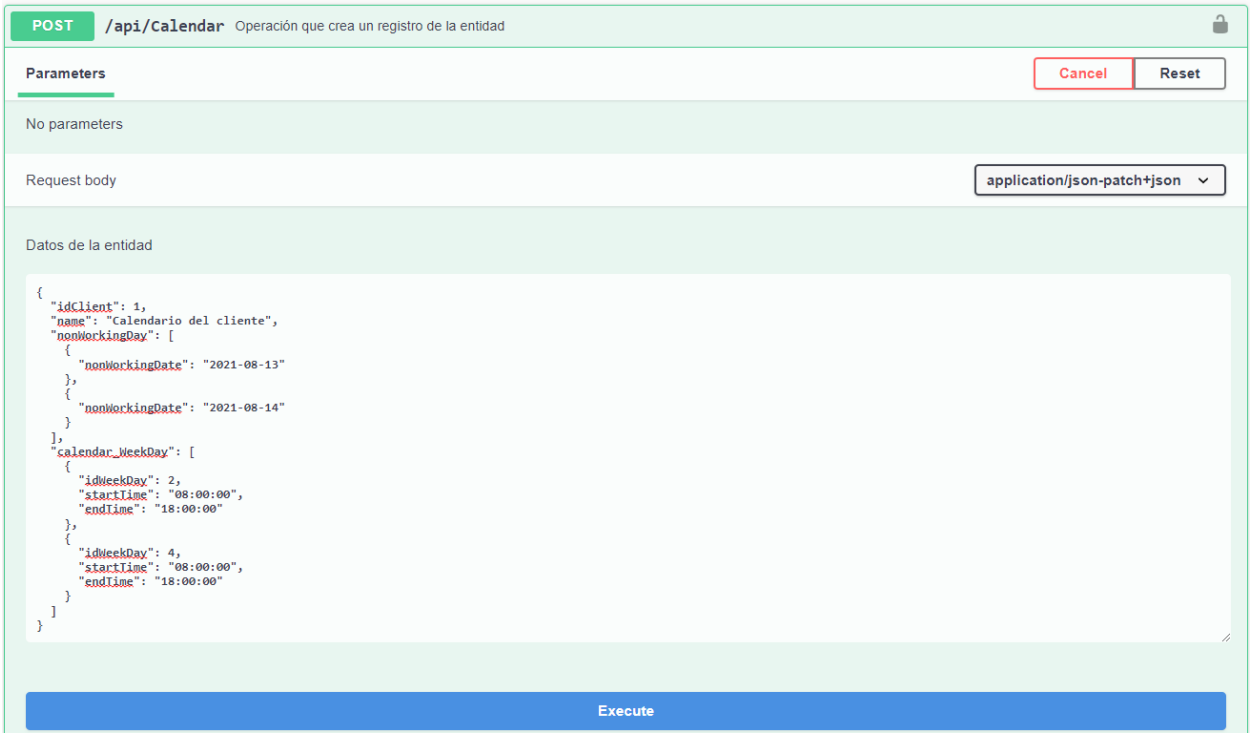

Fuente: el autor.

**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestran los servicios implementados para crear (Post), modificar (Put), eliminar (Delete) y obtener (Get) objetos de calendarios, según las reglas de negocio, relacionados al desarrollo del caso de uso de Jornadas Laborales.

Se definirán días no laborales y días laborales con franjas laborales en los calendarios.

**Figura 38.**

*Configurando una jornada laboral a crear desde Swagger UI*



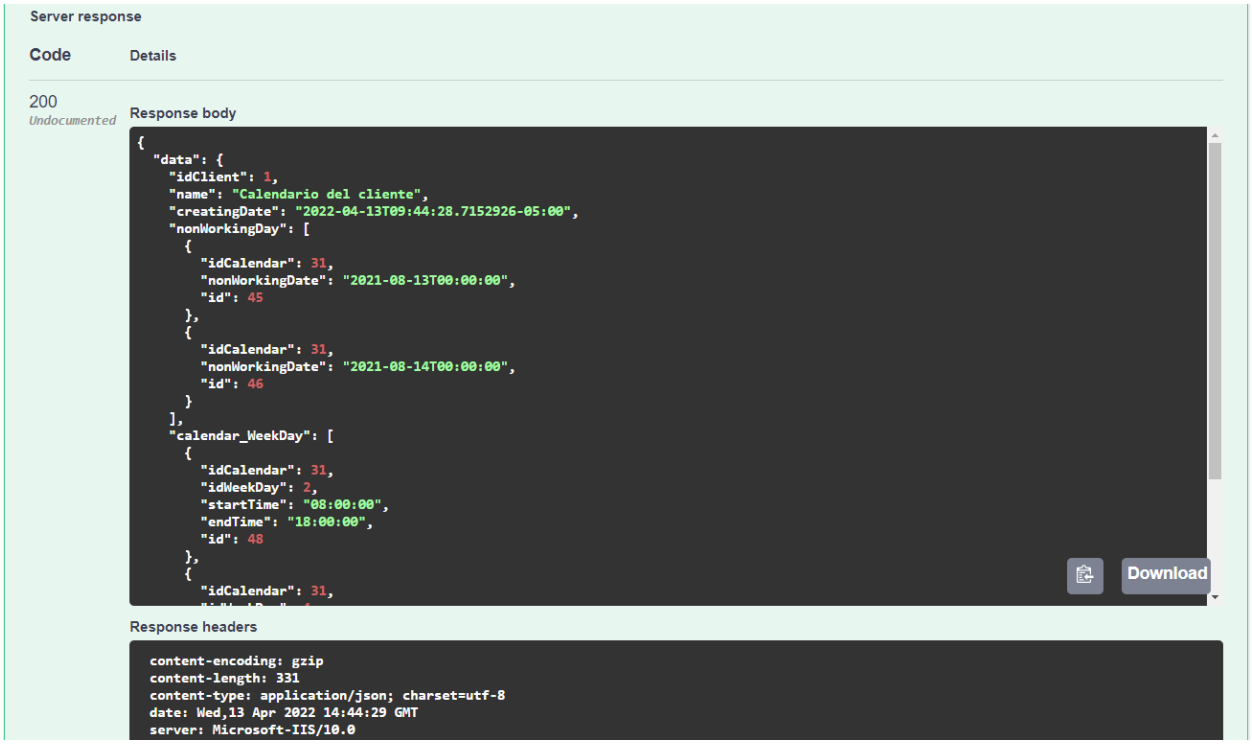

Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se envía la petición y se guarda la jornada laboral en base de datos. Se permiten ingresar días no laborales y días laborales con sus franjas horarias.

Los calendarios definirán los días y horarios de ingreso al aplicativo BSR.

**Figura 39.**

*Respuesta que muestra Swagger UI al crearse la jornada laboral*

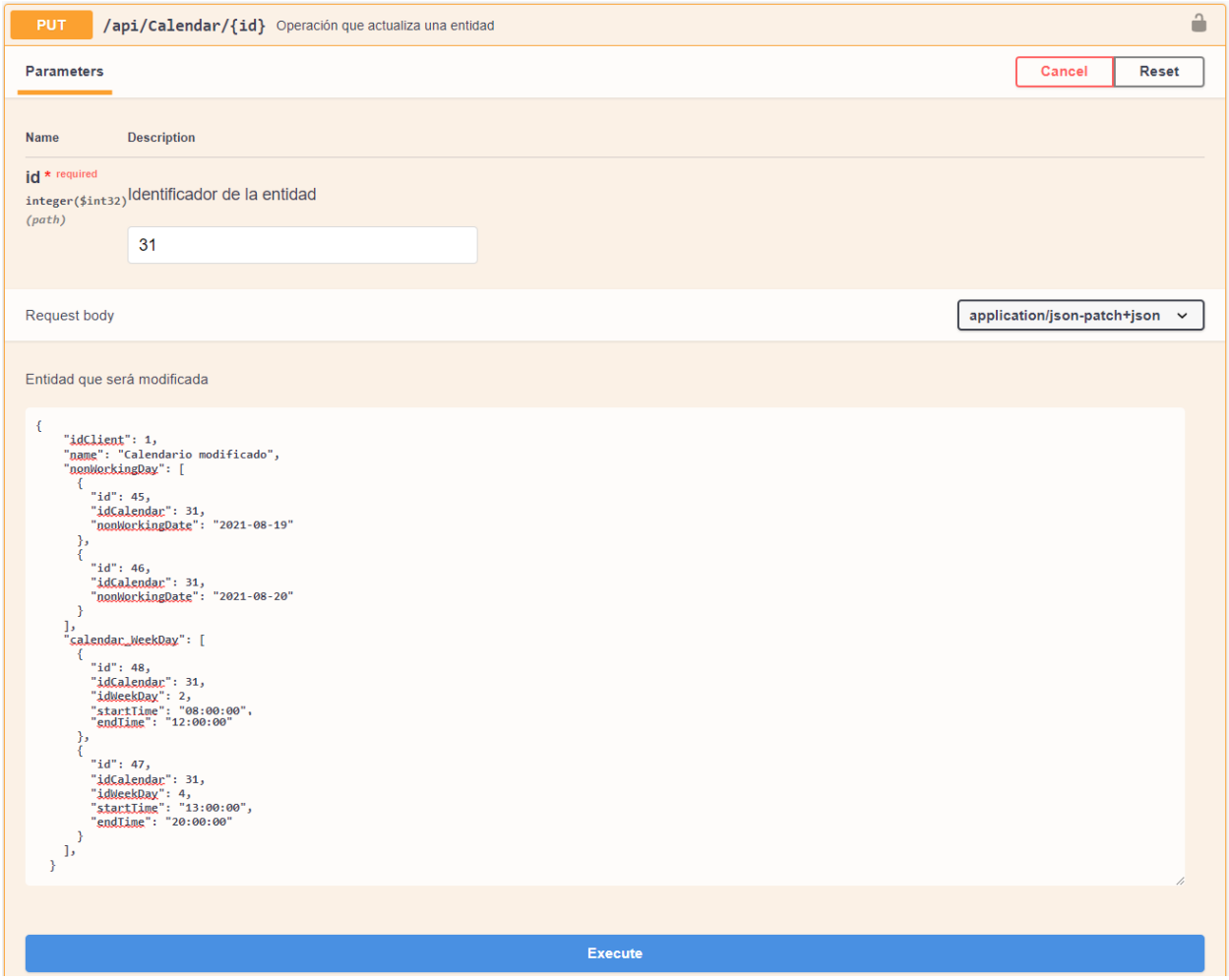



Fuente: el autor.

**Descripción:** Desde el back-end se retorna la configuración de jornada laboral creada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 40.**

*Configurando una jornada laboral a modificar desde Swagger UI*



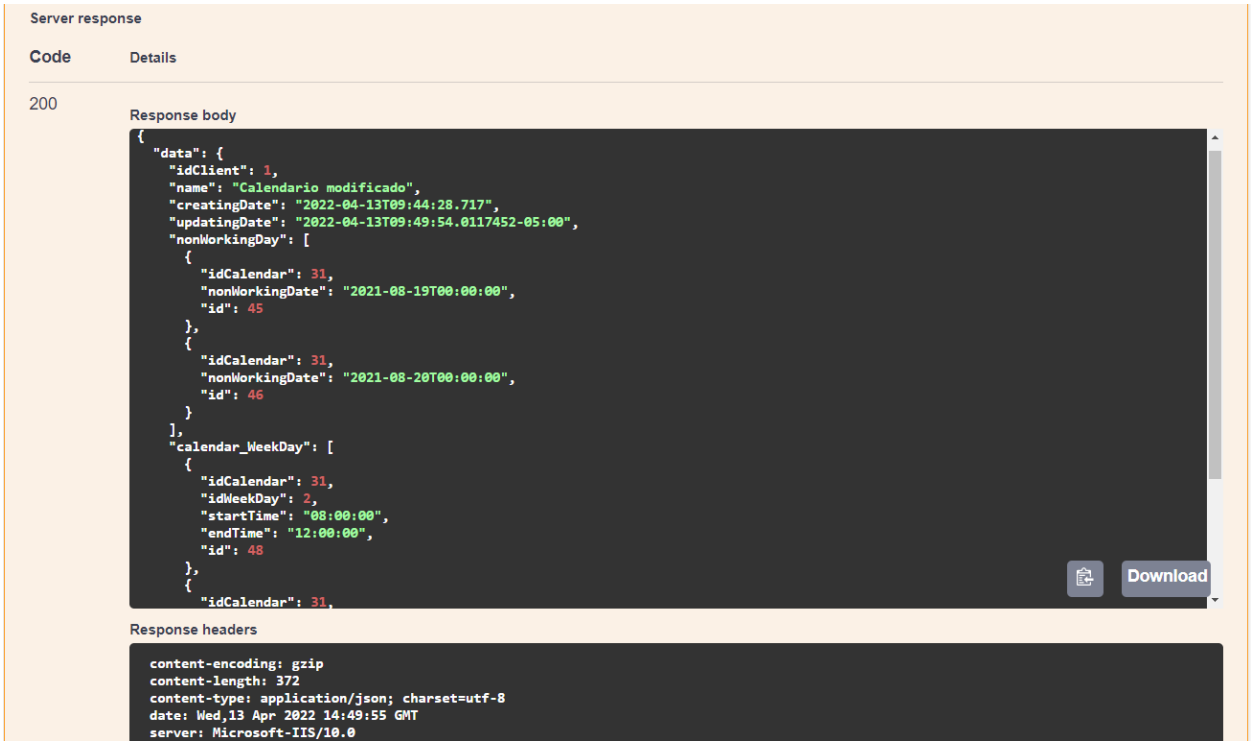

Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se indica el identificador del registro a modificar, se envía la petición y se actualiza la jornada laboral en base de datos. Se permiten ingresar días no laborales y días laborales con sus franjas horarias.

Los calendarios definirán los días y horarios de ingreso al aplicativo BSR.

**Figura 41.**

*Respuesta que muestra Swagger UI al modificarse la jornada laboral*



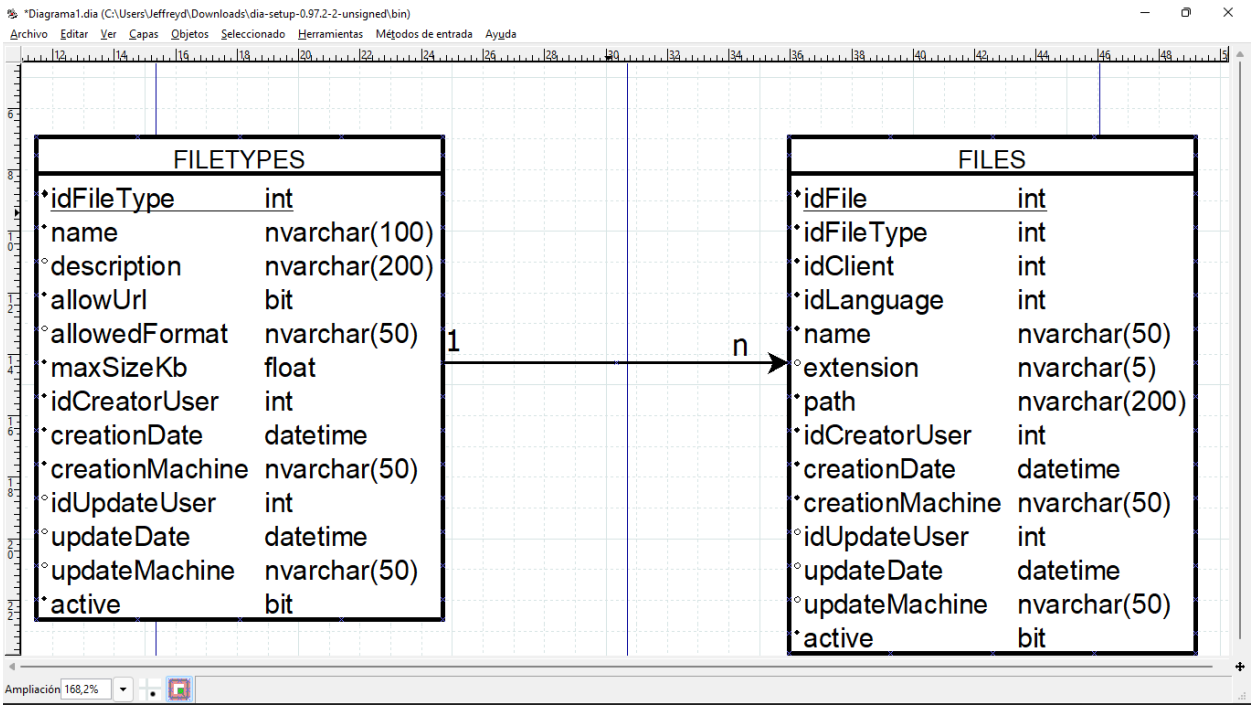

Fuente: el autor.

**Descripción:** Desde el back-end se retorna la configuración de jornada laboral modificada correctamente en base de datos, mostrándose en Swagger UI.

- Análisis y diseño de bases de datos en motor SQL Server

**Figura 42.**

*Diseño la tabla FileTypes y relación con la tabla Files*



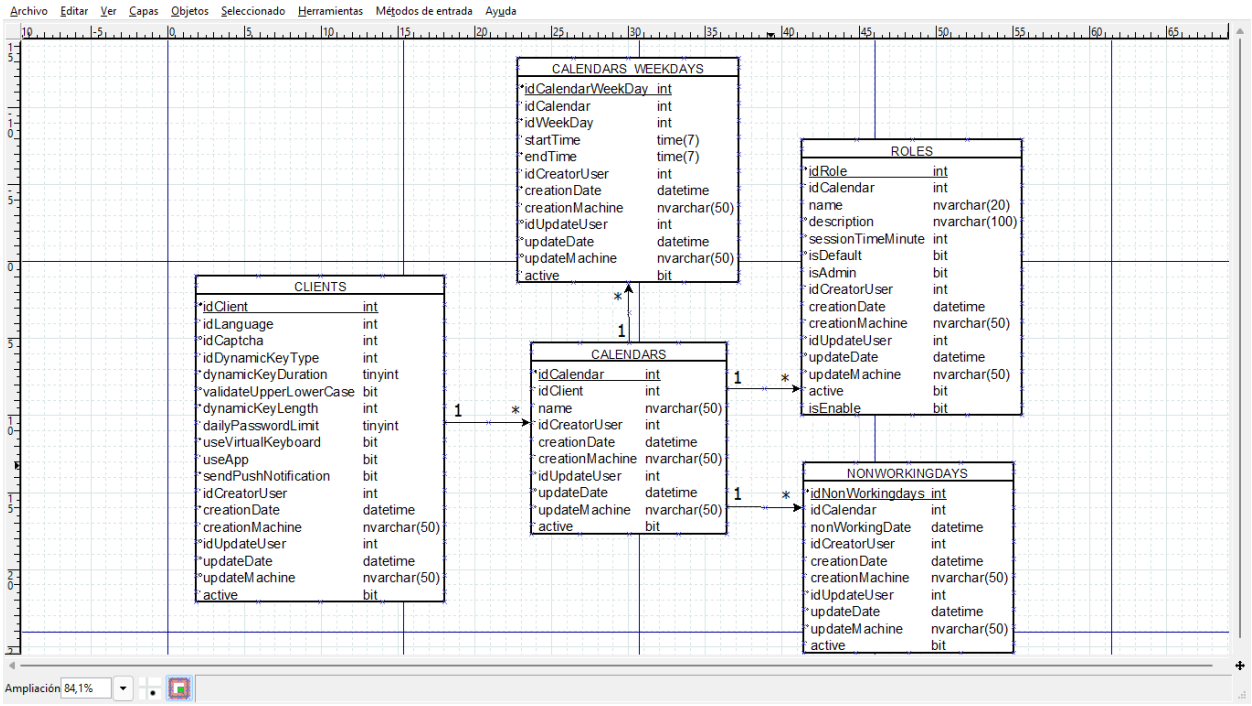

Fuente: el autor.

**Descripción:** Se hizo el modelado de la tabla *FileTypes*, con cada uno de sus atributos y tipo de dato, se añadió la relación de un (1) tipo de archivo a muchos (n) archivos.

Necesario para el desarrollo back-end del caso de uso de Categoría Tipos de Archivo.

**Figura 43.**

*Diseño de las tablas Calendars, Calendars_WeekDays y NonWorkingdays*




Fuente: el autor.

**Descripción:** Se hizo el modelado de las tablas Calendars, Calendars_WeekDays y NonWorkingdays, con cada uno de sus atributos y tipo de dato, se añadió la relación de un (1) cliente a muchos (n) calendarios, un (1) calendario a muchos (n) días de la semana por calendario, un (1) calendario a muchos (n) roles, un (1) calendario a muchos (n) días no laborales

Necesario para el desarrollo back-end del caso de uso de Jornadas Laborales.

- Documentación de casos de uso

**Figura 44.**

*Flujo normal – Agregar esquema*

## 2. Caso de uso

| | |
|---|---|
| Nombre del caso de uso: | [CU31] Administración de Esquema |
| Actores: | • Usuario Administrador: Usuario interno de BSR con rol de Administrador.<br>• Usuario con permisos asignados en el sistema<br>• Sistema: Producto BextSelfReset que provee las funcionalidades descritas en la arquitectura. |
| Descripción: | Este caso de uso indica la forma en que un usuario con permisos de administrador podrá configurar los esquemas usados en el aplicativo BSR. |
| FLUJO NORMAL – AGREGAR ESQUEMA | |
| Precondiciones: | • Se requiere que el usuario exista en el sistema (Proveedor de identidad) con permisos de Administración en BSR.<br>• Se requiere una configuración por defecto en el sistema. |
| Paso | Acción |
| 1 | El usuario inicia sesión en BSR (*Ver CU [101] Inicio de sesión BSR SSO*). |
| 2 | El sistema visualiza los módulos de BSR, entre ellos Administración. |
| 3 | El Administrador elige la opción de **"Administración"** |
| 4 | El sistema visualiza el menú de administración con diferentes módulos. |
| 5 | El usuario elige el módulo **"Administración de Esquema"** |
| 6 | El sistema visualiza los Esquemas previamente creados. |
| 7 | El usuario da clic en el botón **"Nuevo Esquema"** |
| 8 | El sistema solicita seleccionar un tipo de esquema a configurar:<br>• LDAP – Genérico.<br>• Web servicie.<br>• Azure AD |
| 9 | El usuario selecciona el tipo de esquema:<br>• **LDAP – Genérico**, continúa en el paso 8.1 del flujo alterno 1.<br>• **Web Service**, continúa en el paso 8.2 del flujo alterno 2.<br>• **Azure AD**, continúa en el paso 8.3 del flujo alterno 3. |

Fuente: el autor.

**Descripción:** Describe todos los pasos que un usuario administrador debe realizar en orden para agregar una nueva configuración de esquema en el sistema y reglas de negocio.

**Figura 45.**

*Flujo normal – Modificar esquema*

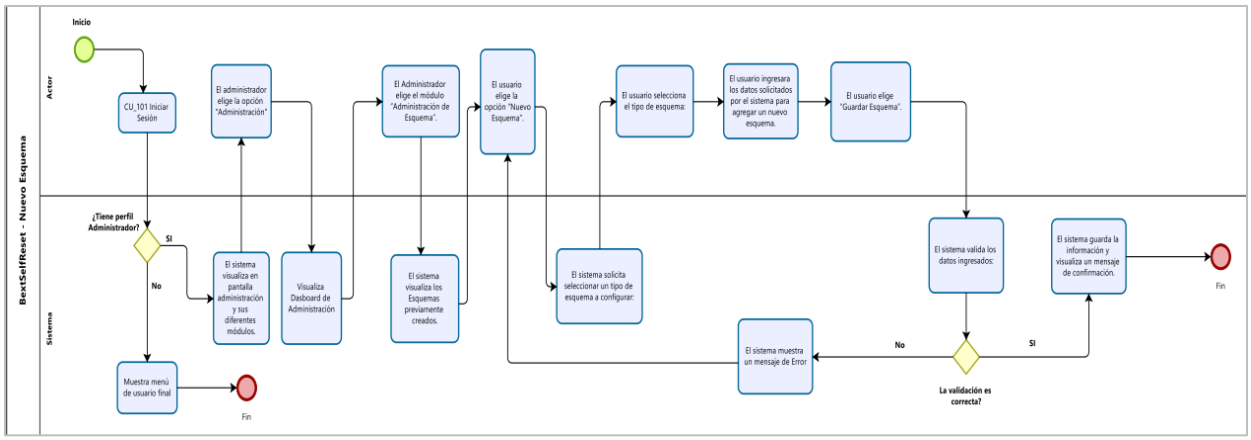
| | | | |
|---|---|---|---|
| | FLUJO NORMAL – MODIFICAR ESQUEMA | | |
| | Precondiciones: | • Se requiere que el usuario exista en el sistema (Proveedor de identidad) con permisos de Administración en BSR.<br>• Se requiere que exista al menos un esquema configurado o agregado previamente asociado a un dominio. | |
| | Paso | Acción | |
| | 1 | El usuario inicia sesión en BSR (*Ver CU [101] Inicio de sesión BSR SSO*). | |
| | 2 | El sistema visualiza los módulos de BSR, entre ellos Administración. | |
| | 3 | El Administrador elige la opción de **"Administración"** | |
| | 4 | El sistema visualiza el menú de administración con diferentes módulos. | |
| | 5 | El usuario elige el módulo **"Administración de Esquema"** | |
| | 6 | El sistema carga la información de los Esquemas previamente configurados. | |
| | 7 | El usuario selecciona el registro o ítem que contiene el Esquema a modificar. | |
| | 8 | El usuario selecciona **"Modificar Esquema"** | |
| | 9 | El sistema habilitará para edición los distintos controles HTML del formulario, permitiendo al usuario editar el Esquema. | |
| | 10 | El usuario editará la información correspondiente según la selección de esquema. | |
| | 11 | El usuario elige **"Guardar cambios".** | |
| | 12 | El sistema valida los datos ingresados:<br>• Si los datos son erróneos, el sistema visualiza un mensaje de error (Especificando el error presentado) y continúa en el paso 8.<br>• Si son correctos, el sistema continuo en el paso 13. | |
| | 13 | El sistema guarda la información y visualiza un mensaje de confirmación. | |

Fuente: el autor.

**Descripción:** Describe todos los pasos que un usuario administrador debe realizar en orden para editar una configuración de esquema en el sistema, incluyendo reglas de negocio a tener en cuenta para el desarrollo back-end y front-end.

**Figura 46.**

*Diagrama BPMN flujo básico – Nuevo esquema*



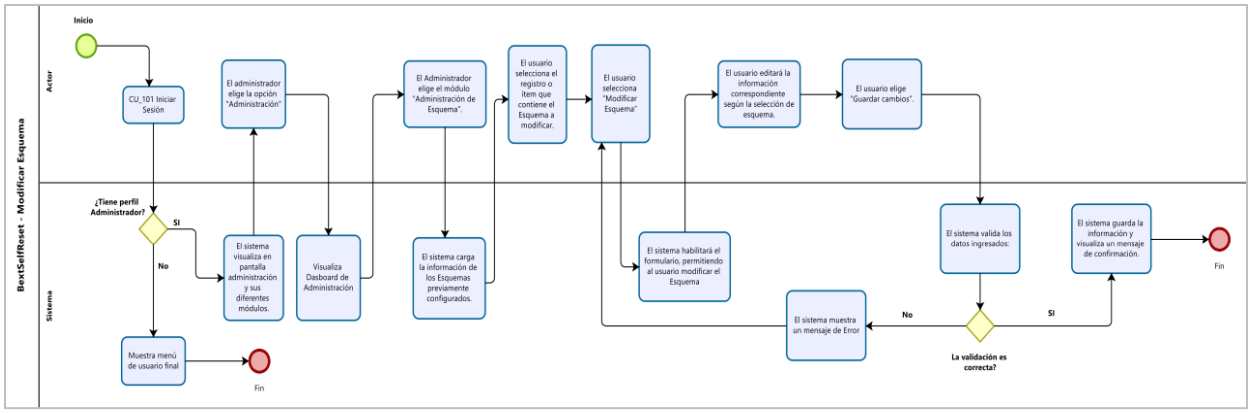

Fuente: el autor.

**Descripción:** Un diagrama BPMN es un estándar para modelar procesos de negocio empresariales. Mediante un diagrama de flujo se modelan procesos de inicio a fin con el objetivo de representar flujos de trabajo de manera sencilla y entendible para todos los involucrados en el negocio.

En este diagrama BPMN se representan los diferentes procesos de negocio a realizar de manera ordenada para agregar una nueva configuración de esquema desde el sistema "Bext Self Reset".

**Figura 47.**

*Diagrama BPMN flujo básico – Modificar esquema*




Fuente: el autor.

**Descripción:** Un diagrama BPMN es un estándar para modelar procesos de negocio empresariales. Mediante un diagrama de flujo se modelan procesos de inicio a fin con el objetivo de representar flujos de trabajo de manera sencilla y entendible para todos los involucrados en el negocio.

En este diagrama BPMN se representan los diferentes procesos de negocio a realizar de manera ordenada para editar una configuración existente de esquema desde el sistema "Bext Self Reset".

**3.** En el tercer mes (6 de diciembre del 2021 a 5 de enero del 2022) se hizo:

- El estudio y desarrollo del caso de uso [CU32] Asignación de Roles a Cuenta de Usuario, añadiendo las reglas de negocio y funcionalidades necesarias en el código para permitir que un usuario administrador del sistema pueda gestionar la configuración que permita conceder los permisos y privilegios para acceder a la plataforma BSR, de acuerdo a los roles asignados a los usuarios y/o dominios. El administrador podrá asignar uno o varios roles a un usuario o dominio específico. La asignación de roles permite que el usuario o el dominio al cual le son asignados, pueda hacer uso de los permisos y privilegios propios del rol.
- El modelado de la tabla Domains_Roles y User_Roles con cada uno de sus campos/atributos y la relación que tendrá con la tabla Users, Domains y Roles en el software Día.
- El estudio y desarrollo del caso de uso [CU35] Flujo de Notificaciones, añadiendo las reglas de negocio y funcionalidades necesarias en el código para permitir que un usuario administrador del sistema pueda gestionar la configuración que permita establecer los destinatarios que recibirán notificaciones de tipo email, SMS o push, ya sean usuarios específicos del sistema o un conjunto de usuarios que pertenezcan a uno o varios roles, junto con la notificación que será enviada, esto para que al momento en que un usuario ejecute ciertas acciones en el sistema, se pueda saber a qué destinatarios les será enviada la notificación.
- La adaptación y ejecución de algunas pruebas en el software Insomnia para consumir algunos servicios RESTful que permiten interactuar con la configuración del servidor de identidad WSO2.

- La configuración de un proveedor de servicios y un proveedor de identidad mediante la consola de administración del servidor de identidad WSO2.
- La adaptación y ejecución de algunas pruebas en el software Insomnia para la generación de tokens de acceso (JWT), con los cuales se pueden consumir los diferentes servicios Rest del proyecto “Bext Self Reset” en desarrollo.
- La documentación de casos de uso del software multiplataforma a desarrollar “Bext Self Reset” que se tenía planeado realiza, no se realizó, ya que esta actividad se completó en el mes anterior, reemplazándolo por desarrollo back-end.

A continuación, se comparten algunas vistas que detallan las actividades realizadas durante este mes:

- Desarrollo back-end de los casos de uso

**Figura 48.**

*Clase Entidad del desarrollo de Asignación de Roles a Cuenta de Usuario*

```
namespace Bext.Identity.Solution.Domain.Entities
{
    35 references
    public class Domain_Role : BaseEntity
    {
        21 references
        public int IdRole { get; set; }
        12 references
        public int IdDomain { get; set; }
        20 references
        public bool Enable { get; set; }

        15 references
        public virtual Role IdRoleNavigation { get; set; }
        2 references
        public virtual Domain IdDomainNavigation { get; set; }
    }
}
```

Fuente: el autor.

**Descripción:** Se muestran cada una de las propiedades requeridas y llaves foráneas de la clase Domain_Role para el desarrollo de asignación de roles por dominio. Tiene propiedades para indicar el identificador del rol y del dominio y si rol por dominio quedara habilitado.

**Figura 49.**

*Método Post() para crear, en la clase de servicio Domail_RoleService*

```
File Edit View Git Project Build Debug Team Test Analyze Tools Extensions Window Help Full Screen
Domain_RoleService.cs*
Bext.Identity.Solution.Core | Bext.Identity.Solution.Core.Services.Domain_RoleService | Get(Domain_RoleQueryFilter filters)
3 references
72   public async Task<Domain_Role> Post(Domain_Role entity)
73   {//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
74       if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Crear).FirstOrDefault() == null)
75           throw new UnauthorizedBusinessException("El usuario no puede agregar información");
76       //
77       entity = SetAuditFields(entity);
78       entity.Enable = true;
79
80       // Consulta el dominio y valida si esta activo
81       Domain.Entities.Domain domain = await _unitAdmin.domainRepository.GetById(entity.IdDomain);
82
83       if(domain == null)
84           throw new NotFoundBusinessException("El Dominio no existe o no se encuentra Activo");
85
86       // Valida que el rol exista, este activo y habilitado
87       if (!(_unitAdmin.roleRepository.Role_IsEnabled(entity.IdRole)))
88           throw new NotFoundBusinessException("El Rol no existe, no está Habilitado o no se encuentra Activo");
89
90       // Consulta y valida si ya existe el rol asociado al dominio en la BD
91       var domainRole = _unitAdmin.domainRoleRepository.GetDomainRole(entity.IdDomain, entity.IdRole);
92
93       if (domainRole != null)
94       {
95           if (domainRole.Enable == false && domainRole.IdRole == entity.IdRole)
96               throw new NotFoundBusinessException("El Dominio ya tiene asignado el rol seleccionado (Inhabilitado)");
97           else if (domainRole.Enable == true && domainRole.IdRole == entity.IdRole)
98               throw new NotFoundBusinessException("El Dominio ya tiene asignado el rol seleccionado (Habilitado)");
99       }
100      await _unitAdmin.domainRoleRepository.Add(entity);
101      await _unitAdmin.SaveChangesAsync();
102      return entity;
103  }
```

Fuente: el autor.

**Descripción:** El método Post() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta creación de roles por dominio en el sistema. Se valida que el dominio a asociar al rol exista y este activo, también que un dominio no tenga asignado más de una vez un rol, que el rol a asociar al dominio exista, este habilitado y activo.

**Figura 50.**

*Método Put() para modificar, en la clase de servicio Domail_RoleService*

```
105         public async Task<Domain_Role> Put(Domain_Role entity)
106         {//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
107             if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Actualizar).FirstOrDefault() ==
108                 throw new UnauthorizedBusinessException("El usuario no puede actualizar los datos");
109             //
110             entity = SetAuditFields(entity);
111
112             if (entity.Id > 0)
113             {
114                 // Consulta y valida si el registro de rol asociado a dominio existe y esta activo
115                 Domain_Role domain_role = await _unitAdmin.domainRoleRepository.GetById(entity.Id);
116
117                 if (domain_role != null && domain_role.Active == true)
118                 {
119                     // Consulta el dominio y valida si esta activo
120                     Domain.Entities.Domain domain = await _unitAdmin.domainRepository.GetById(entity.IdDomain);
121
122                     if (domain == null)
123                         throw new NotFoundBusinessException("El Dominio no existe o no se encuentra Activo");
124
125                     // Valida que el rol exista, este activo y habilitado
126                     if (!(_unitAdmin.roleRepository.Role_IsEnabled(entity.IdRole)))
127                         throw new NotFoundBusinessException("El Rol no existe, no está Habilitado o no se encuentra Activo"
128
129                     // Consulta y valida si ya existe el rol asociado al dominio en la BD
130                     var domainRole = _unitAdmin.domainRoleRepository.GetDomain_Role(entity.Id, entity.IdDomain, entity.IdRo
131
132                     if (domainRole != null)
133                     {
134                         if (domainRole.Enable == false && domainRole.IdRole == entity.IdRole)
135                             throw new NotFoundBusinessException("El Dominio ya tiene asignado el rol seleccionado (Inhabili
136                         else if (domainRole.Enable == true && domainRole.IdRole == entity.IdRole)
137                             throw new NotFoundBusinessException("El Dominio ya tiene asignado el rol seleccionado (Habilita
138                     }
139                 } else throw new NotFoundBusinessException("Se requiere que el Rol por Dominio exista y este activo para mo
140             } else throw new NotFoundBusinessException("El 'Id' del Rol por Dominio debe ser mayor a 0");
141             await _unitAdmin.domainRoleRepository.UpdateAsync(entity);
142             await _unitAdmin.SaveChangesAsync();
143             return entity;
144         }
145     }
146 }
```

Fuente: el autor.

**Descripción:** El método Put() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta modificación de roles por dominio en el sistema. Se valida que el rol por dominio a modificar exista y este activo, que el dominio a asociar al rol exista y este activo, también que un dominio no tenga asignado más de una vez un rol, que el rol a asociar al dominio exista, este habilitado y activo.

**Figura 51.**

*Clase Modelo/Entidad del desarrollo del caso de uso de Flujo de notificaciones*

```
using Bext.Identity.Solution.Domain.Entities;
using System.Collections.Generic;
using Attribute = Bext.Identity.Solution.Domain.Entities.Attribute;

namespace Bext.Identity.Solution.Core.Entities
{
    36 references
    public class NotificationFlow : BaseEntity
    {
        0 references
        public NotificationFlow()
        {
            NotificationsFlow_Users_Roles = new HashSet<NotificationFlow_User_Role>();
        }

        9 references
        public int IdNotification { get; set; }
        13 references
        public int? IdAttribute { get; set; }

        3 references
        public virtual Notification IdNotificationNavigation { get; set; }
        1 reference
        public virtual Attribute IdAttributeNavigation { get; set; }
        15 references
        public virtual ICollection<NotificationFlow_User_Role> NotificationsFlow_Users_Roles { get; set; }
    }
}
```

Fuente: el autor.

**Descripción:** Se muestran cada una de las propiedades requeridas de la clase NotificationFlow para el desarrollo del caso de configuración de flujos de notificaciones. Tiene propiedades para indicar el identificador de la notificación y atributo al cual se va a asociar.

La propiedad IdNotification es una llave foránea que se deriva de la clase modelo/entidad Notification, la cual tiene una relación de uno a muchos hacia flujo de notificaciones.

La propiedad IdAttribute es una llave foránea que se deriva de la clase modelo/entidad Attribute, la cual tiene una relación de uno a muchos hacia flujo de notificaciones.

Esta clase NotificationFlow tiene una relación de uno a muchos flujos de notificaciones por usuario y rol (NotificationFlow_User_Role).

**Figura 52.**

*Método Post() para crear, en la clase de servicio NotificationFlowService*

```
76          public async Task<NotificationFlow> Post(NotificationFlow entity)
77          {//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
78              if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Crear).FirstOrDefault() == null)
79                  throw new UnauthorizedBusinessException("El usuario no puede agregar información");
81      entity = SetAuditFields(entity);
82
83      var notification = new Notification();
84      string typeProperty = string.Empty;
85      string attributeScim = string.Empty;
86      string notificationType = string.Empty;
87      bool? notificationTypeIsActive = false;
88      bool? notificationIsActive = false;
89      bool sendNotification = false;
90      bool isToAdmin = false;
91      int? idAttribute = entity.IdAttribute == null ? null : entity.IdAttribute;
92      int? idUser = 0;
93      int? idRole = 0;
94
95      NotificationFlow newNotificationFlow = entity;
96      ICollection<NotificationFlow_User_Role> notificationsFlow_Users_Roles = entity.NotificationsFlow_Users_Roles;
97      newNotificationFlow.NotificationsFlow_Users_Roles = new List<NotificationFlow_User_Role>();
98
99      // Se valida que no se ingresen usuarios y roles repetidos
100     var users = notificationsFlow_Users_Roles.Where(x => x.IdUser != null).Select(x => x.IdUser).ToList();
102     if (users.GroupBy(n => n).Any(c => c.Count() > 1))
103         throw new NotFoundBusinessException("Para este flujo de notificación, quedarian usuarios repetidos en la base de dat
104
105     var roles = notificationsFlow_Users_Roles.Where(x => x.IdRole != null).Select(x => x.IdRole).ToList();
106
107     if (roles.GroupBy(n => n).Any(c => c.Count() > 1))
108         throw new NotFoundBusinessException("Para este flujo de notificación, quedarian roles repetidos en la base de datos"
109
110     // Se obtendrán los flujos de notificaciones existentes por 'idNotification, y 'idAttribute', los flujos de notificacior
111     IEnumerable<NotificationFlow> notificationsFlow = _unitAdmin.notificationFlowRepository.GetNotificationsFlow(entity.IdNc
112
113     // Si existen registros en la Base de Datos por 'idNotification, y 'idAttribute'
114     if(notificationsFlow.Count() > 0)
115     {
116         notification = notificationsFlow.Select(notification => notification.IdNotificationNavigation).FirstOrDefault();
117         notificationIsActive = notification.Active;
118         notificationTypeIsActive = notification.IdNotificationTypeNavigation.Active;
119     }
120     else
121     {   // Si no existen registros en la Base de Datos por 'idNotification, y 'idAttribute', se obtienen los datos asociados
122         notification = await _unitAdmin.notificationRepository.GetById(entity.IdNotification);
123         notificationIsActive = notification.Active;
124         notificationTypeIsActive = notification.IdNotificationTypeNavigation.Active;
125     }
126
127     // Se valida que la notificación y el tipo de notificación estén activos
128     if (notificationIsActive == true)
129     {
```

Fuente: el autor.

**Descripción:** El método Post() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta creación de flujos de notificaciones en el sistema. Se valida que la configuración sea la adecuada según el tipo de notificación (SMS, push, email) y el tipo de destinatario asociado a la notificación (usuarios finales o administradores).

**Figura 53.**

*Método Put() para modificar, en la clase de servicio NotificationFlowService*

```
246  public async Task<NotificationFlow> Put(NotificationFlow entity)
247  {//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
248      if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Actualizar).FirstOrDefault() == null)
249      throw new UnauthorizedBusinessException("El usuario no puede actualizar los datos");
250  //
251  entity = SetAuditFields(entity);
252
253  string typeProperty = string.Empty;
254  string attributeScim = string.Empty;
255  string notificationType = string.Empty;
256  bool? notificationTypeIsActive = false;
257  bool sendNotification = false;
258  bool isToAdmin = false;
259  int? idAttribute = entity.IdAttribute == null ? null : entity.IdAttribute;
260  int? idUser = 0;
261  int? idRole = 0;
262
263  // Se valida que el identificador de la entidad sea mayor a cero
264  if (entity.Id > 0)
265  {
266      // Se obtienen los datos en BD del flujo de notificación a modificar
267      NotificationFlow notificationFlow = await _unitAdmin.notificationFlowRepository.GetById(entity.Id);
268      IEnumerable<NotificationFlow_User_Role> notificationFlow_Users_Roles = notificationFlow.NotificationsFlow_Users_Role
269
270      // Se valida si el flujo de notificación y la notificación a modificar estan activos, incluyendo el tipo de notifica
271      if (notificationFlow != null && notificationFlow.Active == true)
272      {
273          Notification notification = await _unitAdmin.notificationRepository.GetById(entity.IdNotification);
274
275          if (notification != null)
276  {
277      isToAdmin = notification.IsToAdmin;
278      notificationTypeIsActive = notification.IdNotificationTypeNavigation.Active;
279
280      // Se valida que el tipo de notificación este activa
281      if (notificationTypeIsActive == true)
282          notificationType = notification.IdNotificationTypeNavigation.Name.ToLower();
283      else
284          throw new NotFoundBusinessException("El tipo de notificación asociado a la notificación no está activa o no exis
285  }
286  else
287      throw new NotFoundBusinessException("La notificación no está activa o no existe");
288
289  // Se separan los datos por flujo de notificación y flujo de notificación por roles y usuarios
290  NotificationFlow newNotificationFlow = entity;
291  ICollection<NotificationFlow_User_Role> notificationsFlow_Users_Roles = entity.NotificationsFlow_Users_Roles;
292  newNotificationFlow.NotificationsFlow_Users_Roles = new List<NotificationFlow_User_Role>();
293
294  // Se valida que no se ingresen usuarios y roles repetidos
295  var users = notificationsFlow_Users_Roles.Where(x => x.IdUser != null).Select(x => x.IdUser).ToList();
296
297  if (users.GroupBy(n => n).Any(c => c.Count() > 1))
298      throw new NotFoundBusinessException("Para este flujo de notificación, quedarian usuarios repetidos en la base de dat
```

Fuente: el autor.

**Descripción:** El método Put() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta modificación de flujos de notificaciones en el sistema. Se valida que el flujo de notificación a modificar exista y este activo, que la configuración sea la adecuada según el tipo de notificación (SMS, push, email) y el tipo de destinatario asociado a la notificación (usuarios finales o administradores).

- Pruebas de los servicios implementados en desarrollo back-end

**Figura 54.**

*Servicios implementados para el desarrollo de Asignación de Roles a Usuarios*

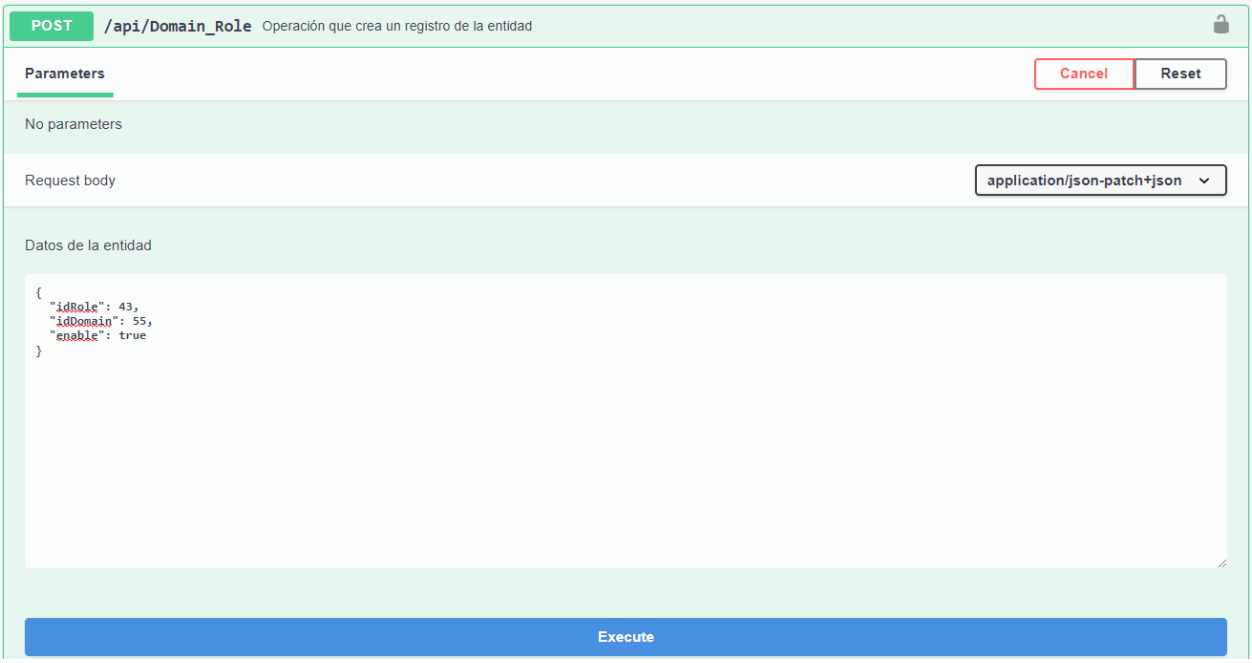
Fuente: el autor.

**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestran los servicios implementados para crear (Post), modificar (Put), eliminar (Delete) y obtener (Get) objetos de roles por usuarios y roles por dominio, según las reglas de negocio, relacionados al desarrollo del caso de uso de Asignación de Roles a Cuenta de Usuario.

Los administradores del sistema podrán establecer los diferentes roles que aplicarán para dominios y usuarios específicos.

**Figura 55.**

*Configurando una asignación de rol por dominio a crear desde Swagger UI*



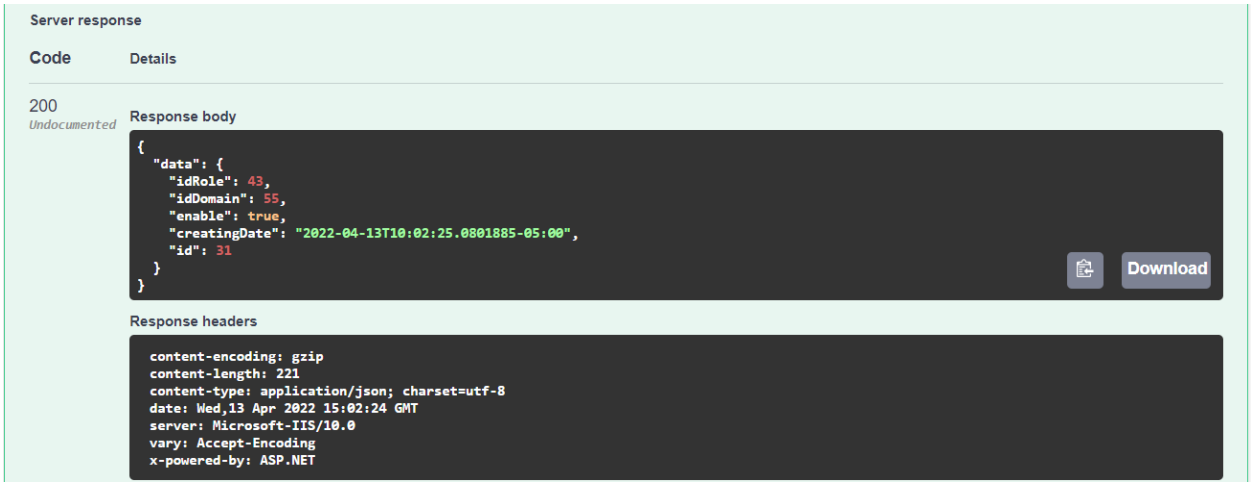

Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se envía la petición y se guarda la asignación de rol por dominio en base de datos.

Los roles definirán los permisos y acciones que el usuario podrá realizar en las diferentes páginas o módulos del aplicativo.

**Figura 56.**

*Respuesta que muestra Swagger UI al asignarse un rol por dominio*

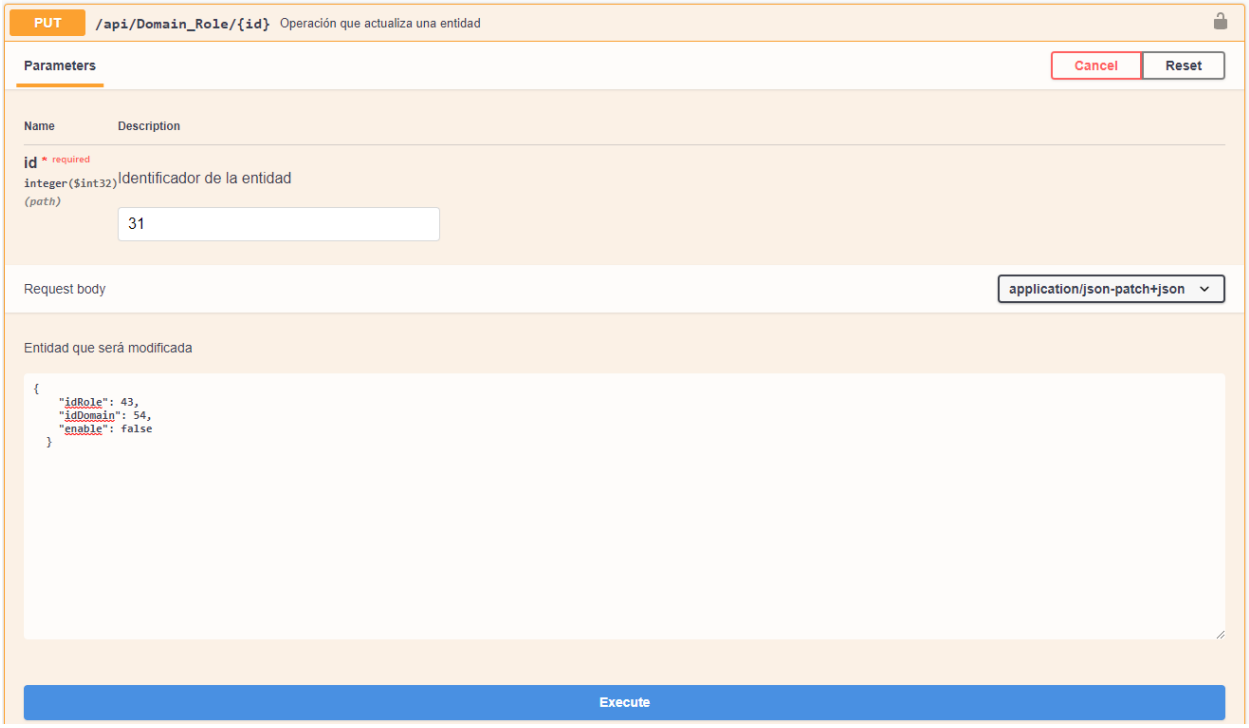



Fuente: el autor.

**Descripción:** Desde el back-end se retorna la configuración de rol por dominio creada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 57.**

*Configurando una asignación de rol por dominio a modificar desde Swagger UI*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se indica el identificador del registro a modificar, se envía la petición y se actualiza la asignación de rol por dominio en base de datos.

Los roles definirán los permisos y acciones que el usuario podrá realizar en las diferentes páginas o módulos del aplicativo.

**Figura 58.**

*Respuesta que muestra Swagger UI al modificarse un rol por dominio*

Fuente: el autor.

**Descripción:** Desde el back-end se retorna la configuración de rol por dominio modificada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 59.**

*Servicios implementados para el desarrollo de Flujo de Notificaciones*

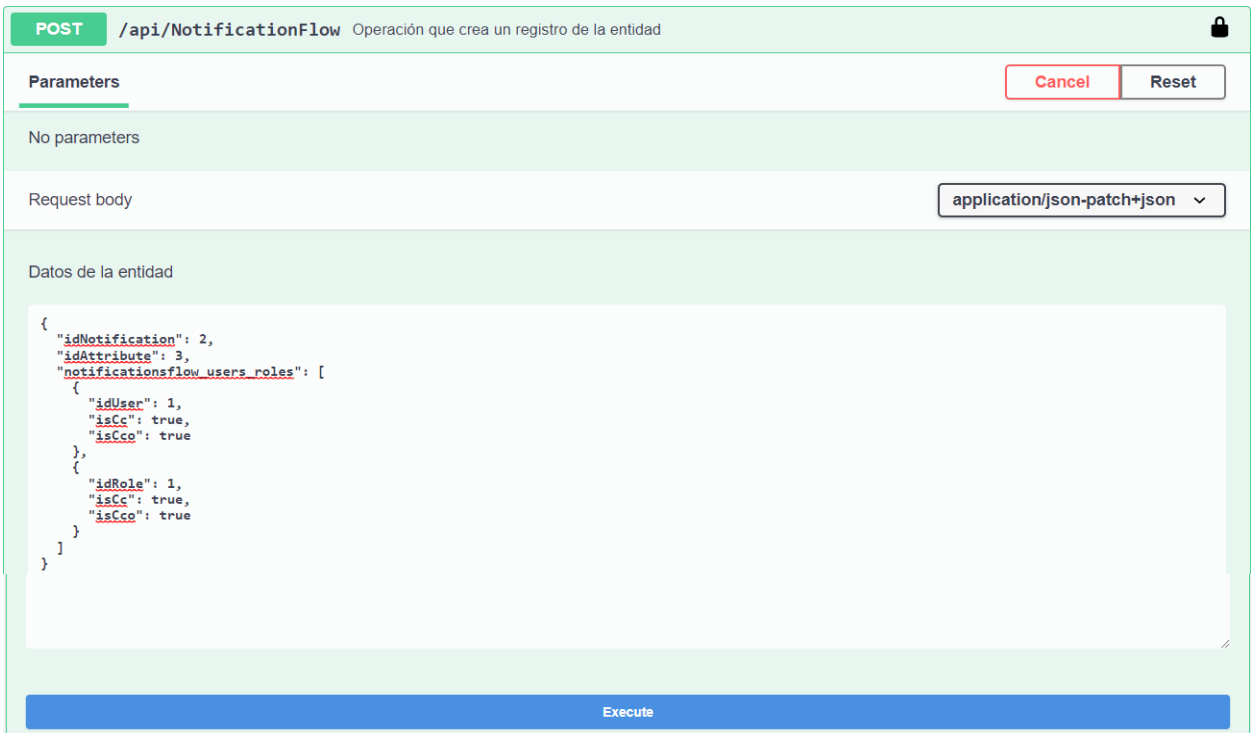
Fuente: el autor.

**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestran los servicios implementados para crear (Post), modificar (Put), eliminar (Delete) y obtener (Get) objetos de flujo de notificación por usuario y rol, según las reglas de negocio, relacionados al desarrollo del caso de uso de Flujo de Notificaciones.

Un flujo de notificación establecerá los destinatarios de una notificación por usuarios específicos y todos aquellos usuarios que pertenezcan a un rol.

**Figura 60.**

*Configurando un flujo de notificación a crear desde Swagger UI*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se envía la petición y se guarda el flujo de notificación por usuarios y roles en base de datos asociado a una notificación específica.

**Figura 61.**

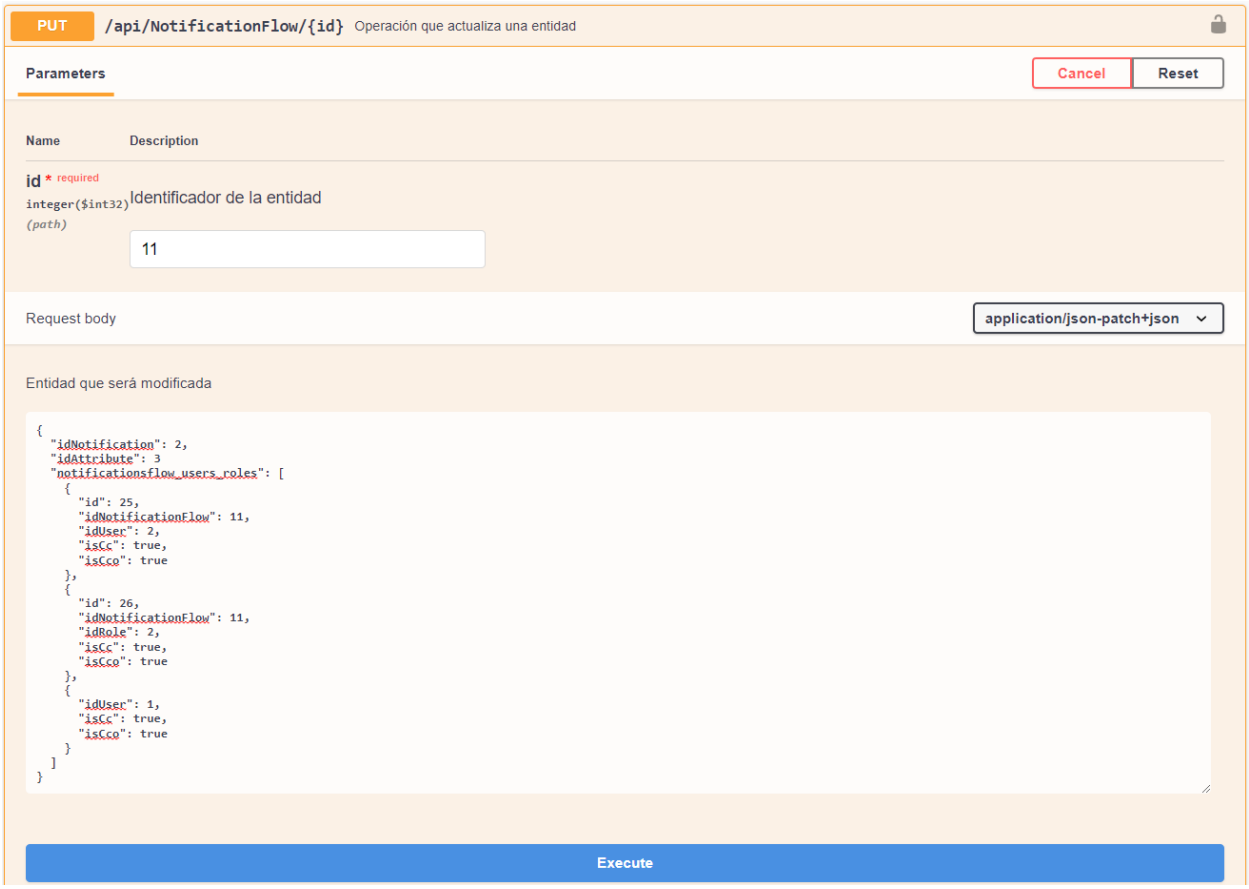
*Respuesta que muestra Swagger UI al crearse un flujo de notificación*

Fuente: el autor.

**Descripción:** Desde el back-end se retorna la configuración de flujo de notificación creada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 62.**

*Configurando un flujo de notificación a modificar desde Swagger UI*

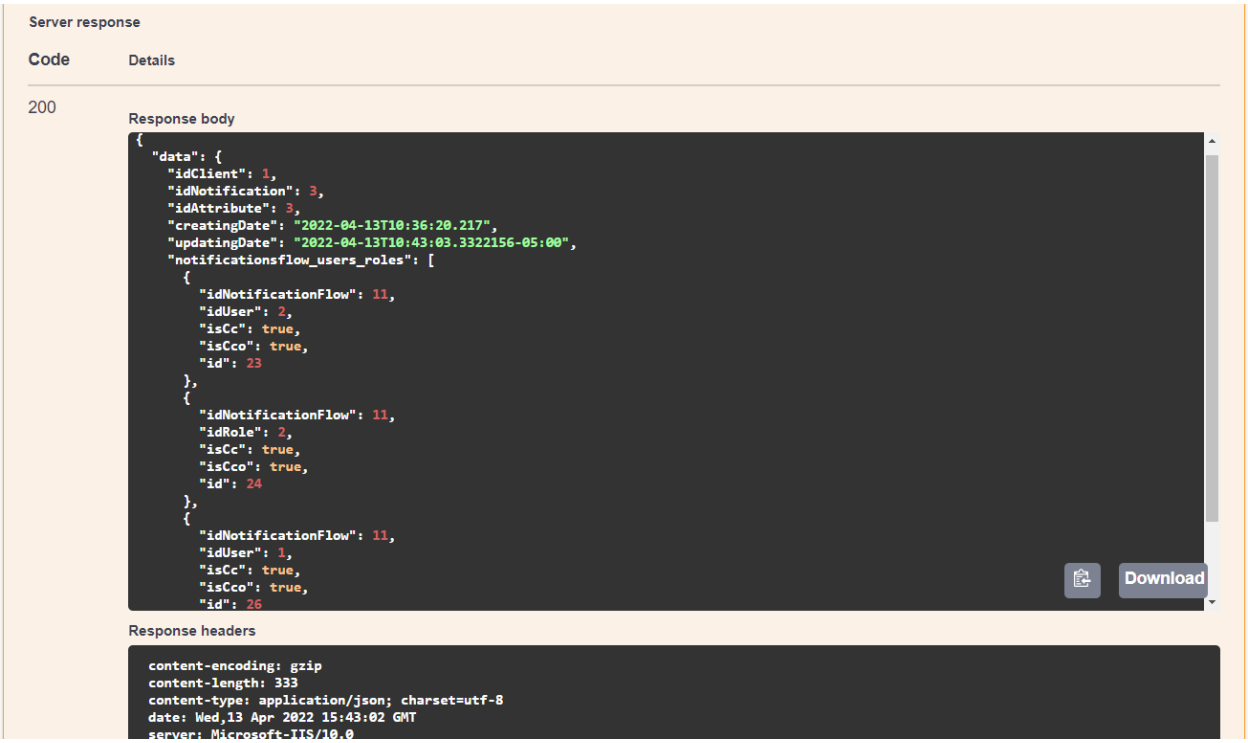



Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se indica el identificador del registro a modificar, se envía la petición y se actualiza el flujo de notificación por usuarios y roles en base de datos asociado a una notificación específica.

**Figura 63.**

*Respuesta que muestra Swagger UI al modificarse un flujo de notificación*



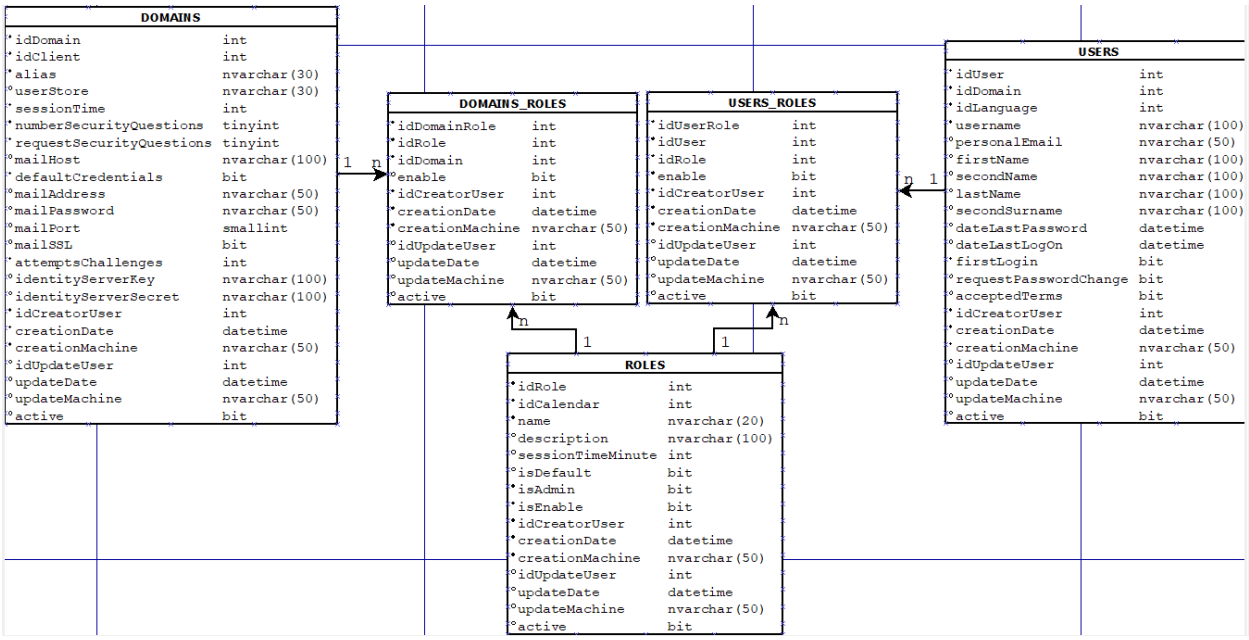

Fuente: el autor.

**Descripción:** Desde el back-end se retorna la configuración de flujo de notificación modificada correctamente en base de datos, mostrándose en Swagger UI.

- Análisis y diseño de bases de datos en motor SQL Server

**Figura 64.**

*Diseño de las tablas Domains_Roles y Users_Roles con relaciones*



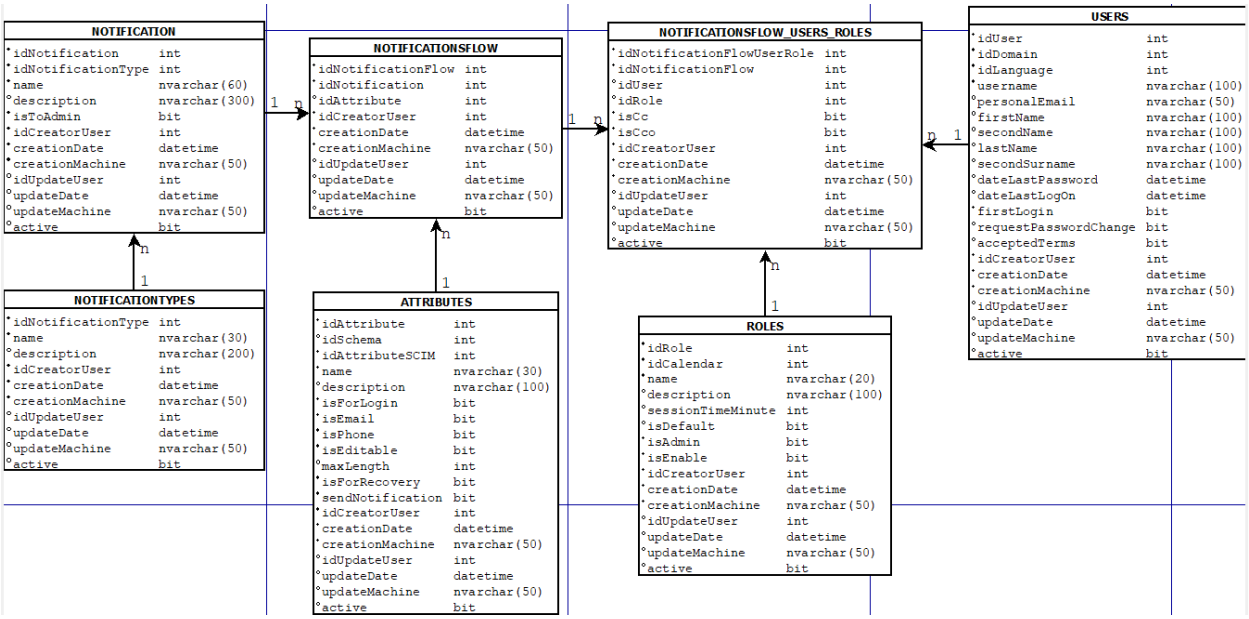

Fuente: el autor.

**Descripción:** Se hizo el modelado de las tablas Domains_Roles y Users_Roles, con cada uno de sus atributos y tipo de dato, se añadió la relación de un (1) dominio a muchos (n) roles por dominio, un (1) rol a muchos (n) roles por dominio, un (1) rol a muchos (n) roles por usuario, un (1) usuario a muchos (n) roles por usuario.

Necesario para el desarrollo back-end del caso de uso de Asignación de Roles a Cuenta de Usuario.

**Figura 65.**

*Diseño de las tablas NotificationsFlow y NotificationsFlow_Users_Roles*




Fuente: el autor.

**Descripción:** Se hizo el modelado de las tablas NotificationsFlow y NotificationsFlow_Users_Roles, con cada uno de sus atributos y tipo de dato, Se añade relación de una (1) notificación a muchos (n) flujos de notificación, un (1) flujo de notificación a muchos (n) flujos de notificación por usuario y rol, un (1) atributo a muchos (n) flujos de notificación, un (1) rol a muchos (n) flujos de notificación por usuario y rol, un (1) usuario a muchos (n) flujos de notificación por usuario y rol

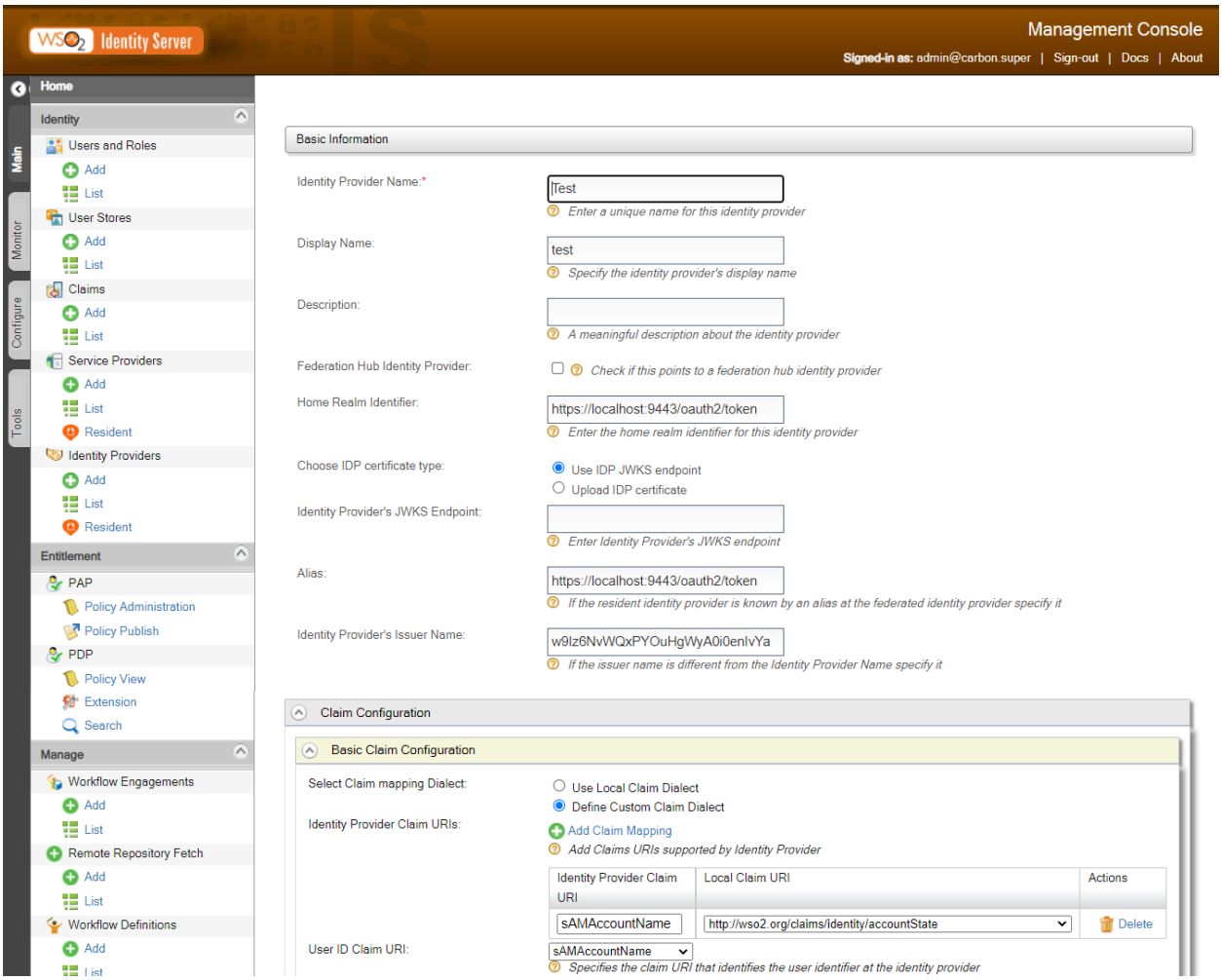
Necesario para el desarrollo back-end del caso de uso de Flujo de Notificaciones.

- Implementación y configuración de servidores de identidad

Para el desarrollo, se hizo necesario el uso de “WSO2 Identity Server”, un servidor de identidad de la empresa WSO2 que brinda seguridad y facilidad en la gestión de identidades.

**Figura 66.**

*Configurar un proveedor de identidad en servidor de identidad WSO2*

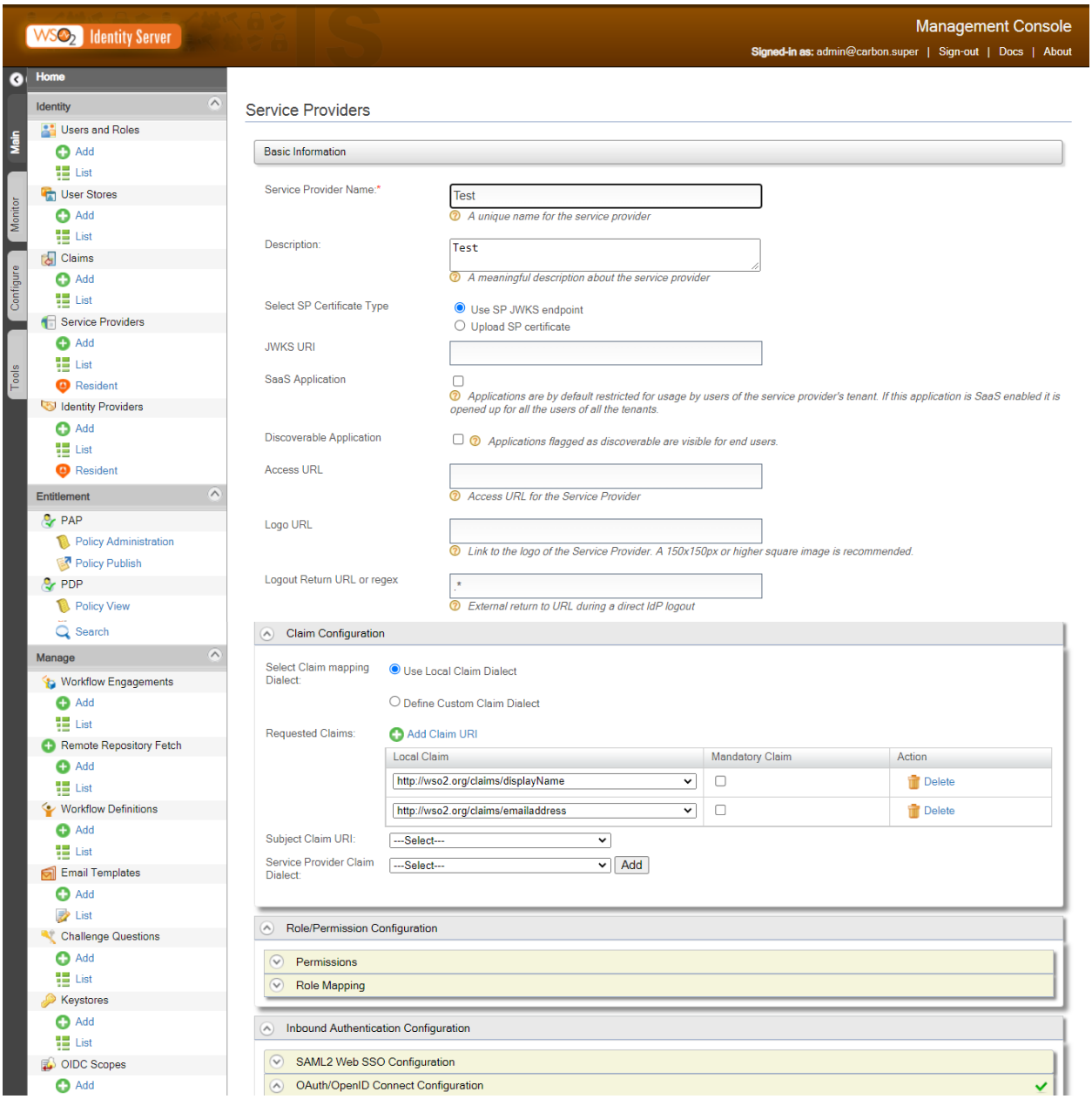



Fuente: el autor.

**Descripción:** Se configura un proveedor de identidad, el cual cumplirá la función de autenticar y autorizar usuarios del proveedor de servicios en procesos de inicio de sesión.

**Figura 67.**

*Configurar un proveedor de servicio en servidor de identidad WSO2*

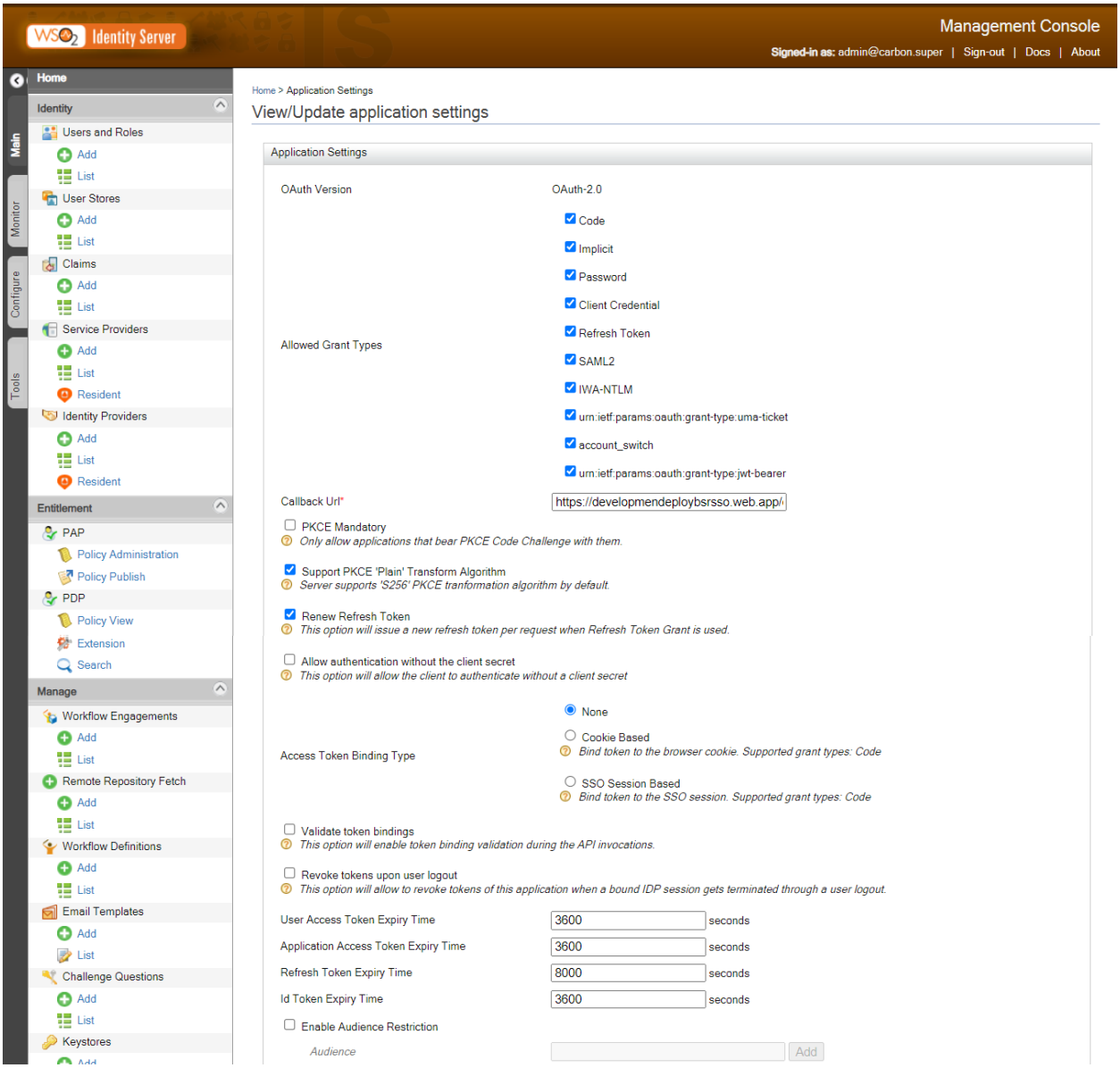



Fuente: el autor.

**Descripción:** Se configura un proveedor de servicios, el cual cumplirá la función de proveer servicios web y recursos a los usuarios; para ello, envía una solicitud al proveedor de identidad para autenticar y autorizar usuarios antes de conceder el acceso.

**Figura 68.**

*Configuración en detalle del proveedor de servicio*

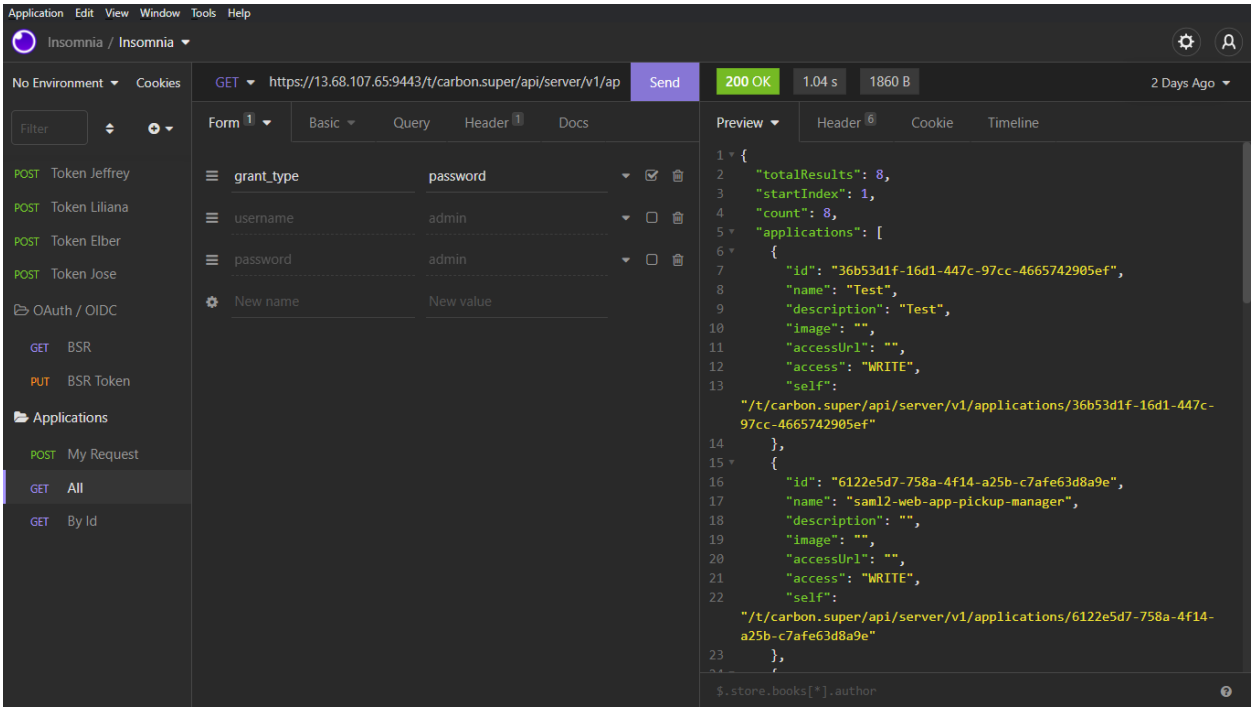



Fuente: el autor.

**Descripción:** Configuración relacionada a la solicitud de autenticación y autorización OAuth/OpenID Connect que se genera desde el proveedor de servicios y se envía al componente de autenticación entrante, establecer tiempo de expiración y refrescar token de acceso.

**Figura 69.**

*Prueba en Software Insomnia para obtener todos los proveedores de servicio*



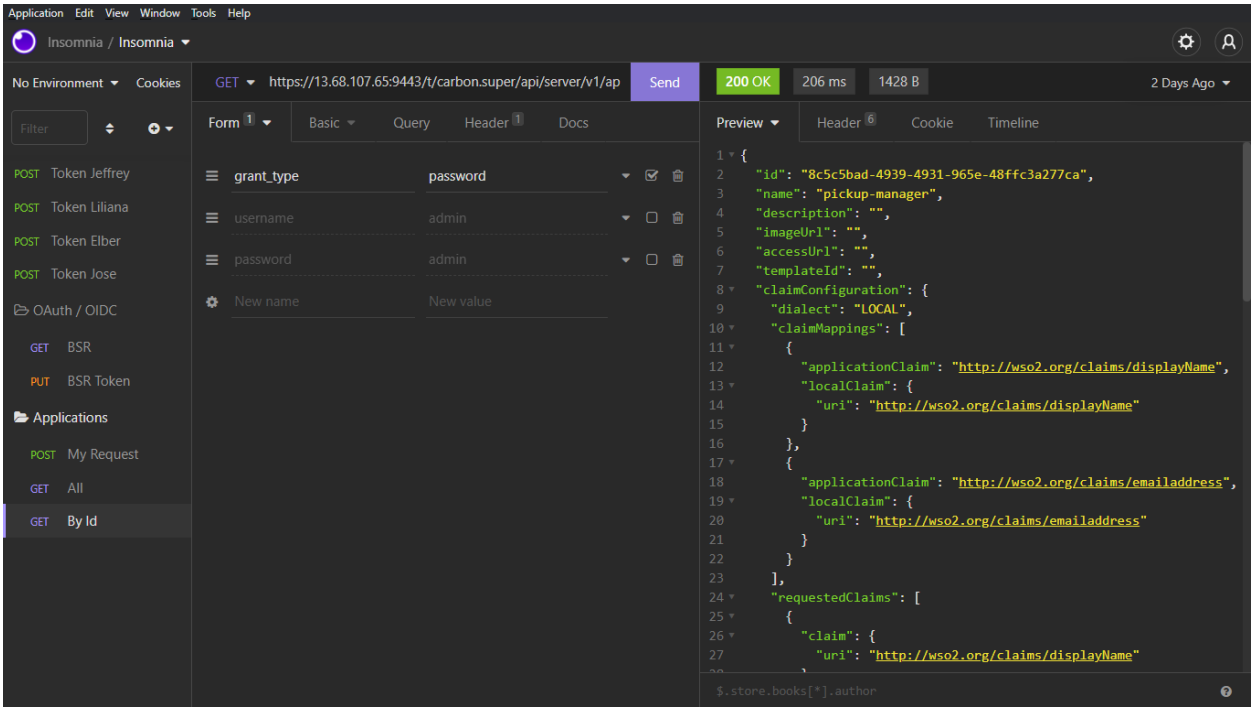

Fuente: el autor.

**Descripción:** Se configuró una petición en el software Insomnia para consumir un servicio Rest que retorna todos los proveedores de servicios que estén configurados en el servidor de identidad WSO2.

Los resultados se retornan en un arreglo de objetos.

**Figura 70.**

*Prueba en Software Insomnia para consultar proveedor de servicio específico*



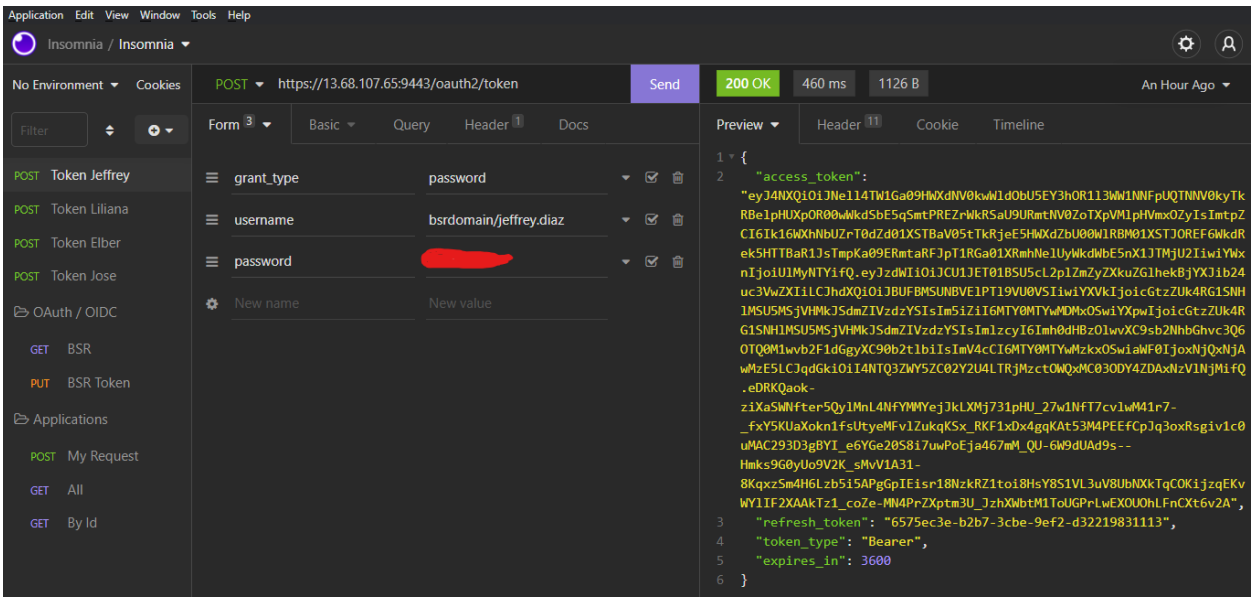

Fuente: el autor.

**Descripción:** Se configuró una petición en el software Insomnia para consumir un servicio Rest que retorna los detalles asociados a un proveedor de servicios específico configurado en el servidor de identidad WSO2.

La consulta se realiza mediante un identificador que posee cada proveedor de servicios.

**Figura 71.**

*Generar token de acceso (JWT) en software Insomnia para consumir servicios*




Fuente: el autor.

**Descripción:** Al configurar adecuadamente un proveedor de identidad y un proveedor de servicios, se puede configurar la petición a un punto final (endpoint) que permite obtener token de acceso. El punto final se obtiene de la configuración del proveedor de identidad creado.

En el resultado se retorna el token de acceso con el cual se permitirá autorizar la API adecuada para consumir los diferentes servicios Rest que el usuario tenga permiso, ya sea solo de lectura, escritura o ambos. También se obtiene un código que usara el aplicativo cliente para obtener un nuevo token de acceso (refresh token) del servidor de identidad cuando el token actual esté próximo a espirar o haya vencido, esto para que el inicio de sesión del usuario sea continuo y no tenga que ingresar las credenciales cada vez que expire el token de acceso actual. También el tipo de token de acceso generado y el tiempo en segundos en que expirará.

La petición configurada fue una comunicación directa al servidor de identidad WSO2.

**4.** En el cuarto mes (6 de enero del 2022 a 5 de febrero del 2022) se hizo:

- El estudio y desarrollo del caso de uso [CU110] Consultar y actualizar datos, añadiendo las reglas de negocio y funcionalidades necesarias en el código para permitir que un usuario final (básico) del sistema pueda realizar el proceso de consultar y actualizar sus datos personales en la plataforma BSR SSO, teniendo en cuenta las configuraciones del módulo de administración de atributos donde se establecen que atributos son tipo lectura y escritura.
- El estudio y desarrollo del caso de uso [CU108] Cambiar contraseña, añadiendo las reglas de negocio y funcionalidades necesarias en el código para permitir que un usuario final (básico) del sistema pueda realizar el proceso de cambiar la contraseña de su cuenta personal en la plataforma de BSR SSO, teniendo en cuenta las diferentes políticas de seguridad que se apliquen por dominio al cambio de contraseña, tales como: caracteres requeridos, longitud mínima, longitud máxima, restricciones (texto y datos personales) y expresiones regulares.

A continuación, se comparten algunas vistas que detallan las actividades realizadas durante este mes:

- Desarrollo back-end de los casos de uso

**Figura 72.**

*Método Get() que retorna atributos editables en clase de servicio AttributeService*

```
File Edit View Git Project Build Debug Team Test Analyze Tools Extensions Window Help Full Screen
Domain_SchemaRepository.cs*  Attributes.cs  AttributeService.cs  AttributeController.cs
Bext.Identity.Solution.Core  Bext.Identity.Solution.Core.Services.AttributeService  GetAttributesScim(int idUser)
2 references
88  public async Task<IEnumerable<AttributesScimDto>> GetAttributesScim(int idUser)
89  {
90      //Este código verifica si el usuario tiene permiso para poder acceder a este servicio
91      if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Consultar).FirstOrDefault() == null)
92          throw new UnauthorizedBusinessException("El usuario no puede consultar la informacion");
93      //
94      List<AttributesScimDto> attributesScim = new List<AttributesScimDto>();
95
96      if (idUser > 0)
97      {
98          // Se consulta el usuario para obtener el dominio
99          User user = await _unitAdmin.userRepository.GetById(idUser);
100
101         if (user != null)
102         {
103             // Se obtiene toda la información de los esquemas, atributos editables y Scim
104             IEnumerable<Domain_Schema> domains_schemas = _unitAdmin.domainSchemaRepository.GetEditableAttributes(user.IdDom
105             // Se seleccionan los esquemas
106             IEnumerable<Schema> schemas = domains_schemas.Select(x => x.IdSchemaNavigation).Where(x => x.Active == true).To
107
108             foreach (var schema in schemas)
109             {
110                 // Se obtiene la información del usuario en el AD en estándar Scim por esquema
111                 UserResponseSCIM userFromAD = await _unitAdmin.userSCIMRepository.GetById(user.Username, schema);
112
113                 // Se recorren los atributos editables por esquema y se añaden a una lista por objetos
114                 foreach (var attribute in schema.Attributes)
115                 {
116     int idAttributeScim = attribute.IdAtributesScimNavigation.Id;
117     string attributeName = attribute.Name;
118     string attributeScim = attribute.IdAtributesScimNavigation.Atribute;
119     string typeProperty = attribute.IdAtributesScimNavigation.TypeProperty;
120
121     switch (attributeScim)
122     {
123         case Attributes.Department:
124             {
125                 if (userFromAD.extensionEnterpriseUser.department != null)
126                     attributesScim.Add(new AttributesScimDto
127                     {
128                         id = idAttributeScim, attribute = attributeName,
129                         attributeScim = attributeScim, value = userFromAD.extensionEnterpriseUser.department
130                     });
131                 else
132                     attributesScim.Add(new AttributesScimDto
133                     {   id = idAttributeScim, attribute = attributeName,
134                         attributeScim = attributeScim, value = null
135                     });
136                 break;
137             }
```

Fuente: el autor.

**Descripción:** El método GetAttributesScim() para el desarrollo del caso de uso de consultar y actualizar datos, retorna el listado de atributos personales que un usuario final puede modificar en el sistema. También se indicará el atributo Scim asociado y el valor actual.

**Figura 73.**

*Método Patch() que actualiza atributos editables en clase de servicio UserService*

```
32  public async Task<bool> UpdateUserData(string username, Dictionary<string, AttributeValue> userData)
33  {
34      var Operations = new List<object>();
35      var schemas = userData.Values.Select(x => x.Attribute.IdSchemaNavigation).Distinct().ToList();
36
37      foreach (var schema in schemas)
38      {
39          var attributes = userData.Values.Select(x => x.Attribute).Where(x => x.IdSchema == schema.Id).ToList();
40
41          foreach (var attribute in attributes)
42          {
43              if(userData.TryGetValue(attribute.Name, out AttributeValue attributeValue))
44              {
45                  var attributeScim = attribute.IdAtributesScimNavigation.Atribute;
46                  var value = attributeValue.Value;
47
48                  switch (attributeScim)
49                  {
50                      case Attributes.Country:
51                          {
52                              if (Attributes.TypeWork == attribute.IdAtributesScimNavigation.TypeProperty)
53                                  Operations.Add(new
54                                  {
55                                      op = PatchOperations.Replace,
56                                      path = "addresses[type eq \"work\"].country",
57                                      value = value
58                                  });
59                              if (Attributes.TypeHome == attribute.IdAtributesScimNavigation.TypeProperty)
60                                  Operations.Add(new
61                                  {
62                                      op = PatchOperations.Replace,
63                                      path = "addresses[type eq \"home\"].country",
64                                      value = value
65                                  });
66                              if (Attributes.TypeOther == attribute.IdAtributesScimNavigation.TypeProperty)
67                                  Operations.Add(new
68                                  {
69                                      op = PatchOperations.Replace,
70                                      path = "addresses[type eq \"other\"].country",
71                                      value = value
72                                  });
73                              break;
74                          }
75                      case Attributes.Department:
76                          {
77                              Operations.Add(new
78                              {
79                                  op = PatchOperations.Replace,
80                                  path = "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:department",
81                                  value = value
82                              });
83                              break;
84                          }
85                      case Attributes.DisplayName:
86                          {
87                              Operations.Add(new
88                              {
89                                  op = PatchOperations.Replace,
90                                  path = "displayName",
91                                  value = value
92                              });
```

Fuente: el autor.

**Descripción:** Desde el método de tipo Patch en la clase UserService se llama al método UpdateUserData() preparando la petición con el tipo de operación, ruta y valor a actualizar en los atributos editables del usuario; para el desarrollo del caso de uso de consultar y actualizar datos.

**Figura 74.**

*Método Patch() que consulta políticas de seguridad por dominio*

```
281     public async Task<bool> PatchUserAsync(int id, JsonPatchDocument<User> userPatch, string device)
282     {//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
283         if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Actualizar).FirstOrDefault() == null)
284             throw new UnauthorizedBusinessException("El usuario no puede actualizar los datos");
285         //
286         if (userPatch != null)
287         {
288             bool response = false, search = false;
289             string personalEmail = string.Empty;
290             Dictionary<string, AttributeValue> userData = new Dictionary<string, AttributeValue>();
291             User user = _unitAdmin.userRepository.GetAll().Where(x => x.Id == id).FirstOrDefault();
292             IEnumerable<Domain_Schema> schemasAndAttributes = null;
293
294             //Elementos que pueden ser modificados a través del patch
295             foreach (Microsoft.AspNetCore.JsonPatch.Operations.Operation operation in userPatch.Operations)
296             {
297                 switch (operation.path)
298                 {
360 case UserAttributeNames2Patch.Password:
361 {
362     if (user == null)
363         throw new NotFoundBusinessException("El usuario no fue encontrado");
364
365     if (operation.OperationType == OperationType.Replace)
366     {
367         string value = operation.value.ToString();
368         string newValue = Regex.Replace(value, @"\s", "");
369         string unfulfilledPolicies = String.Empty;
370
371         if (!string.IsNullOrEmpty(value) && newValue.Length > 0)
372         {
373             // Se obtiene información del dominio, esquema primario y políticas de seguridad por dominio
374             Domain.Entities.Domain domain = _unitAdmin.domainRepository.GetDomainWithClient(user.IdDomain);
375
376             // Se obtienen los esquemas por dominio del usuario
377             IEnumerable<Domain_Schema> domains_Schemas = _unitAdmin.domainSchemaRepository.GetDomains_Schemas(user.IdDomain
378
379             // Esquemas del usuario
380             var schemas = domains_Schemas.Select(x => x.IdSchemaNavigation).Where(x => x.Active == true).Distinct().ToList(
381
382             Dictionary<string, string> atributesScim = _unitAdmin.atributesScimRepository.GetAtributesScim();
383
384             // Políticas de seguridad por dominio del usuario aplicadas a la contraseña
385             var securityPolicies = domain.Securitypolicies.ToList();
386             var scimUserData = new List<UserResponseSCIM>();
387
388             // Se obtiene la información del usuario en el AD en estándar SCIM por esquema y se añade a una lista
389             foreach (var schema in schemas)
390     {
391         UserResponseSCIM userFromAD = await _unitAdmin.userSCIMRepository.GetById(user.Username, schema);
392         scimUserData.Add(userFromAD);
393     }
394
395     var result = _unitAdmin.utilitiesRepository.AllowPassword(securityPolicies, scimUserData, atributesScim, value, out List<Security
396     unfulfilledPolicies = String.Join(", ", unfulfilled.SelectMany(language => language.SecurityPolicyLanguages.Where(x => x.IdLangua
397
398     // Si se cumple con todas las políticas de seguridad aplicadas a la contraseña, se procede a cambiarla
399     if (result)
400     {
401         try
402         {
403             var res = await _unitAdmin.userSCIMRepository.ChangePassword(user.Username, value, domain.Domains_Schemas.FirstOrDefault(
404         }
```

Fuente: el autor.

**Descripción:** Desde el método de tipo Patch en la clase UserService se consultan las políticas de seguridad por dominio aplicadas al cambio de contraseña y los datos del usuario en los diferentes esquemas; para el desarrollo del caso de uso de Cambiar contraseña del usuario.

**Figura 75.**

*Se validan algunas políticas de seguridad para el cambio de contraseña*

```
2 references
public bool AllowPassword(List<SecurityPolicy> securityPolicies, List<UserResponseSCIM> scimUserData,
                          Dictionary<string, string> atributesScim, string password,
                          out List<SecurityPolicy> unfulfilled)
{
    unfulfilled = new List<SecurityPolicy>();
    bool isValid = true;
    var passwordToLower = password.ToLower();

    if (securityPolicies.Count > 0)
    {
        foreach (SecurityPolicy securityPolicy in securityPolicies)
        {
  // Se validan los Valores restringidos (texto no permitido, primera ocurrencia)
  List<string> deniedValues = securityPolicy.SecurityPoliciesValue.Where(x => x.IdSecurityPolicyType ==
                                                                          (int)SecurityPoliciesTypes.Deny
                                                                          && x.Active == true).Select(x => x.Value).ToList();
  if (deniedValues.Count > 0)
  {
      if (deniedValues.Any(reg => passwordToLower.Contains(reg.ToLower())))
      {
          unfulfilled.Add(securityPolicy);
          isValid = false;
      }
  }

  // Se validan los Valores requeridos (Caracteres requeridos, debe cumplir con al menos una ocurrencia)
  List<string> requiredValues = securityPolicy.SecurityPoliciesValue.Where(x => x.IdSecurityPolicyType ==
                                                                           (int)SecurityPoliciesTypes.Required
                                                                           && x.Active == true).Select(x => x.Value).ToList();
  if (requiredValues.Count > 0)
  {
      if (!requiredValues.Any(reg => passwordToLower.Contains(reg.ToLower())))
      {
          unfulfilled.Add(securityPolicy);
          isValid = false;
      }
  }

  // Se validan los Datos personales (no permitidos)
  List<string> idAttributesScim = securityPolicy.SecurityPoliciesValue.Where(x => x.IdSecurityPolicyType ==
                                                                             (int)SecurityPoliciesTypes.PersonalData
                                                                             && x.Active == true).Select(x => x.Value).ToList();
  if (idAttributesScim.Count > 0)
  {
      // Se recorre cada identificador de atributo Scim y se obtiene su valor del diccionario
      foreach (var attributeScim in idAttributesScim)
      {
          atributesScim.TryGetValue(attributeScim, out string attribute);

          // Se recorre cada objeto que contiene los datos del usuario en estándar Scim por esquema
          foreach (var userData in scimUserData)
          {
              switch (attribute)
              {
```

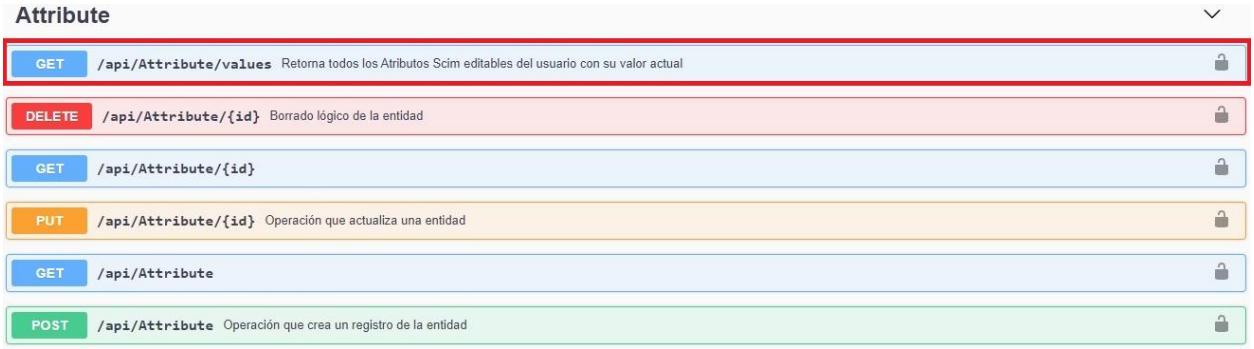
Fuente: el autor.

**Descripción:** En el método AllowPassword() se evalúa si la nueva contraseña del usuario cumple con las diferentes políticas de seguridad por dominio para el cambio de contraseña. Se comprueba tamaño mínimo y máximo permitidos, caracteres y palabras restringidas, caracteres requeridos, que no contenga datos personales y el cumplimiento de expresiones regulares.

- Pruebas de los servicios implementados en desarrollo back-end

**Figura 76.**

*Servicio implementado para el desarrollo de Consultar y actualizar datos*



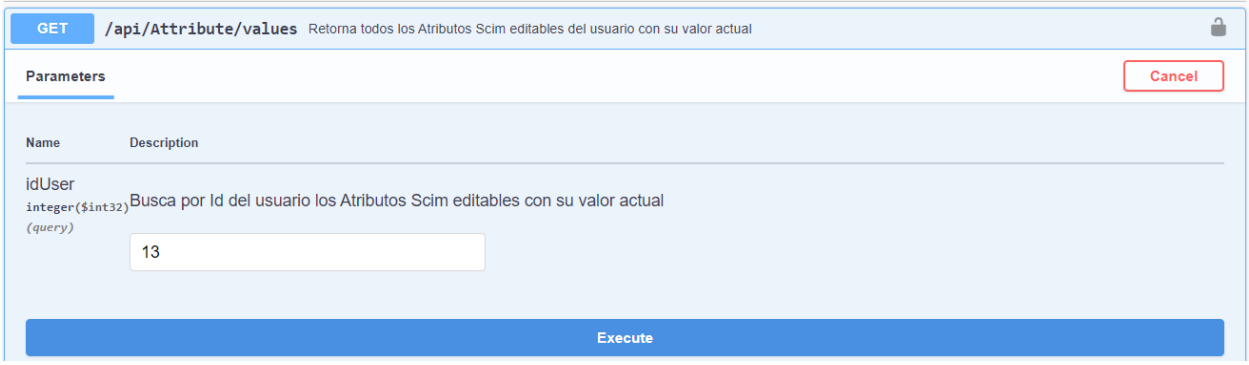

Fuente: el autor.

**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestra el servicio implementado para obtener (Get) los atributos editables del usuario con su valor actual, incluyendo la homologación con el estándar Scim, según las reglas de negocio, relacionados al desarrollo del caso de uso de Consultar y actualizar datos.

Los atributos que se retornen al ejecutar este servicio serán aquellos que el usuario podrá modificar. Hacen referencia a aspectos personales como el nombre, apellidos, número de celular, correo electrónico, etc.

**Figura 77.**

*Consultar los atributos editables del usuario en Swagger UI*




Fuente: el autor.

**Descripción:** Se indica el identificador del usuario en base de datos para consultar los atributos editables que tenga asociados por dominio y esquemas, incluyendo la homologación con el estándar Scim y se envía la petición.

El sistema consultará todos los esquemas por dominio del usuario y para cada esquema obtendrá los atributos editables correspondientes a características e información personal que el usuario podrá modificar. También se mostrará el valor actual para cada atributo.

**Figura 78.**

*Respuesta que muestra Swagger UI al consultar atributos editables del usuario*

Server response

Code | Details

200

Response body


```
{
  "data": [
    {
      "attribute": "Email Empresarial",
      "atribute": "emails",
      "value": "jeffreyd@bextsa.com",
      "typeProperty": "work",
      "isPrimary": true,
      "isEmail": true,
      "isPhone": false,
      "id": 30
    },
    {
      "attribute": "Celular Empresarial",
      "atribute": "phoneNumbers",
      "value": "11243800",
      "typeProperty": "work",
      "isEmail": false,
      "isPhone": true,
      "id": 33
    },
    {
      "attribute": "Idioma preferido",
      "atribute": "preferredLanguage",
      "value": "ruso",
      "isEmail": false,
      "isPhone": false,
      "id": 23
```


Download

Response headers

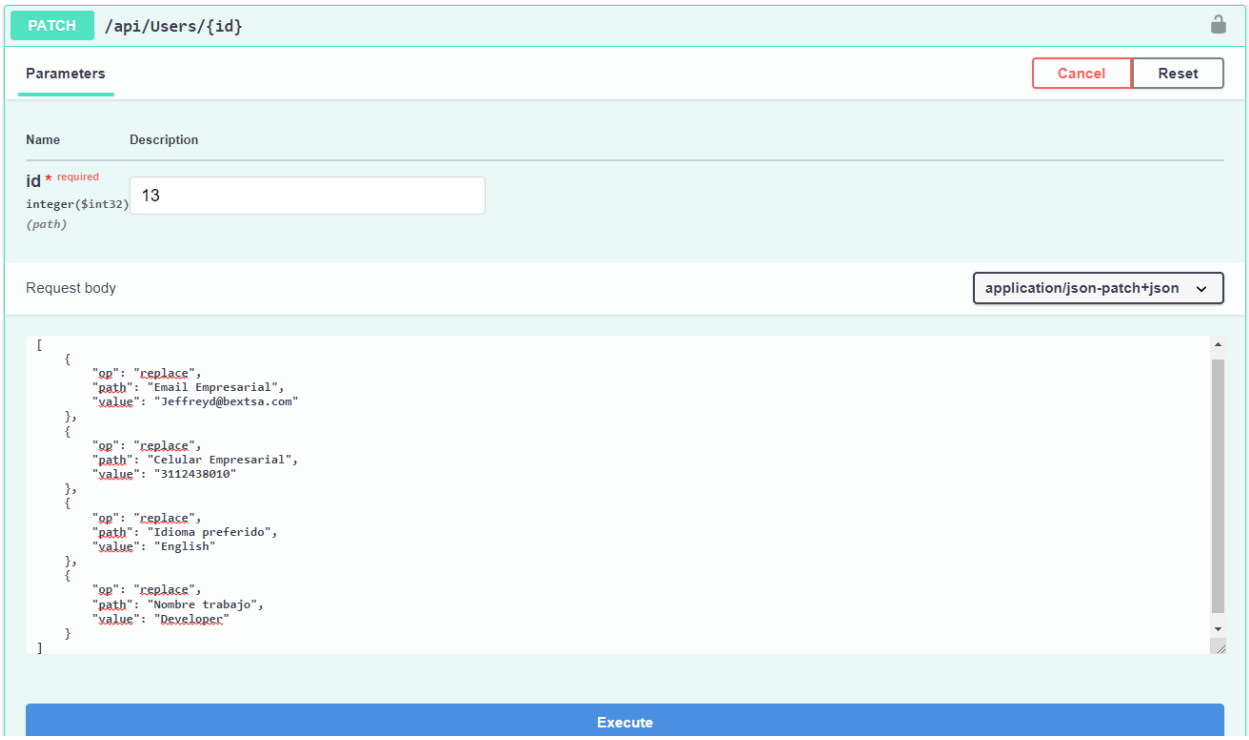
```
content-encoding: gzip
content-length: 521
content-type: application/json; charset=utf-8
date: Wed,13 Apr 2022 15:53:03 GMT
server: Microsoft-IIS/10.0
```

Fuente: el autor.

**Descripción:** Desde el back-end se retornan los atributos editables del usuario con su valor actual, incluyendo la homologación con el estándar Scim, mostrándose en Swagger UI.

**Figura 79.**

*Configurando los atributos editables del usuario a actualizar en Swagger UI*

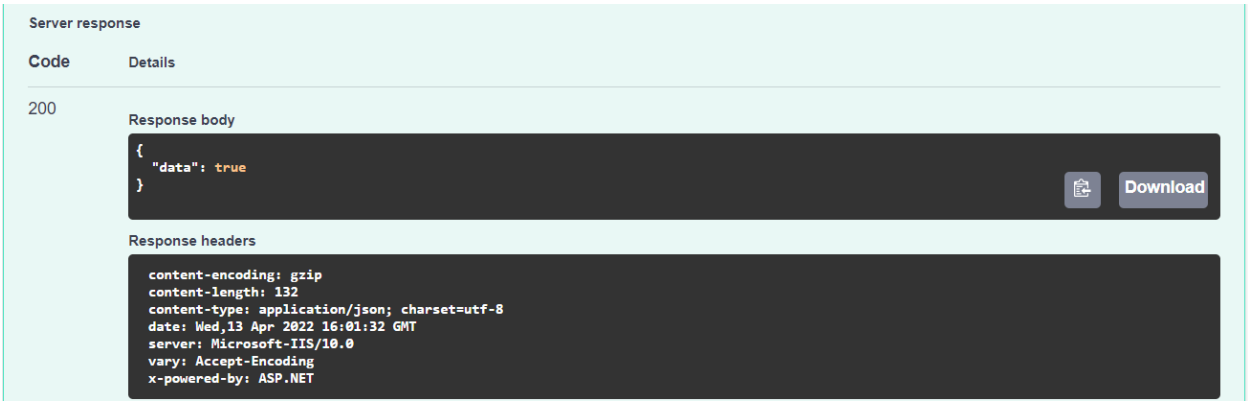



Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el nuevo valor para cada uno de los atributos editables del usuario, en un arreglo de objetos en formato JSON, se indica el identificador del usuario en base de datos, se envía la petición y se actualiza la información personal y empresarial del usuario en cada uno de los repositorios de usuarios en que este se encuentre habilitado.

**Figura 80.**

*Respuesta que muestra Swagger UI al actualizarse los atributos editables*




Fuente: el autor.

**Descripción:** Desde el back-end se retorna un valor de tipo lógico (true), indicando que la actualización de los atributos editables de la cuenta del usuario, se realizó correctamente, mostrándose en Swagger UI.

**Figura 81.**

*Servicio en el que se implementó el desarrollo de Cambiar contraseña*

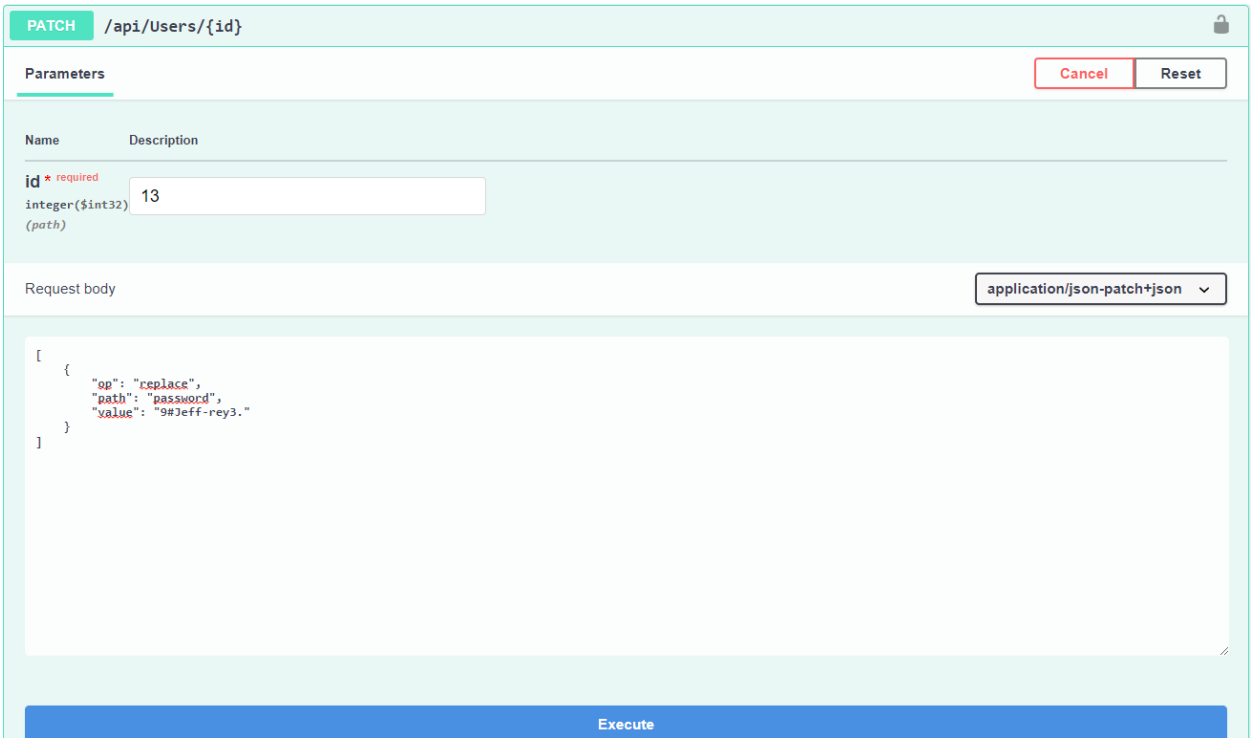
Fuente: el autor.

**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestra el servicio de actualizaciones parciales (Patch) en el que se implementaron las reglas de negocio necesarias para el desarrollo del caso de uso de Cambiar contraseña del usuario, teniendo en cuenta las políticas de seguridad por dominio aplicadas a la contraseña.

Las políticas de seguridad hacen referencia a valores requeridos, no permitidos y patrones de expresiones regulares que debe cumplir la nueva contraseña.

**Figura 82.**

*Petición para actualizar la contraseña de la cuenta de usuario en Swagger UI*

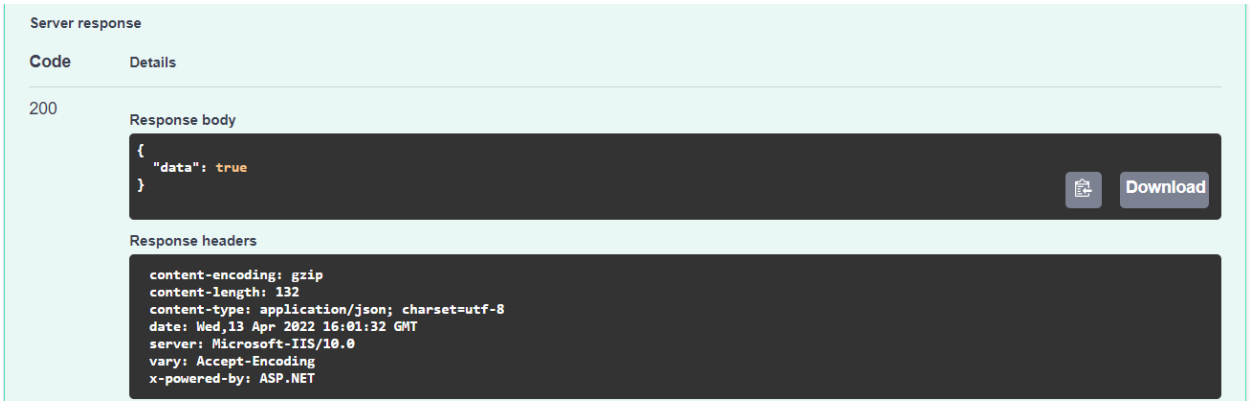



Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el nuevo valor que tendrá la contraseña de la cuenta del usuario en el directorio activo, desde un arreglo de objetos en formato JSON, se indica el identificador del usuario en base de datos al cual le aplicará, se envía la petición y se actualiza la contraseña.

**Figura 83.**

*Respuesta que muestra Swagger UI al actualizarse la contraseña de la cuenta*




Fuente: el autor.

**Descripción:** Desde el back-end se retorna un valor de tipo lógico (true), indicando que la actualización de la contraseña de la cuenta del usuario se realizó correctamente, ya que la nueva contraseña cumplió con todas las políticas de seguridad por dominio, mostrándose en Swagger UI.

**5.** En el quinto mes (6 de febrero del 2022 a 5 de marzo del 2022) se hizo:

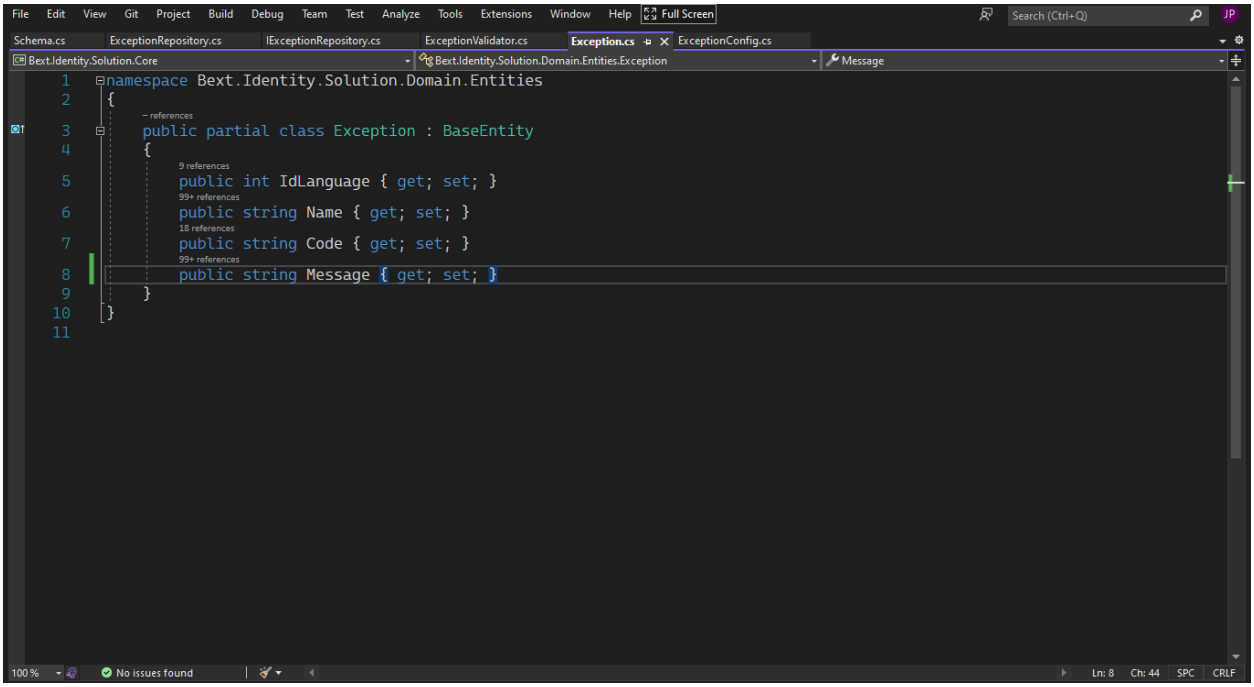
- El estudio y desarrollo del caso de uso [CU13] Configuración de mensajes de error / Excepción, añadiendo las reglas de negocio y funcionalidades necesarias en el código para permitir que un usuario administrador del sistema pueda gestionar la configuración que permita personalizar los mensajes de errores y excepciones por idioma en la plataforma BSR SSO. Estos mensajes de error se mostrarán a los usuarios finales (básicos) y administradores cada vez que ejecuten peticiones incorrectas en el sistema.
- El estudio y desarrollo del caso de uso [CU31] Administración de Esquema, añadiendo las reglas de negocio y funcionalidades necesarias en el código para permitir que un usuario administrador del sistema pueda gestionar la configuración de los diferentes esquemas o repositorios de usuarios a los que se podrá conectar el aplicativo BSR SSO para leer y actualizar información, cambiar contraseña de cuentas de usuarios.
- Una implementación de despliegue continuo en Azure DevOps mediante App Service para un sitio web de pruebas QA.

A continuación, se comparten algunas vistas que detallan las actividades realizadas durante este mes:

- Desarrollo back-end de los casos de uso

**Figura 84.**

*Clase Entidad del desarrollo del caso de uso de configuración de excepciones*




Fuente: el autor.

**Descripción:** Se muestran cada una de las propiedades requeridas de la clase Exception para el desarrollo del caso de uso de configuración de mensajes de error/Excepción. La propiedad IdLanguage es el identificador del idioma asociado a la excepción, se indica también un nombre, código y mensaje para la excepción.

**Figura 85.**

*Método Get() que retorna todas las excepciones en clase ExceptionService*

```
3 references
public PagedList<Exception> Get(ExceptionQueryFilter filters)
{//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
    if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.ConsultarTodos).FirstOrDefault()
    {
        UnauthorizedBusinessException ex = new UnauthorizedBusinessException(_unitAdmin.exceptionRepository.GetByCo
        throw ThrowException(ex, LogEvents.InsufficientPermissions, ex.Message);
    }

    filters.pageNumber = filters.pageNumber == 0 ? _paginationOptions.DefaultPageNumber : filters.pageNumber;
    filters.pageSize = filters.pageSize == 0 ? _paginationOptions.DefaultPageSize : filters.pageSize;

    var exception = _unitAdmin.exceptionRepository.GetAll();

  if (!string.IsNullOrEmpty(filters.search))
  {
      exception = exception.Where(x =>
         !string.IsNullOrEmpty(x.Code) && x.Code.ToLower().Contains(filters.search.ToLower()) ||
         !string.IsNullOrEmpty(x.Name) && x.Name.ToLower().Contains(filters.search.ToLower()) ||
         !string.IsNullOrEmpty(x.Message) && x.Message.ToLower().Contains(filters.search.ToLower())
      );
  }
  if (filters.idlanguage > 0)
      exception = exception.Where(x => x.IdLanguage == filters.idlanguage);

  if (filters.code != null)
      exception = exception.Where(x => !string.IsNullOrEmpty(x.Code) && x.Code.ToLower().Contains(filters.code.ToLower

  if (exception == null)
  {
      NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-1302",
      throw ThrowException(ex, LogEvents.GetItemsNotFound, ex.Message);
  }
  return PagedList<Exception>.Create(exception, filters.pageNumber, filters.pageSize);
}
```

Fuente: el autor.

**Descripción:** Desde el método Get() en la clase ExceptionService se consultan todas las excepciones del sistema, incluso mediante filtros personalizados y se retorna el listado de excepciones resultantes.

**Figura 86.**

*Método Put() para modificar, en la clase de servicio ExceptionService*

```
3 references
public async Task<Exception> Put(Exception entity)
{//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
    if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Actualizar).FirstOrDefault() == null)
    {
        UnauthorizedBusinessException ex = new UnauthorizedBusinessException(_unitAdmin.exceptionRepository.GetByCode("F
        throw ThrowException(ex, LogEvents.InsufficientPermissions, ex.Message);
    }

    entity = SetAuditFields(entity);
    if (entity.Id > 0)
    {
        Exception exception = await _unitAdmin.exceptionRepository.GetById(entity.Id);

        if (exception != null && exception.Active == true)
        {
            if(entity.Code != exception.Code)
            {
                BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCod
                throw ThrowException(ex, LogEvents.UpdateItem, ex.Message);
            }

            if (await _unitAdmin.languageRepository.GetById(entity.IdLanguage) == null)
            {
                NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("A
                throw ThrowException(ex, LogEvents.GetItemNotFound, ex.Message);
            }

            if (_unitAdmin.exceptionRepository.ExceptionExists(entity.Id, entity.IdLanguage, entity.Code))
            {
                BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.Get
                throw ThrowException(ex, LogEvents.UpdateItem, ex.Message);
            }
            }
            else
            {
                NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("
                throw ThrowException(ex, LogEvents.UpdateItemNotFound, ex.Message);
            }
        }
        else
        {
            NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-1
            throw ThrowException(ex, LogEvents.UpdateItemNotFound, ex.Message);
        }
        await _unitAdmin.exceptionRepository.UpdateAsync(entity);
        await _unitAdmin.SaveChangesAsync();
        return entity;
    }
```

Fuente: el autor.

**Descripción:** El método Put() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta modificación de excepciones en el sistema. Se valida que la excepción a modificar exista y este activa, que no se cambie el código original de la excepción y si ya existe previamente la excepción por idioma y código indicado.

**Figura 87.**

*Función ResetExceptions() para restablecer las excepciones a idiomas base*

```
Bext.Identity.Solution.Bsr.01 - ExceptionRepository.cs
ExceptionRepository.cs
Bext.Identity.Solution.Infraestructure | Bext.Identity.Solution.Infraestructure.Repositories.ExceptionRepository | ExceptionRepository(BextProductoBSRXContext context)

50
51          // Función que restablece las excepciones a los idiomas base (Español, Inglés, Francés)
            2 references
52          public int ResetExceptions(int idUpdateUser, string updateMachine)
53          {
54              // Crea un 'DbCommand' configurado para ejecutar esta consulta
55              using (var cmd = _entities.CreateDbCommand())
56              {
57                  // El nombre del SP que se desea ejecutar y el tipo de comando a ejecutar
58                  cmd.CommandText = "[dbo].[SP_ResetExceptions]";
59                  cmd.CommandType = CommandType.StoredProcedure;
60
61                  // Los parámetros a enviar y que recibirá el procedimiento almacenado
62                  cmd.Parameters.Add(new SqlParameter("@idUpdateUser", SqlDbType.Int) { Value = idUpdateUser });
63                  cmd.Parameters.Add(new SqlParameter("@updateMachine", SqlDbType.NVarChar) { Value = updateMachine });
64
65                  // Abre una conexión a la base de datos con la configuración especificada, si no lo esta
66                  if (cmd.Connection.State != ConnectionState.Open)
67                      cmd.Connection.Open();
68
69                  // Ejecuta el SP y retorna la cantidad de filas modificadas
70                  using (var dataReader = cmd.ExecuteReader())
71                      return dataReader.RecordsAffected;
72              }
73          }
74      }
75  }

100 %   No issues found   Ln: 1   Ch: 1   SPC   CRLF
```

Fuente: el autor.

**Descripción:** Desde el método de tipo Patch en la clase ExceptionService se llama a la función *ResetExceptions()* la cual se encarga de ejecutar un procedimiento almacenado en base de datos, enviándole dos parámetros, el identificador del usuario que ejecuto la petición y la dirección IP. El procedimiento almacenado al ejecutarse, actualiza los nombres y mensajes de las excepciones por código e idioma en la base de datos, restableciéndolos a la forma original en los idiomas español, francés e inglés.

**Figura 88.**

*Clase Entidad del desarrollo del caso de uso de administración de esquemas*

```
5   namespace Bext.Identity.Solution.Core.Entities
6   {
        50 references
7       public partial class Schema : BaseEntity
8       {
            0 references
9           public Schema()
10          {
11              Attributes = new HashSet<Attribute>();
12              Domains_Schemas = new HashSet<Domain_Schema>();
13          }
14
            4 references
15          public int IdClient { get; set; }
            12 references
16          public int IdSchemaType { get; set; }
            9 references
17          public string DomainAddress { get; set; }
            7 references
18          public string Endpoint { get; set; }
            9 references
19          public string TenantId { get; set; }
            1 reference
20          public bool Ssl { get; set; }
21
            1 reference
22          public virtual SchemaType IdSchemaTypeNavigation { get; set; }
            1 reference
23          public virtual Client IdClientNavigation { get; set; }
            9 references
24          public virtual ICollection<Attribute> Attributes { get; set; }
            6 references
25          public virtual ICollection<Domain_Schema> Domains_Schemas { get; set; }
26      }
27  }
100 %   No issues found                                   Ln: 7   Ch: 32   SPC   CRLF
```

Fuente: el autor.

**Descripción:** Se muestran cada una de las propiedades requeridas de la clase Schema para el desarrollo del caso de uso de administración de esquemas. Se permite ingresar una dirección de dominio, punto final, tentant Id de Azure y si utiliza certificado SSL.

La propiedad IdClient es una llave foránea que se deriva de la clase modelo/entidad Client, la cual tiene una relación de uno a muchos hacia esquemas.

La propiedad IdSchemaType es una llave foránea que se deriva de la clase modelo/entidad SchemaType, la cual tiene una relación de uno a muchos hacia esquemas.

Esta clase Schema tiene una relación de uno a muchos atributos (Attribute) y otra relación de uno a muchos esquemas por dominio (Domain_Schema).

**Figura 89.**

*Método Post() para crear, en la clase de servicio SchemaService*

```
4 references
public async Task<Schema> Post(Schema entity)
{//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
    if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Crear).FirstOrDefault() == null)
    {
        UnauthorizedBusinessException ex = new UnauthorizedBusinessException(_unitAdmin.exceptionRepository.GetByCode("F
        throw ThrowException(ex, LogEvents.InsufficientPermissions, ex.Message);
    }

    entity = SetAuditFields(entity);

    if (_unitAdmin.schemaRepository.GetSchema(entity.DomainAddress))
    {
        BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-31
        throw ThrowException(ex, LogEvents.InsertItem, ex.Message);
    }

    SchemaType schemaType = await _unitAdmin.schemaTypeRepository.GetById(entity.IdSchemaType);

    if (schemaType != null && schemaType.Active == true)
    {
        if (entity.IdSchemaType == 2 && entity.TenantId == null)
        {
            BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCode("A
            throw ThrowException(ex, LogEvents.InsertItem, ex.Message);
        }

        if (entity.IdSchemaType == 1 && entity.TenantId != null || entity.IdSchemaType == 3 && entity.TenantId != null)
        {
            BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCode("A
            throw ThrowException(ex, LogEvents.InsertItem, ex.Message);
        }
    }
    else
    {
        NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-3106",
        throw ThrowException(ex, LogEvents.GetItemNotFound, ex.Message);
    }
    await _unitAdmin.schemaRepository.Add(entity);
    await _unitAdmin.SaveChangesAsync();
    return entity;
}
```

Fuente: el autor.

**Descripción:** El método Post() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta creación de esquemas en el sistema. Se valida que solo para el tipo de esquema Azure AD se ingrese tentant Id y no para los tipos de esquemas LDAP y Web Service, también que no se repitan esquemas con la misma dirección de dominio.

**Figura 90.**

*Método Put() para modificar, en la clase de servicio SchemaService*

```
145     public async Task<Schema> Put(Schema entity)
146     {//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
147         if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Actualizar).FirstOrDefault() == null)
148         {
149             UnauthorizedBusinessException ex = new UnauthorizedBusinessException(_unitAdmin.exceptionRepository.GetByCode("F
150             throw ThrowException(ex, LogEvents.InsufficientPermissions, ex.Message);
151         }
152
153         entity = SetAuditFields(entity);
154     if (entity.Id > 0)
155     {
156         Schema schema = await _unitAdmin.schemaRepository.GetById(entity.Id);
157
158         if (schema != null && schema.Active == true)
159         {
160             if (_unitAdmin.schemaRepository.GetDomainAddress(entity.Id, entity.DomainAddress))
161             {
162                 BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCode("A
163                 throw ThrowException(ex, LogEvents.UpdateItem, ex.Message);
164             }
165
166             SchemaType schemaType = await _unitAdmin.schemaTypeRepository.GetById(entity.IdSchemaType);
167
168             if (schemaType != null && schemaType.Active == true)
169             {
170                 if (entity.IdSchemaType == 2 && entity.TenantId == null)
171                 {
172                     BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCod
173                     throw ThrowException(ex, LogEvents.UpdateItem, ex.Message);
174                 }
175                 if (entity.IdSchemaType == 1 && entity.TenantId != null || entity.IdSchemaType == 3 && entity.TenantId != nu
176                 {
177                     BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCod
178                     throw ThrowException(ex, LogEvents.UpdateItem, ex.Message);
179                 }
180             }
181             else
182             {
183                 NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-31
184                     throw ThrowException(ex, LogEvents.GetItemNotFound, ex.Message);
185                 }
186             }
187             else
188             {
189                 NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-31
190                 throw ThrowException(ex, LogEvents.UpdateItemNotFound, ex.Message);
191             }
192         }
193         else
194         {
195             NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-3107",
196             throw ThrowException(ex, LogEvents.UpdateItemNotFound, ex.Message);
197         }
198         await _unitAdmin.schemaRepository.UpdateAsync(entity);
199         await _unitAdmin.SaveChangesAsync();
200         return entity;
201     }
```

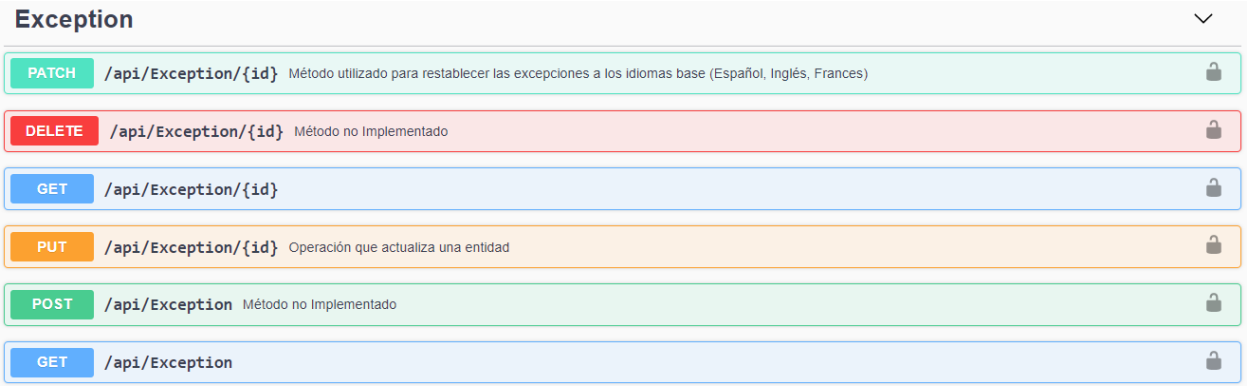
Fuente: el autor.

**Descripción:** El método Put() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta modificación de esquemas en el sistema. Se valida que el esquema a modificar exista y este activo, que solo para el tipo de esquema Azure AD se ingrese tentant Id y no para los tipos de esquemas LDAP y Web Service, también que no se repitan esquemas con la misma dirección de dominio.

- Pruebas de los servicios implementados en desarrollo back-end

**Figura 91.**

*Servicios implementados para el desarrollo de Configuración de Excepciones*




Fuente: el autor.

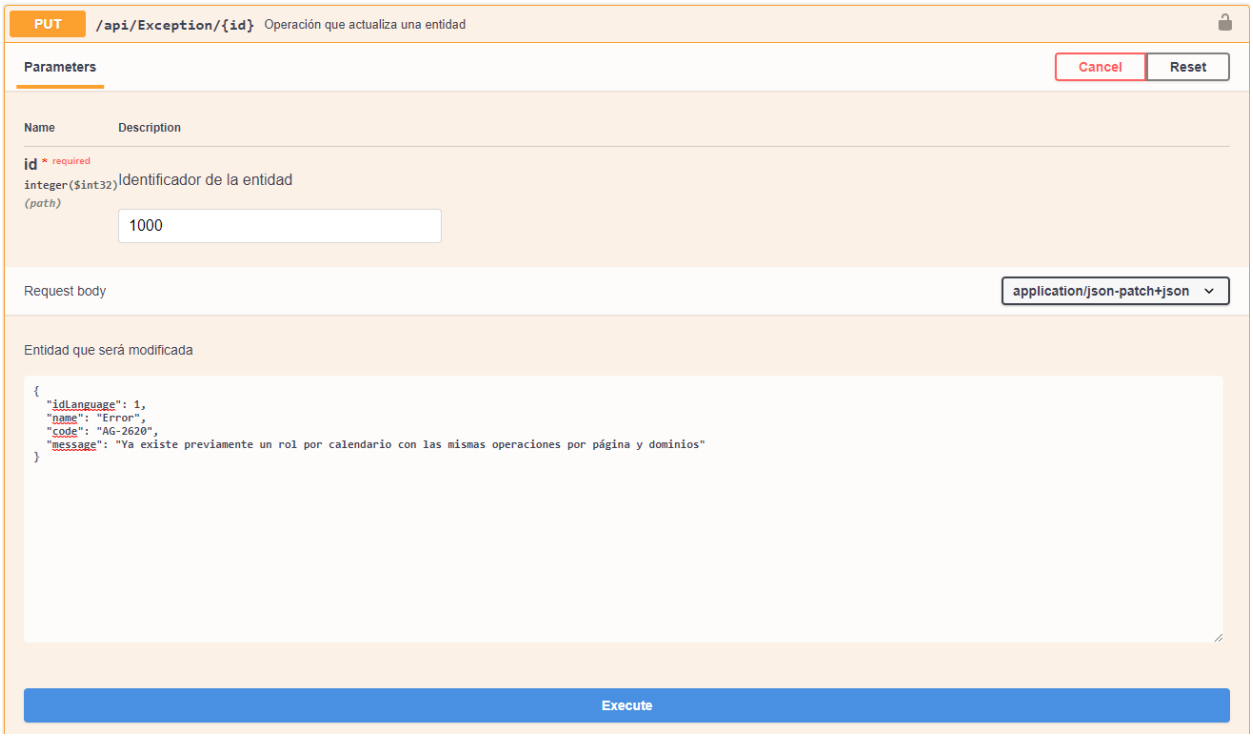
**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestran los servicios implementados para crear (Post), modificar (Put), eliminar (Delete) y obtener (Get) objetos de excepciones, según las reglas de negocio, relacionados al desarrollo del caso de uso de Configuración de mensajes de error/Excepción.

Los mensajes de error/excepción son aquellos que el sistema muestra al usuario al momento en que este realiza peticiones incorrectas o se genera un error interno del sistema que no se pudo resolver.

El administrador podrá personalizar los mensajes por idioma de las excepciones que el sistema mostrará cuando sea necesario.

**Figura 92.**

*Configurando un mensaje de error/excepción a modificar desde Swagger UI*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se indica el identificador del registro a modificar, se envía la petición y se actualiza la configuración de mensaje de error/excepción en base de datos.

**Figura 93.**

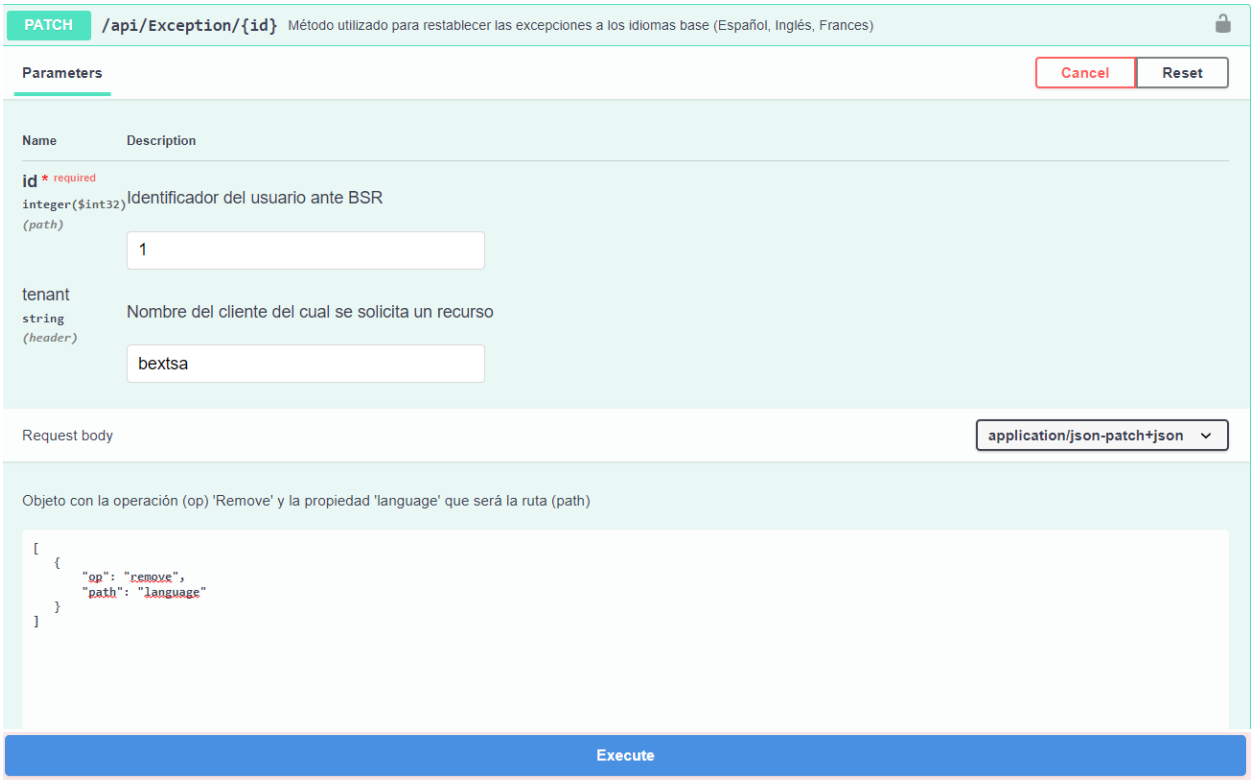
*Respuesta que muestra Swagger UI al modificarse un mensaje de excepción*

Fuente: el autor.

**Descripción:** Desde el back-end se retorna la configuración de excepción modificada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 94.**

*Petición para restablecer los mensajes de error/excepción a idiomas base*



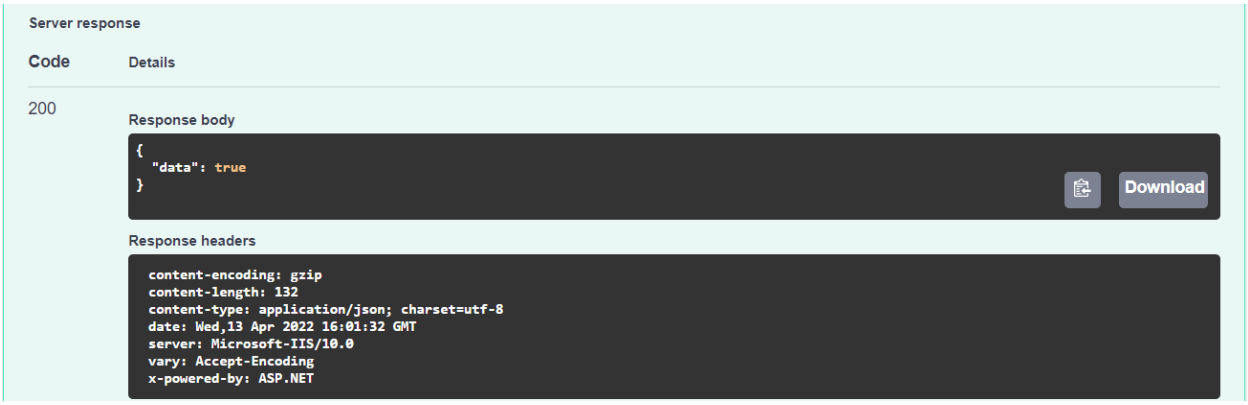

Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa la operación a realizar (replace) y la ruta (language) que será tenida en cuenta para realizar el proceso de restablecer los mensajes de error/excepción a los idiomas base (español, inglés, francés), desde un arreglo de objetos en formato JSON.

Como parámetros, el identificador del usuario que realiza la petición y el nombre del cliente.

**Figura 95.**

*Respuesta que muestra Swagger UI al restablecerse a los idiomas base*




Fuente: el autor.

**Descripción:** Desde el back-end se retorna un valor de tipo lógico (true), indicando que todas las excepciones del sistema en base de datos se restablecieron correctamente a los idiomas base (español, inglés, francés), mostrándose en Swagger UI.

**Figura 96.**

*Servicios implementados para el desarrollo de Administración de Esquemas*

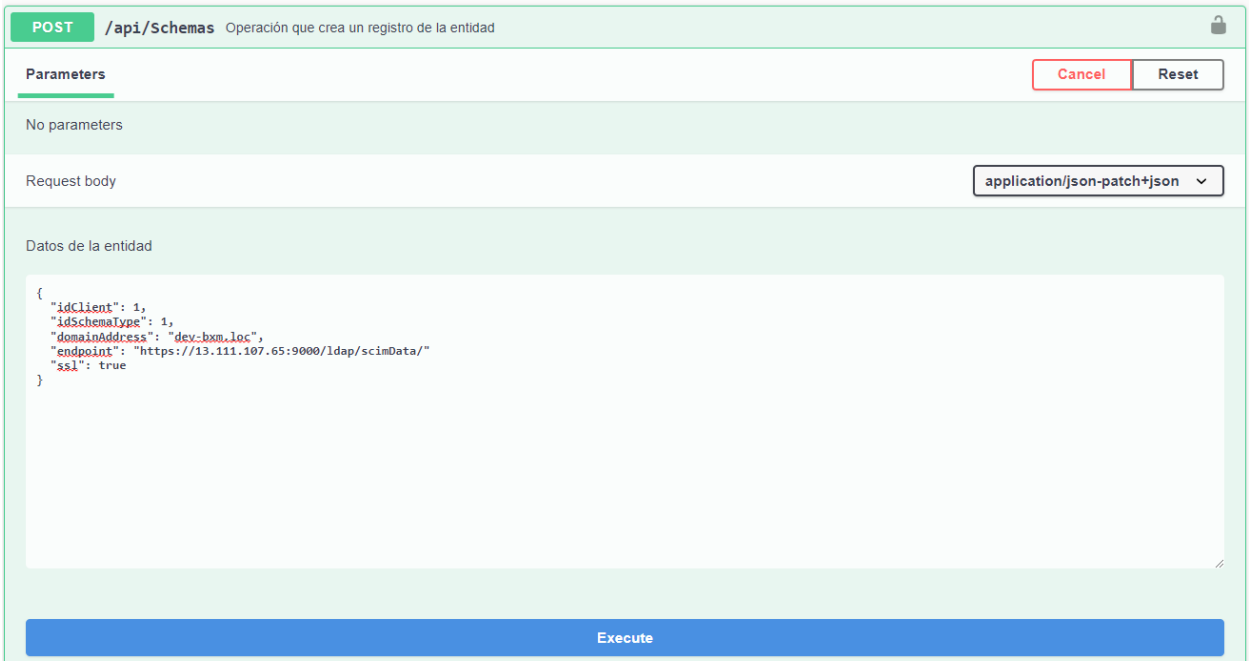
Fuente: el autor.

**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestran los servicios implementados para crear (Post), modificar (Put), eliminar (Delete) y obtener (Get) objetos de esquemas, según las reglas de negocio, relacionados al desarrollo del caso de uso de Administración de Esquema.

Los esquemas son los diferentes repositorios de usuarios, directorios activos, bases de datos o servicio web a los que se podrá conectar el proyecto BSR para leer información, actualizar datos del usuario y cambiar la contraseña de la cuenta.

**Figura 97.**

*Petición para configurar un esquema a crear desde Swagger UI*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se envía la petición y se guarda la configuración de esquema en base de datos.

**Figura 98.**

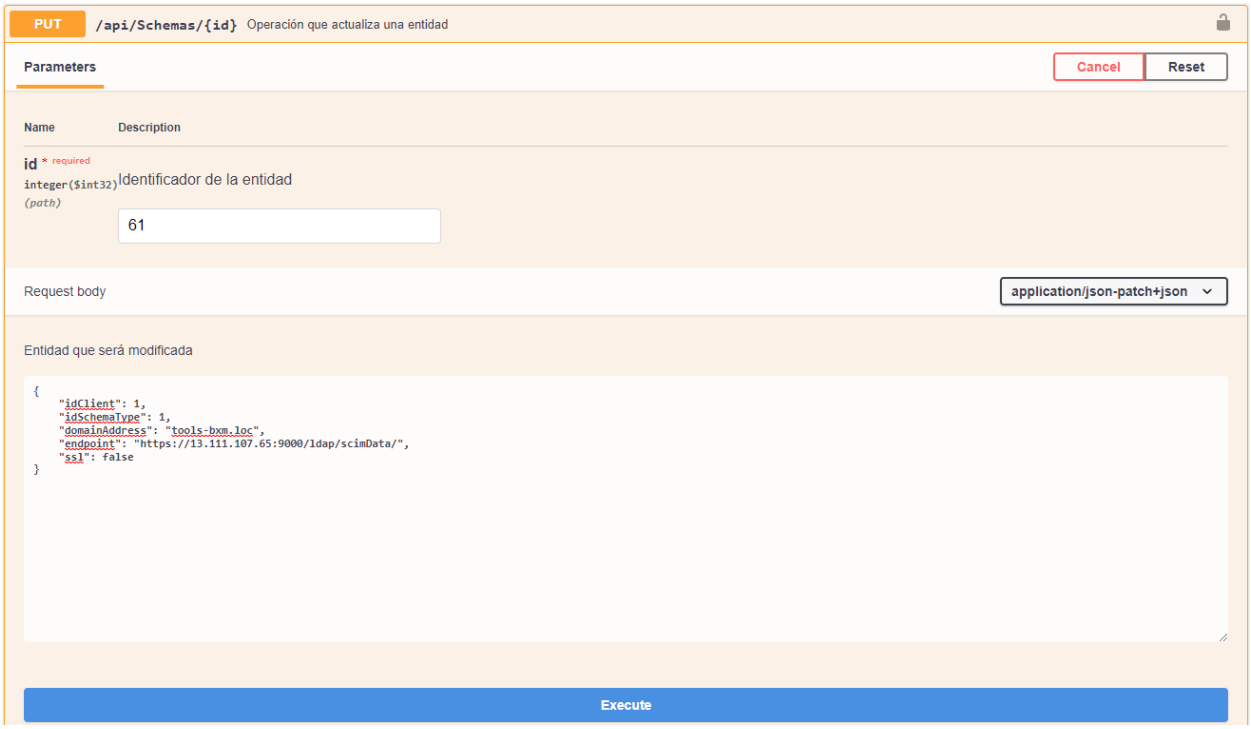
*Respuesta que muestra Swagger UI al crearse una configuración de esquema*

Fuente: el autor.

**Descripción:** Desde el back-end se retorna la configuración de esquema creada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 99.**

*Configurando un esquema a modificar desde Swagger UI*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se indica el identificador del registro a modificar, se envía la petición y se actualiza la configuración de esquema en base de datos.

**Figura 100.**

*Respuesta que muestra Swagger UI al modificarse un esquema*

Fuente: el autor.

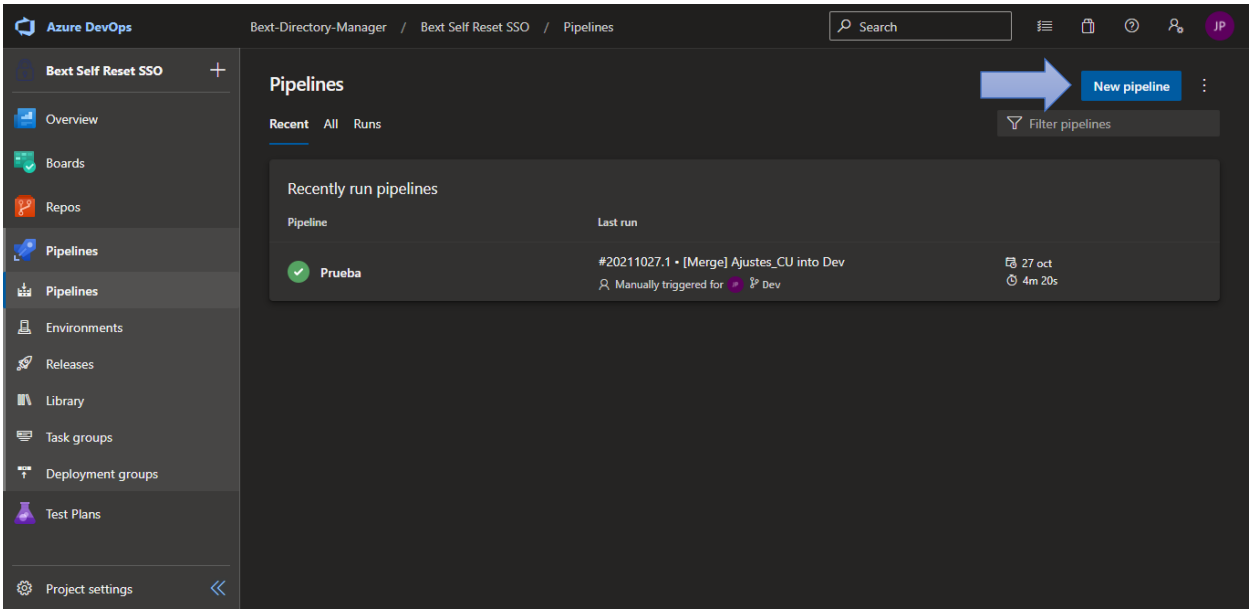
**Descripción:** Desde el back-end se retorna la configuración de esquema modificada correctamente en base de datos, mostrándose en Swagger UI.

- Despliegue de soluciones para ambientes QA y de producción en la nube Microsoft Azure

Mediante la integración continua (CI) se pueden ejecutar automáticamente pruebas unitarias cada vez que los desarrolladores integran sus desarrollos a la rama principal de un repositorio, esto con el fin de identificar posibles errores de integración y solucionarlos. También, se genera una compilación con un archivo resultante, el cual se usará en procesos de entrega continua.

**Figura 101.**

*Añadir nueva tubería en Azure DevOps (CI/CD)*



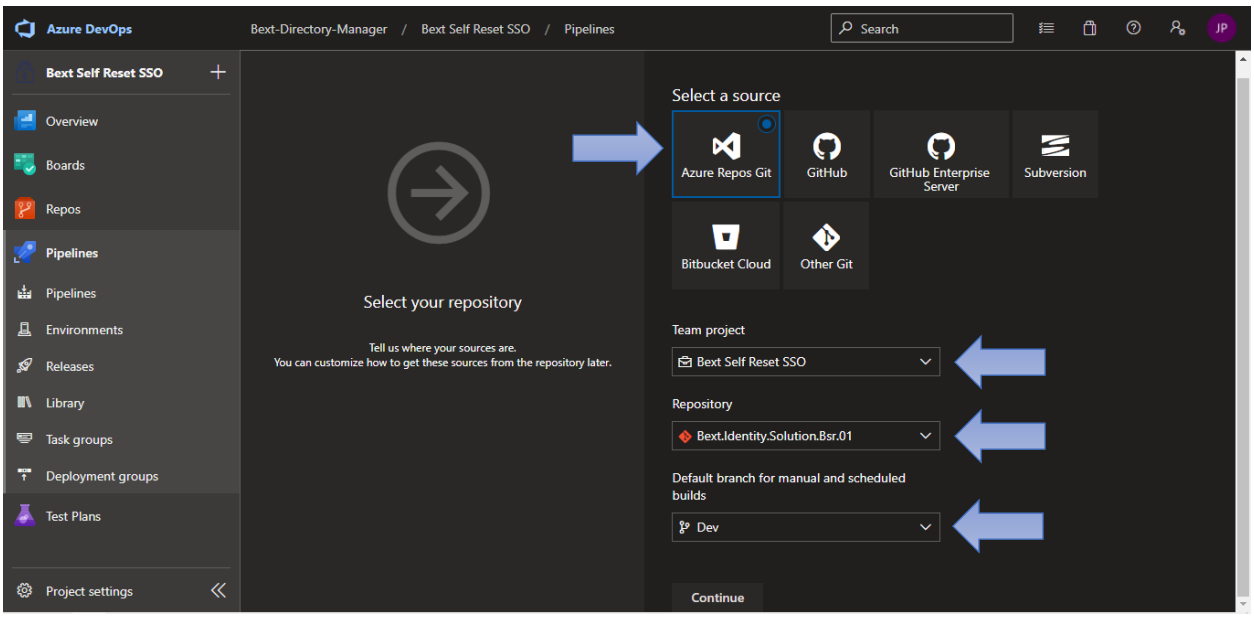

Fuente: el autor.

**Descripción:** Se inicia la creación de una nueva tubería desde el portal Azure DevOps en la sección “Pipelines”, mediante la cual se implementará automáticamente el proceso de integración continua.

Mediante la entrega continua (CD) se entregan compilaciones finales, las últimas características probadas y disponibles para usar a usuarios finales y clientes de manera "transparente" y parcialmente automática. Por lo general, un desarrollador líder siempre decidirá si aprueba la ejecución de la entrega continua cada vez que esté disponible, siendo así un proceso en su mayoría automatizado. La entrega continua trabaja en conjunto con la integración continua, haciendo uso de un archivo (artefacto) generado después de la compilación de la solución durante la integración continua, este artefacto será el que se implemente a nivel de infraestructura durante el proceso de entrega continua.

**Figura 102.**

*Seleccionar proyecto, repositorio y rama de compilación (CI/CD)*

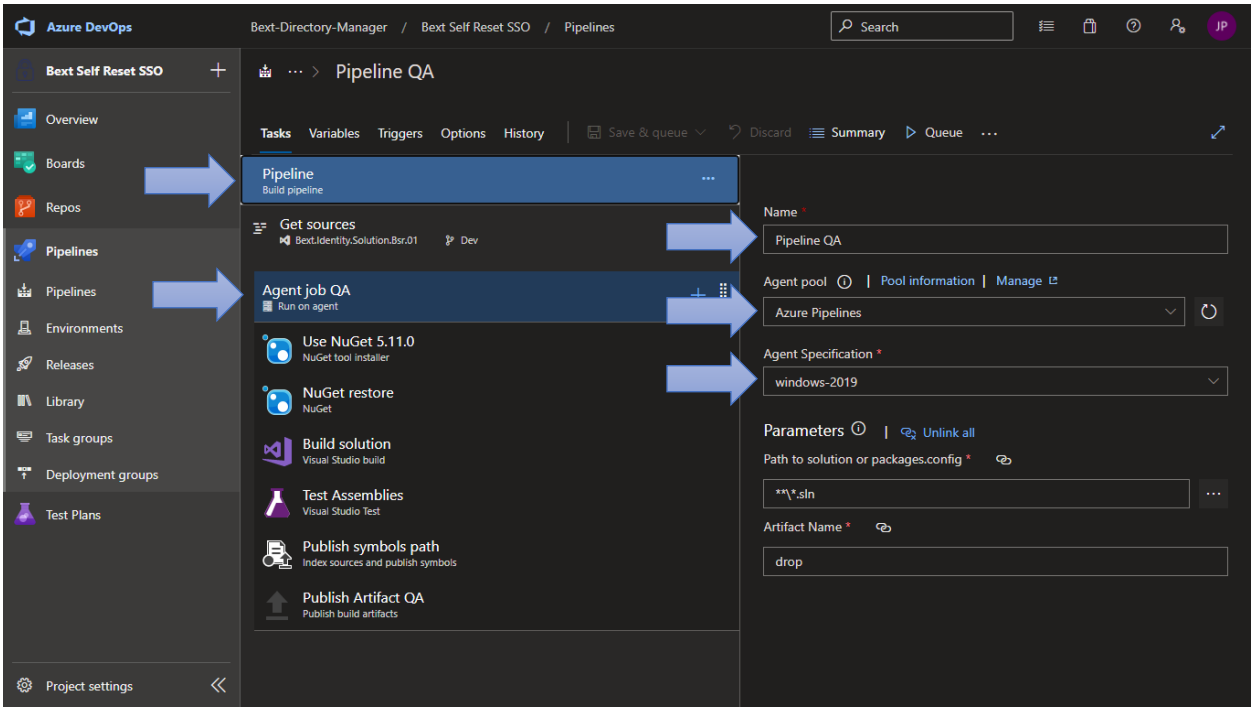



Fuente: el autor.

**Descripción:** Se selecciona una fuente de código, proyecto, repositorio y rama en la que se ejecutara la integración continua (CI)

**Figura 103.**

*Configuración general de la nueva tubería (CI/CD)*

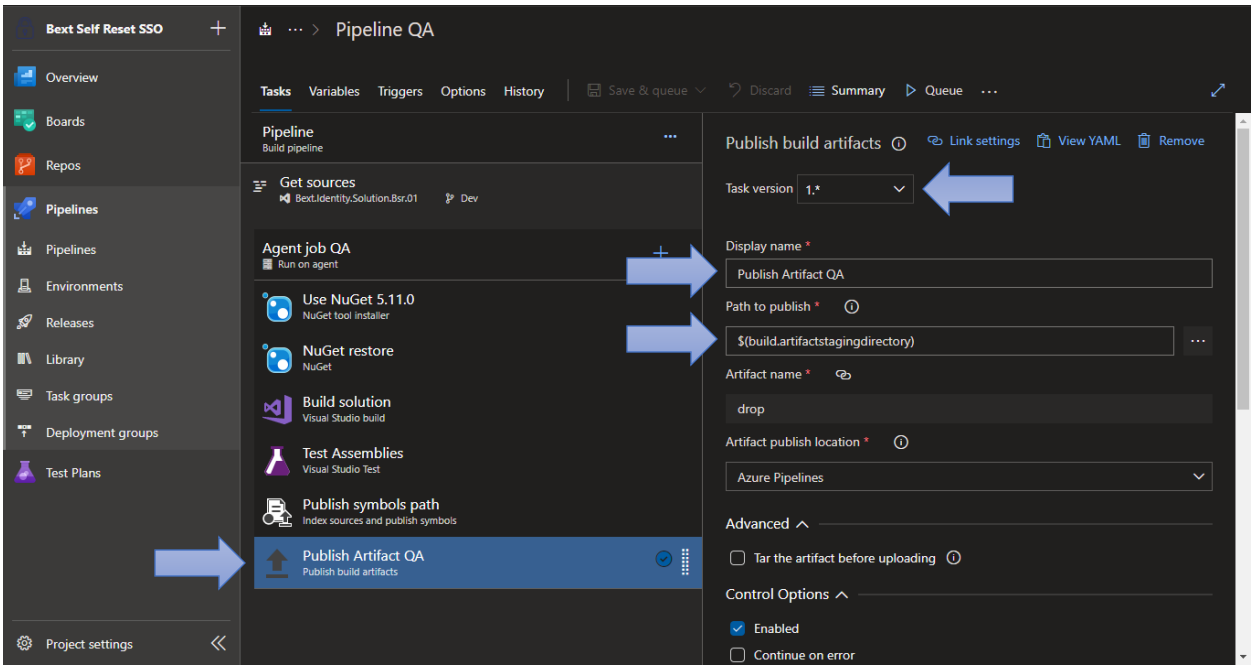



Fuente: el autor.

**Descripción:** Se selecciona el servicio a utilizar (Azure Pipelines) y la imagen de máquina virtual que se usara en cada trabajo de canalización.

**Figura 104.**

*Configuración del artefacto (CI/CD)*



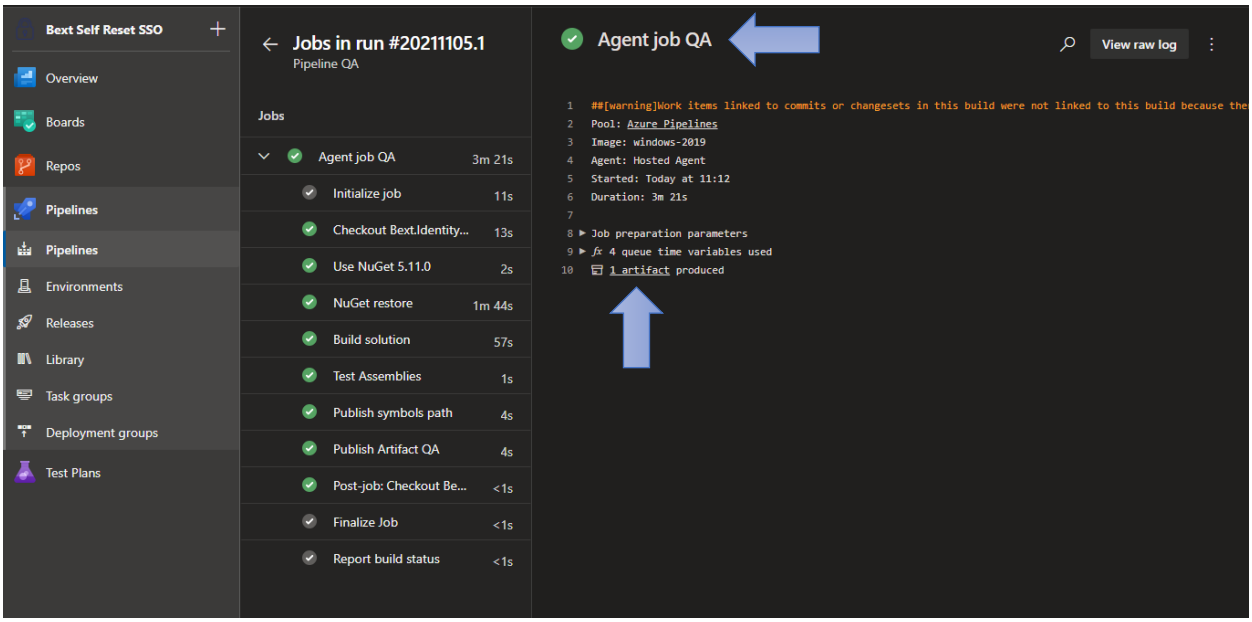

Fuente: el autor.

**Descripción:** Se selecciona la última versión para la tarea de crear el artefacto, se selecciona la ubicación donde se alojará el artefacto ya creado después de compilarse la solución.

El artefacto generado se deberá ubicar y seleccionar en la etapa de configurar la entrega continua (release), ya que de no hacerse no se completará correctamente la entrega.

**Figura 105.**

*La nueva tubería al ejecutarse automáticamente (CI/CD)*




Fuente: el autor.

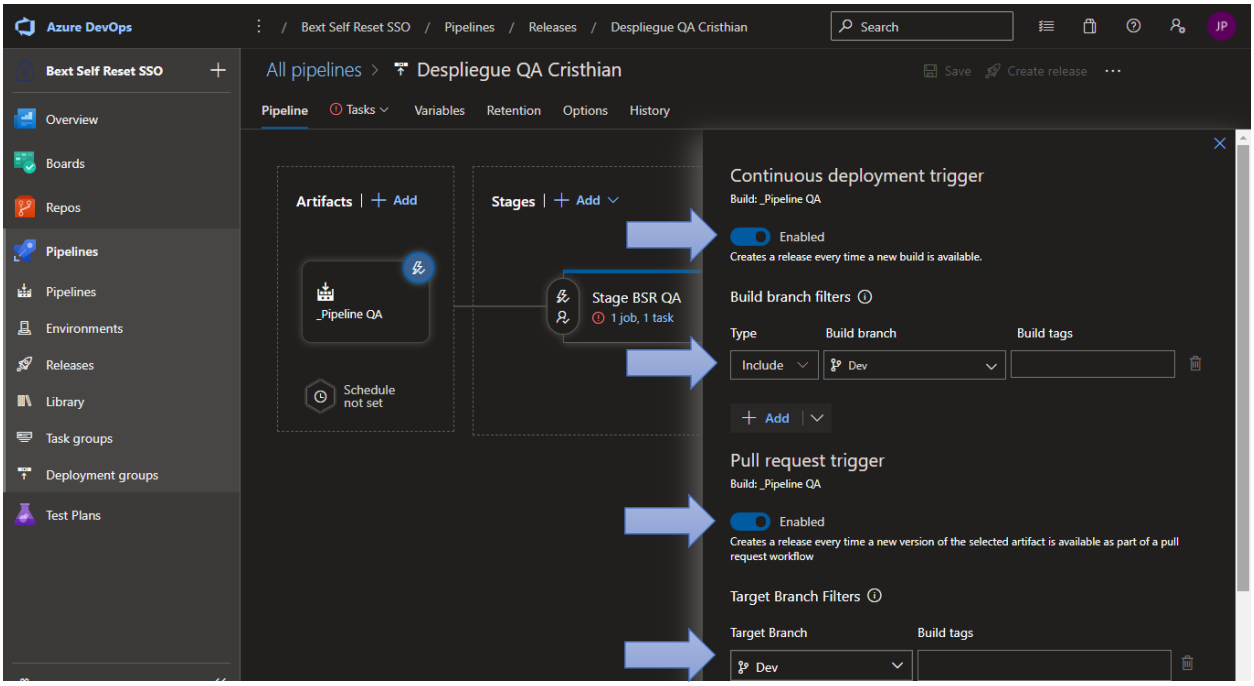
**Descripción:** Se muestra el resultado de ejecutarse todo el proceso de integración continua de manera exitosa.

Se puede acceder al artefacto generado y que será usado en la entrega continua.

En este caso no había pruebas unitarias configuradas para ejecutarse.

**Figura 106.**

*Configurando un nuevo proceso de liberación (CI/CD)*



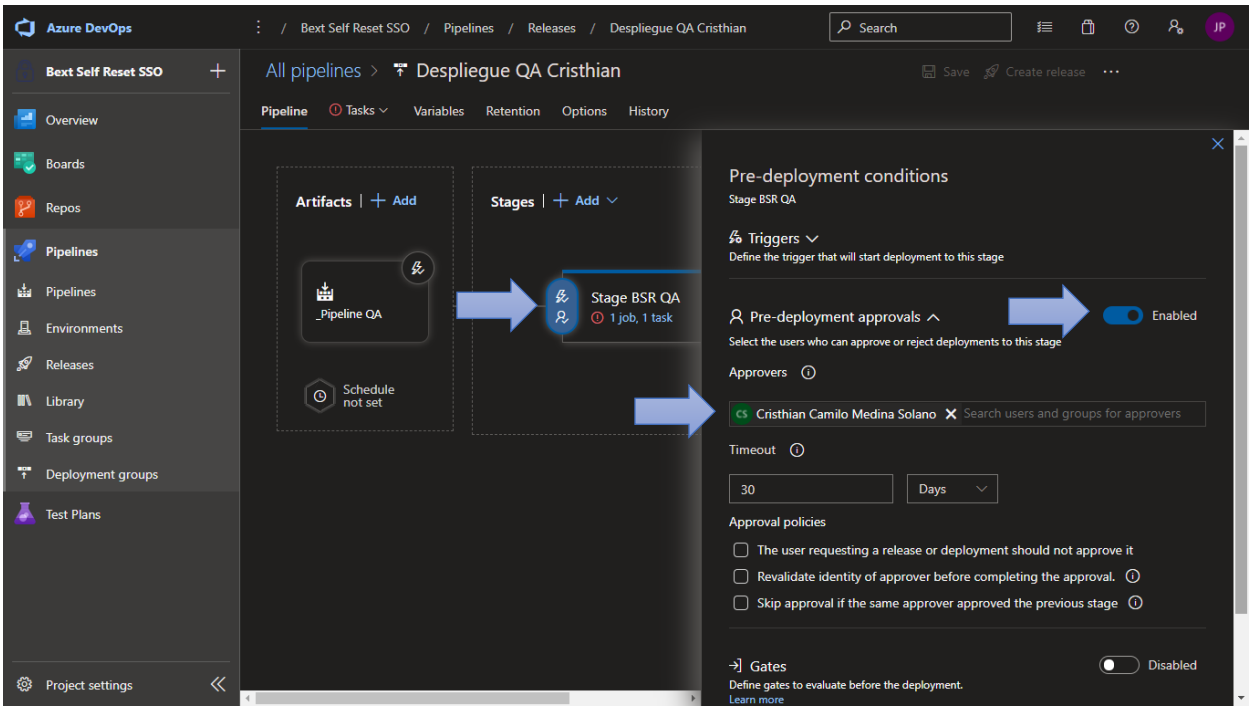

Fuente: el autor.

**Descripción:** Se observan las diferentes acciones (triggers) que generaran un evento de entrega continua.

La primera opción es cuando hay un artefacto disponible producto de una compilación en la integración continua a la rama Dev, en última opción, cuando se completa una solicitud de Pull Request a la rama destino Dev, por lo que, se generara un artefacto disponible para usar.

**Figura 107.**

*Configurando condiciones previas al despliegue (CI/CD)*




Fuente: el autor.

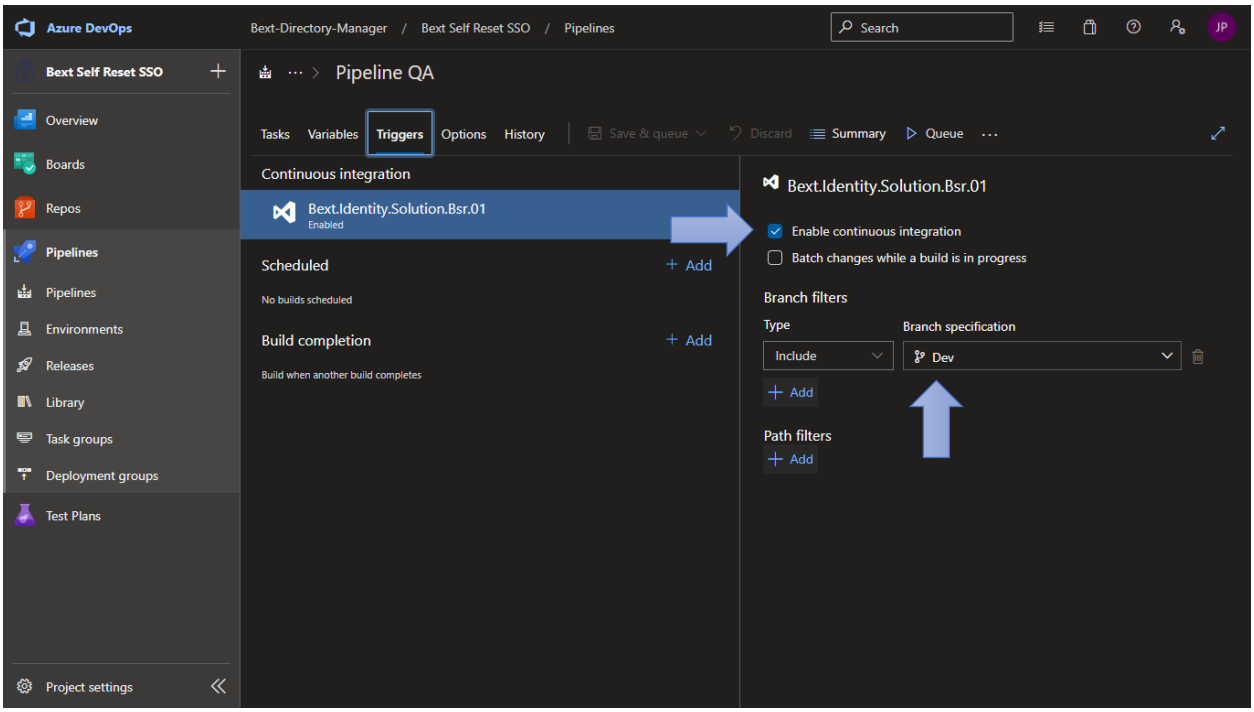
**Descripción:** Se configuran las precondiciones previas a la entrega continua.

En este caso, se requiere que una persona específica apruebe el proceso de entrega continua que ya está preparado para ejecutarse e implementarse.

La persona al aprobar el proceso dará “vía libre” para que se desplieguen los últimos cambios aplicados a la rama “Dev”, en el sitio de pruebas de la web API.

**Figura 108.**

*Habilitar la integración continua (CI/CD)*



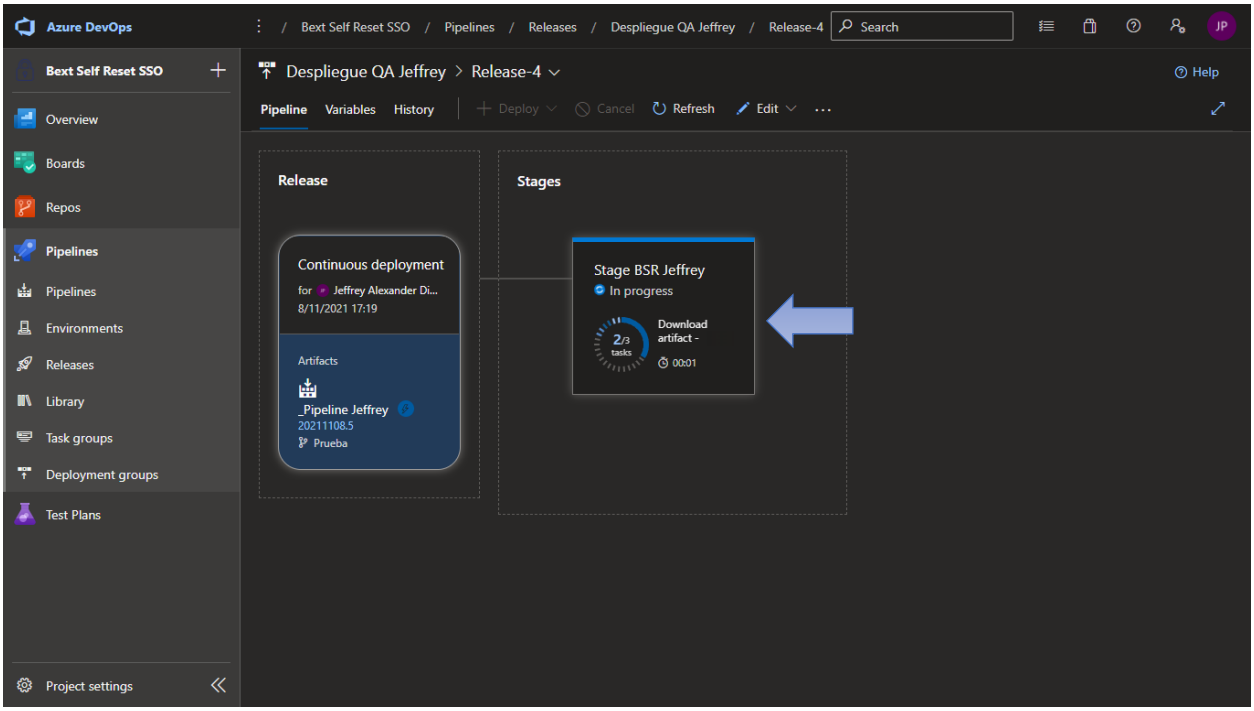

Fuente: el autor.

**Descripción:** Se habilita el proceso automático de integración continua (CI) aplicado a la rama Dev de desarrollo, ya que, si este proceso no está habilitado, no se podrá ejecutar el proceso de entrega continua (CD) correctamente al no disponerse del artefacto generado durante la integración continua.

La integración continua es una condición previa a la ejecución de la entrega continua.

**Figura 109.**

*La entrega continua en ejecución (CI/CD)*



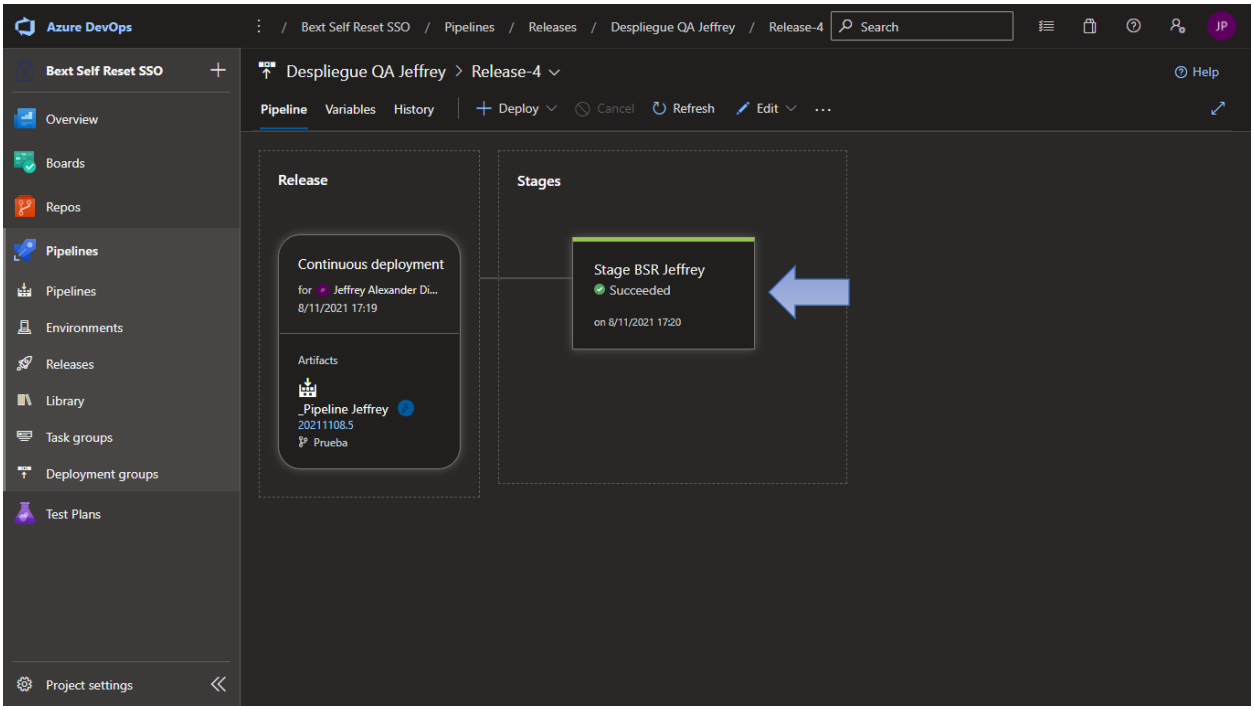

Fuente: el autor.

**Descripción:** Después de que se habilita y aprueba la ejecución de la entrega continua desde la plataforma de Azure DevOps, se procede a implementar los últimos cambios y nuevas características funcionales en la Web API expuesta a ser consumida desde Internet.

Este proceso es rápido, no demorando más de unos pocos segundos.

**Figura 110.**

*La entrega continúa realizada correctamente (CI/CD)*



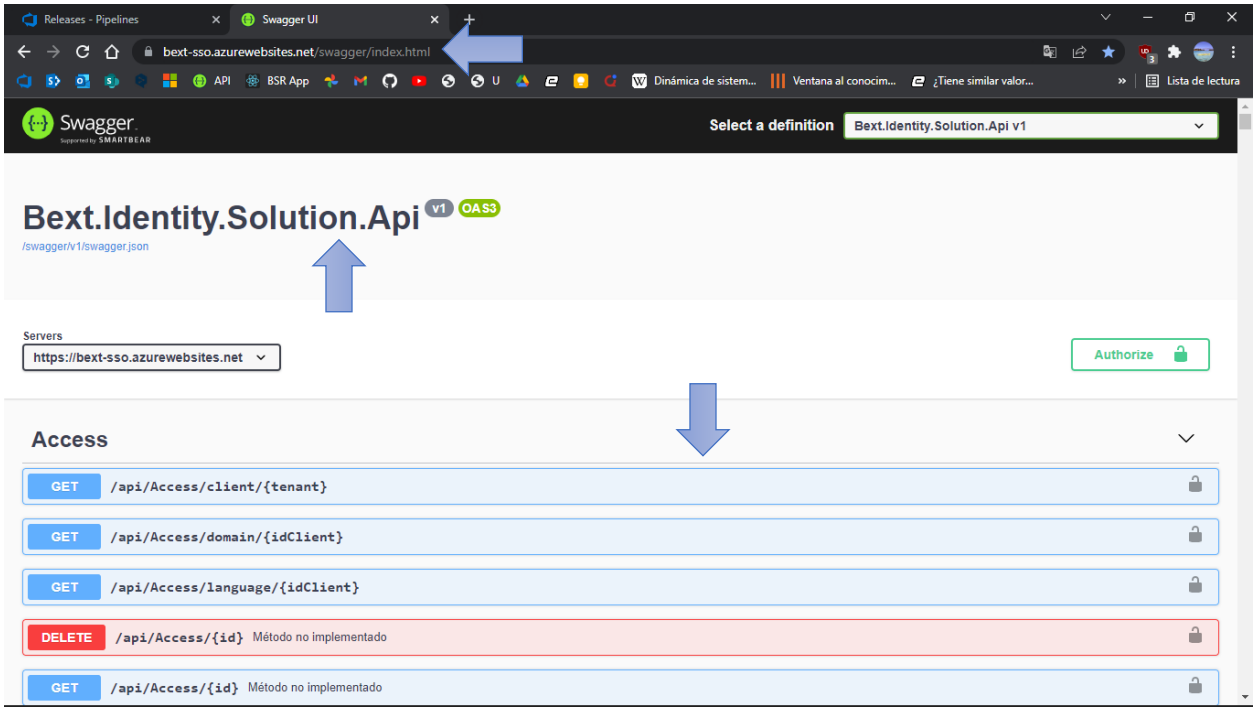

Fuente: el autor.

**Descripción:** Se observa que el proceso de entrega continua se realizó correctamente.

En este momento ya se puede hacer uso de las últimas características y funcionalidades añadidas al sistema a nivel de back-end, consumiendo los diferentes servicios en ambiente de producción o de pruebas QA.

**Figura 111.**

*Acceder a la Web API desde Internet (CI/CD)*




Fuente: el autor.

**Descripción:** Se observa la API en ejecución expuesta a ser consumida desde Internet.

Para el desarrollo del proyecto “Bext Self Reset” se hizo necesario implementar el proceso de integración y entrega continua, principalmente para que el equipo de pruebas QA hiciera pruebas manuales de los diferentes servicios implementados, comprobando el cumplimiento de los requisitos funcionales y reglas de negocio para cada caso de uso.

En este momento la etapa de pruebas manuales de los servicios ya finalizo, ahora la Web API se usa para que los diferentes servicios se puedan consumir desde el desarrollo Front-end del producto.

**6.** En el sexto mes (6 de marzo del 2022 a 5 de abril del 2022) se hizo:

- El estudio y desarrollo del caso de uso [CU33] Configuración Canales de Envío, añadiendo las reglas de negocio y funcionalidades necesarias en el código para permitir que un usuario administrador del sistema pueda gestionar la configuración de los diferentes Endpoint y las cuentas de servicios de terceros que se integraran a BSR SSO, para procesos de doble factor y los desafíos por cliente, para los diferentes procesos de la plataforma BSR SSO. Permitirá que los clientes puedan integrar sus cuentas de servicio para la generación de claves dinámicas (OTP) usados en procesos de doble factor de autenticación y los desafíos por dominio.
- El estudio y desarrollo del caso de uso [CU109] Asociar preguntas de seguridad, añadiendo las reglas de negocio y funcionalidades necesarias en el código para permitir que un usuario final (básico) del sistema pueda realizar el proceso de gestionar las múltiples preguntas de seguridad que desee asociar con su respuesta según políticas del dominio al que se encuentre asociado, pudiendo realizar las operaciones de crear, editar y eliminar la configuración de preguntas asociadas en la plataforma BSR SSO. Las preguntas de seguridad que el usuario asocie con su respuesta serán solicitadas por el sistema cuando se requiera validar la identidad del usuario
- El estudio y desarrollo del caso de uso [CU28] Configuración de Desafíos, añadiendo las reglas de negocio y funcionalidades necesarias en el código para permitir que un usuario administrador del sistema pueda gestionar la configuración de los diferentes desafíos por dominio, con la finalidad de que un

usuario pueda validar su identidad al momento de querer realizar ciertos procesos y /o acceder a ciertas páginas del sistema, ya sea respondiendo preguntas de seguridad asociadas, ingresando código de verificación, clave dinámica, lectura de código QR; pudiendo realizar las operaciones de crear y editar la configuración de desafíos en la plataforma BSR SSO.

- El desarrollo Front-end mediante React.js se tenía acordado realizar, pero finalmente no se hizo debido a que la empresa contrato personas especializadas en desarrollo Front-end, por lo que ya no me requerían. Esta actividad se reemplazó por desarrollo back-end.
- La configuración de despliegue de soluciones para ambientes QA y de producción en la nube Microsoft Azure se tenía acordado realizar, pero no fue necesario, ya que todo este proceso se culminó en el mes anterior. Esta actividad se reemplazó por desarrollo back-end.
- La implementación y configuración de servidores de identidad se tenía acordado realizar, pero no fue necesario, ya que todo este proceso se culminó en el mes anterior. Esta actividad se reemplazó por desarrollo back-end.
- La actualización del documento [BSR] Diccionario Base de Datos.docx se tenía acordado realizar, no se realizó debido a que se consideró una tarea no prioritaria, reemplazando esta actividad por el desarrollo back-end.

A continuación, se comparten algunas vistas que detallan las actividades realizadas durante este mes:

- Desarrollo back-end de los casos de uso

**Figura 112.**

*Clase Entidad del desarrollo del caso de uso de Configurar Canales de Envío*

```
using Bext.Identity.Solution.Domain.Entities;

namespace Bext.Identity.Solution.Core.Entities
{
    27 references
    public partial class NotificationChannel : BaseEntity
    {
        7 references
        public int IdNotificationType { get; set; }
        7 references
        public int IdClient { get; set; }
        3 references
        public string EndPointProvider { get; set; }
        3 references
        public string UserProvider { get; set; }
        1 reference
        public string PasswordProvider { get; set; }
        3 references
        public string NameProvider { get; set; }

        1 reference
        public virtual NotificationType IdNotificationTypeNavigation { get; set; }
        1 reference
        public virtual Client IdClientNavigation { get; set; }
    }
}
```

Fuente: el autor.

**Descripción:** Se muestran cada una de las propiedades requeridas de la clase NotificationChannel para el desarrollo del caso de uso de Configuración de Canales de Envío. Se permite ingresar el punto final del proveedor y datos asociados a la cuenta del proveedor de servicios.

La propiedad IdNotificationType es una llave foránea que se deriva de la clase modelo/entidad NotificationType, la cual tiene una relación de uno a muchos hacia canales de envío.

La propiedad IdClient es una llave foránea que se deriva de la clase modelo/entidad Client, la cual tiene una relación de uno a muchos hacia canales de envío.

**Figura 113.**

*Método Post() para crear, en la clase de servicio NotificationChannelService*

```
Bext.Identity.Solution.Bsr.01 - NotificationChannelService.cs
NotificationChannelService.cs
Bext.Identity.Solution.Core | Bext.Identity.Solution.Core.Services.NotificationChannelService | _userService

106
    4 references
107 public async Task<NotificationChannel> Post(NotificationChannel entity)
108 {//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
109     if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Crear).FirstOrDefault() == null)
110     {
111         UnauthorizedBusinessException ex = new UnauthorizedBusinessException(_unitAdmin.exceptionRepository.GetByCode("
112         throw ThrowException(ex, LogEvents.InsufficientPermissions, ex.Message);
113     }
114     else
115     {
116         _userService.AddLogOperationUser(_userLogged.userId, (int)Enumerations.OperationsType.Crear, _currentPage.ToStr
117     }
118
119     entity = SetAuditFields(entity);
120     if (_unitAdmin.notificationChannelRepository.GetNotificationChannel(entity.IdNotificationType, entity.IdClient))
121     {
122         BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-
123         throw ThrowException(ex, LogEvents.InsertItem, ex.Message);
124     }
125     await _unitAdmin.notificationChannelRepository.Add(entity);
126     await _unitAdmin.SaveChangesAsync();
127     return entity;
128 }
129
```

Fuente: el autor.

**Descripción:** El método Post() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta creación de canales de envío. Se valida que no se repitan canales de envío por tipo de notificación y cliente.

**Figura 114.**

*Método Put() para modificar, en la clase de servicio NotificationChannelService*

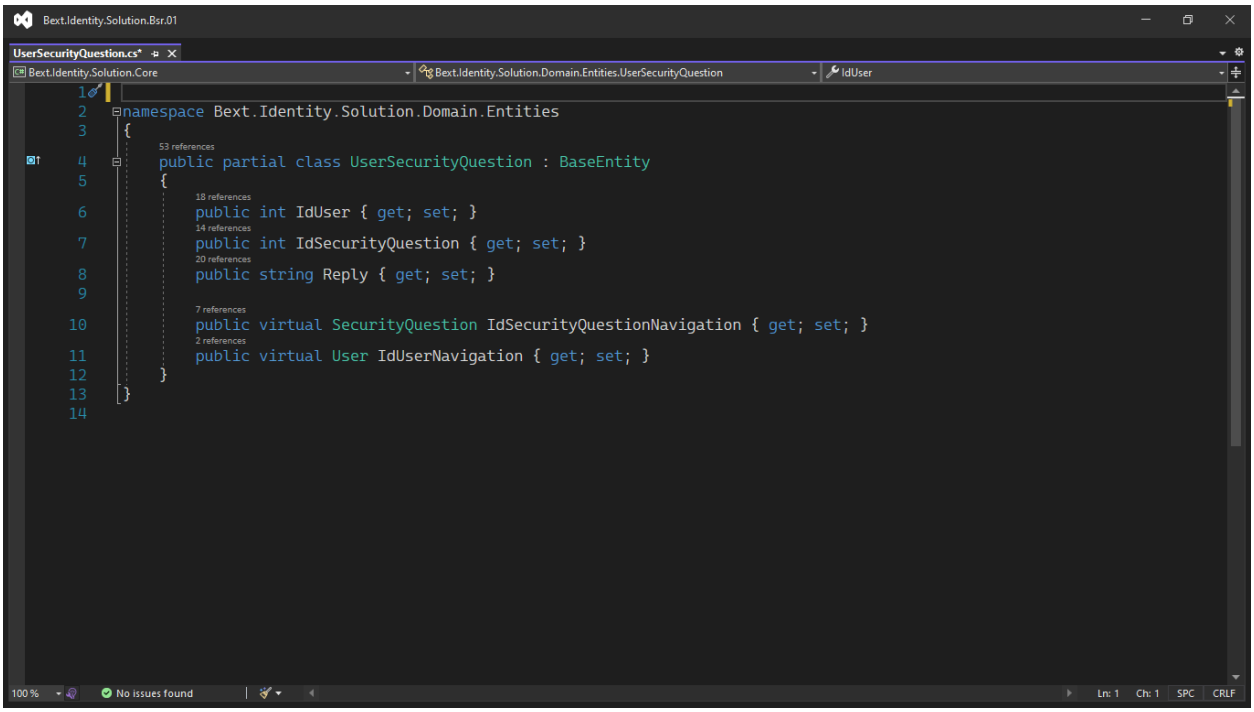
```
128  public async Task<NotificationChannel> Put(NotificationChannel entity)
129  {//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
130      if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Actualizar).FirstOrDefault() == null)
131      {
132          UnauthorizedBusinessException ex = new UnauthorizedBusinessException(_unitAdmin.exceptionRepository.GetByCode("FC
133          throw ThrowException(ex, LogEvents.InsufficientPermissions, ex.Message);
134      }
135      else
136      {
137          _userService.AddLogOperationUser(_userLogged.userId, (int)Enumerations.OperationsType.Actualizar, _currentPage.Tc
138      }
139
140      entity = SetAuditFields(entity);
141      if (entity.Id > 0)
142      {
143          NotificationChannel notificationChannel = await _unitAdmin.notificationChannelRepository.GetById(entity.Id);
144
145          if (notificationChannel != null && notificationChannel.Active == true)
146          {
147              if (_unitAdmin.notificationChannelRepository.NotificationChannelExists(entity.Id, entity.IdNotificationType,
148              {
149                  BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCode
150                  throw ThrowException(ex, LogEvents.UpdateItem, ex.Message);
151              }
152          }
153          else
154          {
155              NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-330
156              throw ThrowException(ex, LogEvents.UpdateItemNotFound, ex.Message);
157          }
158              }
159              else
160              {
161                  NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-
162                  throw ThrowException(ex, LogEvents.UpdateItemNotFound, ex.Message);
163              }
164              await _unitAdmin.notificationChannelRepository.UpdateAsync(entity);
165              await _unitAdmin.SaveChangesAsync();
166              return entity;
167          }
168      }
169  }
```

Fuente: el autor.

**Descripción:** El método Put() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta modificación de canales de envío. Se valida que el canal de envío a modificar exista y este activo, que no se repitan canales de envío por tipo de notificación y cliente.

**Figura 115.**

*Clase Entidad del desarrollo del caso de uso de asociar preguntas de seguridad*




Fuente: el autor.

**Descripción:** Se muestran cada una de las propiedades requeridas de la clase UserSecurityQuestion para el desarrollo del caso de uso de asociar preguntas de seguridad. Tiene propiedades para indicar la respuesta, una pregunta y usuario específico.

La propiedad IdUser es una llave foránea que se deriva de la clase modelo/entidad User, la cual tiene una relación de uno a muchos hacia preguntas de seguridad del usuario.

La propiedad IdSecurityQuestion es una llave foránea que se deriva de la clase modelo/entidad SecurityQuestion, la cual tiene una relación de uno a muchos hacía preguntas de seguridad del usuario.

**Figura 116.**

*Método Get() que retorna todas las preguntas de seguridad de los usuarios*

```
3 references
public PagedList<UserSecurityQuestion> Get(UserSecurityQuestionQueryFilter filters)
{//Este código verifica si el usuario tiene permiso para poder acceder a este servicio
    if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.ConsultarTodos).FirstOrDefault() == n
    {
        UnauthorizedBusinessException ex = new UnauthorizedBusinessException(_unitAdmin.exceptionRepository.GetByCode("F
        throw ThrowException(ex, LogEvents.InsufficientPermissions, ex.Message);
    }
    else
    {
        _userService.AddLogOperationUser(_userLogged.userId, (int)Enumerations.OperationsType.ConsultarTodos, _currentPa
    }

    filters.pageNumber = filters.pageNumber == 0 ? _paginationOptions.DefaultPageNumber : filters.pageNumber;
    filters.pageSize = filters.pageSize == 0 ? _paginationOptions.DefaultPageSize : filters.pageSize;

    var userSecurityQuestion = _unitAdmin.userSecurityQuestionRepository.GetAll().Where(x => x.Active == true);

    foreach (var user_SecurityQuestion in userSecurityQuestion)
        user_SecurityQuestion.Reply = _unitAdmin.utilitiesRepository.DecryptValue(user_SecurityQuestion.Reply, EncryptTy
    if (filters.idUser > 0)
        userSecurityQuestion = userSecurityQuestion.Where(x => x.IdUser == filters.idUser);

    if (filters.idSecurityQuestion > 0)
        userSecurityQuestion = userSecurityQuestion.Where(x => x.IdSecurityQuestion == filters.idSecurityQuestion); ;

    if (userSecurityQuestion == null)
    {
        NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("FG-0904",
        throw ThrowException(ex, LogEvents.GetItemsNotFound, ex.Message);
    }

    return PagedList<UserSecurityQuestion>.Create(userSecurityQuestion, filters.pageNumber, filters.pageSize);
}
```

Fuente: el autor.

**Descripción:** Desde el método Get() en la clase ExceptionService se consultan todas las preguntas de seguridad por usuario del sistema, incluso mediante filtros personalizados y se retorna el listado de preguntas de seguridad por usuario resultantes.

**Figura 117.**

*Método Post() para asociar múltiples preguntas de seguridad a un usuario*

```
115     // Asociar múltiples preguntas de seguridad a un usuario
        2 references
116     public async Task<IEnumerable<UserSecurityQuestion>> AssociateSecurityQuestions(List<UserSecurityQuestion> userSecurityQ
117     { //Este código verifica si el usuario tiene permiso para poder acceder a este servicio
118         if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Crear).FirstOrDefault() == null)
119         {
120             UnauthorizedBusinessException ex = new UnauthorizedBusinessException(_unitAdmin.exceptionRepository.GetByCode("F
121             throw ThrowException(ex, LogEvents.InsufficientPermissions, ex.Message);
122         }
123         else
124         {
125             _userService.AddLogOperationUser(_userLogged.userId, (int)Enumerations.OperationsType.Crear, _currentPage.ToStri
126         }
127
128         // Listas con los datos finales a añadir y modificar
129         List<UserSecurityQuestion> addUserSecurityQuestions = new List<UserSecurityQuestion>();
130         List<UserSecurityQuestion> updateUserSecurityQuestions = new List<UserSecurityQuestion>();
131
132         // Se consultan las preguntas de seguridad originales asociadas al usuario
133         var userSecurityQuestion = _unitAdmin.userSecurityQuestionRepository.GetAllUserSecurityQuestions(_userLogged.userId)
134
135         // Obtener todas las preguntas de seguridad
136         IEnumerable<SecurityQuestion> securityQuestions = _unitAdmin.securityQuestionRepository.GetSecurityQuestions();
137
138         // Obtener los datos del dominio asociado
139         Domain.Entities.Domain domain = await _unitAdmin.domainRepository.GetById(_userLogged.domainId);
140
141         // Se separan los objetos a añadir y modificar
142         var toAdd = userSecurityQuestions.Where(x => x.Id == 0).ToList();
143         var toEdit = userSecurityQuestions.Where(x => x.Id > 0).ToList();
144
145     if (userSecurityQuestions.Select(x => x.IdSecurityQuestion).GroupBy(n => n).Any(c => c.Count() > 1) ||
146         userSecurityQuestions.Count() > domain.RequestSecurityQuestions)
147     {
148         BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCode("MG-0001",
149         throw ThrowException(ex, LogEvents.InsertItem, ex.Message);
150     }
151
152     foreach (var user_SecurityQuestion in userSecurityQuestion)
153     {
154         var selectRecord = toEdit.Where(x => x.Id == user_SecurityQuestion.Id).FirstOrDefault();
155
156         if (selectRecord != null)
157         {
158       if (user_SecurityQuestion.IdUser != selectRecord.IdUser)
159       {
160           UnauthorizedBusinessException ex = new UnauthorizedBusinessException(_unitAdmin.exceptionRepository.GetByCode("FG-
161           throw ThrowException(ex, LogEvents.InsufficientPermissions, ex.Message);
162       }
163       else if (VerifySecurityQuestion(selectRecord, securityQuestions))
164       {
165           user_SecurityQuestion.IdSecurityQuestionNavigation = null;
166           user_SecurityQuestion.IdSecurityQuestion = selectRecord.IdSecurityQuestion;
167           user_SecurityQuestion.IdUser = _userLogged.userId;
168           user_SecurityQuestion.IdUpdateUser = _userLogged.userId;
169           user_SecurityQuestion.UpdateDate = System.DateTime.Now;
170           user_SecurityQuestion.UpdateMachine = _userLogged.publicIP;
```

Fuente: el autor.

**Descripción:** El método Post() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta creación y modificación de preguntas de seguridad asociadas a un usuario. Desarrollo del caso de uso de Asociar preguntas de seguridad.

**Figura 118.**

*Método Post() para asociar múltiples factores de autenticación a un dominio*

```
35  public async Task<AuthenticationFactorsConfigurationDto> PostAuthenticationFactorConfiguration(AuthenticationFactorsConf
36  {   //Este código verifica si el usuario tiene permiso para poder acceder a este servicio
37      if (_userLogged.operations.Where(x => x.Id == (int)Enumerations.OperationsType.Crear).FirstOrDefault() == null)
38      {
39          UnauthorizedBusinessException ex = new UnauthorizedBusinessException(_unitAdmin.exceptionRepository.GetByCode("F
40          throw ThrowException(ex, LogEvents.InsufficientPermissions, ex.Message);
41      }
42      else
43      {
44          _userService.AddLogOperationUser(_userLogged.userId, (int)Enumerations.OperationsType.Crear, _currentPage.ToStri
45      }
46
47      // Se asigna el dominio a la configuración de desafíos
48      entity.configuration.idDomain = entity.idDomain;
49      ChallengesConfiguration challenges_Configuration = null;
50      ChallengesConfiguration configuration = null;
51
52      // Se valida la existencia del dominio y se obtienen los desafíos asociados
53      Domain.Entities.Domain domain = _unitAdmin.domainRepository.GetDomainWithChallenges(entity.idDomain);
54      if (domain == null)
55      {
56          NotFoundBusinessException ex = new NotFoundBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-3208",
57          throw ThrowException(ex, LogEvents.GetItemNotFound, ex.Message);
58      }
59
60      // Se consulta la configuración previa de desafíos por dominio
61      configuration = _unitAdmin.challengesConfigurationRepository.GetChallengeConfigurationByDomain(entity.idDomain);
62
63      // Se muestra excepción si ya existe otro registro con el mismo dominio a actualizar
64      if (configuration != null && configuration.Id != entity.id)
65  {
66      BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-2814",
67      throw ThrowException(ex, LogEvents.ChallengeConfiguration, ex.Message);
68  }
69
70  // Lista que tendrá los nuevos desafíos a agregar
71  var toAdd = entity.authenticationFactors.Where(x => x.id == 0).ToList();
72
73  // Lista que tendrá los desafíos originales a modificar
74  var toEdit = entity.authenticationFactors.Where(x => x.id > 0).ToList();
75
76  // Se selecciona la cantidad de desafíos aleatorios para confirmar la identidad
77  var numberRequiredChallenges = entity.configuration.numberRequiredChallenges;
78
79  // Se selecciona la cantidad de desafíos requeridos
80  var challengesRequired = entity.authenticationFactors.Where(x => x.isRequired == true).Count();
81
82  // Validando reglas de negocio
83  if (numberRequiredChallenges > 0 && challengesRequired > 0 || numberRequiredChallenges == 0 && challengesRequired == 0 |
84      numberRequiredChallenges > 3 || numberRequiredChallenges > entity.authenticationFactors.Count || numberRequiredChall
85  {
86      BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-2816",
87      throw ThrowException(ex, LogEvents.ChallengeConfiguration, ex.Message);
88  }
89  else if (!(entity.configuration.timeAnswerChallenges >= 1 && entity.configuration.timeAnswerChallenges <= 10))
90  {
91      BadRequestBusinessException ex = new BadRequestBusinessException(_unitAdmin.exceptionRepository.GetByCode("AG-2817",
92      throw ThrowException(ex, LogEvents.ChallengeConfiguration, ex.Message);
93  }
94  else if (!(entity.configuration.numberDailyAttempts >= 3 && entity.configuration.numberDailyAttempts <= 10))
```

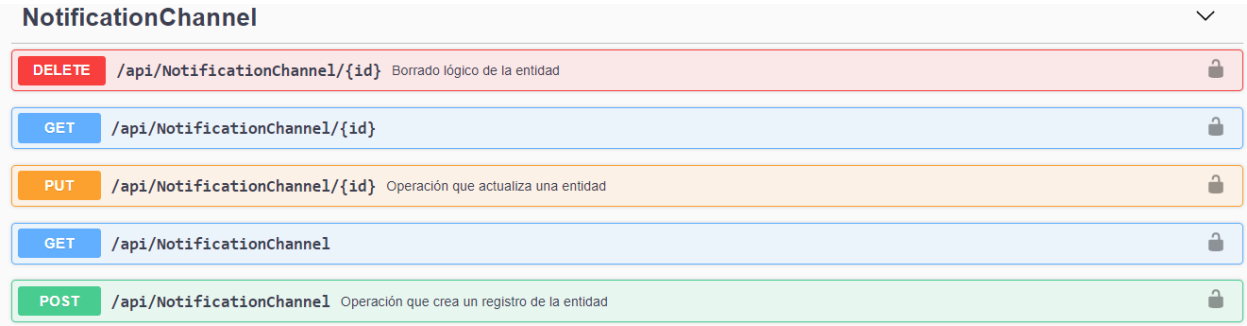
Fuente: el autor.

**Descripción:** El método Post() de servicio contiene la lógica y reglas de negocio que se requieren aplicar para validar la correcta creación y modificación de factores de autenticación asociados a un dominio. Desarrollo del caso de uso de Configuración de Desafíos.

- Pruebas de los servicios implementados en desarrollo back-end

**Figura 119.**

*Servicios implementados para el desarrollo de Configuración Canales de Envío*




Fuente: el autor.

**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestran los servicios implementados para crear (Post), modificar (Put), eliminar (Delete) y obtener (Get) objetos de canales de envío, según las reglas de negocio, relacionados al desarrollo del caso de uso de Configuración Canales de Envío.

Mediante los canales de envío se integrarán las cuentas de servicios de terceros a BSR SSO, para procesos de doble factor y los desafíos por cliente.

**Figura 120.**

*Petición para configurar un canal de notificación a crear desde Swagger UI*

Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se envía la petición y se guarda la configuración de canal de notificación en base de datos.

Como parámetros, el nombre del cliente relacionado al usuario que realiza la petición.

**Figura 121.**

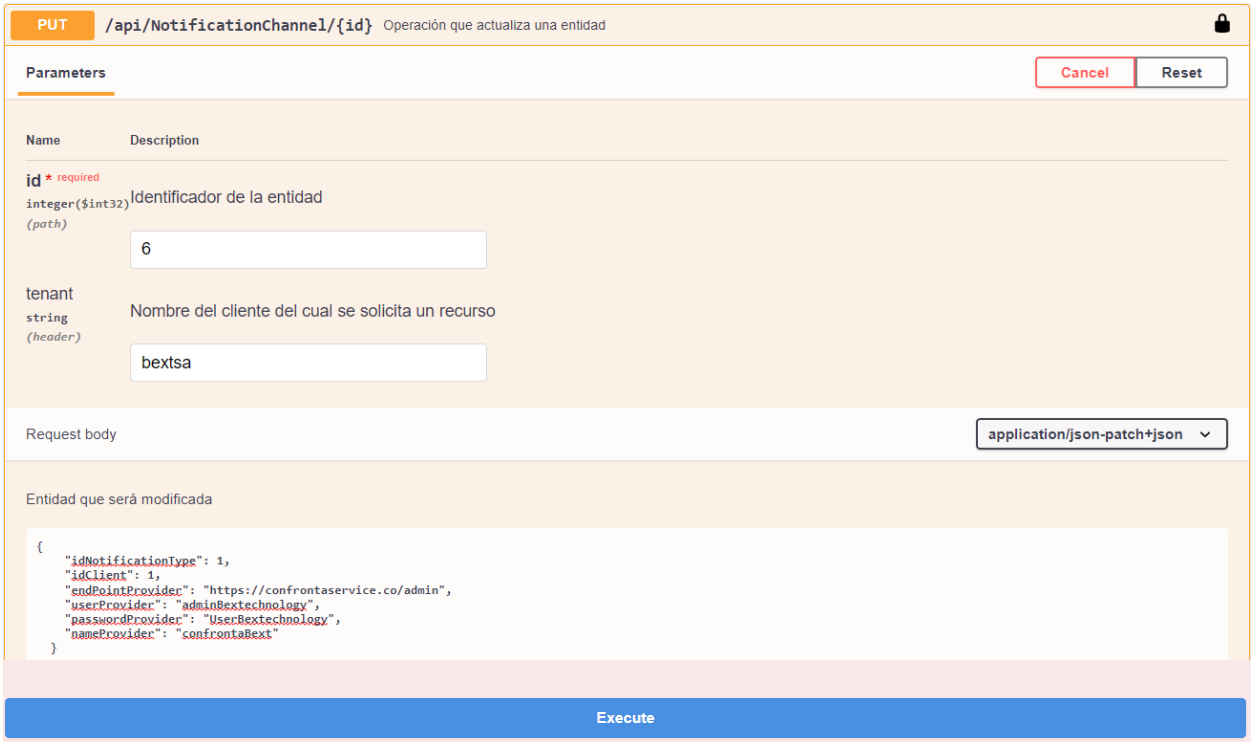
*Respuesta que muestra Swagger UI al crearse un canal de notificación*

Fuente: el autor.

**Descripción:** Desde el back-end se retorna la configuración de canal de notificación creada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 122.**

*Configurando un canal de notificación a modificar desde Swagger UI*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se envía la petición y se actualiza la configuración de canal de notificación en base de datos.

Como parámetros, el identificador del registro a modificar y el nombre del cliente relacionado al usuario que realiza la petición.

**Figura 123.**

*Respuesta que muestra Swagger UI al modificarse un canal de notificación*

Server response

| Code | Details |
|---|---|
| 200 | Response body |


```
{
  "data": {
    "idNotificationType": 1,
    "idClient": 1,
    "endPointProvider": "https://confrontaservice.co/admin",
    "userProvider": "adminBextechnology",
    "passwordProvider": "UserBextechnology",
    "nameProvider": "confrontaBext",
    "creatingDate": "2022-03-22T08:09:16.267",
    "updatingDate": "2022-03-22T08:16:15.920553-05:00",
    "id": 6
  }
}
```


Download

Response headers

```
content-encoding: gzip
content-length: 367
content-type: application/json; charset=utf-8
date: Wed,13 Apr 2022 13:59:55 GMT
server: Microsoft-IIS/10.0
vary: Accept-Encoding
x-powered-by: ASP.NET
```

Fuente: el autor.

**Descripción:** Desde el back-end se retorna la configuración de canal de notificación modificada correctamente en base de datos, mostrándose en Swagger UI.

**Figura 124.**

*Servicios implementados para el desarrollo de Asociar preguntas de seguridad*

Fuente: el autor.

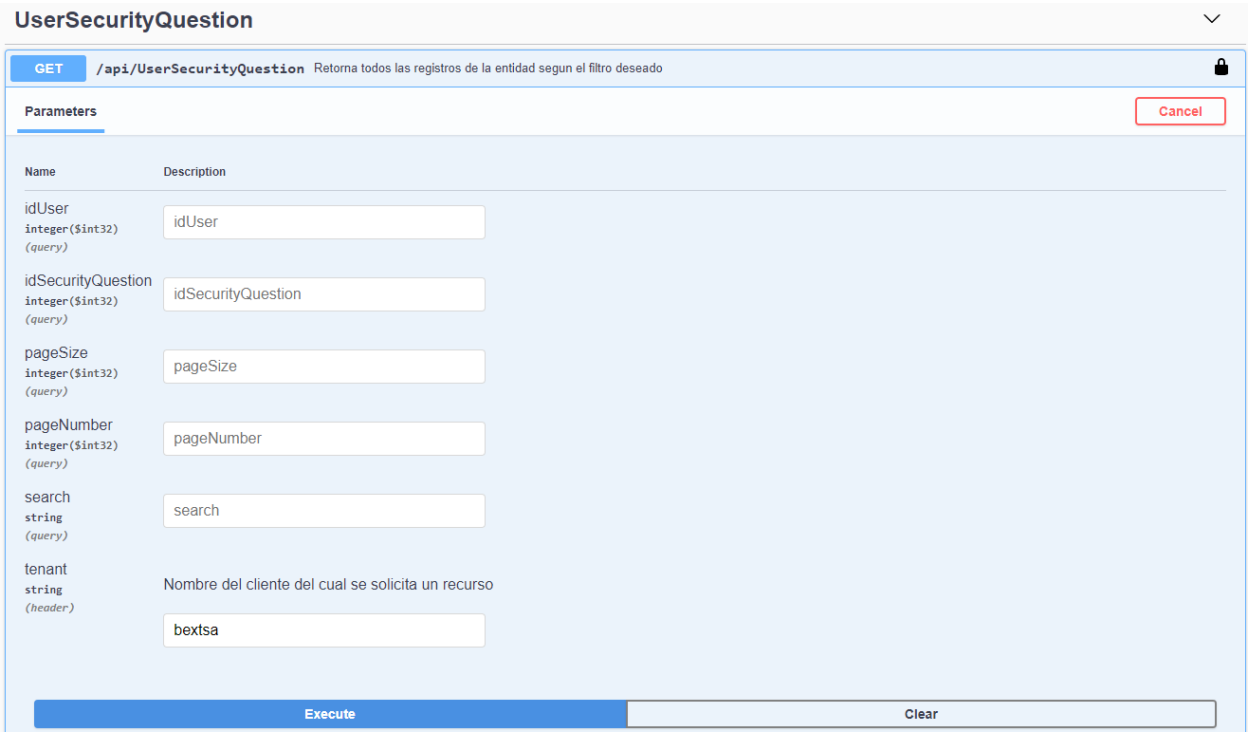
**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestran los servicios implementados para crear (Post), modificar (Put), eliminar (Delete) y obtener (Get) objetos de preguntas de seguridad del usuario, según las reglas de negocio, relacionados al desarrollo del caso de uso de Asociar preguntas de seguridad.

El usuario básico del sistema seleccionara preguntas deseadas a las cuales les asociara una respuesta.

El sistema le solicitará al usuario ingresar las respuestas correctas a las preguntas que asocio previamente, en procesos que requieran verificar la identidad del usuario.

**Figura 125.**

*Petición para obtener las preguntas de seguridad asociadas con su respuesta*




Fuente: el autor.

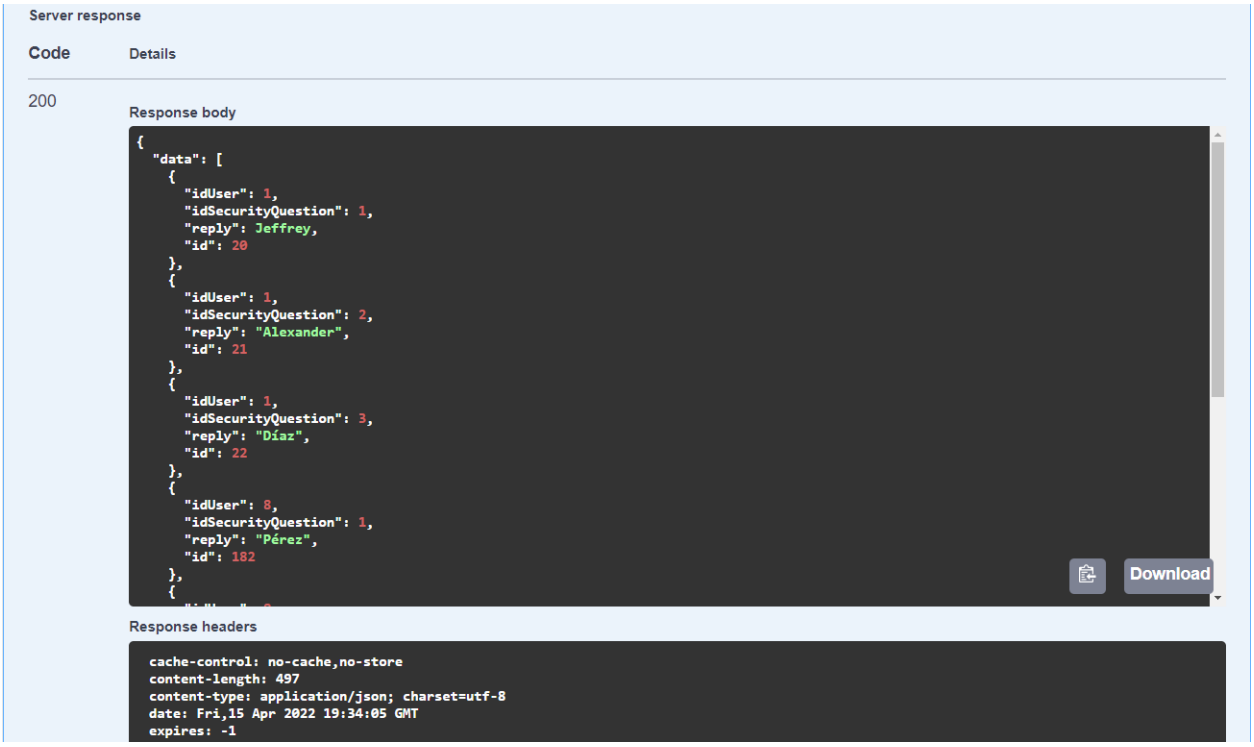
**Descripción:** Servicio para obtener todas las preguntas de seguridad con su respuesta almacenadas en base de datos.

Los resultados se pueden filtrar por usuario y pregunta de seguridad específica.

Como parámetros, el identificador del usuario, identificador de pregunta de seguridad y el nombre del cliente relacionado al usuario que realiza la petición.

**Figura 126.**

*Respuesta que muestra Swagger UI al obtener preguntas de seguridad asociadas*

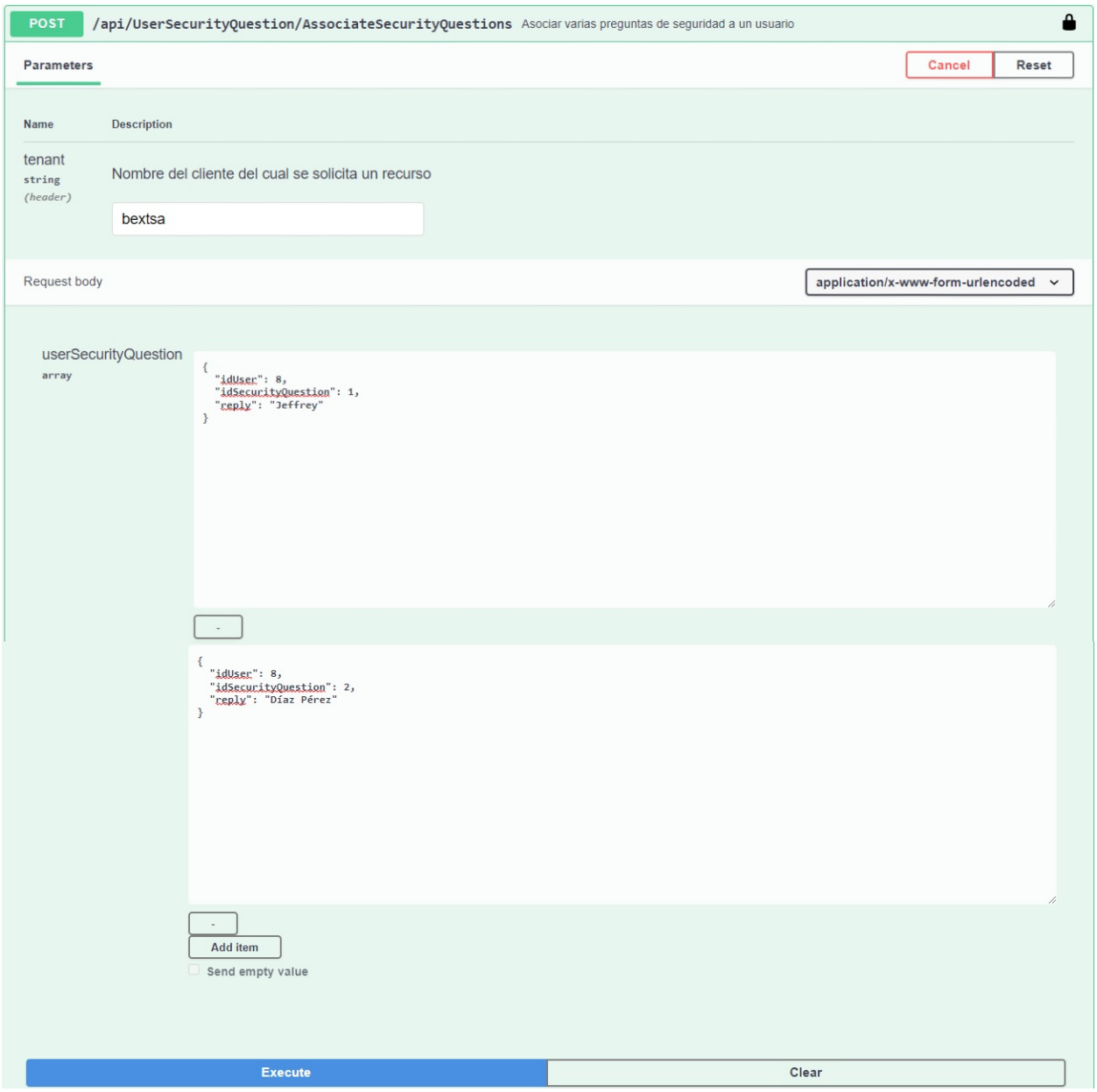



Fuente: el autor.

**Descripción:** Desde el back-end se retorna el listado actual de preguntas de seguridad asociadas del usuario con su respuesta, mostrándose en Swagger UI.

**Figura 127.**

*Asociar múltiples preguntas de seguridad con su respuesta a un usuario*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los objetos del arreglo de objetos en formato JSON, para asociar múltiples preguntas de seguridad a un usuario en una sola petición y se guarda en base de datos.

**Figura 128.**

*Respuesta que muestra Swagger UI al asociar múltiples preguntas de seguridad*

Server response

| Code | Details |
|---|---|
| 200 | Response body |


```
{
  "data": [
    {
      "idUser": 8,
      "idSecurityQuestion": 1,
      "reply": "W7YTW9nRcYw=",
      "id": 145
    },
    {
      "idUser": 8,
      "idSecurityQuestion": 2,
      "reply": "rdjaGBh4pavHXIKMsQVrAA==",
      "id": 146
    }
  ]
}
```


Download

Response headers

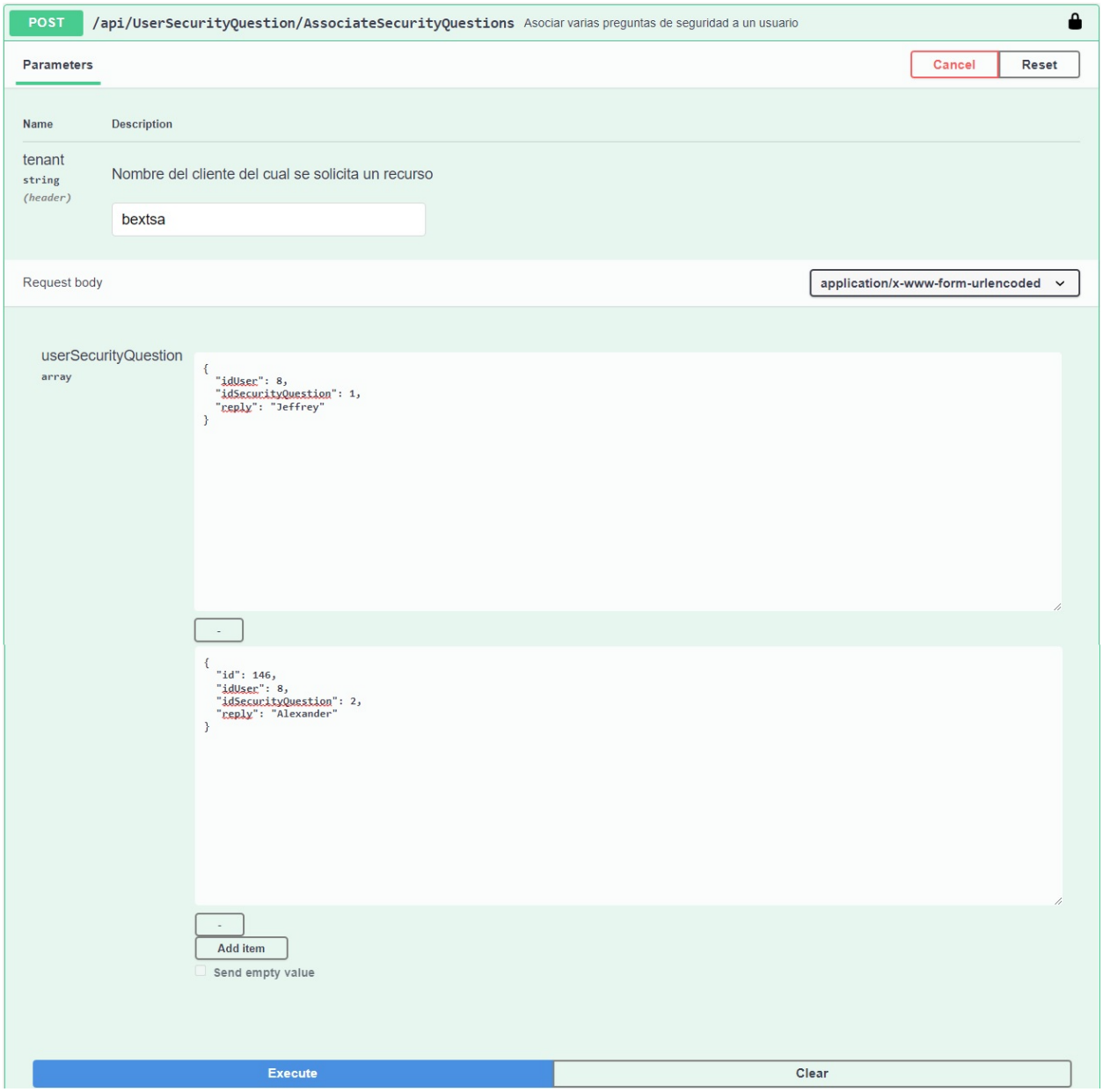
```
content-encoding: gzip
content-length: 367
content-type: application/json; charset=utf-8
date: Wed,13 Apr 2022 13:59:55 GMT
server: Microsoft-IIS/10.0
vary: Accept-Encoding
x-powered-by: ASP.NET
```

Fuente: el autor.

**Descripción:** Desde el back-end se retorna el listado actual de preguntas de seguridad asociadas del usuario que se guardaron en base de datos, mostrándose en Swagger UI.

**Figura 129.**

*Crear y modificar preguntas de seguridad con su respuesta a un usuario*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los objetos del arreglo de objetos en formato JSON, para asociar y modificar múltiples preguntas de seguridad a un usuario en una sola petición y se guarda en base de datos.

**Figura 130.**

*Respuesta que muestra Swagger UI al crear y modificar preguntas de seguridad*

Server response

| Code | Details |
|---|---|
| 200 | Response body |


```
{
  "data": [
    {
      "idUser": 8,
      "idSecurityQuestion": 1,
      "reply": "W7YTW9nRcYw=",
      "id": 148
    },
    {
      "idUser": 8,
      "idSecurityQuestion": 2,
      "reply": "siCwTltqMLAT0ni4vV9Fzg==",
      "id": 146
    }
  ]
}
```


Download

Response headers

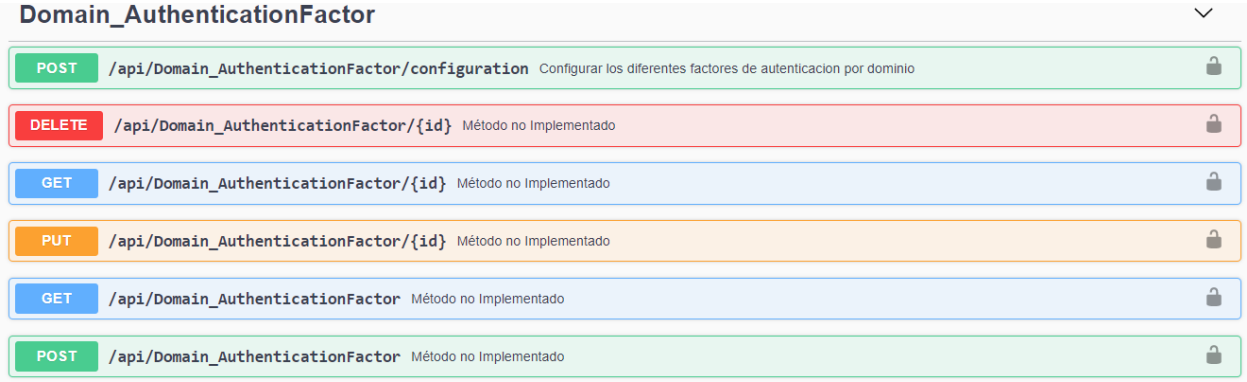
```
content-encoding: gzip
content-length: 367
content-type: application/json; charset=utf-8
date: Wed,13 Apr 2022 13:59:55 GMT
server: Microsoft-IIS/10.0
vary: Accept-Encoding
x-powered-by: ASP.NET
```

Fuente: el autor.

**Descripción:** Desde el back-end se retorna el listado actual de preguntas de seguridad asociadas del usuario que se crearon y actualizaron en base de datos, mostrándose en Swagger UI.

**Figura 131.**

*Servicios implementados para el desarrollo de Configuración de Desafíos*




Fuente: el autor.

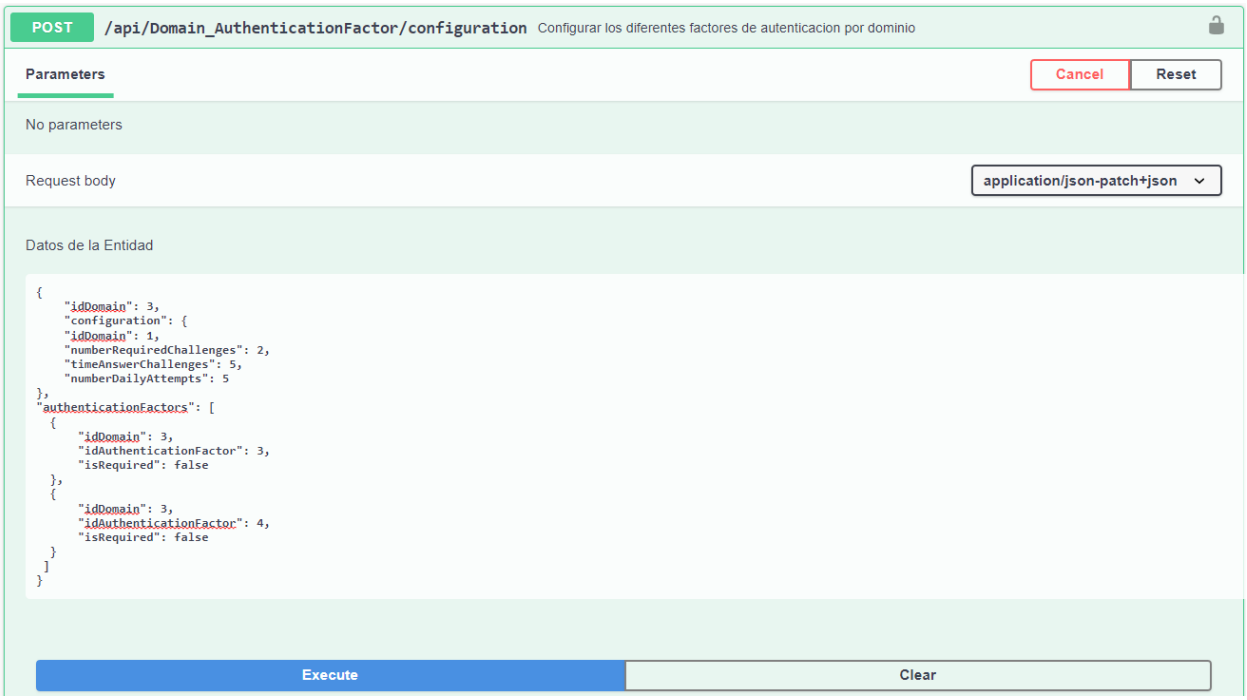
**Descripción:** Al ejecutar el proyecto desarrollado desde el entorno de desarrollo integrado Visual Studio, se muestra el servicio implementado para crear (Post) una configuración de desafíos, según las reglas de negocio, relacionados al desarrollo del caso de uso de Configuración de Desafíos.

En la configuración se establecerá el tiempo en que el usuario final deberá resolver todos los desafíos/factores de autenticación, la cantidad de desafíos requeridos, la cantidad de intentos diarios de verificación de identidad y el dominio al cual aplicará la configuración.

Se indicarán también los diferentes desafíos/factores de autenticación para el dominio y si son requeridos; los cuales el usuario deberá resolver.

**Figura 132.**

*Configurar múltiples factores de autenticación por dominio desde Swagger UI*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se envía la petición y se guarda la configuración de factores de autenticación por dominio en base de datos.

Se indica la configuración general por dominio que aplicara a todos los desafíos indicados y cada uno de los desafíos que el sistema solicitara al usuario realizar al momento de requerir la validación de identidad.

**Figura 133.**

*Respuesta que muestra Swagger UI al configurar múltiples desafíos por dominio*

Server response

| Code | Details |
|---|---|
| 200 | Response body |


```
{
  "data": {
    "idDomain": 3,
    "configuration": {
      "idDomain": 3,
      "numberRequiredChallenges": 2,
      "timeAnswerChallenges": 5,
      "numberDailyAttempts": 5,
      "id": 65
    },
    "authenticationFactors": [
      {
        "idDomain": 3,
        "idAuthenticationFactor": 3,
        "isRequired": false,
        "creatingDate": "2022-03-20T20:20:17.9699632-05:00",
        "id": 275
      },
      {
        "idDomain": 3,
        "idAuthenticationFactor": 4,
        "isRequired": false,
        "creatingDate": "2022-03-20T20:20:27.6344508-05:00",
        "id": 276
      }
    ],
    "id": 65
  }
```


Download

Response headers

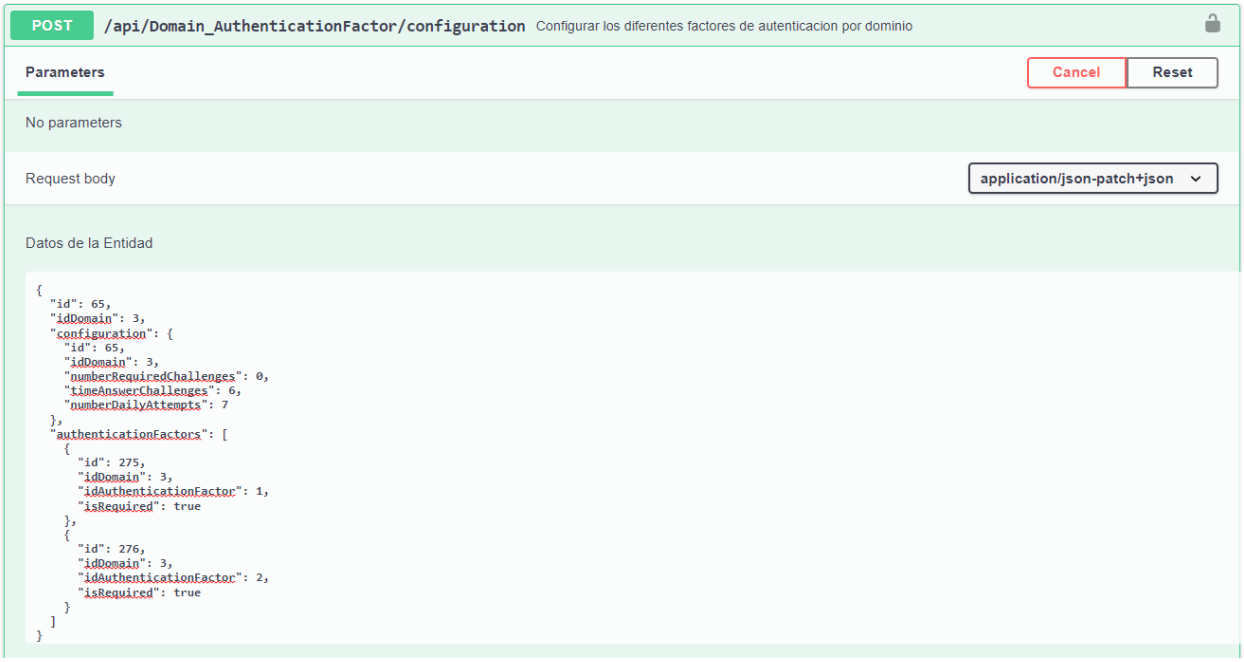
```
content-encoding: gzip
content-length: 367
content-type: application/json; charset=utf-8
date: Wed,13 Apr 2022 13:59:55 GMT
server: Microsoft-IIS/10.0
vary: Accept-Encoding
x-powered-by: ASP.NET
```

Fuente: el autor.

**Descripción:** Desde el back-end se retorna el listado actual de desafíos por dominio y la configuración asociada que se guardó en base de datos, mostrándose en Swagger UI.

**Figura 134.**

*Asociar y modificar múltiples factores de autenticación por dominio*




Fuente: el autor.

**Descripción:** En el cuerpo de la petición se ingresa el valor para cada uno de los atributos del objeto en formato JSON, se envía la petición, se guarda y actualiza la configuración de factores de autenticación por dominio en base de datos.

Se indica la configuración general por dominio que aplicara a todos los desafíos indicados y cada uno de los desafíos que el sistema solicitara al usuario realizar al momento de requerir la validación de identidad.

En este servicio se implementó la funcionalidad de poder crear y actualizar la configuración general de desafíos y cada uno de los desafíos asociados.

**Figura 135.**

*Respuesta que muestra Swagger UI al asociar y modificar múltiples desafíos*

Server response

Code | Details

200

Response body


```
{
  "data": {
    "idDomain": 3,
    "configuration": {
      "idDomain": 3,
      "numberRequiredChallenges": 0,
      "timeAnswerChallenges": 6,
      "numberDailyAttempts": 7,
      "id": 65
    },
    "authenticationFactors": [
      {
        "idDomain": 3,
        "idAuthenticationFactor": 1,
        "isRequired": true,
        "creatingDate": "2022-03-20T21:32:19.0650558-05:00",
        "id": 275
      },
      {
        "idDomain": 3,
        "idAuthenticationFactor": 2,
        "isRequired": true,
        "creatingDate": "2022-03-20T21:32:19.8214087-05:00",
        "id": 276
      }
    ],
    "id": 65
  }
```


Download

Response headers

```
content-encoding: gzip
content-length: 367
content-type: application/json; charset=utf-8
date: Wed,13 Apr 2022 13:59:55 GMT
server: Microsoft-IIS/10.0
vary: Accept-Encoding
x-powered-by: ASP.NET
```

Fuente: el autor.

**Descripción:** Desde el back-end se retorna el listado actual de desafíos por dominio y su configuración asociada que se creó y actualizo en base de datos, mostrándose en Swagger UI.

**Modernización del software multiplataforma “Bext Self Reset”**

“Bext Self Reset”, es una nueva aplicación web y multiplataforma actualmente en desarrollo que permitirá a usuarios finales la autogestión de contraseñas, permitiendo al usuario cambiar la contraseña de su cuenta vinculada al directorio activo, incluso cuando la haya olvidado, con lo cual se tendrá una única contraseña para todas las aplicaciones que hagan uso de la cuenta de usuario.

El desarrollo y funcionalidad del sistema está orientado a tres (3) aspectos principales: administración, usuario final y móviles.

Los diferentes módulos/páginas web de administración, permitirán a usuarios con roles de administrador ajustar la configuración del aplicativo según necesidades de la organización/empresa; por ejemplo: la personalización del aplicativo, las políticas de seguridad aplicadas al cambio de contraseña, la configuración de desafíos para validación de identidad, la conexión a diferentes repositorios de usuarios, dar soporte a los usuarios que lo requieran.

Los diferentes módulos/páginas web de usuario final, permitirán a usuarios básicos del sistema la autogestión de información personal y laboral, recordar nombre de usuario, habilitar cuenta y el cambio de contraseña, el cual se aplicará a todas las aplicaciones que utilice.

Los diferentes módulos orientados a uso móvil para usuarios finales (básicos), en los cuales se reutiliza la funcionalidad implementada en los módulos/páginas web de usuario final.

Esta nueva versión del producto “Bext Self Reset” está orientada a implementarse mediante Software como servicio (SaaS). No es obligación para el cliente administrar la infraestructura subyacente para el funcionamiento del aplicativo, esas tareas se pueden delegar a un proveedor de servicios externo en la nube, quien también podrá garantizar la disponibilidad, seguridad e integridad de la aplicación y los datos. Los diferentes clientes alquilan el uso del

aplicativo y los usuarios, ya sean básicos o administradores, hacen uso del sistema mediante un explorador web o aplicación móvil.

La nueva versión del producto “Bext Self Reset” se desarrolla haciendo uso de buenas prácticas como lo es la arquitectura limpia, mediante la cual la estructura interna del aplicativo se organiza en capas/componentes con el fin de separar responsabilidades y para que los datos del sistema fluyan desde la capa más exterior hacia la capa más interior según como es recomendado. En desarrollo, la solución general del producto está compuesta por cuatro (4) proyectos: la capa más exterior que se implementó en back-end fue el proyecto API el cual contiene los diferentes controladores y la presentación de la web API al ejecutar el proyecto, en la capa interior que le sigue se encuentra el proyecto de infraestructura para validaciones de datos y peticiones a la base de datos y el proyecto de persistencia el cual es un mapeo de objetos relacionales a la base de datos, en la capa más interior de la arquitectura limpia se encuentra el proyecto Core, el cual contiene el dominio y lógica de negocio (entidades y casos de uso).

A nivel de desarrollo se aplica el patrón de diseño de software de inyección de dependencias. Una clase puede tener múltiples dependencias, las cuales a su vez pueden tener otras dependencias, resolverlas todas manualmente será tedioso a medida que “crece” el desarrollo del proyecto. Para ello, ASP.NET Core provee una funcionalidad integrada que se encarga de resolver automáticamente en tiempo de ejecución todas las dependencias que tenga el proyecto. Así, si se declara la dependencia y abstracción “IAttributeService” como parámetro en el constructor de la clase “AttributeController” podemos configurar en un Contenedor de Inversión de Controles para que en dado caso se nos entregue una instancia de la clase AttributeService con todas las dependencias ya resueltas que esta clase también pudiera tener. Esto permite tener una aplicación más modular, escalable y mantenible en el tiempo.

También se hace uso del patrón de diseño de repositorio con la finalidad de desacoplar la lógica de negocios del acceso a los datos (persistencia). No es una buena práctica que las clases que contienen lógica de negocio se comuniquen y manipulen directamente la fuente de datos a la que se conecta el sistema, es sugerible separar responsabilidades añadiendo una capa intermedia/mediadora que recibe las peticiones originadas desde la capa de lógica de negocios, orientando la petición a clases repositorio que se comunican con la fuente de datos, recibiendo los resultados devueltos por la fuente de datos y retornando estos resultados a la capa de lógica de negocios que inicio la petición.

De igual forma se aplica el patrón de unidad de trabajo, mediante el cual se puede hacer uso de los diferentes repositorios en cualquier clase que aplique lógica de negocios. Por lo que, si desde la clase de servicios de usuarios se requiere hacer uso de la clase repositorio de dominios, ya sea para consultar los datos del dominio asociado al usuario; solamente se hace uso de la instancia de unidad administrativa invocando a la propiedad deseada de repositorio, seguido del método de consulta deseado a la base de datos, pasándole como parámetro el identificador de llave primaria del usuario. De esta forma se hace uso de manera centralizada de la unidad administrativa de trabajo desde cualquier clase de servicio que implemente lógica de negocios, invocando al repositorio necesario que se comunicara con la base de datos para una instrucción u operación específica, ya sea añadir, actualizar, eliminar o consultar/obtener datos.

## Reflexiones finales

Generar algunas de las funcionalidades y características necesarias para el software multiplataforma “Bext Self Reset” en el transcurrir de la pasantía, me ayudo a mejorar mis habilidades y conocimientos de desarrollo de software, gestión de base de datos y documentación de casos de uso, fue un importante aporte a las bases teóricas aprendidas en la universidad.

Haber participado en algunas etapas de desarrollo de software, tales como: análisis de requerimientos y documentación de casos de uso, desarrollo en back-end, pruebas de las funcionalidades implementadas, configuración de despliegues para ambiente de preproducción y producción en la nube para la entrega continua de nuevas características del producto; fue una experiencia muy gratificante, se adquiere una mayor adaptación y conocimientos para poder trabajar en diferentes actividades de desarrollo.

Aunque al inicio de la pasantía tuve algunas dificultades para cumplir con el desarrollo de algunas actividades acordadas en el plan de trabajo, logré realizarlas satisfactoriamente, dando cumplimiento a todas las actividades del plan de trabajo de pasantía en el tiempo acordado.

## Conclusiones

Se realizó el desarrollo de la pasantía durante veintiséis (26) semanas trascurridas desde el 6 de octubre del año 2021 hasta el 5 de abril del año 2022, en cumplimiento del plan de trabajo acordado, aplicando los conocimientos adquiridos durante el proceso formativo del programa de Ingeniería de Sistemas de la Universidad Nacional Abierta y a Distancia (UNAD), siendo muy satisfactorio este proceso en cuanto a resultados generados y conocimientos adquiridos.

El objetivo principal al realizar el plan de trabajo de pasantía era aportar al desarrollo de nuevas funcionalidades y características a integrar en la nueva versión del producto “Bext Self Reset”. Este objetivo se cumplió de manera satisfactoria y completa al haber realizado cada una de las actividades acordadas y que me fueron asignadas en los tiempos establecidos.

Se logró elaborar satisfactoriamente la documentación de casos de uso requeridos, el diseño de base de datos para nuevas tablas y relaciones necesarias, la implementación de casos de uso en desarrollo back-end, la implementación de una canalización en Azure DevOps para la entrega continua de nuevas características del producto, la puesta en marcha de un proveedor de identidad y de servicios para la autenticación y autorización de usuarios en el sistema, la validación de funcionalidad de los servicios RESTful implementados; dando cumplimiento según lo acordado en el plan de trabajo de pasantía.

**Bibliografía**

Bagui, S., & Earp, R. (2014). *SQL Server 2014: A Step by Step Guide to Learning SQL*. Nova Science Pub Inc.

BEXTechnology S.A. (2022). *Sobre nosotros*. https://jp.linkedin.com/company/bextechnology

BEXTechnology S.A. (2022). *Cultura Gente*. https://www.bextsa.com/cultura-gente

Chakray. (21 de diciembre de 2016). *¿Qué diferencias hay entre REST y SOAP?.* Chakray. https://www.chakray.com/es/que-diferencias-hay-entre-rest-y-soap/

Grijalva , O. C., & Mancheno, S. D. (Mayo de 2012). *Investigacion cualitativa.* https://www.monografias.com/trabajos95/investigacioncualitativa/investigacioncualitativa

Kriegel, A. (2011). *Discovering SQL: A Hands-On Guide for Beginners*. Wrox.

Meyer, M. (2014). *Continuous Integration and Its Tools*. IEEE Software, 14-16.

Microsoft. (29 de diciembre de 2021). *Implement Continuous Integration and Continuous deployment using Azure DevOps.* Microsoft Docs. https://docs.microsoft.com/en-us/sharepoint/dev/spfx/toolchain/implement-ci-cd-with-azure-devops

Microsoft. (s.f.). *Entity Framework documentation.* Microsoft Docs. https://docs.microsoft.com/en-us/ef/

Microsoft. (s.f.). *Microsoft SQL documentation - SQL Server.* Microsoft Docs. https://docs.microsoft.com/en-us/sql/?view=sql-server-ver15

Pérez, G. (03 de 09 de 2021). *.NET 6 – Minimal API: La nueva manera de hacer APIs.* Visionarios. https://blogvisionarios.com/e-learning/articulos-data/net-6-minimal-api-la-nueva-manera-de-hacer-apis/

Rabazo, T. (09 de noviembre de 2018). *WSO2 Identity Server Tutorial: Introducción*. Chackray. https://www.chakray.com/es/wso2-identity-server-tutorial-introduccion/

Red Hat. (31 de enero de 2018). *¿Qué son la integración y la distribución continuas (CI/CD)?* Red Hat. https://www.redhat.com/es/topics/devops/what-is-ci-cd

Red Hat. (08 de mayo de 2020). *¿Qué es una API de REST?* Red Hat. https://www.redhat.com/es/topics/api/what-is-a-rest-api

Timms, S. (14 de 09 de 2018). *Entity Framework Core Tutorial*. Stackify. https://stackify.com/entity-framework-core-tutorial/

WSO2. (2022). *WSO2 Identity Server Documentation.* WSO2. https://is.docs.wso2.com/en/latest/