RESUMEN – TRABAJO DE GRADO

AUTORES	**Andres Felipe Montaño Sanchez**
FACULTAD	**Ingeniería**
PLAN DE ESTUDIOS	**Ingeniería de Sistemas**
DIRECTOR	**Juan Camilo Jaimes Fernández**
TÍTULO DE LA TESIS	**Desarrollo de nuevas funcionalidades en el software de gestión del conocimiento (Valko) de la empresa Bancolombia, haciendo uso de tecnologías Spring Boot.**
TITULO EN INGLES	**Development of new functionalities in the knowledge management software (Valko) of the company Bancolombia, using Spring Boot technologies.**

RESUMEN
(70 palabras)

El proyecto Realizado en Bancolombia desarrolló nuevas funcionalidades en su plataforma de gestión del conocimiento, eliminando la dependencia de Microsoft Forms y mejorando la experiencia del usuario con avanzados filtros. Se diseñó una nueva interfaz en Figma y se ajustó la base de datos en MySQL. Usando Java 11 con Spring Boot, se optimizó la eficiencia operativa y se capacitó al equipo, mejorando la gestión de encuestas y la adaptabilidad de la empresa.

RESUMEN EN INGLES

The Realozando project at Bancolombia developed new functionalities in its knowledge management platform, eliminating dependence on Microsoft Forms and improving the user experience with advanced filters. A new interface was designed in Figma and the database was adjusted in MySQL. Using Java 11 with Spring Boot, operational efficiency was optimized and the team was trained, improving survey management and company adaptability.

PALABRAS CLAVES	Funcionalidades, Encuestas, Filtros, Eficiencia
PALABRAS CLAVES EN INGLES	Features, Surveys, Filters, Efficiency

CARACTERÍSTICAS

PÁGINAS: 72	PLANOS: 0	ILUSTRACIONES: 40	CD-ROM: 0

Desarrollo de nuevas funcionalidades en el software de gestión del conocimiento (Valko) de la empresa Bancolombia, haciendo uso de tecnologías Spring Boot.

Andres Felipe Montaño Sanchez

facultad de ingenierías, universidad francisco de paula Santander Ocaña

ingeniería de sistemas

Mgtr. Juan Camilo Jaimes Fernandez

23 de agosto de 2024

Índice

Resumen 8

Introducción 10

1 Desarrollo de nuevas funcionalidades en el software de gestión del conocimiento (Valko) de la empresa Bancolombia, haciendo uso de tecnologías Spring Boot. 11

1.1 Descripción de la empresa 11

1.1.1 Misión 11

1.1.2 Visión 12

1.1.3 Objetivos de la empresa 12

1.1.4 Descripción de la estructura organizacional. 13

1.1.5 Descripción de la dependencia y/o proyecto al que fue asignado. 14

1.2 Diagnóstico inicial de la dependencia asignada. 14

1.2.1 Planteamiento del problema 16

1.3 Objetivos de la pasantía 17

1.3.1 General 17

1.3.2 Específicos 18

1.4 Descripción de las actividades a desarrollar en la misma 18

2 Enfoques referenciales 20

2.1 Enfoques conceptuales 20

2.1.1 Metodologia Scrum 20

2.1.2 Bases de Datos Relacionales 22

2.1.3 Seguridad de Datos 23

2.1.4 Framework 24

2.1.5 Pruebas Unitarias 25

2.1.6 Modelado de Base de Datos 27

2.1.7 Pruebas de Aceptación 29

2.2 Enfoques legales 30

2.2.1 Proteccion de datos personales 30

2.2.2 Ley de derechos de autor y propiedad intelectual 31

2.2.3 Accesibilidad a la web 32

2.2.4 Seguridad informática 32

2.2.5 Norma iso 27001 32

3 Informe de cumplimiento 33

3.1 Presentación de resultados 33

3.1.1 Formación 33

3.1.2 Análisis del estado de la base de datos 35

3.1.3 Revisión de documentación del sistema de diseño Galatea 36

3.1.4 Análisis de los requisitos propuestos 39

3.1.5 Propuesta y diseño del modelo de datos 40

3.1.6 Diseño en Figma..... 42

3.1.7 Codificación de filtros para módulo de formación..... 46

3.1.8 Codificación funcionalidad Encuestas (CRUD). 56

3.1.9 Desarrollar y ejecutar pruebas tanto unitarias como de aceptacion. 62

3.1.10 Documentar la forma de uso de EndPoint para entrega al front. 66

4 Diagnostico Final..... 67

5 Conclusión..... 68

6 Recomendaciones..... 69

7 Referencias 70

Lista de figuras

Figura 1 Estructura Organizacional. 13

Figura 2 Valko 34

Figura 3 Estado base de datos 36

Figura 4 Sistema de diseño 37

Figura 5 Sistema de diseño Back 38

Figura 6 Base de datos con los nuevos cambios. 41

Figura 7 Diseño anterior 43

Figura 8 Capturas de los nuevos diseños de todo el apartado de formación. 43

Figura 9 Diseños 44

Figura 10 Diseños 44

Figura 11 Diseños 45

Figura 12 Toma General de los Diseños 45

Figura 13 Código Controller 46

Figura 14 Código servicio. 47

Figura 15 Código repository. 47

Figura 16 Código lista de cursos. 48

Figura 17 Controller para listar todos los cursos ofertados. 49

Figura 18 Controller para listar el histórico de cursos del empleado. 49

Figura 19 Controller para listado de cursos ofertados por el cargo del empleado. 50

Figura 20 Service para listado de cursos ofertados según el cargo del empleado. 50

Figura 21 Service para listar el historico de cursos del empleado. 51

Figura 22 Service para listar el total de las ofertas 51

Figura 23 Para listado de cursos activos del empleado. 52

Figura 24 Código consultas 52

Figura 25 Código DTO 53

Figura 26 Código de cursos ofertados 54

Figura 27 EndPoint encuenstas. 56

Figura 28 EndPoint relaciones 56

Figura 29 EndPoint leer encuesta 57

Figura 30 57

Figura 31 EndPoint actualizar encuestas 58

Figura 32 EndPoint eliminar encuestas 59

Figura 33 Servicio 60

Figura 34 Mapeo de errores 61

Figura 35 Consultas a la base de datos luego del servicio 61

Figura 36 Pruebas de Aceptacion 63

Figura 37 Pruebas de software mockito 64

Figura 38 Pruebas 64

Figura 39 Gherkin 65

Figura 40 Entrega de EndPoint 66

Lista de tablas

Tabla 1 Análisis FODA de la dependencia asignada. 15

Tabla 2 Actividades 18

Informe Final

Resumen

El proyecto consistió en desarrollar e implementar nuevas funcionalidades la plataforma de gestión del conocimiento de Bancolombia, con el objetivo de eliminar la dependencia de Microsoft Forms para la elaboración de encuestas y tambien mejorar la experiencia del usuario mediante la implementación de avanzados filtros en el apartado de formación. Inicialmente, se realizó un análisis detallado de los procesos existentes y los requisitos necesarios, seguido por el diseño de una nueva interfaz en Figma y ajustes al modelo de la base de datos en MySQL. Se utilizaron tecnologías modernas como Java 11 con Spring Boot y el entorno de desarrollo IntelliJ para asegurar una integración eficiente y mantener la integridad de la información. Las funcionalidades de filtrado permitieron a los usuarios acceder de manera más sencilla y eficiente a la información relevante, optimizando la interacción con la plataforma.

El desarrollo del backend del módulo de formación ha representado un avance significativo en la capacidad de Bancolombia para gestionar y optimizar las encuestas internas, mejorando considerablemente la eficiencia operativa. La eliminación de la dependencia de Microsoft Forms, junto con la incorporación de nuevas funcionalidades, ha permitido a los usuarios realizar encuestas directamente en la plataforma, mejorando la eficiencia y reduciendo la complejidad operativa. La capacitación del equipo aseguró una correcta comprensión y utilización de las nuevas funcionalidades implementadas. Este proyecto ha reforzado la capacidad de la empresa para adaptarse rápidamente a cambios y optimizar continuamente sus procesos internos basándose en datos precisos. En conjunto, estos avances no solo garantizan una infraestructura sólida y funcional, sino que también potencian la gestión de información en la

plataforma, beneficiando a todos los usuarios y contribuyendo al éxito continuo de Valko.

Introducción

Este proyecto se centró en desarrollar e implementar nuevas funcionalidades en Valko, la plataforma de gestión del conocimiento de Bancolombia, con el fin de mejorar la eficiencia operativa y la experiencia del usuario. La dependencia de Microsoft Forms para la elaboración de encuestas generaba complejidades operativas, por lo que se decidió eliminar esta dependencia y permitir la realización de encuestas directamente en Valko. Además, se implementaron filtros avanzados en el apartado de formación para optimizar la gestión de la información.

1 Desarrollo de nuevas funcionalidades en el software de gestión del conocimiento (Valko) de la empresa Bancolombia, haciendo uso de tecnologías Spring Boot.

1.1 Descripción de la empresa

Bancolombia es un grupo financiero líder, con más de 146 años de experiencia y que evoluciona para asumir los desafíos que trae un entorno en constante transformación. Sus acciones tienen como principios básicos la ética y la integridad, con el propósito de promover desarrollo económico sostenible para lograr el bienestar de todos.

Son una organización en crecimiento, rentable, con casi 30,000 colaboradores y presencia en Colombia, Panamá, Guatemala y El Salvador. Desde hace 25 años estamos en la Bolsa de Valores de Nueva York (NYSE), siendo la primera entidad financiera colombiana listada en ese mercado y provee acceso a soluciones financieras y no financieras para personas, emprendedores, pymes, empresas e instituciones, al tiempo que invertimos en iniciativas educativas, de emprendimiento y para el cuidado del medio ambiente que ayuden a generar transformaciones positivas en la sociedad. Tenemos acceso permanente a fondeo en el mercado de capitales local e internacional.(*Conócenos | Grupo Bancolombia*, n.d.)

1.1.1 Misión

Somos el mejor aliado de los clientes en la satisfacción de sus necesidades financieras. Proveemos una amplia gama de productos y servicios con innovación, eficiencia y amabilidad, y

generamos valor a nuestros clientes, colaboradores, accionistas y a la comunidad.(*Gobierno Corporativo | Grupo Bancolombia*, n.d.)

1.1.2 Visión

Ser una Organización comprometida con la excelencia, que satisface las necesidades financieras de los clientes, con soluciones integrales e innovadoras.(*Gobierno Corporativo | Grupo Bancolombia,* n.d.)

1.1.3 Objetivos de la empresa

El mundo enfrenta grandes retos económicos, ambientales y sociales que exigen acciones rápidas de gobiernos, empresas y sociedad civil. Para articular estas acciones, las Naciones Unidas establecieron una agenda global con 17 Objetivos de Desarrollo Sostenible destinados a poner fin a la pobreza, proteger al planeta y garantizar que todas las personas gocen de paz y prosperidad en el 2030.(*Objetivos y Metas de Desarrollo Sostenible - Desarrollo Sostenible*, n.d.)

El sector privado juega un papel importante en la creación de conciencia sobre la adopción de medidas urgentes, la movilización de recursos y la promoción de una economía próspera, compatible con un ambiente saludable y que aporte bienestar a las personas.

En el Grupo Bancolombia nos hemos movilizado para que nuestras acciones tengan un

impacto positivo y aporten a la calidad de vida de los territorios en los que operamos, de manera que la experiencia y capacidades acumuladas por décadas nos permitan contribuir a esa agenda global en forma activa y decidida.

Nuestro rol como apalancadores de la actividad productiva, movilizadores de la economía y facilitadores de oportunidades nos permite aportar desarrollo económico, perdurable y generador de bienestar a las generaciones actuales y futuras.

Son estos compromisos los que definen nuestro propósito, en el que participaron de forma activa nuestros empleados, teniendo presentes las necesidades de la sociedad, nuestro conocimiento, nuestras capacidades, nuestras fortalezas y la pasión por lo que hacemos.

Este propósito, que hoy hacemos explícito y que nos sirve de guía para nuestras actuaciones y decisiones, expresa nuestra esencia, la razón por la cual existimos desde hace más de 145 años: una forma de ser y de actuar que va más allá de una simple narrativa.(*Nuestro Propósito | Grupo Bancolombia*, n.d.)

1.1.4 Descripción de la estructura organizacional.

Figura 1

Estructura Organizacional.



Figura 1, Estructura organizacional (Estructura | Grupo Bancolombia, n.d.)

1.1.5 Descripción de la dependencia y/o proyecto al que fue asignado.

El proyecto busca desarrollar una innovadora funcionalidad en un aplicativo que es clave para la gestión del conocimiento dentro de la empresa. La iniciativa tiene como objetivo eliminar la dependencia de la herramienta Microsoft Forms para la gestión de encuestas de distintos tipos. Con la implementación de esta nueva funcionalidad lo que se busca es consolidar todo el proceso dentro de valko(Herramienta para la gestión del conocimiento interno), permitiendo llevar a cabo las encuestas sin necesidad de recurrir a herramientas externas. Ademas, se busca optimizar la experiencia del usuario eliminando la necesidad de completar datos internos del banco en otra aplicación.

1.2 Diagnóstico inicial de la dependencia asignada.

Tabla 1

Análisis FODA de la dependencia asignada.

	Fortalezas (F)	**Debilidades (D)**
Matriz DOFA	- Manejo de la metodología SCRUM para agilizar el trabajo colaborativo y comunicación. - Herramienta de control de versiones. - Asignación de un mentor técnico.	- No existe documentación sobre el aplicativo. - El ser nuevo ya sea en proyectos nuevos o antiguos lleva a que el tiempo de desarrollo se ve afectado por la gestión del conocimiento.
Oportunidades (O)	**FO**	**DO**
- Acceso a nuevas funcionalidades. - Mejora en la seguridad - Mejorar la experiencia del usuario al eliminar la necesidad de utilizar múltiples aplicaciones.	- Aprovechar la experiencia para desarrollar funciones innovadoras en Valko. - El contar con profesionales expertos sobre las herramientas utilizadas a diario facilita la ampliación del conocimiento.	- Fortalecer las habilidades internas para el desarrollo y mantenimiento de la nueva funcionalidad en Valko. - Explorar e implementar tecnologías que reduzcan la dependencia de Microsoft Forms.
Amenazas (A)	**FA**	**DA**
-Cambios inesperados por parte del dueño del producto.	- Aprovechar las fortalezas del equipo de desarrollo para mitigar los retrasos.	- Identificar y abordar los riesgos asociados a la dependencia de Microsoft

- Retraso en el proyecto.	- Fortalecer la capacidad del equipo para respuesta rápida ante problemas. - Facilidad del manejo de metodologías agiles.	Forms y la complejidad de llenado de datos internos. - Estar al tanto de las tendencias tecnológicas y adaptarse para mantenerse competitivos.

Nota: Elaboración propia

1.2.1 Planteamiento del problema

Actualmente, la elaboración de encuestas en el aplicativo de gestión de conocimiento de Bancolombia (Valko) se ve limitada por la dependencia de Microsoft Forms, una herramienta externa que separa el proceso de recopilación de información y afecta la efectividad operativa. Además, los usuarios finales se encuentran con la tediosa tarea de llenar información interna del banco en otra aplicación, generando una complejidad adicional en el proceso.(*Formularios de Microsoft | Encuestas, Sondeos y Cuestionarios*, n.d.)

La asimilación de una Solución transversal se presenta como una necesidad para superar las limitaciones. En este sentido, se propone el desarrollo de una nueva funcionalidad en Valko, el aplicativo principal para la gestión del conocimiento en Bancolombia. Este avance en el proyecto tiene como objetivo principal eliminar la dependencia de la herramienta externa Microsoft Forms, permitiendo a los usuarios realizar encuestas, y para lograr una mejora adicional en la experiencia del usuario, se incorporará una actividad específica que sería la implementación de funcionalidades que permitan realizar diversos filtrados en el apartado de

formación. Estas funcionalidades ofrecerán opciones de filtrado avanzadas, mejorando así la capacidad de los usuarios para acceder a la información más adecuada para ellos y gestionar información de manera más eficiente.

Desde una perspectiva técnica, se planea trabajar con java 11 con el framewok Spring Boot y el entorno de desarrollo Intellij, utilizando la base de datos actual que se encuentra en el motor de MySQL para garantizar la integridad de la información. Se presentará un esquema de los ajustes propuestospara la base de datos, alineados a los objetivos propios y de los integrantes del equipo, ajustándonos a las prácticas y estándares de la empresa.(*Bota de Primavera*, n.d.; *IntelliJ IDEA: El IDE Líder Para Java y Kotlin*, n.d.; *Mysql*, n.d.; *Software Java | Oracle Colombia*, n.d.)

La implementación de funcionalidades de filtrado en el apartado de formación representa una actividad clave para potenciar la experiencia de los usuarios en Valko. Estas funcionalidades permitirán a los usuarios el acceso fácil a la información relevante y optimiza la interacción con la plataforma.

1.3 Objetivos de la pasantía.

1.3.1 General

Desarrollar nuevas funcionalidades en Valko, que permitirá elaborar encuestas y habilitar nuevos filtros para el apartado de formación siguiendo los criterios de aceptación empleados por el Product Owner(PO).

1.3.2 Específicos

- Analizar los requisitos, el estado actual del aplicativo y el sistema de diseño de la empresa.
- Diseñar la nueva interfaz en figma y los ajustes al modelo de la base de datos en MySQL utilizada por Valko para que se pueda dar soporte a las nuevas funcionalidades.
- Desarrollar las nuevas funcionalidades siguiendo las recomendaciones y criterios planteados por el dueño del producto.

1.4 Descripción de las actividades a desarrollar en la misma

El cronograma está estipulado para un total de 70 días con un horario laboral de lunes a viernes, excluyendo los días festivos.

Tabla 2

Actividades

Objetivo General	Objetivos Específicos	Actividades a desarrollar
Desarrollar nuevas funcionalidades en Valko, que permitirá elaborar encuestas y habilitar nuevos filtros para el apartado de formación siguiendo los criterios de aceptación	Analizar los requisitos, el estado actual del aplicativo y el sistema de diseño de la empresa.	Análisis del estado de la BD.
		Revisión de documentación del sistema de diseño Galatea

empleados por el Product Owner(PO).		analizar los requisitos propuestos
	Diseñar la nueva interfaz en figma y los ajustes al modelo de la base de datos en MySQL utilizada por Valko para que se pueda dar soporte a las nuevas funcionalidades.	Proponer el modelo.
		Diseñar el modelo de Datos.
		Participar en la creación del diseño figma para él equipo front.
	Desarrollar las nuevas funcionalidades siguiendo las recomendaciones y criterios planteados por el dueño del producto.	Codificación de filtros para módulo de formación.
		Codificación funcionalidad Encuestas (CRUD).
		Desarrollar y ejecutar pruebas tanto unitarias como de aceptacion.
		Documentar la forma de uso de EndPoint para entrega al front.

Nota: Elaboración propia

2 Enfoques referenciales

2.1 Enfoques conceptuales

2.1.1 Metodologia Scrum

Scrum es un marco de trabajo ágil para la gestión de proyectos que ayuda a los equipos a trabajar de forma colaborativa y entregar valor de manera incremental. Se basa en ciclos cortos de trabajo llamados sprints, durante los cuales el equipo se enfoca en un conjunto definido de tareas. Al final de cada sprint, se entrega un producto funcional que se puede mostrar a las partes interesadas y obtener retroalimentación.

Valores y principios de Scrum

Scrum se basa en un conjunto de valores y principios que guían la forma en que los equipos trabajan. Estos valores son:

- **Compromiso:** El equipo está comprometido con el éxito del proyecto.
- **Valor:** El equipo se enfoca en entregar valor al cliente.
- **Entrega:** El equipo entrega software funcional con frecuencia.
- **Colaboración:** El equipo trabaja de manera colaborativa para lograr objetivos comunes.
- **Enfoque:** El equipo se enfoca en una cosa a la vez y evita la multitarea.
- **Respeto:** El equipo se respeta mutuamente.
- **Apertura:** El equipo es abierto a la inspección y adaptación.

Roles en Scrum

Scrum define tres roles principales para los equipos:

- **Scrum Master:** El Scrum Master es el facilitador del equipo y ayuda a que el proceso Scrum funcione sin problemas.
    - **Dueño del producto:** El Dueño del producto es responsable de la visión y el backlog del producto.
    - **Equipo de desarrollo:** El equipo de desarrollo es responsable de diseñar, desarrollar y probar el software.

Eventos en Scrum

Scrum define cinco eventos principales que se llevan a cabo a lo largo del ciclo de vida del proyecto:

- **Sprint Planning:** En la reunión de planificación del sprint, el equipo decide qué trabajo se realizará en el próximo sprint.
- **Daily Scrum:** La reunión diaria de Scrum es una reunión breve en la que los miembros del equipo se actualizan sobre su progreso y discuten cualquier obstáculo.
- **Sprint Review:** En la revisión del sprint, el equipo demuestra el trabajo que se completó durante el sprint a las partes interesadas y obtiene retroalimentación.
- **Sprint Retrospective:** En la retrospectiva del sprint, el equipo reflexiona sobre el sprint anterior e identifica áreas para mejorar.
- **Scrum of Scrums:** La reunión de Scrum of Scrums es una reunión para que los equipos de Scrum que trabajan en el mismo producto se coordinen.

Artefactos en Scrum

Scrum define tres artefactos principales que se utilizan para gestionar el trabajo:

- **Product Backlog:** El Product Backlog es una lista ordenada de prioridades de las características del producto.
- **Sprint Backlog:** El Sprint Backlog es una lista de tareas que el equipo se compromete a completar durante el sprint.
- **Burndown Chart:** El Burndown Chart es un gráfico que muestra el progreso del equipo hacia la finalización del Sprint Backlog.(*Guía de Scrum | Guías de Scrum*, n.d.)(Certificación Profesional Scrum MasterTM I | Scrum.Org, n.d.)

2.1.2 Bases de Datos Relacionales

El modelo de base de datos relacional, propuesto por Edgar F. Codd en 1970(Baxendale & Codd, 1970), organiza los datos en tablas que se relacionan entre sí mediante claves primarias y foráneas. Esta estructura permite almacenar y gestionar grandes volúmenes de datos de manera eficiente y consistente. Los principios fundamentales del modelo relacional incluyen:

- Atomicidad: Cada unidad de datos (atributo) es indivisible y debe tener un valor único.
- Consistencia: Los datos deben estar libres de contradicciones y reflejar el estado real del mundo.
- Normalización: Las tablas deben estar estructuradas para minimizar la redundancia y maximizar la integridad de los datos.
- Aislamiento: Las operaciones sobre los datos deben ser independientes entre sí y no afectar la integridad de otros datos.
- Dependencia de datos: Los datos en una tabla deben estar relacionados con los

datos en otra tabla a través de claves foráneas.

- Beneficios de las bases de datos relacionales:
- Escalabilidad: Pueden almacenar y gestionar grandes volúmenes de datos de manera eficiente.
- Flexibilidad: Permiten realizar consultas complejas y recuperar información de manera precisa.
- Integridad de datos: Garantizan la consistencia y confiabilidad de los datos.
- Estándarización: Se basan en un modelo de datos ampliamente aceptado y utilizado.

2.1.3 Seguridad de Datos

La seguridad de datos se refiere a la protección de la información contra accesos no autorizados, divulgación, alteración o destrucción. La tríada CIA (Confidencialidad, Integridad y Disponibilidad) describe los objetivos fundamentales de la seguridad de datos:

- Confidencialidad: La información solo debe ser accesible para las personas o entidades autorizadas.
- Integridad: La información debe ser precisa, completa y consistente.
- Disponibilidad: La información debe estar disponible para los usuarios autorizados cuando la necesiten.

Implementación de la seguridad de datos:

- Controles de acceso: Restringir el acceso a la información mediante mecanismos como autenticación, autorización y control de roles.

- Cifrado: Proteger la información confidencial al codificarla para que solo pueda ser descifrada por usuarios autorizados.
- Protección contra malware: Implementar medidas para prevenir y detectar software malicioso que pueda dañar o robar información.
- Copias de seguridad y recuperación de desastres: Asegurar la disponibilidad de la información mediante la creación de copias de seguridad regulares y planes de recuperación en caso de desastres.(*Effective Cybersecurity: A Guide to Using Best Practices and Standards - William Stallings - Google Libros*, n.d.)

2.1.4 Framework

Un framework de software proporciona una estructura y herramientas para desarrollar aplicaciones de manera eficiente y organizada. Los frameworks promueven la reutilización de código, simplifican el desarrollo y facilitan el mantenimiento de las aplicaciones.

Beneficios de utilizar frameworks:

- Aumento de la productividad: Los frameworks proporcionan componentes prediseñados y herramientas que agilizan el desarrollo.
- Mejora de la calidad del código: Los frameworks promueven buenas prácticas de programación y ayudan a evitar errores comunes.
- Facilitan el mantenimiento: El código estructurado y modularizado que promueven los frameworks facilita la comprensión y modificación del código.
- Reducen la curva de aprendizaje: Los frameworks proporcionan documentación y comunidades de apoyo que facilitan el aprendizaje y la adopción de nuevas

tecnologías.

Ejemplo de framework popular:

- Spring Boot: Un framework de Java para desarrollar aplicaciones web backend robustas y escalables. Spring Boot simplifica el desarrollo al proporcionar configuraciones predeterminadas, herramientas integradas para la gestión de dependencias y la configuración del servidor. (*Spring Boot in Action - Craig Walls - Google Libros*, n.d.)

En este proyecto, se utilizará Spring Boot para desarrollar las nuevas funcionalidades de encuestas en Valko(*Spring Boot*, n.d.)

2.1.5 Pruebas Unitarias

Las pruebas unitarias son una técnica fundamental en el desarrollo de software que verifica el correcto funcionamiento de unidades individuales de código, como funciones, módulos o clases. Estas pruebas se ejecutan de forma aislada y se enfocan en validar el comportamiento de cada unidad de código de manera independiente.

Objetivos de las pruebas unitarias:

- Detectar errores en etapas tempranas del desarrollo: Al realizar las pruebas unitarias durante la fase de codificación, se pueden identificar errores de manera temprana, reduciendo significativamente los costos y el tiempo de corrección en etapas posteriores del ciclo de vida del software.
- Mejorar la calidad del software: Las pruebas unitarias garantizan que cada unidad de código funcione correctamente, lo que contribuye a un código más robusto,

confiable y menos propenso a errores.

- Facilitar el mantenimiento: Un código bien probado con pruebas unitarias es más fácil de entender, modificar y mantener, ya que los cambios realizados en una unidad de código no afectarán negativamente a otras partes del sistema.

Beneficios de las pruebas unitarias:

- Reducción de costos: Al detectar errores temprano, se evitan costos asociados a la depuración y corrección en etapas posteriores del desarrollo.
- Mejora de la productividad: Un código bien probado permite a los desarrolladores trabajar de manera más eficiente y segura, ya que tienen la confianza de que cada unidad de código funciona correctamente.
- Mayor satisfacción del cliente: Un software con menos errores y mayor calidad genera mayor satisfacción en los usuarios finales.

Metodologías para pruebas unitarias:

- Test-Driven Development (TDD): Enfoque en el que las pruebas unitarias se escriben antes de la implementación del código, lo que guía el desarrollo y garantiza que cada unidad de código cumpla con los requisitos. (Fraser et al., 2003)
- Behavior-Driven Development (BDD): Enfoque en el que las pruebas unitarias se escriben desde la perspectiva del comportamiento deseado del software, promoviendo una mejor comprensión de los requisitos y una mayor colaboración entre desarrolladores y usuarios.

Herramientas para pruebas unitarias:

- JUnit: Framework popular para pruebas unitarias en Java.(*JUnit 5*, n.d.)(*Mockito*

Framework Site, n.d.)

- PyUnit: Framework popular para pruebas unitarias en Python.
- xUnit: Framework popular para pruebas unitarias en .NET.

2.1.6 Modelado de Base de Datos

El modelado de base de datos es un proceso fundamental en el diseño de sistemas de información que involucra la creación de una representación gráfica o conceptual de la estructura lógica y física de una base de datos. Este proceso involucra identificar las entidades, sus atributos, las relaciones entre ellas y las restricciones que aplican a los datos.

Objetivos del modelado de base de datos:

- Asegurar la integridad de los datos: Un modelo de base de datos bien diseñado garantiza que los datos almacenados sean precisos, consistentes y completos, minimizando la redundancia y los errores.
- Optimizar el rendimiento de las consultas: Un modelo de base de datos eficiente permite realizar consultas sobre los datos de manera rápida y efectiva, satisfaciendo las necesidades de los usuarios.
- Facilitar la comprensión y el mantenimiento: Un modelo de base de datos claro y documentado facilita la comprensión del sistema por parte de desarrolladores, administradores y usuarios, simplificando el mantenimiento y las modificaciones futuras.

Etapas del modelado de base de datos:

- Análisis de requisitos: Comprender las necesidades del negocio y los requisitos de

los usuarios para identificar las entidades, atributos y relaciones relevantes.

- Diseño conceptual: Crear un modelo conceptual que represente la estructura lógica de la base de datos, utilizando diagramas de entidad-relación (E-R) o notación UML.
- Diseño físico: Traducir el modelo conceptual a una representación física que detalle la implementación de la base de datos en un sistema de gestión de bases de datos (SGBD) específico.

Metodologías para el modelado de base de datos:

- Modelado orientado a objetos: Enfoque basado en los conceptos de la programación orientada a objetos, utilizando clases, objetos y relaciones entre ellos para representar la estructura de la base de datos.
- Modelado relacional: Enfoque tradicional basado en el modelo de base de datos relacional, utilizando tablas, columnas, claves primarias y relaciones entre tablas para representar la estructura de la base de datos.

Herramientas para el modelado de base de datos:

- Herramientas de diagramación: Permiten crear diagramas de entidad-relación (E-R) y otros diagramas para visualizar la estructura de la base de datos.
- Herramientas de diseño de bases de datos: Facilitan la creación y gestión de modelos de base de datos, incluyendo la generación de código SQL.

(De & Hall, n.d.)

2.1.7 Pruebas de Aceptación

Las pruebas de aceptación, también conocidas como pruebas de validación, son un tipo de prueba de software que verifica si el sistema cumple con los requisitos y expectativas del usuario final. Estas pruebas se realizan en las últimas etapas del desarrollo del software, una vez que las pruebas unitarias y de integración se han completado satisfactoriamente.(José et al., n.d.)

Objetivos de las pruebas de aceptación:

- Validar que el software cumple con los requisitos funcionales: Asegurar que el software cumple con las funcionalidades y comportamientos esperados por los usuarios finales.
- Verificar que el software es usable y aceptable para los usuarios: Evaluar la facilidad de uso, la interfaz de usuario y la experiencia general del usuario con el software.
- Garantizar que el software cumple con los criterios de aceptación definidos: Asegurar que el software cumple con los estándares de calidad y los criterios establecidos por el Product Owner o los stakeholders.

Tipos de pruebas de aceptación:

- Pruebas de caja negra: Pruebas que se realizan sin conocimiento interno del código o la arquitectura del software.
- Pruebas de caja blanca: Pruebas que se realizan con conocimiento del código o la arquitectura del software, utilizando técnicas como la cobertura de código.
- Pruebas de usabilidad: Pruebas que se enfocan en evaluar la facilidad de uso, la interfaz de usuario y la experiencia general del usuario con el software.

Metodologías para las pruebas de aceptación:

- Pruebas de aceptación guiadas por el comportamiento (BDD): Enfoque en el que las pruebas se escriben desde la perspectiva del comportamiento deseado del software, promoviendo una mejor comprensión de los requisitos y una mayor colaboración entre desarrolladores y usuarios.
- Pruebas de aceptación de la TDD: Enfoque en el que las pruebas de aceptación se escriben antes de la implementación del software, guiando el desarrollo y asegurando que el software cumpla con los requisitos desde el principio.

2.2 Enfoques legales

2.2.1 Proteccion de datos personales

La Ley Estatutaria 1581 de 2012, también conocida como Ley de Habeas Data, establece el marco legal para la protección de datos personales en Colombia. Esta ley define los principios rectores para el tratamiento de datos personales, los derechos de los titulares de los datos y las obligaciones de los responsables y encargados del tratamiento.(*Política de Protección de Datos Personales* -, n.d.)

Principios rectores para el tratamiento de datos personales:

- Legalidad: Los datos personales solo pueden ser tratados si existe una base legal para ello, como el consentimiento del titular, una obligación legal o la protección de los intereses legítimos del responsable del tratamiento.
- Calidad: Los datos personales deben ser exactos, completos, adecuados y

pertinentes para el propósito para el cual se recolectan.

- Finalidad: Los datos personales solo pueden ser tratados para fines específicos, explícitos e informados al titular.
- Proporcionalidad: El tratamiento de datos personales debe ser proporcional al propósito para el cual se recolectan.
- Seguridad: Los datos personales deben ser tratados de manera segura para evitar su pérdida, alteración, acceso no autorizado o destrucción.
- Transparencia: El responsable del tratamiento debe informar al titular de los datos sobre cómo se recolectan, usan y procesan sus datos personales.

Recursos en caso de incumplimiento:

Los titulares de datos que consideren que sus derechos han sido vulnerados pueden presentar una queja ante la Superintendencia de Industria y Comercio.(*SIC | Sede Electronica*, n.d.)

2.2.2 Ley de derechos de autor y propiedad intelectual

La Ley 23 de 1982 en Colombia regula los derechos de autor y la propiedad intelectual. Su objetivo principal es proteger las obras literarias y artísticas, así como los derechos de sus autores o creadores. Establece las condiciones bajo las cuales se pueden utilizar, reproducir y distribuir obras protegidas por derechos de autor.(*Ley 23 de 1982 - Gestor Normativo - Función Pública*, n.d.)

2.2.3 Accesibilidad a la web

La Ley 1680 de 2013 en Colombia regula la accesibilidad a las tecnologías de la información y las comunicaciones para las personas con discapacidad. Establece normas para que los sitios web y las aplicaciones sean accesibles para todos los usuarios, independientemente de sus capacidades físicas o sensoriales.(*Ley 1680 de 2013 - Gestor Normativo - Función Pública*, n.d.)

2.2.4 Seguridad informática

La Ley 1273 de 2009 en Colombia define los delitos informáticos y establece normas para combatirlos. Regula aspectos como el acceso no autorizado a sistemas informáticos, la interceptación de datos electrónicos y la falsificación informática, entre otros. Promueve la seguridad de la información y establece penas para quienes cometan delitos informáticos.(*Ley 1273 de 2009 - Gestor Normativo - Función Pública*, n.d.)

2.2.5 Norma iso 27001

ISO 27001 es una norma internacional que establece los requisitos para un sistema de gestión de seguridad de la información (SGSI). La norma ISO 27001 es ampliamente reconocida y adoptada internacionalmente como un estándar de buenas prácticas en seguridad de la información.(*Sistemas de Gestión de Seguridad |ICONTEC*, n.d.)

3 Informe de cumplimiento

3.1 Presentación de resultados

3.1.1 Formación

Durante mi pasantía en Bancolombia, tuve la oportunidad de participar en un programa de formación integral que me permitió adquirir conocimientos y habilidades en diversas áreas clave para el desarrollo de software, con un enfoque particular en las prácticas de DevOps, implementación de ramas, metodologías Scrum y pruebas de software. Este programa de formación fue fundamental para mi proceso de aprendizaje y me brindó la oportunidad de alinearme con las prácticas y metodologías que utiliza la empresa en sus proyectos de desarrollo.

El programa de formación incluyó una serie de cursos que me permitieron adquirir conocimientos y habilidades en las siguientes áreas:

DevOps: Este curso me introdujo a los principios y prácticas de DevOps, una metodología que busca integrar el desarrollo y las operaciones de software para mejorar la agilidad, la eficiencia y la calidad del software. Aprendí sobre diferentes herramientas y técnicas de DevOps, como la integración continua, la entrega continua y la monitorización.

Implementación de ramas: Este curso me enseñó a utilizar el control de versiones Git para crear y gestionar ramas de desarrollo de software. Aprendí a crear ramas para trabajar en diferentes características o correcciones de errores sin afectar el código principal, y a fusionar las ramas de manera segura una vez que se completa el trabajo.

Metodologías Scrum: Este curso me introdujo a la metodología Scrum, una metodología

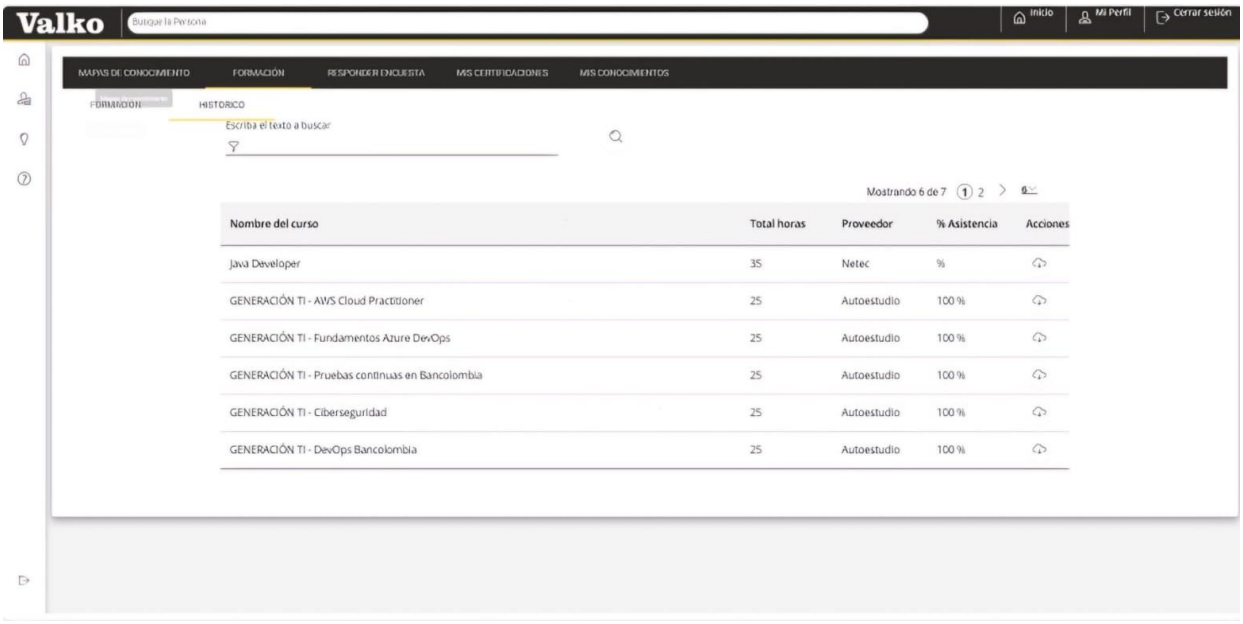
de gestión de proyectos ágil que se basa en ciclos de desarrollo cortos e iterativos. Aprendí sobre los roles y responsabilidades del equipo Scrum, así como sobre las diferentes ceremonias y prácticas de Scrum, como la planificación del sprint, la revisión diaria y la retrospectiva.

Pruebas de software: Este curso me enseñó a diseñar y ejecutar pruebas de software para verificar la calidad y el correcto funcionamiento del software. Aprendí sobre diferentes tipos de pruebas, como pruebas unitarias, pruebas de integración y pruebas de aceptación, y sobre diferentes herramientas y técnicas de pruebas.

En la siguiente Imagen se dan a conocer los nombres de las tres primeras formaciones que me fueron asignadas, durante el proceso de pasantías se asignaron otras que son internas de la organización.

Figura 2

Valko



Nota. Autor (2024)

3.1.2 Análisis del estado de la base de datos

En esta fase del proyecto se realizó un análisis exhaustivo de la estructura actual de la base de datos (BD) del sistema. Este análisis involucró la identificación y comprensión de las diferentes tablas, sus relaciones y los datos almacenados en cada una de ellas. El objetivo principal era determinar las tablas y campos relevantes para el desarrollo de las nuevas funcionalidades de encuestas post-curso, sin afectar la integridad y el funcionamiento del sistema existente.

A través del análisis detallado, se identificaron aproximadamente 100 tablas dentro del sistema. Se prestó especial atención a aquellas tablas que almacenan información crítica para el desarrollo de las nuevas funcionalidades, especialmente las que contienen datos relacionados con:

Personas: Esta tabla almacena información esencial sobre los usuarios del sistema, como nombres, direcciones, correos electrónicos y datos de contacto. Estos datos son fundamentales para la identificación y selección de los participantes en las encuestas post-curso.

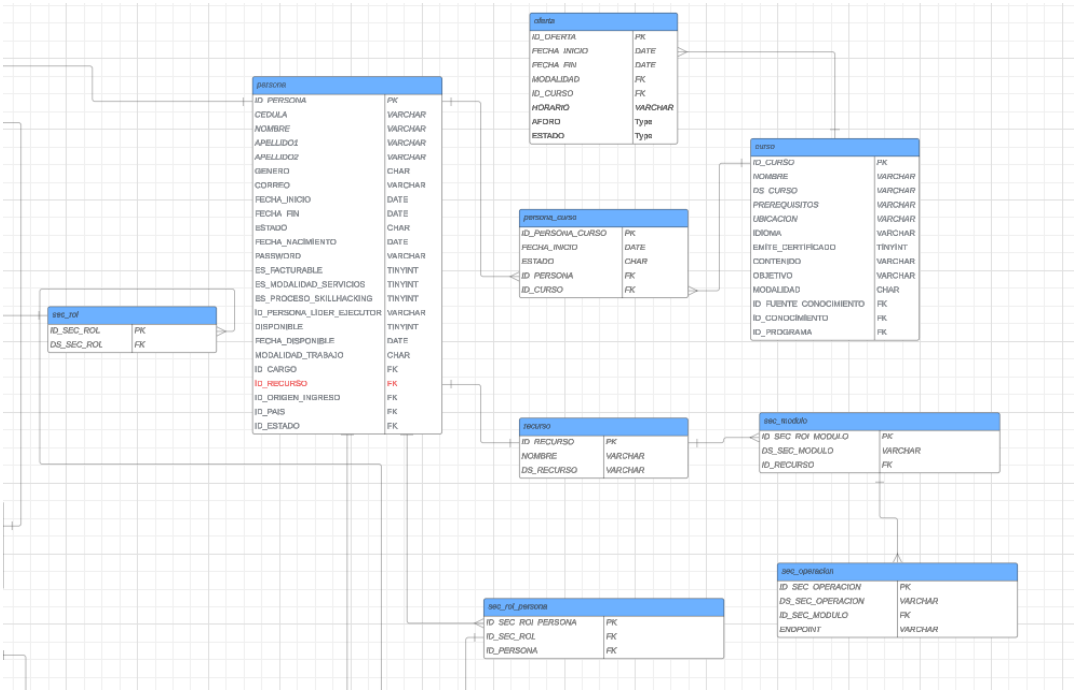
Roles: Esta tabla almacena información sobre los diferentes roles que los usuarios pueden desempeñar dentro del sistema, como administrador, instructor o estudiante. Esta información es crucial para determinar los permisos de acceso a las encuestas y para segmentar la distribución de las mismas.

Registros de cursos: Esta tabla almacena información sobre los cursos realizados por los usuarios, incluyendo el nombre del curso, la fecha de realización y la calificación obtenida. Estos datos son esenciales para asociar las encuestas a los cursos específicos y para analizar la participación y satisfacción de los usuarios.

Entre otras tablas encontramos las que inician por SEC donde se almacena información sobre el cargo que ejercen dentro de la organización las diferentes personas, como lo son Analistas, ingenieros, desarrolladores, dueños de producto y cualquier variedad de cargo en toda la organización y las tablas de rol, donde esta almacenada la información del tipo de rol que cumple cada empleado dependiendo de su cargo, es decir que si su cargo es desarrollador sus roles podrían ser Backend, Frontend, entre otros.

Figura 3

Estado base de datos



Nota. Autor (2024)

3.1.3 Revisión de documentación del sistema de diseño Galatea

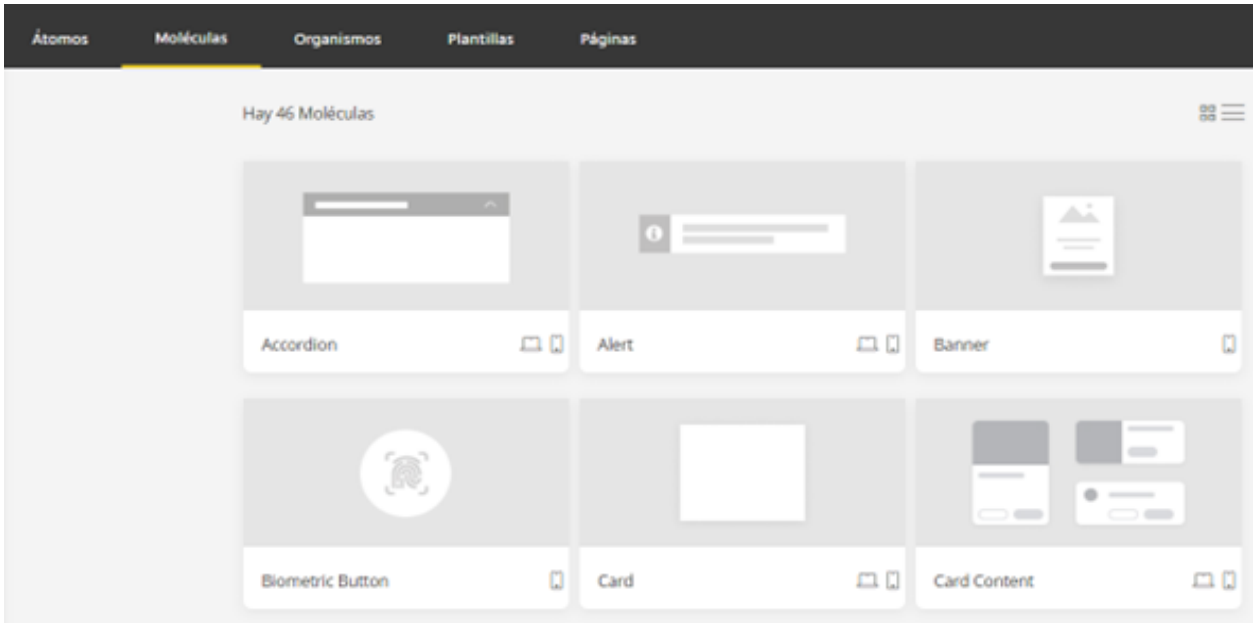
Esta actividad se centró en la revisión exhaustiva de la documentación del sistema de

diseño Galatea, un repositorio web interno crucial que alberga los diseños tanto FrontEnd como BackEnd de la empresa. Este sistema es fundamental para mantener una identidad coherente en todos los desarrollos organizacionales.

En el apartado de FrontEnd, se identificaron varios tipos de componentes, clasificados como átomos, moléculas, organismos, plantillas y páginas. Estos comprenden desde elementos individuales como botones hasta componentes completos como barras de navegación. Este hallazgo facilitó la integración de diseños estandarizados y cohesivos en las nuevas funcionalidades del proyecto.

Figura 4

Sistema de diseño

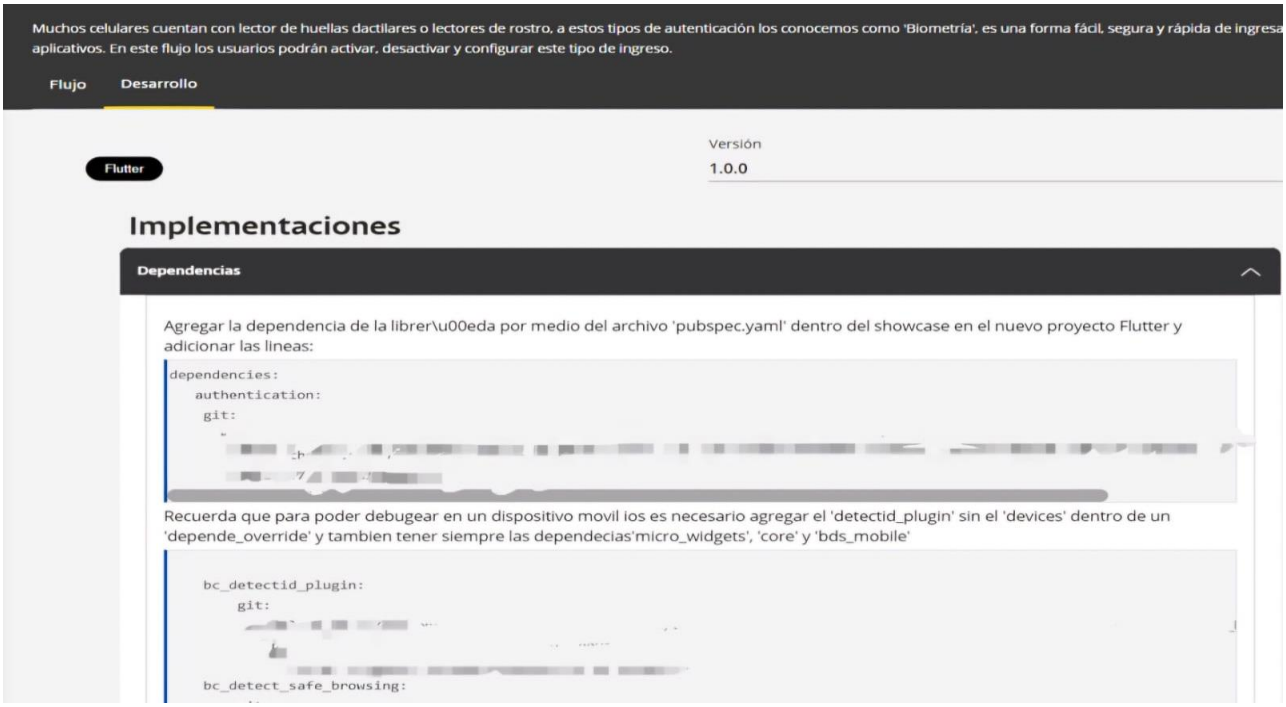


Nota. Autor (2024)

Por otro lado, en el apartado de BackEnd, se revisó el código existente que puede ser reutilizado y adaptado para los desarrollos necesarios. Además, se identificaron dependencias diversas que son utilizadas internamente para garantizar la eficiencia y coherencia en los desarrollos de software.

Figura 5

Sistema de diseño Back



Nota. Autor (2024)

Esta revisión de la documentación del sistema de diseño Galatea proporcionó una base sólida para el diseño de ajustes en la base de datos y la implementación de las nuevas funcionalidades del proyecto, asegurando una integración fluida y coherente con los estándares establecidos por la empresa.

3.1.4 Análisis de los requisitos propuestos

Durante esta fase, se realizó un análisis detallado de los requisitos específicos del proyecto, enfocándose en las siguientes funcionalidades clave:

En la implementación de la propuesta de diseño visual utilizando el Sistema de Diseño de Bancolombia, Se determinó la necesidad de integrar el Sistema de Diseño de Bancolombia para asegurar consistencia y estandarización en la interfaz de usuario del aplicativo. Esto incluye la adopción de principios de diseño centrados en la experiencia del usuario y la integración de componentes visuales predefinidos para mejorar la usabilidad y accesibilidad del sistema.

En la habilitación de la funcionalidad de ver el historial de cursos de una persona, Se identificó la necesidad de desarrollar una función que permita a los usuarios revisar el historial completo de los cursos en los que han participado. Esta característica es crucial para proporcionar a los usuarios una visión integral de su progreso y participación en actividades formativas.

En la implementación de filtros avanzados para las ofertas de cursos, Se especificó la necesidad de habilitar opciones de filtrado avanzado para las ofertas de cursos. Estos filtros deben permitir a los usuarios buscar ofertas según criterios específicos como el cargo al que están orientadas, el conocimiento relacionado y el nivel de la oferta. Esta funcionalidad mejorará la capacidad de los usuarios para encontrar cursos relevantes de manera eficiente.

En la eliminación de la dependencia de herramientas externas como Forms, Se incluyó la

tarea de eliminar la dependencia de herramientas externas, particularmente de Forms, mediante el desarrollo de funcionalidades internas que permitan la captura y gestión de datos de encuestas de manera integrada en el sistema. Esto optimizará la eficiencia y la seguridad de la plataforma.

En la garantía del diligenciamiento de encuestas por parte de los asistentes a los programas de formación: Se estableció la necesidad de implementar un proceso que asegure el diligenciamiento efectivo de encuestas por parte de los participantes en los programas de formación. Esta funcionalidad facilitará la recopilación de retroalimentación valiosa para mejorar continuamente la calidad de los programas ofrecidos.

Uso transversal del módulo de encuestas: Se subrayó la importancia de poder utilizar el módulo de encuestas de manera transversal en diferentes contextos dentro del aplicativo. Esto permitirá a los usuarios administrar y desplegar encuestas de forma flexible, adaptándose a diversas necesidades organizacionales y de formación.

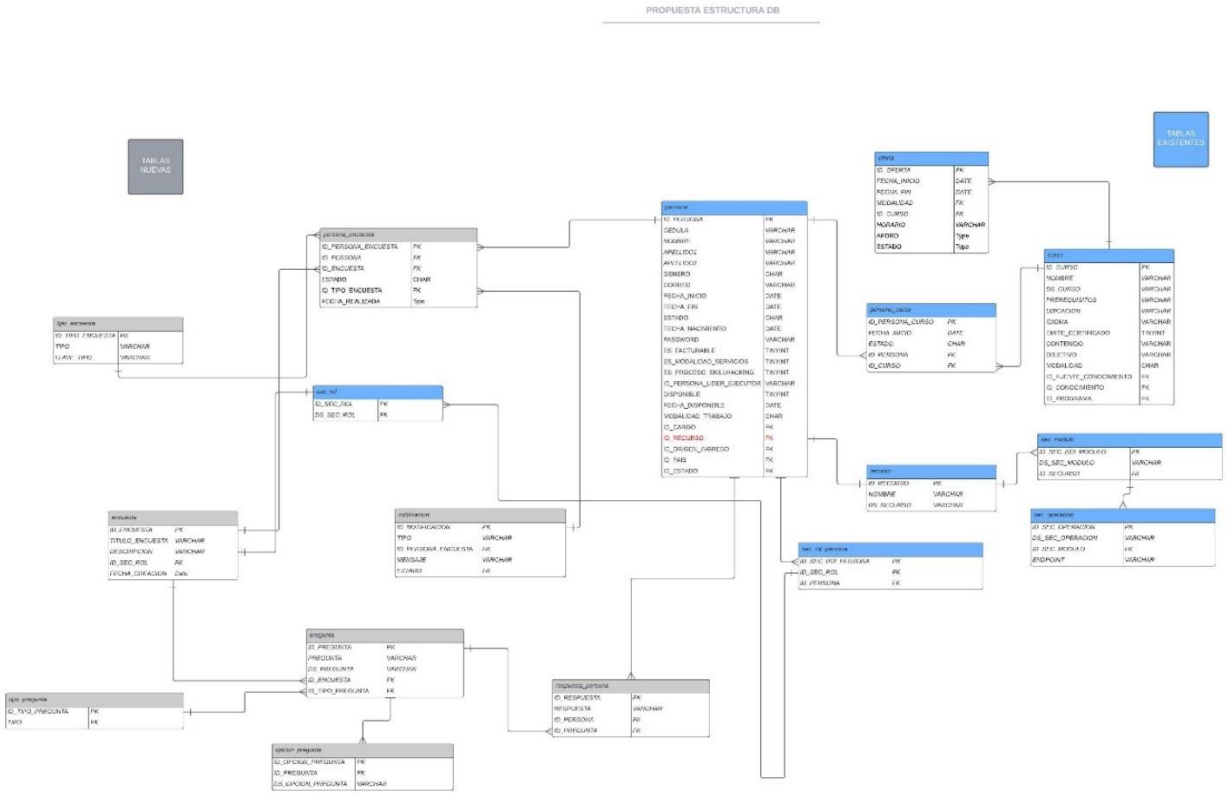
3.1.5 Propuesta y diseño del modelo de datos

El primer paso fundamental fue realizar un análisis exhaustivo de los requisitos del sistema para comprender las necesidades de almacenamiento y gestión de datos para las encuestas. Esto involucró identificar los tipos de datos que se debían almacenar, como preguntas de la encuesta, opciones de respuesta, datos demográficos de los participantes y respuestas a las encuestas. Además, se definieron las relaciones entre las entidades del modelo de datos, como encuestas, preguntas, opciones de respuesta, participantes y cursos.

Con base en el análisis de requisitos, se procedió a diseñar un modelo de datos relacional que cumpliera con los principios de normalización y eficiencia. Se crearon diagramas de entidad-relación (E-R) para visualizar la estructura del modelo de datos y se definieron las características de cada tabla, incluyendo los campos, tipos de datos, claves primarias, claves foráneas y restricciones. La documentación clara y concisa del modelo de datos facilitó su comprensión y uso posterior.

Figura 6

Base de datos con los nuevos cambios.



Nota. Autor (2024)

Una vez finalizado el diseño del modelo de datos, se procedió a implementarlo en la base

de datos existente de Valko. Esto involucró la creación de las tablas necesarias, el establecimiento de las relaciones entre las tablas mediante claves foráneas y la implementación de las restricciones de integridad de datos para garantizar la coherencia y calidad de los datos. Finalmente, se realizaron pruebas exhaustivas para verificar el correcto funcionamiento del modelo de datos.

3.1.6 Diseño en Figma

Para asegurar un diseño de interfaz coherente con las necesidades del equipo de FrontEnd y la identidad visual de la empresa, se estableció una estrecha colaboración con ellos. Se participaron activamente en sesiones de trabajo colaborativo para discutir y conceptualizar el diseño de la interfaz, aportando ideas y sugerencias para mejorar la usabilidad, la estética y la coherencia con el Sistema de Diseño de la organización.

A través de lluvias de ideas y prototipos de baja fidelidad, se exploraron diferentes ideas y conceptos para el diseño de la interfaz. Se consideraron los principios del Sistema de Diseño de la organización, como la paleta de colores, la tipografía y los elementos de interfaz, para lograr un diseño atractivo, intuitivo y coherente.

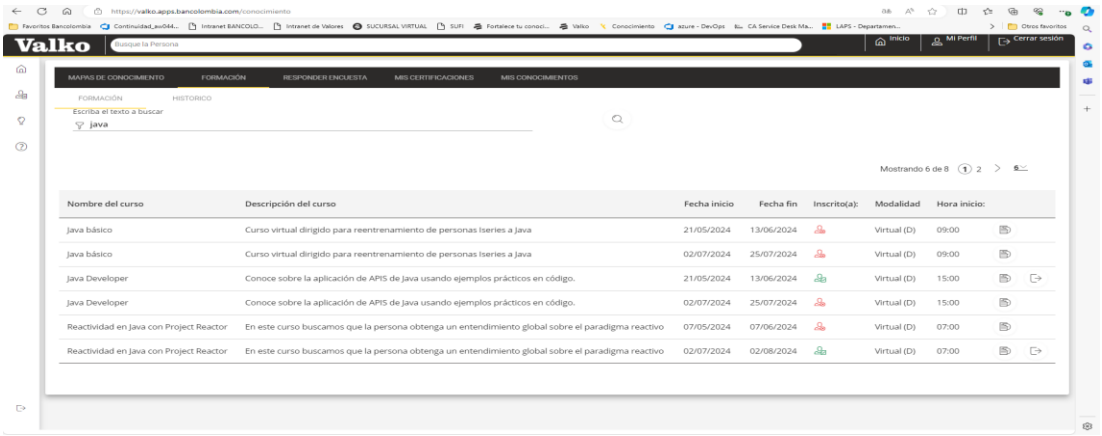
Utilizando la herramienta Figma, se crearon prototipos de alta fidelidad del diseño de la interfaz, refinando los detalles visuales como la disposición de los elementos, la tipografía y los colores. Se aseguró que el diseño final fuera atractivo, intuitivo y coherente con la identidad visual de la empresa.

El diseño final se presentó al equipo de FrontEnd y a otras partes interesadas para obtener retroalimentación y realizar ajustes y mejoras en función de las sugerencias recibidas.

Finalmente, se documentó el diseño final para facilitar su implementación por parte del equipo de FrontEnd. Imagen de Referencia de cómo se encontraba el Diseño anterior.

Figura 7

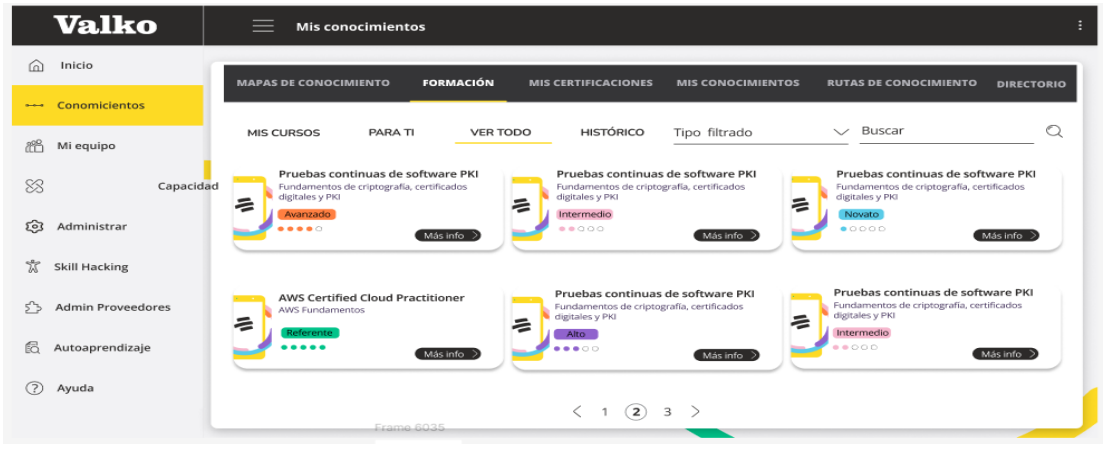
Diseño anterior



Nota. Autor (2024)

Figura 8

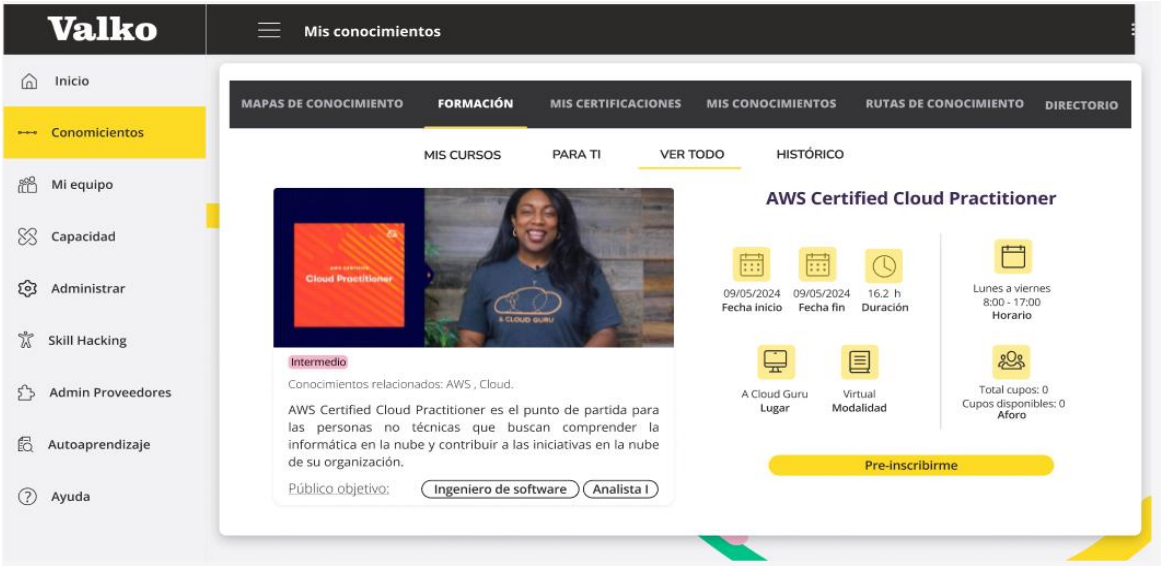
Capturas de los nuevos diseños de todo el apartado de formación.



Nota. Autor (2024)

Figura 9

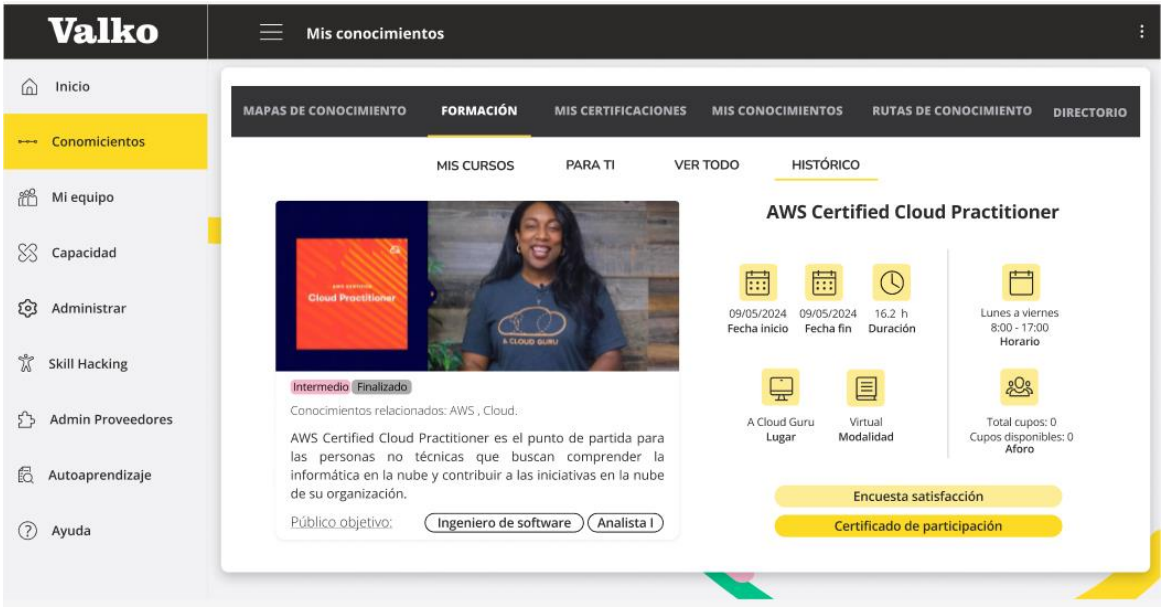
Diseños



Nota.Autor (2024)

Figura 10

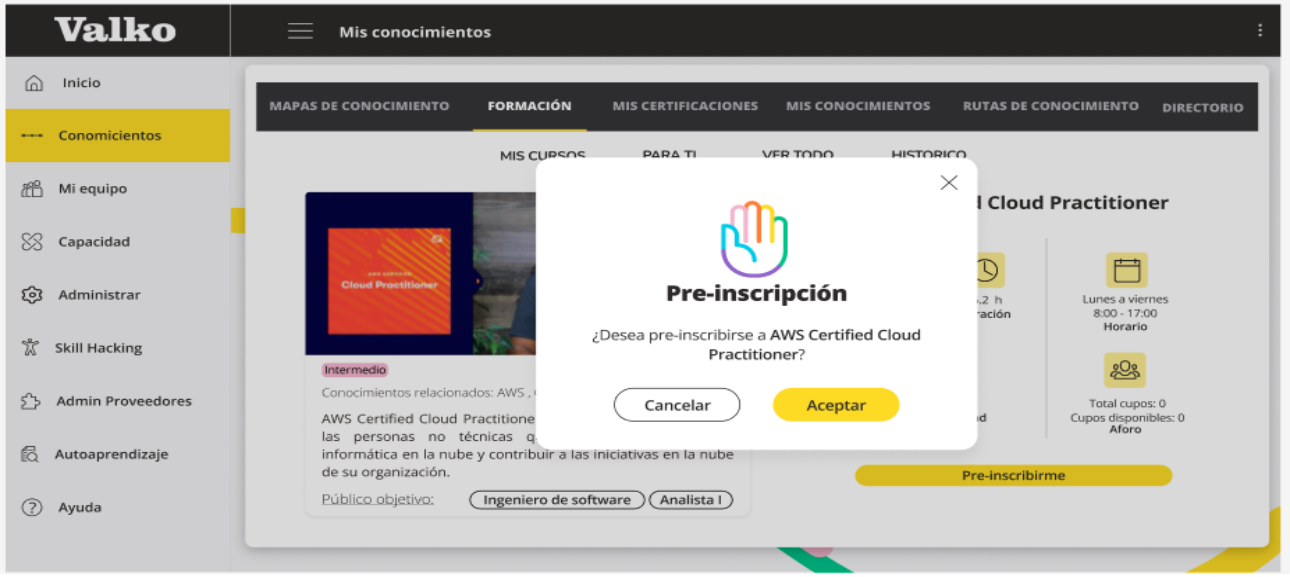
Diseños



Nota. Autor (2024)

Figura 11

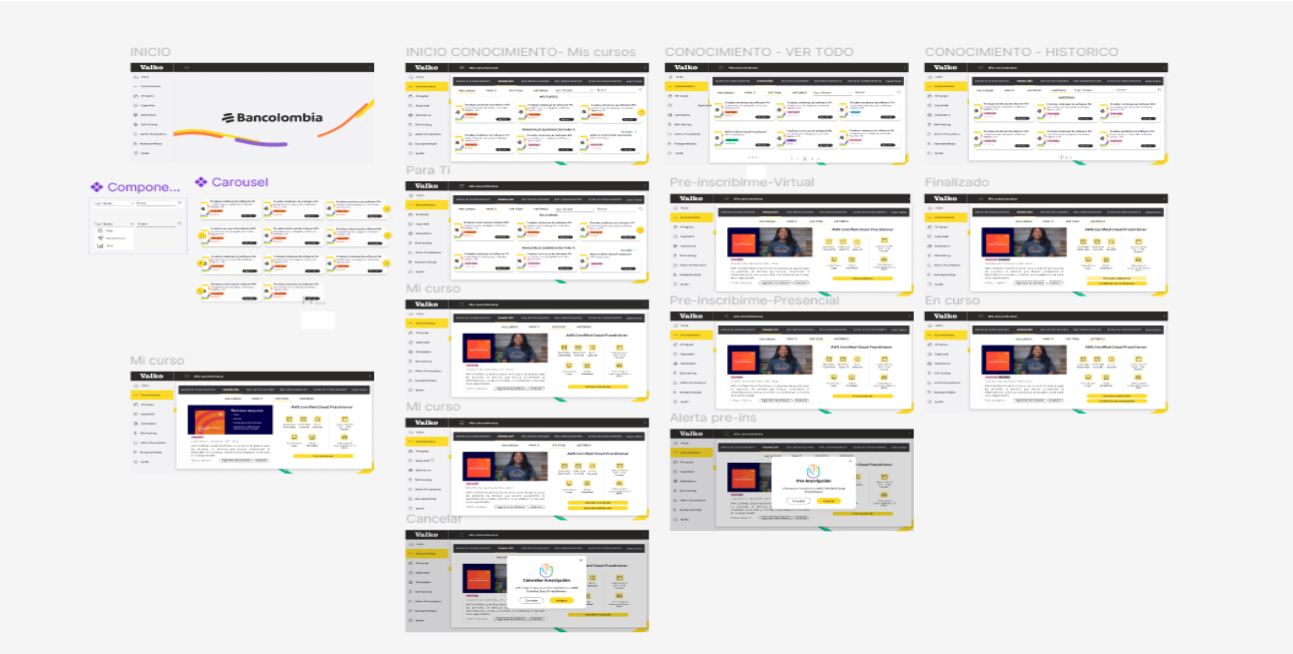
Diseños



Nota. Autor (2024)

Figura 12

Toma General de los Diseños



Nota. Autor (2024)

3.1.7 Codificación de filtros para módulo de formación

En este apartado, se detallará el desarrollo del backend del módulo de formación. Se explicará la estructura del código, desde el controller hasta el repository, donde se realizan las consultas a la base de datos. A continuación, se presentarán las capturas de pantalla del código implementado en cada una de estas capas.

Estructura del Backend

El desarrollo del backend se divide en tres clases principales: el controlador, el servicio y el repository. A continuación, se describe cada uno de estos componentes.

Controller

El controlador es el punto de entrada para las solicitudes HTTP. Se encarga de recibir las solicitudes del cliente, procesarlas y delegar la lógica de negocio al servicio correspondiente. A continuación, se muestra una captura de pantalla del código del controlador.

Figura 13

Código Controller

```
Andres Felipe Montaño Sanchez
@GetMapping("/listMyCourseOffer")
public ResponseEntity<ControllerDto> ListMyCourseOffert(@RequestParam("idPersona") Persona idPersona,
        @RequestParam(value = "cargo", required = false) String cargo,
        @RequestParam(value = "conocimiento", required = false) String conocimiento,
        @RequestParam(value = "nivel", required = false) String nivel,
        @RequestParam(value = "page", required = false) Integer page,
        @RequestParam(value = "limit", required = false) Integer limit){
    ServiceResponseDto serviceResponseDto = ofertaService.ListOffferMyCourse(idPersona,cargo, conocimiento, nivel, page, limit);
    ControllerDto controllerDto = new ControllerDto();
    controllerDto.setStatusCode(serviceResponseDto.getStatusCode());
    controllerDto.setBody(serviceResponseDto.getStatusCode(), serviceResponseDto.getData(), true);
    return ResponseEntity.status(HttpStatus.OK).body(controllerDto);
}
```

Nota. Autor (2024)

Servicio

El servicio contiene la lógica de negocio de la aplicación. Aquí se gestionan las operaciones necesarias para cumplir con los requisitos de la solicitud recibida. El servicio interactúa con la capa repository para realizar las operaciones de acceso a datos. A continuación, se presenta una captura del código del servicio.

Figura 14

Código servicio.



```
Andres Felipe Montaño Sanchez *
@Override
@Transactional(readOnly = true)
public ServiceResponseDto ListOfertasByCharge(String idPersona , String conocimiento, String nivel, Integer page, Integer limit){

    int statusCode = Constants.SUCCESS_STATUS_CODE;
    Map<String, Object> data = new LinkedHashMap<String, Object>();
    try {
        String nombreIdPerfil = perfilRepository.findName(idPersona);
        List<OfferCursoPerfilDto> offerObject = ofertaHome.findByChargee(nombreIdPerfil, conocimiento, nivel, page, limit);
        Integer total = ofertaHome.findByChargee(nombreIdPerfil, conocimiento, nivel,null,null).size();
        data.put("objects", offerObject);
        data.put("total", total);

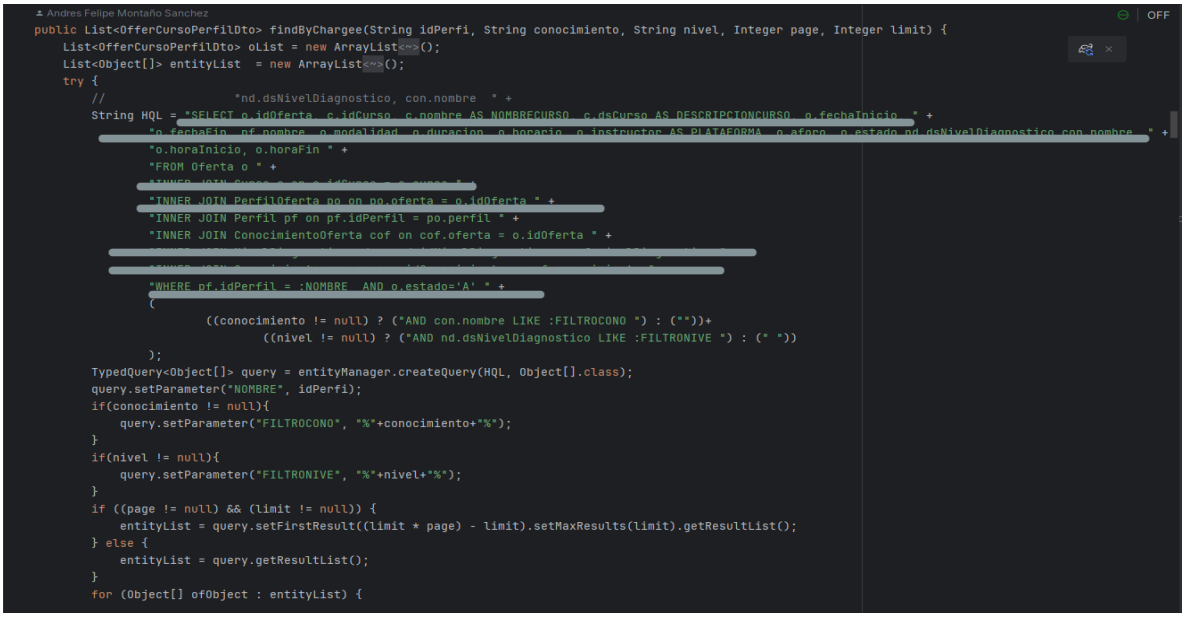
    } catch (Exception error) {
        statusCode = Constants.INTERNAL_SERVER_ERROR_STATUS_CODE;
        data.put("message", "Error al listar la oferta");
        log.error("listOferta failed", error);
    }
    ServiceResponseDto serviceResponseDto = new ServiceResponseDto();
    serviceResponseDto.setStatusCode((int) statusCode);
    serviceResponseDto.setData((Map<String, Object>) data);
    return (serviceResponseDto);
}
```

Nota. Autor (2024)

La clase repository es responsable de la interacción directa con la base de datos. En esta capa se realizan las consultas, actualizaciones y otras operaciones de manipulación de datos. A continuación, se muestra una captura del código del repository.

Figura 15

Código repository.



```
Andres Felipe Montaño Sanchez
public List<OfferCursoPerfilDto> findByChargee(String idPerfi, String conocimiento, String nivel, Integer page, Integer limit) {
    List<OfferCursoPerfilDto> oList = new ArrayList<>();
    List<Object[]> entityList  = new ArrayList<>();
    try {
        //                  "nd.dsNivelDiagnostico, con.nombre  " +
        String HQL = "SELECT o.idOferta, c.idCurso, c.nombre AS NOMBRECURSO, c.dsCurso AS DESCRIPCIONCURSO, o.fechaInicio, " +
                "o.fechaFin, pf.nombre, o.modalidad, o.ducacion, o.hocacio, o.instructor AS PLATAFORMA, o.aforo, o.estado,nd.dsNivelDiagnostico,con.nombre, " +
                "o.horaInicio, o.horaFin " +
                "FROM Oferta o " +
                "INNER JOIN PerfilOferta po on po.oferta = o.idOferta " +
                "INNER JOIN Perfil pf on pf.idPerfil = po.perfil " +
                "INNER JOIN ConocimientoOferta cof on cof.oferta = o.idOferta " +
                "WHERE pf.idPerfil = :NOMBRE  AND o.estado='A' " +
                (
                        ((conocimiento != null) ? ("AND con.nombre LIKE :FILTROCONO ") : (""))+
                                ((nivel != null) ? ("AND nd.dsNivelDiagnostico LIKE :FILTRONIVE ") : (" "))
                );
        TypedQuery<Object[]> query = entityManager.createQuery(HQL, Object[].class);
        query.setParameter("NOMBRE", idPerfi);
        if(conocimiento != null){
            query.setParameter("FILTROCONO", "%"+conocimiento+"%");
        }
        if(nivel != null){
            query.setParameter("FILTRONIVE", "%"+nivel+"%");
        }
        if ((page != null) && (limit != null)) {
            entityList = query.setFirstResult((limit * page) - limit).setMaxResults(limit).getResultList();
        } else {
            entityList = query.getResultList();
        }
        for (Object[] ofObject : entityList) {
```

Nota. Autor (2024)

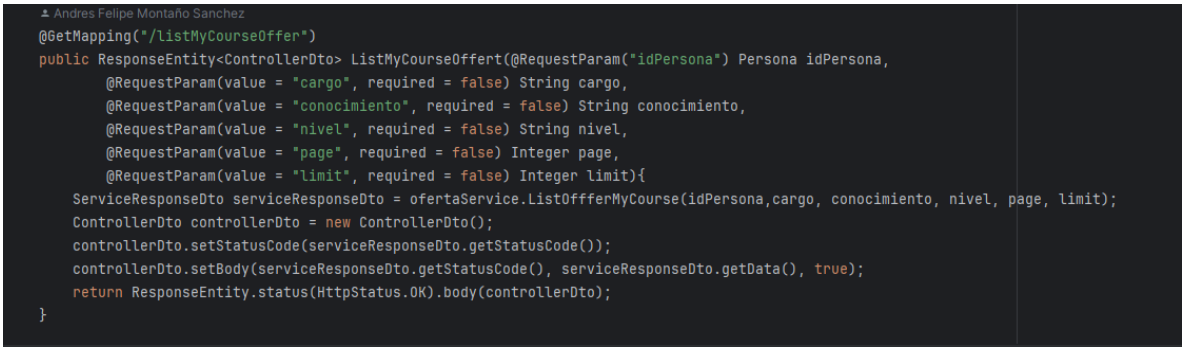
A continuación describo brevemente el código realizado:

En primera instancia, se preparó el código de la clase Controller, que actúa como el punto de entrada para las solicitudes realizadas por el equipo de Frontend. En esta clase, se definió el tipo de petición, los parámetros a recibir mediante el request y se ajustó la respuesta utilizando un JSON estructurado con las clases DTO y el código de estado.

Controller para listado de cursos activos del empleado

Figura 16

Código lista de cursos.

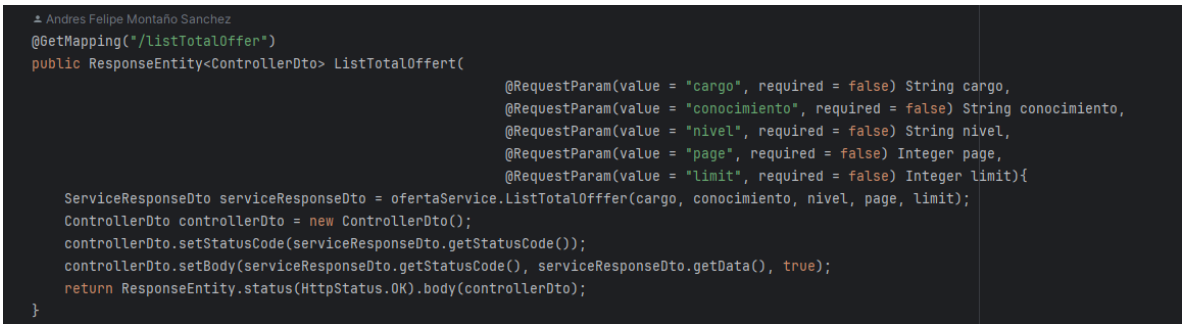


```
Andres Felipe Montaño Sanchez
@GetMapping("/listMyCourseOffer")
public ResponseEntity<ControllerDto> ListMyCourseOffert(@RequestParam("idPersona") Persona idPersona,
        @RequestParam(value = "cargo", required = false) String cargo,
        @RequestParam(value = "conocimiento", required = false) String conocimiento,
        @RequestParam(value = "nivel", required = false) String nivel,
        @RequestParam(value = "page", required = false) Integer page,
        @RequestParam(value = "limit", required = false) Integer limit){
    ServiceResponseDto serviceResponseDto = ofertaService.ListOffferMyCourse(idPersona,cargo, conocimiento, nivel, page, limit);
    ControllerDto controllerDto = new ControllerDto();
    controllerDto.setStatusCode(serviceResponseDto.getStatusCode());
    controllerDto.setBody(serviceResponseDto.getStatusCode(), serviceResponseDto.getData(), true);
    return ResponseEntity.status(HttpStatus.OK).body(controllerDto);
}
```

Nota. Autor (2024)

Figura 17

Controller para listar todos los cursos ofertados.



```
Andres Felipe Montaño Sanchez
@GetMapping("/listTotalOffer")
public ResponseEntity<ControllerDto> ListTotalOffert(
                                                    @RequestParam(value = "cargo", required = false) String cargo,
                                                    @RequestParam(value = "conocimiento", required = false) String conocimiento,
                                                    @RequestParam(value = "nivel", required = false) String nivel,
                                                    @RequestParam(value = "page", required = false) Integer page,
                                                    @RequestParam(value = "limit", required = false) Integer limit){
    ServiceResponseDto serviceResponseDto = ofertaService.ListTotalOfffer(cargo, conocimiento, nivel, page, limit);
    ControllerDto controllerDto = new ControllerDto();
    controllerDto.setStatusCode(serviceResponseDto.getStatusCode());
    controllerDto.setBody(serviceResponseDto.getStatusCode(), serviceResponseDto.getData(), true);
    return ResponseEntity.status(HttpStatus.OK).body(controllerDto);
}
```

Nota. Autor (2024)

Figura 18

Controller para listar el histórico de cursos del empleado.



```
Andres Felipe Montaño Sanchez
@GetMapping("/listOfferHistorical")
public ResponseEntity<ControllerDto> ListOffertHistorical(@RequestParam("idPersona") Persona idPersona,
                                                          @RequestParam(value = "cargo", required = false) String cargo,
                                                    @RequestParam(value = "conocimiento", required = false) String conocimiento,
                                                    @RequestParam(value = "nivel", required = false) String nivel,
                                                    @RequestParam(value = "page", required = false) Integer page,
                                                    @RequestParam(value = "limit", required = false) Integer limit){
    ServiceResponseDto serviceResponseDto = ofertaService.ListOffferHistorical(idPersona,cargo, conocimiento, nivel, page, limit);
    ControllerDto controllerDto = new ControllerDto();
    controllerDto.setStatusCode(serviceResponseDto.getStatusCode());
    controllerDto.setBody(serviceResponseDto.getStatusCode(), serviceResponseDto.getData(), true);
    return ResponseEntity.status(HttpStatus.OK).body(controllerDto);
}
```

Nota. Autor (2024)

Figura 19

Controller para listado de cursos ofertados por el cargo del empleado.



```
Andres Felipe Montaño Sanchez
@GetMapping("/listOfferCharge")
public ResponseEntity<ControllerDto> ListOffertCharge(@RequestParam("idPersona")String idPersona,
                                                      @RequestParam(value = "conocimiento", required = false) String conocimiento,
                                                      @RequestParam(value = "nivel", required = false) String nivel,
                                                      @RequestParam(value = "page", required = false) Integer page,
                                                      @RequestParam(value = "limit", required = false) Integer limit){
    ServiceResponseDto serviceResponseDto = ofertaService.ListOfertasByCharge(idPersona, conocimiento, nivel, page, limit);
    ControllerDto controllerDto = new ControllerDto();
    controllerDto.setStatusCode(serviceResponseDto.getStatusCode());
    controllerDto.setBody(serviceResponseDto.getStatusCode(), serviceResponseDto.getData(), true);
    return ResponseEntity.status(HttpStatus.OK).body(controllerDto);
}
```

Nota. Autor (2024)

A continuación, se presentarán los servicios (Service), los cuales pertenecen a la clase que contiene la lógica del negocio. Estos servicios son creados para cada uno de los controladores, ya que los controladores realizan llamadas a estos,

Figura 20

Service para listado de cursos ofertados según el cargo del empleado.



```
Andres Felipe Montaño Sanchez *
@Override
@Transactional(readOnly = true)
public ServiceResponseDto ListOfertasByCharge(String idPersona , String conocimiento, String nivel, Integer page, Integer limit){

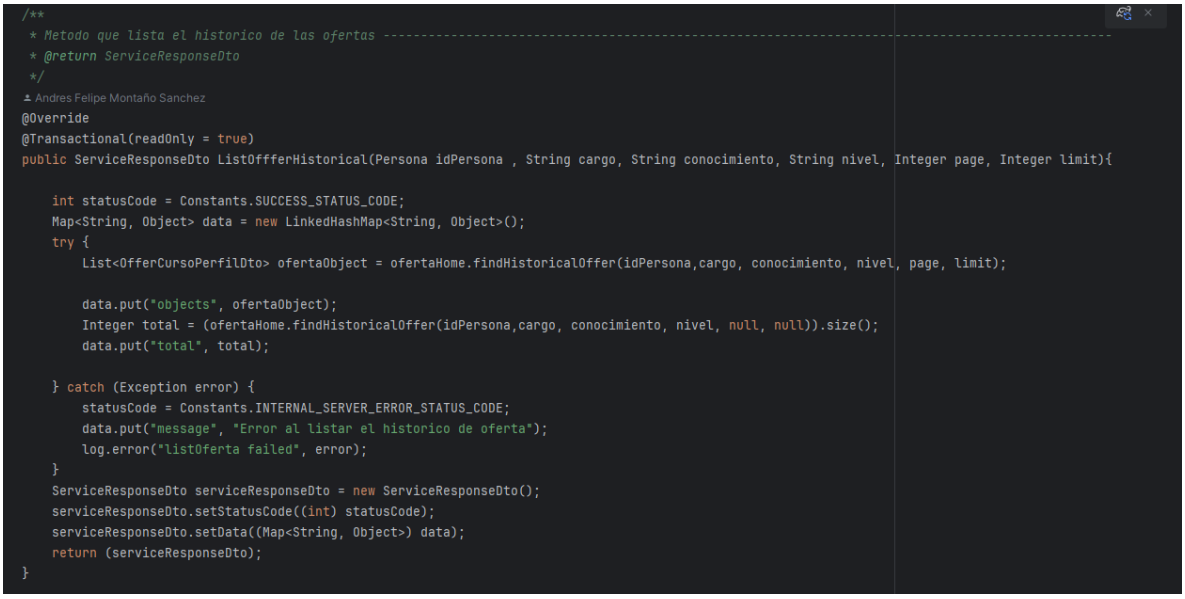
    int statusCode = Constants.SUCCESS_STATUS_CODE;
    Map<String, Object> data = new LinkedHashMap<String, Object>();
    try {
        String nombreIdPerfil = perfilRepository.findName(idPersona);
        List<OfferCursoPerfilDto> offerObject = ofertaHome.findByChargee(nombreIdPerfil, conocimiento, nivel, page, limit);
        Integer total = ofertaHome.findByChargee(nombreIdPerfil, conocimiento, nivel,null,null).size();
        data.put("objects", offerObject);
        data.put("total", total);

    } catch (Exception error) {
        statusCode = Constants.INTERNAL_SERVER_ERROR_STATUS_CODE;
        data.put("message", "Error al listar la oferta");
        log.error("listOferta failed", error);
    }
    ServiceResponseDto serviceResponseDto = new ServiceResponseDto();
    serviceResponseDto.setStatusCode((int) statusCode);
    serviceResponseDto.setData((Map<String, Object>) data);
    return (serviceResponseDto);
}
```

Nota. Autor (2024)

Figura 21

Service para listar el historico de cursos del empleado.



```
/**
 * Metodo que lista el historico de las ofertas ----------
 * @return ServiceResponseDto
 */
Andres Felipe Montaño Sanchez
@Override
@Transactional(readOnly = true)
public ServiceResponseDto ListOfrferHistorical(Persona idPersona , String cargo, String conocimiento, String nivel, Integer page, Integer limit){

    int statusCode = Constants.SUCCESS_STATUS_CODE;
    Map<String, Object> data = new LinkedHashMap<String, Object>();
    try {
        List<OfferCursoPerfilDto> ofertaObject = ofertaHome.findHistoricalOffer(idPersona,cargo, conocimiento, nivel, page, limit);

        data.put("objects", ofertaObject);
        Integer total = (ofertaHome.findHistoricalOffer(idPersona,cargo, conocimiento, nivel, null, null)).size();
        data.put("total", total);

    } catch (Exception error) {
        statusCode = Constants.INTERNAL_SERVER_ERROR_STATUS_CODE;
        data.put("message", "Error al listar el historico de oferta");
        log.error("listOferta failed", error);
    }
    ServiceResponseDto serviceResponseDto = new ServiceResponseDto();
    serviceResponseDto.setStatusCode((int) statusCode);
    serviceResponseDto.setData((Map<String, Object>) data);
    return (serviceResponseDto);
}
```

Nota. Autor (2024)

Figura 22

Service para listar el total de las ofertas



```
Andres Felipe Montaño Sanchez *
@Override
@Transactional(readOnly = true)
public ServiceResponseDto ListTotalOfffer(String cargo, String conocimiento, String nivel, Integer page, Integer limit){
    int statusCode = Constants.SUCCESS_STATUS_CODE;
    Map<String, Object> data = new LinkedHashMap<String, Object>();
    try {
        List<OfferCursoPerfilDto> ofertaObject = ofertaHome.findTotalOffer(cargo, conocimiento, nivel, page, limit);

        data.put("objects", ofertaObject);
        Integer total = (ofertaHome.findTotalOffer(cargo, conocimiento, nivel, null, null)).size();
        data.put("total", total);

    } catch (Exception error) {
        statusCode = Constants.INTERNAL_SERVER_ERROR_STATUS_CODE;
        data.put("message", "Error al listar el historico de oferta");
        log.error("listOferta failed", error);
    }
    ServiceResponseDto serviceResponseDto = new ServiceResponseDto();
    serviceResponseDto.setStatusCode((int) statusCode);
    serviceResponseDto.setData((Map<String, Object>) data);
    return (serviceResponseDto);
}
```

Nota. Autor (2024)

Figura 23

Para listado de cursos activos del empleado.



```
Andres Felipe Montaño Sanchez
@Override
@Transactional(readOnly = true)
public ServiceResponseDto ListOffferMyCourse(Persona idPersona , String cargo, String conocimiento, String nivel, Integer page, Integer limit){

    int statusCode = Constants.SUCCESS_STATUS_CODE;
    Map<String, Object> data = new LinkedHashMap<String, Object>();
    try {
        List<OfferCursoPerfilDto> ofertaObject = ofertaHome.findMyCourseOffer(idPersona,cargo, conocimiento, nivel, page, limit);

        data.put("objects", ofertaObject);
        Integer total = (ofertaHome.findMyCourseOffer(idPersona,cargo, conocimiento, nivel, null, null)).size();
        data.put("total", total);

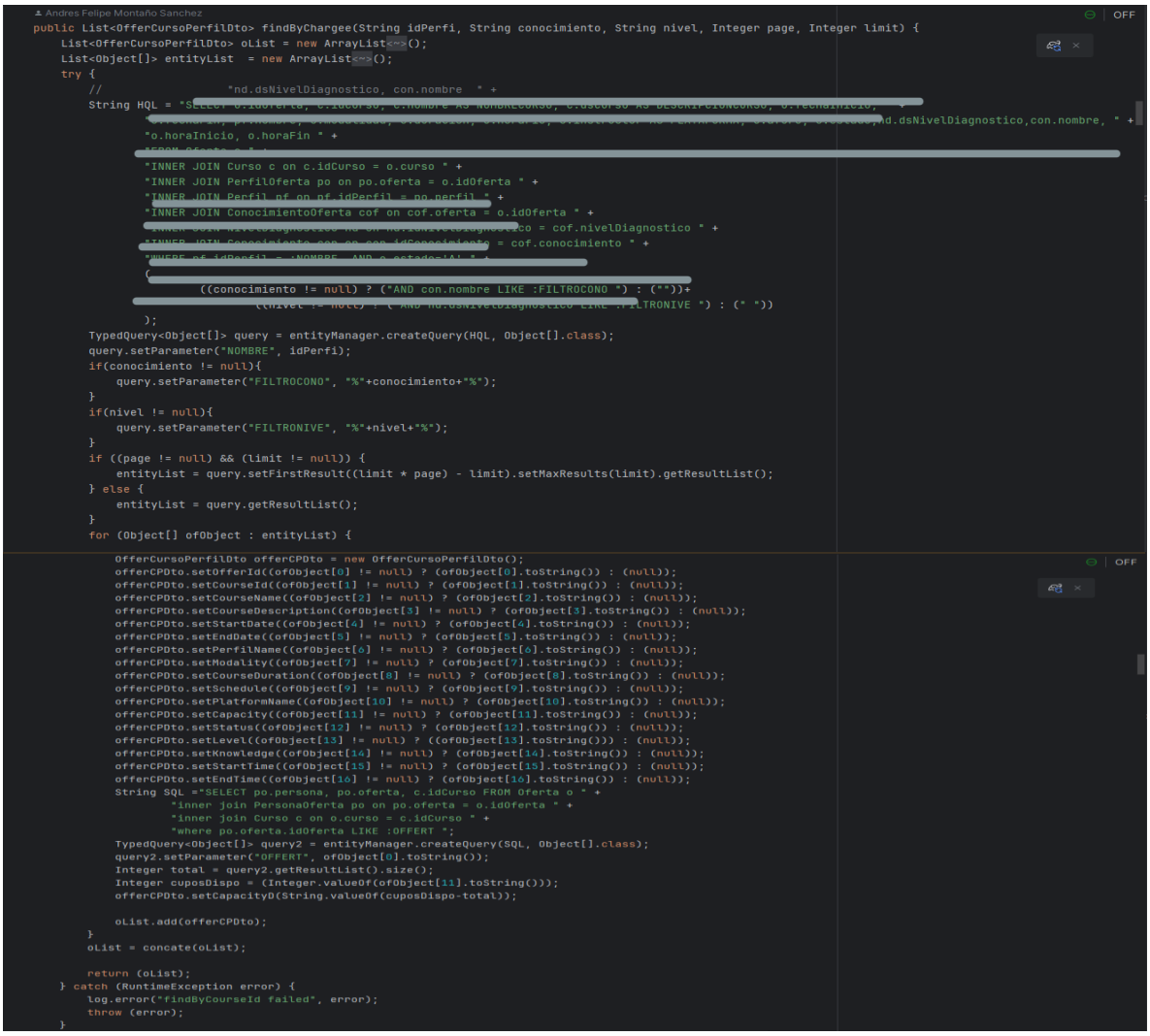
    } catch (Exception error) {
        statusCode = Constants.INTERNAL_SERVER_ERROR_STATUS_CODE;
        data.put("message", "Error al listar el historico de oferta");
        log.error("listOferta failed", error);
    }
    ServiceResponseDto serviceResponseDto = new ServiceResponseDto();
    serviceResponseDto.setStatusCode((int) statusCode);
    serviceResponseDto.setData((Map<String, Object>) data);
    return (serviceResponseDto);
}
```

Nota. Autor (2024)

A continuación, se presenta el código de las clases Repository, donde se codificaron las consultas utilizando lenguaje HQL a través de Hibernate. Estas consultas incluyen varias utilidades, como la funcionalidad de paginado para el Frontend. Además, se implementaron condiciones dentro de las consultas para agregar o eliminar partes del código, con el objetivo de minimizar la cantidad de código y evitar la creación de múltiples consultas:

Figura 24

Código consultas



```
Andres Felipe Montaño Sanchez
public List<OfferCursoPerfilDto> findByChargee(String idPerfi, String conocimiento, String nivel, Integer page, Integer limit) {
    List<OfferCursoPerfilDto> oList = new ArrayList<>();
    List<Object[]> entityList  = new ArrayList<>();
    try {
        //                "nd.dsNivelDiagnostico, con.nombre  " +
        String HQL = "SELECT o.idOferta, c.idCurso, c.NOMBRE AS NOMBRECURSO, c.DESCRIPCION AS DESCRIPCIONCURSO, o.fechaInicio,  " +
                "o.horaInicio, o.horaFin " +
                "FROM Oferta o " +
                "INNER JOIN Curso c on c.idCurso = o.curso " +
                "INNER JOIN PerfilOferta po on po.oferta = o.idOferta " +
                "INNER JOIN Perfil pf on pf.idPerfil = po.perfil " +
                "INNER JOIN ConocimientoOferta cof on cof.oferta = o.idOferta " +
                "INNER JOIN NivelDiagnostico nd on nd.nivelDiagnostico = cof.nivelDiagnostico " +
                "INNER JOIN Conocimiento con on con.idConocimiento = cof.conocimiento " +
                "WHERE pf.idPerfil = :NOMBRE  AND o.estado='A' " +
                (
                        ((conocimiento != null) ? ("AND con.nombre LIKE :FILTROCONO ") : (""))+
                                ((nivel != null) ? ("AND nd.dsNivelDiagnostico LIKE :FILTRONIVE ") : (" "))
                );
        TypedQuery<Object[]> query = entityManager.createQuery(HQL, Object[].class);
        query.setParameter("NOMBRE", idPerfi);
        if(conocimiento != null){
            query.setParameter("FILTROCONO", "%"+conocimiento+"%");
        }
        if(nivel != null){
            query.setParameter("FILTRONIVE", "%"+nivel+"%");
        }
        if ((page != null) && (limit != null)) {
            entityList = query.setFirstResult((limit * page) - limit).setMaxResults(limit).getResultList();
        } else {
            entityList = query.getResultList();
        }
        for (Object[] ofObject : entityList) {

            OfferCursoPerfilDto offerCPDto = new OfferCursoPerfilDto();
            offerCPDto.setOfferId((ofObject[0] != null) ? (ofObject[0].toString()) : (null));
            offerCPDto.setCourseId((ofObject[1] != null) ? (ofObject[1].toString()) : (null));
            offerCPDto.setCourseName((ofObject[2] != null) ? (ofObject[2].toString()) : (null));
            offerCPDto.setCourseDescription((ofObject[3] != null) ? (ofObject[3].toString()) : (null));
            offerCPDto.setStartDate((ofObject[4] != null) ? (ofObject[4].toString()) : (null));
            offerCPDto.setEndDate((ofObject[5] != null) ? (ofObject[5].toString()) : (null));
            offerCPDto.setPerfilName((ofObject[6] != null) ? (ofObject[6].toString()) : (null));
            offerCPDto.setModality((ofObject[7] != null) ? (ofObject[7].toString()) : (null));
            offerCPDto.setCourseDuration((ofObject[8] != null) ? (ofObject[8].toString()) : (null));
            offerCPDto.setSchedule((ofObject[9] != null) ? (ofObject[9].toString()) : (null));
            offerCPDto.setPlatformName((ofObject[10] != null) ? (ofObject[10].toString()) : (null));
            offerCPDto.setCapacity((ofObject[11] != null) ? (ofObject[11].toString()) : (null));
            offerCPDto.setStatus((ofObject[12] != null) ? (ofObject[12].toString()) : (null));
            offerCPDto.setLevel((ofObject[13] != null) ? ((ofObject[13].toString())) : (null));
            offerCPDto.setKnowledge((ofObject[14] != null) ? (ofObject[14].toString()) : (null));
            offerCPDto.setStartTime((ofObject[15] != null) ? (ofObject[15].toString()) : (null));
            offerCPDto.setEndTime((ofObject[16] != null) ? (ofObject[16].toString()) : (null));
            String SQL ="SELECT po.persona, po.oferta, c.idCurso FROM Oferta o " +
                    "inner join PersonaOferta po on po.oferta = o.idOferta " +
                    "inner join Curso c on o.curso = c.idCurso " +
                    "where po.oferta.idOferta LIKE :OFFERT ";
            TypedQuery<Object[]> query2 = entityManager.createQuery(SQL, Object[].class);
            query2.setParameter("OFFERT", ofObject[0].toString());
            Integer total = query2.getResultList().size();
            Integer cuposDispo = (Integer.valueOf(ofObject[11].toString()));
            offerCPDto.setCapacityD(String.valueOf(cuposDispo-total));

            oList.add(offerCPDto);
        }
        oList = concate(oList);

        return (oList);
    } catch (RuntimeException error) {
        log.error("findByCourseId failed", error);
        throw (error);
    }
```

Nota. Autor (2024)

Figura 25

Código DTO



```
Andres Felipe Montaño Sanchez
public List<OfferCursoPerfilDto> concate(List<OfferCursoPerfilDto> lista){
    List<OfferCursoPerfilDto> listaCon = new ArrayList<>();
    if(lista.isEmpty()){
        return lista;
    }
    List<String> conocimientos = new ArrayList<>();
    List<String> perfiles = new ArrayList<>();
    OfferCursoPerfilDto offerObj = lista.get(0);
    int i =0;
    for (OfferCursoPerfilDto iter: lista) {
        for (int j = 0; j < listaCon.size(); j++){
            if(listaCon.get(j).getOfferId().equals(offerObj.getOfferId())
                    && listaCon.get(j).getCourseId().equals(offerObj.getCourseId())
                    && listaCon.get(j).getLevel().equals(offerObj.getLevel())){
                i=1;
                if(listaCon.get(j).getKnowledge() != null){
                conocimientos = new ArrayList<>(Arrays.asList(listaCon.get(j).getKnowledge().split(",")));}
                if(listaCon.get(j).getPerfilName() != null){
                perfiles = new ArrayList<>(Arrays.asList(listaCon.get(j).getPerfilName().split(",")));}
                if (!conocimientos.contains(offerObj.getKnowledge()) && offerObj.getKnowledge() != null){
                    conocimientos.add(offerObj.getKnowledge());
                    listaCon.get(j).setKnowledge(conocimientos.stream().map(Object::toString).collect(Collectors.joining(",")));
                }
                if (!perfiles.contains(offerObj.getPerfilName()) && offerObj.getPerfilName() != null){
                    perfiles.add(offerObj.getPerfilName());
                    listaCon.get(j).setPerfilName(perfiles.stream().map(Object::toString).collect(Collectors.joining(",")));
                }
                conocimientos.clear();
                perfiles.clear();
                offerObj = iter;
                if (!conocimientos.contains(iter.getKnowledge()) && iter.getKnowledge() != null){
                    conocimientos.add(iter.getKnowledge());
                    offerObj.setKnowledge(conocimientos.stream().map(Object::toString).collect(Collectors.joining(",")));
                }
                if (!perfiles.contains(iter.getPerfilName()) && iter.getPerfilName() != null){
                    perfiles.add(iter.getPerfilName());
                    offerObj.setPerfilName(perfiles.stream().map(Object::toString).collect(Collectors.joining(",")));
                }
            }
        }
        if(i==0){

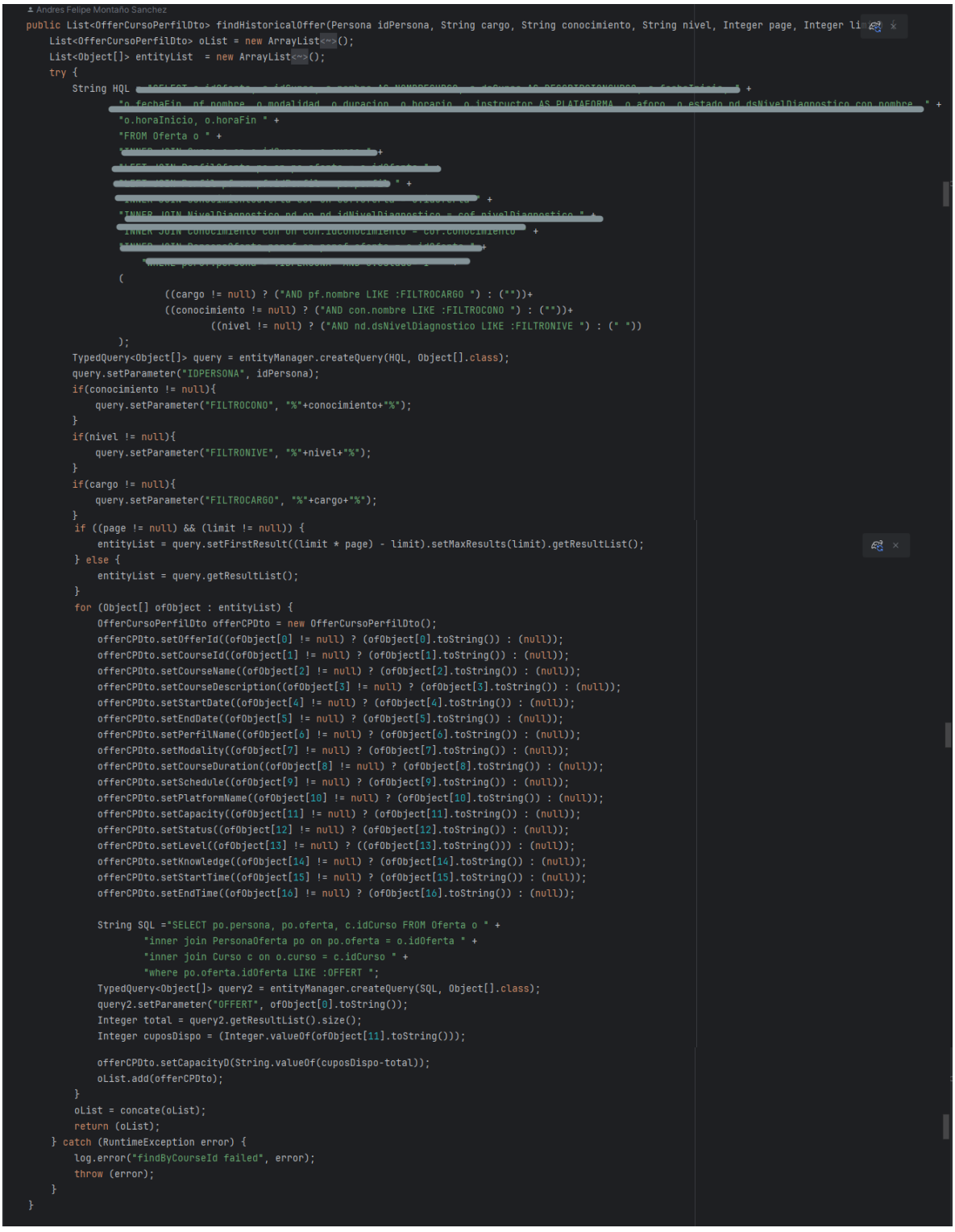
        if(!iter.getOfferId().equals(offerObj.getOfferId()) || !iter.getCourseId().equals(offerObj.getCourseId()) || !iter.getLevel().equals(offerObj.getLevel()
            listaCon.add(offerObj);
            offerObj = iter;
            conocimientos.clear();
            perfiles.clear();
        }
        if (!conocimientos.contains(iter.getKnowledge()) && iter.getKnowledge() != null){
            conocimientos.add(iter.getKnowledge());
            offerObj.setKnowledge(conocimientos.stream().map(Object::toString).collect(Collectors.joining(",")));
        }
        if (!perfiles.contains(iter.getPerfilName()) && iter.getPerfilName() != null){
            perfiles.add(iter.getPerfilName());
            offerObj.setPerfilName(perfiles.stream().map(Object::toString).collect(Collectors.joining(",")));
        }}
        i=0;
    }
    if(!listaCon.contains(offerObj)){
        listaCon.add(offerObj);
    }

    return listaCon;
}
```

Nota. Autor (2024)

Figura 26

Código de cursos ofertados



```
Andres Felipe Montaño Sanchez
public List<OfferCursoPerfilDto> findHistoricalOffer(Persona idPersona, String cargo, String conocimiento, String nivel, Integer page, Integer li
    List<OfferCursoPerfilDto> oList = new ArrayList<>();
    List<Object[]> entityList  = new ArrayList<>();
    try {
        String HQL = "SELECT [redacted] +
                "o.fechaFin, pf.nombre, o.modalidad, o.duracion, o.horario, o.instructor AS PLATAFORMA, o.aforo, o.estado, nd.dsNivelDiagnostico con nombre " +
                "o.horaInicio, o.horaFin " +
                "FROM Oferta o " +
                "[redacted] +
                "[redacted] " +
                "[redacted] +
                "INNER JOIN NivelDiagnostico nd on nd.idNivelDiagnostico = conf.nivelDiagnostico " +
                "INNER JOIN Conocimiento con on con.idConocimiento = conf.conocimiento " +
                "[redacted] +
                "[redacted] +
                (
                        ((cargo != null) ? ("AND pf.nombre LIKE :FILTROCARGO ") : (""))+
                        ((conocimiento != null) ? ("AND con.nombre LIKE :FILTROCONO ") : (""))+
                                ((nivel != null) ? ("AND nd.dsNivelDiagnostico LIKE :FILTRONIVE ") : (" "))
                );
        TypedQuery<Object[]> query = entityManager.createQuery(HQL, Object[].class);
        query.setParameter("IDPERSONA", idPersona);
        if(conocimiento != null){
            query.setParameter("FILTROCONO", "%"+conocimiento+"%");
        }
        if(nivel != null){
            query.setParameter("FILTRONIVE", "%"+nivel+"%");
        }
        if(cargo != null){
            query.setParameter("FILTROCARGO", "%"+cargo+"%");
        }
        if ((page != null) && (limit != null)) {
            entityList = query.setFirstResult((limit * page) - limit).setMaxResults(limit).getResultList();
        } else {
            entityList = query.getResultList();
        }
        for (Object[] ofObject : entityList) {
            OfferCursoPerfilDto offerCPDto = new OfferCursoPerfilDto();
            offerCPDto.setOfferId((ofObject[0] != null) ? (ofObject[0].toString()) : (null));
            offerCPDto.setCourseId((ofObject[1] != null) ? (ofObject[1].toString()) : (null));
            offerCPDto.setCourseName((ofObject[2] != null) ? (ofObject[2].toString()) : (null));
            offerCPDto.setCourseDescription((ofObject[3] != null) ? (ofObject[3].toString()) : (null));
            offerCPDto.setStartDate((ofObject[4] != null) ? (ofObject[4].toString()) : (null));
            offerCPDto.setEndDate((ofObject[5] != null) ? (ofObject[5].toString()) : (null));
            offerCPDto.setPerfilName((ofObject[6] != null) ? (ofObject[6].toString()) : (null));
            offerCPDto.setModality((ofObject[7] != null) ? (ofObject[7].toString()) : (null));
            offerCPDto.setCourseDuration((ofObject[8] != null) ? (ofObject[8].toString()) : (null));
            offerCPDto.setSchedule((ofObject[9] != null) ? (ofObject[9].toString()) : (null));
            offerCPDto.setPlatformName((ofObject[10] != null) ? (ofObject[10].toString()) : (null));
            offerCPDto.setCapacity((ofObject[11] != null) ? (ofObject[11].toString()) : (null));
            offerCPDto.setStatus((ofObject[12] != null) ? (ofObject[12].toString()) : (null));
            offerCPDto.setLevel((ofObject[13] != null) ? ((ofObject[13].toString())) : (null));
            offerCPDto.setKnowledge((ofObject[14] != null) ? (ofObject[14].toString()) : (null));
            offerCPDto.setStartTime((ofObject[15] != null) ? (ofObject[15].toString()) : (null));
            offerCPDto.setEndTime((ofObject[16] != null) ? (ofObject[16].toString()) : (null));

            String SQL ="SELECT po.persona, po.oferta, c.idCurso FROM Oferta o " +
                    "inner join PersonaOferta po on po.oferta = o.idOferta " +
                    "inner join Curso c on o.curso = c.idCurso " +
                    "where po.oferta.idOferta LIKE :OFFERT ";
            TypedQuery<Object[]> query2 = entityManager.createQuery(SQL, Object[].class);
            query2.setParameter("OFFERT", ofObject[0].toString());
            Integer total = query2.getResultList().size();
            Integer cuposDispo = (Integer.valueOf(ofObject[11].toString()));

            offerCPDto.setCapacityD(String.valueOf(cuposDispo-total));
            oList.add(offerCPDto);
        }
        oList = concate(oList);
        return (oList);
    } catch (RuntimeException error) {
        log.error("findByCourseId failed", error);
        throw (error);
    }
}
```

Nota. Autor (2024)

3.1.8 Codificación funcionalidad Encuestas (CRUD).

Para el desarrollo de la funcionalidad de Encuestas en Valko, que incluye operaciones CRUD (Crear, Leer, Actualizar, Eliminarse se siguió un enfoque metodológico y detallado.

Se diseñó e implementó la funcionalidad para la creación de nuevas encuestas en Valko:

Backend:

Figura 27

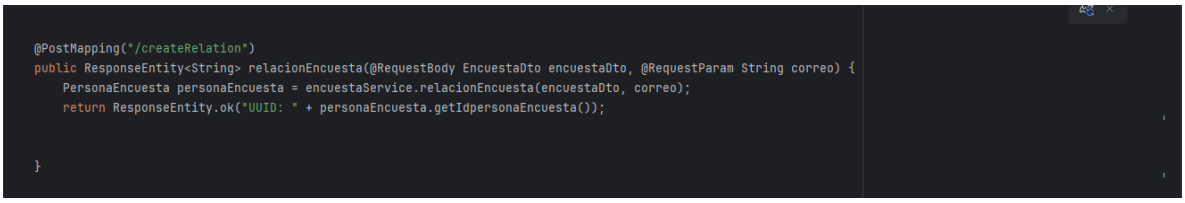
EndPoint encuenstas.



Nota. Autor (2024)

Figura 28

EndPoint relaciones



Nota. Autor (2024)

- Se desarrolló una ruta en el servidor para recibir y procesar los datos del formulario.
- Se validaron los datos de entrada para asegurar la integridad y consistencia de la información.

Se implementó la lógica necesaria para persistir las nuevas encuestas en la base de datos, garantizando la correcta relación con otras entidades.

Leer Encuesta (Read).

La funcionalidad para leer y visualizar encuestas existentes ha sido implementada:

Backend:

Figura 29

EndPoint leer encuesta



Nota. Autor (2024)

Figura 30



```
@GetMapping("/byEstado")
public ResponseEntity<List<PersonaEncuesta>> getEstado(@RequestParam char estado) {
    List<PersonaEncuesta> filterEmail = encuestaService.obtenerAlertasPorEstados(estado);
    return ResponseEntity.ok(filterEmail);
}

/*
@PostMapping("/procesar-alertas")
public ResponseEntity<String> procesarAlertas() {encuestaService.procesarAlertas();
    return ResponseEntity.ok("Alertas procesadas correctamente"); }
}

 */



@GetMapping("/EncuestasPersona")
public ResponseEntity<List<Encuesta>> obtenerEncuestasPorIdPersona(@RequestParam(name = "idPersona")String idPersona){
    List<Encuesta> encuestas = encuestaService.obtenerEncuestasByIdPersona(idPersona);
    return ResponseEntity.ok(encuestas);
}
```

Nota. Autor (2024)

- Se establecieron rutas para obtener la lista completa de encuestas y detalles específicos de cada una.
- Se implementó la lógica para manejar consultas y filtrados según los requisitos planteados.

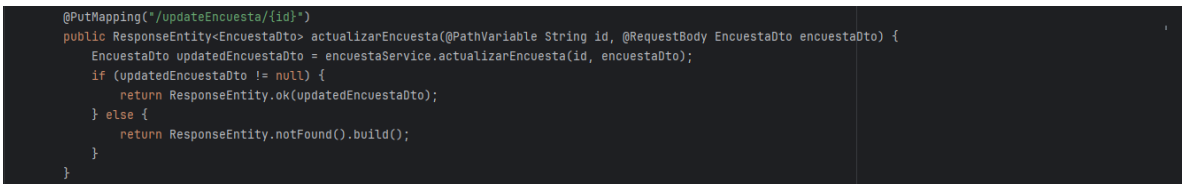
Actualizar Encuesta (Update).

Se ha diseñado y desarrollado la funcionalidad para actualizar encuestas existentes:

Backend:

Figura 31

EndPoint actualizar encuestas



```
@PutMapping("/updateEncuesta/{id}")
public ResponseEntity<EncuestaDto> actualizarEncuesta(@PathVariable String id, @RequestBody EncuestaDto encuestaDto) {
    EncuestaDto updatedEncuestaDto = encuestaService.actualizarEncuesta(id, encuestaDto);
    if (updatedEncuestaDto != null) {
        return ResponseEntity.ok(updatedEncuestaDto);
    } else {
        return ResponseEntity.notFound().build();
    }
}
```

Nota. Autor (2024)

- Se implementó una ruta para recibir y procesar los datos actualizados de la encuesta.
- Se validaron y aplicaron los cambios a la encuesta correspondiente en la base de datos, asegurando la consistencia y cumplimiento de los criterios establecidos.

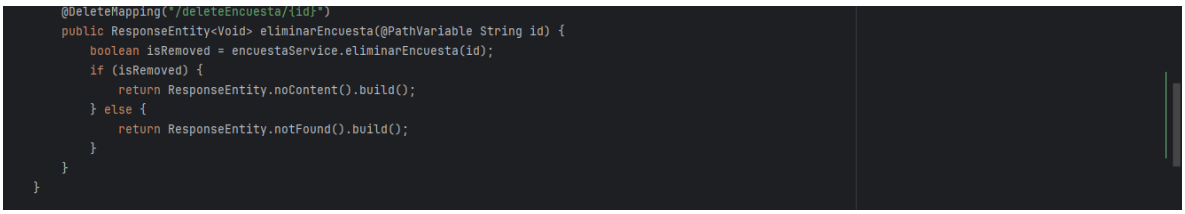
Eliminar Encuesta (Delete).

La funcionalidad para eliminar encuestas ha sido diseñada e implementada:

Backend:

Figura 32

EndPoint eliminar encuestas



Nota. Autor (2024)

Se desarrolló una ruta para manejar las solicitudes de eliminación de encuestas, se implementó la lógica necesaria para eliminar de manera segura la encuesta y todas las preguntas asociadas, manteniendo la integridad referencial de la base de datos.

Consideraciones adicionales:

Seguridad: Se han implementado medidas de seguridad robustas para proteger los datos de las encuestas y prevenir accesos no autorizados.

Documentación: Se ha mantenido una documentación detallada del proceso de codificación y de los requisitos funcionales y no funcionales implementados.

Pruebas: Se realizaron pruebas exhaustivas de la funcionalidad CRUD para asegurar su correcto funcionamiento antes de su implementación en producción.

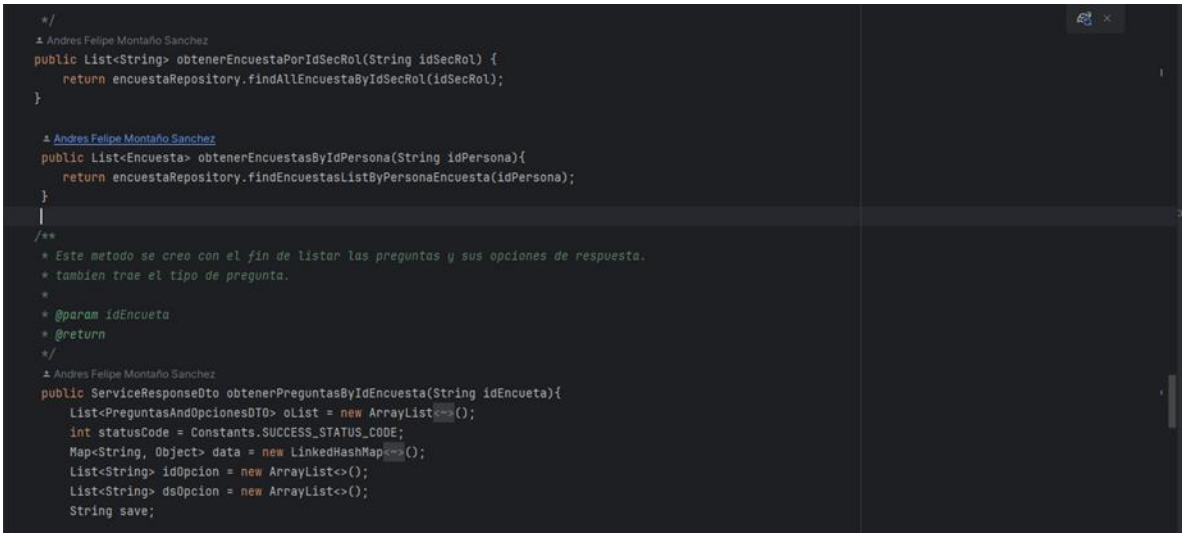
Con estas acciones completadas, se asegura que la funcionalidad de encuestas en Valko cumple con los criterios establecidos por el Product Owner y está lista para su implementación y uso por parte de los usuarios finales.

De tal forma la implementación de la funcionalidad de encuestas en Valko, se diseñó un sistema completo que permite a los usuarios crear encuestas personalizadas, agregar y gestionar preguntas según diversos tipos y configuraciones. Se facilitó la modificación de preguntas existentes para adaptarse a las necesidades cambiantes de los usuarios, asegurando así la flexibilidad y adaptabilidad del sistema. Con estas capacidades implementadas, se logra una gestión integral de encuestas que cumple con los criterios establecidos por el Product Owner, facilitando una experiencia de usuario intuitiva y eficiente dentro de la plataforma Valko.

A continuación, se presenta el código de las clases Service y Repository, donde se codificaron las consultas utilizando lenguaje HQL a través de Hibernate y la lógica del negocio.

Figura 33

Servicio



```
 */
 Andres Felipe Montaño Sanchez
public List<String> obtenerEncuestaPorIdSecRol(String idSecRol) {
    return encuestaRepository.findAllEncuestaByIdSecRol(idSecRol);
}

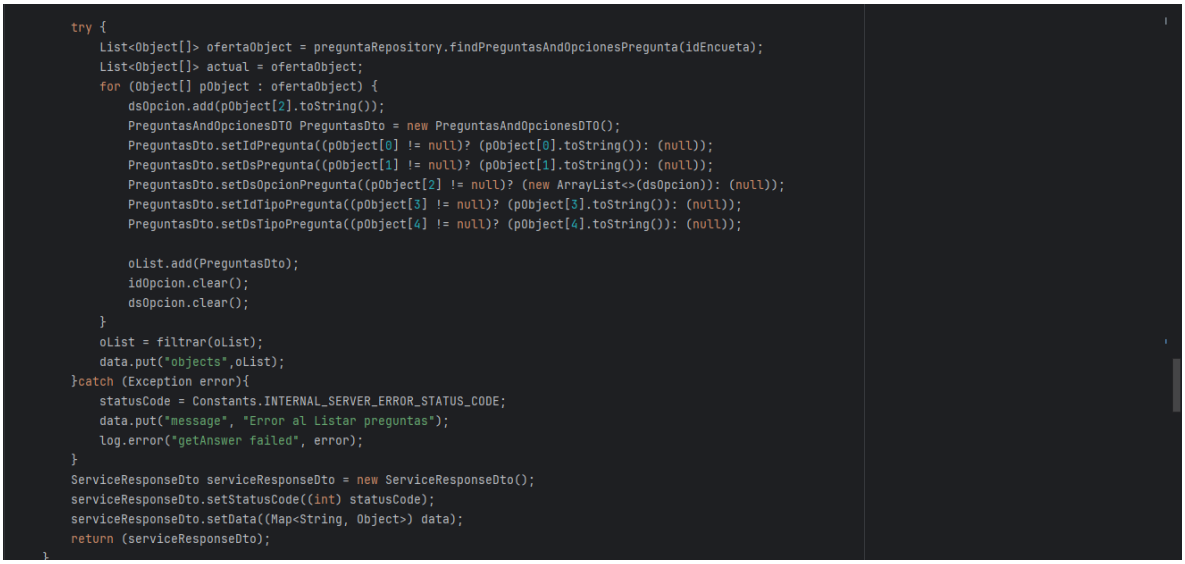
 Andres Felipe Montaño Sanchez
public List<Encuesta> obtenerEncuestasByIdPersona(String idPersona){
    return encuestaRepository.findEncuestasListByPersonaEncuesta(idPersona);
}

/**
 * Este metodo se creo con el fin de listar las preguntas y sus opciones de respuesta.
 * tambien trae el tipo de pregunta.
 *
 * @param idEncueta
 * @return
 */
 Andres Felipe Montaño Sanchez
public ServiceResponseDto obtenerPreguntasByIdEncuesta(String idEncueta){
    List<PreguntasAndOpcionesDTO> oList = new ArrayList<>();
    int statusCode = Constants.SUCCESS_STATUS_CODE;
    Map<String, Object> data = new LinkedHashMap<>();
    List<String> idOpcion = new ArrayList<>();
    List<String> dsOpcion = new ArrayList<>();
    String save;
```

Nota. Autor (2024)

Figura 34

Mapeo de errores



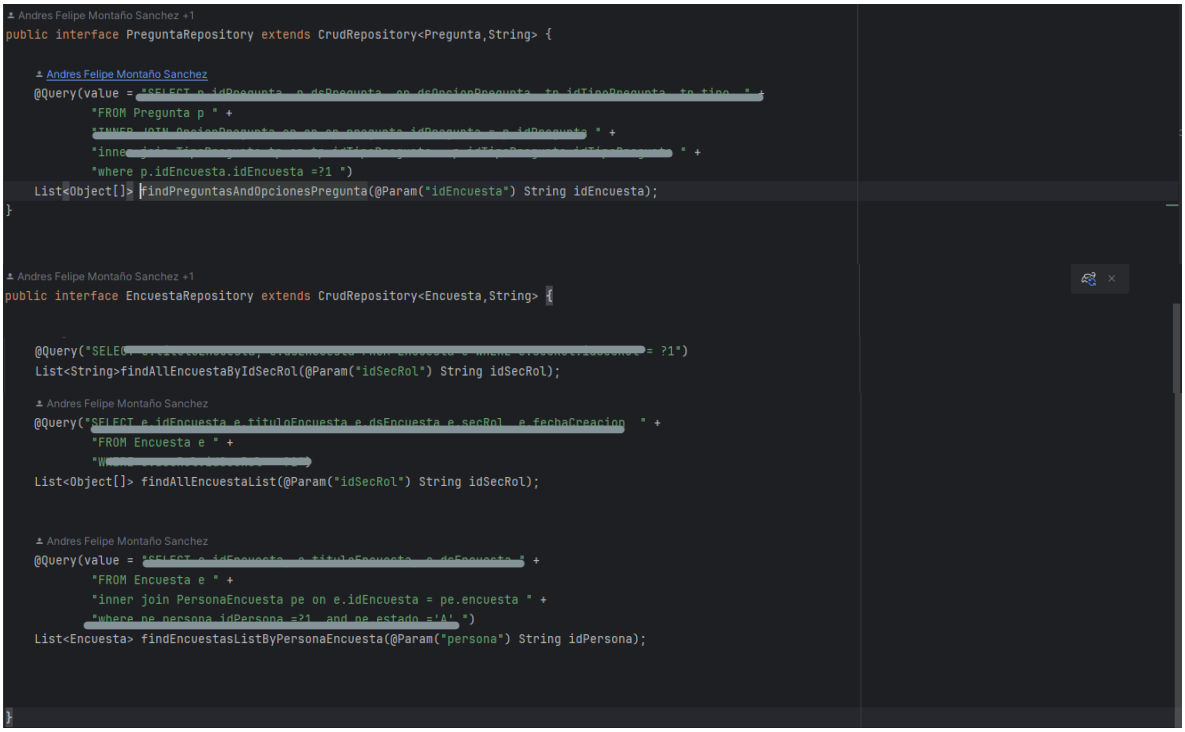
```
    try {
        List<Object[]> ofertaObject = preguntaRepository.findPreguntasAndOpcionesPregunta(idEncueta);
        List<Object[]> actual = ofertaObject;
        for (Object[] pObject : ofertaObject) {
            dsOpcion.add(pObject[2].toString());
            PreguntasAndOpcionesDTO PreguntasDto = new PreguntasAndOpcionesDTO();
            PreguntasDto.setIdPregunta((pObject[0] != null)? (pObject[0].toString()): (null));
            PreguntasDto.setDsPregunta((pObject[1] != null)? (pObject[1].toString()): (null));
            PreguntasDto.setDsOpcionPregunta((pObject[2] != null)? (new ArrayList<>(dsOpcion)): (null));
            PreguntasDto.setIdTipoPregunta((pObject[3] != null)? (pObject[3].toString()): (null));
            PreguntasDto.setDsTipoPregunta((pObject[4] != null)? (pObject[4].toString()): (null));

            oList.add(PreguntasDto);
            idOpcion.clear();
            dsOpcion.clear();
        }
        oList = filtrar(oList);
        data.put("objects",oList);
    }catch (Exception error){
        statusCode = Constants.INTERNAL_SERVER_ERROR_STATUS_CODE;
        data.put("message", "Error al Listar preguntas");
        log.error("getAnswer failed", error);
    }
    ServiceResponseDto serviceResponseDto = new ServiceResponseDto();
    serviceResponseDto.setStatusCode((int) statusCode);
    serviceResponseDto.setData((Map<String, Object>) data);
    return (serviceResponseDto);
}
```

Nota. Autor (2024)

Figura 35

Consultas a la base de datos luego del servicio



```
Andres Felipe Montaño Sanchez +1
public interface PreguntaRepository extends CrudRepository<Pregunta,String> {

    Andres Felipe Montaño Sanchez
    @Query(value = "SELECT p.idPregunta, p.dsPregunta, op.dsOpcionPregunta, tp.idTipoPregunta, tp.tipo " +
            "FROM Pregunta p " +
            "INNER JOIN OpcionPregunta op on op.pregunta.idPregunta = p.idPregunta " +
            "inner join TipoPregunta tp on p.idTipoPregunta = tp.idTipoPregunta " +
            "where p.idEncuesta.idEncuesta =?1 ")
    List<Object[]> findPreguntasAndOpcionesPregunta(@Param("idEncuesta") String idEncuesta);
}


Andres Felipe Montaño Sanchez +1
public interface EncuestaRepository extends CrudRepository<Encuesta,String> {


    @Query("SELECT e.idEncuesta, e.tituloEncuesta, e.dsEncuesta, e.secRol, e.fechaCreacion FROM Encuesta e WHERE e.idEncuesta = ?1")
    List<String>findAllEncuestaByIdSecRol(@Param("idSecRol") String idSecRol);

    Andres Felipe Montaño Sanchez
    @Query("SELECT e.idEncuesta, e.tituloEncuesta, e.dsEncuesta, e.secRol, e.fechaCreacion " +
            "FROM Encuesta e " +
            "WHERE e.secRol = :idSecRol")
    List<Object[]> findAllEncuestaList(@Param("idSecRol") String idSecRol);


    Andres Felipe Montaño Sanchez
    @Query(value = "SELECT e.idEncuesta, e.tituloEncuesta, e.dsEncuesta " +
            "FROM Encuesta e " +
            "inner join PersonaEncuesta pe on e.idEncuesta = pe.encuesta " +
            "where pe.persona.idPersona =?1 and pe.estado ='A' ")
    List<Encuesta> findEncuestasListByPersonaEncuesta(@Param("persona") String idPersona);


}
```

Nota. Autor (2024)

En las imágenes anteriores se evidencia el uso de métodos diferentes a los empleados en los apartados de formación. En los apartados de formación, se siguió la metodología previamente establecida para su creación. Sin embargo, dado que el módulo de encuestas es nuevo, se decidió utilizar los métodos de CRUD repository, los cuales permiten ahorrar un número significativo de líneas de código. Cabe aclarar que no se incluyen todas las capturas del código para evitar que el documento se vuelva excesivamente extenso.

3.1.9 Desarrollar y ejecutar pruebas tanto unitarias como de aceptacion.

El proceso comenzó con el desarrollo exhaustivo de pruebas unitarias para verificar cada componente de la funcionalidad de encuestas en Valko:

Framework y Herramientas Utilizadas:

Se implementó JUnit junto con Mockito para escribir y ejecutar pruebas unitarias en el entorno de desarrollo de Spring Boot.

Mockito se utilizó para simular y testear interacciones entre componentes, asegurando pruebas independientes y aisladas.

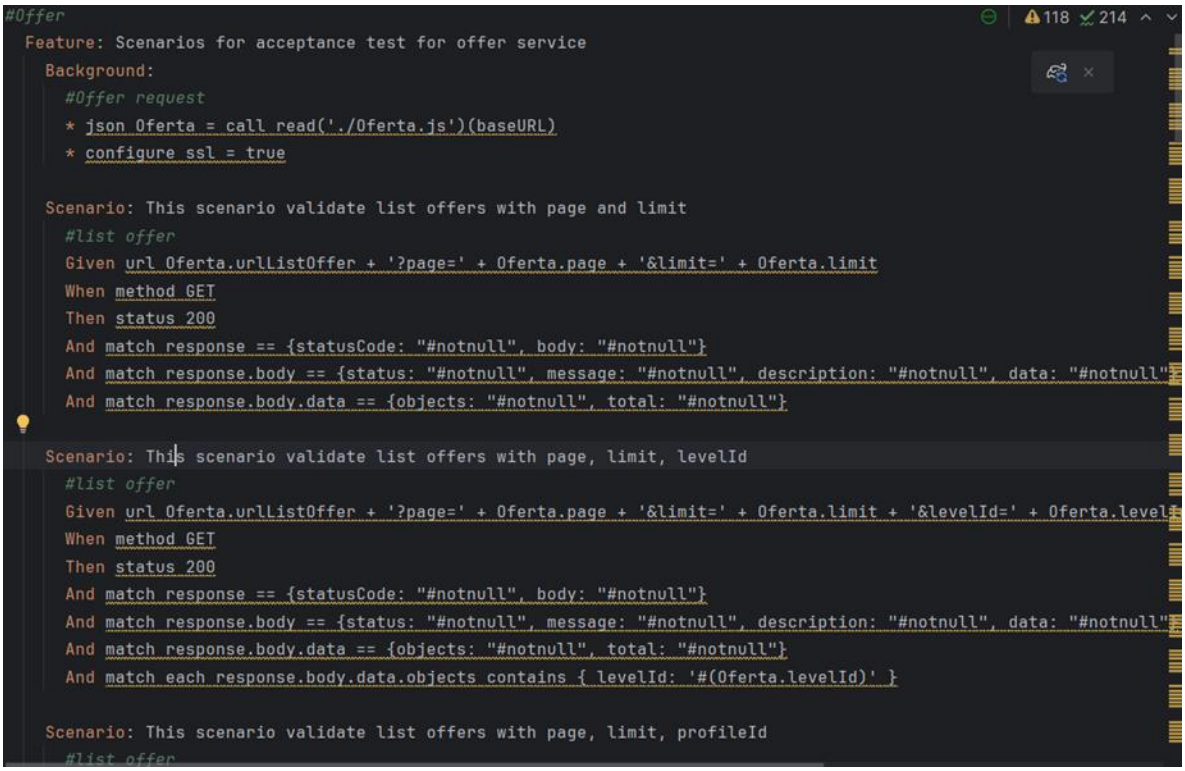
Cada método y función fue probado rigurosamente para garantizar su correcto funcionamiento conforme a los requisitos funcionales y no funcionales definidos.

Ejecución de Pruebas

En las siguiente imagen se muestra la manera en la que se estructuran las pruebas de aceptación, dado a que son bastante extensas solo se pondrá una imagen inicial.

Figura 36

Pruebas de Aceptacion



Nota. Autor (2024)

En las siguiente imagen se muestra la manera en la que se estructuran las pruebas de unitarias, dado a que son bastante extensas solo se pondrá una imagen inicial.

Figura 37

Pruebas de software mockito

```
Andres Felipe Montaño Sanchez +1
@Test
public void testObtenerEncuestaPorIdSecRol() {
    String idSecRol = "testSecRolId";
    List<String> encuestas = new ArrayList<>();

    when(encuestaRepository.findAllEncuestaByIdSecRol(idSecRol)).thenReturn(encuestas);

    List<String> resultado = encuestaService.obtenerEncuestaPorIdSecRol(idSecRol);


    assertNotNull(resultado);

}
```

Nota. Autor (2024)

Figura 38

Pruebas

```
@DisplayName("Filtrar - No hay duplicados")
@Test
public void testFiltrar_NoDuplicados() {
    // Arrange
    List<PreguntasAndOpcionesDTO> lista = new ArrayList<>();
    PreguntasAndOpcionesDTO pregunta1 = new PreguntasAndOpcionesDTO();
    pregunta1.setIdPregunta("1");
    pregunta1.setDsOpcionPregunta(Arrays.asList("Opción 1 para pregunta 1"));
    PreguntasAndOpcionesDTO pregunta2 = new PreguntasAndOpcionesDTO();
    pregunta2.setIdPregunta("2");
    pregunta2.setDsOpcionPregunta(Arrays.asList("Opción 1 para pregunta 2"));
    lista.add(pregunta1);
    lista.add(pregunta2);

    // Act
    List<PreguntasAndOpcionesDTO> listaFiltrada = EncuestaService.filtrar(lista);

    // Assert
    assertEquals(lista, listaFiltrada);
}
```

Nota. Autor (2024)

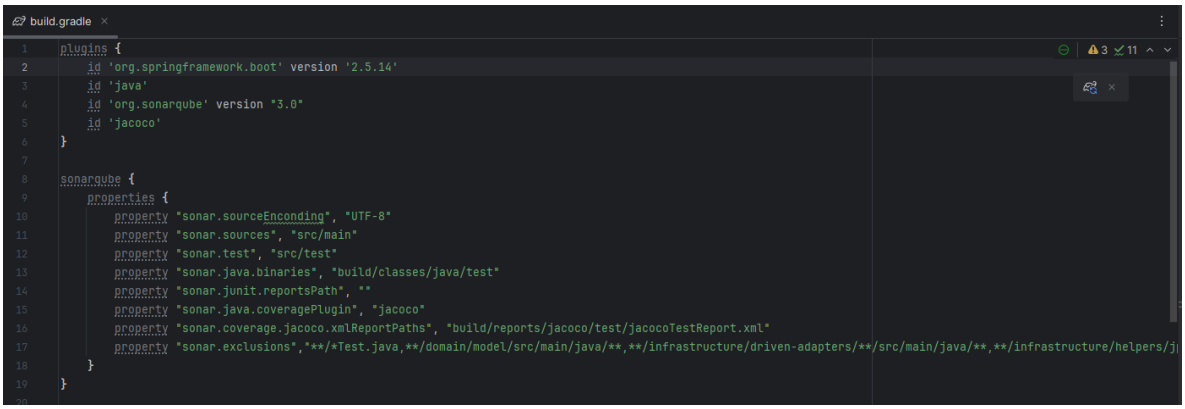
Después de validar las pruebas unitarias, se procedió con las pruebas de aceptación para verificar el comportamiento integral del sistema:

Herramientas y Enfoque:

Se empleó Cucumber junto con Spring Boot Test para escribir y ejecutar escenarios de prueba en lenguaje Gherkin.

Figura 39

Gherkin



Nota. Autor (2024)

Estos escenarios fueron diseñados para cubrir flujos de usuario completos, desde la creación inicial de encuestas hasta la gestión y modificación de preguntas.

Validación Integral del Sistema:

Se verificó que todas las funcionalidades, incluyendo la creación, modificación y eliminación de encuestas y preguntas, funcionaran coherentemente según los criterios de aceptación establecidos.

|Resultados y Ajustes

Durante el ciclo de pruebas:

Se realizaron iteraciones adicionales según los hallazgos de las pruebas, implementando ajustes y mejoras para optimizar el rendimiento y la fiabilidad del sistema.

Se aseguró que todas las pruebas unitarias y de aceptación fueran exitosas antes de

proceder con la entrega.

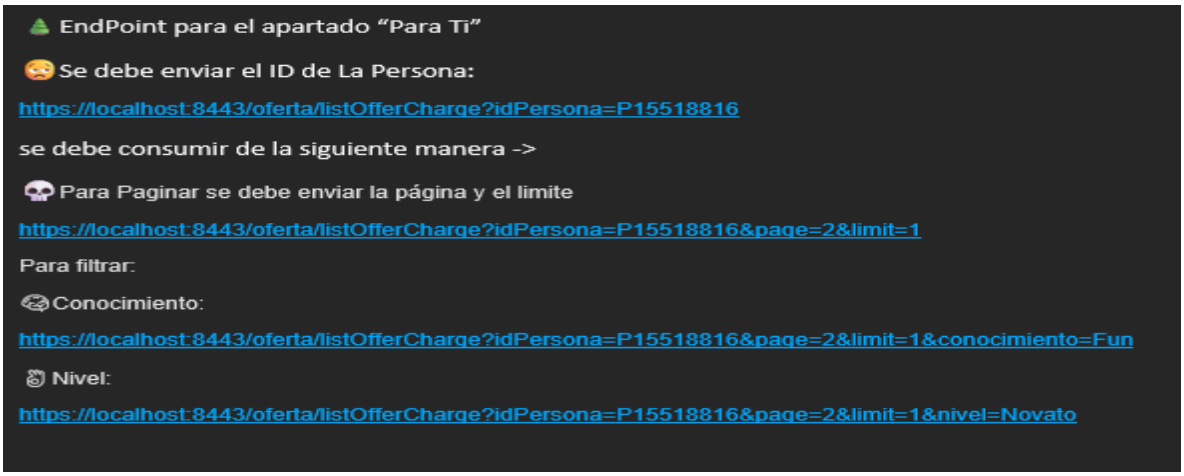
Con esta metodología estructurada y enfocada en la calidad, se aseguró que la funcionalidad de encuestas en Valko desarrollada con Spring Boot y Java 8 cumpliera con los estándares esperados, proporcionando una solución robusta y confiable para los usuarios finales y cumpliendo con las expectativas del Product Owner.

3.1.10 Documentar la forma de uso de EndPoint para entrega al front.

Se elaboró un documento en el que se detallaron los endpoints y las distintas formas de consumirlos. Este documento incluía instrucciones sobre la paginación, el envío de parámetros y otras funcionalidades relevantes. Posteriormente, se organizó una reunión con los compañeros para explicarles en detalle el uso de estos endpoints, abordando específicamente la manera de paginar, cómo enviar los parámetros y otras consideraciones importantes.

Figura 40

Entrega de EndPoint



Nota. Autor (2024)

4 Diagnostico Final

El proyecto de Desarrollo de nuevas funcionalidades en el software de gestión del conocimiento (Valko) de la empresa Bancolombia, haciendo uso de tecnologías Spring Boot se ha completado con éxito, cumpliendo integralmente con los objetivos y requisitos establecidos. Durante el proceso, se estructuró el código siguiendo las practicas de la organizacion utilizando Java 11 con Spring Boot y la base de datos MySQL, logrando una interacción eficiente entre el frontend y la base de datos. Se implementaron controladores robustos que manejan solicitudes del frontend, servicios que gestionan la lógica de negocio y repositorios optimizados con funcionalidades como paginado y consultas condicionadas, minimizando el código redundante. Además, se elaboró un documento detallado que describe los endpoints y las formas de consumirlos, y se realizó una reunión de capacitación con el equipo, explicando en detalle el uso de los endpoints.

El proyecto no solo garantizó una infraestructura sólida y funcional, sino que también proporcionó una base documental y formativa que facilita la comprensión y el uso efectivo del sistema por parte del equipo.

5 Conclusión

En resumen, el desarrollo del backend del módulo de formación en Valko, la plataforma de gestión del conocimiento de Bancolombia, ha representado un avance significativo en nuestra capacidad para gestionar y optimizar eficazmente las encuestas internas. La eliminación de la dependencia de Microsoft Forms ha mejorado considerablemente la eficiencia operativa, permitiendo a los usuarios realizar encuestas directamente en la plataforma. Además, las funcionalidades avanzadas de filtrado en el apartado de formación han permitido a los usuarios acceder a información relevante de manera más sencilla y eficiente, mejorando la experiencia del usuario.

La elaboración de un documento detallado sobre los endpoints y la capacitación del equipo han asegurado una correcta comprensión y utilización de las nuevas funcionalidades implementadas. Este proyecto ha reforzado nuestra capacidad para adaptarnos rápidamente a cambios y optimizar continuamente nuestros procesos internos basándonos en información derivada de los datos. En conjunto, este avance no solo garantiza una infraestructura sólida y funcional, sino que también optimiza la gestión de información en la plataforma, beneficiando a todos los usuarios y contribuyendo al éxito continuo de Valko.

6 Recomendaciones

Para seguir mejorando la plataforma es crucial :

- Feedback de Usuarios: Establecer mecanismos para recibir y analizar feedback de los usuarios de la plataforma. Esto ayudará a identificar áreas de mejora y a desarrollar funcionalidades que realmente aporten valor a los usuarios finales
- **Continuar con el desarrollo y Capacitación Continua:** Mantener el desarrollo continuo de los sistemas y funcionalidades implementadas es crucial. Además, organizar sesiones de capacitación periódicas ayudará a que el equipo esté siempre al tanto de las nuevas herramientas y procesos.

7 Referencias

Baxendale, P., & Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, *13*(6), 377–387. https://doi.org/10.1145/362384.362685

Bota de primavera. (n.d.). Retrieved February 18, 2024, from https://spring.io/projects/spring-boot

Certificación Profesional Scrum Master™ I | Scrum.org. (n.d.). Retrieved June 26, 2024, from https://www.scrum.org/assessments/professional-scrum-master-i-certification

Conócenos | Grupo Bancolombia. (n.d.). Retrieved February 6, 2024, from https://www.grupobancolombia.com/corporativo/conocenos?_ga=2.90474266.1822331360.1707352187-1227725765.1685563431&_gl=1*ayxe6y*_ga*MTAwOTQ1OTkzMi4xNjgwNTU1NjE1*_ga_GVYK05G3HJ*MTcwNzM1MjE4Ny42LjEuMTcwNzM1MjIwOC4zOS4wLjA.

De, S., & Hall, P. (n.d.). *UNIVERSIDAD FEDERICO SRNTA MARIA Introducción a los BASES DE DATOS*.

Effective Cybersecurity: A Guide to Using Best Practices and Standards - William Stallings - Google Libros. (n.d.). Retrieved June 26, 2024, from https://books.google.es/books?hl=es&lr=&id=JMZlDwAAQBAJ&oi=fnd&pg=PT26&dq=Stallings,+W.+(2017).+High+performance+computer+security:+Concepts,+algorithms,+protocols.+Pearson+Education.&ots=q-RqgJ8KFE&sig=QRRXXqR44KubQ5lAJxfirW9Ial4#v=onepage&q&f=false

Estructura | Grupo Bancolombia. (n.d.). Retrieved February 12, 2024, from https://www.grupobancolombia.com/corporativo/gobierno-

corporativo/estructura?_ga=2.158614586.1822331360.1707352187-1227725765.1685563431&_gl=1*o6d0zb*_ga*MTAwOTQ1OTkzMi4xNjgwNTU1NjE1*_ga_GVYK05G3HJ*MTcwNzM2ODY0MC43LjEuMTcwNzM3MDEwNC41NC4wLjA.

Formularios de Microsoft | Encuestas, sondeos y cuestionarios. (n.d.). Retrieved February 18, 2024, from https://www.microsoft.com/en-us/microsoft-365/online-surveys-polls-quizzes

Fraser, S., Beck, K., Caputo, B., Mackinnon, T., Newkirk, J., & Poole, C. (2003). Test Driven Development (TDD). *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2675*, 459–462. https://doi.org/10.1007/3-540-44870-5_84

Gobierno Corporativo | Grupo Bancolombia. (n.d.). Retrieved February 7, 2024, from https://www.grupobancolombia.com/corporativo/gobierno-corporativo?_ga=2.177046565.1822331360.1707352187-1227725765.1685563431&_gl=1*1lwc86o*_ga*MTAwOTQ1OTkzMi4xNjgwNTU1NjE1*_ga_GVYK05G3HJ*MTcwNzM2ODY0MC43LjAuMTcwNzM2ODY0MC42MC4wLjA.

Guía de Scrum | Guías de scrum. (n.d.). Retrieved June 26, 2024, from https://scrumguides.org/scrum-guide.html

IntelliJ IDEA: el IDE líder para Java y Kotlin. (n.d.). Retrieved February 18, 2024, from https://www.jetbrains.com/es-es/idea/

José, I., Pardo, M., Autor, M., Febles, D. A., & Tutora, E. (n.d.). *D DE EC CL LA AR RA AC CI IÓ ÓN N D DE E A AU UT TO OR RÍ ÍA A*.

JUnit 5. (n.d.). Retrieved June 26, 2024, from https://junit.org/junit5/

Ley 23 de 1982 - Gestor Normativo - Función Pública. (n.d.). Retrieved June 26, 2024, from https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=3431

Ley 1273 de 2009 - Gestor Normativo - Función Pública. (n.d.). Retrieved June 26, 2024, from https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=34492

Ley 1680 de 2013 - Gestor Normativo - Función Pública. (n.d.). Retrieved June 26, 2024, from https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=55611

Mockito framework site. (n.d.). Retrieved June 26, 2024, from https://site.mockito.org/

mysql. (n.d.). Retrieved February 18, 2024, from https://www.mysql.com/

Nuestro Propósito | Grupo Bancolombia. (n.d.). Retrieved February 7, 2024, from https://www.grupobancolombia.com/corporativo/conocenos/nuestro-proposito

Objetivos y metas de desarrollo sostenible - Desarrollo Sostenible. (n.d.). Retrieved February 18, 2024, from https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

Política de Protección de Datos Personales -. (n.d.). Retrieved June 26, 2024, from https://www.minambiente.gov.co/politica-de-proteccion-de-datos-personales/

SIC | Sede Electronica. (n.d.). Retrieved June 26, 2024, from https://sedeelectronica.sic.gov.co/

Sistemas de gestión de seguridad |ICONTEC. (n.d.). Retrieved June 26, 2024, from https://www.icontec.org/eval_conformidad/certificacion-iso-27001-sistemas-de-gestion-de-seguridad-de-la-informacion-2/

Software Java | Oracle Colombia. (n.d.). Retrieved February 18, 2024, from https://www.oracle.com/co/java/

Spring Boot. (n.d.). Retrieved June 26, 2024, from https://spring.io/projects/spring-boot

Spring Boot in Action - Craig Walls - Google Libros. (n.d.). Retrieved June 26, 2024, from https://books.google.es/books?hl=es&lr=&id=IzkzEAAAQBAJ&oi=fnd&pg=PT11&dq=+Spring+Boot+in+action.+Manning+Publications.&ots=kJHJW0ueC2&sig=DcxZNe_5dSjJx

xjQ5J-

8S32NaOI#v=onepage&q=Spring%20Boot%20in%20action.%20Manning%20Publications

.&f=false