Acta de Correcciones al Documento de Trabajo de Grado

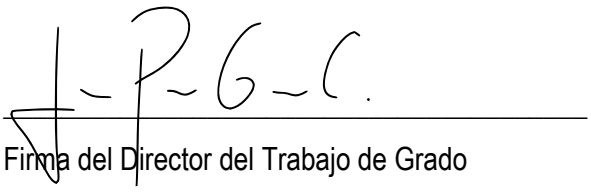
Santiago de Cali, 25 de Julio de 2024

Autor: Johnny Andrés Chinchajoa Taimal

Título del Trabajo de Grado: Dynamic Forms: Plataforma No-Code para la Recolección de Información en Campo por Diferentes Actores

Director: MSc, Juan Pablo García Cifuentes

Como indica el artículo 2.13 de las Directrices para Trabajo de Grado de Maestría, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuaran, como consta en el Acta de Evaluación correspondiente.



Firma del Director del Trabajo de Grado

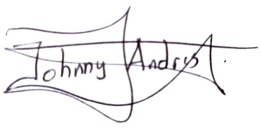
Santiago de Cali, 25 de Julio de 2024

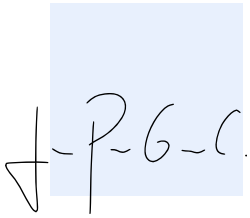
Ingeniero:
Juan Carlos Martínez Arias
Director Posgrados de Ingeniería
Facultad de Ingeniería y Ciencias
Pontificia Universidad Javeriana - Cali

Con el fin de cumplir con los requisitos exigidos por la Universidad para llevar a cabo el Trabajo de Grado y posteriormente optar por el título de Magíster en Ingeniería de Software, nos permitimos presentar a su consideración el proyecto de Trabajo de Grado denominado Dynamic Forms: Plataforma No-Code para la Recolección de Información en Campo por Diferentes Actores, el cual será realizado por el estudiante Johnny Andrés Chinchajoa Taimal con código 8967320, bajo la dirección del MSc, Juan Pablo García Cifuentes.

El suscrito director del Trabajo de Grado autoriza para que se proceda a hacer la evaluación de este Proyecto ante el Tribunal que para el efecto se designe, toda vez que ha revisado cuidadosamente el documento y avala que ya se encuentra listo para ser presentado oficialmente.

Atentamente,





Firma
Johnny Andrés Chinchajoa Taimal

C.C. 1.085.302.446 de Pasto

Firma
Juan Pablo García Cifuentes

C.C. 1.112.759.052 de Cartago

Maestría en Ingeniería de Software
Facultad de Ingeniería y Ciencias

FICHA RESUMEN
TRABAJO DE GRADO DE MAESTRÍA

TITULO: “Dynamic Forms: Plataforma No-Code para la Recolección de Información en Campo por Diferentes Actores”

1. ÉNFASIS: Ingeniería de Software
2. TIPO DE PROYECTO: Aplicado
3. ÁREA DE TRABAJO: Aplicaciones No-Code
4. ESTUDIANTE: Johnny Andrés Chinchajoa Taimal
5. CORREO ELECTRÓNICO: jact1001@javerianacali.edu.co
6. DIRECCIÓN Y TELÉFONO: calle 33 # 94 – 100, apto 608 Nova club, 3176257286
7. DIRECTOR: MSc, Juan Pablo García Cifuentes.
8. VINCULACIÓN DEL DIRECTOR (en la universidad): Planta
9. CORREO ELECTRÓNICO DEL DIRECTOR: jpgarcia@javerianacali.edu.co
10. CO-DIRECTOR(ES) (Si aplica): N/A
11. GRUPO O EMPRESA QUE LO AVALA (Si aplica): N/A
12. OTROS GRUPOS O EMPRESAS: N/A
13. PALABRAS CLAVE (al menos 5): formularios, dinámico, no-code, recolección, información
14. ODS QUE APLICA EL PROYECTO (Agenda 2030): Industria, Innovación e infraestructuras
15. FECHA DE INICIO (Desarrollo del proyecto): 1/09/2022
16. RESUMEN (máximo 400 palabras).

El proyecto aborda el desafío de la transformación digital empresarial en la recolección de datos, la cual ha migrado principalmente hacia formularios digitales. Esta transición ha generado dificultades para las empresas debido a los costos asociados con el desarrollo, mantenimiento y adquisición de soluciones digitales. A pesar de las alternativas ofrecidas por las aplicaciones Low-Code y No-Code, los usuarios aún enfrentan obstáculos, como la rigidez en la estructura de datos, que dificulta la evolución o modificación de los formularios según las necesidades cambiantes de la empresa. El objetivo de este proyecto es diseñar una solución No-Code que permita la creación, modificación y gestión eficiente de formularios digitales, adaptándose a las necesidades de cada empresa sin requerir conocimientos técnicos avanzados. Además, esta solución busca simplificar la recolección de datos, permitiendo la participación eficiente de múltiples actores y ofreciendo la opción de diligenciar los formularios en modo offline.

Dynamic Forms: Plataforma No-Code para la Recolección de Información en Campo por Diferentes Actores

Johnny Andrés Chinchajoa Taimal

Proyecto presentado como requisito parcial para optar al título de:

Magíster en Ingeniería de Software

Director:

MSc. Juan Pablo García Cifuentes

Pontificia Universidad Javeriana Cali Facultad de Ingeniería y Ciencias

Departamento de Electrónica y Ciencias de la Computación

Cali-Colombia

25 de julio de 2024

Contenido

Lista de Tablas

Lista de Figuras

Capitulo 1: Definición del problema

1.1. Planteamiento del problema

La transformación digital, según Galindo (2019), es un cambio que muchas organizaciones están implementando para adaptar y mejorar sus modelos de negocio mediante recursos tecnológicos, con el objetivo de mantenerse competitivas. Uno de los procesos que ha migrado a formatos digitales gracias a esta transformación es la recolección de datos. En este contexto, Cisneros Caicedo et al. (2022) destacan que la encuesta es la técnica más utilizada para obtener información cualitativa y cuantitativa. Esta técnica puede llevarse a cabo tanto en entornos virtuales como físicos, utilizando cuestionarios estructurados en diferentes etapas o secciones, que en algunos casos son completados por diversos actores.

En la actualidad, las empresas están experimentando una creciente necesidad de desarrollo de software. En algunos casos, optan por crear sus propios sistemas y formar equipos de desarrollo internos. Sin embargo, esta estrategia conlleva la necesidad de inversión en educación y formación continua. Lo que a su vez genera costos que pueden superar los presupuestos iniciales del proyecto, especialmente si se considera la inclusión de diseños internos (Sommerville, 2011).

De ahí que, surge como alternativa el software como servicio (SaaS), que, según la definición de Mohammed & Zeebaree (2021), hace que las empresas solo se preocupen de cómo utilizar las herramientas, olvidándose del hardware, sistemas operativos, aplicaciones, etc. Un ejemplo de éxito de estas soluciones es Google Forms de la suite de Google, que se están usando para construir formularios y recolectar información.

Bock y Frank (2021) añaden las aplicaciones Low-Code y No-Code a la lista de opciones para construir herramientas digitales, estableciendo una tendencia tecnológica para mejorar los entornos de transformación digital. No obstante, es necesario realizar un análisis detallado para determinar la mejor opción según las necesidades de cada empresa. Aunque

existen muchas soluciones, estas pueden ser muy costosas y las aplicaciones gratuitas podrían resultar difíciles de mantener. Por ello, es importante llevar a cabo un análisis para identificar la opción más adecuada que satisfaga las necesidades específicas de cada empresa, considerando sus requisitos particulares.

Por ejemplo, Power Apps es una aplicación Low-Code de Microsoft para generar formularios. Hervás Roig (2021) destaca su facilidad para construir soluciones y el ahorro de tiempo en la persistencia de datos. Sin embargo, esta aplicación está limitada a bases de datos compatibles y, dependiendo de la complejidad de la solución, requiere conocimiento técnico para definir el modelo de datos. Por ejemplo, si se necesita asignar partes de un formulario a diferentes actores, un usuario final sin conocimiento técnico podría tener dificultades para crear sus propios modelos de datos. Esto podría resultar en aplicaciones con estructuras de datos rígidas, que impiden agregar o eliminar campos de un formulario, haciendo compleja la evolución del modelo para adaptarse a las características cambiantes del negocio.

Hoy en día las empresas necesitan recopilar datos de diferentes fuentes de información y para lograrlo generan formularios para cada fuente que generalmente es ejecutada por actores de forma independiente. En algunos casos la información recolectada debe ser consolidada en una sola fuente de datos, lo cual genera procesos adicionales y si se realizan de forma manual hacen que la información recolectada pierda integridad.

Por otro lado, las necesidades cambiantes de las empresas requieren actualizaciones frecuentes en los formularios. Para las aplicaciones Low-Code y No-Code, adjuntar dinamismo a un formulario representa un reto significativo. Según Moskal (2021), más del 70 % de las respuestas en Stack Overflow indican que aún no hay una solución aceptada para adjuntar eventos dinámicos a un formulario. Esta situación posiblemente contribuye a la persistencia de soluciones con modelos de datos estáticos, lo que implica la necesidad de soporte técnico para

sus actualizaciones. Esto genera costos adicionales en el proceso de modificación y mantenimiento.

Por último, todavía existen aplicaciones Low-Code y No-Code que carecen de la funcionalidad offline, lo cual resulta crucial para muchas empresas en Colombia. Por ejemplo, en la recolección de información en áreas urbanas con alta densidad de población y zonas rurales sin acceso a internet.

De esta manera, la complejidad y los altos costos que siguen teniendo las aplicaciones Low-Code y No-Code como solución digital factible a la hora de crear formularios, hacen que diferentes negocios con esta necesidad no se hayan migrado a formularios digitales. Soluciones como Microsoft Power Apps, que es una de las plataformas Low-Code populares de acuerdo a Moskal (2021), añaden complejidad adicional en el momento de segmentar la recolección de información en diferentes pasos para asignarlos a diferentes actores, posee limitaciones a la hora de modificar los formularios y aunque permita la recolección de información en modo offline, su configuración requiere de un nivel de conocimiento avanzado sobre la herramienta.

1.2. Formulación del problema

¿Qué estructura de datos permitirá agregar, actualizar o eliminar propiedades de un formulario sin afectar los datos ya recolectados?

¿Qué modelo de datos permitirá segmentar formularios y asignarlos a diferentes actores para su ejecución?

¿Cuál es el diseño de una solución digital para recolectar información en modo offline?

¿Cómo una aplicación No-Code permitirá a las empresas migrar la recolección de información a formularios digitales con la posibilidad de asignarlos a diferentes actores para recolección de información en campo?

Capitulo 2: Objetivos del proyecto

2.1. Objetivo General

Diseñar una solución digital No-Code que permita crear formularios para la recolección de datos en modo online y offline modificables en el tiempo con la opción de segmentarlos para la participación de múltiples actores en el levantamiento de información.

2.2. Objetivos específicos

Definir un esquema de datos que permita agregar, actualizar o eliminar campos de los formularios.

Diseñar un modelo de datos para segmentar los formularios y asignarlos a diferentes actores.

Diseñar una solución digital a partir del modelo de datos que permita crear y modificar formularios, así como recolectar información en campo de manera offline y online, accesible para diferentes actores.

Desarrollar una aplicación web No-Code que implemente y permita validar la solución digital.

Capitulo 3: Alcance

La solución debe soportar como mínimo los siguientes tipos de entradas para formularios digitales: texto, numérico, correo, fecha, combo, checkbox y radio.

La solución diseñada deberá cumplir como mínimo los siguientes atributos de calidad: escalabilidad y desempeño.

La solución tendrá la capacidad de desplegarse en al menos una nube.

La aplicación web No-Code permitirá:

- Construir formularios con la funcionalidad seleccionar, arrastrar y soltar.
- Segmentar el formulario en secciones.
- Crear actores y asignarlos a secciones del formulario.
- Acceder al formulario a través de una URL y mostrar la sección del formulario que corresponda a cada actor.
- Descargar los datos recolectados.
- Modificar los formularios con información almacenada.

Capitulo 4: Justificación

La solución propuesta en este trabajo prioriza la ejecución de partes de un formulario por diferentes actores. La administración de sus campos de acuerdo con las necesidades del negocio y la recolección de información en modo offline, pues son características que difícilmente se encuentran juntas en las herramientas actuales.

Según Lapicki & Terlato (2021) la transformación digital trae consigo gestionar cambios rápidos en los modelos de negocios para seguir siendo competitivos. La inmediatez y la flexibilidad sumadas a la necesidad de innovar son propiedades que modelos de datos tradicionales no contemplan. Estos modelos resultan lentos y demasiado rígidos, y aquí toman importancia modelos de datos que puedan hacer frente a la dinámica de los datos que se presenta en el entorno actual.

Por otro lado, aunque el Gobierno nacional ha desarrollado una serie de esfuerzos para dar conectividad a internet de alta velocidad en todo el territorio colombiano. Quintero (2021) da cifras en las que para el año 2019 había aún población que no contaba con este servicio, pues la conectividad inalámbrica había llegado a 815 municipios de los 1021 que hay en Colombia. Por otro lado, el gran crecimiento de los dispositivos móviles hace que las redes colapsen en las ciudades. Lo que provoca inestabilidad o baja velocidad en la conexión a internet. Por esta razón toma importancia que aplicaciones para recolectar información funcionen en modo offline, para permitir ingresar datos en los formularios si no hay conexión a internet.

Finalmente, en un formulario de recolección de información, los procesos pueden ser llevados a cabo por diferentes actores. Los cuales proporcionan información en áreas específicas del formulario. Por esta razón, es importante que, al construir formularios, además de poder seccionar y agrupar los campos, se agreguen estas secciones a diferentes actores y restringir el diligenciamiento y acceso a otras de ser necesarios.

Capitulo 5: Marco teórico de referencia

En el desarrollo de los objetivos del trabajo es importante resaltar la existencia de antecedentes y un marco de referencia como forma de sustentar el desarrollo de la aplicación. En ese sentido, en primer lugar, se establecen los conceptos claves a tener en cuenta y en segundo lugar un análisis de las tecnologías y herramientas disponibles para llevar a cabo el desarrollo de la solución.

5.1. Marco conceptual y teórico

Transformación digital. De acuerdo con la definición de Martinelli (2021) la transformación digital es el proceso que realizan las organizaciones como estrategia competitiva y modelo de negocio para adoptar herramientas y servicios digitales. En Colombia, según Franco Trujillo (2022) el gobierno está apoyando este proceso con la definición de unos niveles de madurez de transformación para las empresas, de acuerdo con la implementación de: (1) La digitalización de la documentación, (2) herramientas ofimáticas, (3) analítica de datos y (4) software especializado o aplicaciones web. Uno de los procesos que se están migrando dentro de los niveles 1 y 4 es la recolección de datos.

Recolección de datos. La recolección de datos es el proceso que se realiza para adquirir información a través de formularios impresos, formularios digitales, entrevistas telefónicas o personales. Este es un proceso que muchas organizaciones realizaban a menudo con cuestionarios físicos. Los cuales están expuestos a manipulaciones o errores humanos difíciles de controlar. Con la digitalización surge la oportunidad de realizar esta tarea sobre formularios digitales (Kamlofsky y Bergamini, 2014).

Formulario digital. Es el equivalente a un documento de papel que se utiliza para recopilar datos. Estos reproducen todos los elementos de un formulario en papel, como por ejemplo redactar texto, dibujar un boceto, rellenar un checkbox o añadir una firma y además

permiten recolectar los datos de una manera rápida, ordenada y validando los datos ingresados. Esta es la razón por la que las organizaciones, en busca de la transformación digital de este proceso, han empezado a usar alternativas para la construcción de formularios digitales, como es el caso del software como servicio (Huerta y Hurtado, 2021).

Software as a Service (SaaS). De acuerdo con la definición de Bello et al. (2021), estas son soluciones en la nube que proveen de software con un modelo de pago por su uso. El proveedor se encarga de administrar el hardware y software y el usuario solo se debe conectar a través de internet a la aplicación. De esta manera, las empresas solo deben preocuparse de cómo estructurar y construir los formularios, y para ello surgen las herramientas No-Code y Low-Code.

Plataformas No-Code y Low-Code. Hylton et al. (2021) considera una plataforma No-code a una aplicación web o móvil que puede usarse para crear soluciones digitales sin escribir una sola línea de código. Por otro lado, Bock & Ulrich (2021) definen a las plataformas Low-Code como herramientas que permiten el desarrollo, la implementación, la ejecución y la administración de aplicaciones mediante abstracciones de programación declarativas de alto nivel, como lenguajes de programación basados en metadatos y basados en modelos. Según Moskal (2021) los beneficios de usar plataformas de código bajo, también incluyen flexibilidad y agilidad, tiempo de desarrollo que permite una respuesta rápida a las demandas del mercado, corrección de errores reducida, menor esfuerzo de implementación y mantenimiento más fácil.

En ese sentido, herramientas de este tipo les permitirá a las empresas construir formularios de forma sencilla. Por ejemplo, seleccionar y arrastrar los campos que requieren para la recolección organizada de información, que posteriormente se almacena en un base de datos para su recuperación, modificación y eliminación de manera eficiente.

5.1.1. Bases de datos

Las bases de datos, según la definición de Rob & Coronel (2011), son conjuntos de datos organizados para satisfacer las necesidades de información de una organización. Entre

los tipos más comunes se encuentran las bases de datos relacionales, las cuales utilizan tablas interconectadas para estructurar los datos de manera lógica y matemática, donde cada fila representa una instancia de entidades o conceptos.

Sin embargo, la diversidad de información ha impulsado el surgimiento de nuevas opciones de almacenamiento, como las bases de datos NoSQL. Estas, según Benymol & Sajimon (2017), se dividen principalmente en cuatro tipos: clave-valor, documentos, familia de columnas y bases de datos de grafos. Las cuales se destacan por su rapidez en la manipulación de datos, gracias a su enfoque flexible y escalable, convirtiéndolas en una elección óptima para aplicaciones que demandan interacciones fluidas y estructuras de datos adaptables.[1]

Ambos tipos de bases de datos, NoSQL y relacionales, ofrecen opciones diversas en términos de estructuras de datos. En este proyecto, se analizarán estas estructuras para elegir aquella que mejor se adapte a la construcción de formularios dinámicos.

5.1.2. Estructura y modelo de datos

En ese sentido, las estructuras de datos son aquellas que permiten organizar la información de manera eficiente y diseñar la solución correcta para un determinado problema; estos pueden ser árboles, grafos, tablas, etc. Una estructura de datos permite trabajar en alto nivel de abstracción almacenando información para luego acceder a ella, modificarla y manipularla. Según Aguilar (2016) las estructuras de datos se pueden dividir en: datos estructurados, no estructurados y semiestructurados.[2]

Las estructuras de datos desempeñan un papel fundamental en el diseño del modelo de datos, el cual es una representación visual que comprende y define los elementos relacionados con los datos de un sistema, incluyendo tipos de datos, relaciones y restricciones para mantener

[1] Ampliar la definición de base de datos relacionales y No-SQL en el apartado de anexos.
[2] Ampliar información de los tipos de esquemas en el apartado de anexos.

la integridad de los datos. Entre los modelos más utilizados se encuentran los modelos conceptuales, lógicos y físicos.

Los modelos de datos son fundamentales para asegurar que el sistema sea eficiente, escalable, seguro y capaz de adaptarse a los cambios en los requisitos del negocio. Al establecer la estructura y las relaciones de los datos, los modelos de datos proporcionan una base sólida para el diseño de una arquitectura de software.

5.1.3. Arquitectura y diseño de software

La arquitectura de software define la estructura abstracta de un proyecto, incluyendo aspectos como el almacenamiento de datos, las relaciones entre componentes y las capas de implementación. Diferentes definiciones, como la de Fuentes et al (2000) y el documento de IEEE Std 1471-2000 Group, describen la arquitectura de software como la estructura general y organización fundamental del sistema, respectivamente. [3]

En este proyecto, la arquitectura de software desempeñaría un papel fundamental al establecer la visión general y la estructura del sistema, incluyendo cómo se gestionan y procesan los formularios, con el objetivo de garantizar la usabilidad y la experiencia del usuario durante su creación y diligenciamiento.

El diseño de software es un paso esencial para lograr una representación detallada del sistema antes de su implementación, definiendo su estructura, comportamiento y características mediante diversas técnicas y herramientas. Es crucial para desarrollar sistemas de calidad que cumplan con los requisitos y objetivos establecidos. Durante este proceso, se crean representaciones de los componentes del sistema, que proporcionará una base sólida para su implementación posterior. Además, estos diseños pueden evolucionar y adaptarse según las necesidades operativas o comerciales (Cedillo et al., 2022).

[3] Ampliar información y definiciones de arquitectura de software en el apartado de anexos.

En el diseño de formularios, es esencial considerar tanto los aspectos técnicos como los relacionados con el dominio del negocio, comprendiendo las necesidades y requisitos específicos. El enfoque de Diseño Dirigido por el Dominio busca asegurar que el diseño sea eficiente y satisfaga las necesidades del negocio, abordando los desafíos desde el inicio mediante la definición de contextos delimitados. Estos representan los límites en los cuales cada modelo de dominio opera, proporcionando un marco claro para definir las reglas de negocio y las interacciones entre los componentes del sistema, como lo señalan Evans (2023) y Fowler (2014).

La separación de responsabilidades es clave en el diseño de software, reflejándose en enfoques como las arquitecturas limpias y hexagonales. Ambas dividen el software en capas para gestionar dependencias en el código, asegurando una estructura desacoplada. Sin embargo, la arquitectura hexagonal, según Vernon (2013), va más allá al permitir el intercambio de la interfaz de usuario, lo que destaca la importancia de separar la lógica de negocio de los detalles de implementación, promoviendo un diseño modular y adaptable, como propone Cockburn (2005). Esto facilita la evolución del sistema conforme cambian los requisitos y tecnologías.

5.2. Estado del Arte

A continuación, se presenta un análisis de las herramientas disponibles para construir formularios digitales, para cada una se evalúan tres características: a) La ejecución de los formularios sin conexión a internet, b) La modificación de los formularios después de haber recolectado información y c) La segmentación de un formulario para ser ejecutado por diferentes actores. El análisis se realiza a partir de la documentación disponible en cada herramienta.

5.2.1. Plataformas No-Code y Low-Code disponibles para la construcción de formualrios

Dentro de las plataformas analizadas se pudo encontrar que existen alternativas que cumplen con alguna de las características evaluadas en este proyecto, pero no en simultaneo.

Se revisaron 19 aplicaciones [4], de las cuales, en cuanto a la ejecución de los formularios en modo offline, 10 de ellas permiten este modo para el diligenciamiento de los formularios y el restante no lo ofrecen o en su documentación solo se especifica que es para la creación o edición de la solución. En cuando a la edición de los formularios después de haber recolectado información con ellos, las aplicaciones que lo permiten generalmente tienen limitaciones en cuenta a cambiar el tipo de dato y cambiar la estructura del formulario y las que lo permiten generarán un formulario nuevo. Finalmente, en cuanto a asignar actores a secciones específicas de un formulario, la mayoría de las aplicaciones permiten definir permisos para administrar los formularios con diferentes niveles de acceso, pero solo Quickbase define restricciones para el diligenciamiento, sin embargo, su configuración requiere de un nivel de experiencia alto sobre la herramienta y no se especifica si es posible agrupar estas restricciones o niveles de acceso a secciones de un formulario.

A continuación, se presenta una comparación de 10 de las aplicaciones evaluadas para tener una visión más clara de cuáles cumplen o no con los requisitos establecidos.

Tabla 1

Herramientas existentes

Aplicaciones evaluadas	Asignar secciones de formularios a Diferentes Actores	Ejecutar el formulario en Modo Offline	Después de publicar Modificar formularios

[4] Ampliar el análisis de todas las plataformas en el apartado de anexos.

Mendix	Cumple	No cumple	Cumple
Bonita Soft	Cumple	Cumple	No cumple
Appian	No cumple	No cumple	Cumple
Quick Base	No cumple	Cumple	Cumple
Google Forms	No cumple	No cumple	Cumple
Bizagi	No cumple	Cumple	Cumple
Mighty Forms	No cumple	Cumple	No cumple
Suervey Sparrow	No cumple	Cumple	No cumple
Forms On Fire	No cumple	Cumple	No cumple
Power Apps	Cumple	Cumple	No cumple

Nota. Comparación de herramientas para crear formularios. Elaborado por Chinchajoa (2023)

Capitulo 6: Esquema de datos para agregar, actualizar o eliminar campos en un formulario

En estudios realizados por Jose y Abraham (2017) se han demostrado que los entornos actuales, como las redes sociales, la web, los repositorios de imágenes, documentos y formularios, presentan una amplia gama de características y formatos de datos. Esto ha provocado un cambio significativo en las necesidades de gestión y procesamiento de bases de datos en los últimos años. Además, investigadores como Sevilla Ruiz et al. (2015) señalan que las arquitecturas de software modernas requieren un enfoque de escalado horizontal de las bases de datos. Esto ha dado lugar al surgimiento de nuevas tecnologías, como las bases de datos NoSQL, que complementan a las bases de datos relacionales en el almacenamiento y gestión de datos.

En este capítulo se lleva a cabo un análisis de diferentes esquemas de datos presentes en las bases de datos relacionales y NoSQL, con el objetivo de identificar el esquema más adecuado para la construcción de formularios. Estos esquemas pueden ser clasificados como estructurados, semiestructurados y no estructurados, como se muestra en la tabla 2.

El análisis se centra en factores clave como la naturaleza de los datos, los requisitos del sistema, la escalabilidad y la capacidad de procesamiento. Estos elementos serán considerados para determinar cuál esquema de datos se ajusta mejor a las necesidades de los formularios para este proyecto.

Tabla 2

Modelo de datos estructurados y semiestructurados

BD Relacionales	BD NoSQL	
	Modelos agregados	Modelos basados en grafos

Modelo de datos	Modelo Relacional	Llave-valor Documentos Familias de columnas	Grafos con propiedades RDF
	Estructurado	Semiestructurado	Semiestructurado
Consistencia	Fuerte	Débil	Fuerte
Manejo de transacciones	ACID	BASE	ACID
Eficiente en ambientes	Centralizados	Distribuidos	Centralizados
Consultas eficientes	Propósito general	Adecuadas al modelo	Propósito general

Nota. Esquema de datos (Maria Pabon Burbano, Material de la asignatura BD)

6.1. Esquemas de datos estructurados

De acuerdo con Sahatqija et al. (2018) los datos estructurados se utilizan para almacenar información de manera organizada en tablas con una estructura fija. Estos datos se caracterizan por su alta organización y están interrelacionados mediante el uso de llaves primarias y foráneas. [5]

Estas estructuras ofrecen una serie de ventajas. En primer lugar, facilitan la validación y la consistencia de los datos, asegurando que la información ingresada cumpla con los criterios establecidos. Además, simplifican el procesamiento y almacenamiento de los datos, permitiendo una gestión más eficiente de la información recopilada. Otra ventaja importante es que los datos estructurados facilitan el análisis y la extracción de información relevante. Al contar con una estructura clara y organizada, resulta más sencillo realizar consultas y generar informes con base en los datos recopilados.

El proceso de recolección de información utilizando este tipo de esquema se desarrollaría de la siguiente manera:

[5] Ampliar información esquemas de datos estructurados en el apartado de anexos.

1. Definición del esquema: Se identifican los campos necesarios para un formulario, junto con sus tipos de datos y restricciones. Cada campo se convierte en una columna dentro del esquema.
2. Almacenamiento de datos: En un sistema de gestión de bases de datos relacionales como MySQL, se crea una tabla para el formulario según el esquema definido, y cada registro del formulario se almacena como una fila en la tabla correspondiente.
3. Manipulación y análisis de datos: Con los datos almacenados bajo un esquema estructurado, se pueden ejecutar consultas SQL para extraer información específica, generar informes, realizar análisis estadísticos y muchas otras operaciones.

No obstante, el proceso descrito anteriormente puede presentar ciertas limitaciones en la creación de formularios dinámicos. En particular, la definición del esquema de datos con los campos necesarios, su tipo de datos y validaciones se ve restringida a las columnas predefinidas, lo cual puede dificultar su manipulación y actualización. La rigidez de los esquemas de datos estructurados puede generar dificultades al adaptarse a cambios en los requisitos del formulario. Si se requiere agregar nuevos campos o modificar la estructura existente, puede resultar complicado y requerir modificaciones significativas en el esquema de datos. Esto implica un esfuerzo adicional en el proceso de actualización de los formularios.

Asimismo, la gestión de cambios en los formularios puede volverse compleja debido a las limitaciones impuestas por el esquema de datos estructurados, teniendo en cuenta que, cualquier modificación en la estructura puede tener ramificaciones en otras partes del sistema, lo que demanda un cuidadoso mantenimiento y actualización de los datos existentes.

En resumen, aunque los esquemas de datos estructurados brindan beneficios en términos de validación y consistencia de los datos, su rigidez puede dificultar la manipulación y actualización de los formularios. La gestión de cambios y la adaptación a nuevos requisitos pueden requerir esfuerzos adicionales y un cuidadoso mantenimiento del esquema de datos.

6.2. Esquemas de datos semiestructurados

Los esquemas de datos semiestructurados brindan flexibilidad en la organización de la información. Según Khan et al. (2019), estos esquemas permiten almacenar datos en diversos formatos, tales como objetos, llave-valor, familias de columnas y documentos, lo que facilita su adaptación a cambios en la estructura o contenido de los mismos.[6]

Llave-valor. De acuerdo con Edward & Sabharwal (2015) ofrecen una ventaja importante al permitir la incorporación y modificación de campos de manera dinámica, sin la necesidad de realizar una reestructuración completa de la base de datos. Esta característica resulta beneficiosa en el contexto de los formularios, donde los campos y las estructuras de datos pueden cambiar según los requisitos del usuario.

Familias de columnas. Tienen un almacenamiento flexible y eficiente, permitiendo la adición y modificación de columnas sin reestructurar por completo la base de datos (Khan et al., 2019). Esto proporcionaría flexibilidad para ajustar los formularios a necesidades cambiantes sin comprometer la integridad de los datos, evitando cambios estructurales complejos a medida que evolucionan los formularios. Además, brindan una eficiente compresión de datos, lo que reduce el consumo de almacenamiento y maximiza la utilización de recursos disponibles (Chang et al., 2008), beneficiando especialmente a formularios con numerosos campos opcionales o que varían en la cantidad de datos ingresados.

Sin embargo, la flexibilidad de este tipo de esquemas, aunque beneficiosa en muchos aspectos, también puede plantear desafíos en la manipulación y actualización de datos. Por ejemplo, al modificar formularios existentes, la versatilidad puede traducirse en una mayor complejidad, ya que ajustar los campos puede ser más complicado debido a la falta de una estructura fija. Además, las consultas para acceder a datos específicos en columnas pueden

[6] Ampliar información de los esquemas semiestructurados y sus tipos en el apartado de anexos.

resultar más complejas en comparación con esquemas estructurados. Esto se debe a que la naturaleza dinámica de las columnas requiere un conocimiento más profundo de la estructura y la sintaxis de las consultas, lo que implica una curva de aprendizaje adicional para los desarrolladores y analistas de datos. En consecuencia, mientras que la flexibilidad de estos esquemas ofrece ventajas notables, también es importante tener en cuenta los desafíos que pueden surgir en su implementación y mantenimiento.

Documentos. Una característica importante de los documentos es su capacidad para tener estructuras flexibles, lo que significa que no todos los documentos necesitan tener la misma configuración de campos. Esta flexibilidad resulta especialmente útil en el diseño de formularios para este proyecto, donde los campos pueden variar según las necesidades específicas de cada formulario. Esto facilita la incorporación de nuevos campos, la eliminación de campos existentes y las modificaciones en la estructura, sin afectar a otros documentos existentes en el sistema. De acuerdo con Angeles et al. (2018) esta flexibilidad en la estructura de los documentos permite adaptarlos de manera eficiente a los requisitos cambiantes de las aplicaciones.

Una ventaja adicional es la capacidad de almacenar datos relacionados en un solo documento, lo que ocasiona una mejora significativa en la eficiencia y velocidad de acceso a la información de acuerdo con Gupta et al. (2017). En el contexto de formularios, esto implica que todos los datos ingresados en un formulario pueden ser almacenados en un solo documento, evitando así la necesidad de realizar consultas múltiples o combinar datos de varias tablas.

De esta manera, el proceso de recopilación de información utilizando este tipo de esquema se desarrollaría de la siguiente manera:

1. Definición de esquema: No es necesario tener un esquema estricto y predefinido. En su lugar, es posible definir un esquema flexible que permita la inclusión

de campos variables y adaptarse a diferentes tipos de datos. Para el caso del proyecto con el mismo esquema es posible construir diferentes formularios.

2. Almacenamiento de datos: Una vez que los datos son recolectados, se pueden almacenar en una base de datos que admita datos semiestructurados. Un ejemplo común de este tipo de base de datos es MongoDB, que utiliza un formato de documentos JSON.

3. Manipulación y análisis de datos: Este tipo de esquemas permite hacer consultas de búsqueda y filtrado, agregación de datos, análisis de tendencias y cualquier otra operación que sea relevante para los datos recopilados.

La principal ventaja de utilizar un esquema de datos semiestructurados radica en su flexibilidad para adaptar los formularios a diversos requisitos, ya que se pueden agregar nuevos campos o modificar la estructura existente sin realizar cambios significativos en el esquema. No obstante, es importante considerar que la ausencia de una estructura fija puede implicar un procesamiento adicional para validar y extraer información de los datos almacenados.

En resumen, los esquemas de datos semiestructurados brindan flexibilidad en la organización de los datos, ya que permiten el almacenamiento en árboles, grafos o documentos sin requerir una estructura predefinida. Esta flexibilidad facilitaría la adaptación a cambios y la gestión de los formularios.

A continuación, se llevará a cabo un análisis de los grafos con propiedades, que se podrían considerar como una categoría de estructura semiestructurada dentro de los grafos. Esta clasificación se basa en la capacidad de agregar atributos o propiedades a los nodos y aristas del grafo.

Grafos. Se enfocan en el almacenamiento y consulta de datos con relaciones y conexiones entre entidades, de acuerdo con Angles et al. (2017) [7]. Pueden ser de gran utilidad en el diseño de formularios, ya que permiten representar de manera eficiente las relaciones

[7] Ampliar información grafos en el apartado de anexos.

entre los campos y agregar propiedades a los nodos y aristas que los componen. Al modelar los campos como nodos y las conexiones entre ellos como aristas etiquetadas con propiedades, se puede capturar de manera efectiva la interdependencia y la conexión entre los distintos elementos de los formularios, considerando tanto las relaciones estructurales como las propiedades asociadas. Esta representación gráfica enriquecida facilita la administración y el análisis de las relaciones y características de los datos del formulario, lo que ocasiona un diseño flexible y adaptable a medida que evolucionan los requisitos, en este caso, las necesidades específicas de cada formulario. En resumen, los grafos con propiedades ofrecen una solución eficaz para gestionar y optimizar las relaciones y atributos de los campos de un formulario.

Además, la utilización de grafos podría resultar beneficiosa cuando los campos de un formulario están sujetos a reglas de validación, condicionales, dependencias o restricciones de visualización. Estos esquemas permiten representar y almacenar de manera eficiente la información relacionada con estas reglas y conexiones entre los campos. Esto facilita la implementación de lógica dinámica en el formulario, adaptando su comportamiento en tiempo real de acuerdo con las condiciones y dependencias establecidas. Según el estudio de Patil et al. (2014), el uso de grafos agiliza la gestión de reglas y relaciones, proporcionando una interacción más eficiente y personalizada con los usuarios.

Sin embargo, aunque los grafos pueden resultar útiles en ciertos contextos de recolección de datos, es importante tener en cuenta que pueden volverse complejos cuando se trabaja con una gran cantidad de campos o relaciones entre ellos. Esta complejidad puede dificultar la comprensión y el mantenimiento de los formularios, especialmente si se requieren cambios frecuentes. Además, la estructura y la navegación de los grafos pueden resultar confusas y poco amigables para aquellos usuarios que no están familiarizados con este tipo de representación.

6.3. Esquema de datos no estructurados

Los esquemas de datos no estructurados se caracterizan por no seguir una estructura específica para organizar la información, por lo cual la construcción de formularios puede resultar desafiante. Existen enfoques y técnicas que se pueden utilizar para abordar esta situación. A continuación, se presentan algunas técnicas de acuerdo Henriksen et al. (2022): [8]

1. Anotación de datos: Permitiría extraer información relevante de formularios, facilitando su comprensión y análisis.
2. Uso de metadatos: Servirían como una guía para validar y normalizar los datos en formularios, mejorando su calidad y procesamiento.
3. Aprendizaje automático y clasificación: Generaría formularios adaptables, mejorando la eficiencia y precisión en la captura y organización de datos.
4. Interfaz intuitiva y adaptativa: Una interfaz adaptable minimiza errores y se ajusta a las necesidades de cada usuario, facilitando por ejemplo el ingreso eficiente de información en los formularios.
5. Extracción y transformación de datos: Después de recopilar los datos, se transforman en una estructura organizada para un análisis eficiente y la toma de decisiones informadas.

A pesar de la utilidad de estas técnicas, sigue siendo desafiante extraer información relevante de manera precisa. Esto se debe a la complejidad inherente de identificar y seleccionar los datos importantes, así como a las tareas de análisis y procesamiento que les siguen, las cuales pueden requerir normalización, limpieza y transformación de los datos. Además, la falta de una organización estricta puede dificultar la validación y garantía de

[8] Ampliar información de las técnicas para el procesamiento de datos no estructurados en el apartado de anexos.

integridad de los datos recopilados, presentando desafíos adicionales en términos de interoperabilidad con otros sistemas y aplicaciones.

6.4. Ventajas y desventajas de los esquemas de datos

Las tablas 3 y 4 presentan el comparativo que resume las ventajas y desventajas de los esquemas de datos, relevantes para la construcción de formularios.

Tabla 3

Ventajas esquemas de datos

Ventajas		
Esquemas estructurados	Esquemas semiestructurados	Esquemas no estructurados
Facilitan la organización y coherencia de los datos.	Ofrece flexibilidad al capturar información, permitiendo la inclusión de datos personalizados dentro de un formato predefinido.	Alta flexibilidad al permitir ingresar información en el formato y estilo que se requiera.
Se pueden establecer reglas y restricciones para garantizar la validez y consistencia de los datos ingresados.	Es posible realizar modificaciones en la estructura de acuerdo con las necesidades específicas del negocio.	Se puede agregar, modificar o eliminar campos según sea necesario sin afectar estructuras existentes.
Facilidad de análisis y procesamientos a través de consultas y de uso de herramientas de análisis.	La capacidad de adaptarse a datos variables permite una experiencia más fluida del usuario. Esto mejora la eficiencia en la captura de	Mayor expresividad de la información porque permite capturar información de manera más descriptiva y detallada.

	información y la satisfacción del usuario.	
Promueven la precisión y coherencia en la entrada de información.	Los usuarios tienen la posibilidad de proporcionar información adicional, lo que aumenta la expresividad de los datos capturados. Esto permite obtener detalles más precisos y contextualizados, enriqueciendo la calidad de la información recopilada.	Se captura información compleja o subjetiva que no se ajusta a categorías o formatos predefinidos

Nota. Ventajas de los diferentes tipos de estructura de datos en la construcción de formularios.

Elaborado por Chinchajoa (2023)

Tabla 4

Desventajas esquemas de datos

Desventajas		
Esquemas estructurados	Esquemas semiestructurados	Esquemas no estructurados
Limitan la flexibilidad en la captura de información, lo cual puede generar dificultades y costos adicionales en el mantenimiento.	Presentan dificultades en la validación y el control de calidad de la información ingresada debido a su flexibilidad y dificultad para definir reglas y restricciones.	Dificultad en la validación y control de calidad de la información ingresada.

La creación de formularios requiere de más tiempo, esfuerzo porque se debe definir campos, las relaciones y considerar validaciones necesarias.	Mayor complejidad en el análisis, búsqueda y procesamiento de los datos.	Mayor complejidad en el análisis, búsqueda y procesamiento de los datos.
Las restricciones en la forma de ingresar los datos limitan la libertad del usuario para expresar información detallada y contextualizada.	Dificultan la comunicación, el intercambio de datos con otros sistemas o aplicaciones que requieren datos estructurados.	Dificultad en la integración, comunicación y reutilización de recursos con otros sistemas o aplicaciones que requieren datos estructurados o semiestructurados.
Limitaciones de rendimiento y escalabilidad.	Mayor posibilidad de errores e inconsistencias en el momento de capturar datos.	
Dificultad para manejar datos de diferentes tipos/formatos.		

Nota. Cuadro comparativo de los diferentes tipos de estructura de datos. Elaborado por Chinchajoa (2023)

6.5. Storytelling de como añadir campos a un formulario

A continuación, se presentan dos ejemplos de storytelling que ilustran las implicaciones de agregar un campo a un formulario en un esquema de datos estructurado y semiestructurado.

Añadir un nuevo campo a un formulario con esquema de datos estructurado. Érase una vez un usuario responsable de gestionar el proceso de atención a clientes en una empresa. Este usuario se enfrentaba a un desafío: necesitaba incorporar un nuevo campo al formulario de registro de clientes para recopilar información relevante sobre las preferencias de estos.

Sin embargo, descubrió que, para llevar a cabo esta modificación, debía contactar al departamento de sistemas, ya que la única manera de realizarla era modificando directamente la base de datos. Este formulario estaba respaldado por una base de datos relacional, en la cual cada campo del formulario tenía su propia columna. Esto implicaba que añadir un nuevo campo requería la intervención de un experto en bases de datos capaz de ajustar el esquema de la base de datos insertando una nueva columna.

Consciente de la importancia de este nuevo campo para comprender mejor las necesidades de los clientes, el usuario se puso en contacto con el departamento de TI, donde le aseguraron que pronto sería contactado por un experto en bases de datos. Cuando llegó el momento, discutieron los requisitos del nuevo campo y cómo se integraría de manera efectiva en el esquema de la base de datos existente. Tras una cuidadosa planificación y ejecución por parte del experto en bases de datos, el nuevo campo fue agregado con éxito al formulario, permitiendo al usuario recopilar la información necesaria para mejorar aún más su aplicación.

Añadir un nuevo campo a un formulario con esquema de datos no estructurado. En otro rincón del mundo digital, otro usuario se enfrentaba a un desafío similar. Este usuario también necesitaba agregar un nuevo campo al formulario de registro de clientes, pero en este caso, su aplicación estaba respaldada por una base de datos no relacional con un esquema de datos semiestructurado.

Afortunadamente, este usuario tenía una herramienta a su disposición: una interfaz de usuario intuitiva dentro de una aplicación No-Code que le permitía editar el formulario directamente. Con solo unos pocos clics y sin necesidad de conocimientos técnicos profundos, el usuario pudo agregar el nuevo campo al formulario, adaptándolo perfectamente a sus necesidades cambiantes.

Esta experiencia demostró la flexibilidad y la agilidad de trabajar con una base de datos no relacional y una interfaz de usuario intuitiva. Nuestro usuario pudo adaptarse rápidamente a los cambios y mejorar su aplicación sin depender de la intervención de un experto en bases de datos.

6.6. Esquema de datos para la construcción de formularios dinámicos.

Considerando el análisis previo, los datos estructurados se distinguen por su definición precisa, lo que simplifica su consulta y análisis. Su organización clara facilita la validación y el control de calidad, promoviendo precisión, coherencia e interoperabilidad. No obstante, en la creación de formularios dinámicos, su estructura requiere un diseño más complejo. Además, pueden presentar limitaciones en cuanto a flexibilidad para actualizaciones, necesitando la intervención de expertos.

Por otro lado, los datos no estructurados contienen una gran cantidad de información valiosa, aunque su procesamiento y análisis pueden ser más complejos. En la creación de formularios, estos datos ofrecen flexibilidad en la captura de información y mayor expresividad. Sin embargo, presentan desventajas en términos de validación, análisis, interoperabilidad y búsqueda eficiente de datos.

Finalmente, los datos semiestructurados sobresalen por su capacidad de adaptarse a cambios y requisitos variables al no seguir un formato riguroso. Ofrecen flexibilidad en la captura de información y se ajustan bien a entornos donde los requisitos pueden evolucionar

con el tiempo. Aunque pueden requerir técnicas adicionales para su análisis y consulta, existen herramientas que han mitigado esta limitación. Maté-Jimenez (2014) analiza a fondo las ventajas y desventajas de estos esquemas, considerándolos mediadores entre los esquemas estructurados y no estructurados.

Basándose en este análisis, se concluye que para la construcción de formularios en este proyecto, la opción más acertada es utilizar modelos de datos semiestructurados. Sin embargo, es importante tener en cuenta que la falta de una estructura formal puede dificultar el análisis y el control de calidad de los datos. Para abordar esta preocupación, una alternativa es establecer formas de normalización y mantenimiento de la calidad de los datos. Según Aguilar-Borrero (2022), una metodología de diseño adecuada facilita el tratamiento de los datos semiestructurados, ajustándose al modelo de negocio y las necesidades de tratamiento requeridas.

6.5. Diseño de esquema de datos

A continuación, se presenta un esquema de datos semiestructurados utilizando documentos JSON, los cuales permiten ajustar la estructura a cambios, tal como se analiza en Lv et al. (2018).

Un dato semiestructurado se compone de uno o más pares clave-valor, donde las claves representan propiedades de los datos. Según Charito (2018), los valores pueden ser:

1. Atómicos (números, strings o booleanos).
2. Objetos embebidos.
3. Una referencia a otro objeto que será un string o un entero que coincide con el valor de un campo del objeto referenciado (similar a joins en bases de datos relacionales).
4. Un array de valores, que puede ser homogéneo o heterogéneo.

Teniendo en cuenta la anterior definición, a continuación, se presenta el diseño de los esquemas para la representación de formularios, que define las secciones, accesos y campos; la representación de casos de un formulario, donde se almacenan los valores que los campos de una sección tomarán en un caso específico para un formulario y la representación de los formularios y sus casos de uso por usuario, que permitirá un acceso eficiente a los formularios y casos de un usuario. [9]

Esquema para formularios.

1. id (string): Identificador de formulario.
2. form_name (string): Nombre de formulario.
3. state (string): Estado del formulario (enable, disable).
4. sections (array): Secciones de formulario.
   - 4.1. id (string): Identificador de sección.
   - 4.2. section_name (string): Nombre de sección.
   - 4.3. access (array): Usuarios que podrán acceder a las secciones.
     - 4.3.1. user_id (string): Identificador de rol o usuario.
     - 4.3.2. user_name (string): Nombre de rol o usuario.
     - 4.3.3. permissions (array): Lista de permisos (lectura y/o escritura).
   - 4.4.fields (array): Lista de campos.
     - 4.4.1. file_id (string): Identificador de campo.
     - 4.4.2. is_required (boolean): Bandera de requerido.
     - 4.4.3. type (string): Tipo de campo.
     - 4.4.4. label (string): Nombre del campo.

Esquema para casos de formulario.

[9] En la sección de "Implementación de esquemas de datos" de los anexos, se proporciona ejemplos de los esquemas de datos.

1. id (string): Identificador de casos.
2. Case_name (string): Nombre de casos.
3. Case_state (string): Estado de casos (pendiente, en progreso y finalizado).
    3.1. Id (string): Identificador de estado
    3.2. Name (string): Nombre del estado
4. Form_id (string): Identificador de formulario.
5. Sections (array): Secciones de formulario.
    5.1. Section_id (string): Identificador de sección.
    5.2. Section_state (string): Estado de sección.
    5.3. Fields (array): Lista de campos.
        5.3.1. File_id (string): Identificador de campo.
        5.3.2. Value: Valor del campo para el caso

Esquema de formularios y casos de formularios por usuario.

1. User_id (string): Identificador de usuario.
2. Forms(array): Formularios de usuario.
    2.1. Form_id (string): Identificador de formulario.
    2.2. Form_name (string): Nombre del formulario.
    2.3. Form_author (string): Autor del formulario.
    2.4. Cases (array): Lista de casos del formulario para el usuario.
        2.4.1. Case_id (string): Identificador de caso de uso.
        2.4.2. Name: Nombre del caso de uso
        2.4.3. State (object): Estado del formulario
            2.4.3.1. Id (string): Identificador de estado
            2.4.3.2. Name (string): Nombre de estado

Con estos esquemas, se brinda una mayor flexibilidad en la construcción de formularios, permitiendo a los usuarios adaptar y modificar la estructura de datos según las necesidades cambiantes del negocio. En el próximo capítulo, se explorará el diseño de un modelo de base de datos que facilite su tratamiento y organización de manera adecuada.

Capitulo 7: Diseño de un modelo de datos para segmentar los formularios y asignarlos a diferentes actores

De acuerdo con Dayley (2014) diseñar el modelo de datos de una aplicación, es fundamental para establecer los tipos de datos que se van a manejar, la forma en que se desea almacenar y cómo se realizarán las consultas a esos datos. Estos aspectos son claves para establecer una estructura adecuada que permita una gestión eficiente y coherente de la información en una aplicación.

Recientemente, un informe de Dataversity Chillón et al. (2023) ha resaltado la importancia del modelado de datos en bases de datos NoSQL. Es esencial que los desarrolladores de software tengan en cuenta el modelo de datos al escribir código, ya que deben comprender su estructura para implementar de manera efectiva la funcionalidad requerida.

En este proyecto, se ha optado por un esquema de bases de datos NoSQL por su flexibilidad y escalabilidad. Estos esquemas ofrecen ventajas sobre las bases de datos SQL, como una mayor capacidad de réplica en varios servidores y un menor consumo de recursos, lo que reduce costos y aumenta la eficiencia. Además, los esquemas NoSQL eliminan muchas complejidades que suelen ser inevitables en los modelos tradicionales de SQL (Herranz-Gómez, 2014).

Antes de crear el modelo de datos, se realizará un análisis detallado de las bases de datos NoSQL para asegurar un almacenamiento y acceso óptimos, cumpliendo con los requisitos del proyecto y teniendo en cuenta el esquema de datos ya definido. Las bases de datos NoSQL ofrecen varios enfoques de almacenamiento, como modelos clave-valor, orientados a columnas y orientados a documentos. Cada uno tiene ventajas únicas, por lo que es crucial evaluar cuál es el más adecuado para el proyecto. Factores importantes que considerar incluyen escalabilidad, consistencia, consultas, replicación, transacciones,

tolerancia a fallos, integración, sincronización móvil y costos de mantenimiento (Herranz-Gómez, 2014).

7.1. Gestores de bases de datos NoSQL

Se puede encontrar tipos de bases de datos NoSQL10, como es el caso de DynamoDB, una base de datos NoSQL desarrollada internamente por Amazon, proporciona una solución económica para almacenar grandes cantidades de datos. Sin embargo, está restringida al entorno de AWS y no es adecuada para almacenar documentos complejos con estructuras anidadas. Es esencial considerar estas limitaciones al evaluar su idoneidad para el proyecto, junto con la necesidad de equilibrar la consistencia y la disponibilidad según los requisitos específicos (Kalid et al., 2017).

También existe, Cassandra, un proyecto de código abierto lanzado por Facebook. Destacada por su distribución, escalabilidad y capacidad para manejar grandes volúmenes de datos, ha ganado reconocimiento en el ámbito del análisis de big data. Sin embargo, es esencial tener en cuenta que Cassandra no admite documentos, un aspecto relevante a considerar para el desarrollo del proyecto.

Voldemort, por otro lado, creada por LinkedIn, es una base de datos distribuida diseñada para abordar los desafíos de escalabilidad presentes en las bases de datos relacionales (Fowler, 2014). Destaca por su capacidad para almacenar datos no estructurados o semiestructurados, como documentos JSON o XML. Sin embargo, carece de soporte para consultas complejas, lo que puede limitar su flexibilidad en la manipulación de datos, especialmente en formularios con estructuras complejas.

Otra opción NoSQL es Hbase, es una base de datos con un modelo llave-valor que prioriza el procesamiento distribuido, ideal para aplicaciones de big data. Aunque es versátil,

[10] Se presenta más información de los gestores de bases de datos NoSQL en el apartado de anexos.

carece de soporte SQL y requiere una interfaz de programación para la manipulación de datos. Funciona en clústeres para escalabilidad y tolerancia a fallos. Su complejidad en la configuración y administración, especialmente al escalar y distribuir datos, puede exigir conocimientos técnicos adicionales (Herranz-Gómez, 2014).

Por su parte, Riak es una base de datos distribuida con un enfoque en tolerancia a fallos y un modelo clave-valor, que utiliza JSON para interoperabilidad. Destaca por su capacidad para manejar cargas de trabajo web intensivas, lo que permite una escalabilidad eficiente y respuestas rápidas (Fuentes et al., 2023). Sin embargo, sus limitaciones en consultas complejas y su enfoque en el modelo clave-valor requieren una cuidadosa consideración al usarlo en formularios con estructuras complejas. Es crucial evaluar si las características clave-valor de Riak satisfacen las necesidades específicas del proyecto, ya que las restricciones en las consultas pueden afectar su capacidad para manejar datos complejos o requisitos avanzados de búsqueda (Herranz-Gómez, 2014).

CouchDB es un sistema de almacenamiento de datos basado en documentos JSON, conocido por su capacidad para replicar datos en varios dispositivos y su flexibilidad para estructurar y distribuir información. Permite duplicar datos en diferentes nodos, aunque tiene limitaciones para consultas dinámicas. A pesar de esto, CouchDB es apreciado por su capacidad de ejecución en diversas plataformas y su enfoque en la seguridad y la preservación de versiones anteriores de los documentos (Campoverde et al., 2015).

BaseX, por su lado, es una base de datos documental de arquitectura cliente-servidor que permite operaciones simultáneas de lectura y escritura, con alta escalabilidad y rendimiento. Admite documentos en XML, JSON y otros formatos binarios, y está desarrollada en Java y XQuery (Johnson et al., 2009). Sin embargo, su diseño está enfocado principalmente en XML, lo que puede complicar el almacenamiento y la consulta de datos en JSON.

Finalmente, esta Mongo BD, es una base de datos orientada a documentos que destaca por su versatilidad, potencia y facilidad de uso. Puede gestionar tanto grandes como pequeños volúmenes de datos con eficiencia, utilizando JSON para la interacción con los usuarios y BSON, una versión binaria de JSON, para almacenamiento interno, lo que ahorra espacio y aumenta la eficiencia y compatibilidad con diversos lenguajes de programación (Salazar, 2014). Una característica clave de MongoDB es su capacidad para realizar consultas dinámicas, permitiendo la exploración de datos sin planificación previa, lo que aporta flexibilidad. Desarrollado en C++, MongoDB se ha posicionado como una solución confiable y escalable en diversas aplicaciones (Capdevila, 2015).

MongoDB ofrece varias características importantes, las cuales son resumidas por Benymol y Sajimon (2017) en: flexibilidad, lenguaje de consulta enriquecido, sharding, facilidad de uso, alto rendimiento, alta disponibilidad y compatibilidad con varios motores de almacenamiento.

7.1.1. Comparación de gestores de datos NoSQL

A continuación, se presenta la comparación de características de los gestores de datos evaluados.

Tabla 5

Análisis de características de bases de datos

Características	Bases de datos				
	MongoDB	DynamoDB	Cassandra	Voldemort	Hbase
Escalabilidad	Excelente	Excelente	Buena	Buena	Buena
Modelo de consistencia	Flexible	Rígido	Rígido	Rígido	Rígido
Replicación	Sí	Sí	Sí	Sí	Sí
Transacciones	Sí	Sí	Sí	No	Sí
Sincronización móvil	Sí	No	No	No	No

Costo	Moderado	Moderado	Moderado	Moderado	Bajo

Nota. Características de Bases de datos. Elaborado por Chinchajoa (2023)

De acuerdo con el análisis previo y la comparación de la tabla 5, tanto MongoDB como DynamoDB son adecuadas para el proyecto debido a su flexibilidad para manejar esquemas de datos semiestructurados, esenciales para formularios dinámicos. Sin embargo, MongoDB ofrece ventajas adicionales, como la capacidad de realizar consultas complejas y análisis, la compatibilidad con diversos lenguajes de programación del lado del servidor, y una comunidad más activa que proporciona mayor soporte y recursos. Estas características agilizan el desarrollo, simplifican el mantenimiento y permiten una extracción eficiente de información. Aunque DynamoDB es una opción viable si se decide utilizar exclusivamente AWS, las ventajas mencionadas de MongoDB lo convierten en la mejor opción para este proyecto. Por estas razones, se elige MongoDB.

7.2. Diseño del modelo de datos

Después de revisar las bases de datos NoSQL, es fundamental respaldar el esquema definido en el capítulo 6 con un modelo de datos que facilite la recolección de información con formularios de manera estructurada. Esto ayuda a garantizar que los datos sean consistentes y estén adecuadamente organizados para un análisis eficaz (Capacho y Nieto ,2017).

Para este trabajo se presenta un diseño de modelo que consta de tres niveles de abstracción: modelo conceptual, modelo lógico y modelo físico. Estos niveles proporcionan una estrategia sólida para el diseño de bases de datos, permitiendo una comprensión clara y progresiva de la estructura y organización de la información (Capacho y Nieto, 2017). A continuación, se detallan cada uno de los niveles señalados.

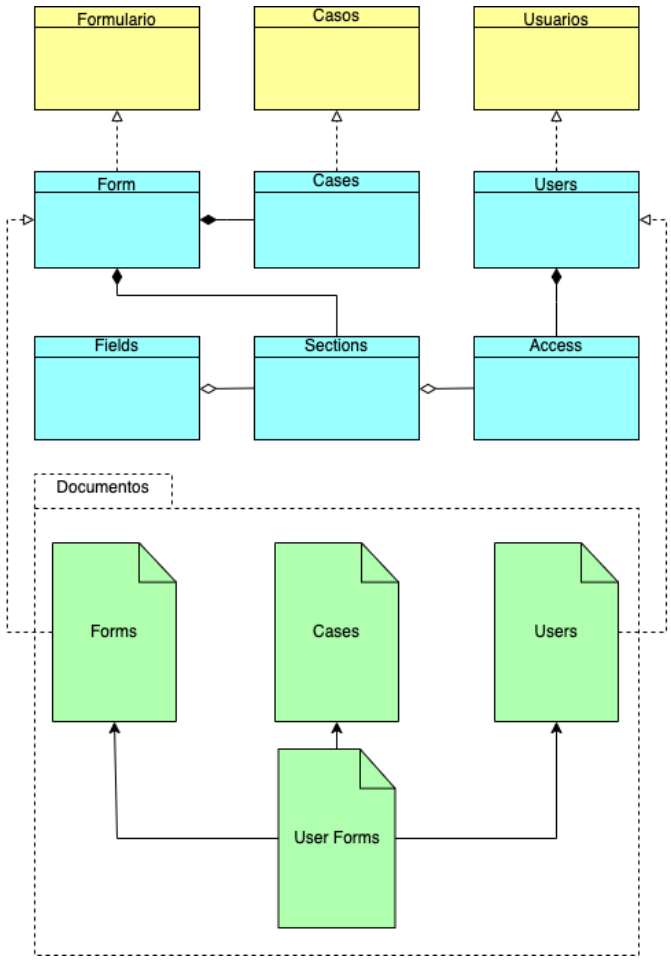
7.2.1. Modelo conceptual

El modelo conceptual proporciona una representación abstracta de alto nivel de los conceptos y relaciones relevantes para el dominio del problema. Se centra en la comprensión conceptual de la información y no se preocupa por los detalles técnicos de su implementación. Permitirá entender la relación y la organización de los conceptos a alto nivel Charito (2018).

- Está dirigido a: Stakeholders y developers.
- Pregunta que responde: ¿Cuáles son las entidades principales y sus relaciones?
- Tipo de diagrama: Information Archimate Viewpoint.

Figura 1

Modelo conceptual



Nota. Elaborado por Chinchajoa (2023)

En este modelo, se identifican tres entidades principales a nivel de negocio: formularios, casos de uso de un formulario y usuarios que los ejecutan. Los formularios están compuestos por secciones, que a su vez contienen campos, y se asignan a usuarios para su diligenciamiento. Tanto los formularios como los usuarios pueden tener al menos un caso asociado. Aunque el modelo de datos abarca más que estas tres entidades, representan el núcleo central de la aplicación y sus principales componentes estructurales.

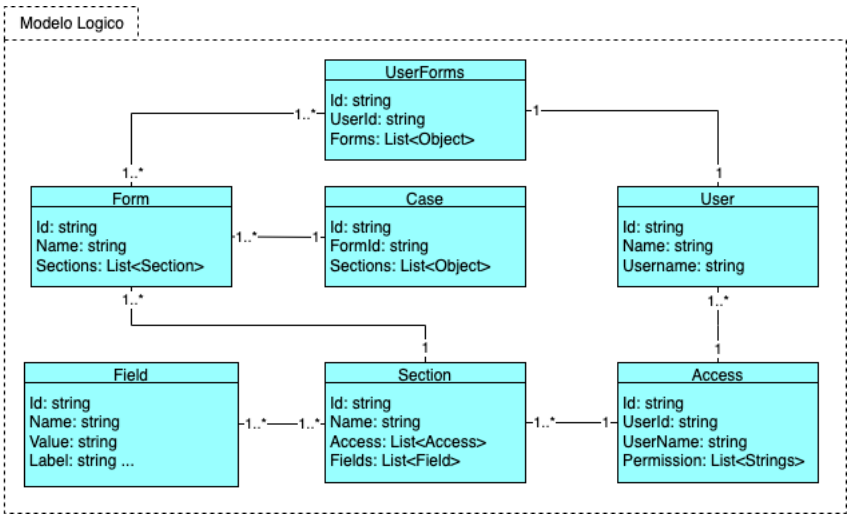
7.2.2. Modelo lógico

El modelo lógico es una representación más detallada del modelo conceptual, que se traduce en estructuras y relaciones específicas. Este nivel de abstracción se enfoca en la organización y estructura lógica de los datos, se trata de un modelo entidad relación que describe las propiedades susceptibles a ser utilizadas, sus tipos y la cardinalidad entre clases (Capacho y Nieto, 2017).

- Está dirigido a: Developers y el gestor de base de datos.
- Pregunta que responde: ¿Cómo se organizan los datos y qué estructura tienen?
- Tipo de diagrama: Information Archimate Viewpoint.

Figura 2

Modelo lógico



Nota. Elaborado por Chinchajoa (2023)

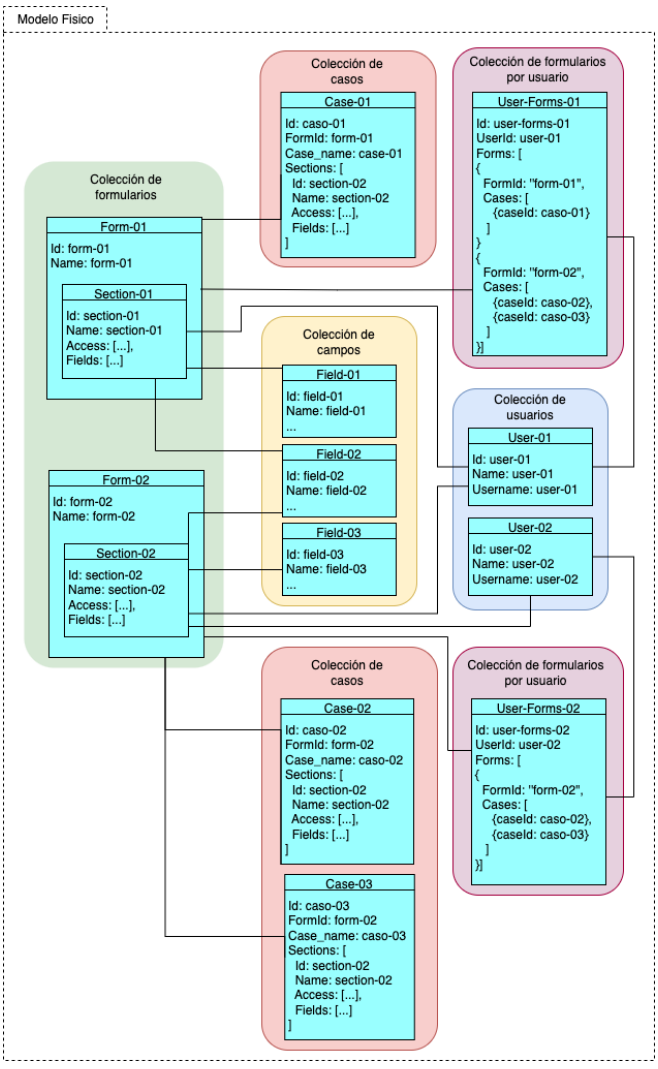
7.2.3. Modelo físico

El modelo físico se refiere a la implementación concreta del modelo en un sistema de gestión de bases de datos específico. Se centra en los detalles técnicos de cómo se almacenan, acceden y gestionan los datos en el entorno físico (Capacho y Nieto, 2017). El propósito del modelo es describir un patrón de cómo debe crecer la base de datos.

- Dirigido a: Developers y gestor de base de datos
- Pregunta que responde: ¿Cómo se implementa el modelo de datos en MongoDB?
- Tipo de diagrama: Representación propia de colecciones y sus relaciones en MongoDB

Figura 3

Modelo Físico



Nota. Elaborado por Chinchajoa (2023)

Con la definición del modelo de datos NoSQL, se busca obtener una comprensión más profunda de la estructura de la información de los formularios y su relación con las entidades asociadas. Este conocimiento será fundamental para el diseño de una solución digital eficiente y adaptable. En el siguiente capítulo, se presenta el diseño de una arquitectura de software que capitalice al máximo las capacidades y flexibilidad que ofrece este modelo de datos, permitiendo así la creación de un sistema robusto y escalable.

Capitulo 8: Diseño de una solución digital a partir del modelo de datos, que permita crear formularios, modificarlos y recolectar información en campo (online/offline) por diferentes actores

A partir de los resultados de los capítulos 6 y 7, en este capítulo se diseña una arquitectura de software que brinde una estructura sólida y escalable para la implementación de la solución digital. Esta arquitectura facilitará la integración de componentes, mejorará la eficiencia operativa y garantizará la adaptabilidad y sostenibilidad del sistema a largo plazo. Según Cambarieri et al. (2023), una arquitectura de software es un factor clave para optimizar el proceso de desarrollo y alcanzar resultados exitosos y de alta calidad en los proyectos de software.

Es importante establecer que, en el contexto de las aplicaciones modernas, los paradigmas informáticos tradicionales y la arquitectura de aplicaciones monolíticas están demostrando ser ineficientes (Alonso, 2021) Por lo tanto, se propone una arquitectura de software limpia y flexible que cumpla con los siguientes atributos de calidad: escalabilidad, seguridad y desempeño.

Tomando en consideración estos aspectos y con el propósito de evaluar la arquitectura, se seguirán las recomendaciones de Harrison y Avgeriou (2011). Primero, se identificarán los requisitos funcionales y no funcionales junto con los casos de uso relevantes para la aplicación. Luego, mediante el análisis de escenarios de atributos de calidad, se definirán las tácticas y patrones de diseño que mejor se adapten a la arquitectura. Este enfoque permitirá desarrollar una arquitectura robusta y alineada con las necesidades del proyecto.

Finalmente, para presentar la arquitectura propuesta, de manera clara y efectiva, se utiliza el modelo C4, descrito en Vivas et al. (2013). Este modelo proporciona una estructura visual que muestra los componentes, relaciones y flujos de información de la arquitectura.

8.1. Requisitos funcionales y no funcionales

Las soluciones software están diseñadas para satisfacer diferentes objetivos de negocio, lo que implica que deben abordar la variabilidad en cuanto a funcionalidad y atributos de calidad (Allian y otros, 2019). A continuación, se enumeran los requerimientos funcionales y no funcionales con el fin de limitar esta variabilidad de acuerdo con el alcance del proyecto. Además, se tienen en cuenta funcionalidades con las que ya cuentan las herramientas No-Code para la construcción de formularios analizados en el estado del arte.

Tabla 6

Requisitos Funcionales

ID	Requerimiento
Creación del formulario	
R01	Descripción: El sistema debe permitir crear formularios. Criterio de aceptación: El sistema debe tener una interfaz de usuario que permita ingresar datos para crear formularios, como el nombre del formulario, las secciones y los campos necesarios.
R02	Descripción: El sistema debe tener una interfaz para seleccionar y añadir diferentes tipos de campos a un formulario. Criterio de aceptación: El sistema debe ofrecer la funcionalidad de seleccionar y agregar diferentes tipos de campos a un formulario mediante drag and drop. Los campos disponibles incluyen texto, numérico, fecha, opción múltiple y casilla de verificación.
R03	Descripción: El sistema debe permitir dividir un formulario en secciones. Criterio de aceptación: El sistema debe tener una interfaz para añadir una o más secciones a un formulario, cada una con un nombre descriptivo.
R04	Descripción: El sistema debe permitir asignar usuarios a una sección del formulario para su diligenciamiento. Criterio de aceptación: El sistema debe tener una interfaz para seleccionar y asignar usuarios a una sección del formulario.
R05	Descripción: El sistema debe permitir guardar el formulario.

	Criterio de aceptación: El sistema debe permitir guardar el formulario, siempre que se hayan completado los campos obligatorios: nombre del formulario, al menos una sección, un diligenciador y al menos un campo en cada sección.
R06	Descripción: El sistema debe permitir publicar el formulario. Criterio de aceptación: El sistema debe permitir a los usuarios publicar un formulario cuando esté completo y listo para su uso.
R07	Descripción: El sistema debe permitir finalizar y despublicar el formulario. Criterio de aceptación: El sistema debe permitir despublicar y finalizar un formulario cuando ya no se necesite. Al despublicar, el formulario deja de estar disponible para los usuarios y no se pueden realizar más diligenciamientos.
R08	Descripción: El sistema debe permitir modificar el formulario después de publicarlo; añadir o eliminar campos del formulario. Criterio de aceptación: El sistema debe permitir modificar un formulario después de su publicación. Debe incluir opciones para añadir nuevos campos y eliminar campos existentes, sin perder datos ya registrados.
R09	Descripción: El sistema debe permitir descargar información recolectada en los formularios. Criterio de aceptación: El sistema debe permitir descargar la información recolectada en los casos de un formulario en formatos como CSV, Excel o PDF, incluyendo todos los campos y respuestas.
Diligenciamiento de formulario	
R11	Descripción: El sistema debe permitir que los usuarios autorizados vean los formularios públicos que incluyen secciones asignadas a ellos. Criterios de aceptación: Los usuarios autorizados deben tener acceso para visualizar esos formularios públicos y crear o editar casos sobre ellos, con una interfaz que sea compatible con distintos dispositivos y tamaños de pantalla.
R12	Descripción: El sistema debe permitir crear y editar uno o más casos para un formulario. Un caso representa la implementación de un formulario en una instancia específica. Criterios de aceptación: El sistema debe permitir a los usuarios crear o editar uno o más casos para un formulario sobre el cual están autorizados.
R13	Descripción: El sistema debe mostrar el estado de los casos de un formulario (pendiente, en curso, finalizado).

	Criterios de aceptación: El sistema debe tener una funcionalidad para mostrar el estado de los casos de un formulario a los usuarios asignados. El estado puede ser pendiente, en curso o finalizado.
R14	Descripción: El sistema debe permitir diligenciar la información de un caso, guardar o completar el caso. Criterios de aceptación: El sistema debe proporcionar una interfaz de usuario que permita diligenciar la información de un caso. Debe existir un mecanismo para guardar los datos ingresados en un caso y marcar un caso como completado una vez que se haya diligenciado toda la información requerida.
Autenticación y enrolamiento	
R21	Descripción: El sistema debe proporcionar una funcionalidad para registrar y autenticar a los usuarios utilizando Google como método de autenticación. Criterios de aceptación: El sistema debe permitir a los usuarios registrarse e iniciar sesión con sus cuentas de Google. El proceso de registro debe ser sencillo y fluido, permitiendo a los usuarios completar la operación sin complicaciones.
R22	Descripción: El sistema debe permitir la construcción de formularios a usuarios con el rol administrador de formularios. Criterios de aceptación: Los usuarios con el rol de administrador de formularios deben tener acceso exclusivo para crear, modificar y gestionar todos los aspectos de un formulario, incluyendo estructura, campos, secciones y configuraciones.

Nota. Requisitos funcionales. Elaborado por Chinchajoa (2023)

Seguidamente, en la tabla 7, se establecen los requisitos no funcionales a tener en cuenta en el diseño de la solución.

Tabla 7

Requisitos no funcionales

ID	Requerimiento
R31	Descripción: El sistema debe presentarse al usuario como una aplicación web responsive para el diligenciamiento de los formularios. Criterios de aceptación: El sistema debe ser una aplicación web responsive, adaptada a varios dispositivos y tamaños de pantalla, como computadoras, tablets y móviles.

R32	Descripción: El sistema debe tener soporte para los navegadores Chrome, Firefox y Safari. Criterios de aceptación: El sistema debe ser totalmente compatible con las últimas versiones estables de Chrome, Firefox y Safari. Todas las funcionalidades, como la creación de formularios y el diligenciamiento, deben funcionar sin problemas en estos navegadores.
R33	Descripción: El sistema debe construirse con tecnologías de punta. Criterios de aceptación: El sistema debe usar tecnologías de última generación, actualizadas y relevantes en el momento del desarrollo. La selección de estas tecnologías debe garantizar calidad, buena documentación y una comunidad activa para soporte.
R34	Descripción: El sistema debe poder escalar y tener un buen desempeño. Criterios de aceptación: El sistema debe funcionar bien incluso con altas cargas de trabajo o picos de tráfico. Debe soportar grandes volúmenes de datos y usuarios simultáneos sin afectar significativamente el rendimiento. Las operaciones críticas deben tener tiempos de respuesta dentro de límites aceptables. Además, el sistema debe ser escalable para no interrumpir el servicio.
R35	Descripción: El sistema debe tener unos niveles de seguridad que garanticen la confidencialidad de los datos. Criterios de aceptación: El sistema debe implementar un mecanismo de autenticación para identificar y perfilar a los usuarios que ingresen a la aplicación. Debe existir un control de acceso que permita definir los permisos de cada usuario, asegurando que solo aquellos autorizados puedan acceder a las funciones o datos específicos para los cuales tienen autorización.
R37	Descripción: El sistema debe estar disponible offline. Criterios de aceptación: El sistema debe permitir a los usuarios acceder a funciones clave, como ver formularios, completar campos y guardar datos, incluso cuando no haya conexión a Internet. Los datos ingresados o modificados mientras están desconectados deben almacenarse de forma segura en el dispositivo local. Al restablecerse la conexión, el sistema debe sincronizar automáticamente estos datos con el servidor para mantener la coherencia y consistencia de la información. La interfaz de usuario debe incluir indicadores claros para mostrar cuándo el sistema está en modo offline. Además, el sistema

	debe detectar automáticamente el restablecimiento de la conexión y sincronizar los datos.

Nota. Requisitos no funcionales. Elaborado por Chinchajoa (2023)

8.2. Casos de Uso

Las tablas 8 a la 13 presentan los casos de uso que se derivan de los requisitos funcionales. Cada caso de uso se presenta con una descripción detallada de su propósito, sus pasos principales y sus precondiciones. Con esto se tendrá una visión más completa de cómo el sistema cumplirá con los requisitos para la creación y diligenciamiento de formularios.

Tabla 8

Caso de uso - crear formulario

Caos de uso	Crear formulario
Actores	Administrador de formularios: Responsable de gestionar y configurar los formularios en el sistema.
Propósito	Permitir al administrador de formularios crear un nuevo formulario en el sistema.
Caso de uso relacionado	Modificar formulario existente.
Flujo básico	1. El administrador de formularios inicia sesión en el sistema. 2. Navega hasta la sección de administración de formularios. 3. Selecciona la opción de crear un nuevo formulario. 4. Define el nombre, la descripción y otras propiedades del formulario. 5. Agrega secciones al formulario. 6. Para cada sección, define el título y la descripción. 7. Agrega campos a cada sección, definiendo su tipo, etiqueta y configuraciones adicionales. 8. Guarda el formulario creado.
Precondiciones	1. El administrador de formularios tiene acceso válido al sistema. 2. Existen los permisos necesarios para crear formularios.

	3. Se cuenta con una conexión a internet estable.

Nota. Elaborado por Chinchajoa (2023)

Tabla 9

Caso de uso - modificar formulario

Caso de uso	Modificar formulario
Actores	Administrador de formularios: Responsable de gestionar y configurar los formularios en el sistema.
Propósito	Permitir al administrador de formularios modificar un formulario existente en el sistema.
Caso de uso relacionado	Crear formulario.
Flujo básico	1. El administrador de formularios inicia sesión en el sistema. 2. Navega hasta la sección de administración de formularios. 3. Selecciona el formulario que desea modificar. 4. Efectúa cambios en las secciones existentes, como agregar, eliminar o modificar secciones. 5. Para cada sección modificada, ajusta el título, la descripción u otras propiedades según sea necesario. 6. Modifica los campos de cada sección, como agregar, eliminar o modificar campos. 7. Guarda los cambios realizados en el formulario.
Precondiciones	1. El administrador de formularios tiene acceso válido al sistema. 2. Existen los permisos necesarios para modificar formularios. 3. El formulario que se desea modificar ya está creado en el sistema. 4. Se cuenta con una conexión a internet estable.

Nota. Elaborado por Chinchajoa (2023)

Tabla 10

Casos de uso – visualizar formularios por usuario

Caos de uso	Visualizar Formularios y casos de formularios disponibles
Actores	Usuario con acceso al sistema.
Propósito	Permitir al usuario visualizar los formularios disponibles y los casos asociados a esos formularios.
Caso de uso relacionado	Diligenciar secciones asignadas de un caso
Flujo básico	1. El usuario inicia sesión en el sistema. 2. Navega hasta la sección de formularios disponibles. 3. Visualiza la lista de formularios disponibles en el sistema. 4. Selecciona un formulario para ver los casos asociados a ese formulario. 5. Visualiza la lista de casos asociados al formulario seleccionado. 6. Para cada caso, se muestra información relevante, como el estado actual del caso (pendiente, en curso, finalizado). 7. Puede hacer clic en un caso para abrirlo y ver más detalles o iniciar el proceso de diligenciamiento de secciones.
Precondiciones	1. El usuario tiene acceso válido al sistema. 2. Existen formularios registrados en el sistema. 3. Existen casos asociados a los formularios disponibles. 4. Se cuenta con una conexión a internet estable.

Nota. Elaborado por Chinchajoa (2023)

Tabla 11

Casos de uso - diligenciar sección de caso de uso

Casos de uso	Diligenciar secciones asignadas de un caso
Actores	Usuario diligenciante: Persona encargada de completar las secciones de un formulario.
Propósito	Permitir al usuario diligenciante completar las secciones asignadas de un formulario para un caso específico.
Caso de uso relacionado	Visualizar información de casos disponibles
Flujo básico	1. El usuario diligenciante inicia sesión en el sistema. 2. Navega hasta la lista de formularios asignados.

	3. Selecciona el caso correspondiente al formulario que desea diligenciar. 4. Visualiza las secciones asignadas del formulario. 5. Para cada sección, ingresa la información requerida en los campos correspondientes. 6. Guarda el progreso del diligenciamiento en cada sección o marca el caso como completado. 7. Si existen secciones adicionales asignadas posteriormente, continúa con el proceso de diligenciamiento.
Precondiciones	1. El usuario diligenciante tiene acceso válido al sistema. 2. Existen secciones de formularios asignados al usuario diligenciante. 3. Se cuenta con una conexión a internet estable.

Nota. Elaborado por Chinchajoa (2023)

Tabla 12

Casos de uso - diligenciar sección de caso de uso en modo Offline

Casos de uso	Diligenciar secciones asignadas de un caso en modo Offline
Actores	Usuario diligenciante: Persona encargada de completar las secciones de un formulario.
Propósito	Permitir al usuario diligenciante completar las secciones asignadas de un formulario para un caso específico sin acceso a internet.
Caso de uso relacionado	Visualizar información de casos disponibles
Flujo básico	1. El usuario diligenciante inicia sesión en el sistema. 2. Navega hasta la lista de formularios asignados. 3. Selecciona el caso correspondiente al formulario que desea diligenciar. 4. Visualiza las secciones asignadas del formulario. 5. Para cada sección, ingresa la información requerida en los campos correspondientes. 6. Guarda el progreso del diligenciamiento en cada sección o marca el caso como completado.

	7. Si existen secciones adicionales asignadas posteriormente, continúa con el proceso de diligenciamiento.
Precondiciones	1. El usuario diligenciante tiene acceso válido al sistema. 2. Existen secciones de formularios asignados al usuario diligenciante. 3. Se cuenta con una conexión a internet estable hasta el numeral 4 del flujo básico, es decir, el usuario debe abrir el caso antes de estar sin internet.

Nota. Elaborado por Chinchajoa (2023)

Tabla 13

Casos de uso - descargar información

Casos de uso	Descargar información recolectada
Actores	Usuario diligenciante: Persona encargada de completar las secciones de un formulario.
Propósito	Usuario con acceso al sistema.
Caso de uso relacionado	Visualizar Formularios y casos de formularios disponibles.
Flujo básico	1. El usuario inicia sesión en el sistema. 2. Navega hasta la sección de descargas o informes. 3. Selecciona los criterios de filtrado para la información que desea descargar. 4. Solicita la descarga de la información recolectada en un formato específico, como CSV, PDF u otro formato compatible. 5. El sistema genera y proporciona el archivo descargable con la información solicitada.

	6. El usuario guarda el archivo en su dispositivo local.
Precondiciones	1. El usuario tiene acceso válido al sistema. 2. Existe información recolectada en los formularios que se puede descargar. 3. Se cuenta con una conexión a internet estable.

Nota. Elaborado por Chinchajoa (2023)

8.3. Escenarios de atributos de calidad

A continuación, se examinan los escenarios de atributos de calidad que ayudan a identificar los aspectos y características relevantes a tener en cuenta para la arquitectura. Con base en los requisitos no funcionales, se considerarán los siguientes atributos de calidad: escalabilidad, seguridad y desempeño.

Tabla 14

Escenario 1

Escenario 1	
El sistema de formularios ha sido lanzado y está siendo utilizado por un número creciente de usuarios. A medida que la popularidad del sistema aumenta, se observa una mayor carga y demanda en el sistema. Se requiere que el sistema maneje una carga de 500 usuarios concurrentes de manera eficiente, sin experimentar degradación en su rendimiento.	
Atributo	Escalabilidad y desempeño
Fuente	Usuario
Estimulo	Aumento y disminución en las solicitudes
Artefacto	Aplicación
Entorno	Operación con alta y baja demanda
Respuesta	Mantener el rendimiento de los servicios con el aumento de usuarios
Medida	- Tiempo de respuesta menor a 500 ms

	- % de error menor al 1%.

Nota. Elaborado por Chinchajoa (2023)

Tabla 15

Escenario 2

Escenario 2	
El sistema de formularios presenta un aumento significativo en el número de registros almacenados en la base de datos.	
Atributo	Escalabilidad y desempeño
Fuente	Usuario
Estimulo	Crecimiento de la base de datos y aumento de las operaciones y consultas relacionadas
Artefacto	Aplicación y base de datos
Entorno	Producción, con una gran cantidad de registros almacenados en la base de datos
Respuesta	Garantizar un buen rendimiento de la aplicación con tiempo de respuesta aceptable.
Medida	Tiempo de respuesta menor a 500 ms.

Nota. Elaborado por Chinchajoa (2023)

Tabla 16

Escenario 3

Escenario 3	
El sistema de formularios debe cargar formularios complejos con un gran número de campos y secciones.	
Atributo	Escalabilidad y desempeño
Fuente	Usuario
Estimulo	Acceso a formularios con numerosos campos y secciones
Artefacto	Aplicación

Entorno	Producción, con una gran cantidad de registros almacenados en la base de datos
Respuesta	Mostrar las secciones y campos garantizando una buena experiencia al usuario
Medida	Tiempo de respuesta menor a 500 ms segundos.

Nota. Elaborado por Chinchajoa (2023)

Tabla 17

Escenario 4

Escenario 4	
El sistema de formularios cuenta con formularios que tienen diferentes permisos asignados a usuarios o roles específicos. Estos permisos determinan qué acciones pueden realizar los usuarios o roles en relación con los formularios, como ver, editar, eliminar, crear, entre otros.	
Atributo	Seguridad
Fuente	Usuario
Estimulo	Acceso y acciones de usuarios o roles en relación con los formularios
Artefacto	Aplicación
Entorno	Producción
Respuesta	Verificar y aplicar los permisos asignados a usuarios o roles en relación con los formularios, permitiendo o denegando las acciones correspondientes.
Medida	Garantizar que los permisos de acceso y acciones en los formularios sean aplicados correctamente según los usuarios o roles correspondientes.

Nota. Elaborado por Chinchajoa (2023)

8.4. Conceptos y tácticas de diseño

En esta sección se evaluarán los atributos de calidad y sus escenarios. El proyecto se basará en una serie de conceptos de diseño para definir tácticas que aseguren que el sistema cumpla con las expectativas de calidad y desempeño, al tiempo que aborda eficazmente las necesidades de los usuarios.

Abstracción y encapsulamiento. Este concepto de diseño busca encapsular las implementaciones en módulos que interpreten el dominio de la aplicación, los objetivos de negocio y el contexto de uso del software. Este enfoque eleva el nivel de abstracción del software, facilitando el desarrollo, aumentando la productividad y mejorando la calidad (Difabio, 2023).

De esta manera, la arquitectura propuesta seguirá dos enfoques fundamentales: el Diseño Dirigido por el Dominio (DDD, Domain Driven Design) y las Arquitecturas Limpias (CA, Clean Architecture).

DDD permite representar el mundo real en la arquitectura de software a través de los contextos delimitados (Evans, 2023). Estos contextos delimitados actúan como lenguaje ubicuo para ayudar con la comunicación entre técnicos y expertos en el dominio (Fowler, 2014). Con esto se espera tener un mejor análisis y detalle de los conceptos clave relacionados con la construcción de formularios dinámicos, como los casos de un formulario, su segmentación en secciones, actores, etc. Esto permite diseñar una solución más precisa y adaptada a las necesidades del dominio específico de los formularios.

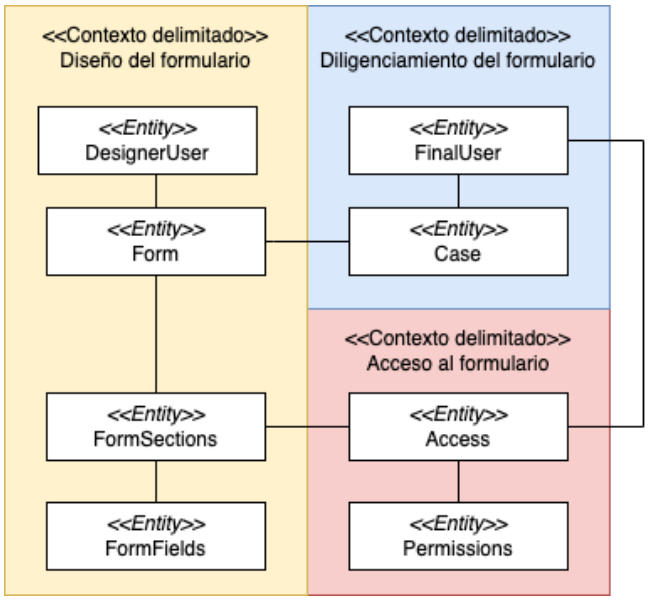
A continuación, se presentan los dominios generalizados para la gestión y diligenciamiento de formularios, junto con los contextos delimitados que aseguran los atributos de calidad necesarios. Para definir estos contextos, se identifican las siguientes entidades clave para el dominio de la aplicación:

- Formulario: Conjunto de secciones diseñado para recopilar información.

- Secciones de formulario: Agrupaciones de campos dentro del formulario.
- Campo: Es un elemento donde los usuarios ingresan datos específicos.
- Actor: Usuario que diligencia una o más sección en un formulario.
- Caso de formulario: Respuesta a un formulario proporcionada por actores.

Figura 4

Contextos delimitados



Nota. Adaptado por Chinchajoa (2023)

De esta manera, se presentan los contextos delimitados a partir de los cuales se construirá el proyecto. Aunque el dominio del proyecto no se caracteriza por una alta complejidad, la consideración de estos contextos, junto con la implementación de arquitecturas limpias, desempeñará un papel fundamental en la separación de responsabilidades. Esto permitirá un desacoplamiento efectivo, lo que a su vez facilitará la evolución aislada de cada contexto, manteniendo el dominio como el núcleo central del sistema (Difabio, 2023). En este sentido, el diseño se plantea de manera que, en caso de éxito del sistema, se pueda lograr un crecimiento sin inconvenientes.

Teniendo en cuenta estos contextos, se implementarán las siguientes tácticas de diseño: abstracción en capas para separar funcionalidades y mantener la lógica de negocio detrás de interfaces claras; una API web para ofrecer funciones de forma estandarizada, permitiendo una interacción eficiente entre componentes y autonomía para cada dominio; y el uso de interfaces para exponer funciones sin revelar detalles internos, promoviendo la modularidad y facilitando futuras actualizaciones.

Disponibilidad y escalamiento. El proyecto requiere que la solución garantice la disponibilidad de información y maneje eficientemente cualquier carga de trabajo. Para lograrlo, se implementará la táctica de diseño: Load Balancer, distribuyendo automáticamente el tráfico hacia el servidor con mayor capacidad. Además, se propone utilizar Kubernetes como orquestador de contenedores, permitiendo autoescalado, resiliencia ante fallos, gestión centralizada y monitoreo integrado. Esta combinación asegurará la disponibilidad y eficiencia del software.

Seguridad. El negocio demanda un tratamiento seguro y confiable de la información, por lo que en este proyecto se llevarán a cabo las siguientes tácticas de diseño:

- Autenticar usuarios (Resistencia a ataques): Esta táctica se enfoca en autenticar a la entidad que dice ser. Se plantea la implementación de un componente de autenticación a través del manejo de usuarios y contraseñas, permitiendo el acceso a los formularios y sus respectivos casos.
- Autorizar usuarios (Resistencia a ataques): Esta táctica se enfoca en definir los permisos para una entidad ya autenticada. Se plantea la implementación de un componente de autorización que administre de manera precisa los accesos y los permisos relacionados con los formularios y casos dentro del sistema.

Así mismo, el negocio requiere operaciones rápidas y una gestión eficiente de la demanda de información. Para cumplir con este objetivo, en el desarrollo de la solución se debe

cuenta tácticas de optimización de consultas, lo que implica utilizar índices adecuados, reducir el número de consultas y refinar la lógica detrás de ellas.

Con los conceptos y tácticas de diseño previamente discutidos, se establecen los principios que guiarán la arquitectura del sistema para lograr sus objetivos de calidad y rendimiento. Estos conceptos y tácticas servirán como base para el modelado de la arquitectura del sistema.

8.5. Modelado C4

Utilizando el modelo C4, se presenta a continuación una arquitectura de software que ofrece una visión abstracta y de alto nivel de los componentes, así como de las relaciones entre ellos. Esta arquitectura tiene como objetivo principal fomentar la reutilización y la evolución del código, tal como lo propone Vivas et al. (2013).

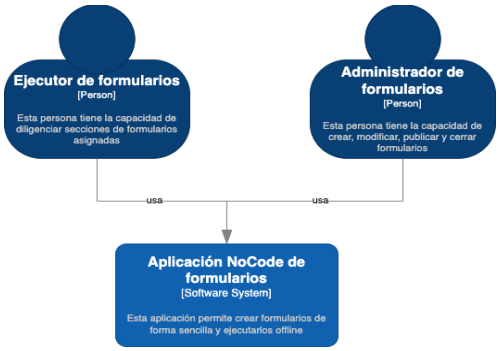
8.5.1. Diagrama de contexto.

En Colombia, diversas empresas requieren la recolección eficiente y precisa de información. Para llevar a cabo esta tarea, se utilizan comúnmente formularios que son completados por diferentes actores. Con el objetivo de simplificar este proceso, se propone el uso de una aplicación No-Code que permita a las empresas crear formularios de manera sencilla. Estos formularios pueden ser divididos en secciones para asignarlas a distintos usuarios para su ejecución. Además, la aplicación ofrece la funcionalidad de ejecutar los formularios en modo offline, lo que facilita la recolección de información en lugares donde la conexión a internet es limitada. Con estas características, se busca agilizar y optimizar el proceso de recolección de información para las empresas.

Este diagrama está destinado a stakeholders interesados en implementar este sistema en sus empresas.

Figura 5

Diagrama de formularios



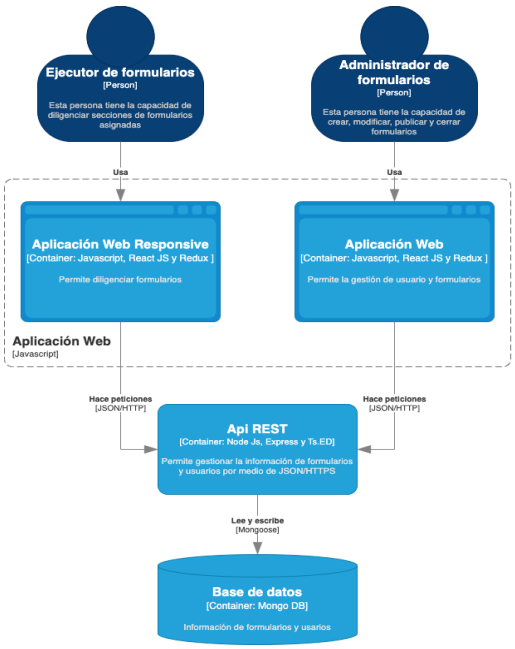
Nota. Diagrama elaborado por Chinchajoa (2023)

8.5.2. Diagrama de contenedores.

Los administradores de formularios tendrán acceso a una aplicación web dedicada para la gestión de los formularios, mientras que los usuarios responsables de completar dichos formularios podrán acceder a ellos mediante una aplicación web optimizada para dispositivos móviles, incluso en entornos offline.

Figura 6

Diagrama de contendedores



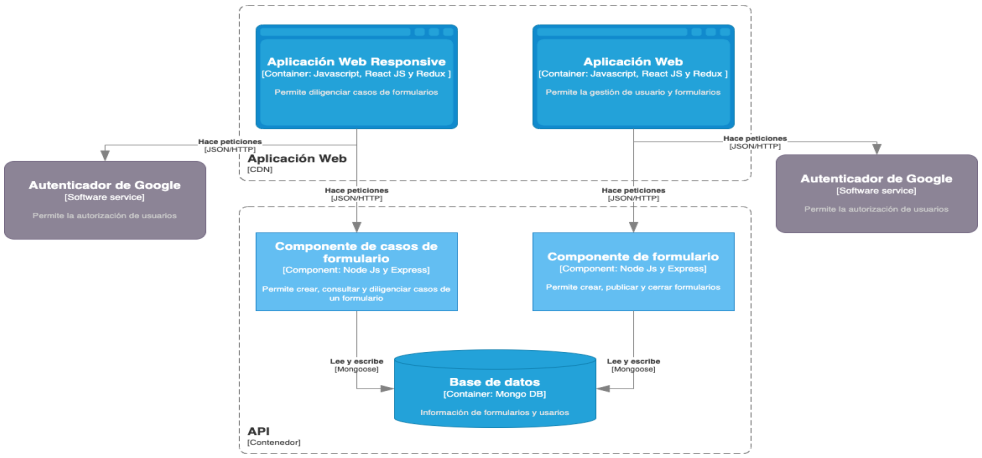
Nota. Diagrama elaborado por Chinchajoa (2023)

8.5.3. Diagrama de componentes.

La aplicación web se compone de dos subaplicaciones: una para la creación de formularios, donde los usuarios pueden crear y editar formularios, y otra para la ejecución de formularios, que incluye un componente para renderizar formularios y otro para listar casos de formularios, permitiendo a los usuarios completarlos y enviar sus respuestas. Ambas subaplicaciones se comunican mediante un API REST, que proporciona funcionalidades para la validación de formularios, la gestión de formularios, la gestión de usuarios, y la conexión a la base de datos.

Figura 7

Diagrama de contenedores



Nota. Diagrama elaborado por Chinchajoa (2023)

A continuación, se detallan los componentes de Frontend y Backend siguiendo una arquitectura Hexagonal en capas, diseñada para ofrecer varios beneficios clave, como el desacoplamiento, la sustitución, la evolución y la facilidad para hacer pruebas. Con esta arquitectura, los componentes pueden reflejar el dominio del negocio, facilitando la creación de un software que esté alineado con la visión central del negocio y optimizado para la recolección de información en campo.

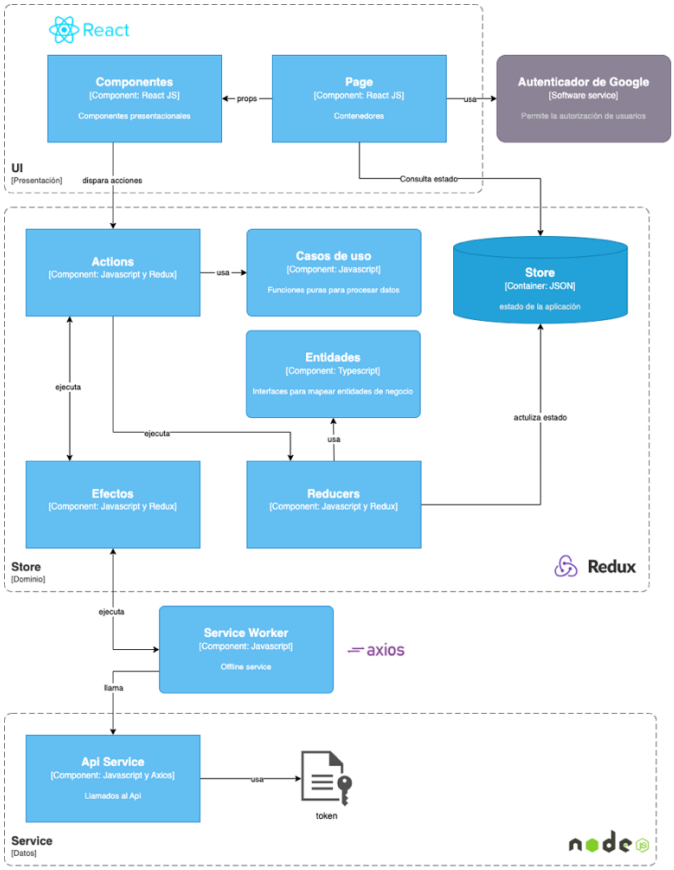
Componentes FrontEnd. Para crear una aplicación FrontEnd escalable, se utiliza una arquitectura en capas que divide las responsabilidades en tres partes: UI (Interfaz de

Usuario), Dominio e Infraestructura. La capa de UI contiene los elementos visuales y la lógica de interacción con los usuarios. La capa de Dominio se encarga de la manipulación y procesamiento de datos. La capa de Infraestructura gestiona las llamadas a la API.

El siguiente diagrama ilustra cómo se estructuran estos componentes en una aplicación construida con React, Redux y Axios.

Figura 8

Componentes para una aplicación web.



Nota. Diagrama elaborado por Chinchajoa (2023)

Paginas. Representan contenedores de React que tienen la responsabilidad de preparar los datos para su presentación en las vistas de la aplicación mediante componentes presentacionales. De acuerdo con los contextos delimitados previamente definidos, se establecerán tres páginas fundamentales: Diseño de formulario, consulta y diligenciamiento de casos de formulario y autenticación de usuarios. Cada una de estas páginas se diseñará con su

propia store, componentes y servicios, cumpliendo así con el principio de responsabilidad única, tal como lo establecen los principios SOLID.

Estado de la aplicación. Es donde se almacenan y organizan los datos necesarios para el funcionamiento de la aplicación (Redux, 2020). La store es la única fuente de verdad, lo que significa que los componentes solo obtienen información de esta fuente. Los componentes interactúan con la store a través de acciones, que además implementa los casos de uso para solicitar o modificar información. Los reducers son funciones que procesan estas acciones para actualizar el estado de la aplicación, reemplazando el estado anterior por uno nuevo. Los efectos permiten realizar tareas auxiliares, como llamadas a una API, facilitando la comunicación entre la store y otras fuentes de datos (Redux, 2020). Con este sistema, el estado de la aplicación se mantiene consistente y adaptable.

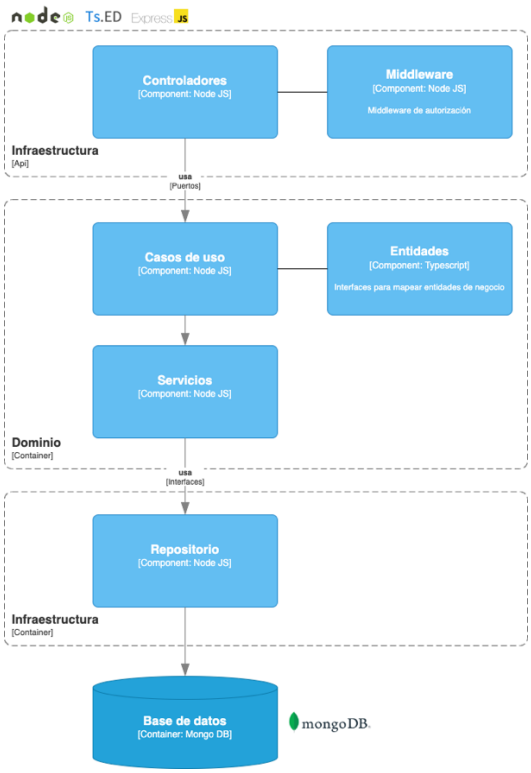
Service Worker. Es un script que se ejecuta en segundo plano en el navegador web del usuario, encargándose de tareas que no requieren interacción directa con la página web. Su función principal es administrar las solicitudes a la API, independientemente de si hay o no conexión a internet disponible. Además, se encarga de actualizar el almacenamiento local del usuario, permitiendo que, cuando se restablezca la conexión, se realicen las llamadas a los servicios expuestos en la API y se sincronicen los datos recolectados en modo offline. De esta manera, se asegura el correcto funcionamiento de la aplicación en ausencia de conexión a Internet y se garantiza la sincronización de los datos cuando la conexión sea restaurada.

Componentes backEnd. En el siguiente diseño, se aplican los principios definidos por Robert C. Martin en 2018 para las arquitecturas limpias, junto con los componentes que conforman la arquitectura Hexagonal, los cuales son: la interfaz de usuario, el núcleo de la aplicación y la infraestructura. Esto permite que la aplicación evolucione de manera flexible y se adapte a los cambios tecnológicos y requisitos del negocio. Además, al seguir este enfoque,

se facilita el mantenimiento y las pruebas de la aplicación, lo que contribuye a un producto sólido y confiable.

Figura 9

Componentes backend



Nota. Diagrama elaborado por Chinchajoa (2023)

En el diagrama anterior, cada capa opera de forma independiente y se comunican a través de puertos e interfaces.

Desde la perspectiva propuesta por Kim & Neil (2023), el dominio de la aplicación es el escenario fundamental para la implementación de la lógica de negocio. En el marco de este proyecto, se llevará a cabo la definición de entidades, casos de uso y servicios para gestionar y completar formularios dinámicos.

Entidades. Desempeñan un papel crucial al encapsular las reglas de negocio y representar los objetos fundamentales de la aplicación. En este proyecto, las entidades comprenden un conjunto de estructuras de datos que representan a los usuarios, los formularios

y los casos de formulario según los contextos delimitados. Además, se debe incluir una entidad específica para gestionar los campos de los formularios y otra para manejar la relación de formularios y casos por usuario, conforme al modelo de datos definido.

Servicios. tienen la responsabilidad de ejecutar la lógica de negocio definida por las entidades.

Casos de Uso. Esta capa mapea los objetos del dominio y los expone a los controladores, sin afectar directamente a las entidades del dominio ni verse influenciada por la base de datos o la interfaz de usuario. Sin embargo, los cambios en el funcionamiento general de la aplicación pueden afectar los casos de uso y, por ende, esta capa. Por ello, es fundamental gestionar el cambio y las actualizaciones para mantener la coherencia y el correcto funcionamiento del sistema.

Controladores. De acuerdo con los contextos delimitados, se implementarán principalmente tres controladores. El primero se encargará de validar las credenciales de los usuarios y de crear sesiones. El segundo se centrará en la gestión de formularios, mientras que el tercero supervisará la gestión de los casos de uso de los formularios. Además, habrá un controlador para consultar los campos de los formularios y otro para la consulta de los formularios y sus casos por usuario.

Puertos. Permiten la comunicación entre el dominio y el entorno externo, facilitando la interacción con los controladores, por ejemplo. Su función principal es invertir las dependencias, evitando que los componentes de nivel superior, como el dominio, dependan de los de nivel inferior, asegurando la compatibilidad y conveniencia en ambos contextos.

Repositorio. Es un adaptador que ejecuta las tareas solicitadas por el dominio para operar con la base de datos, como crear, eliminar o modificar datos.

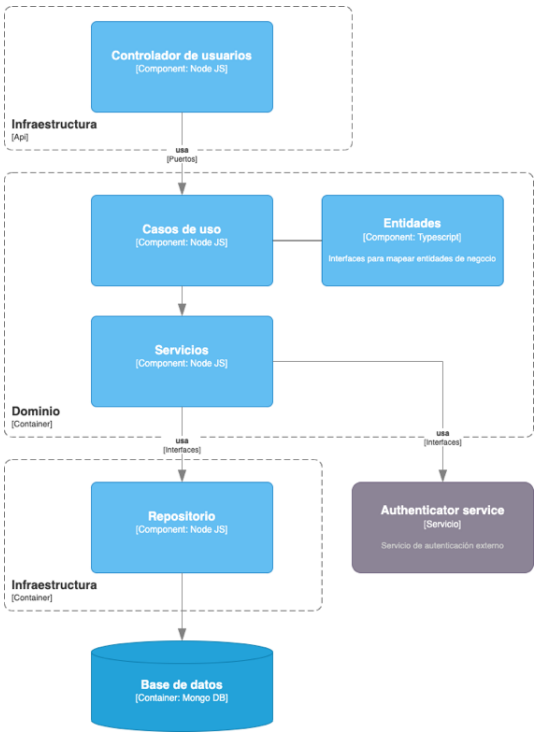
Middleware de autorización. Esta capa valida los niveles de acceso que los diferentes usuarios tienen a la información en las entidades. Se basa en políticas de autorización definidas

en la capa de Dominio para proteger el acceso a servicios y modelos del Dominio. Se promueve el uso de políticas de acceso basadas en atributos y acciones por recurso, evaluando si un usuario puede realizar una acción específica sobre un recurso. Estas políticas son un conjunto de reglas que responden a la pregunta "¿Puede el usuario A realizar la acción B en el recurso C?". El resultado de una política es una respuesta de "permitir" o "denegar" una acción sobre un recurso.

La autorización está estrechamente relacionada con la sesión de usuario que se crea durante el proceso de autenticación. A continuación, se detallan los componentes cuya finalidad es verificar la validez del token enviado desde el cliente. En el caso de que el token sea válido, se procederá a establecer una sesión con los permisos y accesos correspondientes. En el siguiente diagrama de componentes, se presenta una solución que utiliza un servicio de autenticación externo, como Google Sign-In.

Figura 10

Diagrama de autenticación de usuarios



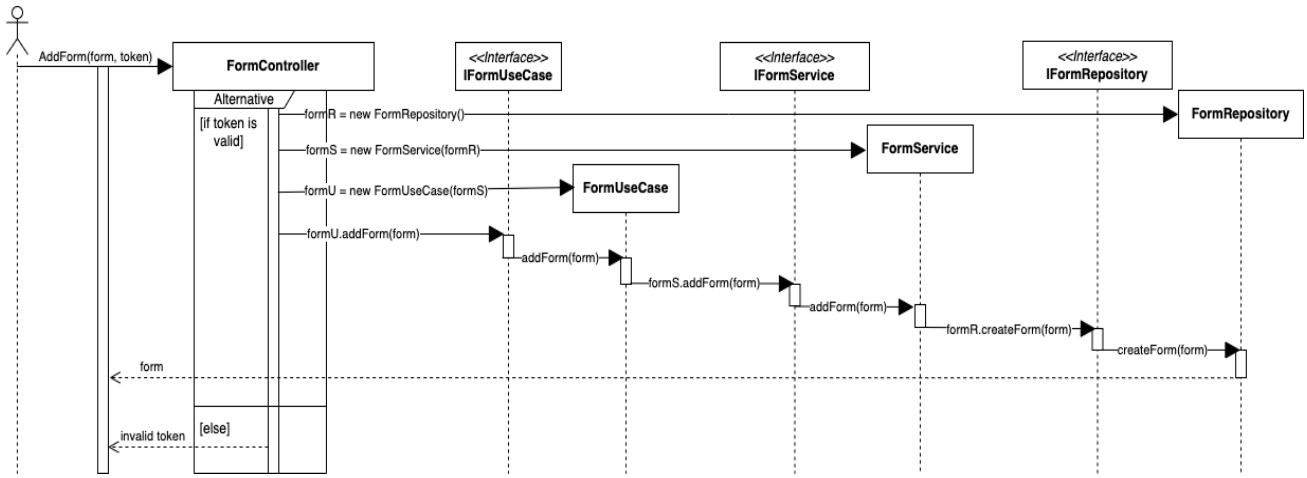
Nota. Diagrama elaborado por Chinchajoa (2023)

8.5.4. Diagramas de secuencia.

Con el fin de ofrecer una descripción más detallada de los flujos principales del sistema, a continuación, se presentan los diagramas de secuencia para la creación y actualización de formularios y sus respectivos casos.

Figura 11

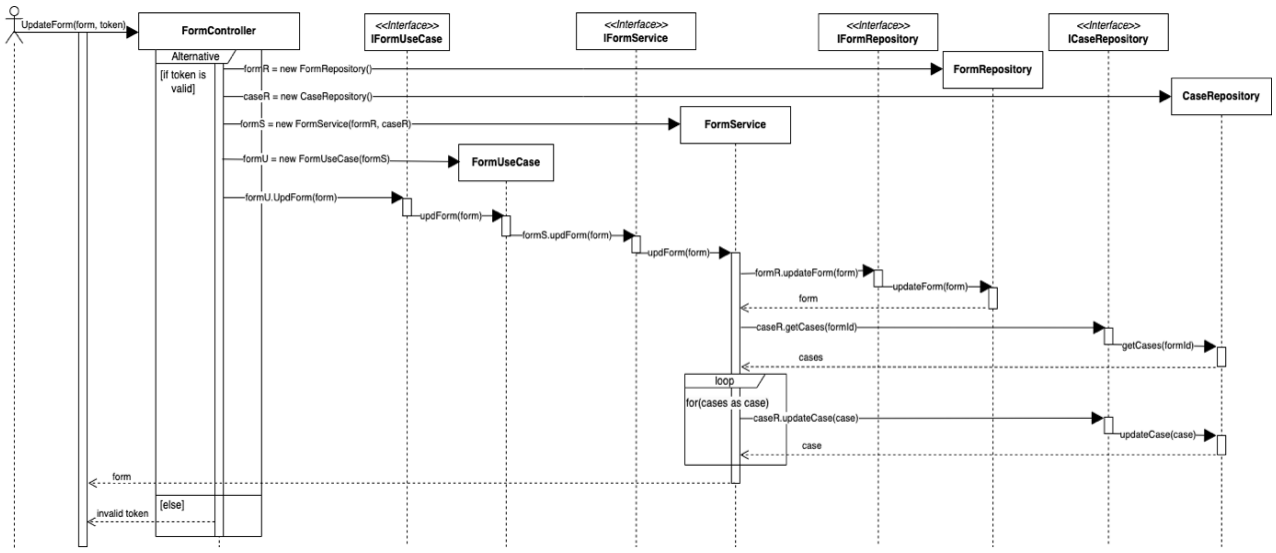
Creación de formularios



Nota. Creación de formularios iniciales Chinchajoa (2023)

Figura 12

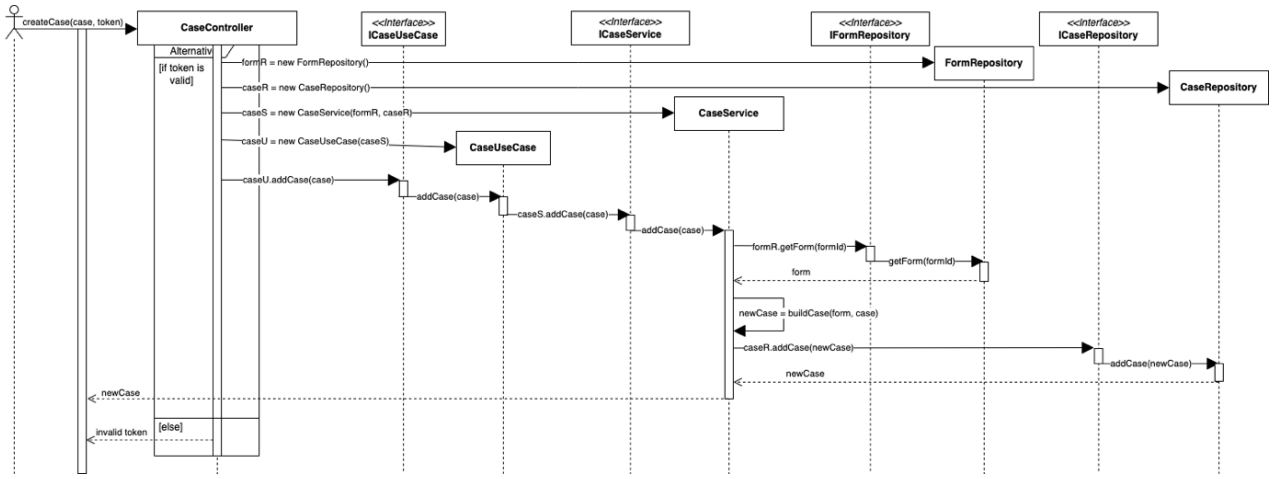
Actualización de formularios



Nota. Actualización de formularios Chinchajoa (2023)

Figura 13

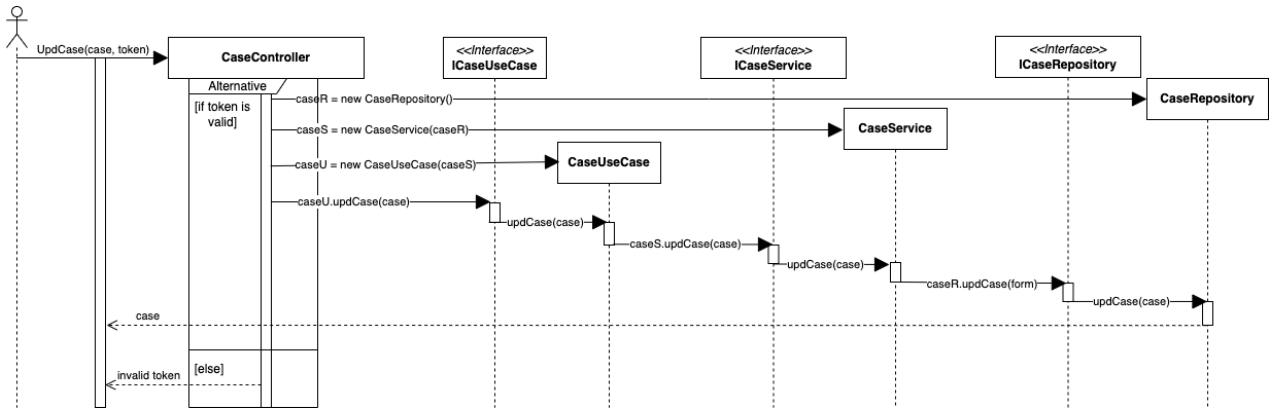
Creación de Casos



Nota. Creación de casos Chinchajoa (2023)

Figura 14

Actualización de Casos



Nota. Actualización de casos Chinchajoa (2023)

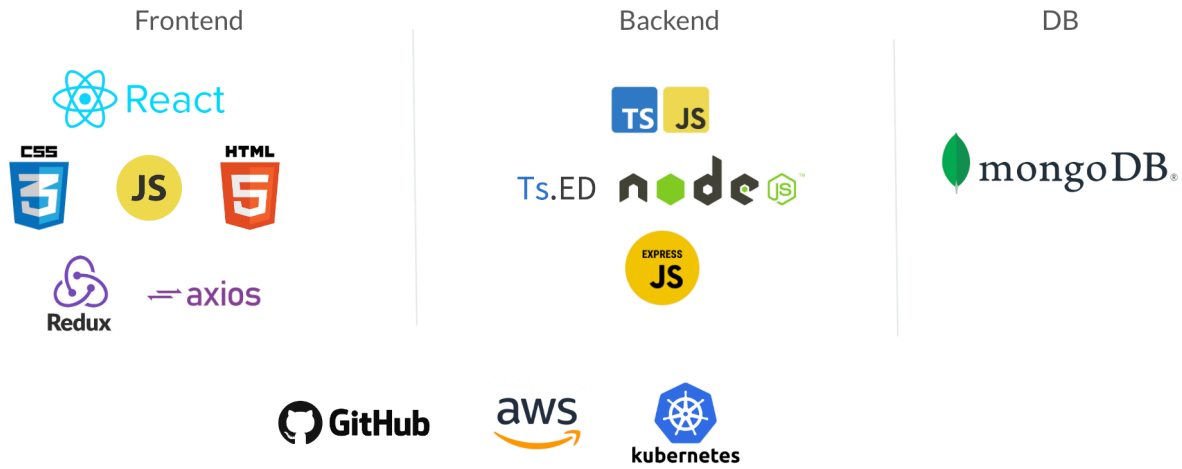
8.6. Tecnologías de desarrollo

[11]Para la implementación de los componentes definidos en la sección anterior, a continuación, se presentan algunas tecnologías que pueden ser utilizadas para desarrollar la solución propuesta:

[11] En la sección de "Tecnologías sugeridas para implementar la aplicación" de los anexos, se proporciona información más detallada sobre cada una.

Figura 15

Tecnologías de desarrollo



Nota. Diagrama elaborado por Chinchajoa (2023)

8.7. Diagrama de despliegue

Además de considerar las tecnologías, es fundamental evaluar la viabilidad de implementar tanto la aplicación como la base de datos en la nube. El diagrama de despliegue que se presenta a continuación utiliza servicios de AWS para crear una arquitectura escalable y segura. El diseño incluye máquinas EC2 que alojan la API y que pueden escalar automáticamente con Kubernetes, dentro de una red privada conectada a un Load Balancer a través de una Gateway. También se incluye una instancia de MongoDB, a la que solo tendrán acceso las máquinas EC2 dentro de la misma red privada.

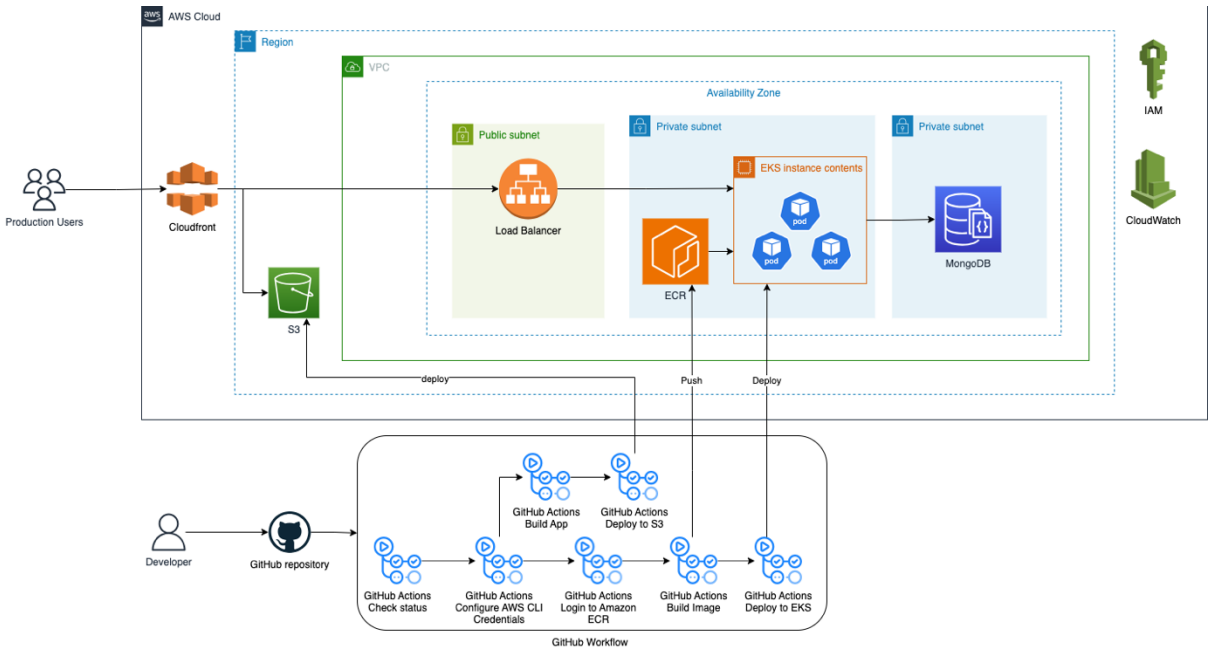
La aplicación web estará alojada en un bucket de S3 y se utilizará CloudFront para distribuir y acelerar la entrega del contenido.

La gestión de roles y permisos a la infraestructura puede hacerse con IAM. Y se propone a CloudWatch para el registro de todas las transacciones y operaciones del sistema, permitiendo un monitoreo completo y eficiente.

Con esta arquitectura, se busca lograr un entorno altamente disponible, seguro y con capacidad de crecimiento automático para brindar una experiencia óptima a los usuarios y garantizar la confiabilidad y estabilidad del sistema.

Figura 16

Diagrama de despliegue



Nota. Diagrama elaborado por Chinchajoa (2023)

Con la solución de software propuesta en este capítulo, se procede a la construcción de un prototipo con el cual se espera evaluar los conceptos y diseños presentados anteriormente. El enfoque del prototipo abordará diversos aspectos clave, tales como la utilización de esquemas de datos semiestructurados, la integración de una base de datos NoSQL y la implementación de la arquitectura de software propuesta. A través de este prototipo, se validará la viabilidad técnica de la solución, y la funcionalidad para la gestión de formularios.

Capitulo 9: Aplicación web No-Code que implemente y valide la solución digital

En este capítulo, se lleva a cabo la evaluación de una aplicación web que implementa la arquitectura previamente definida [12]. La aplicación tiene como objetivo validar la solución de software, para medir la experiencia de los usuarios durante la creación y el diligenciamiento de formularios de acuerdo con los casos de uso definidos en la capitulo 8. Además, se realizan pruebas de desempeño para garantizar que la aplicación pueda manejar un crecimiento sostenido en el número de usuarios y la cantidad de transacciones sin degradar su rendimiento de acuerdo con los escenarios de calidad definidos en el capítulo 8.

9.1. Diseños WEB

Para garantizar una experiencia óptima para los usuarios, se han desarrollado tres diseños clave para la aplicación. El primero facilita la creación de formularios mediante la función de "Drag and Drop". El segundo muestra una lista de formularios y sus casos asociados, junto con sus respectivos estados. El tercer diseño es una pantalla que permite a los usuarios diligenciar los formularios.[13]

9.2. Evaluación de la aplicación

Para validar las funcionalidades y garantizar que se cumplan los casos de uso y atributos de calidad, se ha definido una estrategia de pruebas basada en la taxonomía de Donal Firesmith, que clasifica las pruebas en 16 categorías. De estas, se han seleccionado las que mejor se adaptan al contexto del proyecto: pruebas de usabilidad, pruebas de estrés y rendimiento.

9.2.1. Pruebas de usabilidad.

[12] En la sección “Enlace del repositorio del proyecto” de los anexos se encuentra el enlace de acceso al repositorio de la aplicación desarrollada.
[13] Estos diseños se encuentran disponibles en la sección “Diseños Web” de los anexos.

La aplicación se implementa según los diseños definidos y se evalúa mediante pruebas de usabilidad en un entorno controlado para obtener retroalimentación de los usuarios. Estas pruebas se llevan a cabo con 16 participantes, siguiendo la recomendación de Loranger y Nielsen (2006), que sugiere que cinco participantes pueden cubrir entre el 55% y el 95% de los errores en un prototipo. Durante las pruebas, a los participantes se les pedirá que realicen tareas específicas relacionadas con los casos de uso de la aplicación, mientras se registran observaciones y comentarios. Los datos obtenidos ayudarán a identificar áreas de mejora y posibles obstáculos, además de validar la eficacia de la aplicación en función de los casos de uso.

Diseño de las pruebas. Para el diseño de la prueba se siguen los pasos sugeridos en el workshop de pruebas de usabilidad (Blog.ida, 2020):

- Objetivos de la prueba: Medir la experiencia de usuario de acuerdo con los casos de uso definidos: crear formulario, modificar formulario, visualizar formularios y casos de formularios, diligenciar secciones asignadas de un caso, diligenciar secciones asignadas de un caso en modo offline y descargar información recolectada

Participantes: Para llevar a cabo la prueba, se ha seleccionado un grupo de 16 usuarios con la necesidad de crear formularios para la ejecución de un proceso que involucra la colaboración de múltiples actores.[14]

Guion de la prueba: El guion de las pruebas se estructura en siete etapas, durante las cuales se evalúan los seis casos de uso de la aplicación. En la etapa 7, se llevará a cabo un cuestionario para medir la experiencia del usuario en las pruebas.[15]

Preguntas de retroalimentación: al final de la prueba, se presenta un conjunto de 32 preguntas para evaluar el desempeño de la aplicación y recopilar comentarios sobre la

[14] En la sección “Usuarios de pruebas de usabilidad” de los anexos se recopila información relevante sobre los usuarios participantes en las pruebas.
[15] En la sección “Guion de pruebas de usabilidad” de los anexos se encuentra el detalle completo de este guion.

experiencia del usuario. Estas preguntas permiten medir el cumplimiento de las etapas y obtener sugerencias para mejorar la aplicación. Cabe destacar que este formulario de retroalimentación se creó y ejecuto usando la propia aplicación evaluada. [16]

Evaluación de los resultados de la ejecución de las pruebas. En la siguiente tabla se registra el porcentaje de completitud de cada una de las etapas ejecutadas por los usuarios, medido en una escala del 1 al 100% según el número de pasos completados en cada etapa.

Tabla 18

Porcentajes de cumplimiento de etapas por usuario

Usuarios / Etapas (# pasos)	Etapa 1 (8)	Etapa 2 (2)	Etapa 3 (5)	Etapa 4 (4)	Etapa 5 (5)	Etapa 6 (2)
U01	100%	100%	100%	100%	100%	100%
U02	100%	100%	100%	100%	100%	100%
U03	88%	100%	100%	100%	100%	100%
U04	88%	100%	100%	100%	100%	90%
U05	100%	100%	100%	100%	100%	100%
U06	100%	100%	100%	100%	80%	100%
U07	100%	100%	100%	100%	100%	100%
U08	100%	100%	100%	75%	100%	50%
U09	100%	100%	100%	100%	100%	100%
U10	100%	100%	100%	100%	100%	100%
U11	100%	100%	100%	100%	80%	100%
U12	88%	100%	100%	100%	100%	100%
U13	100%	100%	100%	100%	100%	100%

[16] En la sección “Preguntas de retroalimentación” de los anexos se encuentra el detalle de estas preguntas.

U14	100%	100%	100%	100%	100%	100%
U15	100%	100%	100%	100%	100%	100%
U16	100%	100%	100%	75%	100%	50%

Nota. Elaborado Chinchajoa (2024)

Tabla 19

Promedio de porcentaje completado para cada caso de uso

Etapa	Caso de uso	% de ejecución
1	Crear formulario	98
2	Visualizar formularios y casos	100
3	Diligenciar secciones asignadas de un caso	100
4	Modificar formulario	97
5	Diligenciar secciones asignadas de un caso Offline	98
6	Descargar información recolectada	93

Nota. Elaborado Chinchajoa (2024)

Así, se confirma que el prototipo cumple con los casos de uso establecidos, asegurando su alineación con las expectativas y necesidades de los usuarios. No obstante, se han detectado oportunidades de mejora en 4 de los 6 casos de uso. Estas mejoras serán evaluadas junto con el nivel de satisfacción de los usuarios respecto a los comentarios y sugerencias proporcionados durante la etapa de retroalimentación.

Evaluación del nivel de satisfacción. Se analizan las respuestas a las preguntas para identificar el consenso entre las valoraciones de los participantes en las pruebas. Los resultados se basan en un análisis de consistencia para medir el grado de concordancia en las respuestas positivas. Dado que participan más de dos usuarios y se usa una escala ordinal para las respuestas (con opciones como "Sí" y "No"), se aplican coeficientes para medir la

concordancia, utilizando la Tabla 20, que según Altman (1991), es la escala más utilizada para evaluar este grado. Estos índices permiten clasificar la experiencia de los usuarios como pobre, débil, moderada, buena o muy buena, según el valor obtenido.

Tabla 20

Índices de referencia

Valor del índice	Fuerza de concordancia
<0,20	Pobre
0,21 – 0,40	Débil
0,41 – 0,60	Moderada
0,61 – 0,80	Buena
0,81 – 1,00	Muy buena

Nota. Referencia de valores establecidos en (Herrera y otros, 2021)

La tabla 21 exhibe los índices de concordancia positivos calculados para cada pregunta de acuerdo con la siguiente formula:

$$\text{Índice de concordancia} = \frac{Número\ de\ respuestas\ de\ acuerdo}{Total\ de\ respuestas}$$

Total de respuestas = 16

Tabla 21

Índices de concordancia positivos

Preguntas	Índice de concordancia positiva (SI)	Nivel de experiencia
Diseño de formulario		
¿Es claro cómo añadir el nombre del formulario?	12/16 = 0.75	Buena
¿Es claro cómo asignar campos a una sección del formulario?	16/16 = 1	Muy buena
¿Es claro cómo asignar secciones a un formulario?	15/16 = 0.94	Muy buena

¿Es claro cómo asignar los diligenciadores de una sección del formulario?	13/16 = 0.81	Buena
¿Es claro cómo guardar un formulario?	11/16 = 0.69	Moderada
Lista de formularios y casos		
¿Los formularios fueron asignados correctamente?	16/16 = 1	Muy buena
¿Se entiende que es un caso de un formulario?	14/16 = 0.88	Muy buena
¿Es claro cómo crear un caso?	16/16 = 1	Muy buena
Diligenciamiento de caso de formulario		
¿Las secciones fueron asignadas correctamente?	16/16 = 1	Muy buena
¿Es claro como diligenciar y guardar la información?	16/16 = 1	Muy buena
Modificación de formulario		
¿Es claro cómo modificar un formulario existente?	16/16 = 1	Muy buena
Consulta y descarga de información recolectada		
¿Es claro cómo descargar la información recolectada?	16/16 = 1	Muy buena

Nota. Elaborado Chinchajoa (2024)

Sugerencias y comentarios:

Nivel de Satisfacción. Escala del 1 al 10: La satisfacción general tuvo un promedio de 8.5, lo que indica un alto nivel de satisfacción entre los usuarios.

Cumplimiento de Necesidades. ¿La aplicación cumplió con tus expectativas?: Todos los usuarios respondieron que sí.

Áreas Por Mejorar.

Diseño de formularios: Mejorar la claridad para añadir el nombre del formulario, hacer más intuitivo el botón para crear secciones, y agregar un módulo para usuarios no registrados. También, permitir reorganizar campos y secciones y visualizar a los diligenciadores asignados.

Lista de formularios y casos: Limitar el acceso a casos en progreso, mostrar indicador de carga al crear un caso, y ocultar acciones que no pueden realizar los usuarios no autores.

Diligenciar casos de formulario: Permitir finalizar casos, añadir validaciones a campos y cerrar el caso después de guardar los cambios.

Conocimiento de Otras Herramientas. Los usuarios no conocen otras herramientas que ofrezcan las mismas funciones.

De esta manera se constata que la aplicación satisface de manera adecuada los casos de uso definidos. No obstante, también se identifican numerosas oportunidades de mejora, especialmente en lo que respecta a la experiencia del usuario. Es importante resaltar la facilidad y la intuición con las que se puede crear un formulario de este tipo, con múltiples secciones y diversos usuarios diligenciadores. Estos hallazgos subrayan la importancia de continuar refinando la aplicación para garantizar una experiencia óptima para los usuarios finales.

9.2.2. Pruebas No funcionales

Para evaluar la abstracción y el encapsulamiento del sistema, se lleva a cabo un análisis de algunos aspectos de la implementación y los servicios proporcionados [17]. Estos aspectos nos ofrecen medidas de cohesión y acoplamiento que son indicadores de la calidad del diseño orientado a objetos (Kharbuja, 2016).

Tabla 22

Métricas

Métrica	Símbolo	Valor en el proyecto
Número de mensajes proporcionados	M(S)	9
Número de servicios	ns	5
Número de servicios consumidores	nc	2
Número de servicios dependientes	np	1
Número de servicios conectados	nc + np	3

Nota. Elaborado Chinchajoa (2024)

[17] En la sección “Servicios de la API” de los anexos se encuentra el detalle de los servicios disponibles.

Después de implementar la arquitectura de software basada en el dominio se obtienen los siguientes resultados.

Tabla 23

Resultados

Atributo	Valor
Acoplamiento	0.60
Cohesión	0.56

Nota. Elaborado por Chinchajoa (2024)

Los cálculos de los resultados obtenidos se describen a continuación:

Acoplamiento. El acoplamiento en un sistema se refiere a la interdependencia entre sus servicios, y según la ecuación de Kharbuja (2016), para este caso se calcula en 0.6. Aunque este valor no señala un acoplamiento muy bajo, se atribuye principalmente a la necesidad de actualizar o consultar formularios o casos entre los dominios. No obstante, en la implementación se empleó el principio de inversión de dependencias, permitiendo que los módulos de alto nivel se apoyen en abstracciones en lugar de depender directamente de los de bajo nivel.

$$acoplamiento = (nc+np) / ns$$

Cohesión. La cohesión, que representa la relación entre las funciones implementadas y sus propósitos en un sistema, y según la ecuación de Kharbuja (2016), para este caso se calcula en 0.56. Aunque este valor no asegura que cada servicio responda a un solo mensaje, mediante el enfoque de Diseño Guiado por el Dominio (DDD) se garantiza que cada servicio únicamente responda a mensajes del dominio al que pertenece.

$$cohesión = ns / M(s)$$

Seguridad. Con el propósito de verificar la autorización necesaria para acceder a recursos dentro de la aplicación, así como para evaluar los niveles de acceso de los usuarios a los formularios, se llevó a cabo un Pentesting. A continuación, se presentan los resultados:

1. Informe de Pentesting - Validación de Autorización y Permisos.
2. Información del Proyecto: Aplicación para la construcción de formularios DF.
3. Equipo de Pentesting: Johnny Andres Chinchajoa Taimal, Ivan ricardo Taimal Narvaez
4. Resumen: Se realizaron diferentes tipos de pruebas, como intentos de acceso no autorizado y simulaciones de ataques, para evaluar la efectividad de las medidas de autorización y permisos. Se detectaron algunas vulnerabilidades para las cuales se proporcionan recomendaciones.
5. Hallazgos:
    a. Enumeración de Usuarios y Roles: En la configuración actual de la aplicación, se ha identificado un único tipo de usuario con la capacidad de desempeñar dos roles distintos: creador de formularios y diligenciador de formularios.
    b. Políticas de Autorización: Se confirmó que solo los usuarios designados como creadores de formularios pueden modificarlos y descargar la información de los casos de uso asociados. Además, se verificó que los usuarios designados como diligenciadores de formularios pueden crear casos y visualizar las secciones que se les han asignado dentro del formulario.
    c. Pruebas de Acceso no Autorizado: Durante las pruebas de acceso no autorizado, se verificó que los niveles de autorización del API nunca permitieron el acceso a recursos no autorizados.
    d. Escalamiento de Privilegios: Se realizaron pruebas para evaluar posibles intentos de escalar privilegios mediante la manipulación de parámetros en solicitudes HTTP, y no se encontraron vulnerabilidades que permitieran a un usuario acceder a recursos restringidos o ejecutar acciones no autorizadas.
    e. Configuraciones de Seguridad: Una revisión de las configuraciones de seguridad de la infraestructura del proyecto, alojada en AWS, reveló que todas

las medidas de seguridad estaban correctamente implementadas y cumplían con las mejores prácticas recomendadas por AWS.

f. Pruebas de Inyección de Autorización: Se evaluaron los mecanismos de control de acceso al manipular datos de autorización en solicitudes HTTP, intentando acceder a recursos protegidos mediante la alteración del token de sesión. Sin embargo, todas las pruebas demostraron que el sistema respondía correctamente, denegando el acceso a recursos no autorizados. Cabe destacar que si un atacante obtiene el token valido, podría escalar privilegios mediante suplantación de identidad y acceder a funciones y recursos reservados.

g. Explotación de Vulnerabilidades Conocidas: Se ha identificado una vulnerabilidad en la aplicación por falta de validación de los campos en los formularios. Esta ausencia de validación permite la inyección de código malicioso, como scripts de JavaScript o etiquetas HTML.

6. Recomendaciones:

   a. Implementar una validación estricta de los datos ingresados en los campos de texto para prevenir la inyección de código malicioso.

   b. Realizar pruebas de seguridad regulares y actualizaciones de seguridad para identificar y corregir posibles vulnerabilidades en la aplicación.

   c. Implementar controles de acceso más granulares y revisar las políticas de autorización para garantizar que solo los usuarios autorizados tengan acceso a recursos sensibles.

   d. Realizar pruebas de penetración regulares para identificar y remediar nuevas vulnerabilidades de seguridad.

Además, En las pruebas de usabilidad se verificó el acceso al sistema mediante la autenticación con Google, con un resultado de cumplimiento del 100%. Asimismo, se pidió a

los usuarios que comprobaran si solo las personas asignadas podían acceder a las secciones y formularios, también con un cumplimiento del 100%.

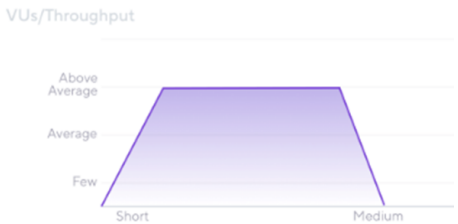
Después de realizar estas pruebas, se concluyó que la arquitectura de software cumple con los requisitos y escenarios definidos en el atributo de calidad de Seguridad. Sin embargo, es necesario fortalecer el mecanismo de autenticación mediante la implementación de un doble o triple factor de autenticación, así como mejorar la validación de la entrada de datos en los campos de los formularios. También se recomienda mejorar el control de sesiones activas desde diferentes ubicaciones.

Escalabilidad y desempeño. A continuación, se presentan los resultados de un conjunto de evaluaciones destinadas a medir tanto el rendimiento como la escalabilidad de la aplicación. En este proyecto, se llevan a cabo pruebas de carga utilizando K6, un software de código abierto diseñado para realizar de este tipo de pruebas en aplicaciones web (K6, s.f.). Una vez recopilados los datos, serán visualizados mediante Grafana, lo que facilitará una interpretación más clara y detallada de los resultados (Grafana, 2024).

Las pruebas de carga simulan el comportamiento promedio en un entorno de producción, reflejando la cantidad de usuarios simultáneos y solicitudes por segundo. Este tipo de prueba gradualmente aumenta la carga de rendimiento o los usuarios virtuales y la mantiene constante durante un período específico. Después se reduce la carga durante un breve período, como se muestra en la siguiente imagen.

Figura 17

prueba de carga



Nota. Grafana (2024)

Durante estas pruebas, se evaluarán los tiempos de respuesta y el porcentaje de errores para garantizar una experiencia de usuario optima. En cuanto a las métricas aceptables, se busca mantener una tasa de error por debajo del 1%, adaptándose a las necesidades específicas de la aplicación. Respecto a los tiempos de respuesta, se considera ideal mantenerlos por debajo de 500 milisegundos.

Tabla 24

Ficha de la prueba

Arquitectura software basada en el dominio para la creación de formularios	
Tipo de prueba	Prueba de carga
Nombre	Creación, consulta y modificación de casos de formulario
Escenario	Descripción: Diferentes usuarios necesitan crear casos de un formulario, consultarlos y modificarlos. Escenarios de atributos de calidad a validar: 1, 2 y 3. Base de datos: relacional y no relacional. Número de usuarios concurrentes: 5, 10, 15, 20, 25, 50, 100 y 500. Tiempo de ejecución de la prueba: - 0 a 5 (número de usuarios concurrentes) por 1 minuto - 5 (número de usuarios concurrentes) por 3 minutos - 5 (número de usuarios concurrentes a 0 por 1 minuto Formulario de prueba: Consta de 8 secciones, cada una con 7 campos.
Prerrequisitos	- Tener disponibles los servicios para crear (POST), consultar (GET) y actualizar (PUT) los casos de un formulario. - Formulario de prueba para crear los casos. - Se debe tener un token de acceso para usar en cada petición.
Insumos	- EKS de AWS con 1 o 2 replicas - Load Balancer de AWS - Mongo DB serverless (2GB, 2vCPUs, 10GB storage) Serverless basic free - Mysql Aurora Serverless basic

Nota. Elaborado Chinchajoa (2024)

Los resultados de las pruebas fueron:

Tabla 25

Creación de casos de formulario

Usuarios virtuales (5 min)	No SQL (# de solicitudes)				SQL (# de solicitudes)			
	post	ok	error	% error	post	ok	error	% error
5	456	454	2	0,44	337	334	3	0,89
10	906	900	6	0,66	551	513	38	6,89
15	1348	1341	7	0,52	992	990	2	0,20
20	1760	1755	5	0,28	1469	1465	4	0,27
25	2193	2182	11	0,50	1838	1835	3	0,16
50	918	3909	9	0,23	2516	2515	1	0,04
100	5346	5335	11	0,20	2767	2758	9	0,33
500	5960	5874	86	1,44	3973	3409	564	14,2

Nota. Elaborado Chinchajoa (2024)

Tabla 26

Consulta de formularios

Usuarios virtuales	No SQL (# de solicitudes)				SQL (# de solicitudes)			
	get	ok	error	% error	get	ok	error	% error
5	464	464	0	0	342	342	0	0
10	923	923	0	0	520	518	2	0,38
15	1369	1369	0	0	974	974	0	0
20	1808	1808	0	0	1533	1533	0	0
25	2253	2253	0	0	1925	1925	0	0
50	4240	4240	0	0	2656	2656	0	0
100	5978	5978	0	0	3031	3031	0	0
500	6733	6733	0	0	3679	3052	627	17,04

Nota. Elaborado Chinchajoa (2024)

Tabla 27

Actualización de casos

Usuarios virtuales	No SQL (# de solicitudes)				SQL (# de solicitudes)			
	put	ok	error	% error	put	ok	error	% error
5	466	465	1	0,21	334	334	0	0
10	923	922	1	0,11	521	520	1	0,19

15	1381	1374	7	0,51	991	988	3	0,30
20	1816	1816	0	0	1555	1555	0	0
25	2273	2263	10	0,44	1958	1958	0	0
50	4303	4301	2	0,05	2777	2772	5	0,18
100	6159	6159	0	0	3194	3186	8	0,25
500	6839	6733	106	1,55	3224	2871	353	10,95

Nota. Elaborado Chinchajoa (2024)

Tabla 28

Percentil 95

Usuarios virtuales	Percentil 95 (ms)	
	No SQL	SQL
5	308	885
10	320	495
15	319	1160
20	337	665
25	334	614
50	467	1370
100	1210	2970
500	11150	12620

Nota. Percentil 95 al crear, consultar y actualizar un caso de un formulario

[18] Con el propósito de garantizar una comparación equivalente entre la base de datos relacional y no relacional, se tomó la decisión de no ampliar la infraestructura del backend a más de dos replicas, ya que con esta capacidad fue suficiente para soportar la carga que pueden soportar las bases de datos utilizadas en la prueba. Es importante destacar que, dado que la infraestructura del backend está configurada con Kubernetes, esta puede escalarse en cualquier momento si es necesario. Esta medida se adoptó para mantener las condiciones de prueba estables y permitir una evaluación precisa de ambas bases de datos, llevándolas a su capacidad máxima sin intervenciones adicionales.

Al analizar los resultados, se verifica que ambas bases de datos cumplen con las métricas de calidad establecidas en términos de tiempo de respuesta y porcentaje de error hasta

[18] En la sección de "Resultados de pruebas no funcionales con MongoDB/MySQL Aurora" de los anexos, se proporciona información más detallada sobre las pruebas realizadas y su comportamiento.

alcanzar los 100 usuarios concurrentes, bajo las condiciones establecidas en la ficha de la prueba. Sin embargo, al aumentar la cantidad de usuarios a 500, la base de datos relacional comienza a mostrar un incremento en su porcentaje de error. Por el contrario, la base de datos no relacional demuestra una capacidad para manejar un mayor volumen de solicitudes con tiempos de respuesta óptimos y bajos porcentajes de error, incluso frente a una carga de trabajo más elevada.

Este análisis se realiza teniendo en cuenta que las pruebas se efectúan utilizando bases de datos bajo servicios gratuitos en la nube.

Capitulo 10: Trabajo futuro

En lo que respecta a la evaluación del trabajo futuro, una vez se obtienen los resultados, se establece que se puede tener en cuenta los siguientes aspectos:

- Implementar un módulo de suscripciones en la aplicación para permitir la compra y gestión de planes, como Gratis, Empresa y Full Empresa, con integración a una pasarela de pagos.
- Implementar un módulo de gestión de usuarios.
- Implementar técnicas de encriptación para proteger información sensible. Se prevé que los campos sensibles puedan ser marcados durante la creación de formularios, permitiendo que el sistema los encripte automáticamente.
- Incorporar validaciones en los campos de los formularios para garantizar la integridad de los datos ingresados.
- Implementar un mecanismo adecuando de sincronización de datos recolectados offline.
- Implementar la autenticación de doble o triple factor.
- Definir e implementar modulo para generar informes y gráficos basados en la información recopilada.
- Implementar herramientas de observabilidad, como mapas de calor y seguimiento de clics, para monitorear la interacción del usuario. Configurar tableros para visualizar el comportamiento de la aplicación y medir métricas. Establecer mecanismos de registro de eventos y análisis de logs para identificar problemas y áreas de mejora.
- Habilitar la sincronización de formularios con otros sistemas, tanto para consultar como para enviar información.
- Integrar inteligencia artificial para autocompletar los campos de los formularios analizando big data a partir de la información ya recolectada.
- Permitir a los usuarios crear y configurar campos según sus necesidades específicas.

Capitulo 11: Conclusiones

- Dynamic Forms se presenta como una solución accesible para usuarios con diversos niveles de habilidades técnicas en computación. Durante las pruebas de usabilidad, los usuarios familiarizados con herramientas no-code demostraron un uso fluido y sin dificultades. Su flexibilidad y facilidad de uso la convierten en una herramienta práctica y adecuada para una variedad de contextos, promoviendo una gestión eficiente de la información en entornos de recolección de datos. Además, su capacidad para personalizar formularios según necesidades específicas y asignar secciones a diferentes usuarios facilitará una colaboración más efectiva en el proceso de recolección de información.
- El desarrollo de Dynamic Forms destaca la importancia de contar con herramientas que funcionen en entornos con conectividad a internet limitada o inexistente. En este contexto, se validó que los Service Workers son una solución efectiva, permitiendo a los usuarios diligenciar formularios sin conexión y asegurando que los datos se almacenen localmente hasta que puedan sincronizarse con la base de datos central. Sin embargo, es fundamental garantizar la integridad y seguridad de la información, minimizando cualquier riesgo de pérdida de datos y asegurando una transición fluida entre el trabajo offline y online.
- La adopción de un modelo de datos basado en un esquema semiestructurado ha demostrado ser fundamental para gestionar la complejidad inherente en la construcción de formularios dinámicos y adaptables en el tiempo. Este enfoque simplifica la representación de datos heterogéneos y permite ajustar la estructura de los formularios sin necesidad de modificar el modelo de datos, garantizando así una mayor flexibilidad y eficiencia en el manejo de la información.

- La arquitectura de la aplicación, basada en el diseño orientado a dominios (Domain-Driven Design, DDD) y en el enfoque de arquitectura hexagonal, destaca por su gran flexibilidad. La separación clara de las preocupaciones y la modularidad facilitan la integración de nuevas funciones y la adaptación a cambios sin comprometer la estabilidad del sistema, lo que promete una evolución sostenible y la incorporación de nuevos componentes en el futuro.
- Dynamic Forms se suma a las herramientas no-code que simplifican la creación de soluciones tecnológicas sin requerir un alto conocimiento técnico. Estas herramientas están transformando el panorama digital al democratizar el acceso a soluciones tecnológicas y al impulsar la competitividad e innovación en un mercado cada vez más digitalizado. Se espera que esta herramienta accesible y flexible permita a pequeñas y medianas empresas ingresar al mundo digital de manera eficiente, mejorando la colaboración en la recolección de información y optimizando la gestión de datos. En este proyecto, se ha demostrado que los usuarios pueden construir sus soluciones con costos y tiempos mínimos utilizando una aplicación no-code, en comparación con los recursos necesarios para desarrollar una solución TI propia.

Capitulo 12: Recomendaciones

- Es fundamental considerar la naturaleza del proyecto y el tipo de datos que se pretenden almacenar, así como la posible evolución y frecuencia de cambios en su estructura para definir el modelo de datos. Este análisis permitirá tomar una decisión informada sobre si optar por una base de datos relacional o no relacional, lo que influirá significativamente en la eficiencia operativa y la escalabilidad del sistema.
- Aunque se logró diligenciar los formularios con el Service Worker en condiciones sin internet, es crucial abordar adecuadamente la sincronización de los datos recolectados offline para evitar problemas de versionamiento y asegurar la integridad de la información recolectada.
- Se recomienda analizar la utilidad de las metodologías, principios y arquitecturas, independientemente de la complejidad del proyecto. Cada proyecto debe incluir una evaluación de cómo estos enfoques pueden adaptarse y aplicarse de manera efectiva para abordar los desafíos específicos. Esta evaluación puede revelar oportunidades para optimizar la arquitectura y mejorar la capacidad del sistema para evolucionar y adaptarse a futuros requerimientos y demandas del usuario.

Referencias

Aguilar, L. J. (2016). *Aguilar, L. J. (2016). Big Data, Análisis de grandes volúmenes de datos en organizaciones.* Alfaomega Grupo Editor.

Aguilar-Borrero, P. (2022). *La generación de ventaja competitiva en PYMES a través de big data y la filosofía de gestión data-driven.* Sevilla: Universidad de Sevilla.

Alexander C. Bock, U. F. (2021). Low-Code Platform. *Business and Information*, 9.

Allian, P., Sena, B., & Nakagama, E. (2019). Evaluating variability at the software architecture level: an overview. *SAC*, https://dl.acm.org/doi/10.1145/3297280.3297511. https://doi.org/https://dl.acm.org/doi/10.1145/3297280.3297511

Alonso, V. (2021). *La arquitectura RDBMS-only : una arquitectura database-centric para aplicaciones Web (Tesis de Grado).* Universidad de la República de Uruguay.

Angeles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., & Vrgoč, D. (2018). Foundations of Modern Query Languages for Graph Databases. *ACM Computing Surveys*, 1-40.

Angles, R., & Gutiérrez, C. (2008). Survey of graph database models. *ACM Computing Surveys*, 1-39.

Appian. (2022). *appian*. appian: https://appian.com/

Axios. (2020). *¿Qué es Axios?* https://axios-http.com/es/docs/intro

Bello, S. A., Oyedele, L. O., Akinade, O. O., Bilal, ,., Davila Delgado, J. M., Akanbi, L. A., & Owolabi, H. A. (2021). Cloud computing in construction industry: Use cases, benefits and challenges . *Automation in Construction*, 18.

Benymol, J., & Sajimon, A. (Julio de 2017). *IEEEXPLORE*. Exploring the merits of nosql: A study based on mongodb. In 2017 International Conference on Networks & Advances in Computational Technologies: https://ieeexplore.ieee.org/document/8076778

Bizagi. (21 de Septiembre de 2022). *Bizagi - Líder en Automatización Inteligente de Procesos*. Bizagi - Líder en Automatización Inteligente de Procesos: https://www.bizagi.com/es

Blog.ida. (2020). *Test de usabilidad: Identificando mejoras con nuestros usuarios*. https://blog.ida.cl/experiencia-de-usuario/workshop-test-de-usabilidad/

Bock, A., & Ulrich, F. (2021). Low-Code Platform. *Business and Information*, 9.

Bonitasoft. (2023). *Bonitasoft*. Bonitasoft: https://es.bonitasoft.com/

bubble. (2022). *The best way to build web apps without code*. The best way to build web apps without code: https://bubble.io/

Cambarieri, M. G., Difabio, F., & Martínez, N. G. (2023). *Implementación de una Arquitectura de Software.* Implementación de una Arquitectura de Software:

http://sedici.unlp.edu.ar/bitstream/handle/10915/115198/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

Campoverde, A., Hernández, D., & Mazón, B. (2015). Cloud computing con herramientas open-source para Internet de las cosas. *Maskana, 6*, 173-182. https://doi.org/https://publicaciones.ucuenca.edu.ec/ojs/index.php/maskana/article/view/712

Capacho, J. R., & Nieto, W. (2017). *Diseño de Bases de Datos.* Universidad del Norte. https://doi.org/978-958-741-285-5

Capdevila, G. (2015). *Replicación de bases de datos NoSQL en dispositivos móviles.* Universidad Nacional de La Plata.

Casciaro, M. (2014). *Node.js Design Patterns.* Packt Publishing. https://doi.org/9781783287314

Cedillo, R., Z. M., & Campana, E. (29 de Agosto de 2022). *Journal Scientific Investigar*. Journal Scientific Investigar: file:///C:/Users/LENOVO/Downloads/V6_3_ART_76.pdf

Cervera Quintero, J. P. (2021). Conectividad de Internet en Colombia y su relación con los Objetivos de Desarrollo Sostenible (2015-2020). *Ciencia y Poder Aéreo*, 16.

Chang, F., Jeffrey, D., Ghemawat, S., Hsieh, W. C., & Wallach, D. A. (2008). Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems*, 1-26. Googleusercontent.com: https://dl.acm.org/doi/10.1145/1365815.1365816

Charito. (3 de agosto de 2018). *Modelos de datos NoSQL.* Modelos de datos NoSQL: https://eaminds.com/2018/08/03/modelando-nosql-data-bases/

Chillón, A. H., Morales, S. F., & Molina, J. G. (2023). *Visualización de Esquemas en Bases de Datos NoSQL basadas en documentos.* Visualización de Esquemas en Bases de Datos NoSQL basadas en documentos: https://biblioteca.sistedes.es/submissions/uploaded-files/JISBD_2017_paper_71.pdf

Cisneros Caicedo Alicia Jacqueline, G. G. (2022). Técnicas e Instrumentos para la Recolección de Datos que Apoyan a la Investigación Científica en Tiempo de Pandemia. *Dominio de las Ciencias*, 21.

Cockburn, A. (2005). *alistair.cockburn.us*. alistair.cockburn.us: https://alistair.cockburn.us/hexagonal-architecture/

Coplien, J. O., & Reenskaug, T. M. (2012). The data, context and interaction paradigm. *In Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, (págs. 227-228).

Córdova Espinoza, R. F., & Cuzco Sarango, B. E. (2013). Análisis comparativo entre bases de datos relacionales con bases de datos no relacionales (Bachelor's thesis). *Universidad Politécnica Salesiana*, 88. Análisis comparativo entre bases de datos relacionales con bases de datos no relacionales: https://dspace.ups.edu.ec/handle/123456789/6977

Dayley, B. (2014). ams Teach Yourself NoSQL with MongoDB in 24 Hours. *Pearson Education*.

Difabio, F. E. (2023). *Diseño e implementación de una Arquitectura de Software guiada por el dominio.* Diseño e implementación de una Arquitectura de Software guiada por el dominio: https://rid.unrn.edu.ar/bitstream/20.500.12049/8462/1/Difabio-Federico_Ezequiel-2021.pdf

Edward, S. G., & Sabharwal, N. (2015). *Practical MongoDB.* APRESS.

Evans, E. (2023). *Domain-Driven Design Reference.* Domain-Driven Design Reference: https://www.domainlanguage.com/wp-content/uploads/2016/05/DDD_Reference_2015-03.pdf

FastField. (201+ de Junio de 2016). *FastField Mobile Forms*. FastField Mobile Forms: https://www.fastfieldforms.com/

Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., & Moon, S. (2017). *HTML 5.2.* HTML 5.2: https://www.w3.org/TR/2017/REC-html52-20171214/

Formsonfire.com. (7 de Junio de 2023). *Forms on fire.* Forms on fire: https://www.formsonfire.com/

Formularios. (2023). *Formularios de Google* . Formularios de Google : https://workspace.google.com/intl/es-419/products/forms/

Fowler, M. (15 de enero de 2014). *BoundedContext*. MartinFowler.com: https://martinfowler.com/bliki/BoundedContext.html

Franco Trujillo, A. (2022). l papel del líder, el capital humano y la cultura organizacional en el proceso de transformación digital. *Colegio de Estudios Superiores de Administración*, 85.

Fuentes, L., Troya, J. M., & Vallecill, A. (s.f.). *Desarrollo de Software Basado en Componentes*. Desarrollo de Software Basado en Componentes: http://www.lcc.uma.es/~av/Docencia/Doctorado/tema1.pdf

Galindo, E. Z. (2019). Modelos de madurez digital en pymes Caso de estudio de una pyme de Colombia. En E. Z. Galindo, *Modelos de madurez digital en pymes Caso de estudio de una pyme de Colombia* (pág. 126). Bogotá, Colombia: Universidad Nacional de Colombia.

Garcia Molina, H., Ullman, J. D., & Widom, J. (2008). *Database Systems: The Complete Book.* Pearson Education India.

GitHub. (2023). *Flujo de GitHub*. Documentación de GitHub: https://docs.github.com/es/get-started/quickstart/github-flow

GoCanvas. (12 de agosto de 2022). *GoCanvas*. GoCanvas: https://www.gocanvas.com/

Grafana. (2024). *Tipos de pruebas de carga*. Grafana: https://grafana.com/load-testing/types-of-load-testing/

Group, I. A. (2000). *IEEE recommended practice for architectural description of software-intensive systems*. IEEE recommended practice for architectural description of software-intensive systems: https://standards.ieee.org/ieee/1471/2187/

Gupta, A., Tyagi, S., Panwar, N., Sachdeva, S., & Saxena, U. (2017). NoSQL databases: Critical analysis and comparison. *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*, (págs. 293-299).

Hahn, E. (2016). *Express.js in Action.* https://doi.org/ISBN: 9781617292422

Harrison, N., & Avgeriou, P. (2011). *Pattern-Based Architecture Reviews.* Pattern-Based Architecture Reviews: https://ieeexplore.ieee.org/document/5661759

Henriksen, M. G., Englander, M., & Nordgaard, J. (2022). Methods of data collection in psychopathology: the role of semi-structured, phenomenological interviews. *Phenomenology and the Cognitive Sciences*, 9-30.

Herranz-Gómez, R. (2014). *Bases de datos NoSQL: Arquitectura y ejemplos de aplicación* . Universidad Carlos III de Madrid .

Herrera, J., Martínez , J. M., & Martinez, L. (2021). Software de sostenibilidad turística para el cumplimiento de la NTS colombiana. *Turismo y Sociedad*, 28. https://doi.org/https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3780217

Hervás Roig, D. (2021). Tecnologías Low-Code y No-Code: Un caso práctico para estudiar su potencial y limitaciones (Doctoral dissertation. *Universitat Politècnica de València*.

Hu, J., Weng, L., Gao, Z., & Yang, B. (2023). State of Health Estimation and Remaining Useful Life Prediction of Electric Vehicles Based on Real-World Driving and Charging Data. *IEEE XPLORE*, 382-394. https://ieeexplore.ieee.org/document/9870554

Huerta, N., & Hurtado, C. (2021). Implementación de formulario digital en auditoría de entrega de mercancía en tiendas de venta al detalle. *Academia Estatal de Ciencias Económico-Administrativas de los Institutos*, 178-190.

Hylton, D., Sung, S., & Xie, c. (2021). Adopting No-Code Methods to Visualize Computational Thinking. *Institute for Future Intelligence,Physics Department, Spelman College*, 6.

IEEE STANDARS ASOCIATION. (JUNIO de 2023). *I1471-2000 EEE Recommended Practice for Architectural Description for Software-Intensive Systems.* I1471-2000 IEEE Recommended Practice for Architectural Description for Software-Intensive Systems: http://standards.ieee.org/findstds/standard/1471-2000.html

Johnson, R., Hoeller, J., Arendsen, A., Risberg, T., & Colin, S. (2009). *Professional Java development with the Spring framework.* John Wiley & Sons.

Jose, B., & Abraham, S. (2017). Exploring the merits of nosql: A study based on mongodb. *2017 International Conference on Networks & Advances in Computational Technologies (NetACT)* (págs. 1-79). IEEExplore.

Jotform. (2023). *Jotform*. Jjotform: https://www.jotform.com/es/

K6. (s.f.). *La mejor experiencia de desarrollador para pruebas de carga*. Código abierto y SaaS para equipos de ingeniería: https://k6.io/

Kalid, S., Syed, A., Mohammad, A., & Halgamuge, M. N. (2017). *Big-Data NoSQL Databases: A Comparison and Analysis of "Big-Table", "DynamoDB", and "Cassandra".* Big-Data NoSQL Databases: A Comparison and Analysis of "Big-Table", "DynamoDB", and "Cassandra": https://ieeexplore-ieee-org.bdbib.javerianacali.edu.co/stamp/stamp.jsp?tp=&arnumber=8078782

Kamlofsky, J., & Bergamini, M. L. (2014). Un Modelo para Soporte de la Investigación Asistido por Técnicas de Visión Artificial. *Congreso Nacional de Ingeniería en Sistemas de Información* (págs. 1-9). Buenos Aires: CAETI - Universidad Abierta Interamericana.

Khan, W., Ahmad, W., Luo, B., & Ahmed, E. (2019). SQL Database with physical database tuning technique and NoSQL graph database comparisons. *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)* (págs. 110-116). IEEEXPLORE.

Kharbuja, R. (2016). Designing a Business Platform using Microservices. TECHNISCHE UNIVERSITÄT MÜNCHEN Master's.

Kim, I., & Neil, H. (2023). *A research publication recommendation web service using hexagonal architecture and RAKE algorithm.* A research publication recommendation

web service using hexagonal architecture and RAKE algorithm: https://www.inhyup.com/projects/research_advisor/Report.pdf

Kumar, V. (2006). *Mobile Database Systems.* John Wiley & Sons.

Lapicki, R. S., & Terlato, A. N. (2021). Empresas ágiles: Claves para sobrevivir a la complejidad del ambiente Serie Documentos de Trabajo, No. 777. *University of CEMA,*, 33.

Le, D. M., Dang, D.-H., & Nguyen, V.-H. (2016). Domain-driven design patterns: A metadata-based approach. *IEEEXPLORE*.

Lerna. (2023). *La herramienta original para Monorrepos de JavaScript*. https://lerna.js.org/

Loranger, H., & Nielsen , J. (2006). *Priorizar la usabilidad web*. Logotipo del grupo Nielsen NormanGrupo Nielsen Norman: https://www.nngroup.com/books/prioritizing-web-usability/

Lv, T., Yan, P., & He, W. (2018). Survey on JSON Data Modelling. *Journal of Physics. Conference Series*, 17-29.

Martinelli, E. M., Farioli, M. C., & Annalisa, T. (2021). New companies' DNA: the heritage of the past industrial revolutions in digital transformation. *Management and Governance*, 28.

Maté-Jimenez, C. (2014). Big data. Un nuevo paradigma de análisis de datos. *Comillas*, 10-16. https://doi.org/https://repositorio.comillas.edu/xmlui/bitstream/handle/11531/4873/IIT-14-153A.pdf?sequence=1&isAllowed=y

Meier, A., & Kaufmann, M. (2019). SQL & NoSQL Databases Models,Languajes,Cosistency Options and Achitectures for Big Data Mangement. (págs. 201-218). Springer Vieweg.

Mendix. (2022). *Mendix Docs*. Mendix Docs: https://docs.mendix.com/

Microsoft. (15 de 03 de 2023). *Microsoft*. Microsoft: https://learn.microsoft.com/es-es/power-apps/powerapps-overview

Mightyforms. (2023). *MightyForms - online form builder and form creator*. MightyForms - online form builder and form creator: https://www.mightyforms.com/

Mohammed, C. M., & Zeebaree, S. R. (2021). Sufficient comparison among cloud computing services: IaaS, PaaS, and SaaS. *International Journal of Science and Business*, 30.

Moskal, M. (2021). NO-CODE APPLICATION DEVELOPMENT ON THE EXAMPLE OFLOGOTEC APP STUDIO PLATFORM. *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*.

Mozilla.org. (2023). *Introduction.* Introduction: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

Padget Lorchumberto, C. R. (30 de Junio de 2023). *UN SISTEMA DE PERSISTENCIA DE DATOS DE ALTO RENDIMIENTO, BAJA LATENCIA Y ALTA*. UN SISTEMA DE PERSISTENCIA DE DATOS DE ALTO RENDIMIENTO, BAJA LATENCIA Y ALTA: http://cybertesis.uach.cl/tesis/uach/2013/bmfcip123s/doc/bmfcip123s.pdf

Patil, S., Vaswani, G., & Bhatia, A. (2014). Graph Databases- An Overview. *International Journal of Computer Science and Information Technologies*, 657-660.

Pega. (2023). *Software para arrasar con la complejidad del negocio*. Software para arrasar con la complejidad del negocio: https://www.pega.com/es

Quickbase. (2023). *Quickbase.com*. Quickbase.com: https://ww.quickbase.com/.

Quincho, E. C. (2021). *Marco de trabajo SCRUM extendido con prácticas de Clean Architecture para la mantenibilidad de software.* Universidad Peruana Unión.

Redux. (2020). *Conceptos Básicos*. Redux: http://es.redux.js.org/

Rob, P., & Coronel, C. (2011). Base de datos: diseño, implementación y administración. *Cengage Learning Editores*.

Robert, C. S. (2017). *Clean architecture: a craftsman's guide to software structure and design.* Pearson prendice hall.

Sahatqija, K., Ajdari, J., Zenuni, X., & Raufi, B. (2018). *Comparison between relational and NOSQL databases*. Comparison between relational and NOSQL databases: https://ieeexplore.ieee.org/abstract/document/8400041

Salazar, J. E. (2014). *Análisis comparativo de dos bases de datos SQL y dos bases de datos no SQL.* Universidad Tecnologica de Pereira.

Sevilla Ruiz, D., Feliciano Morales, S., & García Molina, J. (Octubre de 2015). *Inferring Versioned Schemas from NoSQL*. Inferring Versioned Schemas from NoSQL: https://www.researchgate.net/profile/Diego-Sevilla/publication/299392067_Inferring_Versioned_Schemas_from_NoSQL_Databases_and_Its_Applications/links/56f3a4f008ae81582bebb914/Inferring-Versioned-Schemas-from-NoSQL-Databases-and-Its-Applications.pdf

Smith, B. (2020). JSON básico: conheça o formato de dados preferido da web. En B. Smith, *JSON básico: conheça o formato de dados preferido da web.* Novatec .

Sommerville, I. (2011). *Ingeniería de Software.* Mexico: Pearson Educación. https://gc.scalahed.com/recursos/files/r161r/w25469w/ingdelsoftwarelibro9_compressed.pdf

Stonebraker, M. R., Madden, S. R., Abadi, D. J., Harizopoulos, S., I, H. N., & Helland, P. (2018). The end of an architectural era: it's time for a complete rewrite. *Association for Computing Machinery*, 463–489.

Surveylab. (2022 de 2022). *Surveylab*. Surveylab: www.surveylab.com/.

SurveyMonkey. (2023). *SurveyMonkey by momentive*. SurveyMonkey by momentive: https://es.surveymonkey.com/welcome/sem/

SurveySparrow. (2022). *SurveySparrow*. SurveySparrow: https://surveysparrow.com/

Syed, B. A. (2017). *TypeScript Deep Dive.* TypeScript Deep Dive.

Tab, A. J., Etemad, E. J., & Rivoal, F. (2021). *CSS Snapshot 2021.* CSS Snapshot 2021: https://www.w3.org/TR/css-2021/

Ts.ED. (2016). *¿ Por qué Ts.ED ?* Marco Ts .ED para Node.js y TypeScript: https://tsed.io/

Typeform. (2023). *Typeform.com*. Typeform.com: https://admin.typeform.com/login

Vernon, V. (2013). *Implementing domain-driven design. .* Addison-Wesley.

Vivas, H. L., Cambarieri, M. G., García Martínez, N., Muñoz Abbate, H., & Petroff, M. (2013). *Un Marco de Trabajo para la Integración de Arquitecturas de Software con Metodologías Ágiles de Desarrollo.* Un Marco de Trabajo para la Integración de Arquitecturas de Software con Metodologías Ágiles de Desarrollo: https://rid.unrn.edu.ar/handle/20.500.12049/146

WaveMaker. (2014). *WaveMaker*. WaveMaker: https://www.wavemaker.com/

Anexos

Anexo 1. Conceptos Claves

Bases de datos. Según Rob & Coronel (2011) son el conjunto de datos organizados que se almacenan para cumplir con los requerimientos de información de una organización. Estos datos son recolectados, almacenados y gestionados de forma que puedan ser fácilmente accesibles, actualizables y beneficiosos tanto para las necesidades presentes como futuras de los usuarios y el negocio. Actualmente, existen diferentes tipos de bases de datos. Una de las más usadas en la recolección de información son las bases de datos relacionales.

Bases de datos relacionales. Se basa en tablas interconectadas, ofreciendo una estructura lógica y matemática para la organización y estructuración de datos. En este enfoque, las entidades o conceptos se representan mediante tablas, donde cada fila representa una instancia de esas entidades. Las relaciones entre las tablas se establecen mediante claves primarias y foráneas, lo que posibilita una integración y análisis coherentes de los datos. El modelo relacional se sustenta en el concepto de relación, en el cual las tuplas representan los registros y los atributos proporcionan información acerca de ellos. Para la manipulación de datos, el lenguaje de consulta SQL se emplea ampliamente en bases de datos relacionales como PostgreSQL, MySQL, Oracle y Microsoft SQL Server.

En la actualidad, la diversidad de tipos de información, en la recolección de información ha impulsado mejoras en las bases de datos relacionales, al tiempo que han surgido nuevas alternativas como las bases de datos no relacionales o NoSQL.

Bases de datos no relacionales o NoSQL. El término NoSQL se utiliza para referirse a bases de datos que no utilizan SQL como interfaz de consulta. Estas bases de datos se hicieron relevantes con el auge de Internet y los servicios web a gran escala en la década de 2000. A diferencia de las bases de datos relacionales, las bases de datos NoSQL se centran en la

escalabilidad distribuida y horizontal. Tienen esquemas flexibles y no siguen el modelo ACID de consistencia, en su lugar se basan en el modelo BASE. De acuerdo con Benymol & Sajimon (2017) existen 4 tipos de bases de datos NoSQL: Clave-valor, documentos, familia de columnas y bases de datos de grafos. Estos tipos de bases de datos se destacan por su alta velocidad para la manipulación de datos, gracias a que algunos enfoques utilizan almacenamiento en memoria, según Córdova Espinoza & Cuzco Sarango (2013). Ofrecen una capacidad eficiente y efectiva para capturar y gestionar datos en tiempo real, lo que facilita la adaptación dinámica de formularios. Con su enfoque flexible y escalable, estas bases de datos son una opción ideal para aplicaciones que requieren una interacción fluida y estructuras de datos adaptables. Estas bases de datos fueron analizadas y evaluadas en este proyecto, teniendo en cuenta algunos de los siguientes aspectos: Escalabilidad, consistencia, replicación, transacciones, tolerancia a fallos, integración, sincronización móvil, costo de mantenimiento y flexibilidad.

Propiedades de las Bases de datos.

Escalabilidad: Según Stonebraker et al. (2018) se refiere a la capacidad de un sistema para mantener su rendimiento óptimo a medida que crece en tamaño, datos, usuarios, características fundamentales para la creación de formularios debido al volumen de datos que serán generados por los formularios, permitiendo almacenar y gestionar estos datos sin comprometer el rendimiento y disponibilidad del sistema.

Consistencia: Según Stonebraker et al. (2018), esta característica se refiere a la manera en que los datos son actualizados y cómo se propagan los cambios a través de un sistema distribuido. En el contexto de la creación de formularios, resulta fundamental para asegurar la integridad, coherencia y un control adecuado de los datos. Proporciona una base sólida para el manejo de transacciones, el control de concurrencia y el cumplimiento de reglas. Lo que, a su vez, contribuiría a la confiabilidad y calidad de los formularios generados.

Replicación: Según Padget L. (2023), la replicación es la capacidad de almacenar múltiples copias de datos en distintas ubicaciones, asegurando que estén sincronizadas para mantener la coherencia y disponibilidad de los datos. Esta función juega un papel fundamental en la creación de formularios, al asegurar una disponibilidad constante, un rendimiento escalable, la capacidad de recuperación ante desastres y una distribución equilibrada de la carga. La replicación proporciona una base sólida y confiable para gestionar grandes cantidades de datos, garantizando que los formularios estén siempre disponibles y accesibles.

Transacciones: Según Garcia Molina et al. (2008), una transacción es un conjunto de operaciones que se ejecutan como una unidad indivisible y continua. Estas operaciones pueden incluir inserciones, actualizaciones o eliminaciones de datos. En el contexto de la creación de formularios, las transacciones garantizan que todas estas acciones se completen de manera exitosa o se deshagan en caso de error. Además, permiten que múltiples usuarios trabajen simultáneamente en los formularios sin interferir entre sí. De esta manera, las transacciones aseguran la integridad y la coherencia de los datos, contribuyendo a una gestión eficiente y segura de la información en el proceso de creación y modificación de formularios.

Tolerancia a fallos: se refiere a la capacidad de las bases de datos para mantenerse operativas y disponibles a pesar de situaciones adversas, como fallos de hardware, errores de software o interrupciones de red. En el contexto de la creación de formularios, es esencial garantizar la disponibilidad y continuidad del servicio incluso ante circunstancias imprevistas. La tolerancia a fallos asegura que la base de datos siga funcionando de manera confiable, evitando la pérdida de datos y minimizando el impacto en el flujo de trabajo. Esta característica es fundamental para ofrecer un servicio robusto y confiable a los usuarios, asegurando que puedan acceder y trabajar con los formularios sin interrupciones no deseadas.

Integración: Según Garcia Molina et al. (2008), es la capacidad de interactuar de manera efectiva con otros sistemas y tecnologías. Esto implica permitir un intercambio fluido de datos,

incluyendo la habilidad de importar y exportar información, lograr la interoperabilidad con otros sistemas y establecer conexiones y comunicación con diversas tecnologías. En el contexto de la creación de formularios, la integración es un aspecto esencial que facilita la colaboración con otros sistemas y plataformas, optimizando el flujo de datos y mejorando la eficiencia en el proceso de creación y gestión de formularios.

Sincronización móvil: Según Kumar (2006), se refiere a la capacidad de mantener los datos actualizados y coherentes entre dispositivos móviles y un servidor central, incluso en situaciones de desconexión o conectividad intermitente. Esta funcionalidad es crucial para garantizar la disponibilidad de los formularios en dispositivos móviles, mejorando la experiencia del usuario y permitiendo un trabajo eficiente y sin interrupciones, incluso en entornos con conectividad limitada.

Costo de mantenimiento: se refiere a los recursos, tiempo y esfuerzo requeridos para asegurar el adecuado funcionamiento y administración de la base de datos después de su implementación. Debido a la diversidad de modelos de datos y enfoques que ofrecen las bases de datos NoSQL. Este costo puede variar según el tipo específico de base de datos NoSQL utilizada y las necesidades del proyecto. En comparación con las bases de datos relacionales, en el contexto de la creación de formularios, el mantenimiento de bases de datos NoSQL puede ofrecer potencialmente una ventaja en cuanto a costos. Al considerar adecuadamente la elección de la base de datos NoSQL y planificar una administración eficiente, es posible reducir los costos de mantenimiento y asegurar una operación óptima y rentable para este proyecto.

Flexibilidad: se refiere a la capacidad de la base de datos para adaptarse y manejar cambios frecuentes en la estructura y los tipos de datos utilizados en los formularios. A diferencia de las bases de datos relacionales con esquemas fijos, las bases de datos NoSQL ofrecen una mayor flexibilidad al permitir que los datos sean almacenados sin un esquema predeterminado rígido.

Tipos de Bases de datos NoSQL. De acuerdo con Benymol & Sajimon (2017) existen 4 tipos de bases de datos NoSQL:

Llave-valor: Este tipo de base de datos permite al usuario realizar operaciones utilizando una clave para acceder a un valor asociado. Los almacenes de clave-valor son altamente eficientes y escalables. Ejemplos de bases de datos clave-valor incluyen Riak, Redis, Memcached y Couchbase.

Documentos: Estas bases de datos almacenan y recuperan documentos en formatos como XML, JSON o BSON según lo mencionado por Gupta et al. (2017). Los datos relacionados con un objeto específico se almacenan como una sola instancia en la base de datos. Las bases de datos de documentos son ampliamente utilizadas en el desarrollo de aplicaciones web. Algunos ejemplos de bases de datos de documentos son MongoDB, CouchDB, Terrastore, OrientDB y RavenDB. los datos se organizan en documentos individuales.

Familia de columnas: Según Hu et al. (2023), en este enfoque, los datos se agrupan en conjuntos denominados familias de columnas, donde cada familia contiene múltiples columnas que están relacionadas entre sí. Este enfoque permite el acceso eficiente a conjuntos de datos relacionados. En una base de datos de familias de columnas, los datos se almacenan de manera comprimida y optimizada para consultas eficientes en columnas individuales o subconjuntos de columnas, en lugar de recuperar toda la información de una fila. Esto permite un acceso rápido y eficiente a los datos necesarios para consultas específicas, reduciendo así la carga de lectura y mejorando el rendimiento general (Gupta et al. 2017). Ejemplos de bases de datos de familias de columnas incluyen Cassandra, HBase, Hypertable y Amazon DynamoDB.

Bases de datos de grafos: Estas bases de datos se centran en el almacenamiento de entidades y las relaciones o conexiones entre ellas. Las entidades se representan como nodos en un grafo, y las relaciones se representan como aristas. Este enfoque permite analizar y

descubrir patrones interesantes en los datos basados en las relaciones entre los nodos. Algunas bases de datos de grafos populares son Neo4j, InfiniteGraph, OrientDB y FlockDB.

Gestores de bases de datos NoSQL.

DynamoDB: Desarrollada y probada internamente por Amazon, ofrece una solución fácil y rentable para almacenar grandes volúmenes de información. Los datos se almacenan en unidades de estado sólido SSD (Solid State Drive), que proporcionan una mayor velocidad de acceso a la información en comparación con los discos duros convencionales. Gracias al uso de SSD, se logra un rendimiento óptimo, una mayor confiabilidad y un alto nivel de seguridad de los datos almacenados. Sin embargo, cabe mencionar que esta base de datos está limitada al entorno de AWS (Kalid et al.,2017). Es una opción atractiva para el almacenamiento de datos debido a su destacado rendimiento, confiabilidad y seguridad, respaldados por el uso de unidades de estado sólido. Al considerar esta opción para el proyecto, es importante tener en cuenta algunas limitaciones, como su restricción exclusiva a la plataforma AWS y la necesidad de equilibrar adecuadamente la consistencia y disponibilidad según las necesidades específicas del proyecto. Además, aunque el valor de las llaves puede aceptar números, cadenas, listas o conjuntos, DynamoDB no está diseñada para almacenar documentos complejos con estructuras anidadas (Kalid et al.,2017).

Cassandra: Es un proyecto de código abierto iniciado por Facebook que ha ganado popularidad, coincidiendo con el auge de las redes sociales. Se trata de una base de datos distribuida que almacena los datos en formato clave-valor y fue desarrollada en Java (Evans, 2023). Es una base de datos escalable, lo que implica que puede manejar un mayor número de solicitudes de los usuarios sin que se degrade su rendimiento. También es tolerante a fallos, lo que permite reemplazar nodos defectuosos en el clúster sin tiempo de inactividad (Evans, 2023). En resumen, Cassandra es una base de datos de código abierto que ha ganado popularidad debido a su distribución, escalabilidad, tolerancia a fallos y enfoque en el análisis

de datos de big data. No obstante, Cassandra no acepta documentos, lo cual podría es un aspecto importante que considerar para el desarrollo de este proyecto.

Voldemort: Es una base de datos distribuida desarrollada por LinkedIn para abordar los desafíos de escalabilidad presentes en las bases de datos relacionales. Su enfoque se basa en el almacenamiento de datos en forma de clave-valor. Esta base de datos funciona en un entorno distribuido, donde los datos se replican automáticamente en diferentes nodos o servidores. Cada nodo opera de forma independiente, lo que proporciona una mayor tolerancia a fallos y una mejor capacidad de escalabilidad. Además, Voldemort permite expandir el clúster con relativa facilidad, sin necesidad de reequilibrar todos los datos (Fowler, 2014). Una característica destacada de Voldemort es que permite almacenar datos no estructurados o semiestructurados, como documentos JSON, XML o cualquier otro formato. Esta capacidad de almacenar documentos hace que Voldemort sea una solución versátil y adecuada para aplicaciones que requieren flexibilidad en el formato de los datos. Aunque Voldemort no está diseñado para admitir consultas complejas como lo hacen algunas bases de datos NoSQL más avanzadas. Esto puede limitar la flexibilidad en la recuperación y manipulación de datos, especialmente en formularios con estructuras más complejas.

Hbase: Utiliza un modelo de almacenamiento clave-valor. Proporciona capacidad para almacenar y recuperar datos de manera aleatoria, es decir, los datos se escriben y leen de forma no secuencial. HBase es capaz de manejar diferentes tipos de datos, incluyendo aquellos que no tienen una estructura definida, semiestructurados y estructurados, siempre y cuando no sean de gran tamaño. Sin embargo, HBase no admite consultas SQL, ya que utiliza una interfaz de programación para acceder y manipular los datos. Además, HBase está diseñada para operar en un clúster de servidores, lo que significa que no puede ejecutarse en un solo servidor. Sin embargo, a medida que se agregan más servidores al clúster, HBase se escala sin problemas. Y en caso de que un servidor presente algún problema, puede ser reemplazado por otro sin

interrupciones en la operatividad del sistema (Herranz-Gómez, 2014). HBase se destaca por su enfoque en el procesamiento distribuido de datos, lo que la convierte en una opción atractiva para aplicaciones que requieren alta disponibilidad y rendimiento en entornos de big data. Es importante considerar que HBase también presenta ciertas desventajas. Por ejemplo, su configuración y administración son más complejas, especialmente al escalar y distribuir los datos en un clúster, lo cual puede implicar un mayor esfuerzo y requerir un mayor conocimiento técnico para su implementación y mantenimiento (Herranz-Gómez, 2014).

Riak: Es una base de datos distribuida que utiliza un modelo de almacenamiento clave-valor. Su diseño se basa en la tolerancia a fallos, lo que significa que tiene la capacidad de detectar y corregir errores antes de que se produzca una falla completa, asegurando así la fiabilidad del sistema. Riak utiliza JSON (JavaScript Object Notation) como formato para el intercambio de datos, lo cual facilita la interoperabilidad con otras aplicaciones (Fuentes et al., 2023). Una de las ventajas destacadas de Riak es su capacidad para manejar cargas de trabajo web intensivas, lo cual es especialmente relevante en entornos con múltiples usuarios que realizan solicitudes simultáneas, de acuerdo con Fuentes et al. (2023). Esta capacidad le permite escalar eficientemente y responder de manera rápida y confiable a las peticiones de los usuarios. Al crear formularios, Riak ofrece ventajas significativas, como la capacidad de escalar de manera eficiente y garantizar la disponibilidad de los datos, así como mantener la integridad frente a fallos. No obstante, su enfoque en el modelo clave-valor y las limitaciones en consultas complejas requieren un diseño y una consideración cuidadosa al utilizar Riak en formularios más sofisticados. Es importante evaluar si las características clave-valor de Riak son suficientes para cubrir las necesidades específicas del proyecto y si las limitaciones en consultas pueden afectar la capacidad de manejar datos más complejos o requerimientos de búsqueda avanzados para los formularios (Herranz-Gómez, 2014).

CouchBD: Permite almacenar datos de manera documental a través del formato JSON para el intercambio de datos, se compone de documentos que se almacenan como objetos, una de sus principales características es que se puede replicar y ejecutar en diversos dispositivos, así mismo es escalable y flexible para la estructura y distribución de datos de manera eficiente (Campoverde et al., 2015). La replicación de datos consiste en facilitar que se puedan duplicar en diferentes nodos, en este punto se debe considerar que no estaría disponible para consultas dinámicas. En cuanto a seguridad, existe un control de versiones que permite volver a puntos anteriores a la vez que se guardan las copias de seguridad. En resumen, CouchDB es un servidor de bases de datos documental que utiliza el formato JSON y es ampliamente utilizado por su flexibilidad y capacidad de ejecutarse en diversos dispositivos. Ofrece facilidades para la estructuración y distribución de datos, así como la realización de replicaciones. Aunque presenta limitaciones en cuanto a consultas dinámicas, destaca por su enfoque en la seguridad y la conservación de versiones anteriores de los documentos (Campoverde et al., 2015).

BaseX: Es una base de datos documental que pertenece a la categoría de bases de datos NoSQL. Permite el almacenamiento, recuperación y gestión de datos en forma de documentos. Una de sus características destacadas es su capacidad de escalamiento y su alto rendimiento. BaseX sigue una arquitectura cliente/servidor que permite operaciones simultáneas de lectura y escritura de datos. Además, cumple con el estándar ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). La base de datos es compatible con la manipulación de grandes documentos en formatos como XML, JSON y binarios. BaseX está desarrollado utilizando Java y XQuery, lo que brinda una amplia gama de funcionalidades (Johnson et al., 2009). Sin embargo, BaseX está diseñada específicamente para el almacenamiento de datos en formato XML, lo que puede presentar desafíos al manejar el esquema de datos JSON definido en el capítulo 1 de este trabajo. Debido a su enfoque en datos XML, es posible que BaseX no

proporcione un soporte nativo y eficiente para el almacenamiento y consulta de datos en formato JSON.

Mongo BD: Está orientada a documentos, y se rige por los principios del código abierto. Su importancia radica en su versatilidad, potencia y facilidad de uso, así como en su capacidad para gestionar eficientemente tanto grandes como pequeños volúmenes de datos. Se centra en las operaciones CRUD (Crear, Leer, Actualizar y Borrar), que permiten almacenar y recuperar los datos. Para este propósito, utiliza el formato JSON, aunque internamente emplea BSON, una forma binaria de JSON que ocupa menos espacio en el almacenamiento. BSON también ofrece mayor eficiencia y rapidez al convertirse en un formato de datos compatible con diferentes lenguajes de programación (Salazar, 2014). Una característica destacada de MongoDB es su capacidad para realizar consultas dinámicas, lo que significa que puede ejecutar consultas sin una planificación exhaustiva previa. Esto facilita la flexibilidad y agilidad en la exploración de los datos. MongoDB fue desarrollado en C++ y ha demostrado ser una solución confiable y escalable en diversos escenarios de aplicaciones. Su enfoque en la gestión de documentos y su compatibilidad con JSON y BSON lo convierten en una opción popular para el almacenamiento y recuperación de datos en entornos modernos (Capdevila, 2015). En resumen, MongoDB es una base de datos NoSQL orientada a documentos que destaca por su versatilidad, potencia y facilidad de uso. Ofrece capacidades de almacenamiento y consulta flexibles, utilizando formatos como JSON y BSON. Aunque difiere de las bases de datos relacionales y no cumple con las características ACID, MongoDB ha ganado popularidad gracias a su enfoque innovador y su capacidad para manejar eficientemente grandes volúmenes de datos (Fuentes, 2023), Coplien & Reenskaug(2012), Robert (2017), Vernon (2013).

Características de MongoDB de acuerdo con Benymol y Sajimon (2017)

Flexibilidad: MongoDB almacena datos en formato de documento utilizando JSON, lo que proporciona menos restricciones de esquema y se adapta a los tipos de datos locales del lenguaje de programación.

Lenguaje de consulta enriquecido: MongoDB ofrece consultas dinámicas, clasificación, índices secundarios, actualizaciones enriquecidas y agregaciones fáciles. Estas características son similares a las que se encuentran en los sistemas de gestión de bases de datos relacionales (RDBMS), lo que brinda adaptabilidad y escalabilidad.

Sharding: Esta característica permite escalar el clúster de manera lineal mediante la adición de más máquinas. Gracias al sharding, se puede mantener la eficiencia incluso cuando hay un aumento inesperado de la carga en la aplicación web.

Facilidad de uso: MongoDB es una base de datos de documentos gratuita y fácil de instalar. Además, su uso, mantenimiento y configuración son sencillos de manejar.

Alto rendimiento: MongoDB ofrece un procesamiento de consultas rápido al admitir documentos incrustados e indexación. Esto permite reducir las operaciones de entrada/salida (E/S) en el sistema de base de datos, lo que se traduce en un mejor rendimiento.

Alta disponibilidad: MongoDB permite la configuración de conjuntos de réplicas. Estos conjuntos son grupos de servidores que mantienen el mismo conjunto de datos, lo que garantiza la disponibilidad continua de los datos incluso en caso de fallas.

Compatibilidad con varios motores de almacenamiento: MongoDB utiliza el motor de almacenamiento Wired Tiger, que ofrece soporte para varios motores de almacenamiento. Además, MongoDB admite la API del motor de almacenamiento enchufable, lo que permite a terceros desarrolladores crear sus propios motores de almacenamiento.

Tanto las bases de datos NoSQL como las bases de datos relacionales ofrecen opciones diversas en términos de estructuras de datos. En este proyecto se analizarán estas estructuras para elegir la que permite construir formularios dinámicos.

Estructuras de datos. Las estructuras de datos son aquellas que permiten organizar la información de manera eficiente y diseñar la solución correcta para un determinado problema; estos pueden ser árboles, grafos, tablas, etc. Una estructura de datos permite trabajar en alto nivel de abstracción almacenando información para luego acceder a ella, modificarla y manipularla. Según (Aguilar, 2016) las estructuras de datos se pueden dividir en: Datos estructurados, no estructurados y semiestructurados.

Datos estructurados: Según Aguilar (2016) Los datos estructurados se caracterizan por tener una longitud, formato y tamaño definidos de manera precisa. Se almacenan en formatos tabulares, hojas de cálculo o bases de datos relacionales. Este tipo de datos se clasifica en dos categorías: estáticos, que incluyen arreglos, cadenas de caracteres y registros, y dinámicos, que comprenden listas, pilas, colas, árboles y archivos). Aunque los datos estructurados poseen un nivel de organización y formato establecido, también existen datos que no siguen una estructura específica y pueden presentar diferentes formatos, conocidos como datos no estructurados.

Datos semiestructurados: Representan una combinación de los datos estructurados y los datos no estructurados. A diferencia de los datos estructurados, no tienen una estructura perfectamente definida, pero sí presentan una organización en sus metadatos que describen los objetos y sus relaciones. En algunos casos, esta organización sigue convenciones aceptadas, como los formatos HTML, XML o JSON. De acuerdo con Khan et al. (2019), los esquemas de datos semiestructurados permiten almacenar datos en diferentes formatos, como objetos, llave-valor, familias de columnas y documentos. Este enfoque de almacenamiento ofrece ventajas significativas en situaciones en las que los datos pueden variar en estructura o contenido. Al no estar limitados por una estructura de tabla fija, los esquemas semiestructurados permiten adaptarse a la evolución de los datos y acomodar diferentes tipos de información de manera más flexible. En este tipo de almacenamiento, los datos se organizan en tablas o espacios de nombres, donde cada fila o elemento se identifica mediante una clave única. Según Meier &

Kaufmann (2019), estas claves pueden adoptar diferentes tipos de datos, como cadenas de caracteres, números e incluso objetos complejos. Además, los valores asociados a las claves pueden ser de cualquier tipo, como texto, números, listas, conjuntos u otros tipos de datos estructurados. Además, se destacan por su capacidad de escalabilidad y su eficiencia en términos de velocidad de lectura y escritura. De acuerdo con Gupta et al. (2017), estas estructuras permiten un acceso directo a los datos utilizando la clave correspondiente, evitando así la necesidad de realizar búsquedas o consultas complejas. Esta característica los convierte en una opción especialmente adecuada para aplicaciones que requieren un acceso rápido y un alto rendimiento, como el almacenamiento en caché, la gestión de sesiones y el análisis de registros.

Grafos: Se enfocan en el almacenamiento y consulta de datos con relaciones y conexiones entre entidades. Los datos se representan mediante nodos y relaciones, donde los nodos representan entidades individuales y las relaciones capturan las conexiones entre esos nodos. Esta estructura de datos ofrece una forma efectiva de modelar y analizar relaciones complejas entre elementos de datos, lo que facilita la recuperación y exploración de información significativa, de acuerdo con Angles et al. (2017).

Datos no estructurados: Se caracterizan por carecer de un formato específico. Estos datos se almacenan en diversos formatos, como documentos en PDF o Word, correos electrónicos, archivos multimedia de imagen, audio o video. Esta flexibilidad hace que los esquemas no estructurados sean una opción versátil para gestionar diferentes tipos de información, ya que permiten adaptarse a las necesidades específicas de cada tipo de dato. Ejemplos comunes de datos no estructurados incluyen archivos de texto, documentos de procesadores de palabras, hojas de cálculo electrónicas, imágenes, objetos, archivos de audio, blogs, mensajes de correo de voz, mensajes instantáneos, contenido web y archivos de video, entre otros. Según Aguilar (2016). Es importante destacar que existe un espectro entre los datos

estructurados y los datos no estructurados, ya que algunos datos pueden tener cierto grado de estructura, pero no cumplen completamente con un formato rígido, y se denominan datos semiestructurados.

Técnicas para el manejar datos no estructurados de acuerdo con Henriksen et al. (2022).

Anotación de datos: En lugar de depender de un esquema predefinido, en un esquema de datos no estructurados se puede realizar la anotación de información. Esto implica identificar y extraer elementos relevantes de los datos ingresados en el formulario. Por ejemplo, cuando se recopila información de texto no estructurado, se pueden aplicar técnicas de procesamiento de lenguaje natural (NLP) como el etiquetado de entidades y la extracción de información para identificar y clasificar los componentes pertinentes. Mediante estas técnicas, se pueden reconocer nombres de personas, lugares, fechas u otros elementos clave presentes en el texto, lo que permite estructurar y organizar la información obtenida. Al utilizar la anotación de datos, se facilita la comprensión y el análisis de los datos no estructurados recopilados en el formulario, lo que a su vez permite obtener información valiosa de manera más eficiente.

Uso de metadatos: En la construcción de formularios dinámicos para datos no estructurados, los metadatos desempeñan un papel fundamental. Estos brindan información adicional sobre los datos, como su tipo, formato y contexto. Al incorporar metadatos en el formulario, se puede orientar a los usuarios respecto al tipo de información esperada y cómo debe ingresarse. Asimismo, los metadatos contribuyen a la validación y normalización de los datos recopilados. Al proporcionar pautas claras y restricciones, los metadatos mejoran la calidad de los datos y facilitan su posterior procesamiento y análisis.

Aprendizaje automático y clasificación: El uso del aprendizaje automático (machine learning) resulta valioso en la creación de formularios dinámicos en entornos no estructurados. Al entrenar modelos de clasificación, es posible identificar patrones y características en los

datos ingresados, lo que permite inferir la estructura y organización de la información. Esto posibilita la generación de formularios adaptativos que se ajustan de manera inteligente a los datos no estructurados ingresados por los usuarios. Al aprovechar el poder del aprendizaje automático, se mejora la eficiencia y precisión en la captura y organización de datos, proporcionando una experiencia más intuitiva y fluida para los usuarios.

Interfaz intuitiva y adaptativa: Para gestionar datos no estructurados con diversas formas y formatos, es fundamental desarrollar una interfaz de usuario intuitiva y adaptable que facilite el ingreso eficiente de información por parte de los usuarios. La interfaz debe ser capaz de adaptarse según el tipo de dato ingresado, brindando indicaciones claras y relevantes para guiar a los usuarios en la correcta cumplimentación del formulario. Una interfaz bien diseñada garantiza una experiencia de usuario fluida y ayuda a minimizar errores o confusiones al ingresar los datos. Además, una interfaz adaptable permite una mayor flexibilidad al acomodarse a las particularidades y necesidades específicas de cada usuario o contexto de uso.

Extracción y transformación de datos: Una vez que los datos no estructurados han sido recopilados, puede ser necesario realizar un proceso adicional de extracción y transformación para convertirlos en una estructura más organizada y comprensible. Este proceso implica acciones como la normalización de los datos, la eliminación de información no relevante y la conversión de los datos a un formato estructurado que facilite su análisis y procesamiento. Al llevar a cabo esta etapa, se logra una mayor coherencia y cohesión en los datos, lo que facilita su interpretación y utilización posterior. Además, la extracción y transformación de datos permite realizar comparaciones, búsquedas y cálculos de manera más eficiente, brindando una base sólida para el análisis y la toma de decisiones.

Las estructuras de datos desempeñan un papel fundamental en el diseño del modelo de datos, el cual fue clave en este trabajo para lograr formularios altamente editables y con la capacidad de asignarse a diferentes actores.

Modelo de datos. Es una representación visual que permite comprender y definir los elementos relacionados con los datos de un sistema, incluyendo los tipos de datos y sus relaciones. También se definen las restricciones necesarias para mantener la integridad de los datos, estableciendo un conjunto de condiciones y especificando las operaciones para administrarlos. Los modelos más usados son los modelos de datos conceptuales, también conocidos como modelos de dominio, modelos de datos lógicos y modelos de datos físicos.

Elementos comunes de los modelos de datos de acuerdo con Capacho y Nieto (2017).

Estructuras de datos: Las estructuras definen cómo se organiza y almacena la información en el modelo de datos. Pueden incluir conceptos como tablas, documentos, entidades y relaciones, entre otros.

Operaciones sobre los datos: Las operaciones permiten realizar consultas y modificaciones en los datos almacenados en el modelo. Por lo general, se utilizan lenguajes de consulta y manipulación de datos específicos para interactuar con el modelo de datos.

Reglas de integridad: Estas reglas establecen las condiciones y restricciones que aseguran la consistencia y calidad de los datos en el modelo. Pueden incluir restricciones de clave primaria, reglas de validación y relaciones entre entidades, entre otros.

El análisis y diseño del modelo de datos será quien defina la estructura y organización de los datos recopilados en los formularios. En este contexto, el formato JSON se presenta como una alternativa muy viable para representar, almacenar y transmitir los datos de los formularios.

Notación de objetos JavaScript (JSON). De acuerdo con Benymol & Sajimon (2017) y Smith (2020), JSON es un formato ligero y ampliamente utilizado en aplicaciones web para el intercambio de datos entre servidores y clientes. Debido a su notación clave-valor y su capacidad para almacenar estructuras de datos complejas, se ha convertido en una elección

popular en bases de datos NoSQL. JSON ofrece una forma flexible y fácil de leer y escribir datos y es especialmente útil en entornos web. JSON es compatible con todos los tipos de datos fundamentales, como números, cadenas, valores booleanos, así como matrices y estructuras de clave-valor. Al ser una estructura flexible y dinámica, facilitaría la adición, eliminación o modificación de campos en el formulario de manera eficiente. Además, al ser un formato ampliamente utilizado cuenta con un amplio soporte y herramientas disponibles para su manipulación. Además, el formato JSON proporciona una notación estándar y ampliamente aceptada, lo que facilita el intercambio de datos entre los diferentes componentes definidos en una arquitectura de software.

Arquitectura de software. La arquitectura de software representa el nivel abstracto de un proyecto de software desde la perspectiva de negocio. En ella se definen aspectos clave como el almacenamiento de datos, las relaciones entre componentes y las capas o niveles de implementación. Además, se tienen en cuenta las restricciones relevantes para el sistema. Cada decisión relacionada con la arquitectura de software se basa en las compensaciones y beneficios que puede aportar al proyecto y es esencial comprender el "por qué" detrás de estas decisiones. El enfoque debe estar en comprender y justificar las razones que respaldan las elecciones arquitectónicas, en lugar de enfocarse exclusivamente en el "cómo" se implementarán. Existen diferentes definiciones de arquitectura de software. Por ejemplo, Fuentes et al (2000) la describe como la estructura general del sistema, incluyendo sus componentes, interacciones, patrones de diseño y restricciones. Por otro lado, el documento de IEEE Std 1471-2000 Group, define la arquitectura de software como la organización fundamental de un sistema, que abarca sus componentes, las relaciones entre ellos, el entorno y los principios que guían su diseño y evolución. En este proyecto, la arquitectura de software desempeña un papel fundamental al establecer la visión general y la estructura del sistema, incluyendo cómo se gestionan y procesan los formularios. La definición de la arquitectura se enfocó en un diseño de software

que garantice la usabilidad y experiencia del usuario al interactuar con los formularios en su creación y diligenciamiento.

Diseño de software. Es un proceso crucial que implica la creación de una representación detallada y conceptual de un sistema de software antes de su implementación. Se enfoca en definir la estructura, comportamiento y características del software, utilizando diversas técnicas y herramientas para visualizar y comunicar el diseño. Un buen diseño de software es fundamental para desarrollar sistemas de calidad que cumplan con los requisitos y objetivos establecidos. Durante el diseño, se crean representaciones de los componentes, módulos o clases del sistema, con el objetivo de proporcionar una base sólida para su implementación posterior. Es importante destacar que estos diseños pueden evolucionar y adaptarse a medida que surjan necesidades operativas o comerciales, según lo mencionado por Cedillo et al. (2022). En el caso específico del diseño de formularios, es esencial considerar tanto los aspectos técnicos como los relacionados con el dominio del negocio. Esto implica comprender las necesidades y requisitos específicos del formulario, así como los procesos y reglas de negocio asociados. Un enfoque llamado Diseño Dirigido por el Dominio busca garantizar que el diseño sea eficiente y cumpla con las necesidades del negocio, considerando tanto los aspectos técnicos como los del dominio.

El Diseño Dirigido por el Dominio (DDD). Según Evans (2023), es un enfoque para abordar proyectos de software en dominios complejos, reconociendo que la complejidad principal de muchas aplicaciones radica en las reglas de negocio específicas del dominio. Para enfrentar estos desafíos desde el principio, se requiere abordar sistemáticamente la definición de contextos delimitados, representando los límites dentro de los cuales cada modelo de dominio existe y opera.

Contextos delimitados. Según Le et al. (2016), establecer contextos delimitados promueve la colaboración entre expertos en el dominio y desarrolladores, facilitando la

exploración y refinamiento del modelo de negocio. Esto se basa en dos conceptos clave: el lenguaje ubicuo, que garantiza que el modelo refleje con precisión las necesidades del negocio, y los contextos delimitados, que definen los límites dentro de los cuales opera cada modelo de dominio (Fowler, 2014; Difabio, 2023).

Después de la fase estratégica, comienza el proceso táctico, donde se seleccionan patrones de arquitectura y tecnologías para diseñar la solución, centrándose en establecer con mayor precisión los modelos de dominio y la aplicación de patrones tácticos dentro de contextos específicos, lo que ayuda a identificar los límites de los servicios en la aplicación y mejora la estructura del sistema. En este punto, las arquitecturas limpias se vuelven relevantes para desacoplar los límites identificados con DDD, especialmente en proyectos que buscan una interacción ordenada y desacoplada con los componentes necesarios para la construcción de formularios, promoviendo la separación de responsabilidades y la aplicación de la regla de dependencia para mantener una estructura clara y desacoplada en el código fuente (Fowler, 2014).

Arquitectura Limpia (Clean Architecture). En las últimas décadas, surgieron diferentes ideas sobre las arquitecturas de los sistemas, como la Arquitectura Hexagonal (AH, también conocida como Puertos y Adaptadores), desarrollada por Cockburn (2005) y Datos, Contexto e Interacción, (DCI, por sus siglas en inglés Data, Context and Interaction) de Coplien & Reenskaug (2012) entre otras. Aunque estas arquitecturas varían un poco en sus detalles, son muy similares. Robert (2017) señala que este tipo de arquitecturas produce sistemas que tienen las siguientes características:

Independiente de los frameworks: La arquitectura no depende de la existencia de una librería o biblioteca de software.

Testeable: Las reglas de negocio pueden ser probadas sin la interfaz de usuario, base de datos, servidor web o cualquier otro elemento externo.

Independiente de la Interfaz de Usuario: Se puede cambiar fácilmente, sin cambiar el resto del sistema. Una interfaz de usuario web podría ser reemplazada por una consola, por ejemplo, sin cambiar las reglas del negocio.

Independiente de la base de datos: Dado que las reglas de negocio no están ligadas a la base de datos, es posible cambiar el motor de base de datos.

Independiente de cualquier agente externo: Las reglas de negocio no conocen en absoluto sobre las interfaces con el mundo exterior.

Arquitectura hexagonal. En el contexto de DDD, Vernon (2013) presentó la arquitectura hexagonal como un patrón estructural que enfatiza la clara definición de las entradas y salidas en los límites del sistema. Este enfoque, ilustrado metafóricamente como un hexágono, permite intercambiar componentes externos sin modificar el núcleo de la aplicación, manteniéndolo aislado. El interior del hexágono, que incluye el modelo de dominio y la lógica de negocio, se mantiene independiente de los servicios de aplicación y adaptadores que lo rodean, conforme al principio de inversión de dependencia. Esto proporciona flexibilidad para reemplazar o modificar decisiones tecnológicas a través del desarrollo de nuevos adaptadores (Cockburn, 2005). La Arquitectura Hexagonal prioriza la separación de la lógica de negocio de las preocupaciones tecnológicas, asegurando una comunicación a través de interfaces y puertos.

Características de la arquitectura hexagonal.

Desacoplamiento: Las capas están diseñadas de forma unidireccional, lo que permite controlar las dependencias entre ellas. Esto se traduce en un bajo acoplamiento y una alta cohesión entre los componentes del sistema.

Substitución: La arquitectura Hexagonal permite la sustitución transparente de las capas. Esto significa que se puede reemplazar una capa por otra con relativa facilidad, sin afectar el funcionamiento global del sistema.

Evolución: La arquitectura está diseñada para escalar y adaptarse a medida que surjan nuevos requerimientos. Además, el código se mantiene de manera sencilla, lo que facilita su evolución y extensibilidad.

Facilidad para hacer pruebas: La estructura en capas de la arquitectura Hexagonal facilita la realización de pruebas unitarias. Al tener responsabilidades claramente definidas para cada capa, es más fácil identificar qué aspectos se deben probar durante la implementación del sistema.

Regla de dependencia. En una arquitectura hexagonal, la regla de dependencia sostiene que las dependencias deben estar dirigidas exclusivamente hacia las políticas de nivel superior, promoviendo así la modularidad, independencia y escalabilidad del software (Quincho, 2021). Según Robert (2017), esto implica que los componentes internos no deben tener conocimiento ni hacer referencia a los componentes externos, lo que permite que el software mantenga un nivel de abstracción elevado y encapsule detalles de alto nivel. Al integrar esta práctica en la construcción de formularios, se garantiza una arquitectura flexible y fácilmente mantenible. Esto sienta las bases para explorar tácticas de diseño que promuevan la escalabilidad, desempeño y el crecimiento continuo del sistema.

Escalabilidad. Es un atributo clave, ya que se espera que la aplicación pueda adaptarse a los cambios y crecer de manera efectiva. Esto significa que la arquitectura debe ser capaz de manejar un aumento en la carga de trabajo o en el número de usuarios sin comprometer su funcionamiento al momento de administrar o diligencia los formularios. La seguridad es otro aspecto fundamental para garantizar la confidencialidad de la información, permitiendo el acceso solo a los usuarios autorizados con la implementación de mecanismos de autenticación y autorización para proteger los datos sensibles tanto de los usuarios como de los formularios. El desempeño también es un atributo importante para que la construcción de formularios y su diligenciamiento se realicen de manera óptima. La arquitectura debe estar diseñada para

minimizar los tiempos de respuesta y maximizar la eficiencia en el procesamiento de los formularios (Alonso, 2021).

Desempeño. Se refiere a la capacidad y eficacia con la que una aplicación o sistema realiza sus funciones. Se evalúa la velocidad de ejecución de las tareas, el tiempo de respuesta a las solicitudes de los usuarios y la eficiencia en el uso de los recursos del sistema, como la memoria y la capacidad de procesamiento. Un software de alto desempeño es capaz de realizar sus funciones de manera rápida y efectiva, incluso cuando se enfrenta a grandes volúmenes de datos o a una carga de trabajo intensa. Por lo tanto, el desempeño del software es fundamental para garantizar una experiencia de usuario satisfactoria y para el éxito general de una aplicación o sistema.

Tácticas de diseño. Abstracción en capas: Se aplica el principio de abstracción mediante la creación de capas que permitan separar las funcionalidades del sistema, así como definir una forma coherente de exponer dichas funcionalidades a las capas superiores. Esto facilitará la encapsulación y ocultamiento de la complejidad de los dominios en cada contexto delimitado, permitiendo que la lógica de negocio y la implementación técnica estén abstraídas detrás de interfaces bien definidas.

Api: Se diseña una API web para encapsular las funcionalidades del sistema, proporcionando una interfaz estandarizada que permite a otros componentes interactuar con ellas de forma eficiente y uniforme. Esto facilita que cada dominio ofrezca su información de manera independiente. Las interfaces de usuario pueden entonces consumir los datos proporcionados por la API, adaptando la presentación de la información según las necesidades y preferencias del usuario final, lo que permite satisfacer los contextos delimitados de manera específica.

Interfaces: Se emplea interfaces que permitan a los componentes exponer sus funcionalidades sin revelar los detalles de implementación, fomentando el modularidad y facilitando futuras modificaciones o actualizaciones.

Estos conceptos fueron considerados en el diseño de la solución. Aunque el contexto relacionado con la creación de formularios, que pueden dividirse en secciones y ser completados por diferentes actores, parece inicialmente no presentar demasiada complejidad, se espera que estos enfoques permitan desarrollar una solución escalable y adaptable para la creación dinámica de formularios.

Anexo 2. Descripción de plataformas existentes para crear formularios

Mendix disponible en Mendix (2022). Es una plataforma que permite a desarrolladores de varios niveles técnicos crear aplicaciones de forma rápida, abstrayendo y automatizando el proceso de desarrollo. Mendix se ofrece principalmente como solución para el desarrollo rápido de aplicaciones, bajo el modelo de plataforma como servicio (PaaS), lo que disminuye la dificultad de los usuarios al llevar sus soluciones a producción. El desarrollo de una solución combina componentes para diseño de la interfaz, el modelado de datos, la gestión de la persistencia de datos, el modelado de procesos o la lógica de aplicación y el soporte de la solución. La aplicación ofrece el funcionamiento en modo offline bajo algunas restricciones; sin embargo, en su documentación solo se presenta esta funcionalidad para la administración de los formularios. Permite agregar campos a los formularios después de ser publicados, pero en su documentación no se especifica el nivel de edición sobre campos que ya existían. Finalmente, presenta un módulo de roles para definir los privilegios en la creación y edición de formularios, pero no para la segmentación de un mismo formulario en diferentes actores para su ejecución.

Bonita disponible en Bonitasoft (2023). Esta plataforma utiliza el estándar para el modelado de procesos de negocio (BPM). Ofrece herramientas de programación visual para no programadores y entornos de desarrollo para que los programadores puedan extender la plataforma y personalizar las aplicaciones a múltiples niveles. Bonita se caracteriza por ofrecer muchas opciones de implementación que a su vez la vuelven más compleja de implementar. Por otro lado, queda en duda la recolección de información en campo, pues esto no está soportado en sus características. Permite el diligenciamiento de los formularios en modo offline y la modificación de los formularios después de su publicación sin alterar el tipo de los campos

que ya existen y tienen información almacenada. Finalmente, con respecto a roles, permite crear y asignar funciones en la administración de los formularios.

Appian disponible en Appian (2022). Es una plataforma Low-Code para diseñar y automatizar procesos a nivel empresarial. Con ella es posible crear aplicaciones y flujos de trabajo rápidamente. Al tener un historial como herramienta para la gestión de procesos empresariales (BPM), requiere que los usuarios construyan modelos conceptuales de flujo de trabajo, definan la interfaz de usuario y la estructura de datos. Appian permite las conexiones a sistemas externos y la gestión de roles de usuario, que es un buen acercamiento para la segmentación de formularios por actores. Finalmente, para el paso a producción requieren instalar la solución en un servidor web que también puede presentarse como un obstáculo para los usuarios finales. Esta aplicación permite la ejecución de los formularios en modo offline. Permite modificar los formularios, pero si no se han cargado respuestas o si se eliminan las respuestas ya registradas. Y permite crear roles en el ámbito de administración de los formularios.

Quickbase disponible en Quickbase (2023). Es una plataforma flexible No-Code que permite resolver problemas mediante la creación sencilla de soluciones personalízales. Quickbase permite a los usuarios configurar entornos de aplicaciones simples y administrar los datos, sin necesidad de comprender los detalles de las tecnologías de base de datos. La plataforma brinda soporte a distintos dispositivos como computadoras personales, tabletas y teléfonos inteligentes que podría ayudar en la recolección de información en campo. Para su paso a producción se requiere de la configuración de un ambiente productivo. El modo offline funciona para responder partes de los formularios que se pueden configurar desde un módulo de administración. Esta aplicación permite modificar los formularios después de ser creados y publicados y además crear roles para administrar los formularios y con ellos restringir el acceso a campos específicos de los formularios, pero no a secciones.

Google Forms disponible en Formularios (2023). Es una herramienta gratuita de creación de formularios para encuestas y cuestionarios online, que permite recopilar y organizar información que puede ser exportada a una a una hoja de cálculo. Los formularios creados con esta herramienta, de acuerdo con la documentación disponible, solo se puede responder si existe conexión a internet. La posibilidad de personalizar el formulario es limitada en campos para subir imágenes y documentos. Y permite dividir el formulario y compartirlo con distintos usuarios, pero no asignar secciones a actores específicos.

Bizagi disponible en Bizagi (2022). Es una herramienta para que el modelado de procesos. Entre sus funcionalidades, permite modelar procesos para la generación de formularios. Las soluciones construidas con Bizagi permiten almacenar las respuestas sin necesidad de estar conectado a internet desde su aplicación móvil. Es posible editar los formularios cuando su estructura es simple, con limitaciones para los tipos de datos y su estructura. Finalmente, permite la creación de roles con usuario y contraseña para la administración de los formularios.

Mightyforms disponible en Mightyforms (2023). Es una herramienta No-Code para construir formularios que permite a los participantes y al administrador acceder desde una URL sin necesidad de tener conexión a internet. Permite dividir el formulario en secciones, aunque no tiene un módulo para gestionar roles y por esta razón no es posible asignar secciones a diferentes actores. Finalmente, permite actualizar el formulario cuando este ya se ha publicado para adicionar campos, parámetros y modificar sus formatos.

SurveySparrow disponible en SurveySparrow (2022). Es una herramienta para la construcción de formularios que permite a los usuarios diligenciar el formulario en modo offline, además de modificar sus campos y estructura con limitaciones en cuanto a los tipos de datos y estructura. No permite crear roles ni dividir el formulario para su asignación.

Forms On Fire disponible en Formsonfire.com (2024). Es una aplicación móvil para capturar y enviar formularios móviles como inspecciones, auditorías, informes de seguridad, pedidos y muchas otras tareas. Los usuarios pueden seleccionar de una biblioteca de más de 140 aplicaciones de formulario prediseñadas o producir su propio formulario personalizado utilizando el diseñador de formularios de arrastrar y soltar. La aplicación móvil funciona sin conexión a internet. Esta es una aplicación enfocada en la experiencia de los usuarios para construir formularios; sin embargo, con respecto a la modificación después de su creación, no se especifican funcionalidades. Finalmente, presenta la función para la restricción de accesos, pero no la segmentación de un formulario por roles.

Microsoft Power Apps disponible en Microsoft (2023). Este es un servicio soportado por Microsoft para crear y lanzar aplicaciones web a partir de plantillas, con la técnica de arrastrar y soltar. Esta plataforma permite configurar los formularios para ejecutarse en modo offline y añadir campos o información, sin embargo, para ello se requiere crear lógica con pseudocódigo que podría traer cierto grado de dificultad a los usuarios. En cuanto a la edición de los formularios, la plataforma permite modificar los formularios que no afecten el modelo de datos inicial o sus propiedades, por ejemplo, cambiar el tipo de dato, pero fácilmente es posible añadir nuevos campos o adicionar opciones a un combo. Finalmente, permite definir roles y permitir el acceso a los formularios a determinados roles, pero no la segmentación de un mismo formulario en diferentes roles. Esta funcionalidad es usada generalmente para la administración de los formularios.

Pega disponible en Pega (2023). Es una plataforma No-Code con la cual se realiza la configuración y actualización de aplicaciones sin codificación ni secuencias de comandos. Esta plataforma también permite la definición de procesos, así como el diseño y la modificación de la interfaz de usuario, que es una buena aproximación para la modificación de los formularios después de haber sido usados y sin afectar la información que ya ha sido recolectada. Al ser

una solución para la automatización de procesos robóticos, así como para la gestión de procesos, la complejidad para construir una solución es grande, independientemente de la complejidad del problema. Se requiere el levantamiento del servidor web para llevar la solución a producción. Permite editar y almacenar datos activando el modo offline. Permite editar el formulario y después actualizarlo. Y permite crear usuarios y asignar roles al formulario. Sin embargo, estos accesos solo limitan el acceso a ciertos campos de la información recopilada en los formularios.

GoCanvas disponible en GoCanvas (2022). Es un servicio de suscripción que funciona en la nube y permite a las empresas sustituir los formularios en papel por aplicaciones para smartphones y tablets. Este sistema de gestión permite integrar datos de otras plataformas y, por ser una aplicación personalizable, la información se puede completar de forma automática. Es posible incorporar funciones como GPS, captura de imágenes, distribución, escaneo de códigos de barras, firmas electrónicas, que son campos que podrían implementarse en la solución propuesta en este trabajo. Los formularios pueden ser ejecutados en modo offline y editar bajo ciertas restricciones, como la no actualización de estadísticas y respetando los tipos de datos. Se puede editar la estructura y el formato de los requerimientos del formato. Para esto, se edita la aplicación y, una vez hechos los cambios, se publica nuevamente. Permite crear roles, los cuales pueden editar el formulario o eliminarlo.

TypeForm disponible en Typeform (2023). Es una aplicación No-Code para la generación de formularios. Guarda las respuestas de los usuarios que pierdan conexión a internet mientras se mantengan en la aplicación abierta y carga la información cuando la conexión se restablezca. Permite la edición de los formularios, pero esto genera un formulario con una base de datos nueva. Finalmente, es posible crear roles y asignarles niveles de acceso para la administración de los formularios.

Jotform Mobile Forms disponible en Jotform (2023). Es una aplicación con capa gratuita que brinda a los usuarios la flexibilidad de trabajar desde cualquier lugar, colaborar fácilmente y optimizar el proceso de recolección de datos, creación y edición de formularios. Este permite asignar formularios a integrantes de equipo para que puedan acceder, completar, visualizar y administrar las respuestas. Permite responder el formulario offline únicamente desde la aplicación móvil. Los roles para administrador solo pueden ser asignados en una versión paga. Finalmente, la edición del formulario crea uno nuevo.

SurveyMonkey disponible en SurveyMonkey (2023). Es una plataforma que permite crear encuestas desde cero o utilizar las plantillas prediseñadas. Incluyen acceso a paneles de encuestados, detección de bots y fraudes, perfiles de encuestados, segmentación demográfica, gestión de diseño de encuestas, informes en tiempo real, filtrado de datos y más. Los datos recopilados con SurveyMonkey se pueden analizar mediante informes personalizados en tiempo real, filtros y tabulaciones cruzadas. Las encuestas pueden ser diligenciadas en modo offline. Además, permite editar la totalidad del formulario mientras no haya respuestas y si hay respuestas, las opciones de modificación son limitadas. Y finalmente permite crear roles con diferentes niveles de acceso para administrar los formularios.

MySurveyLab disponible en Surveylab (2022). Es una herramienta que facilita la creación de encuestas personalizadas, automatiza el proceso de recopilación de respuestas y proporciona creación de informes y analíticas en tiempo real. Se puede usar en modo colaborativo con equipos de todos los tamaños, los usuarios pueden trabajar en equipo y asignar roles y derechos de acceso a encuestas y pruebas. La aplicación proporciona cifrado HTTPS/SSL para las encuestas, verificación de seguridad de las contraseñas, gestión de políticas de seguridad y registros de eventos para que las empresas puedan realizar un seguimiento de las operaciones. Permite responder los formularios en modo offline, editar sus

campos antes de ser publicados y crear roles con diferentes niveles de acceso para administrar los formularios.

Bubble disponible en Bubble (2022). Es una herramienta intuitiva y totalmente personalizable que permite crear aplicaciones interactivas para navegadores web, móviles y de escritorio. Además, es posible desarrollar lógica y administrar las bases de datos de las aplicaciones, y desarrollar la interfaz de usuario con total libertad de diseño sin ningún conocimiento de HTML o CSS. En su documentación se especifica el modo offline para algunas funciones de administración del formulario. Permite la edición después de haber creado un formulario, pero se debe publicar nuevamente. Y finalmente es posible crear usuarios para permitir su ingreso a un formulario completo.

FastField disponible en FastField (2016). Es una solución para crear formularios móviles personalizados. Automatiza el flujo de trabajo de formularios digitales con herramientas que incluyen un creador de formularios personalizable, aplicaciones nativas para iOS, Android y web. Permite la recolección de datos offline, integraciones de sistemas de terceros y más. Los formularios guardan los datos y se sincronizan con la plataforma automáticamente una vez recuperada la conexión. Cuando se modifica un formulario, se envía uno nuevo a usuarios seleccionados con las ediciones realizadas. Finalmente, permite crear 8 roles a nivel de administración de los formularios.

Wavemaker disponible en WaveMaker (2014). Es una plataforma de código bajo altamente escalable, fácil de adoptar e integrar para potenciar plataformas empresariales y desarrollar aplicaciones. Permite a los usuarios crear, dentro de los límites de una arquitectura de aplicación genérica, aplicaciones de complejidad baja o moderada. Para cualquier caso se requiere una comprensión básica de desarrollo de software, pues si es necesario implementar requisitos avanzados, generalmente se requiere estar familiarizado con tecnologías como Java,

JavaScript, SQL, HTML, CSS, SOAP y una variedad de APIs. Permite trabajar offline bajo unas reglas definidas por el administrador que delimitan las funciones que pueden estar activas sin conexión. Permite modificar el formulario a nivel técnico, pero debe hacerse una nueva publicación del formulario, que podría generar una nueva fuente de datos. Permite crear tres tipos de roles para administrar formularios, aunque en la documentación no se menciona la segmentación de un mismo formulario para diferentes actores.

Anexo 3. Implementación de los esquemas de datos

Tabla 29

Formularios


```
{
 "form_id": "0001",
 "form_name": "Estudio 1",
 "state": "enable",
 "sections": [
  {
   "id": "0001",
   "section_name": "mi sección",
   "access": [
    {
     "user_id": "0001",
     "user_name": "Secretaria",
     "permission": ["read", "write"]
    },
    {
     "user_id": "1085302445",
     "user_name": "Johnny Chinchajoa",
     "permission": ["read"]
    }
   ],
   "fields": [
    {
     "id": "0001",
     "is_required": true,
     "type": "select",
     "label": "Tipo de documento",
     "values": [
      {
       "id": "cc",
       "text": "CC"
      },
      {
       "id": "nit",
       "text": "NIT"
      }
     ]
    },
    {
     "id": "0002",
     "is_required": true,
     "type": "input",
     "label": "Nombres",
```

```
        "placeholder": "Escribe el nombre",
        "max_length": "60"
      }
    ]
  }
 ]
}
```


Nota. Esquema de datos para formularios. Elaborado por Chinchajoa (2023)

Tabla 30

Casos de uso


```
{
 "id": "0001",
 "case_name": "mi caso",
 "case_state": {
  }
 "form_id": "0001",
 "sections": [
   {
    "section_id": "0001",
    "section_state": "inProgress",
    "file_values": [
      {
       "id": "0001",
       "value": {
        "id": "cc",
        "text": "CC"
       }
      },
      {
       "id": "0002",
       "value": "Johnny Chinchajoa"
      }
    ]
   }
 ]
}
```


Nota. Esquema de datos para casos de formulario. Elaborado por Chinchajoa (2023)

Tabla 31

Formularios y casos de uso por usuario


```
{
 "user_id": "0001",
 "forms": [
   {
    "form_id": "0001",
    "form_name": "Formulario 1",
   "form_author": "jact1001@gmail.com",
    "cases": [
      {
       "case_name": "estudio de suelos Juan",
       "case_id": "0001",
       "state": {
        "id": "pendiente",
        "text": "Pendiente"
       }
      }
    ]
   }
 ]
}
```


Nota. Formularios y casos por usuario. Elaborado por Chinchajoa (2023)

Anexo 4. Tecnologías sugeridas para implementar la aplicación

Frontend. HTML5, es un lenguaje de marcado de hipertexto (HyperText Markup Language en inglés). Es el estándar para la definición del esquema de páginas web (Faulkner et al. (2017). Se ha empleado su última revisión, HTML5, que incorpora las últimas novedades a la vez que incorpora mejoras de compatibilidad para distintos navegadores.

CSS3, Las hojas de estilo en cascada (CSS por sus iniciales en inglés) son el mecanismo principal para aplicar estilos visuales a una página web. Su objetivo es vincular unas ciertas propiedades visuales a distintos elementos HTML Tab et al. (2021).

Javascript, es un lenguaje de programación interpretado basado en el estándar ECMAScript 2. Se incorpora en todos los navegadores modernos desde 2012 Mozilla.org. (2023). Permite la ejecución de código en el lado del cliente, dotando a los sitios web de una mayor interacción con el usuario.

Redux, es una biblioteca de administración de estado para aplicaciones JavaScript basadas en React. Ofrece un contenedor centralizado y predecible para el estado, siguiendo el patrón Flux y el flujo de datos unidireccional (Redux, 2020)

Axios, es una biblioteca JavaScript que se utiliza combinada para realizar solicitudes HTTP desde el navegador web o desde Node.js. Esta biblioteca simplifica la realización de solicitudes AJAX y HTTP, lo que la convierte en una herramienta valiosa para interactuar con servidores y APIs desde aplicaciones web y servidores (Axios, 2020).

Backend. A continuación, se agrupan las definiciones empleadas, cada una tomada desde las fuentes oficiales.

Node Js, es un entorno de ejecución de JavaScript del lado del servidor, que permite construir aplicaciones web y servicios escalables y de alto rendimiento (Casciaro, 2014).

Typescript, es un lenguaje de programación desarrollado por Microsoft. Es una extensión de JavaScript que agrega tipos estáticos opcionales y otras características que

facilitan el desarrollo de aplicaciones más grandes y mantenibles. TypeScript compila un estándar de JavaScript y se utiliza ampliamente en el desarrollo web y de aplicaciones (Syed, 2017).

Express, es un framework web de código abierto para Node.js. Está diseñado para simplificar la creación de aplicaciones web y API RESTful. Express proporciona una serie de utilidades y middleware que facilitan el manejo de rutas, aplicaciones HTTP y otras tareas comunes en el desarrollo web (Hahn, 2016).

Ts.ED, es un framework de desarrollo de API RESTful para Node.js y TypeScript. Está construido sobre Express y se enfoca en la creación de API escalables y mantenibles utilizando TypeScript. Ts.ED proporciona una estructura organizativa y herramientas para desarrollar aplicaciones API de manera eficiente (Ts.ED, 2016).

Lerna, es una herramienta de código abierto para gestionar proyectos de JavaScript y TypeScript con múltiples paquetes. Se utiliza para administrar monorepositorios, lo que significa que puedes tener varios paquetes dentro de un solo repositorio y gestionar sus dependencias y versiones de manera eficiente. Lerna facilita la colaboración en proyectos complejos y la administración de dependencias compartidas entre paquetes (Lerna, 2023).

MongoDB, Como se introduce en el capítulo 2, MongoDB es una de las plataformas más populares para crear bases de datos NoSQL orientadas a documentos. Para el versionamiento y despliegue, se analiza Github, es una de las múltiples herramientas para el control de versiones de código basadas en el sistema Git.

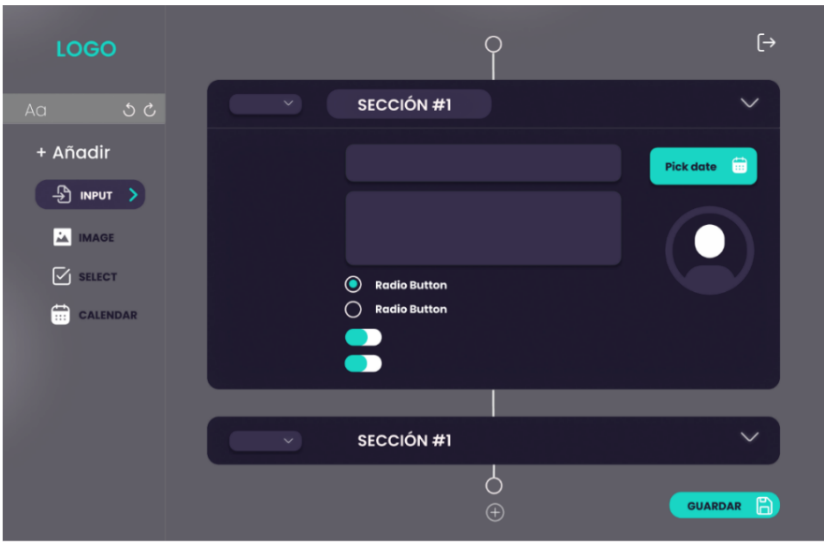
Gtthub flow, se refiere a un conjunto de prácticas y un flujo de trabajo de desarrollo de software que se utiliza combinado en proyectos gestionados en GitHub, una plataforma de alojamiento de repositorios de código (GitHub, 2023).

AWS services, hace referencia a servicios que tiene AWS que pueden ayudar a las organizaciones a crear, implementar y escalar aplicaciones y recursos en la nube de manera efectiva. Cada servicio tiene sus propias características y casos de uso específicos.

Anexo 5. Diseños Web

Figura 18

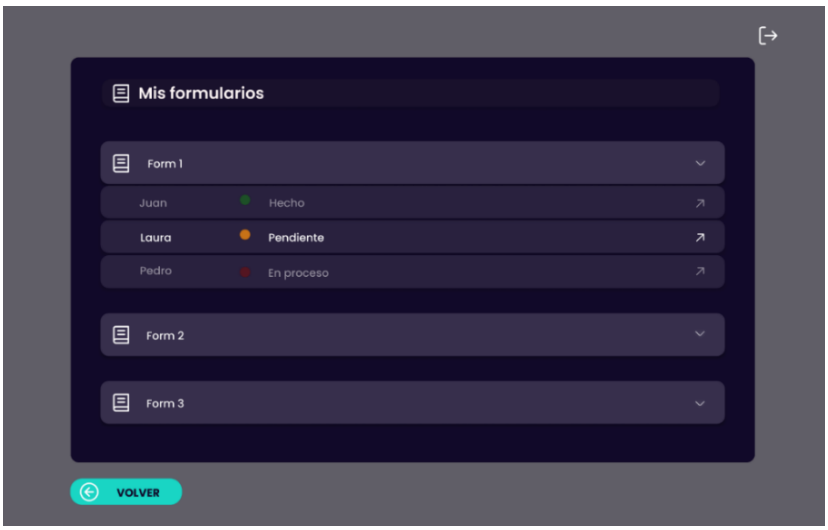
Pantalla para diseñar formularios



Nota. Visual de la pantalla en el diseño de formulario. Elaborado Chinchajoa (2024)

Figura 19

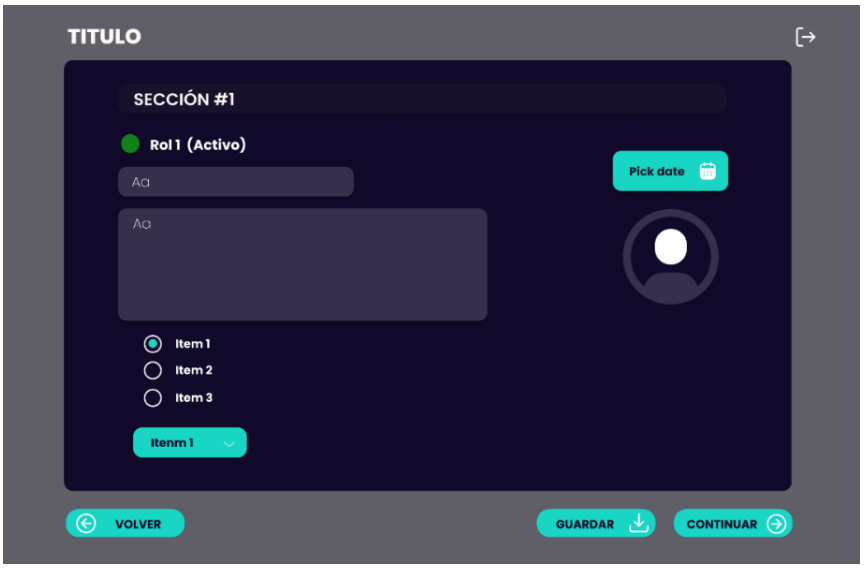
Pantalla con lista de formularios y casos



Nota. Visual de la pantalla Formularios. Elaborado Chinchajoa (2024)

Figura 20

Pantalla para diligenciar casos de formularios



Nota. Visual de la pantalla Formularios. Elaborado Chinchajoa (2024)

Anexo 6. Guion de pruebas de usabilidad

Introducción

¡Bienvenidos a la prueba de usabilidad de la aplicación para crear formularios dinámicos! Con esta herramienta, podrás diseñar formularios personalizados y adaptados a tus necesidades.

La aplicación consta de tres pantallas fundamentales. La primera pantalla permite crear formularios de manera sencilla mediante la función de arrastrar y soltar campos. Además, se tendrá la flexibilidad de diseñar secciones específicas y asignarlas a diferentes actores. Una segunda pantalla se dedica a la consulta de formularios asignados, permitiendo visualizar el estado de los casos de uso relacionados con cada formulario. Aquí, podrán seleccionar los casos de uso que se deseen diligenciar. Es importante destacar que el término "caso de uso" se refiere a una situación particular en la que se aplica un formulario. La tercera pantalla corresponde a la vista de diligenciamiento, donde se podrá completar los campos del formulario asignado. Durante este proceso, se tendrá la opción de guardar el progreso de la sección diligenciada.

Agradecemos tu participación en esta prueba de usabilidad y confiamos en que sus comentarios y observaciones permitirán mejorar la aplicación para brindar una experiencia óptima en la creación y gestión de formularios dinámicos. ¡Comencemos!

Etapa 1: Creación de un formulario

1. Ingresa a la aplicación Dynamic Forms en la siguiente url: https://dsb471zsol61i.cloudfront.net/login he inicia sesión con una cuenta de gmail.
2. Selecciona la opción “Diseño de formulario” en el menú superior.
3. Asígnale un nombre al formulario que puedas recordar.
4. Asígnale un nombre a la primera sección del formulario.

5. Selecciona las personas que podran diligenciar la sección. Es importante que acá asignes una sección al usuario “testform1 testform1” para que más adelante puedas validar la asignación.
6. Asigna uno o más campos a la sección y dales un nombre a los campos.
7. Crea una segunda sección y asígnala a otro usuario. Y repite los items 4 y 6.
8. Guarda el formulario y se te notificará que el proceso fue exitoso.

Etapa 2: Consultar formularios asignados

El objetivo de este paso es verificar que los usuarios asignados a cada sección puedan acceder a la parte correspondiente del formulario. Para ello, puedes salir y hacer login con alguna de estas cuentas de usuario. Las credenciales para 'testform1' son las siguientes:

correo electrónico: testdinamicform1@gmail.com

contraseña: Testdynamicform_1.

1. Ingresa a la página de formularios “Formularios” https://dsb471zsol61i.cloudfront.net/user-forms con las credenciales de una de las personas a los que asignaste alguna sección.
2. Valida que se muestre el formulario que se le asignó.

Etapa 3: Creación y diligenciamiento de un caso de uso a un formulario

Con uno de los actores asignados (acá puedes continuar usando a testform1) crea un caso sobre el formulario que creaste.

1. Abre el caso de uso dando click sobre el nombre de caso de uso. Esto abrirá una pestaña nueva en el navegador.
2. Valida que solo se muestren las secciones del formulario que se le asignó al usuario con el que hiciste login.

3. Diligencia y guarda los cambios. Se te notificará que el proceso fue exitoso. Recarga la página para validar que se guardaron los cambios.
4. Vuelve a la página de formularios “Formularios” https://dsb471zsol61i.cloudfront.net/user-forms y valida que el estado del caso esté “en progreso”.

Etapa 4: Modificación de formulario

1. Ingresa con el usuario con el que diseñaste el formulario a la página de formularios “Formularios” https://dsb471zsol61i.cloudfront.net/user-forms.
2. Busca tu formulario y selecciona editar formulario. Ahora agrega nuevos campos y modifica o elimina los existentes.
3. Guarda el formulario.
4. Ahora ve a la página “Formularios” https://dsb471zsol61i.cloudfront.net/user-forms abre un caso de tu formulario y valida que se vean reflejados los cambios que hiciste al formulario.

Etapa 5: Diligencie el formulario en modo offline

1. Crea un nuevo caso de uso en tu formulario o abre alguno existente.
2. Desconecta el acceso a internet del dispositivo que estás usando.
3. Edita o diligencia los campos.
4. Guarda tus cambios.
5. Ahora Conecta nuevamente el acceso a internet. Se te notificará que tus cambios fueron guardados.
6. Recarga la aplicación y valide que la información ingresada exista.

Etapa 6: Consultar y descargar información recolectada

1. Ingresa con el usuario que diseñaste el formulario a la página de formularios.
2. Selecciona la opción “Descargar casos” sobre tu formulario. Esto iniciará la descarga de un archivo de excel. Valida que la información de este documento coincida con la diligenciada en cada uno de los casos y sus campos.

Etapa 7: Diligencie el formulario de evaluación de la aplicación

1. Crea un caso nuevo en el formulario: Evaluación de Dynamic Forms. Este lo puedes encontrar en la página de formularios “Formularios” https://dsb471zsol61i.cloudfront.net/user-forms.
2. Abre el caso y responde las preguntas en cada una de las secciones de acuerdo con la experiencia que tuviste en esta prueba.

¡Gracias por participar en la prueba de usabilidad! Tu aporte es fundamental para nosotros y nos ayudará a mejorar la aplicación de manera significativa.

Preguntas de retroalimentación

1. Tu experiencia con formularios
    a. ¿Tienes experiencia en el uso de aplicaciones para crear formularios?
    b. ¿Qué tipo de formularios sueles crear en estas aplicaciones?
    c. ¿Qué dispositivos usas en el diligenciamiento de formularios?
2. Diseño de formulario
    a. ¿Es claro cómo añadir el nombre del formulario?
    b. ¿Es claro cómo asignar campos a una sección del formulario?
    c. ¿Es claro cómo asignar secciones a un formulario?

   d. ¿Es claro cómo asignar los diligenciadores de una sección del formulario?
   e. ¿Es claro cómo guardar un formulario?
   f. ¿Pudiste ejecutar los 8 pasos de la etapa 1?
   g. En caso de responder NO a la pregunta anterior ¿qué pasos NO pudiste ejecutar?

3. Lista de formularios
   a. ¿Los formularios fueron asignados correctamente?
   b. ¿Se entiende que es un caso de un formulario?
   c. ¿Es claro cómo crear un caso?
   d. ¿Pudiste ejecutar los 2 pasos de la etapa 2?
   e. En caso de responder NO a la pregunta anterior ¿qué pasos NO pudiste ejecutar?
4. Caso de uso
   a. ¿Las secciones fueron asignadas correctamente?
   b. ¿Es claro como diligenciar y guardar la información?
   c. ¿Pudiste ejecutar los 5 pasos de la etapa 3?
   d. En caso de responder NO a la pregunta anterior ¿qué pasos NO pudiste ejecutar?
5. Modificación de formulario
   a. ¿Es claro cómo modificar un formulario existente?
   b. ¿Pudiste ejecutar los 4 pasos de la etapa 4?
   c. En caso de responder NO a la pregunta anterior ¿qué pasos NO pudiste ejecutar?
6. Diligenciamiento de un caso de formulario en modo offline
   a. ¿Pudiste ejecutar los 5 pasos de la etapa 5?
   b. En caso de responder NO a la pregunta anterior ¿qué pasos NO pudiste ejecutar?
7. Consulta y descarga de información recolectada
   a. ¿Es claro cómo descargar la información recolectada?
   b. ¿Pudiste ejecutar los 2 pasos de la etapa 6?

c. En caso de responder NO a la pregunta anterior ¿qué pasos NO pudiste ejecutar?

8. Preguntas de cierre

a. En una escala del 1 al 10, ¿qué tan satisfecho(a) te sientes con la experiencia general al usar la aplicación?

b. ¿La aplicación se ajusta a tus necesidades ó a lo que esperabas lograr con ella?

c. ¿Hubo aspectos de la aplicación que consideras que podrían mejorarse?

d. ¿Hubo alguna parte de la aplicación que te resultó especialmente útil o que te gustó mucho?

e. ¿Conoces alguna herramienta que permita hacer lo mismo?

Anexo 7. Sugerencias y comentarios de pruebas de usabilidad

En una escala del 1 al 10, ¿qué tan satisfecho(a) te sientes con la experiencia general al usar la aplicación?

Para la cual en promedio se obtuvo un valor de 8.5. Con este resultado se puede concluir que en general, los usuarios están bastante satisfechos con la aplicación.

¿La aplicación se ajusta a tus necesidades o a lo que esperabas lograr con ella?

En esta pregunta, todos los usuarios indicaron que la aplicación cumplió con sus expectativas según lo establecido en la introducción del guion de la prueba.

¿Hubo aspectos de la aplicación que consideras que podrían mejorarse?

Según las respuestas proporcionadas en esta pregunta y los resultados se destacan los siguientes aspectos para mejorar:

1. Diseño de formulario

- No es claro donde añadir el nombre del formulario.
- El botón de crear una sección nueva no es intuitivo
- Falta un módulo para añadir usuarios que no se hayan registrado en la aplicación.
- Recordar guardar cambios antes de salir de la página.
- Permitir añadir descripción a una sección.
- Visualizar los usuarios añadidos como diligenciadores en cada sección.
- Permitir reorganizar los campos y las secciones.
- Mostrar los campos requeridos para crear un formulario.

2. Lista de formularios y casos

- Limitar el acceso a casos en progreso.

- No mostrar acciones que el usuario no pueda hacer, en caso de no ser el autor del formulario.
- Mostrar loading mientras se crea un caso.

3. Diligenciar caso de formulario

- Permitir pasar el caso a finalizado.
- Añadir validaciones a campos.
- Cerrar el caso si ya se guardaron los cambios.

¿Hubo alguna parte de la aplicación que te resultó especialmente útil o que te gustó mucho?

Para esta pregunta se resumen las respuestas en:

- La facilidad para añadir campos y secciones al formulario.
- Poder elegir diferentes personas a secciones del formulario.
- La creación de casos a un formulario.
- Diligenciamiento en modo Offline.

¿Conoces alguna herramienta que permita hacer lo mismo?

Los usuarios no están familiarizados con alguna herramienta que permita realizar las mismas acciones.

Anexo 8. Usuarios de las pruebas de usabilidad

Tabla 32

Usuarios de pruebas de usabilidad

Id	Nombre	Correo	Profesión	Campo laboral
U01	Flavio Martínez	flavioae@gmail.com	Ing. Electrónico	Arquitecto SOA en Alcaldía mayor de Bogotá
U02	Ricard Patiño	patinoricar@gmail.com	Ing. electrónico	Independiente Desarrollador de software
U03	Mariela Taimal	marielataimal@liceoalejandrodehumboldt.edu.co	Magister en Educación Superior	Coordinadora de Colegio Liceo Alejandro Humboldt
U04	Jhonatan Ceballos	jceballost.basing@gmail.com	Ing Civil	Independiente : Estudio de suelos
U05	Daniel Yépez	dafeyeta@gmail.com	Estudiante de pregrado: Ing. Electrónica	Estudiante
U06	Johan Legarda	legardajohan@gmail.com	Licenciado en informática	Docente de primaria y secundaria
U07	Darly Burbano	darlybur98@gmail.com	Ing. de sistemas	Analista de Requerimientos

U08	Jheferson Murillo	jhefersonmurillo@gmail.com	Ing. de sistemas	Analista de software
U09	Marcela Fernandez	fernandezmendezmarcela@gmail.com	Master of Business and Information Technology	Analista de negocio
U10	Magda Ledesma	maceleer@gmail.com	Maestra en Ciencias Agrícolas	Estudiante de postgrado
U11	Camila Narvaez	camn0905@unicauca.edu.co	Estudiante: Ing Civil	Estudiante
U12	Daniela Solarte	danielasol417@gmail.com	Ing Civil	Independiente: Estudio de suelos
U13	Nathalia Diaz	nataliataimal@unicauca.edu.co	Estudiante: Ing Electronica	Estudiante
U14	Ivan Taimal	rtaimal@gmail.com	Magister en Ing de Sotware	Arquitecto de software
U15	Andrea Alarcon	andreaalarcon@unicauca.edu.co	Contadora publica	Contadora publica
U16	Hernan Taimal	htaimal@gmail.com	Magister en Arquitectura de Software	Ing de Software Sr

Nota. Tabla de usuarios participantes en pruebas

Anexo 9. Servicios de la API

Tabla 33

Servicios de API

Formularios /form	POST: Crear formulario
	GET: Consultar formulario
	PUT: Actualizar formulario
Caso de formulario /case	POST: Crear caso de formulario
	GET: Consultar caso de formulario
	PUT: Actualizar caso de formulario
Formularios de usuario /user-forms	GET: Consultar formularios y casos por usuario
Usuarios /users	GET: Consultar usuarios registrados
Campos de formulario /fields	GET: Consultar campos de formulario

Nota. Síntesis de servicios

Anexo 10. Resultados de pruebas no funcionales

Tabla 34

Creación de casos de formulario

Usuarios virtuales	No SQL				SQL			
	post	ok	fail	% error	post	ok	fail	% error
5 (1 min) 5 (3 min) 0 (1 min)	456	454	2	0,44	337	334	3	0,89
10 (1 min) 10 (3 min) 00 (1 min)	906	900	6	0,66	551	513	38	6,89
15 (1 min) 15 (3 min) 00 (1 min)	1348	1341	7	0,52	992	990	2	0,20
20 (1 min) 20 (3 min) 00 (1 min)	1760	1755	5	0,28	1469	1465	4	0,27
25 (1 min) 25 (3 min) 00 (1 min)	2193	2182	11	0,50	1838	1835	3	0,16
50 (1 min) 50 (3 min) 00 (1 min)	918	3909	9	0,23	2516	2515	1	0,04
100 (1 min) 100 (3 min) 000 (1 min)	5346	5335	11	0,20	2767	2758	9	0,33
500 (1 min) 500 (3 min) 000 (1 min)	5960	5874	86	1,44	3973	3409	564	14,2

Nota. Elaborado Chinchajoa (2024)

Tabla 35

Consulta de formularios

Usuarios virtuales	No SQL				SQL			
	get	ok	fail	% error	get	ok	fail	% error
5 (1 min) 5 (3 min) 0 (1 min)	464	464	0	0	342	342	0	0
10 (1 min) 10 (3 min) 00 (1 min)	923	923	0	0	520	518	2	0,38
15 (1 min) 15 (3 min) 00 (1 min)	1369	1369	0	0	974	974	0	0

20 (1 min) 20 (3 min) 00 (1 min)	1808	1808	0	0	1533	1533	0	0
25 (1 min) 25 (3 min) 00 (1 min)	2253	2253	0	0	1925	1925	0	0
50 (1 min) 50 (3 min) 00 (1 min)	4240	4240	0	0	2656	2656	0	0
100 (1 min) 100 (3 min) 000 (1 min)	5978	5978	0	0	3031	3031	0	0
500 (1 min) 500 (3 min) 000 (1 min)	6733	6733	0	0	3679	3052	627	17,04

Nota. Elaborado Chinchajoa (2024)

Tabla 36

Actualización de casos

Usuarios virtuales	No SQL				SQL			
	put	ok	fail	% error	put	ok	fail	% error
5 (1 min) 5 (3 min) 0 (1 min)	466	465	1	0,21	334	334	0	0
10 (1 min) 10 (3 min) 00 (1 min)	923	922	1	0,11	521	520	1	0,19
15 (1 min) 15 (3 min) 00 (1 min)	1381	1374	7	0,51	991	988	3	0,30
20 (1 min) 20 (3 min) 00 (1 min)	1816	1816	0	0	1555	1555	0	0
25 (1 min) 25 (3 min) 00 (1 min)	2273	2263	10	0,44	1958	1958	0	0
50 (1 min) 50 (3 min) 00 (1 min)	4303	4301	2	0,05	2777	2772	5	0,18
100 (1 min) 100 (3 min) 000 (1 min)	6159	6159	0	0	3194	3186	8	0,25
500 (1 min) 500 (3 min) 000 (1 min)	6839	6733	106	1,55	3224	2871	353	10,95

Nota. Elaborado Chinchajoa (2024)

Tabla 37

Percentil 95

Usuarios virtuales	Percentil 95 (ms)	
	No SQL	SQL
5 (1 min) 5 (3 min) 0 (1 min)	308	885
10 (1 min) 10 (3 min) 00 (1 min)	320	495
15 (1 min) 15 (3 min) 00 (1 min)	319	1160
20 (1 min) 20 (3 min) 00 (1 min)	337	665
25 (1 min) 25 (3 min) 00 (1 min)	334	614
50 (1 min) 50 (3 min) 00 (1 min)	467	1370
100 (1 min) 100 (3 min) 000 (1 min)	1210	2970
500 (1 min) 500 (3 min) 000 (1 min)	11150	12620

Nota. Percentil 95 al crear, consultar y actualizar un caso de un formulario

Anexo 11. Resultados de pruebas no funcionales con MongoDB

A continuación, se presentan tres tipos de graficas:

- Usuarios Virtuales: Este gráfico muestra el número de usuarios virtuales que están realizando solicitudes durante un intervalo de 5 minutos.
- Duración de Solicitudes: Se trata de un gráfico de calor que representa los tiempos de respuesta, resaltando las áreas con mayor densidad de solicitudes. Las zonas en rojo indican donde se concentra la mayor cantidad de solicitudes con esos tiempos de respuesta específicos.
- Percentil 90 y 95: En este gráfico se observa el comportamiento de las solicitudes con respecto al tiempo de respuesta. El percentil 95 se destaca en color azul para ilustrar el límite dentro del cual el 95% de las solicitudes fueron respondidas, brindando una idea del rendimiento en condiciones de carga alta.

Figura 21

Resultados con 5 usuarios virtuales por 5 minutos.

Nota. Generación en MongoDB 5 usuarios

Figura 22

Resultados con 10 usuarios virtuales por 5 minutos

Nota. Generación en MongoDB 10 usuarios

Figura 23

Resultados con 15 usuarios virtuales por 5 minutos

Nota. Generación en MongoDB 15 usuarios

Figura 24

Resultados con 20 usuarios virtuales por 5 minutos

Nota. Generación en MongoDB 20 usuarios

Figura 25

Resultados con 25 usuarios virtuales por 5 minutos

Nota. Generación en MongoDB 25 usuarios

Figura 26

Resultados con 50 usuarios virtuales por 5 minutos

Nota. Generación en MongoDB 50 usuarios

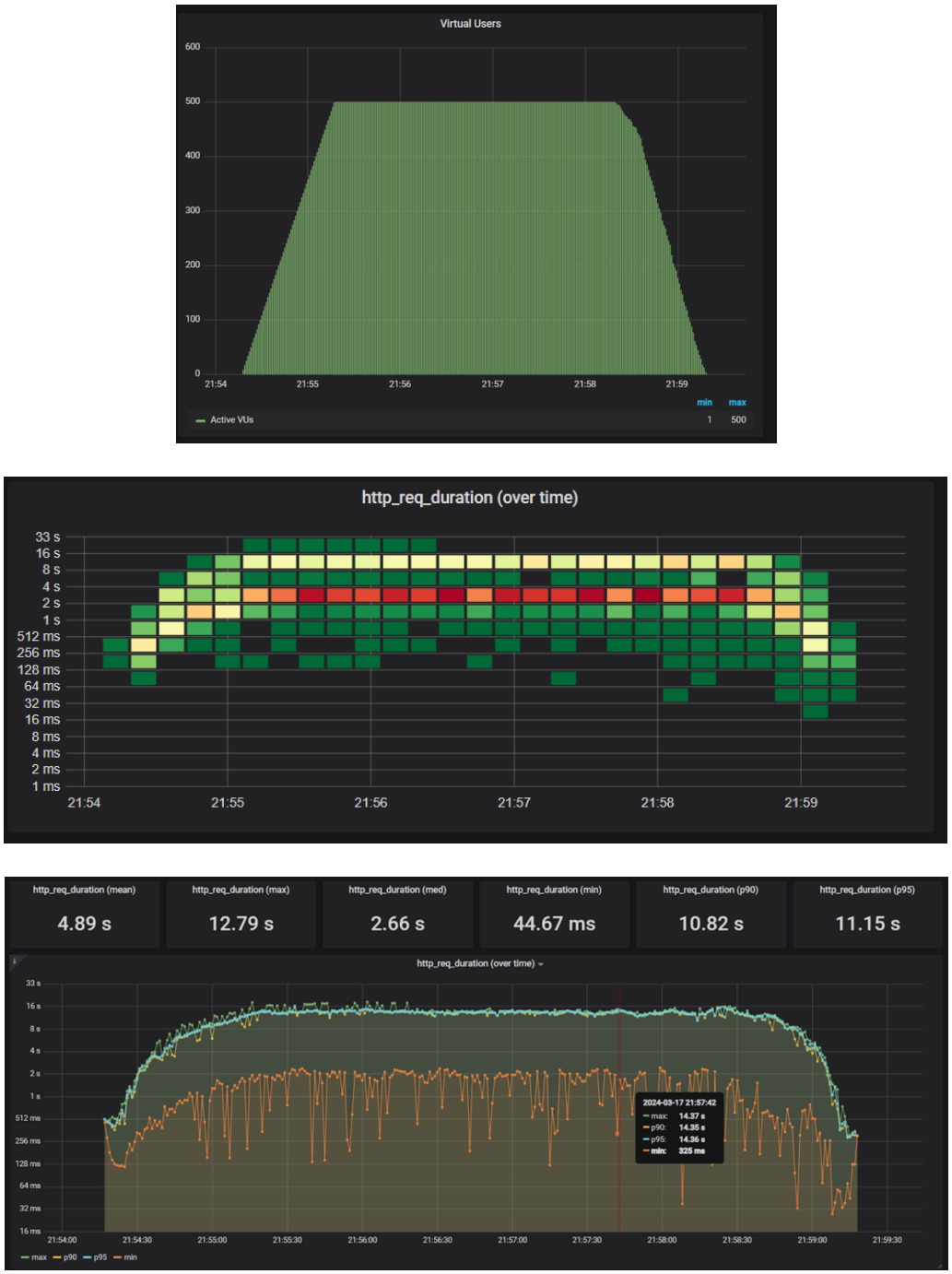
Figura 27

Resultados con 100 usuarios virtuales por 5 minutos

Nota. Generación en MongoDB 100 usuarios

Figura 28

Resultados con 500 usuarios virtuales por 5 minutos

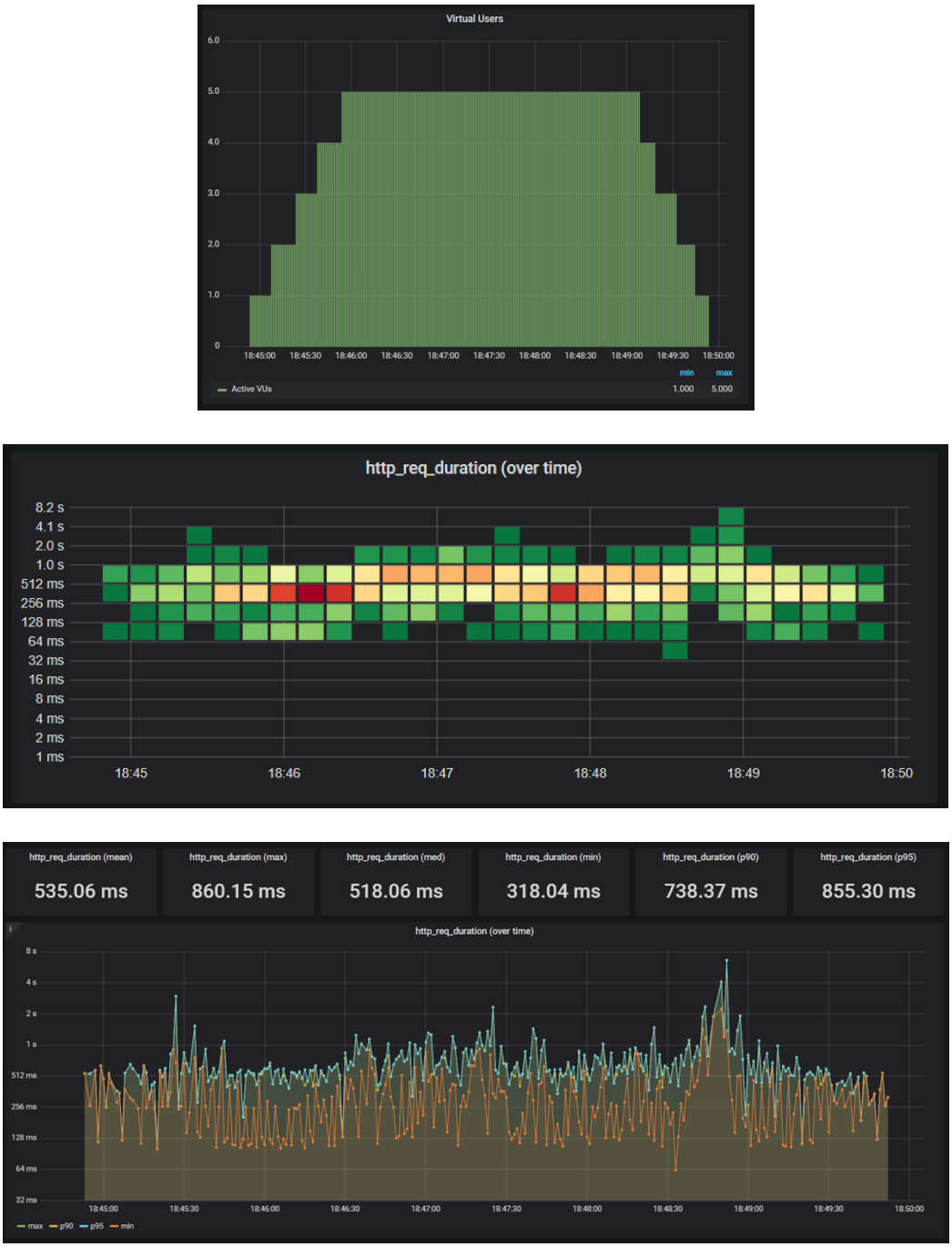


Nota. Generación en MongoDB 500 usuarios

Anexo 12. Resultados de pruebas no funcionales con MySQL Aurora

Figura 29

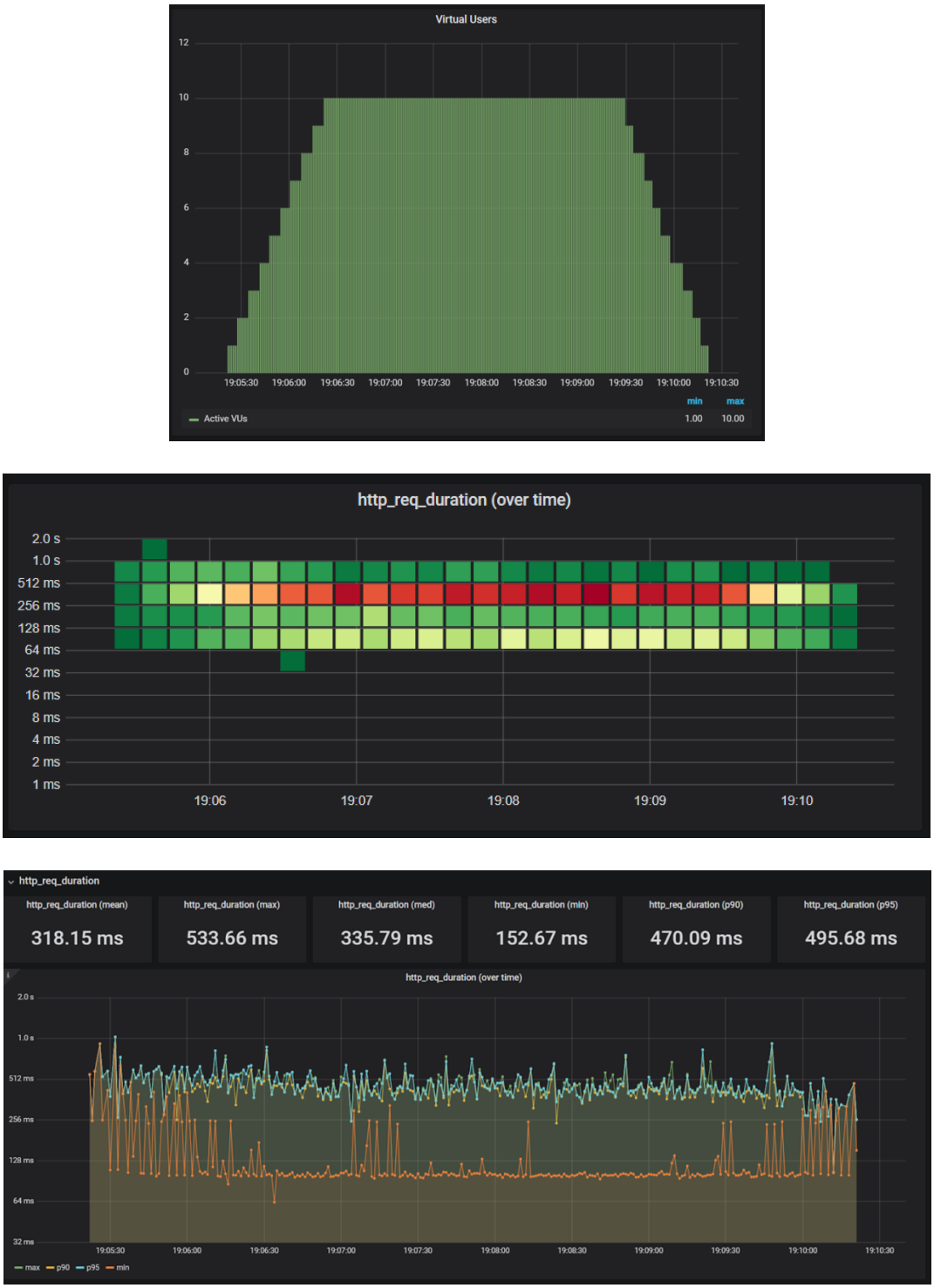
Resultados con 5 usuarios virtuales por 5 minutos



Nota. Generación en MySQL Aurora 5 usuarios

Figura 30

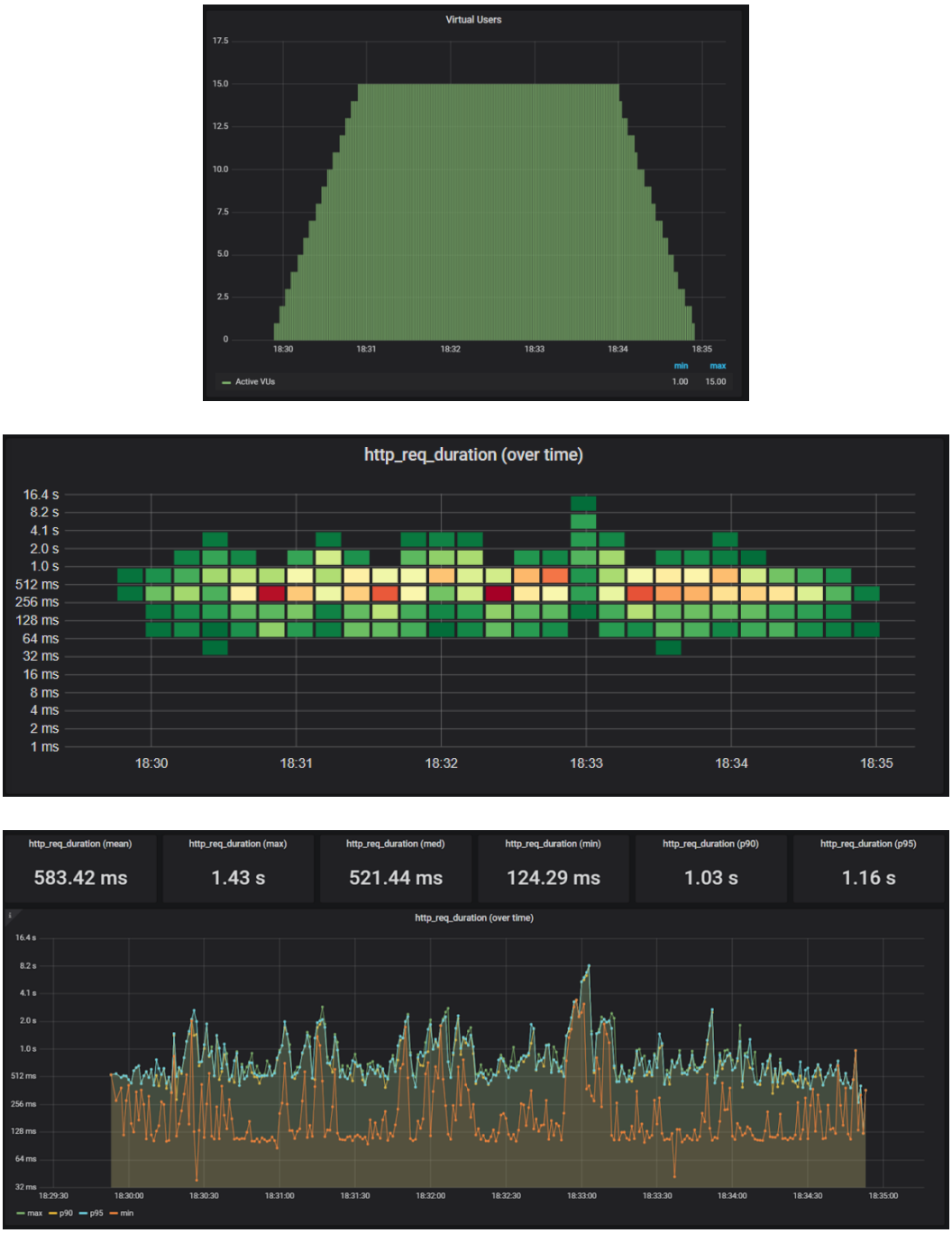
Resultados con 10 usuarios virtuales por 5 minutos



Nota. Generación en MySQL Aurora 5 usuarios

Figura 31

Resultados con 15 usuarios virtuales por 5 minutos



Nota. Generación en MySQL Aurora 10 usuarios

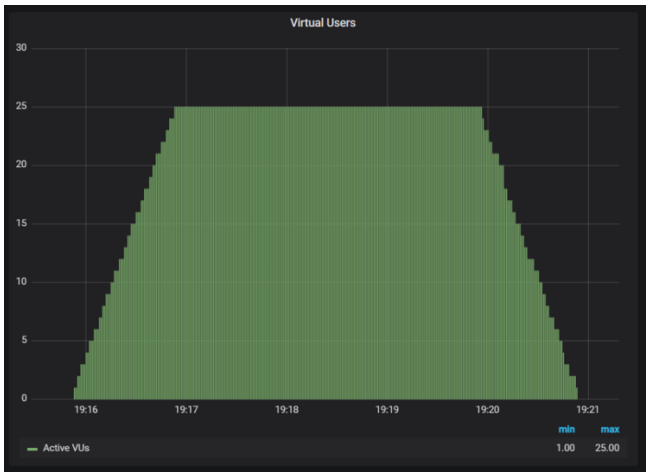
Figura 32

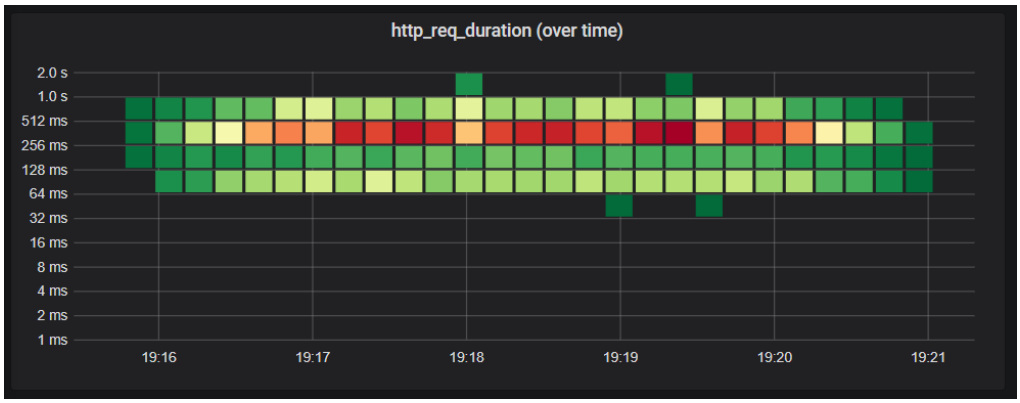
Resultados con 20 usuarios virtuales por 5 minutos

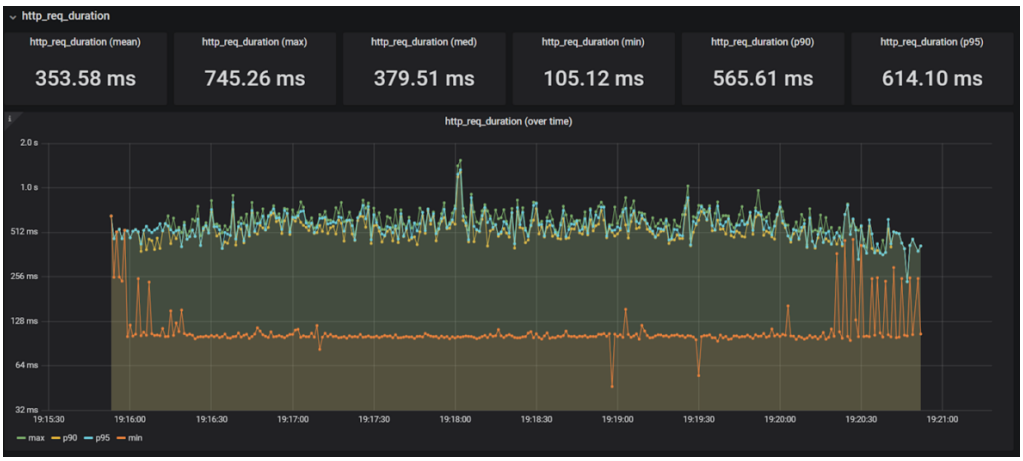
Nota. Generación en MySQL Aurora 20 usuarios

Figura 33

Resultados con 25 usuarios virtuales por 5 minutos







Nota. Generación en MySQL Aurora 25 usuarios

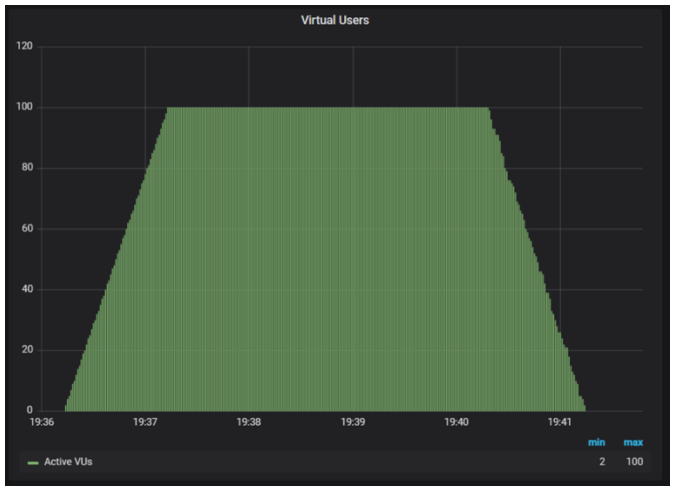
Figura 34

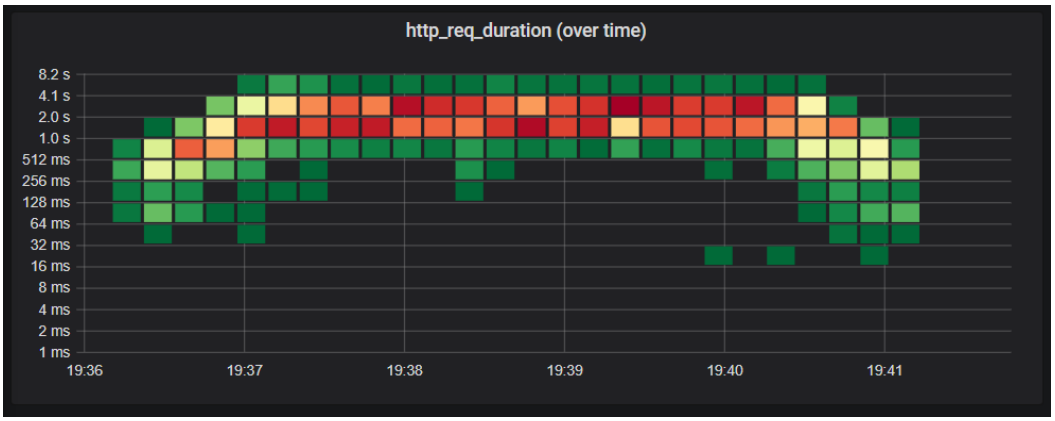
Resultados con 50 usuarios virtuales por 5 minutos

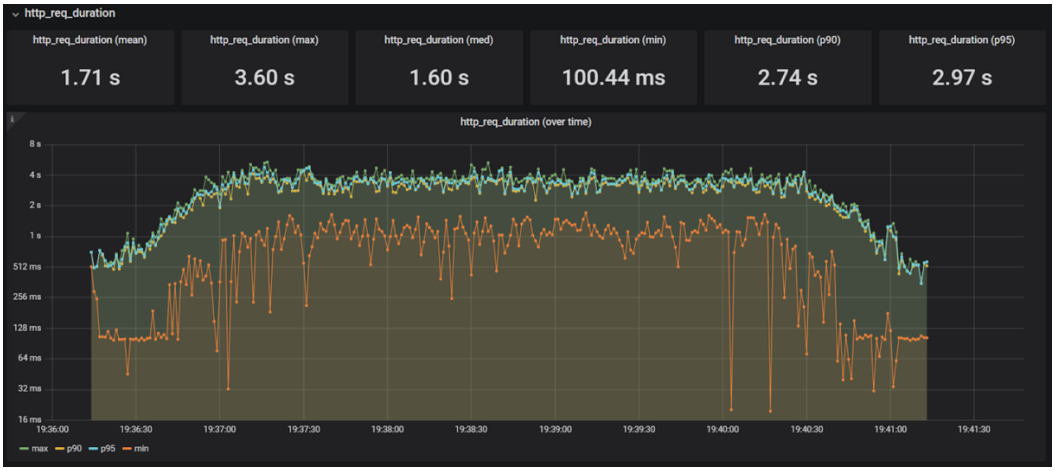
Nota. Generación en MySQL Aurora 50 usuarios

Figura 35

Resultados con 100 usuarios virtuales por 5 minutos



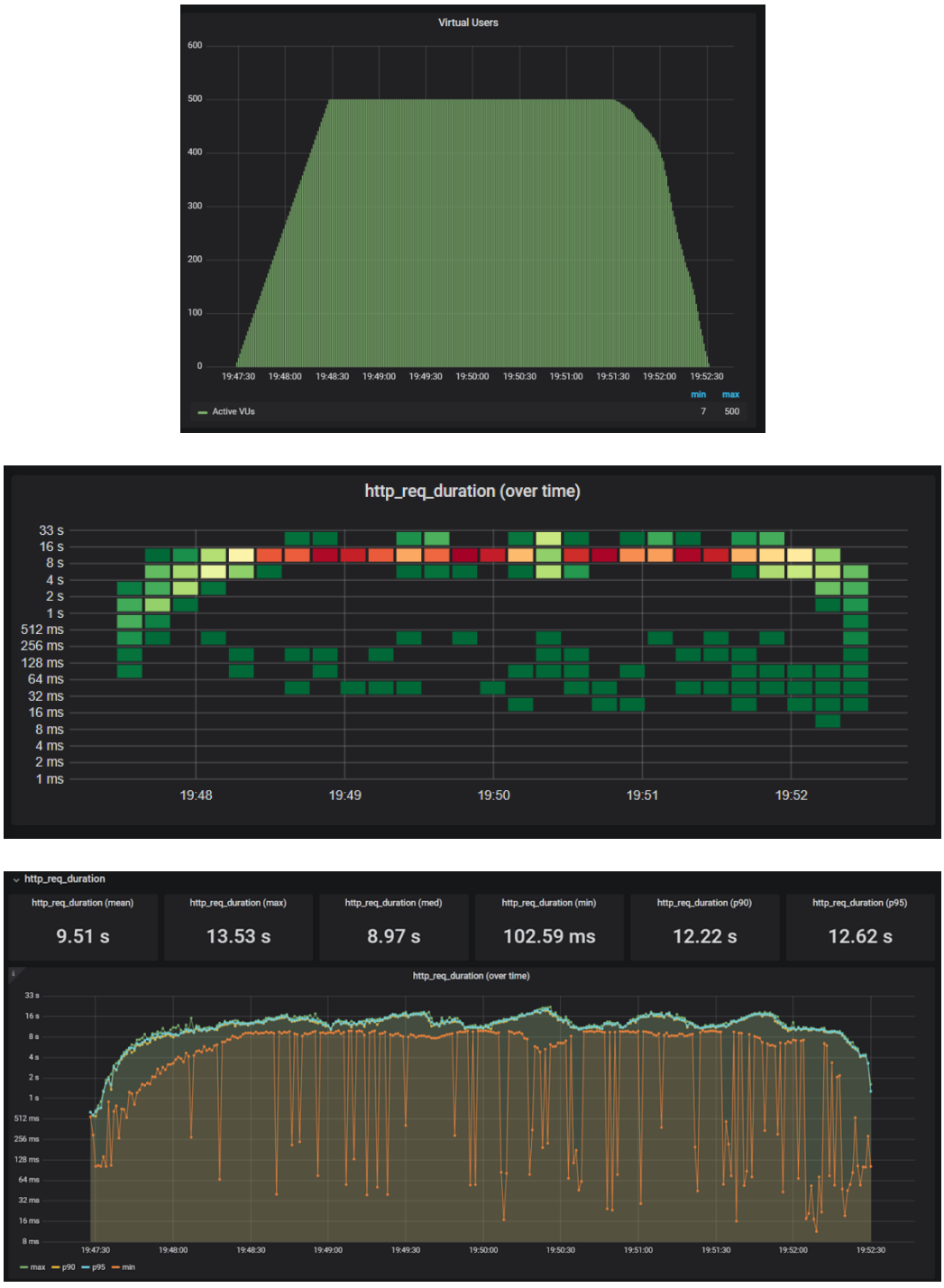




Nota. Generación en MySQL Aurora 100 usuarios

Figura 36

Resultados con 500 usuarios virtuales por 5 minutos



Nota. Generación en MySQL Aurora 500 usuarios

Anexo 13. Enlace del repositorio del proyecto

Repositorio del proyecto en GitHub: https://github.com/jact1001/forms