Pontificia Universidad Javeriana Cali
Facultad de Ingeniería.
Maestría en Ingeniería de Software
Proyecto de Grado.

Comparación entre un framework y una herramienta Low-Code para el desarrollo de un frontend para un backend ya implementado

Juan Pablo Aragón Alzate

Director(a): MSc Juan Pablo García Cifuentes

28 de Noviembre de 2024

Santiago de Cali, 28 de Noviembre de 2024.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software.
Cali.

Cordial Saludo.

Por medio de la presente hago constar que en mi calidad de director de trabajo de grado he revisado el proyecto titulado "Comparación entre un framework y una herramienta Low-Code para el desarrollo de un frontend para un backend ya implementado" realizado por el estudiante de Magister en Ingeniería de Software Juan Pablo Aragón Alzate (cod: 8911921), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,

MSc Juan Pablo García Cifuentes

Santiago de Cali, 28 de Noviembre de 2024.

Señores
Pontificia Universidad Javeriana Cali
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software
Cali.

Cordial Saludo.

Me permito presentar a su consideración el proyecto de grado titulado "Comparación entre un framework y una herramienta Low-Code para el desarrollo de un frontend para un backend ya implementado" con el fin de cumplir con los requisitos exigidos por la Universidad y para que sea sometido a revisión del jurado y cumpla su aprobación, para conseguir posteriormente el título de Magister en Ingeniería de Software.

Atentamente,

Juan Pablo Aragon

Juan Pablo Aragón Alzate
Código: 8911921

Ficha Resumen
Trabajo de Grado Maestría en Ingeniería de Software

TíTULO: Comparación entre un framework y una herramienta Low-Code para el desarrollo de un frontend para un backend ya implementado

1. Énfasis: Ingeniería de Software
2. Área de trabajo: Ingeniería de Software
3. Tipo de proyecto: Aplicado
4. Estudiante: Juan Pablo Aragón Alzate
5. Correo electrónico: jparagon2021@gmail.com
6. Dirección y teléfono: calle 34 #96-79 3043393989
7. Director: Juan Pablo Garcia Cifuentes
8. Vinculación del director: Planta
9. Correo electrónico del director: jpgarcia@javerianacali.edu.co
10. Palabras clave(al menos 5): Low-Code, No-Code, Gestión de reservas, Desarrollo ágil, Back-end, Front-end
11. Fecha de inicio: 13/03/2024
12. Evaluador 1
13. Evaluador 2

Agradecimientos

Quiero expresar mi más sincera gratitud a todas las personas e instituciones que hicieron posible la realización de este trabajo de grado.

A mi padre, **Javier Aragón**, y a mi madre, **Adriana María Alzate**, por impulsarme a asumir el reto de la maestría, por su amor incondicional y por su apoyo constante. Su confianza en mí y su motivación fueron esenciales para superar cada reto. A mi hermano, **Steve Agudelo**, gracias por estar siempre presente y por ayudarme con la solución de dudas técnicas a lo largo de este proceso.

A mi pareja, **Paula Tatiana Naranjo**, por su invaluable acompañamiento a lo largo de este camino. Gracias por estar a mi lado, por escucharme pacientemente incluso cuando el tema podía parecer complicado, y por tu apoyo práctico en la implementación y pruebas de la aplicación. Tu compromiso y dedicación marcaron una gran diferencia en este proyecto.

A mi compañero de trabajo, **Jhonny Agudelo**, por su valioso apoyo en el desarrollo del proyecto. Tu colaboración y disposición fueron clave para alcanzar los objetivos planteados.

A **Daniela Burbano**, por su apoyo esencial en las pruebas y en la definición del alcance de la aplicación. Tu disposición para colaborar y tu aporte en la estructuración de este proyecto fueron de gran valor.

Al **Club Campo Verde** y a su administrador **Juan Camilo Pacheco**, por abrirme las puertas y permitirme desarrollar e implementar mi idea en sus instalaciones. Su apoyo y colaboración en la definición del alcance y comportamiento esperado en el ámbito del tenis fueron invaluables.

Finalmente, al director de este trabajo de grado, **Juan Pablo García Cifuentes**, por su acompañamiento constante, sus consejos y su guía en cada etapa del desarrollo de este proyecto. Su experiencia y orientación fueron clave para alcanzar los resultados esperados.

A todos ustedes, gracias por creer en este proyecto y en mí. Este logro es el resultado de un esfuerzo colectivo que siempre llevaré en mi corazón.

Resumen

El trabajo aborda la necesidad de implementar un sistema de gestión de reservas para clubes de tenis debido al aumento en la demanda de canchas y los desafíos asociados con la gestión manual de reservas. Se describe cómo la falta de acceso en tiempo real a la información y la asignación manual de canchas afectan la experiencia del usuario y los ingresos del club. Se propone como alternativa el uso de herramientas Low-Code No-Code para el desarrollo frontend de la aplicación, con el objetivo de mejorar la accesibilidad, eficiencia y experiencia de los usuarios, y de lanzar al mercado un producto robusto en poco tiempo y a bajo costo. Las preguntas planteadas incluyen la identificación de herramientas disponibles en el mercado, la evaluación de criterios para seleccionar una herramienta y cómo desarrollar un frontend para una aplicación con backend ya desarrollado. Los objetivos del proyecto son diseñar y comparar la implementación de dos interfaces de usuario (frontend) utilizando una herramienta low-code y un framework de desarrollo frontend, conectándolas con un sistema de gestión (backend) desarrollado de manera convencional. Se espera definir criterios para evaluar el desarrollo frontend, tomar una decisión sobre la mejor herramienta, crear un prototipo funcional de la aplicación y recopilar comentarios de usuarios finales. En resumen, el trabajo busca abordar los desafíos de gestión de reservas en clubes de tenis mediante el uso de herramientas Low-Code No-Code, con el objetivo de mejorar la experiencia del usuario y la eficiencia operativa del proceso, a su vez evaluando los beneficios y limitantes del uso de estas herramientas.

Palabras Clave: Low-Code, No-Code, Aplicación de reservas, angular, desarrollo

Abstract

The work addresses the need to implement a reservation management system for tennis clubs due to the increasing demand for courts and the challenges associated with manual reservation management. It describes how the lack of real-time access to information and the manual assignment of courts negatively affect user experience and club revenue. As an alternative, the use of Low-Code No-Code tools is proposed for frontend development of the application, aiming to improve accessibility, efficiency, and user experience, as well as to launch a robust product to market quickly and at a low cost. The questions posed include identifying available tools in the market, evaluating criteria for tool selection, and determining how to develop a frontend for an application with an already developed backend. The project objectives are to design and compare the implementation of two user interfaces (frontend) using a low-code tool and a frontend development framework, connecting them to a management system (backend) developed conventionally. The work aims to define criteria for evaluating frontend development, make a decision on the best tool, create a functional prototype of the application, and gather feedback from end users. In summary, the work seeks to address the reservation management challenges in tennis clubs through the use of Low-Code No-Code tools, with the objective of improving user experience and operational efficiency in the process, while also evaluating the benefits and limitations of using these tools.

Keywords:
Low-Code, No-Code, Appointment Manager app, angular, development

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Hace no mucho tiempo atrás, para desarrollar una aplicación, por más sencilla que fuera, se requería no solo de contar con un equipo especializado en el tema, lo que de por sí es costoso, sino también de un tiempo mínimo para poder llevar la aplicación al mercado; Este tiempo mínimo podía generar que los competidores presentaran una solución antes al mercado, impactando de forma negativa el posicionamiento y la participación de la aplicación. De igual forma, ante los elevados costos de llevar a cabo un desarrollo, muchas ideas podían quedar rezagadas por el simple hecho de no contar con los recursos suficientes para salir al mercado.

En este contexto, gracias a los avances tecnológicos han aparecido herramientas que tienen como objetivo el desarrollo de una aplicación sin que sea necesario tener conocimientos avanzados en ingeniería de software. Haciendo uso de las tecnologías Low-Code y No-Code cualquier idea podría llevarse al mercado en un tiempo y conocimiento inferior al requerido realizarlo de forma convencional, a código puro.

Cuando hablamos de Low-Code nos referimos a una tecnología que proporciona un entorno no solo para desarrollar si no también para desplegar aplicaciones escribiendo muy pocas líneas de código. Por otro lado, en el caso de No-Code, la tecnología va un poco más allá y nos proporciona, de igual manera, un entorno para desarrollar y desplegar una aplicación, pero sin requerir una sola línea de código.

El crecimiento de este tipo de tecnologías, que para el 2023 era de 26.9 billones de dólares según Gartner (2022), ha generado un gran impacto en la industria tecnológica, cambiando la forma en la que se entiende el desarrollo de software, y así mismo a nivel global, llegando incluso a pensar que podrían sustituir el proceso actual del desarrollo de software.

La simplicidad a la que estas herramientas han reducido el desarrollo del código genera múltiples beneficios, entre los que se encuentra la reducción del tiempo de desarrollo en casi un 90 %, según Redhat (2018), lo que permite llevar ideas más rapidez y económicas al mercado, así como probar conceptos sin la carga de una gran inversión económica detrás.

De igual forma, la curva de aprendizaje para una tecnología Low-Code No-Code es mucho mas corta comparada con la del desarrollo tradicional. Esta facilidad ha generado que cada vez más empresas pequeñas o también personas sin mucho presupuesto o conocimiento puedan sacar ideas disruptivas al mercado, algo que era casi imposible 10 años atrás.

Teniendo como base los beneficios ya mencionados y el crecimiento del uso de estas tecnologías, en el presente trabajo se buscó utilizar y poner a prueba las bondades de las herramientas Low-Code No-Code para desarrollar una aplicación de gestión de reservas de cancha de tenis. Para poder comparar su uso frente al de herramientas un desarrollo convencional, se utilizaron dos tecnologías para desarrollar la interfaz de usuario, una herramienta low-code y un framework de desarrollo

frontend, ambas conectándose a un backend ya desarrollado, y se compararon bajo unos criterios específicos.

La aplicación de gestión de reservas de cancha de tenis tiene como objetivo primordial mejorar la experiencia de los usuarios de un club de tenias al permitirles consultar la disponibilidad en tiempo real, eliminando la necesidad de procesos manuales, como llamar al administrador del club. Además, facilitará la gestión integral de reservas, al permitir a los usuarios crear, modificar y eliminar reservas de manera eficiente. Asimismo, se enfocará en asignar automáticamente las canchas a las reservas, con el objetivo de minimizar la división de una reserva entre múltiples canchas y optimizar así la experiencia de juego de los usuarios.

1.1. Definición del problema

1.1.1. Planteamiento del problema

Según El País (2023), para el 2023 el tenis era el tercer deporte más popular a nivel mundial con aproximadamente 1000 millones de espectadores y alrededor de 300 millones de practicantes. En Colombia, si bien el tenis es un deporte ampliamente practicado por una parte de la población, al no ser tan popular y económicamente accesible, el servicio de alquiler de canchas es prestado, en su mayoría, por clubes privados.

Esta popularidad, creciente a nivel nacional desde la pandemia del 2021,según lo conversado con un administrador de un club en la ciudad de Cali, ha llevado a un aumento significativo en la demanda de la utilización de canchas de tenis. Sin embargo, muchos de los clubes de tenis regionales siguen administrando de forma manual las reservas de las instalaciones, generando así una serie de desafíos que afectan tanto a los usuarios como a la eficiencia operativa del proceso.

La necesidad de comunicarse con un funcionario para conocer la disponibilidad de las canchas ha creado un proceso burocrático y lento, que en muchas ocasiones desincentiva a los usuarios de generar una reserva. Los usuarios deben consultar con un tiempo de anticipación teniendo en cuenta que el tiempo de respuesta por parte del administrador es incierto, es posible que este brinde una respuesta después de la hora deseada para la reserva. De igual forma, si un usuario está interesado en una hora especifica, que en el momento no se encuentra disponible, pero posteriormente se libera, no hay un mecanismo para notificar el usuario y asignarle el espacio. Como se puede observar, la falta de acceso a la información en tiempo real impacta de forma directa el numero de reservas generadas en un club, en consecuencia afecta los ingresos del mismo.

Adicionalmente, la asignación manual de las canchas presenta desafíos logísticos que pueden afectar la experiencia de usuario, con la posibilidad de generar solapamiento de reservas o una distribución ineficiente de las reservas que limite la utilización de las instalaciones, un recurso valioso en el contexto de que es limitado y es el que genera los ingresos representativos del club. Ante este escenario, es casi que indispensable la implementación de un sistema de gestión de reservas para mejorar la accesibilidad, eficiencia y experiencia de los usuarios en el club de tenis.

Llevar a cabo el desarrollo de la aplicación de forma tradicional implica por lo general un mayor tiempo de desarrollo, debido a la necesidad de escribir código, definir la arquitectura técnica de

la aplicación, patrones de diseño a utilizar entre otras muchas decisiones y procesos requeridos para generar una aplicación mantenible y escalable. Este proceso requeriría ademas de ingenieros especializados en diferentes áreas para poder hacer el proceso mas rápido y seguro, por el contrario, si lo realiza solo un ingeniero, el tiempo y conocimiento necesario hace que el proyecto pueda ser mas demorado, inestable y costoso.

Por otro lado, las herramientas Low-code ofrecen una alternativa que promete reducir tanto los tiempos como los costos, al permitir construir aplicaciones mediante interfaces visuales y componentes predefinidos que facilitan tareas completas. De este modo, resulta posible llevar al mercado una idea en menor tiempo, iterando rápidamente en función de la retroalimentación de los usuarios.

Por lo anterior, en este trabajo de grado se propone comparar el desarrollo tradicional con el desarrollo en herramientas Low-Code, con el fin de evaluar el impacto en tiempos de desarrollo, costos, y la agilidad para responder a las necesidades del mercado. Para ello, se optó por desarrollar de forma paralela la aplicación haciendo uso también de un framework de desarrollo frontend.

1.1.2. Formulación del problema

¿Cuáles son las herramientas disponibles en el mercado para realizar un desarrollo frontend con Low-Code o No-Code?

¿Bajo cuales criterios se debería evaluar una herramienta Low-Code No-Code para el desarrollo frontend de una aplicación con backend ya desarrollado?

¿Cómo desarrollar un frontend para una aplicación que ya cuenta con un backend desarrollado?

¿Son las herramientas Low-Code No-Code una buena alternativa para sacar aplicaciones robustas al mercado en poco tiempo y a bajo costo?

1.2. Objetivos del proyecto

1.2.1. Objetivo General

Comparar la implementación de dos interfaces de usuario (frontend) utilizando herramientas de desarrollo Low-Code No-Code, conectándolas con un sistema de gestión (backend) desarrollado de manera convencional.

1.2.2. Objetivos específicos

1. Identificar dos herramientas Low-Code No-Code presentes en el mercado para el desarrollo frontend de una aplicación

2. Identificar los criterios bajo los cuales comparar las herramientas utilizadas para el desarrollo frontend

3. Diseñar y desarrollar dos aplicaciones frontend conectadas con un backend ya funcional

4. Evaluar las aplicaciones bajo los criterios definidos

1.2.3. Resultados esperados

1. Definición de los criterios para la evaluación de un desarrollo frontend
2. Una decisión sobre la mejor herramienta, entre las utilizadas, para desarrollo Low-Code No-Code de aplicaciones frontend
3. Un prototipo funcional de una aplicación para la asignación de reservas de canchas de tenis
4. Recopilación de los comentarios de un usuario final sobre el prototipo

1.3. Delimitaciones y alcances

- Se realizarán dos desarrollos, uno con la herramienta low-code y otro con el framework de desarrollo
- El prototipo será de una aplicación Web.
- El prototipo de la aplicación debe:
  1. Mostrar a los usuarios las horas disponibles para reservar una cancha en tiempo real
  2. Permitir las siguientes operaciones con una reserva:
     a) Crear una reserva
     b) Eliminar una reserva
     c) Modificar una reserva
  3. Asignarle una cancha a la reserva, buscando minimizar la asignación de una reserva a múltiple canchas
  4. Mostrarle al administrador del club las reservas y horas disponibles
  5. Soportar hasta mínimo 12 canchas
  6. Permitir al usuario configurar el rango de tiempo en que cada cancha está disponible
  7. Permitir que el club defina las fechas habilitadas para que sus usuarios realicen reservas por la aplicación

1.4. Justificación del trabajo de grado

En Cali son pocos los clubes de tenis que cuentan con una herramienta digital por la cual los usuarios puedan gestionar reservas. Este “atraso” tecnológico genera varios inconvenientes tanto para los usuarios como para el mismo club. Entre los inconvenientes mas comunes, expresados por los usuarios, está que el proceso para una reserva se vuelve lento y burocrático, para el usuario es imposible saber las horas disponibles al consultarle al club, se pierde tiempo esperando una respuesta e inclusive esperando la confirmación de una reserva.

Del lado del club también se presentan diferentes problemas, como la insatisfacción de los usuarios por el tiempo de respuesta, la doble asignación de una reserva, el perder reservas por responder muy tarde a una solicitud, y por último, el no tener la capacidad de asignar nuevamente una reserva cancelada con muy poco tiempo de anticipación.

Si bien existen algunas aplicaciones en el mercado que se pueden utilizar para la gestión de reservas, estas carecen de ciertas características propias del negocio del tenis, como por ejemplo que el usuario pueda reservar más de una hora desde una sola solicitud y que sea la aplicación, según la disponibilidad, le asigne una cancha evitando generar espacio muertos entre las reservas de una cancha. De igual forma, para los usuarios es muy valioso no tener que estar revisando múltiples veces la aplicación verificar si el espacio que deseaban reservar, pero inicialmente no estaba disponible, es liberado en algún momento, si no que sean notificados y se les asigne la hora automáticamente, una funcionalidad que no se encontró en ninguna aplicación que opere localmente.

Ante estos inconvenientes nace la idea de desarrollar una aplicación que permita la gestión de las reservas en tiempo real, con características que se asemejen más al comportamiento actual de los clubes de tenis. Al ser un proyecto personal, no cuenta con un gran presupuesto, sin embargo, se busca contar con una versión inicial cuanto antes, probarla en el mercado, y con base a los comentarios recopilados realizarle mejoras, y así, cíclicamente hasta dar con un producto estable y ampliamente aceptado por la comunidad del tenis.

Desarrollar la aplicación de forma tradicional, requeriría de un amplio conocimiento en diferentes áreas del desarrollo del software o por el contrario de un equipo multidisciplinar lo que de por si elevaría los costos y el tiempo para poder tener la aplicación funcionando en el mercado.Aquí es donde las herramientas Low-Code/No-Code entran a jugar un papel primordial,consistente en permitir llevar estas ideas, que cuentan aun con mucha incertidumbre, al mercado sin la necesidad de contar con un equipo de TI especializado o sin un gran presupuesto.

El presente trabajo de grado busca no solo desarrollar la aplicación apoyándose en las herramientas Low-Code/No-Code, sino que también busca analizar las herramientas que ofrece actualmente el mercado, y con una de ellas generar una comparación con un desarrollo realizado de forma convencional, bajo los criterios sobre los cuales usualmente se evaluaría una aplicación desarrollada de forma convencional, dejando claro los beneficios y desventajas de utilizarlas.

1.5. Metodología de la investigación

Para poder cumplir con los objetivos específicos, y por consiguiente, con el objetivo general de implementar dos interfaces de usuarios y compararlas, el trabajo se desarrolló bajo la metodología Ágil basada en Scrum, que mediante un desarrollo incremental nos permite generar pequeños avances funcionales que nos acercan cada vez mas a la solución planeada. El objetivo del desarrollo incremental es construir el software en etapas y en cada paso agregar funcionalidades nuevas.

Basándonos en el marco de gestión de proyectos de metodología ágil, Scrum, aplicamos las siguientes fases:

- Plantación del sprint: En esta etapa se definen los impactos funcionales que se van a realizar

a lo largo del sprint.

- Sprint: Periodo del tiempo en donde se ejecutara lo planeado en la etapa inicial.
- Revisión del sprint: En esta etapa se revisa lo trabajado en el sprint para saber si se cumplió o no con lo planeado y cual fue el incremento funcional.
- Retrospectiva del Sprint: En esta etapa se levantan las lecciones aprendidas durante el sprint, al igual que los impedimentos que hubo y como se pueden mejorar.

Para cumplir con lo planteado en el proyecto de grado se realizaron 6 Sprints de 3 semanas cada uno, en la Figura 1.1 se puede observar lo planeado para cada uno de ellos.

Sprint 1	Selección de herramientas Low-Code No-code para el desarrollo del front	Semana 1	Semana 3
	Identificar los criterios con los cuales comparar una herramienta para el desarrollo front		
Sprint 2	Implementar inicio de sesion	Semana 4	Semana 6
	Implementar registro de usuarios		
Sprint 3	Implementar Dashboard para usuarios NO administradores	Semana 7	Semana 9
	Implementar registro de Reservas		
	Implementar cancelación de reservas		
Sprint 4	Implementar Dashboard para usuario administrador	Semana 10	Semana 12
	Implementar gestion de reservas para usuario administrador		
Sprint 5	Comparación entre ambos desarrollos front	Semana 13	Semana 15
	Pruebas a los desarrollos		
Sprint 6	Construcción del documento final	Semana 16	Semana 18

Figura 1.1: Planeación Sprints. Elaboración propia

1.6. Resultados obtenidos

El proyecto alcanzó los siguientes resultados clave:

- **Desarrollo de la aplicación funcional**: Se logró desarrollar una aplicación completamente funcional utilizando dos enfoques tecnológicos: Angular y WeWeb. Ambas versiones de la aplicación cumplían con los requisitos básicos, pero se identificaron diferencias significativas en cuanto a desempeño, mantenibilidad,costo y desplegabilidad.
- **Evaluación comparativa entre Angular y WeWeb**: A través de la comparación entre ambas herramientas, se obtuvo una comprensión clara de las fortalezas y limitaciones de cada enfoque. WeWeb permitió un desarrollo más rápido y accesible, ideal para soluciones con requerimientos simples o para proyectos con tiempos de desarrollo limitados. Por otro lado, Angular ofreció mayor control sobre la personalización y mejor rendimiento en escenarios complejos, con una estructura más robusta y adaptable a largo plazo.
- **Mejoras en la experiencia de usuario**: A través de la implementación de la aplicación, se mejoró la experiencia de usuario (UX) mediante la integración de funcionalidades intuitivas

y adaptadas a las necesidades del público objetivo. Ambos enfoques permitieron un diseño visualmente atractivo y una interfaz amigable.

- **Resultados de pruebas funcionales y de rendimiento**: Se realizaron pruebas exhaustivas para evaluar el rendimiento de ambas aplicaciones. Aunque WeWeb ofreció tiempos de desarrollo rápidos, Angular mostró un desempeño superior en cuanto a estabilidad y capacidad para manejar tareas más exigentes, como integraciones con bases de datos y procesos de lógica compleja.
- **Identificación de áreas de mejora**: Durante el proceso de implementación, se identificaron áreas de mejora en ambas plataformas. En WeWeb, se destacaron limitaciones en términos de personalización avanzada y escalabilidad, mientras que en Angular se evidenció la necesidad de un mayor esfuerzo en el desarrollo inicial, pero con mayores ventajas a largo plazo en cuanto a mantenimiento y expansión.

En conclusión, los resultados obtenidos reflejan que, aunque WeWeb es una herramienta adecuada para proyectos de menor complejidad y desarrollo rápido, Angular es más adecuado para proyectos que requieren una mayor personalización, control y escalabilidad.

Capítulo 2

Marco de referencia

2.1. Marco Teórico

2.1.1. Bases Teóricas

Plataformas Low-Code y No-Code

Vincent et al. (2020) define a las plataformas Low-Code (LCP por sus siglas en ingles) como aplicaciones que soportan el desarrollo, despliegue, ejecución y gestión rápida de aplicaciones, utilizando abstracciones de programación declarativas y de alto nivel. Por otro lado, para Hylton et al. (2021) una plataforma No-Code permite desarrollar aplicaciones sin escribir una sola linea de código.

A nivel general, tanto las herramientas No-Code como las Low-Code ofrecen una experiencia basada en técnicas de ”drag-and-drop”por encima de los lenguajes de programación tradicionales Università et al. (2023).

La utilización de estas plataformas tiene como objetivo generar un aumento de la productividad y una reducción en el costo del desarrollo y mantenimiento de una aplicación, mejorando así la habilidad de las compañías para adaptar su sistema a los constantes cambios de requerimientos por parte del mercado Bock and Frank (2021b) . Así mismo Moskal (2021) indica que dentro de los beneficios de estas plataformas también se encuentra la flexibilidad y posibilidad de desarrollar herramientas con un esfuerzo de implementación y mantenimiento menor.

Transformación digital

Para George and Paul (2020) “la transformación digital lleva a las organizaciones a contemplar la implementación de diversas iniciativas tecnológicas. Sin embargo, también implica una serie de acciones en ámbitos sociales, culturales, políticos, económicos, ecológicos y normativos”.

De acuerdo con Okano et al. (2021), estas transformaciones conllevan la reconfiguración del modelo de negocio y el surgimiento de nuevas empresas. Además, en general, dan lugar al establecimiento de formas innovadoras de comerciar y comunicarse.

Según BCG (2020) las compañías pueden esperar un aumento en sus margenes de entre 12 % al 20 % al utilizar palancas digitales, las compañías que se resistan a la transformación digital poco a poco se verán apartadas del mercado.

Frontend y Backend

Amazon Web Services (2023) define el frontend como aquello que ven los usuarios, en donde

se incluyen elementos como los botones, casillas de verificación, gráficos y mensajes de textos. En otras palabras, es el punto de contacto que permite a los usuarios interactuar con la aplicación. Hay tres lenguajes de computación principales para el desarrollo:

1. El lenguaje HTML que define la estructura del frontend
2. La hoja de estilos CSS que define el estilo y diseño de la aplicación.
3. Lenguaje de programación, por lo general JavaScript, para agregar una capa de funcionalidad dinámica

Por otro lado el back-end, a veces denominado servidor, son los datos y la infraestructura que permite que las aplicaciones funcionen. Cuando el usuario realiza una petición por medio del frontend, el backend procesa la solicitud y devuelve una respuesta, para esto usualmente el servidor interactúa con:

1. Base de datos para la recuperar o modificar información
2. Micro servicios para la realización de tareas especificas
3. APIs externos ya sea para recopilar información adicional o realizar funciones adicionales

API Rest

Para poder entender que es una API Rest primero se definirá que es un API. Las API, según la documentación de Amazon Web Services (2023), son mecanismos que permiten a dos componentes de software comunicarse entre si mediante un conjunto de definiciones y protocolos. Por otro lado, REST significa transferencia de estado representacional y es un estilo arquitectónico para diseñar sistemas de software distribuidos que fue propuesto por Roy fielding. Este hace uso de operaciones HTTP estándar como GET, POST, PUT, DELETE, para realizar acciones en recursos identificables mediante URLs.

Teniendo claro la definición de los términos por separado, según REDHAT (2023) “una API Rest es una interfaz de programación de aplicaciones (API o API web) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful”.

Productividad en el desarrollo del software

Las herramientas Low-Code No-Code prometen mejorar la productividad en el desarrollo del software. Para Varajão et al. (2023) la productividad en el ámbito de las tecnologías de desarrollo de software se puede definir como la eficiencia lograda en la producción de bienes o servicios de software. Por lo general, la productividad se expresa como una medida entre la relación de las salidas, las entradas y el tiempo. El diccionario de Cambridge define la productividad como “La tasa a la que una empresa o país fabrica bienes, generalmente juzgada en relación con el número de personas y la cantidad de materiales necesarios para producir los bienes”.

En su articulo, Varajão et al. (2023) mediante un experimento concluye que las herramientas Low-Code No-Code muestran un aumento considerable de la productividad sobre las herramientas basadas en código, este impacto se traduce en menores tiempo de entrega y por lo tanto, en mayor utilidades a la hora de realizar un proyecto. Por ejemplo, en su artículo, Sanchis et al. (2020) señala que según una investigación realizada por Forrester, se comprobó que las plataformas Low-Code aceleran el desarrollo entre 5 y 10 veces más rápido. Asimismo, tras comparar herramientas Low-Code y basadas en código, Trigo et al. (2022) concluye que la productividad en las herramientas Low-Code es aproximadamente tres veces mayor que en las tecnologías basadas en código (Code-based).

2.2. Estado del Arte

A continuación se presentará primero las herramientas No-Code y Low-Code para el desarrollo frontend presentes en el mercado actual, con un pequeño análisis de las características que ofrece cada una. Posteriormente, se investigarán y analizara trabajos ya realizados del impacto del uso de herramientas No-Code Low-Code en el proceso del desarrollo del software. Por último, se hará un análisis de las aplicaciones que se encuentran actualmente disponibles para la gestión de reservas de canchas de tenis y como la aplicación en desarrollo puede ganar terreno en el mercado.

Entre las herramientas más populares actualmente para el desarrollo Low-Code No-Code de aplicaciones frontend se encuentran:

1. WeWeb
2. Bubble
3. Adalo
4. Oracle Apex

Weweb (`https://www.weweb.io/`) es una plataforma No-Code para la construcción y desarrollo frontend que se puede utilizar sobre cualquier back-end. Esta aplicación asegura que haciendo uso de ella se pueden construir aplicaciones 10 veces mas rápido. Dentro de sus principales características se encuentran:

1. Diseño web mediante la técnica de drag-drop
2. Fácil despliegue de la aplicación desarrollada
3. Conexiones con cualquier tipo de API
4. Escalabilidad de la aplicación mediante diferentes paquetes
5. Trabajo en equipo sobre un mismo proyecto, pagando por cada usuario extra
6. Enfoque especializado en el desarrollo frontend

Bubble (https://bubble.io/) a diferencia de Weweb, no solo soporta el desarrollo frontend de una aplicación si no que también soporta el desarrollo back-end. Entre sus promesas se encuentra poder construir cualquier aplicación sin código, solo haciendo uso de drag-and-drop. Entre sus características principales se encuentra:

1. Permite la creación de aplicaciones multiplataformas
2. Permite la creación de aplicaciones multilenguaje de forma automática
3. Promete una escalabilidad sin cambios técnicos.
4. Se encarga del despliegue y el hosting de la aplicaciones.
5. Ofrece integraciones para facilitar procesos como el de pagos o envíos de correos
6. Diferentes personas trabajen al tiempo sobre un proyecto

Adalo (https://es.adalo.com/), al igual que las otras aplicaciones, permite el desarrollo de aplicaciones robustas, tanto frontend como back-end. Entre sus principales características encontramos:

1. Permite publicar las aplicaciones en App stores
2. Diseño completamente visual y en tiempo real
3. Curvas de aprendizaje muy bajas
4. Una verdadera plataforma No-Code
5. Permitir realizar todo el desarrollo de la aplicación desde la plataforma.
6. Plantillas para un desarrollo mas ágil

Oracle Apex (https://apex.oracle.com/es/) es una herramienta Low-Code para crear aplicaciones web y móviles enfocadas en aplicaciones empresariales. Ofrece un editor visual de drag-and-drop para interfaces gráficas y se integra con otras tecnologías de oracle como la base de datos SQL de Oracle. Entre sus principales características se encuentran:

1. Crear aplicaciones móviles y en la nube
2. Convertir hojas de calculo en aplicaciones
3. Formularios modernos
4. Desarrollo sencillo y seguro

Ante el gran abanico de plataformas Low-Code No-code presentes en el mercado, para tomar una decisión sobre cual utilizar se debe de evaluar cada uno bajo unos criterios específicos. En su articulo "Navigating the Low-Code Landscape: A Comparison of Development Platforms" Kirchhof et al. (2023) analiza y compara 21 plataformas diferentes Low-Code (LCDP) bajo ciertos criterios que considera importantes, con el fin de proporcionar orientación a los expertos en el dominio y a los investigadores sobre las características relevantes de estas plataformas.

De igual forma Bock and Frank (2021a) en su articulo "In Search of the Essence of Low-Code: An Exploratory Study of Seven Development Platforms" presenta un estudio exploratorio de siete plataformas de desarrollo Low-Code con el objetivo de entender su esencia y evaluarlas críticamente en relación con la investigación en desarrollo de sistemas de información. Utilizando un marco de análisis que cubre varios criterios relacionados con el desarrollo de sistemas de información profesional, se identifican las características comunes, ocasionales y poco frecuentes de estas plataformas.

Analizando el tema de los beneficios e impactos del uso de tecnologías Low-Code No-Code, se encontraron los siguientes trabajos que abordan dicho tema.

El artículo "Benefits and limitations of using low-code development to support digitalization in the construction industry", deMartinez and Pfister (2023), explora cómo las herramientas No-Code pueden promover la digitalización en la construcción. Utilizando una metodología híbrida de investigación, el estudio analiza cómo estas herramientas pueden mejorar procesos específicos en el sector, destacando tanto sus ventajas como sus limitaciones. Se recomienda adoptar un enfoque estratégico para maximizar su potencial a nivel organizacional y se proponen áreas adicionales para futuras investigaciones, incluyendo la integración con otras tecnologías relevantes para la industria de la construcción.

Adicionalmente, en el articulo "Characteristics and Challenges of Low-Code Development: The Practitioner's Perspective", por Luo et al. (2021), examinan las características y desafíos del desarrollo con herramientas Low-Code (LCD por sus siglas en ingles) a través de datos de comunidades en línea como Stack Overflow y Reddit. Se recopilan y analizan publicaciones relacionadas con LCD, identificando sus ventajas, limitaciones y percepciones de los practicantes. Se concluye que el LCD proporciona una interfaz gráfica para desarrollar aplicaciones con poco o ningún código, siendo especialmente útil en dominios con necesidades de automatización. Sin embargo, existen opiniones divididas sobre sus ventajas y desventajas. Se destaca la importancia de definir claramente los términos relacionados con el LCD y evaluar su idoneidad para proyectos específicos.

En cuanto a las aplicaciones que se encuentran en el mercado para la gestión de reservas de canchas de tenis, se dividiran en nacionales e internacionales, recalcando sus características y también las limitantes con las que cuentan.

Reservas de canchas alcaldía de Barranquilla (`https://appbaq.barranquilla.gov.co:8989/canchas/`), es una plataforma particular, debido a que solo se usa para reservar canchas publicas de la ciudad de Barranquilla Colombia. Dentro de sus funcionalidades se encuentran:

1. Permite registrar una reserva
2. Permite consultar una reserva

Dentro de las limitaciones se encuentran

1. Solo permite el registro de una hora a la vez, lo que puede generar que si el usuario requiere dos horas, ambas no cuenten con la misma cancha disponible.

2. El usuario es el que escoge la cancha, esto puede generar una asignación ineficiente de la ocupación de las canchas.

3. Solo está enfocado en la ciudad de Barranquilla

4. No cuenta con la posibilidad de la creación de usuarios, por lo que para cada reserva se deba ingresar información básica.

5. El proceso de creación de una reserva no ofrece una experiencia de usuario óptima.

6. No permite realizar la cancelación de una reserva desde la plataforma, toca comunicarse con un funcionario.

Easycancha(https://www.easycancha.com/) es quizás la aplicación mas fuerte en la gestión de canchas de tenis con presencia en Colombia aunque va mas orientada a las canchas de padel, pero poco a poco va ganando terreno en las canchas de tenis. La aplicación es chilena pero está presente en 7 países, entre los que se encuentraN Brasil, Argentina, Colombia, Ecuador, Usa, México y Chile.

Dentro de sus funcionalidades para la reservas de canchas de tenis se encuentran:

1. Permite la creación de usuarios en la aplicación.

2. Es una aplicación multideporte.

3. Cuenta con múltiples clubes inscritos, por lo que un usuario puede elegir entre diferentes instalaciones disponibles.

4. Permite que algunas instalaciones solo estén disponibles para socios.

5. Tiene incorporado el proceso de pago de la reserva.

6. La interfaz es amigable con el usuario.

7. Entre los usuarios de la aplicación pueden cuadrar partidos.

8. Cuenta con un ranking para los usuarios de la aplicación que jueguen partidos entre ellos.

Dentro de las limitantes se encuentran:

1. Solo se pueden realizar reservas de una hora definida, por lo que el proceso para reservar mas de una hora es complicado para el usuario.

2. El usuario es el que escoge la cancha, esto puede generar una asignación ineficiente de la ocupación de las canchas.

Courtserve (https://courtreserve.com/) es la aplicación mas completa que se encuentra para la gestión de canchas de tenis pero actualmente solo se encuentra disponible en Estados Unidos y sus procesos se encuentran muy enfocados al mercado americano. Entre sus funcionalidades mas importantes se encuentran

1. El sistema hace automáticamente la asignación de las reservas a las canchas de tenis
2. Permite a los usuarios establecer horarios predilectos para sus reservas
3. Los clubes pueden restringir las reservas en las horas picos
4. Los usuarios pueden reservas mas de una hora, si hay disponibilidad
5. Notifica a los usuarios tiempo antes de la reserva

Dentro de las limitantes encontramos

1. La configuración es complicada
2. La interfaz no es muy amigable con los usuarios
3. Usuarios reportan casos de doble asignación de canchas
4. No cuentan con presencia fuera de los estados unidos
5. La aplicación no está optimizada para dispositivos móviles
6. está muy enfocado en la administración de un club y deja por fuera el mercado de las personas que reservan sin ser miembros de algún lugar

2.3. Resumen del capítulo

En este capítulo se revisan las principales teorías y herramientas asociadas al desarrollo de software mediante plataformas Low-Code y No-Code. Estas herramientas permiten crear aplicaciones rápidamente con poca o ninguna codificación, mejorando la productividad y reduciendo costos, como se observa en diversas investigaciones. Además, se aborda la transformación digital en las organizaciones, la importancia del frontend y backend en el desarrollo de aplicaciones, y las características de las API REST. También se destacan las ventajas de las plataformas Low-Code en términos de eficiencia y rapidez en el desarrollo, con ejemplos de herramientas como WeWeb, Bubble, Adalo y Oracle Apex. Finalmente, se examinan aplicaciones actuales para la gestión de reservas de canchas, destacando tanto sus características como sus limitaciones.

Capítulo 3

Desarrollo del Proyecto

3.1. Selección de las herramientas y criterios para compararlas

3.1.1. Selección de la plataforma low-code

Con solo realizar una consulta en Google sobre las herramientas Low-code/No-code, se se puede observar que el abanico de ofertas no es solo grande sino que también se encuentra en constante expansión. El Cuadrante Magico de Gartner, publicado en 2023, muestra 17 herramientas Low-code/No-code distribuidas en 4 cuadrantes (Lideres, Desafiantes, Jugadores de nicho y visionarios), ver Figura 3.1.

Figura 3.1: Cuadrante mágico de Gartner. Imagen tomada de Gartner (2022)

Como líder principal, tenemos a Mendix, una herramienta para desarrollar aplicaciones móviles nativas y aplicaciones web responsivas. En la Figura 3.1 podemos observar las otras plataformas

que, naturalmente, al estar enfocadas a un mercado empresarial, no ofrecen planes enfocados a pequeños desarrolladores. En muchos casos, es necesario solicitar ser contactado para poder utilizar el software, e incluso la versión gratuita está muy limitada. Herramientas tan robustas y grandes pueden no ser adecuadas para simplemente desarrollar un demo o un proyecto que quiere probar el mercado. Por tal motivo, el presente trabajo se enfocó en herramientas más pequeñas, pero que están en la capacidad de ayudarnos a desarrollar la aplicación para la gestión de reservas.

Entre las herramientas low-code para desarrollo frontend que están orientadas a pequeños desarrolladores y que ofrecen precios accesibles para aplicaciones con presupuestos limitados, se destacan las siguientes: **Weweb**, **Bubble**, **Adalo** y **Webflow**.

Vamos a comparar cada una de ellas bajo unos criterios que consideramos importantes a la hora de hacer uso de una herramienta low-code.

Costo

El primer criterio que se analizará consiste en los planes que ofrece cada una de las plataformas. En la Figura 3.2 se puede observar la comparativa entre los planes que ofrece cada plataforma y las funcionalidades que cubre cada uno.

	Oferta gratis		Starter		Pro		Posición
WeWeb	$-	- Uso de todas las funcionalidades en modo development. - Plantillas y kits de diseño - Sopporte de la comunidad	$ 39	- Despliegue de aplicacion - Dominio customizado - 50.000 visitas al mes - 1gb de almacenamiento - 200GB de ancho de banda - Exportar codigo y auto despliegue - Backups de desarrollo	$ 149	- 250.000 visitas - 10GB de almacenamiento - 400GB de ancho de banda - Memoria Cache - Acceso a paginas por roles - Produccion app staging - Production app versioning - 3 backups de produccion	1
Bubble	$-	- Version de desarrollo - Conerctor a APIS - 1 editor - 50.000 visitas/mes - 6 horas de logs	$ 29	- Despliegue de aplicacion - Dominio customizado - Control de versiones basico - 175.000 visitas - 2 dias de logs	$ 119	- 2 editores de codigo - Control de versiones premium - Autenticacion de dos pasos - 10 Ramas personalizadas - 250.000 visitas - 14 dias de logs	2
Adalo	$-	- Base de datos + colecciones - Pantallas ilimitadas - Aplicaciones de pruebas ilimitadas - 1000 acciones al mes	$ 36	- Dominio personalizado - Despliegue de aplicacion - Poder publicar en AppStore - 10.000 acciones	$ 160	- 5 aplicaciones publicadas - Integracion con xano - 10 editores de aplicaciones - 100.000 acciones mensuales	3
Webflow	$-	- 2 paginas - 20 CMS items - 50 formularios - 1GB de ancho de banda	$ 23	- 150 paginas - 2000 CMS items - 1000 formularios - 50 GB de ancho de banda	$ 39	- 300 paginas - 10000 CMS items - 2500 formularios - 100 GB de ancho de banda	4

Figura 3.2: Costo publicado para el uso de las plataformas

Teniendo en cuenta que para la familiarización del uso de la plataforma se hace uso del plan

gratuito y que por lo general para el despliegue se requiere del plan starter, comparando el costo del plan vs las funcionalidades ofrecidas, se considera que las plataformas de **Weweb** y **Bubble** son las mejores bajo este criterio.

Tamaño de la empresa

Otro criterio importante es el tamaño de la empresa, ya que esto nos puede dar una idea del crecimiento y de la inversión que hay detrás, lo que puede llevar a que la herramienta se consolide y crezca en el mercado en poco tiempo. Empresas más grandes suelen contar con mayores recursos para mantener y mejorar sus productos, proporcionar actualizaciones regulares, y ofrecer un mejor servicio al cliente. Además, su experiencia y posición en el mercado son indicativos de la confiabilidad de la herramienta, lo cual es crucial para la sostenibilidad a largo plazo del proyecto.

Cuando hablamos de tamaño de la empresa, nos referimos principalmente al capital con el que cuenta.Haciendo uso de la herramienta de Crunchbase se puede observar la siguiente información de las cuatro compañías de desarrollo low-code para frontend.

1. **Weweb:** Compañía fundada en 2017, y en la actualidad es una herramienta que cuenta con 3.1 millones de dólares en financiación y 96,299 visitas mensuales. Comparada con Bubble y Adalo, Weweb está en una etapa temprana de crecimiento. La menor cantidad de visitas sugiere que todavía está construyendo su comunidad

2. **Bubble:** Con 106.3 millones de dólares en financiación y 3,437,409 visitas mensuales, Bubble también cuenta con un respaldo financiero sólido y una comunidad de usuarios activa. Fundada en 2012, es la segunda más antigua, lo que le otorga una mayor experiencia y estabilidad en el mercado.

3. **Adalo:** Con 9.8 millones de dólares en financiación y 755,345 visitas mensuales, Adalo se posiciona como una herramienta en crecimiento. Fundada en 2018, es relativamente nueva, pero parece haber ganado una parte significativa del mercado en poco tiempo. Su financiación y visitas indican un crecimiento saludable y un potencial para seguir expandiéndose.

4. **Webflow:** Es la herramienta con mayor financiación, alcanzando los 334.9 millones de dólares. Con 10,110,939 visitas mensuales, es también la más popular de las cuatro. Fundada en 2013, Webflow ha logrado consolidarse en el mercado, ofreciendo una combinación sólida de respaldo financiero, popularidad y experiencia.

En conclusión, **Webflow** es la opción mas segura y robusta gracias a su fuerte respaldo económico, su gran base de usuarios y su tiempo en el mercado. **Adalo:** si bien no es tan grande como Bubble, viene creciendo en el mercado y ofrece una buena alternativa como herramienta en expansión. Por último, **Weweb:** se encuentran en etapas mas tempranas de desarrollo, lo que no quiere decir que no sean buenas opciones, por el contrario, pueden ser un poco más flexibles y accesibles para proyectos pequeños.

Integración con terceros

Como se mencionó anteriormente, el backend para este proyecto ya ha sido desarrollado, y la comunicación con este se realiza a través de llamadas HTTP tipo POST, GET, PUT y DELETE.

Por lo tanto, es esencial que la herramienta seleccionada permita la integración con sistemas externos, específicamente mediante una API REST. Además, su uso e interacción deben ser lo más sencillos posible para facilitar la comunicación efectiva con el backend.

Para poder comparar las 4 herramientas bajo este criterio, se realizó una revisión del proceso necesario para realizar una llamada HTTP y el manejo de la información retornada por los servicios del backend.

1. **Weweb**

    a) La forma de en la que se configuran las llamadas API es bastante sencilla y ofrece buena retroalimentación para el manejo de errores.

    b) Los flujos son sencillos de configurar y modificar.

    c) La definición y asignación de variables es sencilla.

2. **Bubble**

    a) Para realizar llamadas a través de una API REST en Bubble, es necesario instalar un plugin específico. Aunque Bubble cuenta con su propio modelo de datos que simplifica el desarrollo, en nuestro caso, ya tenemos un backend desarrollado y necesitamos comunicarnos con él mediante API REST. La configuración de estas llamadas desde el frontend en Bubble puede ser compleja y no solo requiere una buena documentación, sino también la consulta de información en foros y otros recursos.

3. **Adalo**

    a) Para poder realizar llamadas a un API externo en adalo, se debe pagar la suscripción de starter que tiene un valor de 36 dólares mensuales. Un monto que consideramos es alto para poder simplemente realizar pruebas de un prototipo.

    b) El proceso de configurar las llamadas al API Rest no es muy intuitivo, toca documentarse y buscar como hacerlo en foros. En cuanto al manejo de errores, no hay una pantalla o flujo desde donde se pueda configurar que hacer cuando hay un error.

4. **Webflow**

    a) No va tan orientado al uso de APIS externo, lo que dificulta un poco el tipo de desarrollo que se quiere realizar en este proyecto.

Soporte y documentación

1. **Weweb**

    a) Cuenta con un canal de YouTube en donde explican como realizar varios desarrollos. Los videos son bastante concisos y claros.

b) Fuera del canal oficial, es bastante difícil encontrar videos de otras personas explicando cómo realizar algún desarrollo

c) Cuentan con una comunidad bastante activa para la solución de dudas e inclusive para estar al tanto de limitantes tecnológicas.

d) La documentación oficial es bastante funcional y visual, lo que ayuda a entender y replicar los procesos, además, para algunos temas complementan la documentación con un video practico. Aunque cubre bien las funcionalidades básicas, para temas más avanzados o específicos puede requerir de un soporte mas directo, haciendo uso de la comunidad.

2. **Bubble**

a) Al ser una plataforma tan grande y conocida, su foro es bastante activo por lo que las preguntas se resuelven relativamente rápido.

b) Cuenta con función de chat en vivo para ciertos planes.

c) La documentación es bastante exhaustiva y cubre varios escenarios de desarrollo, desde los básicos a los mas avanzados.

d) Incluye guías, vídeos, tutoriales interactivos y una academia con cursos, algo que no se ve en las otras plataformas.

3. **Adalo**

a) Cuenta con una comunidad y un centro de ayuda con guías y tutoriales.

b) Cuenta con webinars y tutoriales de video para facilitar el entrenamiento y el desarrollo.

c) La documentación es bastante completa para tareas y conceptos básicos de la creación de la aplicación. Incluye guías con diagramas paso a paso, tutoriales y ejemplos prácticos. Sin embargo, puede ser un poco limitada para tema más avanzados y complejos.

4. **Webflow**

a) Incluye Guías, tutoriales y cursos en su plataforma, lo que facilita el aprendizaje y familiarización con la plataforma.

b) Cuenta con una extensa documentación que cubre desde conceptos básicos hasta temas avanzados.

c) Al igual que la demás plataforma, cuenta con un foro para la solución de dudas.

d) Cuenta con una plataforma (webflow university) que facilita una gran variedad de cursos que cubren aspectos del desarrollo con web Flow.

A partir de los criterios establecidos, el siguiente cuadro presenta el peso asignado a cada criterio y la calificación otorgada a cada una de las plataformas evaluadas. Tanto el peso como la calificación se encuentran en una escala del 1 al 5, donde 1 representa la menor importancia y 5 la mayor.

Peso	4	2	4	3	**Total**
	Costo	**Tamaño de la empresa**	**Conexión con API Rest**	**Soporte y documentación**	
WeWeb	3	2	5	3	45
Bubble	3	4	3	4	44
Adalo	2	3	1	3	27
Webflow	2	5	2	4	38

Tabla 3.1: Cuadro comparativo para la selección de la plataforma low-code

Con base en los resultados de la Tabla 3.1 la plataforma seleccionada para el desarrollo del proyecto es WeWeb.

3.1.2. Selección del framework para el desarrollo convencional

Los frameworks de desarrollo frontend son herramientas que permiten construir interfaces de usuario de manera mas eficiente,proporcionando una estructura predefinida y componentes reutilizables que facilitan el desarrollo. De igual forma, estos frameworks permiten gestionar de formas mas organizadas y modular el código, optimizando la experiencia de usuario y la interacción con el backend.

Entre los frameworks mas populares se encuentran:

- **Angular:** Es un framework basado en TypeScript, mantenido por Google. Esta herramienta ofrece una arquitectura modular. escalable y que se ajusta para aplicaciones grandes y complejas.
- **React:** Es una biblioteca mantenida por Meta. Se enfoca en el desarrollo basado en componente y es ideal para la creación de aplicaciones dinámicas y rápidas.
- **Vue.js:** Un framework conocido por su simplicidad y flexibilidad, que permite la integración gradual de proyectos.

Para el desarrollo del proyecto se decidió, con base a la experiencia del manejo del framework, hacer uso de Angular. Este framework nos ofrece las herramientas necesarias para poder hacer el desarrollo de la aplicación para la gestión de reservas de canchas de tenis tales como servicios HTTP y el uso de Observables para el flujo de datos de las reservas y los llamados al API.

De igual forma, al hacer uso de TypeScript, nos permite tener una aplicación mas estructurada y con tipado estático, algo que nos ayuda a reducir la probabilidad de errores.

Por último, por como se tiene pensada la aplicación a futuro, angular nos permite escalar en funcionalidades y en rendimiento. Su capacidad para manejar grandes cantidades de vistas y módulos la hace perfecta para este proyecto.

3.1.3. Selección de los criterios de comparación

Una vez seleccionada las dos herramientas para el desarrollo Frontend, es importante poder definir los criterios que se consideran importante a la hora de seleccionar una herramienta para el desarrollo frontend.

Kaur and Tiwari (2023), en su paper, realizan una comparación entre los frameworks y librerías más populares de desarrollo Frontend. Para poder realizar esa comparación define algunos parámetros de evaluación entre los que se encuentran:

1. **La popularidad de la tecnología**

   Cuando una tecnología es ampliamente aceptada se siente como una apuesta segura. Esto genera que haya más recursos como documentación, tutoriales y foros de soporte lo que facilita el uso y aprendizaje de tecnología. De igual forma, los comentarios y contribuciones de la comunidad generan que haya un continuo mejoramiento y actualización de la tecnología. Por tal motivo es importante examinar el tamaño y la actividad de la comunidad. Una tecnología ampliamente adoptada y soportada, lleva a un ciclo de desarrollo más eficiente y mejores resultados.

2. **Documentación**

   Contar con buenas instrucciones y directrices de cómo utilizar una tecnología es vital para poder sacar el mayor provecho a cualquier tecnología. Una buena documentación disminuye la curva de aprendizaje y les permite a los desarrolladores utilizar las funcionalidades de la herramienta.

   Contar con una buena documentación es vital a la hora de diagnosticar y corregir errores. Por tal motivo la documentación juega un papel crucial a la hora de seleccionar una tecnología.

3. **Desempeño**

   Testear el desempeño de una tecnología es vital para asegurar de que pueda cumplir con el propósito para el cual fue desarrollado. El cómo se maneja la carga de componentes y cómo se comporta al procesar grandes cantidades de datos y peticiones. Al final, la experiencia del usuario se ve grandemente afectada por el desempeño de la tecnología seleccionada.

4. **Escalabilidad**

   A la hora de seleccionar una tecnología para desarrollo frontend, este es un factor crucial. Una tecnología se considera escalable si se puede ajustar al crecimiento del tráfico de datos sin comprometer el rendimiento o estabilidad de la aplicación.

Por otro lado,Sahay et al. (2020) en su articulo define algunos criterios ya no para comparar herramientas de desarrollo frontend, sino para la elección de una plataforma de desarrollo Low-code. Entre los criterios seleccionados se encuentran:

1. **Interfaz gráfica**

   Se define como las herramientas que se le ofrece al usuario para poder desarrollar, por ejemplo cuestionarios, herramientas de drag-and-drop y componentes genéricos que pueden ser utilizados para el desarrollo.

2. **Interoperabilidad con servicios externos**

   Se refiere a la capacidad y facilidad de poder conectarse a diferentes fuentes de datos externos.

3. **Seguridad**

   Se refiere a mecanismos de autenticación, protocolos de seguridad y acceso de los usuarios al control de la infraestructura.

4. **Soporte colaborativo de desarrollo**

   Se refiere a la comunidad de la tecnología, que brinda soporte colaborativo a diferentes desarrolladores ubicados en diferentes zonas.

5. **Reutilización**

   Está relacionado con los mecanismos que la plataforma tiene para facilitar el uso de artefactos ya construidos. Por ejemplo plantillas, tableros de control y formularios pre construido.

6. **Escalabilidad**

   Permite escalar la aplicación para poder soportar grandes cantidades de usuarios activos, flujo de datos y almacenamiento.

7. **Mecanismo de construcción de la aplicación**

   Se refiere a las formas en que se construye la aplicación, ya sea empleando técnicas de generación de código o mediante enfoques de modelos en tiempo de ejecución.

8. **Despliegue de la aplicación**

   Una vez construida la aplicación, en que plataformas se puede publicar y asimismo si se puede desplegar local o en la nube.

De igual forma, Rosłan and Śmiałek (2023), en su paper "Comparative Analysis of Low-Code Computation Systems", también define unos criterios para realizar el análisis, entre los que se encuentran, no mencionados anteriormente:

1. **Costo**

   Costo potencial al desplegar y hacer uso de la aplicación. Por lo general estas herramientas tienen paquetes con cierta capacidad, sea en peticiones, almacenamiento o número de usuarios.

2. **Disponibilidad**

   Se refiere a que tanto tiempo en el año nuestra aplicación no estaría disponible debido a inconvenientes de la herramienta utilizada.

3. **Facilidad de mantenimiento**

   Herramientas que ofrece la plataforma para el diagnóstico y corrección de bugs.

Por último, la norma ISO 25010 establece las características de calidad que se van a tener en cuenta a la hora de evaluar las propiedades de un producto de software. En la Figura 3.3 se pueden observar las 9 características con sus subcaracterísticas que define la norma.

CALIDAD DEL PRODUCTO SOFTWARE								
ADECUACIÓN FUNCIONAL	EFICIENCIA DE DESEMPEÑO	COMPATIBILIDAD	CAPACIDAD DE INTERACCIÓN	FIABILIDAD	SEGURIDAD	MANTENIBILIDAD	FLEXIBILIDAD	PROTECCIÓN
COMPLETITUD FUNCIONAL CORRECCIÓN FUNCIONAL PERTINENCIA FUNCIONAL iso25000.com	COMPORTAMIENTO TEMPORAL UTILIZACIÓN DE RECURSOS CAPACIDAD	COEXISTENCIA INTEROPERABILIDAD	RECONOCIBILIDAD DE ADECUACIÓN APRENDIZABILIDAD OPERABILIDAD PROTECCIÓN FRENTE A ERRORES DE USUARIO INVOLUCRACIÓN DEL USUARIO INCLUSIVIDAD ASISTENCIA AL USUARIO AUTO-DESCRIPTIVIDAD	AUSENCIA DE FALLOS DISPONIBILIDAD TOLERANCIA A FALLOS RECUPERABILIDAD	CONFIDENCIALIDAD INTEGRIDAD NO-REPUDIO RESPONSABILIDAD AUTENTICIDAD RESISTENCIA	MODULARIDAD REUSABILIDAD ANALIZABILIDAD CAPACIDAD DE SER MODIFICADO CAPACIDAD DE SER PROBADO	ADAPTABILIDAD ESCALABILIDAD INSTALABILIDAD REEMPLAZABILIDAD	RESTRICCIÓN OPERATIVA IDENTIFICACIÓN DE RIESGOS PROTECCIÓN ANTE FALLOS ADVERTENCIA DE PELIGRO INTEGRACIÓN SEGURA

Figura 3.3: Calidad del producto software iso (2024)

Evaluando los criterios definidos por los autores y los que se evalúan en la norma ISO25010, podemos observar que los siguientes criterios en común:

- Compatibilidad (Interoperabilidad)
- Desempeño
- Seguridad
- Mantenibilidad

De igual forma, con los criterios identificados se realizó una encuesta a un grupo de expertos para que seleccionaran los 4 criterios que consideraran más importantes, esto con el fin de poder definir los métodos a evaluar el cumplimiento de cada una de las dos tecnologías y así poder compararlas. Los criterios listados fueron:

1. **Documentación**
2. **Desempeño**
3. **Compatibilidad**
4. **Fiabilidad**
5. **Escalabilidad**
6. **Flexibilidad**

7. **Disponibilidad**
8. **Costo**
9. **Desplegabilidad**
10. **Mantenibilidad**
11. **Desplegabilidad**
12. **Seguridad**
13. **Interoperabilidad**
14. **Interfaz gráfica**

La encuesta se realizó haciendo uso de la herramienta Forms de Microsoft, en donde cada persona seleccionó los 4 criterios que consideraba mas importante y el rol que desempeñaba actualmente. En total hubo una participación de 52 personas, distribuidas entre los siguientes roles:

1. Ingeniero de soporte
2. Asegurador de calidad
3. Desarrollador
4. Product manager/owner
5. Líder de desarrollo
6. Arquitecto de soluciones

Como se puede observar en la Figura 3.4, el numero de votantes por cada rol tiende a ser muy parejo pero con una inclinación, a nivel general, a roles involucrados en el desarrollo de software.

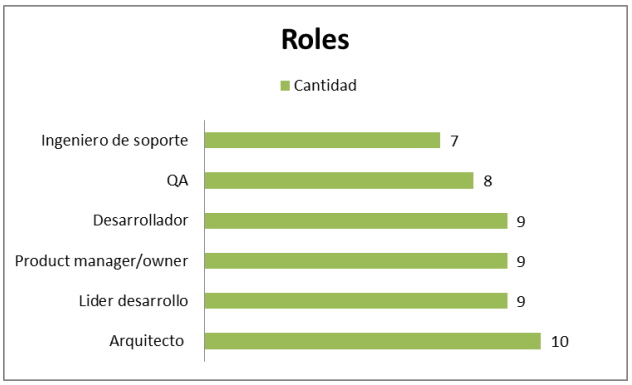

Figura 3.4: Distribución de los roles de los votantes

En la Figura 3.5 se puede observar la distribución de cada uno de los votos realizados para elegir los 4 criterios que consideraban mas importantes a la hora de seleccionar una herramienta de desarrollo frontend.

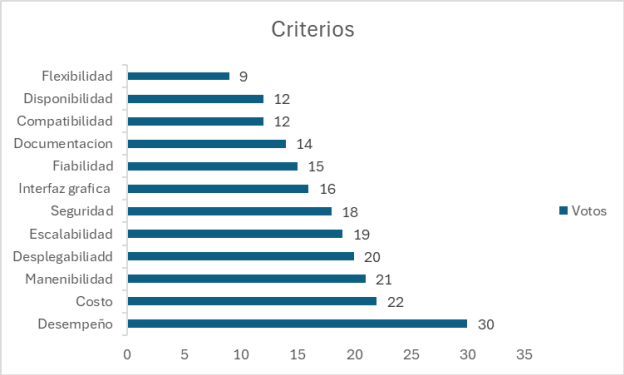

Figura 3.5: Resultado de la votación por los criterios

Con base en los resultados de la encuesta y como los primeros resultados ya fueron seleccionados, se complementas los criterios seccionados con:

1. **Desplegabilidad**

2. **Costo**

De igual forma, teniendo presente que uno de los grandes motivos por los que se promocionan las herramientas Low-code es por la rapidez con la que se puede generar una aplicación robusta, se adiciona el criterio de **Rapidez y robustez en el desarrollo** para quedar al final con los siguientes criterios:

1. **Compatibilidad**

2. **Desempeño**

3. **Seguridad**

4. **Mantenibilidad**

5. **Desplegabilidad**

6. **Costo**

7. **Rapidez y robustez en el desarrollo**

3.1.4. Requisitos funcionales de la aplicación

Para definir los requisitos funcionales de la aplicación, nos apoyamos principalmente en la experiencia de personas cercanas al mundo del tennis, desde administradores de clubes, profesores y afiliados. Mediante diferentes entrevistas y conversaciones se fueron identificando características claves que ayudarían a optimizar diferentes puntos de dolor en el proceso de gestión de una reserva, desde la asignación, actualización y cancelación de la misma. Se debe tener en cuenta que se listaron las funcionalidades básicas para poder operar.

Los requisitos funcionales los vamos a agrupar en dos categorías. La primera como gestión de usuarios y la segunda como gestión de reservas.

Gestión de usuarios

1. Los usuarios deben poder registrarse.
2. Los usuarios deben poder recuperar/modificar su contraseña.
3. Los usuarios deben poder iniciar sesión en la aplicación.

Gestión de reservas

1. Los usuarios deben poder crear una reserva de una o mas horas.
2. Los usuarios deben poder cancelar una reserva.
3. Los usuarios deben poder observar las reservas que tienen.

Administrador del club

1. El usuario administrador debe poder crear una reserva de una o mas horas.
2. El usuario administrador debe poder cancelar una reserva.
3. El usuario administrador debe poder ver las reservas por cada cancha.
4. El usuario administrador debe poder crear una cancha.
5. El usuario administrador debe poder configurar el tiempo en que una cancha está disponible.

3.1.5. Requisitos funcionales de la aplicación

Los requerimientos no funcionales describen las características y restricciones de un sistema que no están directamente relacionadas con las funcionalidades de la aplicación, pero que son cruciales para garantizar el correcto funcionamiento de la misma. Dentro de los requisitos no funcionales tenemos.

- La aplicación debe cargar las paginas principales en menos de 3 segundos con conexiones estándar de internet. Un mínimo de 10Mbps.
- Las principales interacciones del usuario con la aplicación (creación, actualización y cancelación de reservas), deben procesarse en un tiempo máximo de un segundo después de enviada la acción.
- El frontend debe, en la medida de lo posible implementar técnicas de lazy loading para reducir el consumo de recursos y mejorar la experiencia en móviles.
- La aplicación debe ser responsive, ajustándose a diferentes tamaños de pantalla, asegurando que todas las funcionalidad sean accesibles en cualquier dispositivo.
- En casos de errores con algún proceso, la aplicación debe mostrar mensajes claros y amigables en menos de 0.5 segundos después de levantado el error.
- El sistema debe soportar al menos 500 usuarios de forma concurrente sin degradar el rendimiento de la aplicación.

3.1.6. Diseño del Backend

3.1.6.1. Arquitectura general

Como se ha mencionado, el backend ya se encuentra implementado utilizando el framework Spring Boot, siguiendo una arquitectura RESTful que permite la interacción entre el cliente y el servidor mediante solicitudes HTTP estándar. El sistema está diseñado utilizando el patrón MVC (Modelo-Vista-Controlador), lo que permite una separación clara de responsabilidades:

- **Controlador:** Gestiona las solicitudes HTTP entrantes y dirige las peticiones hacia los servicios apropiados. Cada controlador está diseñado para manejar un conjunto específico de endpoints, garantizando que las operaciones se realicen de manera eficiente y segura.
- **Servicios:** Aquí se implementa la lógica de negocio. Los servicios procesan los datos, ejecutan las reglas del sistema y gestionan el flujo de la información entre la capa de presentación (controlador) y la capa de persistencia (repositorio).
- **Repositorio:** Esta capa es responsable de interactuar directamente con la base de datos. Utilizando JPA (Java Persistence API) y Hibernate como proveedor de ORM, el repositorio gestiona las operaciones de creación, lectura, actualización y eliminación de datos (CRUD), permitiendo una abstracción de bajo nivel en la manipulación de la base de datos.

3.1.6.2. Seguridad

Para la seguridad se hace uso de Spring Security, Este se integra con JWT (JSON Web Token) para garantizar la autenticación y autorización en la aplicación. Mediante este sistema se asegura

que solo los usuarios autenticados puedan acceder a los recursos y que se respete los permisos por rol de cada usuario.

Las características principales a nivel de seguridad son:

- **Autenticación mediante JWT:** Los usuarios se autentican mediante el correo y la contraseña, tras lo cual se genera un token JWT que se incluye al momento de realizar peticiones a endpoints que se encuentren protegidos.
- **Autorización basada en roles:** Mediante anotaciones que nos provee spring security, tales como @PreAuthorize y @Secured, se controlan los accesos a diferentes endpoints en función de los roles de cada usuario.

3.1.6.3. Endpoints

El backend expone varios endpoints RESTful que permiten al cliente interactuar con la aplicación. A continuación, agrupados por módulo, se encuentran los endpoints expuestos para poder cumplir con los requerimientos de la aplicación,

1. **Módulo de autenticación:**
   - POST: /api/v1/login
   - POST: /api/v1/auth/register
   - POST: /api/v1/auth/auth-account
   - POST: /api/v1/auth/recover-password
   - POST: /api/v1/auth/refresh-token
   - POST: /api/v1/auth/change-password
   - GET: /api/v1/auth/profile
   - GET: /api/v1/auth/check-email-availability
   - GET: /api/v1/auth/initial-sign-up-data
2. **Módulo de usuarios**
   - GET: /api/v1/user
   - PUT: /api/v1/user
   - DELETE: /api/v1/user
3. **Módulo de reservas**
   - Rol administrador
     - GET: /api/v1/reservation/club/clubId
     - GET: /api/v1/reservation/club/clubId/available-date-times

- GET: /api/v1/reservation/club/clubId/members

- Rol usuario

  - GET: /api/v1/reservation/club/clubId/available-date-times
  - POST: /api/v1/reservation/clubId
  - GET: /api/v1/reservation/user/userId
  - DELETE: /api/v1/reservation/reservationId

Para mayor información sobre los endpoints expuestos, se pueden comunicar a jparagon2021@gmail.com para que reciban la documentación en Swagger del API, en donde se pueden ver cuales paths necesitan autenticación y que parámetros/variables espera cada endpoint.

3.1.6.4. Diagramas

Teniendo ya un mayor contexto del backend ya implementado, en la Figura 3.6 se puede observar el diagrama de componentes del API. En este diagrama se observa que la arquitectura sigue un enfoque de monolito modular, en el que el sistema está organizado en módulos independientes que manejan distintas funcionalidades, pero que se despliegan como una única aplicación.

El diagrama sigue el modelo C4, que permite representar de forma clara los diferentes niveles de abstracción de la arquitectura, desde el sistema global hasta los detalles de los componentes individuales. El enfoque C4 facilita la comprensión de cómo los módulos se comunican entre sí y cómo interactúan con otros sistemas externos, como bases de datos, servicios de autenticación y otros microservicios si los hubiera.

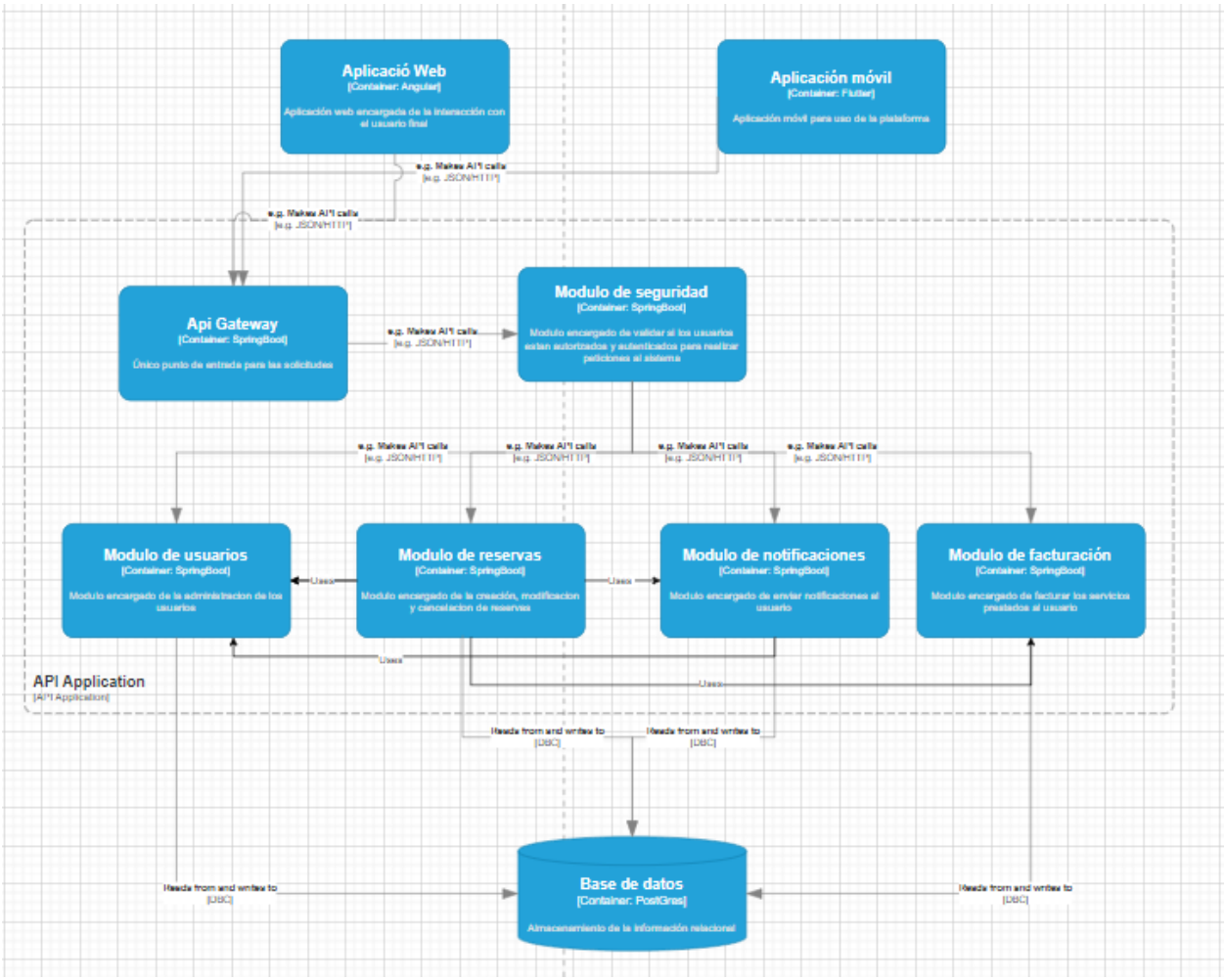

Figura 3.6: Modelo C4 componentes

Entrando un poco mas en detalle, en la Figura 3.7 esta el modelo de entidad relación para el modulo de usuarios y para el modulo de las reservas.

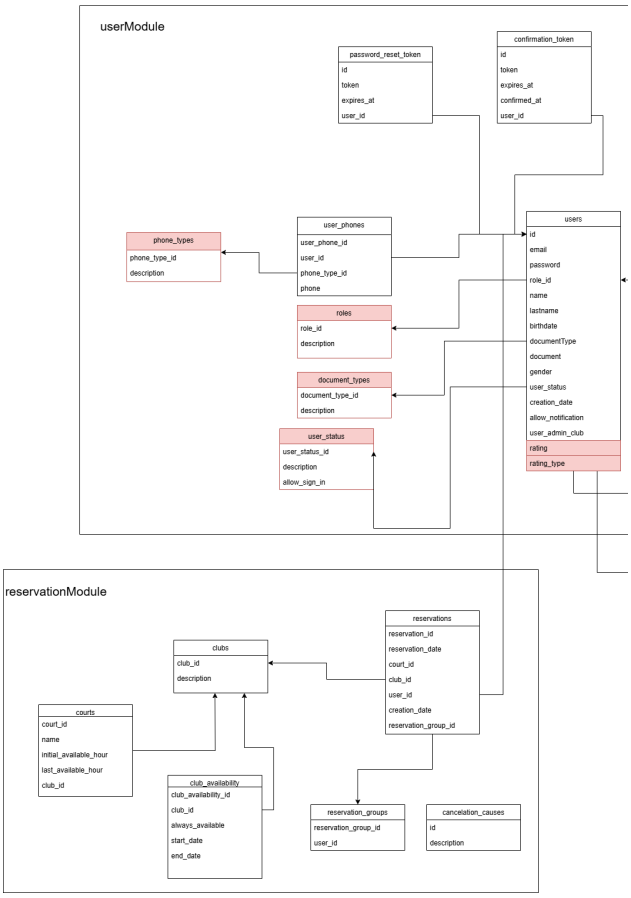

Figura 3.7: Modelo entidad-relación modulo de reservas y usuarios

3.1.7. Diseño Frontend

Como se mencionó anteriormente, ya se encuentran seleccionadas las herramientas low-code para el desarrollo frontend. Teniendo a su vez ya definidos los requerimientos funcionales de la aplicación, se hará uso de los diagramas de usuario para plasmar el flujo de cada uno de los procesos que debe soportar la aplicación y como interactuará el usuario con estos.

En la Figura 3.8 se encuentra plasmado el flujo de la autenticación y registro de usuario, mientras que en la Figura 3.9 se detalla el proceso para la creación de una reserva, y en la Figura 3.19 se describe el flujo para la cancelación de una reserva, cabe resaltar que los dos últimos flujos son para un usuario no administrador.

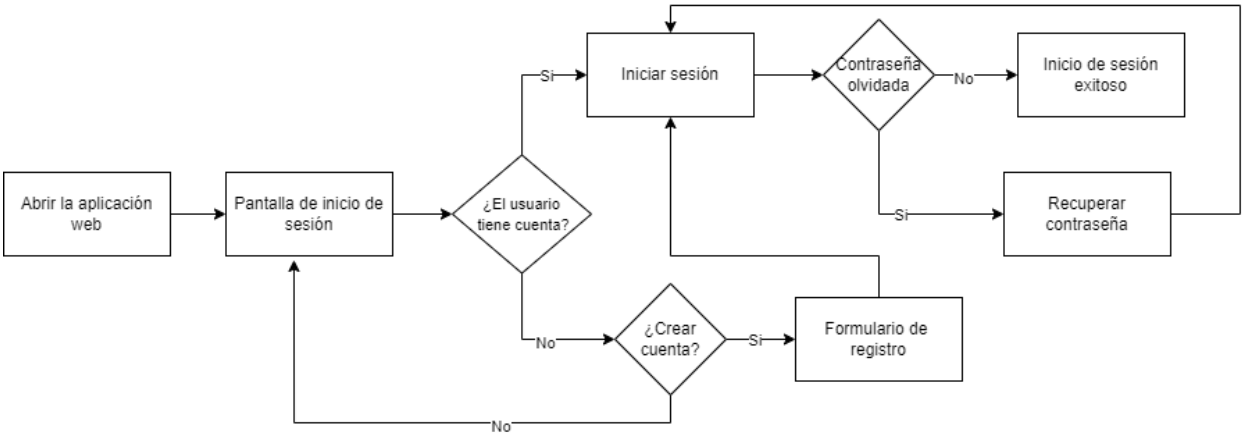

Figura 3.8: Flujo de usuario para autenticación y registro

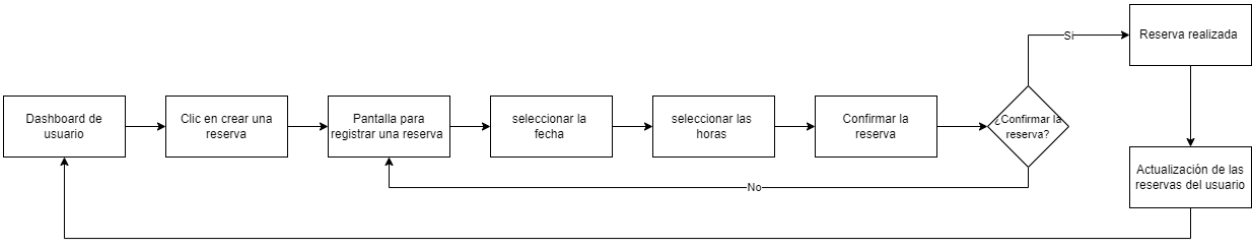

Figura 3.9: Flujo de usuario para creación de una reserva

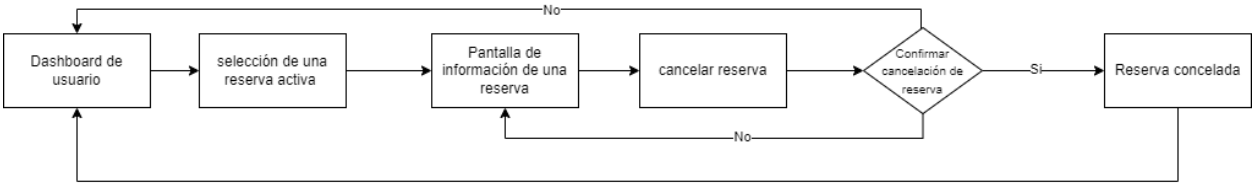

Figura 3.10: Flujo de usuario para cancelar reserva

Por otro lado en la Figura 3.11 se muestra el proceso para la creación de una reserva para un usuario administrador, al igual que en la Figura 3.12 se muestra el proceso para cancelar una.

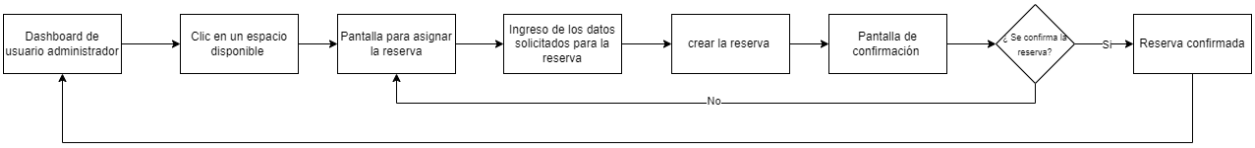

Figura 3.11: Flujo usuario administrador para crear una reserva

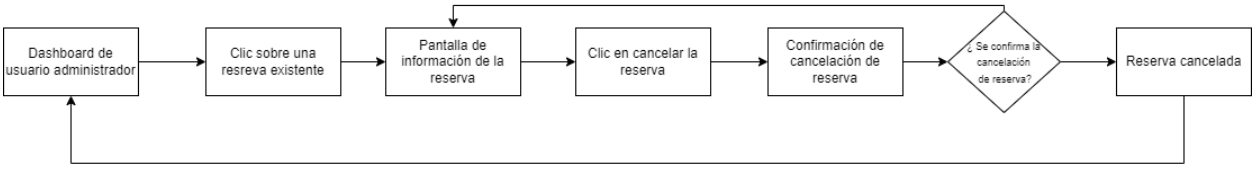

Figura 3.12: Flujo usuario administrador para cancelar una reserva

Como funcionalidad única de un usuario administrador, en la Figura 3.13 se muestra el proceso para la configuración de una cancha.

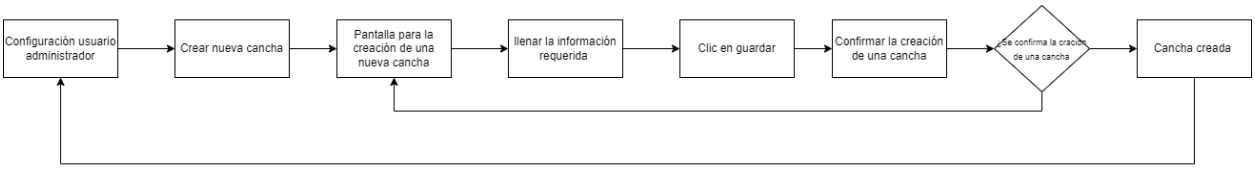

Figura 3.13: Flujo usuario administrador para configurar una cancha

3.1.8. Implementación del Frontend

Con base en los requerimientos funcionales y en el diseño de los flujos de usuario se buscó hacer uso de los componentes provistos por la plataforma de desarrollo low-code para disminuir el tiempo de desarrollo, sacando provecho de las ventajas de esta herramienta.

Con base en el flujo de la Figura 3.8 se crearon las siguientes pantallas para el proceso de login y de registro de usuario.

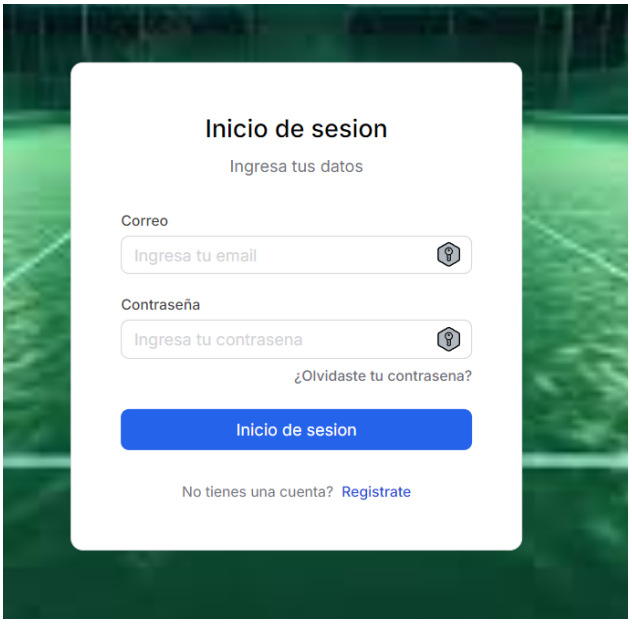

Figura 3.14: Inicio de sesión

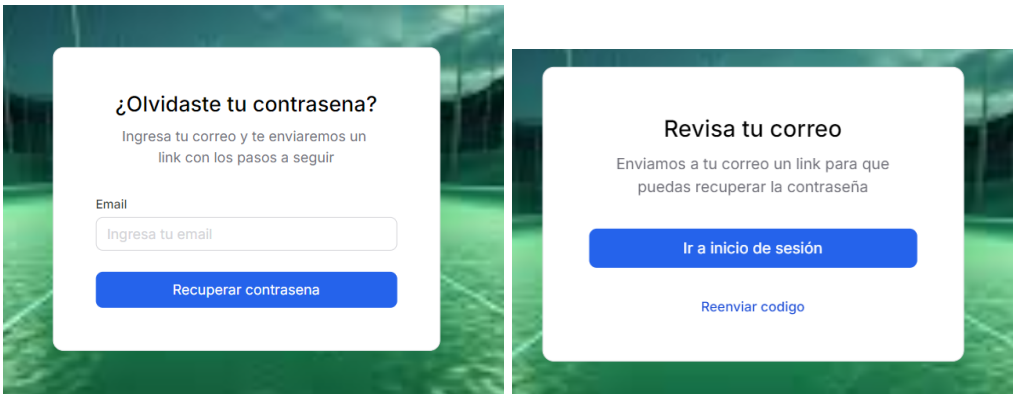

Figura 3.15: Proceso de recuperar contraseña

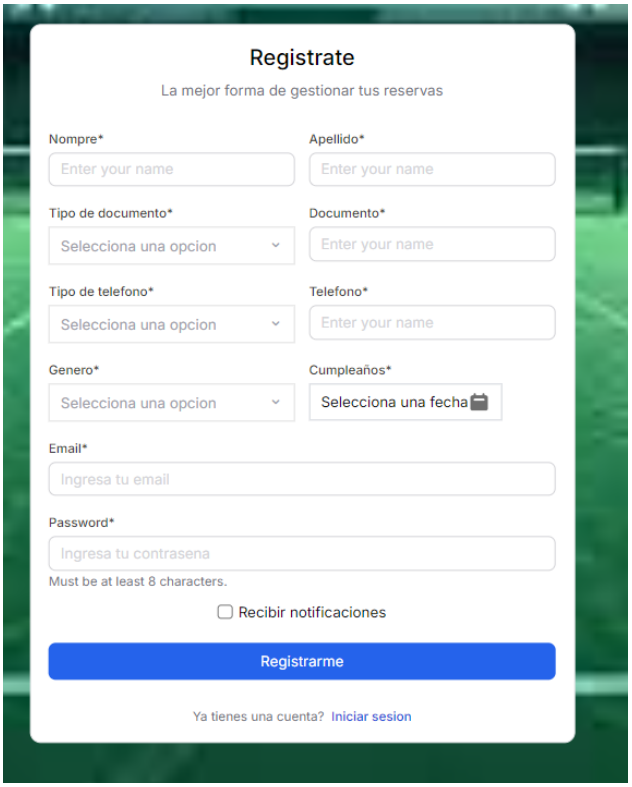

Figura 3.16: Registro

Con base en el flujo de la figura 3.9 se creo el siguiente dashboard para cuando el usuario no es el administrador del club (3.17), desde ahí se pueden gestionar las reservas, tanto crear como cancelar una.

Figura 3.17: Dashboard para usuario no administrador

Para crear una reserva, se debe seleccionar primero una fecha para que se carguen las horas en las cuales hay disponibilidad. Luego de seleccionadas las horas, se da continuar y se confirman las reservas, ver Figura 3.18.

Figura 3.18: Proceso de creación de una reserva

Para la cancelación de una reserva, se da clic sobre una reserva existente y en el modal desplegado se da clic en cancelar, ver Figura 3.19.

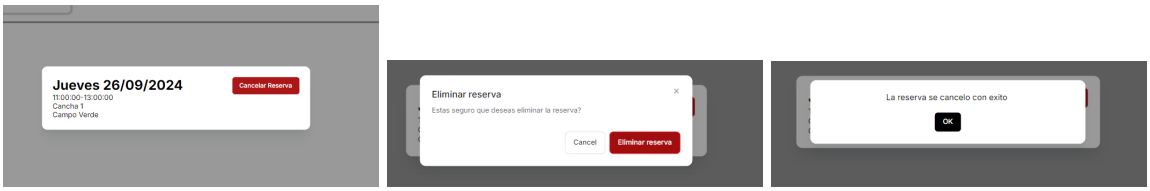

Figura 3.19: Proceso de cancelación de una reserva

Para un usuario administrador, la vista después de iniciar sesión es distinta. En la Figura 3.20 se muestra el panel de control para un administrador, donde es posible visualizar el estado de las canchas por horario y gestionar las reservas de los usuarios, permitiendo tanto asignar como cancelar reservas.

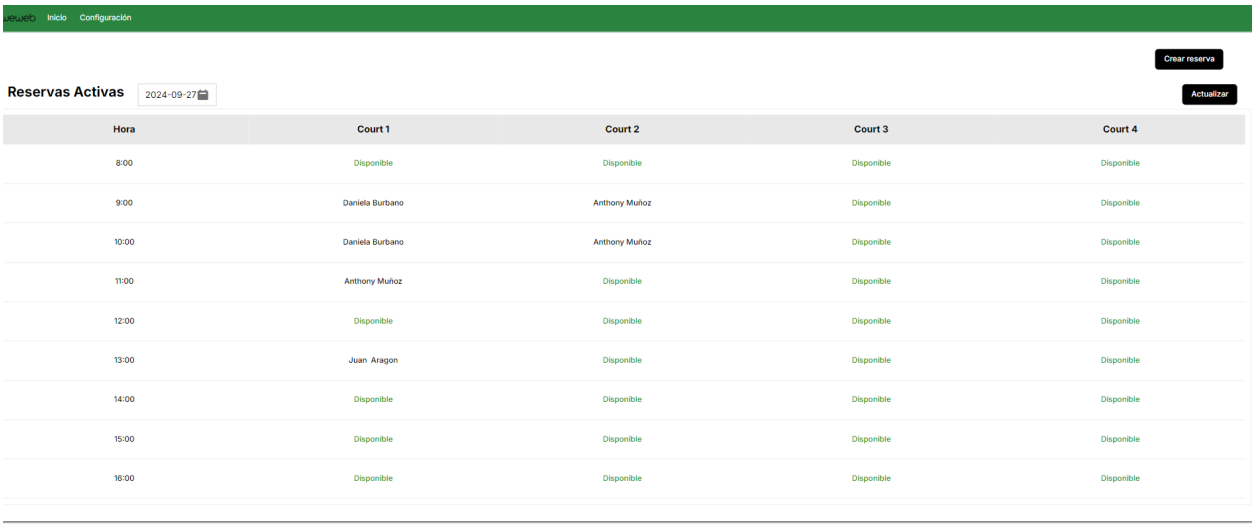

Figura 3.20: Dashboard del usuario administrador

Para crear una reserva, el usuario administrador debe seleccionar un espacio que se encuentre disponible y debe asignarle el usuario realizando la búsqueda por el nombre del miembro del club.En

la Figura 3.21 se puede observar el flujo para la creación de la reserva.

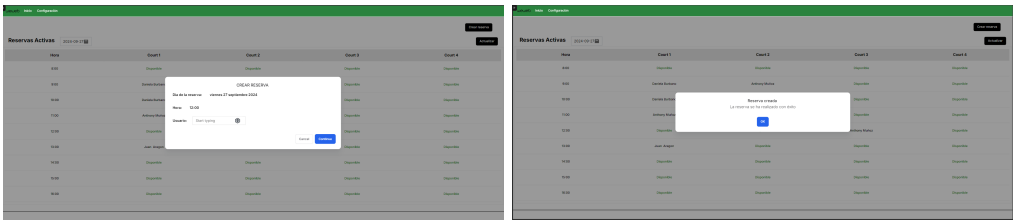

Figura 3.21: Proceso creación de reserva usuario administrador

Para cancelar una reserva, el usuario administrador se debe parar sobre la reserva que desea cancelar e indicar si quiere cancelar solo una hora de la reserva o la reserva completa, esto cuando la reserva es de mas de una hora. Ver Figura 3.12 para el caso cuando es de una sola hora la reserva y la Figura 3.23 para cuando hay mas de una hora en el grupo de reserva.

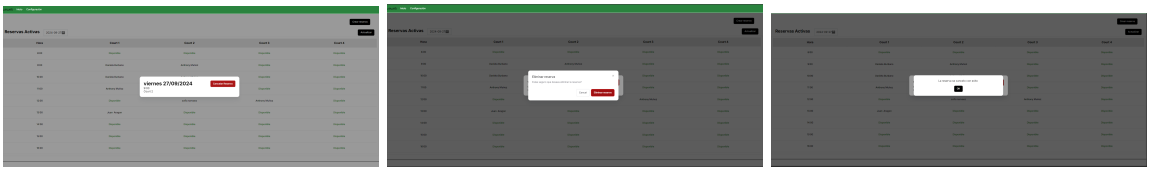

Figura 3.22: Proceso de cancelación de una reserva desde un usuario administrador

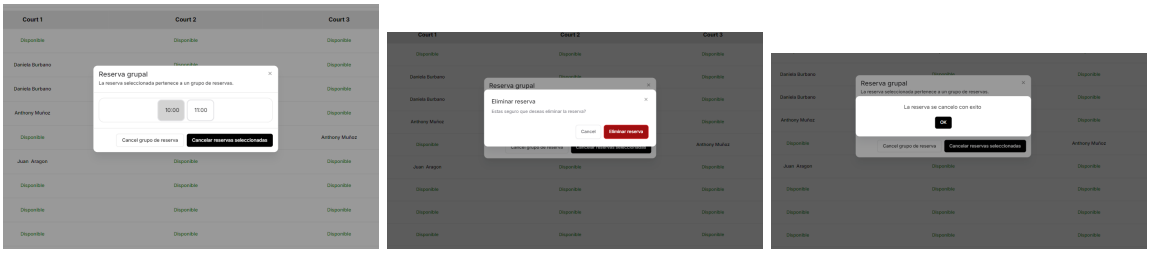

Figura 3.23: Proceso de cancelación de una reserva grupal desde un usuario administrador

Por último, el usuario administrador puede crear y modificar la información de una cancha, esto desde la sección de configuración, ver Figura 3.24.

Figura 3.24: Sección para configurar las canchas

3.2. Comparación de los desarrollos

Teniendo en cuenta los criterios definidos en el numeral 3.1.3 a continuación se realiza la comparación de ambos desarrollos sobre cada uno de ellos.

3.2.1. Compatibilidad

De acuerdo con la norma ISO 25000, la compatibilidad se define como la capacidad de un producto para intercambiar información con otros productos y/o ejecutar las funciones requeridas al compartir el mismo entorno y recursos. Dentro de esta característica, se destaca la interoperabilidad, que se refiere a la habilidad de dos o más sistemas para intercambiar información y utilizar los datos intercambiados de manera efectiva.

Como se ha mencionado anteriormente, la aplicación frontend realiza solicitudes a una API REST, lo que hace que la conexión con servicios externos sea fundamental para el desarrollo del proyecto.

La herramienta WeWeb incluye una sección dedicada a la configuración de todas las llamadas a la API REST. En primer lugar, se especifica el tipo de solicitud (GET, POST, PUT o PATCH); luego, se define la URL, los encabezados y las variables necesarias. Finalmente, se lleva a cabo la petición y se puede verificar si la información retornada es la adecuada. En la Figura 3.25 se presenta un paso a paso de este proceso. Configurar las solicitudes en esta sección permite centralizar su uso y aplicarlas en diferentes flujos a lo largo de la aplicación.

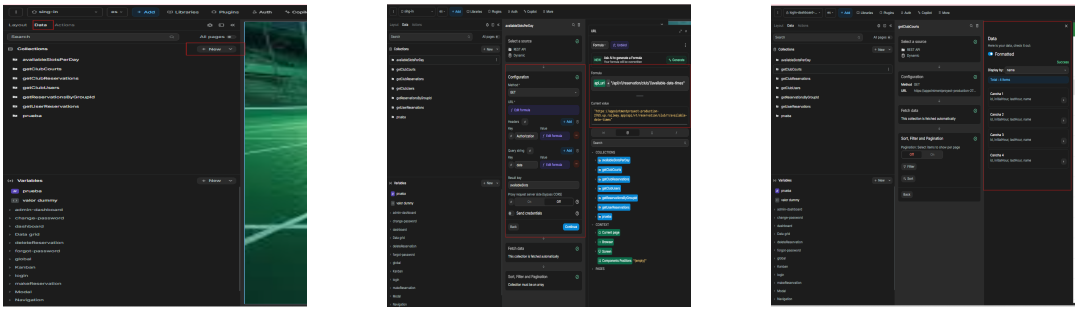

Figura 3.25: Proceso para configurar un llamado al API REST en WeWeb

Si no se configura la petición en la sección de datos, es posible realizarla dentro de un flujo. Sin embargo, esta aproximación tiene desventajas: la petición no sería reutilizable, lo que generaría duplicación de lógica y complejidad en el mantenimiento. Además, se perdería la ventaja de centralizar la petición en un único punto, lo que facilitaría su gestión y actualización de forma más eficiente y organizada. En la Figura 3.26 se puede observar el proceso para configurar la petición dentro de un flujo.

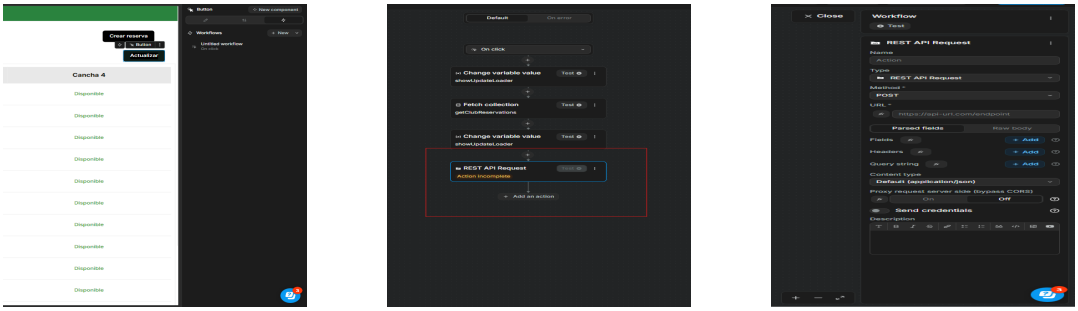

Figura 3.26: Proceso para configurar una petición desde un flujo

En Angular, por otro lado, las peticiones a servicios externos se gestionan mediante servicios centralizados que se inyectan a los componentes. A través del uso de HttpClient, las peticiones se configuran en servicios, lo que permite tener una clara separación de responsabilidades. Además, Angular permite hacer uso de interceptores para gestionar de forma global en la aplicación aspectos como la autenticación y el manejo de errores. En la Figura 3.27 se puede observar como ser configura una petición HTTP en Angular.

```
getAvailableSlotsPerDay(date: string): Observable<AvailableSlotsResponse> {
  const url = `${environment.API_URL}/reservation/club/1/available-date-times?date=${date}`;
  return this.http.get<AvailableSlotsResponse>(url, {
    headers: this.setHeaders(),
  });
}

getUserReservations(): Observable<UserReservationResponse> {
  const url = `${environment.API_URL}/reservation/user/${this.userId}`;

  return this.http
    .get<UserReservationResponse>(url, { headers: this.setHeaders() })
    .pipe(
      tap((response) => {
        this.reservations$.next(response);
      })
    );
}

getReservationsByGroupId(groupId: string): Observable<GroupReservationInfo> {
  const url = `${environment.API_URL}/reservation/group-reservation/${groupId}`;
  return this.http.get<GroupReservationInfo>(url, {
    headers: this.setHeaders(),
  });
}
```

Figura 3.27: Petición HTTTP en Angular

A diferencia de WeWeb, donde la interfaz visual facilita la configuración de las peticiones HTTP, en Angular se requiere de un mayor conocimiento técnico para manejar el código, lo que de por si aporta una mayor flexibilidad pero también implica una curva de aprendizaje más pronunciada. Sin embargo, una vez implementado, angular ofrece un control mas granular sobre la gestión de peticiones y esto puede mejorar el rendimiento de la aplicación.

3.2.2. Desempeño

Con base en la ISO 25010, la eficiencia de desempeño es la característica que representa el desempeño de un producto en la realización de sus funciones dentro de unos parámetros de tiempo y rendimiento especificados y con un uso eficiente de recursos.

Muchas de las pruebas de desempeño recaen sobre el backend ya que es allí en donde se manejan los procesos críticos como el procesamiento de datos y la gestión de las solicitudes. Sin embargo, el frontend también juega un papel importante en la experiencia de usuario y puede ser evaluado en términos de rendimiento y eficiencia. Dentro de las pruebas que se pueden realizar desde el frontend, tenemos las siguientes, definidas también como requerimientos no funcionales de la aplicación.

3.2.2.1. Tiempo de carga inicial

El tiempo de carga inicial es el tiempo que tarda una aplicación o sitio web en cargar su contenido principal, desde que el usuario inicia la solicitud hasta que puede ver y comenzar a interactuar con la interfaz.

Para medir este tiempo, se hizo uso de la herramienta de Lighthouse, que dentro de sus criterios para calificar el rendimiento de la aplicación, se encuentra el tiempo de carga inicial. A continuación se presentan los datos para la aplicación realizada en Weweb y para la realizada en Angular. La prueba se realizó tanto para el Login como para el dashboard principal de un usuario base y un usuario administrador.

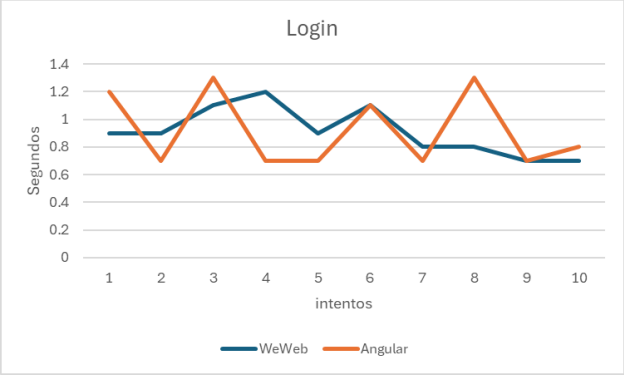

Figura 3.28: Tiempo de carga Login

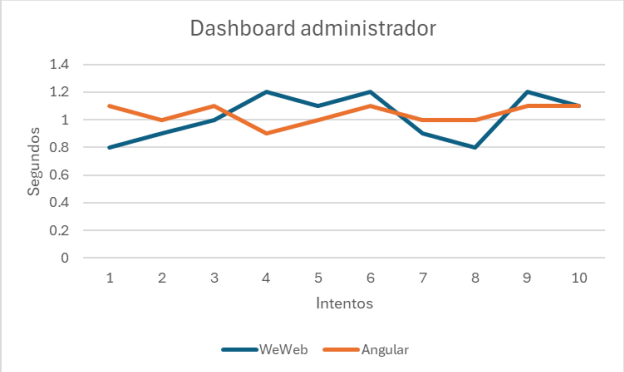

Figura 3.29: Tiempo de carga dashboard administrador

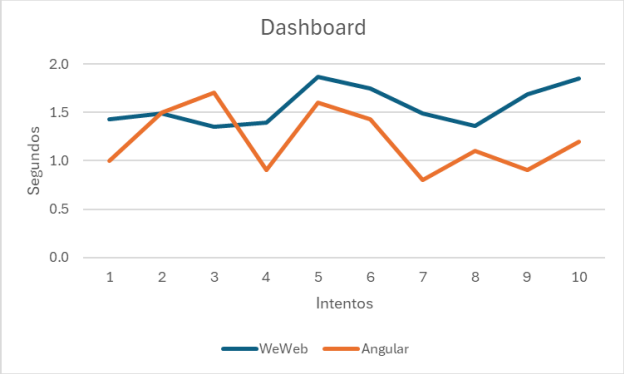

Figura 3.30: Tiempo de carga dashboard usuario básico

A partir de las mediciones realizadas, se observó que el tiempo de carga inicial en la aplicación desarrollada y publicada con WeWeb es, en promedio, mejor que en la versión realizada con Angular, tanto en el módulo de login como en el de dashboard para el usuario base. Sin embargo, en el caso del dashboard para el usuario administrador, la aplicación desarrollada con Angular mostró un mejor rendimiento, superando ligeramente a WeWeb en este aspecto.

A pesar de estas diferencias, la ventaja promedio de WeWeb sobre Angular en cuanto al tiempo de carga inicial no es significativa, siendo de apenas 0.02 segundos en el mejor de los casos. Esto indica que ambas plataformas están bastante igualadas en términos de rendimiento en el tiempo de carga inicial.

3.2.2.2. Tiempo de interactividad

El tiempo de interactividad se refiere al periodo que transcurre desde el momento en que un usuario realiza una acción, como dar clic en un botón para iniciar un proceso importante (por ejemplo, crear una reserva o eliminar una reserva), hasta el momento en que recibe retroalimentación visual o funcional de que la acción ha sido procesada.

Para la prueba se definieron las siguientes iteraciones para usuarios administradores y para usuarios no administradores:

- Usuarios administradores
    1. Crear reserva
    2. Cancelar reserva
    3. Iniciar sesión
- Usuarios no administradores
    1. Crear reserva
    2. Cancelar reserva

3. Iniciar sesión con reservas
4. Iniciar sesión sin reservas

Los resultados obtenidos, que se pueden observar en la Tabla 3.2, muestran que para las operaciones de gestión de reservas, que son operaciones sencillas, ambas aplicaciones manejan un tiempo de interacción muy similar, con una leve ventaja para angular cuando son para un usuario administrador, pero una leve ventaja de WeWeb cuando son no administradores.

Usuario NO administrador	**Angular**	**WeWeb**	**Diferencia (ms)**
Crear reserva	764 ms	700 ms	+64 ms
Cancelar reserva	815 ms	801 ms	+14 ms
Login sin reservas	1173 ms	1512 ms	-339 ms
Login con reserva (5 reservas)	1309 ms	1673 ms	-364 ms
Usuario administrador			
Crear reserva	665 ms	745 ms	-80 ms
Cancelar reserva	689 ms	737 ms	-48 ms
Login (7 canchas)	1127 ms	2450 ms	-1323 ms

Tabla 3.2: Comparación en el tiempo de las iteraciones entre Angular y WeWeb

La diferencia mas significativa se presenta al momento de realizar el Login de un usuario administrador. En angular el tiempo de respuesta es menor desde que el usuario administrador inicia sesión hasta que se cargan las reservas que tiene el club, este es quizás el punto de la aplicación en que el front debe cargar una mayor cantidad de información al usuario.

Asimismo, se realizaron pruebas iniciando sesión con usuarios que contaban con 30, 50,100 y 500 reservas. En estos casos, angular mostró un rendimiento muy superior al de WeWeb, ver Tabla 3.3

Número de reservas	**Angular (ms)**	**WeWeb (ms)**
30	1647	2304
50	1673	2955
100	2640	4660
500	6896	11539

Tabla 3.3: Comparación en el tiempo de carga con diferente número de reservas

Por otro lado, se realizó una prueba cargando la información de las reservas de 50,100 y 500 canchas para un usuario administrador, en la Tabla 3.4 se puede observar el tiempo de carga para cada aplicación.

Como se observa en la Tabla 3.4, al aumentar el número de canchas, el rendimiento de WeWeb disminuye drásticamente. A partir de 500 canchas, la aplicación alcanza su límite, imposibilitando la carga de información y provocando un fallo en la aplicación.

Número de canchas	Angular (ms)	WeWeb (ms)
50	1350	3557
100	2310	8855
500	5568	Error

Tabla 3.4: Comparación en el tiempo de carga con múltiples canchas

En contraste, aunque Angular también muestra una disminución en el rendimiento y un aumento en el tiempo de respuesta, logra cargar la información completa sin colapsar. Esto indica que Angular ofrece una mayor capacidad para gestionar grandes volúmenes de datos, manteniendo la estabilidad de la aplicación incluso bajo condiciones de alta demanda.

En conclusión, con bajos volúmenes de datos, angular presenta un rendimiento ligeramente superior en términos de tiempos de respuesta, especialmente en la gestión de reservas y el proceso de login para usuarios administradores. Por otro lado, con grandes volúmenes de datos el rendimiento de angular es muy superior, soportando el despliegue de la información en menor tiempo e inclusive sin generar un fallo en la aplicación.

3.2.2.3. Usuarios concurrentes

La medición de concurrencia se realiza para poder saber como un sistema maneja múltiples procesos o tareas simultáneamente. Esta es una evaluación que se realizó sobre el back. Si bien se separa un poco de las comparaciones realizadas anteriormente que son solo sobre el fronted, es una prueba importante para poder validar si se cumple con el requerimiento funcional de que la aplicación soporte 500 usuarios concurrentes sin comprometer el rendimiento de la aplicación. Para las pruebas se definieron los siguientes servicios criticos para la aplicación:

- Generales
    1. Crear reserva
    2. Cancelar reserva
    3. Iniciar sesión
    4. Consulta de disponibilidad en un club
    5. Consulta de las reservas de un usuario
- Usuarios administradores de un club
    1. Consulta de las reservas de un club

Para evaluar el rendimiento de la aplicación, se llevaron a cabo pruebas de carga utilizando JMeter, una herramienta ampliamente utilizada para realizar pruebas de rendimiento y carga en aplicaciones. Para estas pruebas se configuraron los siguientes parámetros: En primer lugar se definió el numero de usuarios concurrentes en 500, ademas se estableció el ramp-up period en 10

segundos, lo que significa que Jmeter gradualmente fue aumentando el numero de usuarios hasta alcanzar los 500 en 10 segundos. Por último se definió el loop acouunt en 2, lo que implica que cada usuario realizo dos veces la prueba. En la Figura 3.31 se pueden observar los resultados obtenidos para cada uno de los servicios críticos identificados.

Servicio probado	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughp	Received K	Sent KB/s	Avg. Bytes
login	1000	128	79	390	39.52	0.00%	98.32842	94.58	29.58	985
Eliminar reserva	1000	8	4	25	2.72	0.00%	100.06	43.74	42.41	447.6
Crear reserva	1000	1777	15	5568	1288.82	0.20%	62.61741	35.33	34.98	577.7
Cargar las reservas de un usuario	1000	5	3	10	1.06	0.00%	100.0901	49.95	39.49	511
Cargar las reservas de un club	1000	974	19	2833	503.56	0.00%	79.57349	486.14	34.27	6256
Cargar la disponibilidad de un club	1000	7	4	15	1.3	0.00%	100.05	133.95	43.87	1371

Figura 3.31: Resumen de rendimiento de servicios probados

Los resultados de las pruebas de carga realizadas con JMeter revelan que, a pesar de que la mayoría de los servicios evaluados mostraron un rendimiento aceptable, el servicio de creación de reservas se destacó con un promedio de 1,777 ms y un máximo de 5,568 ms, lo que indica que puede ser un punto crítico en términos de rendimiento bajo altos niveles de estrés. En general los resultados sugieren que la aplicación puede manejar cargas moderadas, pero que hay puntos específicos que requieren de atención para garantizar un rendimiento optimo en situaciones de alta demanda.

3.2.3. Seguridad

La seguridad, con base en la Norma ISO25010, es la capacidad de protección de la información y los datos de manera que las personas u otros productos tengan el grado de acceso a los datos adecuado a sus tipos y niveles de autorización.

Dentro de la seguridad, aparecen dos términos importantes a la hora del desarrollo de una aplicación y son la autenticación y la autorización. La autenticación es el proceso de confirmación de la identidad de un usuario y la autorización es el proceso de verificar a que recursos puede acceder un usuario o que acciones pueden realizar. Usualmente para el manejo de los permisos se utilizan roles, los cuales agrupan a los usuarios y permiten asignarles permisos a ciertos recursos de la aplicación.

Gran parte, por no decir la mayor responsabilidad sobre la autenticación y la autorización recae sobre el backend, desde el frontend, basados usualmente en roles, se puede controlar el desligue de pantallas o acciones que un usuario pueda realizar, de no ser así, un usuario podría verse lleno de errores al intentar realizar una acción para la cual el backend no lo dejaría.

Si bien esta es una funcionalidad necesaria para el tipo de aplicación realizada, en donde hay usuarios básicos y usuarios administradores, en WeWeb no es posible hacer uso de roles si no se cuenta con el plan de 149 dólares mensuales. Dejando aparte este dato, la configuración es relativamente sencilla, se crean los roles y sobre cada página de la aplicación se le asignan los roles que la pueden cargar, ver Figura 3.32.

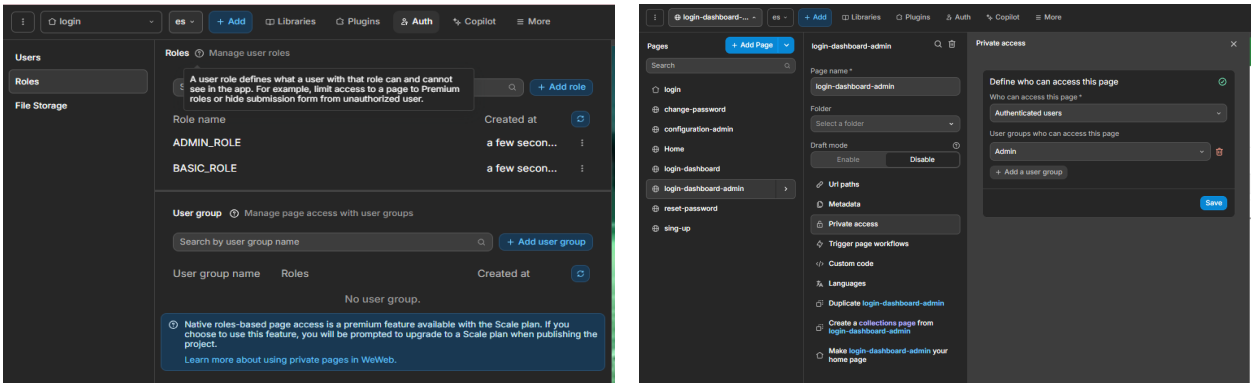

Figura 3.32: Logs en WeWeb al desplegar a producción la aplicación

Por otro lado, con angular se pueden definir los roles de usuario y asociarlos a diferentes componentes y servicios. Esto permite que los usuarios básicos o administradores solo puedan acceder a las funcionalidades y pantallas relevantes para su nivel. Ademas, angular ofrece herramientas como guards e interceptores que permiten proteger rutas, ver La Figura 3.33 en donde se configuro un guard para redirigir al login si el token del usuario no era valido,estos guards se deben configurar a nivel de las rutas de angular.

```
import { TokenService } from '../services/token.service';

export const AuthGuard: CanActivateFn = () => {
  const isValidToken: boolean = inject(TokenService).isValidToken();

  if (!isValidToken) {
    inject(Router).navigate(['/login']);
    return false;
  }
  return true;
};
```

```
const routes: Routes = [
  {
    path: '',
    canActivate: [redirectGuard],
    loadChildren: () =>
      import('./modules/auth/auth.module').then((m) => m.AuthModule),
  },
  {
    path: 'home',
    canActivate: [AuthGuard],
    loadChildren: () =>
      import('./modules/layout/layout.module').then((m) => m.LayoutModule),
  },
  {
    path: '**',
    redirectTo: '',
  },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Figura 3.33: Configuración de guards con angular

La configuración para el control de la autorización de los usuarios es visualmente y procesalmente más sencilla en WeWeb, ya que se realiza a través de una interfaz gráfica en la que simplemente se asignan roles a las páginas o componentes correspondientes. Esto permite a usuarios sin conocimientos técnicos definir las reglas de acceso de manera rápida, sin necesidad de escribir código. Sin embargo, es importante mencionar que el plan en el que se desarrolló el proyecto no incluía esta funcionalidad, y el plan que sí lo permite incrementa considerablemente los costos.

3.2.4. Mantenibilidad

Con base en la ISO 25010 la mantenibilidad representa la capacidad del producto de software para ser modificado efectivamente, debido a necesidades evolutivas, correctivas o perfectivas. Para llevar a cabo la comparación, se hará uso de las siguientes subcaracterísticas, definidas igualmente en la norma ya mencionada.

- Modularidad
- Reusabilidad
- Analizabilidad
- Capacidad para ser probado

Modularidad

"Es la capacidad de un producto para evitar que los cambios en un componente afecten a otro componente", iso (2024)

La plataforma de WeWeb como tal no permite la creación de módulos si no que divide la aplicación en paginas (ver Figura 3.34), sobre cada pagina se pueden hacer uso de diferentes componentes, servicios o acciones configuradas en la aplicación. Al momento del desarrollo, las paginas no podían ser agrupadas, lo que generaba que para poder identificar las paginas de una funcionalidad, tocara utilizar una palabra como por ejemplo admin para las paginas que eran únicas para los usuarios administradores, ver Figura 3.35.

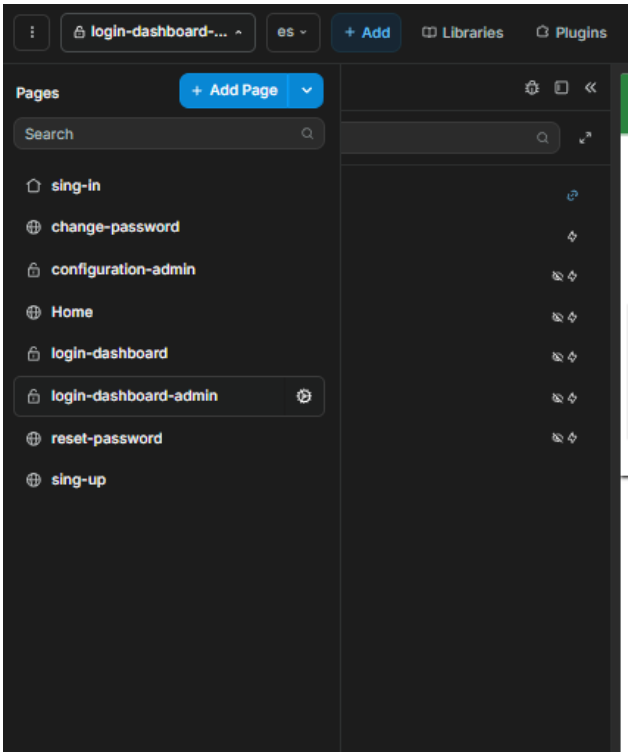

Figura 3.34: Sección para configurar una pagina en WeWeb

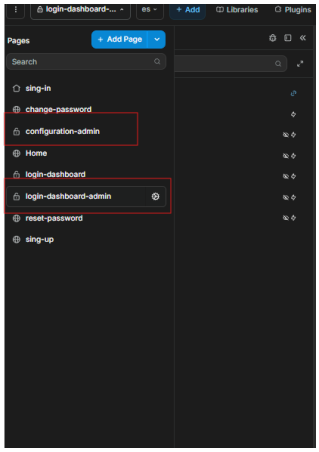

Figura 3.35: Identificador para las paginas de administrador

Si bien WeWeb no permite la creación de módulos, todos los cambios realizados en una página e incluso en un componente creado solo afectan a esa misma página o componente. De tal forma que se mantiene la independencia y la integridad de los demás componentes de la aplicación. Esto garantiza que las modificaciones en una funcionalidad específica no alteren el funcionamiento de

otras páginas, lo que es esencial para el mantenimiento y la estabilidad del sistema.

Por otro lado, Angular si ofrece un enfoque mas robusto para la modularidad en el desarrollo de una aplicación. A diferencia de WeWeb, Angular permite crear de manera independiente módulos que encapsulan tanto componentes como servicios, facilitando mucho mas la organización del código y la reutilización del mismo, ver Figura 3.36. Con la utilización de módulos se puede cargar de manera perezosa la información, lo que mejora notablemente el rendimiento de la aplicación y permite crear rutas sobre las cuales poder navegar sin tener que actualizar toda la pagina, algo que no se encontró como hacer desde WeWeb.

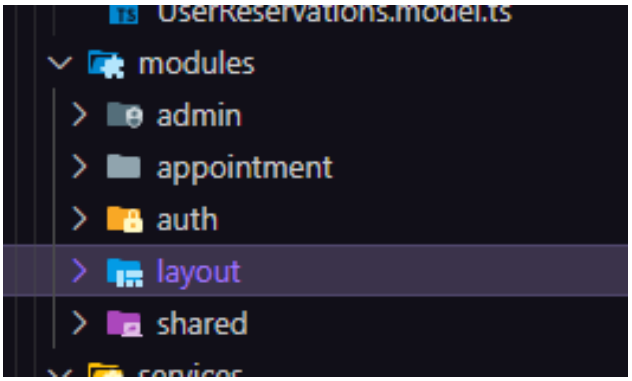

Figura 3.36: Módulos en Angular

3.2.4.1. Reusabilidad

La reusabilidad es definida, en la ISO 25010, como la capacidad de un activo que permite que sea utilizado en mas de un sistema de software o en la construcción de otros activos.

Sobre este item tanto WeWeb como Angular ayudan a cumplir con esta característica. Con ambas herramientas se pueden crear componentes que sean reutilizables a lo largo de la aplicación, lo que evita la duplicidad del código y centraliza la lógica en un solo lugar. Cualquier cambio realizado en un componente se refleja automáticamente en todos los lugares de la aplicación donde se utilice.

Si bien ambas cumplen con esta característica, la forma de implementarlo si es diferente. En WeWeb basta con pararse sobre un la figura que queremos transformar y darle clic en transformar en un componente, . Cuando se crea el componente se le pueden crear propiedades, variables, acciones y eventos que serán utilizados por este, ver Figura 3.37 en donde el componente recibe una información que pinta en el modal de la Figura 3.38.

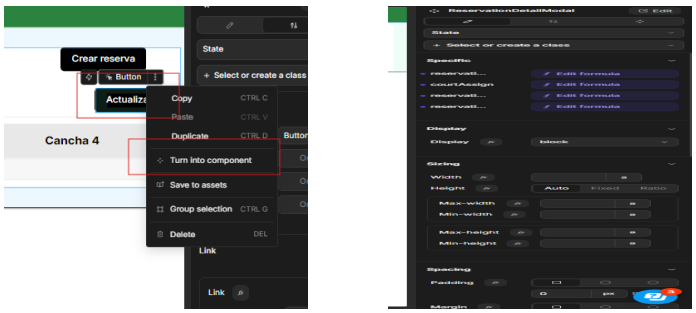

Figura 3.37: Creación de un componente en WeWeb

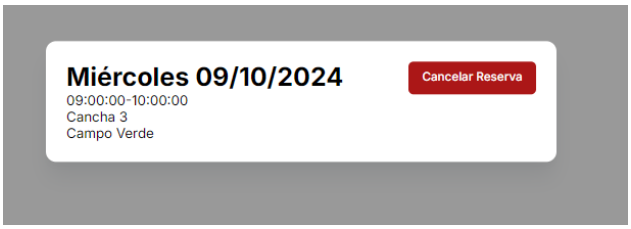

Figura 3.38: Componente creado para el modal de información de una reserva

En Angular por otro lado el proceso se puede realizar mediante el Angular CLI. Primero se ejecuta el comando *ng generate component nombre-del-componente* o su versión abreviada *ng g c nombre-del componente*, ver Figura 3.39. Este comando nos genera los archivos necesarios, el archivo typescripc en donde se define la lógica del componente, el archivo HTML que contiene el código para la vista, el archivo CSS para los estilo y un archivo de pruebas .spec.ts, ver Figura 3.40. El componente se registra automáticamente en el módulo correspondiente , archivo .module.ts, donde es importado y se añaden sus declaraciones, ver Figura 3.41. Una vez creado el componente, se puede hacer uso de el invocando la etiqueta personalizada que se encuentra en el archivo .ts, ver Figura 3.42 y Figura 3.43.

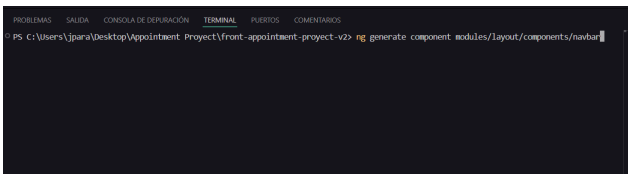

Figura 3.39: Comando para crear un componente en Angular

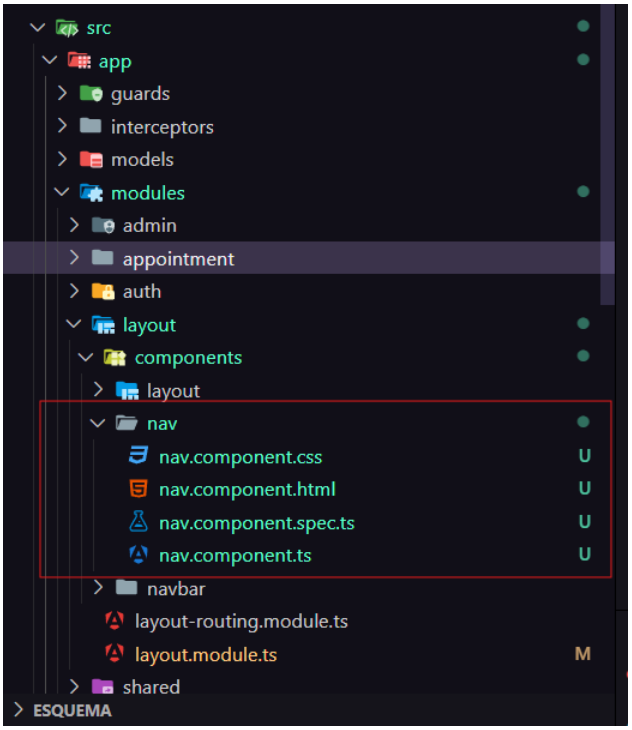

Figura 3.40: Archivos creados por el comando de Angular CLI

```
import { MatDialogModule } from '@angular/material/dialog';
import { MatButtonModule } from '@angular/material/button';
import { AppointmentModule } from '../appointment/appointment.module';
import { NavComponent } from './components/nav/nav.component';

@NgModule({
  declarations: [LayoutComponent, NavbarComponent, NavComponent],
  imports: [
    CommonModule,
    LayoutRoutingModule,
    OverlayModule,
    FontAwesomeModule,
    SharedModule,
    MatDialogModule,
    MatButtonModule,
    AppointmentModule,
  ],
})
export class LayoutModule {}
```

Figura 3.41: Actualización en el archivo module.ts

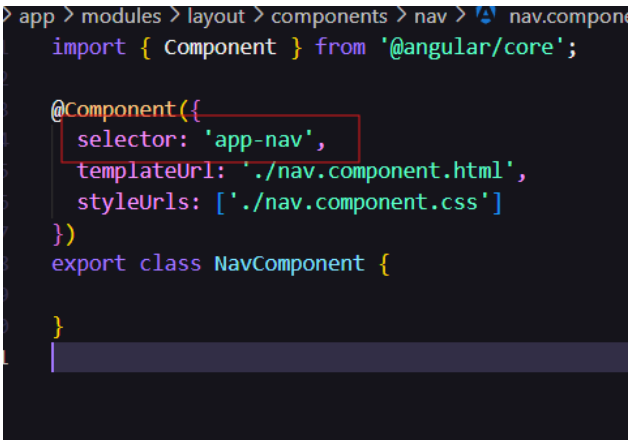

Figura 3.42: Etiqueta para utilizar el componente

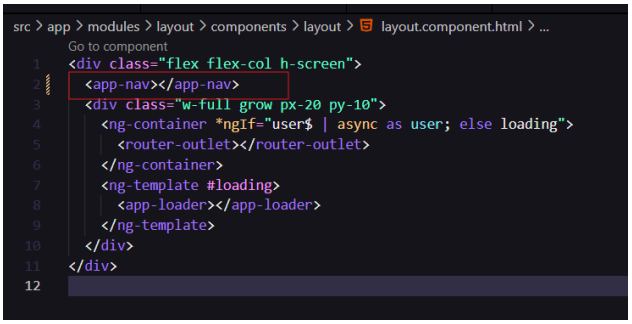

Figura 3.43: Utilización de componente nav en el Layout de la aplicación

3.2.4.2. Analizabilidad

En la ISO 25010, la analizabilidad se define como la facilidad con la que se puede evaluar el impacto de un determinado cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos o identificar partes a modificar.

Sobre esta característica, WeWeb cuenta con un sección para depurar el codigo (ver Figura 3.44), con esta funcionalidad se puede ver el estado de las variables definidas en la plataforma, la ejecución de los flujos configurados y la respuesta de las peticiones realizadas al API, lo que facilita a simple vista poder diagnosticar errores sencillos sobre el comportamiento de variables y datos. Si bien es una herramienta que ayuda a diagnosticar gran parte de los errores, se queda corto a la hora de poder identificar si el cambio en alguna información retornada ha afectado algún componente que la utilizara, esto debido a que principalmente no se pueden generar pruebas automáticas sobre los componentes, una gran desventaja que afecta la calidad y la mantenibilidad del código.

De igual forma, en WeWeb no es posible visualizar los lugares en donde se esta utilizando un componente, por lo que realizar un análisis de impacto se torna complicado y toca hacerlo manual, revisando si en las diferentes paginas configuradas se encuentra el componente.

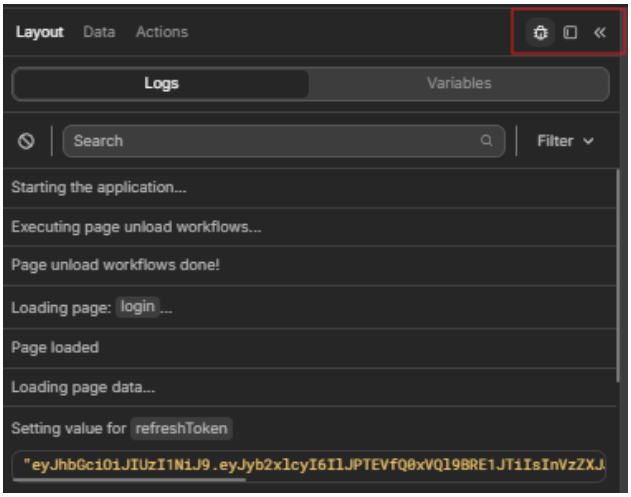

Figura 3.44: Sección para depurar el código en WeWeb

Por otro lado, Angular permite realizar pruebas unitarias y de integración de forma sencilla, lo que permite a los desarrolladores estar seguro de que un cambio no ha afectado el comportamiento esperado de la aplicación, ver Figura 3.45 para el ejemplo de una prueba para un componente. De igual forma, la inyección de dependencias nos brinda una visión clara de como interactúan los diferentes componentes entre si, identificando dependencias y posibles errores por modificaciones.

```
describe('CancelReservationModalComponent', () => {
  let component: CancelReservationModalComponent;
  let fixture: ComponentFixture<CancelReservationModalComponent>;

  const mockDialogRef = {
    close: jasmine.createSpy('close')
  };

  const mockData = {
    selectedReservation$: of({
      date: '2023-10-10',
      initialHour: '10:00',
      endHour: '11:00',
      groupCourtId: '1',
      club: 'Club Deportivo'
    })
  };

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [CancelReservationModalComponent],
      imports: [MatDialogModule],
      providers: [
        { provide: MatDialogRef, useValue: mockDialogRef },
        { provide: MAT_DIALOG_DATA, useValue: mockData }
      ]
    }).compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(CancelReservationModalComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });
```

Figura 3.45: Prueba unitaria para el componente de confirmar reserva en Angular

Para el análisis de impacto, haciendo uso del editor de código es mucho mas sencillo encontrar los puntos en donde un componente se esta utilizando a lo largo de toda la aplicación, ver Figura

3.46 como ejemplo para identificar los puntos en donde se utiliza el componente de input.

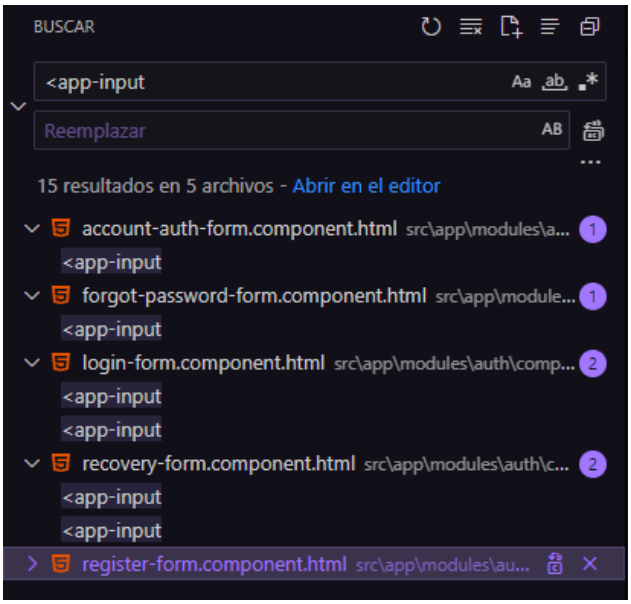

Figura 3.46: Análisis de impacto sobre el componente de app-input

En conclusión, sobre esta característica evaluada, Angular, con el apoyo de herramientas como editores de código, permite diagnosticar de manera mas sencilla y robusta el diagnosticar errores y sobre todo realizar análisis de impacto obre la aplicación.

3.2.4.3. Capacidad para ser probado

La capacidad para ser probado se define, con base en la ISO 25010, como la facilidad con los que se pueden establecer criterios de prueba para un componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.

Como se ha señalado, Angular supera a la plataforma de WeWeb en términos de la capacidad para ser probado el desarrollo. Angular permite crear pruebas unitarias para cada componente, lo que garantiza que cada parte de la aplicación funcione como se espera y de no hacerlo se detectaría inmediatamente el error.

Por otro lado, WeWeb carece de una funcionalidad para la creación de pruebas sobre componentes o sobre la aplicación en general. Esta limitación dificulta el poder validar los cambios y aumenta el riesgo de incluir errores en la aplicación. Sin la capacidad de generar pruebas unitarias o de integración, el desarrollo en WeWeb se vuelve mas propenso a fallos, lo que afecta la calidad de software y complica su mantenimiento a largo plazo.

3.2.5. Desplegabilidad

La desplegabilidad (Deployability en ingles) se refiere a la capacidad de un software para ser desplegado, es decir, asignado a un entorno de ejecución, dentro de un tiempo y esfuerzo predecibles y aceptables. Para comparar la desplegabilidad vamos a evaluar tres puntos que consideramos importantes:

- Configuración para el despliegue de la aplicación
- Tiempo requerido para cargar los cambios a producción
- Logs de despliegue y rollback de los cambios

3.2.5.1. Configuración para el despliegue de la aplicación

Las plataformas Low-code como WeWeb, ofrecen mecanismos automatizados para el despliegue de las aplicaciones, lo que reduce la necesidad de configuración manual y por lo tanto le permite a los desarrolladores enfocarse en el diseño y construcción del producto, sin tener que preocuparse por la complejidad de la integración continua, como pipeline de despliegue o la configuración de infraestructura.

Para hacer un despliegue en WeWeb basta con, desde el proyecto, dar clic en desplegar. Una vez cargue el proyecto, se puede acceder desde la URL provista por la plataforma, esta URL se puede personalizar de así desearlo, ver Figura 3.47.

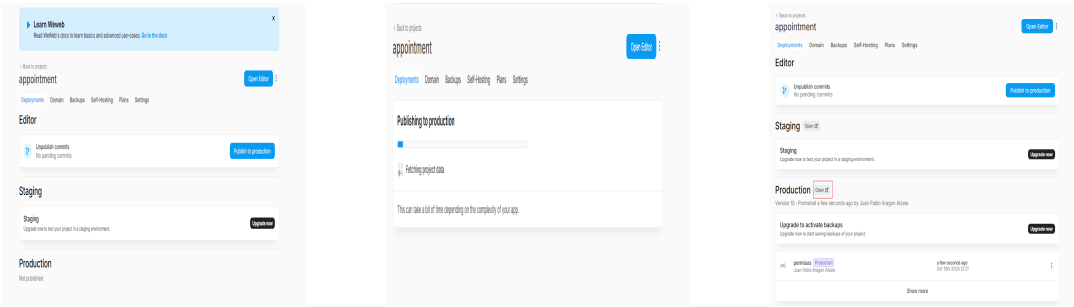

Figura 3.47: Proceso para el despliegue de la aplicación en WeWeb

Por otro lado, Angular, al ser un framework no cuenta con una funcionalidad para el despliegue automático del código a producción. El desarrollador debe configurar manualmente el proceso de despliegue, lo que implica pasos adicionales, como la generación de los archivos de producción, la configuración de pipeline de CI/CD mediante herramientas como Jenkins y la gestión de la infraestructura. Esto de por si ya requiere de un conocimiento mucho mas profundo a simplemente el desarrollo del frontend.

Sin embargo, existe herramientas que simplifican este proceso y permiten realizar el despliegue de forma automática, simplemente cargando el proyecto desde Github. Servicios como Netlify o

vercel ofrecen integración continua y despliegue automatizado al detectar cambios en el repositorio de github, muy similar a lo ofrecido por herramientas Low-Code como WeWeb.

Netlify, en su plan gratuito ofrece despliegues automáticos con cada push generado al repositorio en git. De igual forma hasta 100 GB de ancho de banda y 300 minutos de compilación al mes. Además, ofrece hasta 125,000 invocaciones a funciones serverless al mes y permite hasta 100 envíos de formularios al mes, lo que lo convierte en una buena opción para el despliegue de un pequeño proyecto como el realizado en este trabajo.

La configuración para el despliegue automático de un proyect que se encuenta alojado en Github es bastante sencillo. Una vez iniciada sesión en Netlify, se da clic en " Add new site", se selecciona la plataforma en donde tenemos el proyecto, en esta caso Github y seleccionamos el proyecto, ver Figura 3.53. A continuación, se llena la configuración solicitada, como la rama sobre la que se hará el deploy (en este caso master), el comando para construir el proyecto (en este caso npm run build) y el directorio de publicación(en este caso dist/front-appointment-proyect-v2), ver Figura 3.49. finalmente, y si toda la configuración esta bien, el proyecto queda desplegado sobre una URL provista por netlify, ver Figura 3.50.

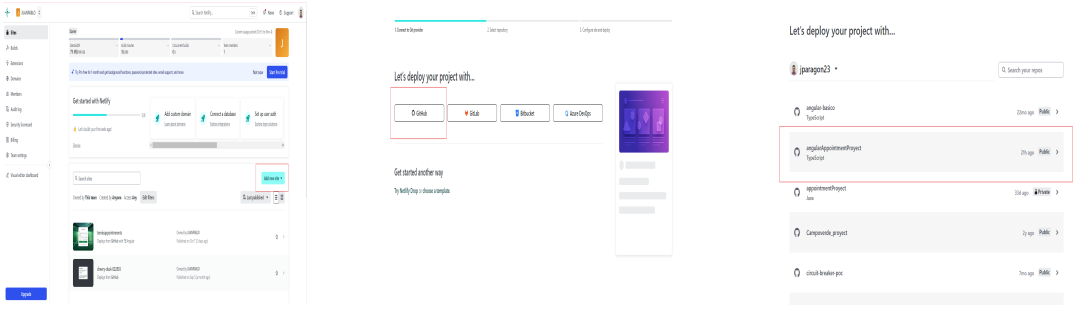

Figura 3.48: Creación de un despliegue en netlify y selección del proyecto en GitHub

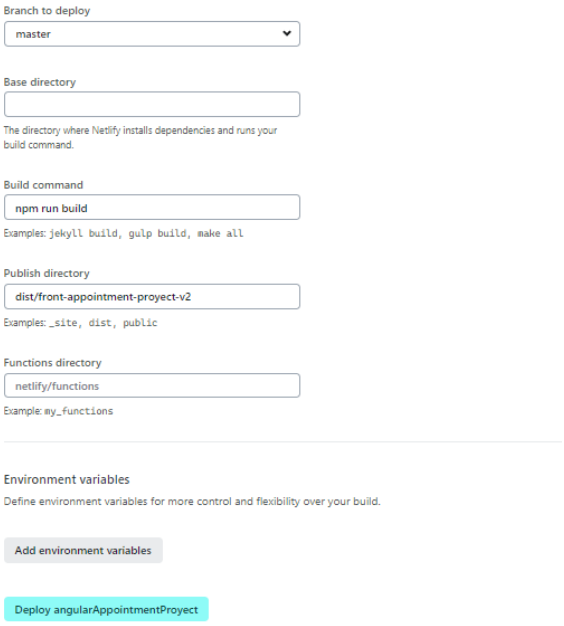

Figura 3.49: Configuración requerida para el despliegue

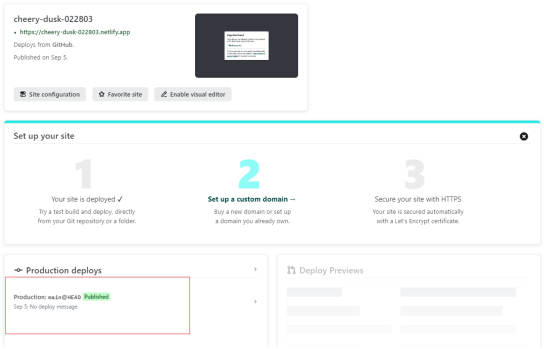

Figura 3.50: Despliegue de la aplicación

En conclusión, aunque WeWeb ofrece la ventaja de tener el proceso de despliegue completamente integrado, la configuración que se debe realizar en Netlify para desplegar un proyecto de Angular no es tan compleja como para considerarla una desventaja significativa y si a eso le sumamos que para poder hacer el despliegue en WeWeb toca tener el plan Starter,por el que hay que pagar 49 dólares mensuales, mientras que hasta cierto punto el despliegue en Netlify es gratis, la ventaja se vuelve difusa.

3.2.5.2. Tiempo requerido para cargar los cambios a producción

Este criterio busca medir la rapidez con la que el sistema puede aplicar las actualizaciones en el entorno de producción, partiendo desde el momento en que se realiza el commit en la rama main. En el caso de WeWeb, para aplicar los cambios a producción se debe, desde el editor del proyecto, dar clic en "Publish", ver Figura 3.51. Por otro lado, con Angular, haciendo uso de Netlify, basta con hacer un push sobre la rama master.

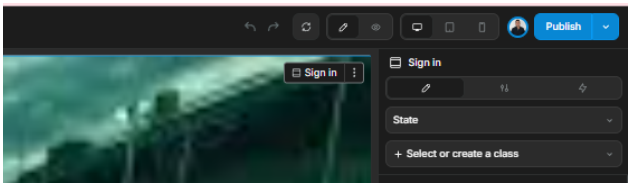

Figura 3.51: Publicar los cambios al ambiente productivo en WeWeb

Para evaluar el rendimiento tanto de WeWeb, como de Netlify en representación del proyecto realizado en angular, se compararon los tiempos en aplicar los últimos 15 commits realizados en cada plataforma. En la Figura 3.52 se puede observar como los tiempos son bastante diferentes, en promedio para WeWeb el tiempo es de 216 segundos, mientras que con Netlify el tiempo es de 49 segundos.

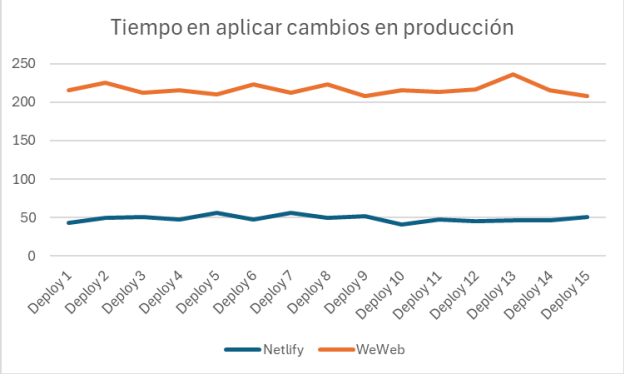

Figura 3.52: Tiempo entre el commit de un cambio y su aplicación en ambiente productivo

En conclusión, aunque ambos sistemas facilitan el proceso de despliegue de la aplicación, al comparar los tiempos de despliegue se observa que Netlify es considerablemente más rápido. En promedio, su rendimiento es casi 4.5 veces superior al de WeWeb.

3.2.5.3. Logs de despliegue y rollback de los cambios

Cuando se despliega una aplicación en un ambiente productivo, tener logs detallados del proceso facilita el proceso de identificación y corrección de errores. Por su parte, el permitir reversar los

cambios a una versión previa (rollback), minimiza los impactos a usuarios y asegura la continuidad del servicio en producción.

WeWeb cuenta con una sección en donde despliega logs al momento de hacer el despliegue a producción pero al ingresar, estos son bastantes sencillos y no ofrecen gran detalle, ver Figura 3.53

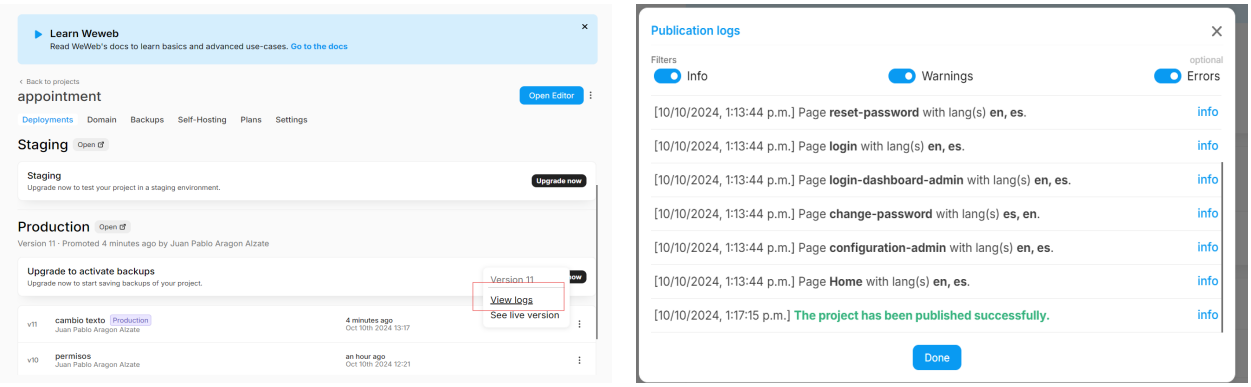

Figura 3.53: Logs en WeWeb al desplegar a producción la aplicación

Por otro lado, netlify, el servicio utilizado para el despliegue de la aplicación en Angular, ofrece logs mucho mas detallados, incluso los agrupa en diferentes categorías como se puede observar en la Figura 3.54. Cuando hay un error, al ser tan detallado los logs, es sencillo poder identificarlo y corregirlo, ver Figura 3.55 en donde se pueden observar los logs generados por un error de despliegue que hubo en el proyecto. Incluso netlify tiene una funcionalidad en donde ofrece un diagnostico del error, ver Figura 3.56.

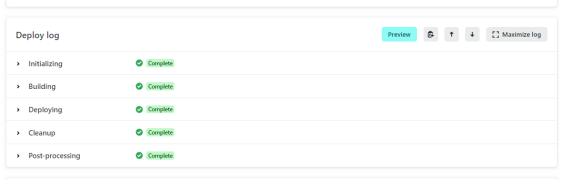

Figura 3.54: Logs disponibles en Netlify

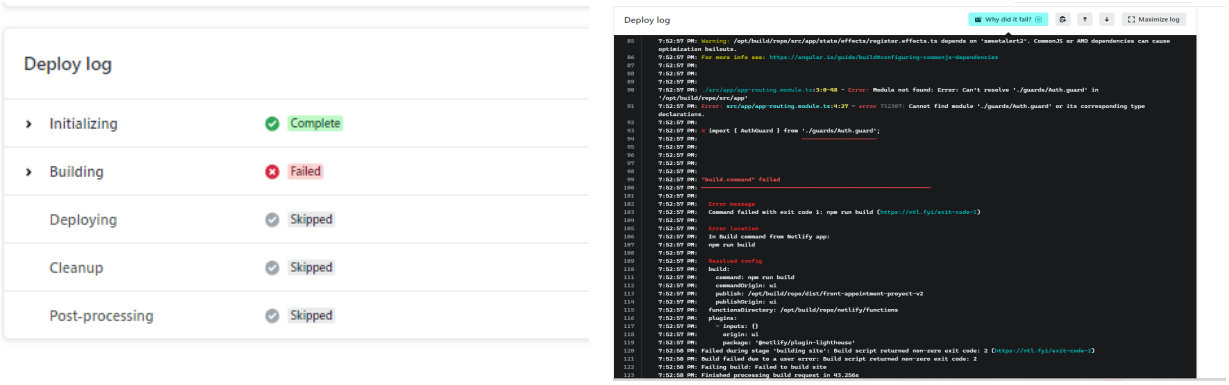

Figura 3.55: Logs generados cuando hay un error en el despliegue en Netlify

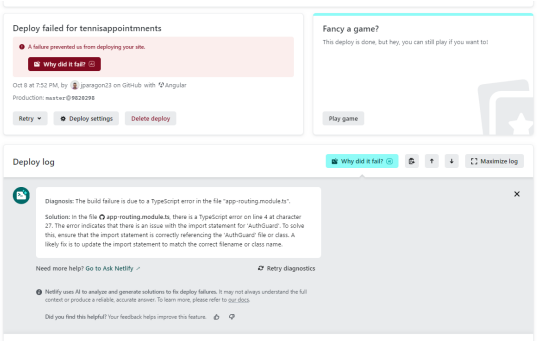

Figura 3.56: Diagnostico ofrecido por Netlify

En cuanto al rollback, WeWeb no permite desplegar directamente una versión anterior ya publicada. Para revertir la aplicación a un versión previa, primero es necesario restaurar el proyecto a un commit anterior desde la pestaña “Backups” y luego volver a publicarlo. Esto implica que técnicamente se realiza un nuevo despliegue en lugar de un simple rollback, como se muestra en la Figura 3.57.

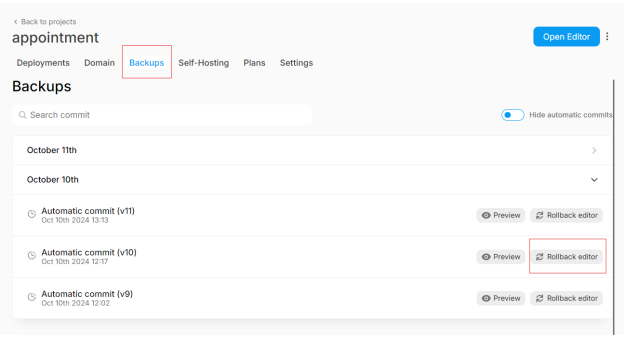

Figura 3.57: Proceso de rollback en WeWeb

Por otro lado, Netlify permite seleccionar un despliegue anterior desde su historial. Al ingresar a la información de un despliegue especifico, dentro de las opciones se encuentra la opción de publicar la versión de ese despliegue a producción, ver Figura 3.58. Una vez publicado, la versión seleccionada será la que se encuentre en el ambiente productivo, ver Figura 3.59.

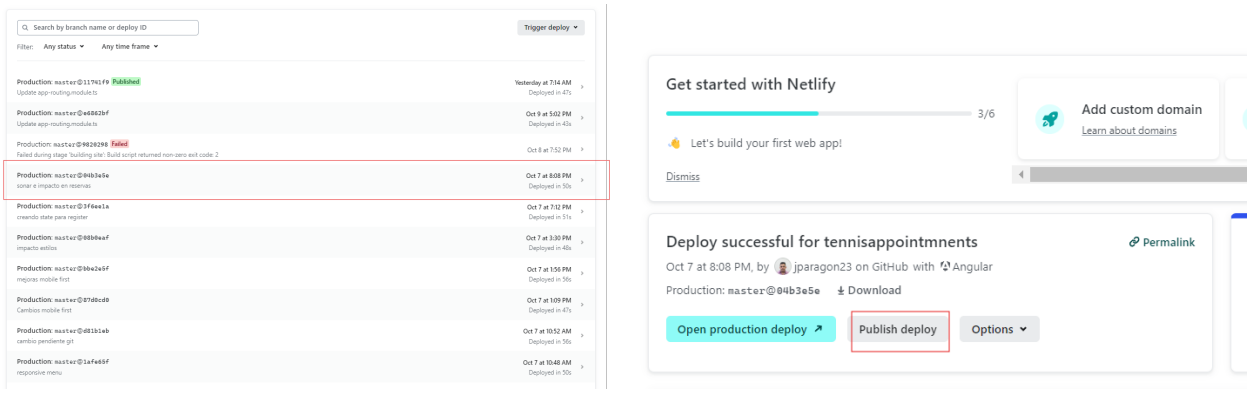

Figura 3.58: Proceso para realizar un rollback a un deploy especifico en Netlify

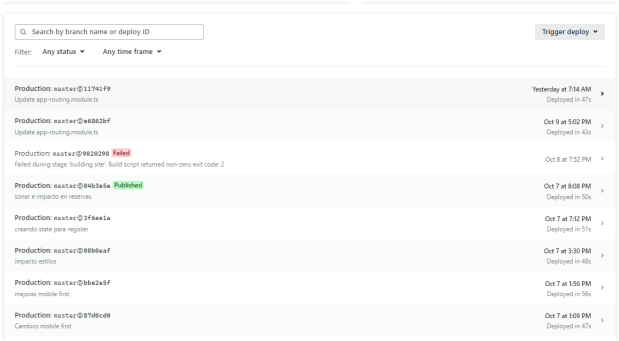

Figura 3.59: Roll back completo en netlify

En conclusión, en términos de trazabilidad del despliegue, Netlify ofrece información más detallada y precisa, lo que facilita enormemente el diagnóstico de errores durante el proceso de despliegue. Además, la herramienta de diagnóstico que proporciona en caso de fallos es especialmente útil para resolver problemas de manera rápida y eficiente.

De igual forma, aunque ambas herramientas permiten realizar un rollback, el proceso es bastante diferente. En Netlify, el rollback es sencillo y directo, permitiendo restaurar una versión anterior con solo unos clics. En cambio, en WeWeb, es necesario revertir el proyecto a un estado anterior desde la pestaña de “Backups” y luego volver a publicarlo, lo que implica modificar el proyecto y realizar un nuevo despliegue. Esto hace que el proceso en WeWeb sea más complejo que en Netlify.

3.2.6. Costo

Como se ha mencionado a lo largo del proyecto, el costo es uno de los factores mas importantes a la hora de poder realizar un proyecto y es el punto en donde las herramientas Low-code prometen impactarlo positivamente.

Para la herramienta de WeWeb el costo total para poder llevar a producción el producto desarrollado se ve representado por el valor que se debe pagar por el plan seleccionado. El plan mas económico, que permite publicar la aplicación es de 49 dólares mensuales. Entre los principales beneficios se encuentra:

- Aplicación en vivo
- Podes utilizar dominios propios
- 50.000 visitas al mes
- 1GB de almacenamiento de archivos
- 200 GB de ancho de banda
- Exportar código y hosting propio
- 1 copia de seguridad diaria

Si bien el plan nos ofrece algunas funcionalidades esenciales, deja por fuera muchas que podrían ser consideradas fundamentales para sacar una aplicación robusta al mercado y por el que habría que pagar 1479 dólares mensuales, como lo pueden ser:

- Acceso a paginas por roles
- Versionamiento de la aplicación

Por otro lado, dejando por fuera el costo humano del desarrollo, que tampoco se tiene en cuenta en el desarrollo con WeWeb, el mayor costo para llevar la aplicación realizada con angular a producción seria el del servicio de Hosting, que recordemos se realizó con Netlify y que en su plan gratuito ofrece:

- 100 GB de ancho de banda
- 300 minutos en construcción de la aplicación
- Rollbacks instantáneos
- Funciones dinámicas serverless

Ya pasando al plan Pro, por 19 dólares mensuales, el ancho de banda aumenta a 1TB mensual y el tiempo para construcción de aplicaciones aumenta a 25.000 minutos.

Enfocándonos netamente en el costo mensual para poder desarrollar una aplicación robusta que este disponible para los usuarios, el desarrollo con angular, apoyándose en Netlify, es mensualmente mas económico y completo que lo que se puede realizar con WeWeb con el plan que es medianamente accesible para un proyecto inicial .

3.2.7. Rapidez

La rapidez en el desarrollo es quizás la principal característica de las herramientas Low-code. El utilizar componentes y flujos de trabajo preconstruidos, disminuye considerablemente el tiempo de desarrollo. Para la comparación se tomaron los tiempos de desarrollo para cada una de las principales funcionalidades de la aplicación. Se buscó que el alcance funcional y visual fuera muy parecido, esto para que los tiempos no se diferenciaran por la diferencia en el diseño. En la Tabla 3.5 se puede observar el tiempo, en horas, que demoró el desarrollo de cada funcionalidad con WeWeb y con Angular.

Funcionalidad	**WeWeb**	**Angular**
Inicio de sesión	2	5
Registro de usuarios	3	5
Recuperación de contraseña	1	2
Dashboard usuario base	5	8
Crear una reserva usuario base	4	6
Eliminar una reserva usuario base	2	3
Dashboard usuario administrador	4	6
Vista de configuración para un club	2	3
Crear una reserva usuario administrador	2	3
Eliminar una reserva usuario administrador	3	5
Despliegue de la aplicación	3	1
TOTAL	**28**	**46**

Tabla 3.5: Comparación del tiempo de desarrollo

En términos generales, el desarrollo de cada funcionalidad en WeWeb tomó menos tiempo en comparación con Angular, principalmente debido al uso de componentes preconstruidos que la plataforma ofrece. Esta característica permite una integración rápida y eficiente de las funcionalidades, reduciendo la necesidad de implementar lógica personalizada o manipulación de interfaces desde cero.

Por el contrario, replicar los mismos comportamientos en Angular requirió de un 64 % más de tiempo, ya que fue necesario realizar investigaciones adicionales y buscar soluciones específicas para implementar ciertos comportamientos. Esto se debe a que Angular proporciona un mayor control y flexibilidad, pero al mismo tiempo exige una mayor dedicación en la construcción y configuración

de las funcionalidades, desde la base hasta los detalles más específicos del frontend.

3.2.8. Robustez en el desarrollo

En términos de robustez, el desarrollo de angular destaca sobre el de WeWeb. Si bien, como se mencionó anteriormente, Weweb como plataforma low-code permite un desarrollo mas ágil, la robustez se ve limitada por el uso de los componentes y de la lógica preconstruida por la herramienta. Estas limitaciones pueden hacer que el manejo de casos complejos sea más difícil de lograr sin recurrir a soluciones externas o personalizaciones que escapan al alcance de la herramienta.

Por otro lado, angular al ser mas flexible y robusto, permite un mayor control sobre el código y la arquitectura en general de la aplicación. Esta flexibilidad hace que el desarrollo sea mas complejo y demorado, pero a su vez hace que el desarrollador conozca la forma en que funciona la aplicación y pueda personalizarla según las necesidades del proyecto. Además, esta comprensión profunda del código facilita el mantenimiento y la evolución de la aplicación a largo plazo, ya que desarrollador puede ajustar o extender las funcionalidades de manera precisa y eficiente, sin estar limitado por las restricciones impuestas por una plataforma de bajo código.

3.2.9. Conclusión de la comparación realizada

Sobre cada uno de los criterios y subcriterios se ha realizado una pequeña conclusión, pero en esta sección se busca realizar un resumen y una calificación cuantitativa sobre cada uno de los criterios. Primero se asigna un peso a cada uno de los 7 criterios seccionados, del 1 al 5 en donde 5 es lo mas importante, basados principalmente en los resultados de la consulta de expertos realizada en la sección 3.1.3 y en las siguientes premisas:

- Las aplicaciones front-end dependen de un backend para su funcionamiento, que por lo general se comunican mediante llamadas HTTP a un API, por tal motivo, la compatibilidad es sumamente importante para facilitar la configuración y comunicación entre los dos sistemas, por tal motivo se le asigna un valor de 4.
- El desempeño, si bien es una característica muy importante, quizás en las fases iniciales, cuando el numero de usuarios no es tan grande,no resulta tan crítico. Por tal motivo se asigna un peso de 3.
- La seguridad es un criterio critico, por mas completa que sea una aplicación, si no es segura, puede comprometer no solo su funcionamiento si no la información delicada de sus usuarios. Por tal motivo, se asigna un valor de 5.
- En la fase inicial del desarrollo de un producto, la mantenibilidad quizás no es una característica que sea crítica pero conforme vaya creciendo la aplicación si es indispensable, por tal motivo, enfocado en la fase inicial se asigna un valor de 3.
- El poder desplegar la aplicación sin mucha configuración y sobre todo contando con información pertinente por si falla el proceso es bastante importante a la hora de tener una aplicación en producción, por tal motivo se asigna un peso de 4.

- El costo es un criterio bastante importante, esto debido a que por lo general, para emprendimientos no se cuenta con grandes cantidades de recursos para lanzarlos al mercado, por tal motivo se le asigna un peso de 5.
- Teniendo en cuenta que para que un producto tenga mayor probabilidad de sobrevivir en el mercado debe llegar primero que sus competidores, y sobre la marcha ir validando, probando e iterando, se asigna un valor de 5 al criterio de rapidez.
- En la fase inicial del desarrollo de un producto, la robustez es importante para asegurar que la aplicación funcione correctamente bajo diferentes condiciones, aunque no es esencial que sea perfecta en esta etapa. No obstante, se le asigna un valor de 4, ya que es fundamental minimizar errores y garantizar una experiencia estable para los primeros usuarios.

Con base en las calificaciones anteriormente, en la Tabla 3.6 se encuentra el resumen del peso y la calificación, de igual forma de 1 a 5, donde 5 es la máxima calificación, de las herramientas en cada criterio.

Peso	**4**	**3**	**5**	**3**	**4**	**5**	**5**	**4**	**TOTAL**
	Compatibilidad	**Desempeño**	**Seguridad**	**Mantenibilidad**	**Desplegabilidad**	**Costo**	**Rapidez**	**Robustez**	
WeWeb	4	2	3	3	3	2	5	3	105
Angular	5	5	4	5	5	5	3	5	150

Tabla 3.6: Comparación de plataformas según criterios ponderados

Con base en los resultados presentados en la tabla, se puede observar que a nivel general el desarrollo realizado en angular resalta sobre los criterios definidos. Si bien algunos de los criterios están parejos,en criterios importantes como el costo, la mantenibilidad, desempeño, desplegabilidad y la robustez angular está por encima de WeWeb.

3.3. Resumen del capítulo

En este capítulo se detallan los pasos realizados en el desarrollo de dos aplicaciones, comenzando con la selección de la herramienta No-Code, que se eligió por su capacidad de permitir el desarrollo rápido y eficiente sin necesidad de escribir código. Posteriormente, se seleccionó un framework adecuado para la implementación de la solución, tomando en cuenta sus características y ventajas. Se establecieron criterios de comparación basados en factores clave como la facilidad de uso, escalabilidad, integración con otras plataformas y coste.

A continuación, se definieron los requisitos funcionales y no funcionales de la aplicación, considerando aspectos como la gestión de reservas, la experiencia del usuario y el rendimiento. Con estos elementos claros, se procedió al desarrollo de las dos aplicaciones, cada una utilizando la herramienta y el framework seleccionados. Finalmente, se realizó una comparación entre ambas aplicaciones con base en los criterios previamente establecidos, evaluando su rendimiento, usabilidad y alineación con los objetivos planteados al inicio del proceso.

Capítulo 4

Conclusiones

4.1. Conclusiones

1. Para pequeños proyectos hacer uso de herramientas empresariales solo genera un aumento en los costos y la complejidad del desarrollo. Para estos casos, como se realizó en el proyecto, hacer uso de herramientas menos complejas y económicas puede ayudar a desarrollar ideas y probarlas en el mercado. Dentro de las herramientas identificadas en el trabajo, Weweb presento un buen equilibrio entre costo y beneficios, es una aplicación con la que se pueden realizar aplicaciones funcionales en poco tiempo.

2. Dentro de los criterios seleccionados y para el contexto del desarrollo de este proyecto, la herramienta de WeWeb solo destacó en la rapidez del desarrollo lo que lo hace ideal para demos o aplicaciones que no cuenten con una gran lógica o complejidad.

3. Aunque el costo suele ser uno de los argumentos principales para promover el uso de herramientas low-code, se evidencia que estas plataformas, si bien ofrecen funcionalidades básicas de forma gratuita, imponen costos significativamente altos para acceder a características necesarias para llevar una aplicación al mercado, especialmente en proyectos pequeños. Sin embargo, en el caso de que el proyecto sea desarrollado por alguien sin conocimientos técnicos en desarrollo, podría resultar más económico pagar la suscripción mensual de la plataforma que contratar a un desarrollador externo para realizar el proyecto.

4. La simplicidad con la que se puede desarrollar una aplicación con una herramienta low-code, genera que características como la mantenibilidad se vean grandemente afectadas. El desconocer como se estructura y genera el código hace que su diagnostico y mantenibilidad sea mas complicado.

5. Aunque la curva de aprendizaje en Angular es más pronunciada que en una herramienta de low-code, el proceso de aprendizaje se realiza una sola vez y, con los conocimientos adquiridos, es posible desarrollar proyectos mucho más complejos y eficientes. Además, como profesional, dominar un framework como Angular resulta ventajoso, ya que muchas empresas trabajan con este tipo de tecnologías y, en consecuencia, requieren de profesionales con estos conocimientos para el mantenimiento, evolución y escalabilidad de sus proyectos. Esto no solo amplía las oportunidades laborales, sino que también permite estar mejor preparado para abordar desafíos técnicos de mayor envergadura.

6. Si bien hacer uso de los componentes pre-fabricados de weweb facilita el desarrollo, nos vemos limitados por la funcionalidad que estos tengan ya. Generar nuevos componentes con comportamientos mas específicos se hace mucho mas complicado en esta herramienta que haciéndolos directamente desde angular.

7. Aunque WeWeb permite desarrollar aplicaciones con funcionalidades básicas de manera efectiva, optar por Angular ofrece mayores ventajas en términos de escalabilidad a largo plazo. Además, los costos asociados con llevar la aplicación a producción y mantenerla pueden resultar más económicos en comparación, especialmente en proyectos que requieran personalización avanzada o crecimiento continuo.

8. La selección de una herramienta adecuada para un emprendedor puede no ser tan compleja, ya que depende principalmente del nivel de conocimiento técnico que posea. Para una persona sin experiencia en programación, las herramientas low-code ofrecen una oportunidad invaluable para materializar ideas que, en el pasado, habrían sido casi imposibles de desarrollar sin una gran inversión o un equipo especializado. Sin embargo, desde el ámbito empresarial, la decisión es considerablemente más compleja, ya que implica evaluar factores como la escalabilidad de la solución, los costos a largo plazo, la compatibilidad con los sistemas existentes, la capacidad de personalización, y el soporte necesario para mantener y evolucionar la herramienta en un entorno más exigente. Estas consideraciones hacen que la elección de la tecnología sea un aspecto estratégico clave para el éxito del negocio.

4.2. Trabajos futuros

Se espera darle continuidad a la aplicación desarrollada con angular, recopilando los comentarios de los usuarios a esta primera versión e ir ajustando y adicionándole funcionalidades.

1. Añadir un módulo para la gestión de torneos
2. Añadir un ranking para los usuarios de la aplicación
3. Mejorar la interfaz de usuario para que cada interacción sea lo mas sencilla posible
4. Soportar múltiples clubes desde una sola aplicación

De igual forma se espera poder profundizar en las siguientes investigaciones:

1. Comparación entre dos herramientas low-code no empresariales
    - Selección de herramientas orientadas a proyectos pequeños (ej.: Adalo, Glide).
    - Análisis de características clave:
        - Facilidad de uso e interfaz de usuario.
        - Opciones de personalización.

    - Costos y planes de suscripción.
    - Integraciones disponibles (APIs, bases de datos).
  - Revisión de casos de uso relevantes para evaluar su efectividad.

2. Desarrollo grupal haciendo uso de herramientas low-code
   - Capacidades de colaboración en tiempo real.
   - Manejo de control de versiones y roles de usuario.
   - Desafíos comunes en equipos grandes o multidisciplinarios.
   - Recomendaciones para optimizar la coordinación en proyectos grupales.
3. Desarrollo convencional con el apoyo de la inteligencia artificial
   - Herramientas relevantes: GitHub Copilot, ChatGPT, CodeGPT.
   - Impacto en la productividad del desarrollador.
   - Casos de uso: generación de código, refactorización, depuración.
   - Comparación entre desarrolladores principiantes y avanzados en el uso de IA.
4. Desarrollo con herramientas low-code impulsadas por agentes de inteligencia artificial
   - Introducción a herramientas que integran IA para generación automática de lógica y diseño (ej.: AppGyver, Betty Blocks).
   - Comparativa de soluciones generadas por IA frente a configuraciones manuales.
   - Impacto en la personalización y velocidad de desarrollo.
   - Limitaciones actuales y potencial de mejora.
5. Experiencia empresarial con el uso de herramientas low-code
   - Casos de uso en empresas: automatización de procesos, creación de dashboards, prototipado rápido.
   - Beneficios empresariales: reducción de costos, disminución del tiempo de desarrollo, independencia del área técnica.
   - Limitaciones empresariales: personalización avanzada, escalabilidad y dependencias de la herramienta.
   - Ejemplos de empresas que han implementado con éxito low-code en sus operaciones.
6. Eficiencia en el desarrollo de prototipos con herramientas low-code
   - Ventajas de utilizar low-code para prototipos rápidos.
   - Evaluación de la fidelidad del prototipo en comparación con el producto final.
   - Impacto en la comunicación entre equipos técnicos y no técnicos.

- Casos prácticos donde el prototipado con low-code ha sido clave para el éxito.

7. Comparativa entre herramientas low-code empresariales y no empresariales
   - Diferencias en capacidades y casos de uso (ej.: Mendix vs. Adalo).
   - Costos y licenciamiento según la escala del proyecto.
   - Análisis de escalabilidad y soporte técnico.
   - Contextos en los que se recomienda cada tipo de herramienta.
8. Integración de herramientas low-code con ecosistemas existentes
   - Integraciones comunes: CRM, ERP, bases de datos externas.
   - Revisión de compatibilidad y facilidad de implementación.
   - Ejemplos de integración exitosa en entornos híbridos (low-code y desarrollo tradicional).
9. Perspectiva futura del desarrollo low-code
   - Potencial evolución de estas herramientas con avances tecnológicos.
   - Rol de la inteligencia artificial en el desarrollo low-code.
   - Predicciones sobre la adopción de low-code en el mercado global.
   - Análisis de posibles desafíos futuros: seguridad, regulación y personalización.

Bibliografía

(2024). Iso/iec 25010. https://iso25000.com/index.php/normas-iso-25000/iso-25010. Accedido: 2024-10-16.

Amazon Web Services (2023). Front End frente a back-end: diferencia entre el desarrollo de aplicaciones - AWS. https://aws.amazon.com/es/compare/the-difference-between-frontend-and-backend/.

BCG (2020). La transformación digital puede aumentar la rentabilidad de las empresas y construir resiliencia. https://www.bcg.com/press/26november2020-digital-transformation-increase-business-profitability-build-resilience.

Bock, A. C. and Frank, U. (2021a). In search of the essence of low-code: An exploratory study of seven development platforms. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 57–66.

Bock, A. C. and Frank, U. (2021b). Low-code platform. *Business and Information Systems Engineering*, 63:733–740.

El País (2023). Descubra el top 5 de deportes más populares en 2023.

Gartner (2022). Gartner forecasts worldwide low-code development technologies market to grow 20 % in 2023.

George, B. and Paul, J., editors (2020). *Digital transformation in business and society: Theory and cases*. Springer.

Hylton, D., Sung, S., and Xie, C. (2021). Adopting no-code methods to visualize computational thinking.

Kaur, G. and Tiwari, R. G. (2023). Comparison and analysis of popular frontend frameworks and libraries: An evaluation of parameters for frontend web development. *2023 4th International Conference on Electronics and Sustainable Communication Systems, ICESC 2023 - Proceedings*, pages 1067–1073.

Kirchhof, J. C., Jansen, N., Rumpe, B., and Wortmann, A. (2023). Navigating the low-code landscape: A comparison of development platforms. pages 854–862. Institute of Electrical and Electronics Engineers Inc.

Luo, Y., Liang, P., Wang, C., Shahin, M., and Zhan, J. (2021). Characteristics and challenges of low-code development: The practitioners perspective. IEEE Computer Society.

Martinez, E. and Pfister, L. (2023). Benefits and limitations of using low-code development to support digitalization in the construction industry. *Automation in Construction*, 152:104909.

Moskal, M. (2021). No-code application development on the example of logotec app studio platform. *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Srodowiska*, 11:54–57.

Okano, M. T., Inoue, P. K., Simões, E. A., and Batista, R. (2021). Business models in the digital transformation era. In Otola, I. and Grabowska, M., editors, *Business models: Innovation, digital transformation, and analytics*, pages 13–31. CRC Press.

Redhat (2018). Intelligent process automation and the emergence of digital automation platforms the transformation of application development and its emerging role to enable new competitive advantage f e b r u a r y 2 0 1 8 pat h f i n d e r r e p o rt.

REDHAT (2023). ¿qué es una api rest?

Rosłan, A. and Śmiałek, M. (2023). Comparative analysis of low-code computation systems. volume 36, pages 103–110. PTI.

Sahay, A., Indamutsa, A., Ruscio, D. D., and Pierantonio, A. (2020). Supporting the understanding and comparison of low-code development platforms. *Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020*, pages 171–178.

Sanchis, R., Óscar García-Perales, Fraile, F., and Poler, R. (2020). Low-code as enabler of digital transformation in manufacturing industry. *Applied Sciences (Switzerland)*, 10.

Trigo, A., Varajao, J., and Almeida, M. (2022). Low-code versus code-based software development: Which wins the productivity game? *IT Professional*, 24:61–68.

Università, G. M., Studi, D., Tre, R., and Masili, G. (2023). No-code development platforms: Breaking the boundaries between it and business experts.

Varajão, J., Trigo, A., and Dias, M. F. (2023). Low-code development productivity. *ACM queue*, 21(5):87–107.

Vincent, P., Natis, Y., Iijima, K., Wong, J., Ray, S., Jain, A., and Leow, A. (2020). Licensed for distribution magic quadrant for enterprise low-code application platforms.