UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Agilizando los cambios de UI-UX sobre el ambiente productivo mediante Figma

AUTORES: Rodrigo Nicolás Martin – Germán Nicolás Ollé

DIRECTOR: Dra. Alejandra Garrido

CODIRECTOR: Dr. Julián Grigera

ASESOR PROFESIONAL: -

CARRERA: Licenciatura en Sistemas

Resumen

Actualmente las metodologías ágiles son la opción elegida por prácticamente todos los equipos de desarrollo para crear un producto de software. En este marco, no es común que un equipo de desarrollo priorice tiempo y esfuerzo en la mejora de estos sistemas en base a su diseño de interfaz, experiencia de usuario o usabilidad. En este trabajo proponemos un nuevo enfoque de desarrollo utilizando una integración con la plataforma de diseño Figma, la cual permite al diseñador poder aplicar cambios de interfaz en ambientes funcionales desde la plataforma, generando un ahorro de tiempo significativo en la entrega de tareas. Se realizó un experimento con resultados muy positivos los cuales dieron lugar a la propuesta de diferentes trabajos futuros para lograr una posible adopción por parte de la industria.

Palabras Clave

Figma, integración, plugin, metodologías ágiles, diseño de interfaz, experiencia de usuario, usabilidad, kit UI, componentes visuales, Scrum, experimento, React, Ruby on Rails, Material-UI.

Conclusiones

El trabajo realizado permite al diseñador poder aplicar cambios de interfaz en ambientes funcionales generando un ahorro de tiempo significativo en la entrega de software y logrando mayor consistencia en el diseño de interfaz del sistema. Se realizó un experimento con el fin de comparar tiempos entre el enfoque convencional y el propuesto en este trabajo, dando resultados muy positivos en este último. A su vez se obtuvo feedback valioso que dio lugar a la propuesta de diferentes trabajos futuros para la adopción de este enfoque por parte de la industria.

Trabajos Realizados

- *Desarrollo de plugin capaz de integrar la plataforma Figma con cualquier aplicación web para poder realizar transformaciones en la interfaz de usuario sin la necesidad de un desarrollador de software como intermediario entre la aplicación y el diseñador.*
- *Se realizó una especificación tanto de las convenciones utilizadas para construir diseños anotados y comentados en Figma como de las que se deben utilizar a la hora de construir la aplicación que consuma esos diseños.*
- *Se realizó un experimento para obtener conclusiones en cuanto a mejoras en el tiempo de desarrollo utilizando dicha integración.*

Trabajos Futuros

- *Cambio de tecnologías utilizadas en el plugin.*
- *Mayor integración de componentes y conceptos.*
  - *Mayor integración de componentes visuales.*
  - *Integración de posicionamiento.*
  - *Integración responsive.*
- *Aplicabilidad de A-B testing.*
- *Draft entre Figma y aplicación web asociada.*

Fecha de la presentación: **Agosto 2020**

Agradecimientos

Queremos darle las gracias a todos los profesores, ayudantes, compañeros y amigos de la Facultad que a lo largo de la carrera nos han ayudado a crecer compartiendo sus conocimientos y experiencias.

También queremos agradecer a la Dra. Alejandra Garrido y al Dr. Julián Grigera, quienes nos apoyaron continuamente en el desarrollo de este trabajo con la mejor predisposición y paciencia.

Mención especial debemos hacer a nuestros amigos Sofía Fernández Gavio, Matías Pérez, Juan Manuel Machado, María Guillermina Vescovo y Alan Toris, quienes con total voluntad y entusiasmo nos ayudaron a concretar exitosamente el experimento propuesto en el final de este trabajo.

Por último, queremos agradecer a nuestras familias, amigos y afectos durante este largo y difícil camino. Nada de esto hubiese sido posible sin ustedes.

CAPÍTULO 1

Introducción

1.1 Motivación

Internet ha revolucionado el ámbito de las comunicaciones de una manera radical hasta el punto de llegar a convertirse en un medio global de comunicación cotidiano en nuestras vidas. Lo utilizamos para muchas cosas, desde compartir un momento con un amigo enviando una foto a través de mensajería instantánea hasta pedir una pizza, comprar electrodomésticos o incluso acceder a nuestra cuenta bancaria para realizar transacciones.

El proceso de desarrollo de estos productos de software que utilizamos día a día ha avanzado a pasos agigantados en los últimos años donde el uso de metodologías ágiles [Beck04; Rubin12] predomina como opción principal a la hora de comenzar un desarrollo debido a los resultados positivos que ha conseguido. Éstas metodologías se caracterizan por ser muy reactivas al cambio y por realizar entregas frecuentes de software funcional en cortos períodos de tiempo a partir del feedback inmediato que reciba del cliente o usuarios finales.

No obstante, estos sistemas pueden presentar problemas de diseño y usabilidad en su interfaz de usuario (UI) que los hacen muy difíciles de usar [Nielsen06]. La mayoría de los usuarios en línea tienen menos probabilidades de regresar a un sitio web después de una mala experiencia, por lo tanto, proporcionar a los visitantes del sitio web una experiencia completa y agradable es de suma importancia [Krug14; Bačíková15].

Dentro de este marco ágil es muy poco común que un equipo de desarrollo priorice tiempo y esfuerzo en la mejora de la aplicación en base a su diseño de interfaz, experiencia de usuario, usabilidad, responsiveness, entre otros, en contraste con el desarrollo que agrega valor funcional al producto [Giacomelli18; DaSilva18].

Este proceso de cambio de interfaz puede llevar días, semanas e incluso meses en caso que tengamos un equipo de desarrollo con muchas personas, donde la cantidad de filtros que deba pasar el nuevo código sea bastante grande. Por lo tanto, el diseñador tiene una larga espera sólo para poder realizar una nueva prueba de usuario con su nuevo diseño donde no está asegurado al 100% que tendrá éxito. Incluso en el peor de los casos, que no es poco común, puede que el desarrollador ni siquiera llegue a desarrollar estos cambios pedidos por el diseñador, porque en general las tareas relacionadas a la funcionalidad, arreglo de errores, seguridad, conectividad, performance, suelen tener mayor prioridad.

A su vez no es tarea fácil lograr una buena interacción entre el equipo de desarrollo y el equipo de diseño dentro de un contexto ágil y a su vez centrado en el usuario. Ambos

enfoques presentan tareas diversas que conllevan tiempos muy distintos, y la sincronización se hace muy dificultosa. Existen diversos estudios que recomiendan la comunicación fluida entre el equipo de desarrollo y el equipo de diseño para lograr sobrepasar estas diferencias, pero no existe evidencia empírica de un marco de trabajo donde se pueda lograr una buena sincronización.

Investigando diferentes plataformas utilizadas por diseñadores, nos encontramos con Figma [Figma19], una herramienta de diseño que provee una API para consumir información de sus archivos de diseño. A partir de esta API, se puede obtener y extraer cualquier objeto o capa de un archivo con sus respectivas propiedades relacionadas a estilos, para después ser consumidas, por ejemplo, por una aplicación web. Utilizando esta herramienta, los diseñadores podrían manipular los cambios pertinentes a la interfaz de usuario, y esos cambios serían aplicados directamente al producto de software a través de la API de Figma.

1.2 Objetivos

En este contexto, esta tesina tiene como objetivo darle la posibilidad al diseñador de poder tomar la responsabilidad de realizar los cambios pertinentes a interfaz de usuario que tienen que ver con factores como usabilidad y experiencia del usuario, y que el desarrollador sólo se concentre en el trabajo relacionado con los aspectos funcionales y lógica de negocio de la aplicación. De este objetivo principal se desprenden los siguientes objetivos específicos:

- Proponer la integración del diseñador de UI/UX al equipo de desarrollo ágil, de manera que pueda no sólo proveer diseños para la aplicación, sino también aplicar cambios en la interfaz y usabilidad directamente sobre ambientes funcionales.
- Permitir que el diseñador pueda hacer pruebas de alternativas de UI/UX durante el desarrollo, sobre ambientes funcionales y libres de riesgo.
- Integrar la API de Figma a una aplicación funcional para realizar pruebas alternativas de desarrollo donde los cambios UI/UX puedan coexistir con los procesos ágiles.
- Reducir el tiempo de espera entre una propuesta de cambio sobre la interacción/UI y el momento en que efectivamente se implementa.
- Realizar pruebas sobre la API de Figma integrada a una aplicación funcional para hacer una evaluación preliminar de nuestra hipótesis de reducción de tiempos de desarrollo y la posibilidad de adopción por parte de la industria.

1.3 Logros alcanzados

Para concretar los objetivos mencionados, nos propusimos:

- Analizar el contexto actual de trabajo de los diseñadores en relación al grupo de desarrolladores para proponer un proceso más ágil y productivo en la construcción y modificación de interfaces de usuario.

- Desarrollar una aplicación web funcional como contexto para realizar nuestra investigación.
- Crear un plugin que pueda conectar la aplicación web con la API de Figma para poder realizar transformaciones en la UI sin la necesidad de un desarrollador de software como intermediario entre la aplicación y el diseñador.
- Realizar una especificación tanto de las convenciones utilizadas para construir diseños anotados y comentados en Figma como de las que se deben utilizar a la hora de construir la aplicación que consuma esos diseños, evitando conceptos técnicos que puedan llegar a bloquear al diseñador.
- Realizar un experimento para obtener conclusiones en cuanto a mejoras en el tiempo de desarrollo.

1.4 Organización de la tesina

Capítulo 2
Se expone el marco teórico de esta tesina, se definen las diferentes metodologías utilizadas en el contexto moderno del desarrollo de aplicaciones web, se describe el marco de trabajo en el que se desenvuelve el diseñador, se introduce la herramienta Figma junto con su API y se detalla el proceso convencional de modificación de una interfaz de usuario.

Capítulo 3
Se detalla el enfoque de trabajo que se utilizará en base a la integración con la API de Figma, sus ventajas ante el enfoque de desarrollo convencional, y posibles aplicaciones por parte del diseñador para A-B testing y pruebas de concepto.

Capítulo 4
Se expone una guía de construcción de diseños auto-adaptativos junto con la especificación de la arquitectura del plugin a entregar.

Capítulo 5
Se presenta en detalle el experimento realizado dentro de un marco de trabajo ágil sobre una aplicación funcional integrada con la API de Figma. Se especifican los datos obtenidos y métricas utilizadas.

Capítulo 6
Se describen las conclusiones de esta tesina y futuros desarrollos.

CAPÍTULO 2

Trabajos relacionados y Marco teórico

2.1 Desarrollo de un sistema

2.1.1 Proceso de desarrollo

Cuando nos ponemos a pensar en cuánto utilizamos la informática en el día a día de nuestra vida cotidiana, empezamos a notar que la encontramos prácticamente en todo lo que hacemos. Nos despertamos con la alarma del celular, miramos cómo está el tiempo en alguna aplicación del clima, miramos las noticias en algún portal web, organizamos nuestras reuniones o eventos que tendremos en el día en nuestro calendario del celular, controlamos nuestra cuenta bancaria desde el homebanking. Podríamos seguir, pero teniendo en cuenta que acabamos de tomar sólo la primera media hora del día de una persona promedio, nos llevaría varias páginas enumerar todos las tareas relacionadas con la informática que tiene esta persona en todo el día.

Pero para que estos sistemas de software salgan al mercado y tengan el impacto que tienen primero deben pasar por un proceso extenso y exhaustivo de desarrollo. Durante este proceso las personas involucradas realizan diferentes tareas, cada una agregando un valor sustancial al producto, hasta finalizarlo.

No podemos afirmar que existe un proceso de desarrollo universal, sino que cada grupo de trabajo adoptará el proceso que más se adecue a sus necesidades dependiendo del tipo de software que se vaya a desarrollar, las dependencias que éste tenga con otros sistemas, la cantidad de usuarios a la que esté expuesta, la cantidad de personas que integren el equipo, la experiencia del equipo, su estructura, las habilidades individuales, el tiempo que se disponga para desarrollar, el tipo de cliente y el grado de comunicación que tengamos con éste, entre muchos otros factores.

2.1.2 Metodologías ágiles

El desarrollo de software ha pasado por diferentes modelos o enfoques, cada uno diferentes formas de interpretar la estructura con la cual se planifica y controla el procedimiento en la creación del sistema. Entre ellos podemos nombrar el modelo en cascada, el modelo en espiral, el modelo de prototipos, el modelo en V, entre otros [Sommerville11]. Cada uno posee sus ventajas y desventajas, pero en los últimos años hay un tipo de metodología que ha prevalecido y predomina como mejor opción actualmente a la hora de hacer un desarrollo.

Este grupo de metodologías son las que denominamos metodologías ágiles y, como su nombre lo indica, se caracterizan por ser capaces de incorporar cambios con rapidez en el desarrollo del software.

En la actualidad, debemos afrontar constantes cambios a medida que desarrollamos un sistema no sólo por parte del cliente, sino también en nuestro equipo de trabajo.

- Los requerimientos del sistema: durante la etapa de desarrollo puede ocurrir que se agreguen nuevos requerimientos, o que haya requerimientos que no sean necesarios, o quizás algunos requerimientos necesiten de alguna modificación que afecte su complejidad, tiempo de desarrollo, objetivo, etc.
- Las estimaciones de las tareas: el equipo de trabajo posiblemente estime en dificultad las tareas de desarrollo a realizar para medir tiempos y dividir el trabajo, pero las estimaciones no son más que eso, un valor aproximado. Por lo tanto, la dificultad que creíamos que un tarea tenía al principio puede que sea distinta a la real.
- Las fechas de entrega: es muy común que nuestro stakeholder necesite cambiar los plazos de entrega (por lo general, a un tiempo menor al estipulado en un principio) y el equipo de trabajo debe poder responder positivamente a esto.
- Las tecnologías que se utilizan: puede darse que el stakeholder decida cambiar alguna tecnología que se utilice ya que la competencia utiliza esta misma tecnología, o el mismo equipo de desarrollo note que la tecnología utilizada en un principio no satisface las necesidades del sistema o hay alguna tecnología que lo hace de una mejor manera.
- Los presupuestos: en el mundo de los negocios esto es muy común, nuestro cliente puede quedarse sin el presupuesto que se había acordado en un principio entonces necesitará un sistema de menor envergadura, o todo lo contrario, el presupuesto aumenta y necesita un sistema más complejo con un espectro mayor.
- El equipo de trabajo: no debemos olvidarnos que el equipo de trabajo son en definitiva personas, que pueden enfermarse, cambiar de trabajo, irse de vacaciones, y los plazos de entrega del sistema no tendrían por qué cambiar.

Es por esto que la forma en la que gestionamos el desarrollo de software debe ser ágil, donde los requisitos y las soluciones evolucionan para adaptarse a estos cambios.

Existen diferentes metodologías ágiles con diferentes conceptos y estrategias, pero podemos asegurar que cada una cumplen con el Manifiesto Ágil, un documento creado en 2001 por críticos de mejora del proceso de software. En este documento se plantearon un conjunto de técnicas y procesos como alternativa a las metodologías más rígidas que dependían de una fuerte normativa y planificaciones previas al desarrollo. El Manifiesto Ágil [ManifiestoAgil20] está compuesto por doce principios agrupados en cuatro valores fundamentales:

- Individuos e interacciones sobre procesos y herramientas.
- Software funcionando sobre documentación extensiva.

- Colaboración con el cliente sobre negociación contractual.
- Respuesta ante el cambio sobre seguir un plan.

2.1.3 Scrum

Dentro de las metodologías ágiles podemos nombrar Scrum, Extreme Programming (XP), Agile Inception, Design Sprint, Lean, entre otras. Este trabajo será realizado en el marco de las metodologías ágiles y en particular en la metodología Scrum.

A Scrum, lo podemos definir como un framework de manejo de proyectos liviano y ágil caracterizado por tener un enfoque iterativo e incremental. Iterativo ya que el ciclo de vida del proyecto será dividido en múltiples iteraciones llamadas Sprints, donde cada una equivale a un período de tiempo por lo general corto (entre 1 y 4 semanas por lo general), durante el cual, el equipo trabajará lograr sumar funcionalidad al sistema. Incremental porque al final de cada Sprint, el equipo habrá obtenido un incremento de software potencialmente entregable.

Los principales componentes de Scrum son:
- 3 roles: Scrum Master, Product Owner y Equipo de trabajo.
- 3 artefactos: Product Backlog, Sprint Backlog e incremento.
- 4 ceremonias: Sprint Planning, Daily Stand-Up, Sprint Review y Sprint Retrospective.

No es necesario explicar uno por uno cada uno de estos componentes pero sí debemos repasar el concepto de Product Backlog ya que aquí se encuentra uno de los puntos importantes a tratar en este trabajo. El **Product Backlog** es el componente donde encontraremos todas las tareas a desarrollar para completar nuestro sistema, y es un elemento al cual podemos agregar, modificar o quitar tareas durante el ciclo de vida de nuestro proyecto. Por lo tanto, todas las tareas relacionadas con cambios en la interfaz o usabilidad de nuestro sistema se encontrarán allí. Uno de los principales objetivos de este trabajo será decrementar la cantidad de tareas asignadas al desarrollador (al menos las relacionadas con cambios de UI/UX) que se encuentren en el Product Backlog, delegando esas tareas al diseñador y su plataforma de diseño.

2.1.4 Agile + Refactoring

Ahora bien, las metodologías ágiles exigen iteraciones que retroalimentan al producto final de forma rápida, se entrega una porción del producto final cada dos semanas aproximadamente y esto nos da la ventaja de poder integrar cambios rápidamente [Gaurav12]. Pero para lograr tener un “pipeline” de entregas lo suficientemente sofisticado y robusto para integrar funcionalidad constantemente, debemos adherirnos no sólo a la metodología ágil sino también a la construcción de software ágil.

Dentro de las metodologías ágiles podemos encontrar algunas buenas prácticas que contribuyen a la calidad, flexibilidad y sostenibilidad del software para lograr ir de la teoría a la práctica en estos ambientes cambiantes:

- Diseño incremental.
- Trabajo en equipo.
- Integración continua [Fowler06].
- *Refactoring de código.*

Dentro de todas estas prácticas la que quizás más se acerque en cuanto a interacción con nuestra estrategia de desarrollo es el refactoring de código.

El refactoring de código [Fowler18] es una práctica en la que se modifica la estructura interna del código de software sin alterar su comportamiento. En otras palabras, la refactorización se centra exclusivamente en mejorar el diseño y la calidad del código, llevándolo a un estado que permita al equipo de desarrollo trabajar más rápido y con mayor comodidad.

El software a medida que crece se vuelve más complejo, más difícil de entender, más difícil de mantener y más difícil de cambiar. Para luchar contra estas desventajas, el refactoring se encarga de aplicar una serie de pequeñas transformaciones para mejorar su calidad mientras se preserva el comportamiento, donde el efecto acumulativo de cada una de estas transformaciones es bastante significativo. Al hacerlo en pequeños pasos, se reduce el riesgo de introducir errores y evita que el sistema se rompa mientras se realiza la reestructuración, lo que permite refactorizar gradualmente un sistema durante un período prolongado de tiempo.

Creemos que aplicar refactoring lo más frecuente posible dentro del código, permite concretar todas las modificaciones que nuestro sistema nos demanda a medida que el código base crece y cambia, sin afectar negativamente en la legibilidad, mantenibilidad y extensibilidad.

Pero por qué afirmamos que esta práctica se puede aplicar a nuestro enfoque? La razón es que existen trabajos de investigación [Garrido, Grigera 11] que proponen el refactoring no sólo para mejorar el código base sino también la usabilidad de las aplicaciones. Por ejemplo lograr cambios positivos en la estructura de la navegación, mejorar la presentación de la información, y la aplicación de un mejor soporte al usuario a la hora de concretar tareas son algunos de los refactorings que se pueden lograr dentro de la usabilidad de un sistema.

Siguiendo esta línea de pensamiento, [Grigera17] publicó un trabajo en el cual se creó una herramienta llamada USF (Usability Smells Finder), la cual se encarga de detectar y diagnosticar “bad smells” [Fowler99] de usabilidad dentro de una aplicación y proveer posibles soluciones de refactoring.

A partir de estos dos artículos y líneas de pensamiento, se pensó en la posibilidad de poder aplicar estos refactorings de usabilidad mediante el uso de la API de Figma dentro de una

aplicación. Al encontrar posibles problemas de usabilidad dentro de los componentes visuales de la aplicación, el diseñador podría realizar los refactorings de usabilidad desde Figma. Sin embargo, este enfoque resultó muy ambicioso, ya que luego de analizar la API de Figma comprendimos que no está en un nivel de maduración en el cual se puedan enviar datos complejos de los componentes visuales que terminen resultando en un refactoring de usabilidad. Por ejemplo, no es viable ni posible en Figma:

- Cambiar un input text por un select.
- Cambiar un input text por un date picker.
- Agregar una validación de formulario.
- Agregar loaders entre pantallas.

Estos cambios requieren de mucha lógica de negocio para poder implementarse desde una plataforma donde el objetivo principal es dar estilo a componentes visuales. Y si aún así pudiésemos lograr pequeños cambios que puedan realizarse desde Figma como por ejemplo:

- Cambiar un texto por un link.
- Proveer un default option para un input o un select.
- Proveer una máscara para un input.
- Agregar un autocomplete a un input.

Estos cambios son sensibles a la lógica de negocio de la aplicación y se estaría delegando una responsabilidad y autoría al diseñador donde no debe tenerla.

Lo que sí resulta posible es realizar cambios de estilo de los componentes que mejoren la usabilidad y la experiencia del usuario (UX) desde el impacto visual de los mismos. Aunque este tipo de cambios parece más simple a priori, son el tipo de cambios que está dentro de las competencias del diseñador, y de hecho pueden llegar a provocar mayor comodidad, accesibilidad o placer para el usuario (características del la UX) que en definitiva impactará en la fidelización y el grado de conversión de usuarios tan buscado.

2.1.5 Agile + UCD

Dentro de los paradigmas modernos en el desarrollo de software que afectan positivamente las necesidades del usuario no sólo nos encontramos con el uso de metodologías ágiles. Si dejamos por un momento de lado el área de la programación y nos acercamos al área de diseño encontramos también allí paradigmas que tienen un impacto positivo en el usuario final, el más conocido de todos es el Diseño Centrado en el Usuario (UCD) [Garrett10]. Este proceso de diseño consiste en tomar al usuario como parte del proyecto durante todas las etapas del desarrollo, analizando y visualizando la forma en la que estos probablemente consuman el producto, haciendo tests y evaluaciones que se tomarán en consideración hasta el resultado final.

Sin lugar a dudas este enfoque de trabajo logra muy buenos resultados en cuanto a la usabilidad y longevidad del producto, ya que se realiza un uso crítico y temprano de la aplicación antes que esta llegue siquiera a una etapa beta. De esta forma se logra evitar futuros errores y pérdidas de tiempo destinado a corregirlos.

Aunque el enfoque de las metodologías ágiles y el de UCD apuntan a la satisfacción del usuario y cliente, las prácticas relacionadas con UCD requieren mucho más tiempo que las que se utilizan en un desarrollo ágil, y es muy complejo hacer coexistir ambos enfoques de forma eficiente.

Las métodos ágiles se encargan de entregar pequeños pedazos de software funcional a los clientes lo más rápido posible en breve iteraciones, mientras antes haya resultados, mejor. Por otro lado, UCD utiliza tiempo considerable en investigación y análisis antes de que comience un desarrollo de software, y como consecuencia, se retrasa el inicio de éste. Éste esfuerzo no sólo se utiliza antes del desarrollo, sino que se repite durante el mismo también.

El estudio de Da Silva et al. [DaSilva11] basado en una revisión sistemática entre UCD y Agile, provee una solución a esta disyuntiva con un enfoque de trabajo que combina prácticas UCD dentro de los agile sprints, pero no posee un respaldo empírico. A su vez se afirma que aunque existe un gran número de documentos promoviendo la integración de UCD y Agile, tampoco ninguno de ellos es validado con experimentos controlados.

Según Beux et al. [Beux18], existe un gran respaldo académico en que el uso de metodologías ágiles son las mejores opciones para implementar un sistema siguiendo UCD para lograr un buen UX, en particular Scrum y XP. Sin embargo, no hay suficientes estudios y pruebas que recomienden y respalden un marco de trabajo óptimo para integrar estos dos enfoques.

Un estudio desarrollado por Chamberlain et al. [Chamberlain06] donde se integraron prácticas de UCD con desarrollos ágiles se concluyó que la colaboración entre integrantes del equipo, especialmente la colaboración entre diseñadores y los desarrolladores es de alta importancia. Sin embargo, aún no se han desarrollado mecanismos y herramientas efectivos de comunicación entre ellos.

Por esto creemos que es de suma importancia poder incorporar a los proyectos que utilizan metodologías ágiles, buenas prácticas de UCD para lograr productos de buena calidad y que perduren en el tiempo. El enfoque que utilizaremos en nuestro trabajo de investigación se encargará de encontrar una interacción positiva entre desarrolladores y diseñadores, delegando tareas de los primeros en los segundos. En la práctica, la construcción de mockups por parte del diseñador es mucho más rápida que la implementación final del lado del desarrollador ya que el uso del lenguaje de programación es mucho más lento para generar el resultado que las interfaces de diseño.

Nuestro marco de trabajo intentará achicar la brecha de velocidad de trabajo entre el diseñador y el desarrollador, y hará más eficiente la comunicación entre desarrollador y

diseñador resolviendo tareas relacionadas con la interfaz, y por tanto en gran parte relacionadas con UCD y UX, del lado del diseñador.

2.2 El lado del diseñador

2.2.1 Conceptos de diseño

En un contexto donde la mayoría de las personas pasan gran parte de su día navegando por la web, contar con sitios de internet atractivos visualmente, capaces de retener a los visitantes y que sea intuitivo a la hora de usar resulta de suma importancia.

Por esta razón es que en el desarrollo de software, el diseñador UI/UX tiene un papel fundamental a la hora de concretar el producto al que se quiere llegar. Éste se encarga de obtener y evaluar, junto a los desarrolladores y project/product managers, cada requerimiento del usuario para con el sistema de forma de entender cuál es la mejor solución desde el punto de vista del diseño.

Por lo general, cuando se empieza a desarrollar un producto de software de cero, es habitual que los diseños no están hechos y se necesite del diseñador para realizarlos. En este punto el diseñador debe entender las necesidades del usuario final y comunicar eficientemente los posibles diseños del producto hasta llegar a una idea que cumpla las expectativas del cliente. Para llegar a esto no se avanza directamente y se itera sobre un diseño final, sino que se comienzan por diseños preliminares de bajo detalle donde a medida que se vayan procesando y afianzando, se irá avanzando sobre diseños más específicos y reales.

Los Wireframes, Prototipos y Mockups son herramientas que nos ayudan a comunicar y definir estos diseños dentro de cada etapa en el proceso de diseño.

2.2.2 Definición de conceptos Wireframe, Mockup y Prototipo

Existen varios términos a la hora de empezar a trabajar con el diseño de aplicaciones, y tener una clara noción de ellos antes de iniciar con cualquier desarrollo nos puede permitir ahorrar mucho tiempo y evitar confusiones.

Wireframe

Es una representación de baja fidelidad visual (lo-fi) [Mkrtchyan18] de un diseño, están representados en escala de grises sin dedicar demasiado tiempo al aspecto o estética del diseño. Permite comunicar la estructura de la solución del diseño final que estamos trabajando. Por ejemplo se pueden comunicar conceptos como:

1. Posición de los menús.
2. Presencia de logo.
3. Acciones principales.
4. Jerarquía de los elementos presentes.

Como se puede ver en la Figura 2.2.2.1 suele ser en blanco y negro para que el análisis se centre en la estructura y no en el contenido evitando así trivialidades. Puede que el contenido sean textos no definidos, lo principal es que se tenga una idea del espacio y jerarquía que van a ocupar. Es importante destacar que en esta etapa se dejan de lado aspectos secundarios como colores, tipo de contenido y diseño visual de la interfaz. No se debe invertir demasiado tiempo en ellos, ya que cuanto más rápidamente se defina, más rápidamente nos pondremos de acuerdo en la entrega final.

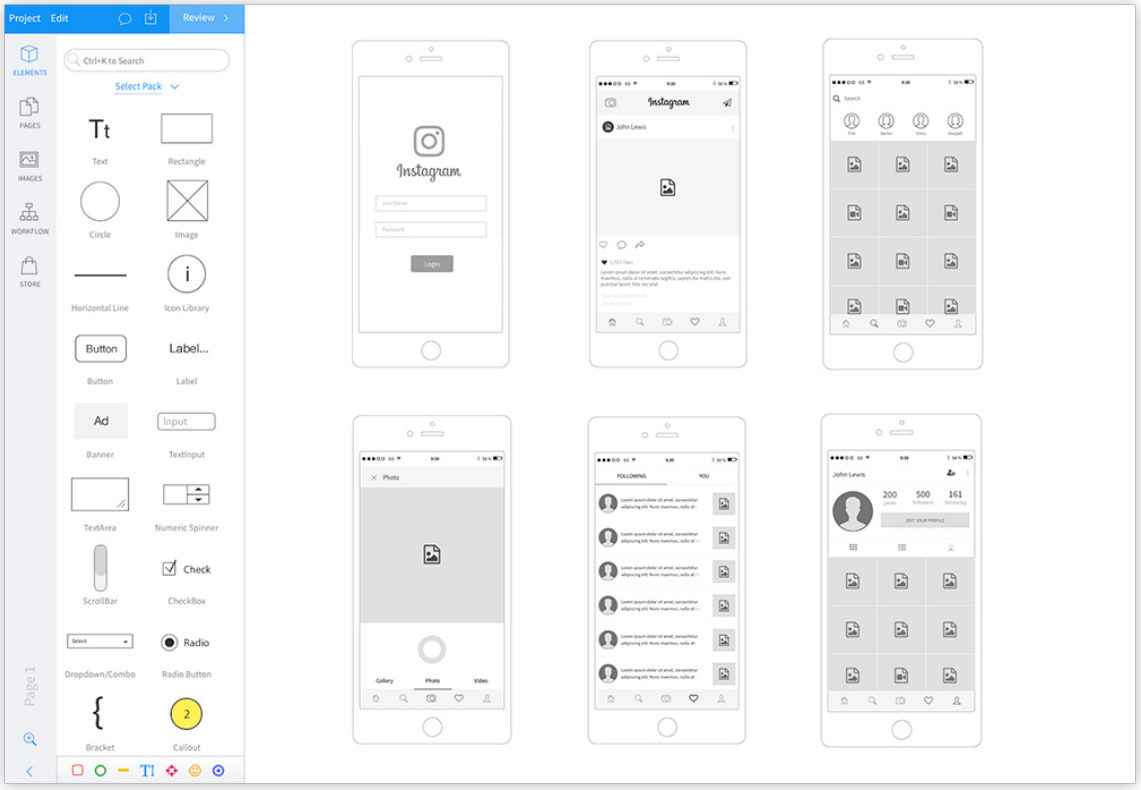


Figura 2.2.2.1: Conjunto de imágenes pertenecientes a un diseño de wireframe.

Esta etapa suele ser la primera para poder entender limitaciones básicas a nivel diseño y pensar alternativas en equipo. También sirven para documentación del proyecto, para entender decisiones previas que fueron tomadas a lo largo de una tarea.

Prototipo

Los prototipos son representaciones de media-alta fidelidad que incluyen o simulan la interacción con la interfaz [Mkrtchyan18]. En esta representación los usuarios ya sí podrán experimentar la experiencia de uso del producto. Aquí se define cómo se comporta el producto, por ello, aquí la interacción debe estar ya muy definida. Al tener una funcionalidad simulada y un aspecto visual más definido que el wireframe, son más adecuados para evaluar la experiencia que tienen los usuarios interactuando con el producto.

En esta etapa pueden utilizarse Mockups y Wireframes, dependiendo mucho la herramienta que elijamos. Algunos ejemplos pueden ser Invision, Flinto, Origami, Zeplin, etc. Cada programa puede dar diversas y más complejas formas de representar la interacción, queda a manos del diseñador la cual usar en base a tiempos y esfuerzo.

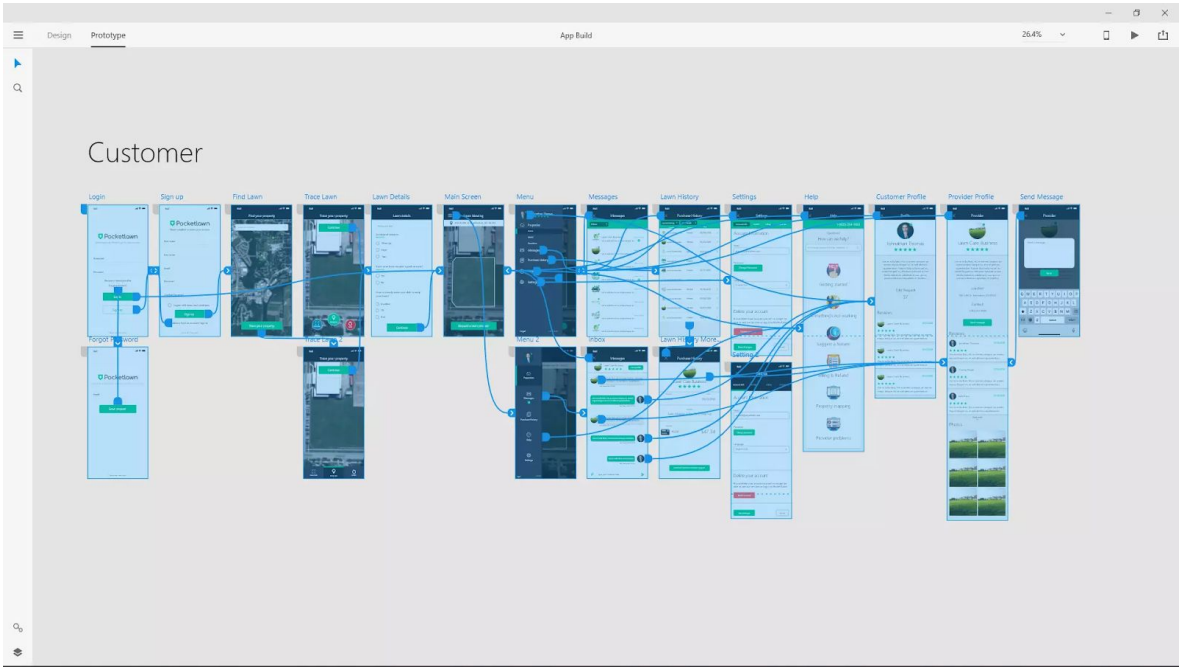


Figura 2.2.2.2: Conjunto de imágenes que representan la interacción entre páginas de un prototipo de aplicación móvil.

El objetivo principal en esta etapa es la de experimentar, testear y documentar todas las interacciones de nuestro diseño como podemos observar en la Figura 2.2.2.2 . De esta forma podemos validarlas, documentarlas y plantearlas para una posible solución final.

Mockup

Como se puede ver en la Figura 2.2.2.3, los mockups se caracterizan por tener una media-alta fidelidad y por ser representaciones completamente estáticas del diseño visual [Mkrtchyan18]. Por lo tanto, su objetivo es demostrar cómo se van a representar visualmente los elementos definidos, por ejemplo, en el wireframe.

Al contrario que en el wireframe, en el mockup ya se profundiza en el detalle, definiendo elementos visuales como el color, la tipografía, las sombras, etc. Si el wireframe asienta la base y estructura del diseño, el mockup definiría su apariencia.

Este producto permitirá discutir con el cliente si el aspecto visual y la comunicación encaja con lo que se buscaba. También puede ser muy útil para evaluar con los usuarios dicho aspecto visual y comunicación: si les parece atractivo, si les transmite o comunica aquello que se busca de forma clara o si se entiende.

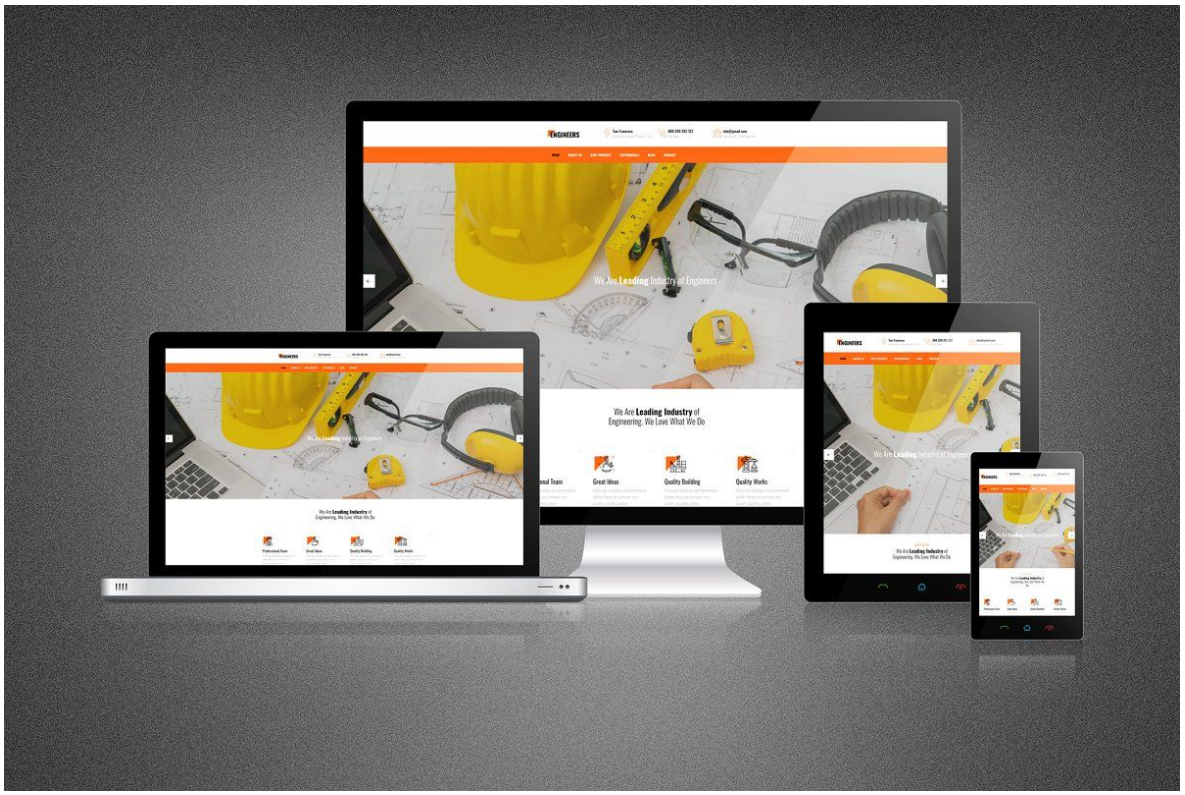


Figura 2.2.2.3: Conjunto de imágenes que representan el mockup de una aplicación web en diferentes dispositivos.

2.2.3 Impacto de integración de Figma con aplicación

La integración a Figma debe hacerse dentro de la etapa final del diseño, donde los cambios en el diseño tendrán impacto dentro de la aplicación funcional. Por lo tanto, la estrategia de desarrollo propuesta tendrá incidencia en la etapa de desarrollo de Mockups o Prototipos, siendo alguna de estas la utilizada por el diseñador como etapa de más alta fidelidad. No podemos afirmar que siempre el Mockup o el Prototipo será el diseño de más alta fidelidad ya que eso suele variar entre las convenciones que utilice cada diseñador.

Si bien la propuesta se centrará en agilizar los tiempos a la hora de impactar cambios, facilitar la comunicación entre el diseñador y desarrollador (diseño y aplicación) y otorgarle más responsabilidad al diseñador sobre las tareas relacionadas con los cambios de UX en vez del desarrollador, de ninguna manera esta estrategia eliminará tareas que ya esté haciendo el diseñador.

El diseño de wireframes, prototipos y mockups (entre otros) seguirán existiendo ya que son esenciales para lograr el objetivo del día a día del diseñador, y nuestra integración de Figma con la aplicación será un complemento en los diseños.

2.3 Herramientas de diseño

2.3.1 Comparación entre las herramientas más usadas

Dentro de las plataformas o herramientas de diseño más utilizadas por los diseñadores en la actualidad nos encontramos con Figma, Sketch, InVision y Adobe XD, entre otras. Cada una posee sus diferentes ventajas y desventajas, pero nos dispusimos a investigar si alguna poseía una API que nos permitiera construir nuestra aplicación.

Puntualmente encontramos que Figma, Sketch y Adobe XD poseen API, algunas con más documentación y maduración que otras, mientras que InVision no posee ninguna API. Por ejemplo, Sketch y Figma poseen una API madura con bastante documentación y una comunidad lo suficientemente grande como para atender las diferentes problemáticas que puedan aparecer con el uso de la misma. Adobe XD posee una API menos documentada que Sketch y Figma y la comunidad no es comparable en tamaño con las anteriores dos.

Estas APIs tienen el objetivo de permitir a desarrolladores y usuarios crear plugins para ampliar la plataforma en cuanto a:

- Actualización de tareas complejas.
- Lectura, escritura, modificación y generación de contenido en el documento.
- Integración de servicios externos.

- Agregado de controles UI nativos y personalizados para mejorar los flujos de trabajo creativos.

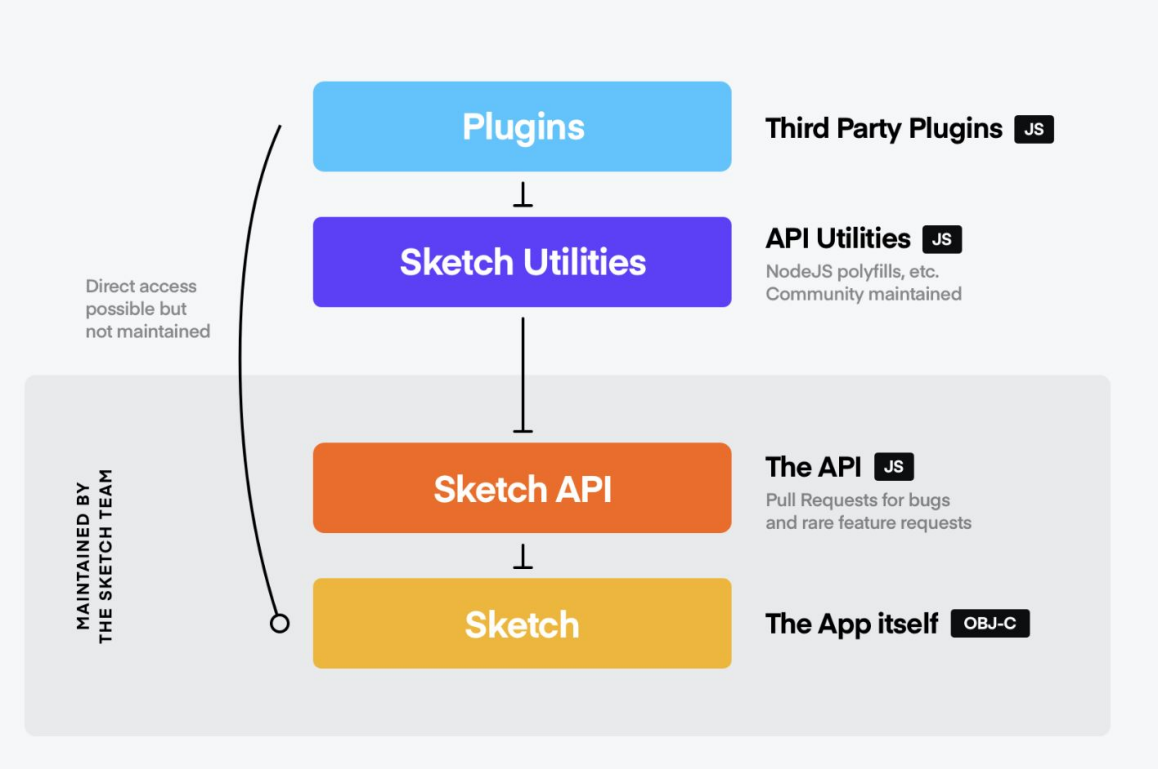


Figura 2.3.1.1: Diagrama de flujo para la creación de plugins para Sketch.

Aunque la documentación de la API de Sketch [SketchApi20] es la más completa de todas, el objetivo o su utilidad no concordaba con nuestro propósito, o al menos no existe información relacionada con el uso de su API en una aplicación web para el consumo de su información para la generación de contenido. La extensiva documentación para generar plugins, como podemos ver en la Figura 2.3.1.1 hace que esta plataforma sea la más completa en cuanto a la cantidad y calidad de plugins para beneficiar el trabajo del diseñador.

En cuanto a Adobe XD, encontramos documentación para los dos propósitos, para generar plugins [AdobeXDPluginApi20] y para desarrollar integraciones [AdobeXDContentApi20]. Aunque la información que había para construir plugins estaba bastante completa, la información que había para generar integraciones era muy escasa y casi inservible para lo que necesitamos nosotros. Había más información para obtener una api-key que para el uso de la API en sí misma, sólo encontramos 3 endpoints utilizables.

En cambio Figma [FigmaApi20] posee una muy buena documentación que hace hincapié, no sólo en el uso de la API para la ampliación de la plataforma, sino también en la

posibilidad de hacer uso de la API en sistemas externos para consumo de la información de los proyectos en Figma.

En cuanto a las características de cada plataforma respecto de su API podemos explayar las diferencias y similitudes en un cuadro comparativo:

	Figma	Sketch	Adobe XD
Autenticación	JWT / OAuth Token.	No.	JWT / OAuth Token.
API Key	Sí - Generada al instante.	La API viene incluida en el bundle de Sketch, por lo que no se requiere instalación y obtención de API Key.	Sí - Completar formulario y esperar aceptación.
Objetivo	Integración de diseños en la aplicación y construcción de plugins para extender la plataforma.	Construcción de plugins para extender la plataforma.	Construcción de plugins para extender la plataforma y escasa información acerca de integración de diseños en la aplicación.
Endpoints utilizables	Posee 17 endpoints utilizables para la integración de Figma con nuestra aplicación. Dentro de los más importantes tenemos endpoints para obtener: ● Archivos ● Nodos de archivos ● Imágenes ● Comentarios ● Usuarios. ● Proyectos.	No utiliza endpoints, la API está basada en Javascript y viene dentro del bundle de Sketch. Sólo se necesita hacer un *require('sketch')* dentro del código: `var sketch = require('sketch')` De esta forma podemos programar en Javascript nuevos plugins para la plataforma.	Posee 3 endpoints: ● OPTIONS api para permitir intercambio de información con la API. ● GET document. ● GET artboard.
Precio	[FigmaPricing20] Versión Gratis: Hasta 3 proyectos y almacenamiento ilimitado. Versión Paga: 12 USD por mes con más	[AdobeXDPricing20] 99 USD por primera compra y luego una licencia que se renueva anualmente por 79 USD. Para la suscripción por equipos hay un costo de 9 USD	[SketchPricing20] Versión Gratis con pocas funcionalidades y almacenamiento. Versión Paga: 10 USD si el uso es individual o 23 USD por mes por

	beneficios.	por persona extra.	persona para equipos.
Soporte de Plataforma	Basado en navegador. Aplicación de escritorio disponible para Mac y Windows sin soporte offline. Se puede usar en Linux desde navegador.	Soporta sistemas Mac únicamente.	Soporta sistemas Mac y Windows.
Soporte offline	No.	Sí.	Sí.

Con Sketch no seguimos avanzando en la investigación para añadirlo a la propuesta ya que principalmente su API está creada con otro objetivo, y además presenta otras dificultades a la hora de utilizarlo como son los precios de licencias y el soporte único para mac OS.

Y en cuanto a Adobe XD y Figma nos encontramos en este último con muchas ventajas sobre el primero. Figma presenta una documentación de API mucho más amplia y completa que se amolda a la funcionalidad que intentamos desarrollar y no hace tanto hincapié en el desarrollo de plugins que es lo que nosotros no necesitamos. En cambio, la documentación de Adobe XD para desarrollar plugins es completa mientras que la documentación para integrar archivos de la plataforma es muy poca.

La forma de obtener una API-Key para comenzar a hacer pruebas con Figma es instantáneo mientras que con Adobe XD se necesita completar un formulario y esperar una respuesta de aprobación. Además los documentos en la nube XD son privados o públicos, si se está creando una integración que requiere acceso a los documentos privados de la nube XD, la aplicación necesitará un token de acceso que se obtiene a través de una integración OAuth.

Por último, pensando en la simplicidad a la hora de posibles extensiones que se puede llegar a hacer de esta investigación, Figma posee 2 ventajas por sobre las otras plataformas. Primero, creemos que la funcionalidad que provee la versión gratis cubre las necesidades para realizar nuestra prueba de concepto. Y segundo, al ser una plataforma basada en el navegador, tenemos cubierto el soporte para mac OS, Windows y Linux, los 3 sistemas operativos más utilizados.

En conclusión, Figma es la plataforma de diseño que elegimos y creemos más útil para nuestra investigación.

2.3.2 Introducción a la herramienta Figma

Figma es una aplicación basada en navegador para diseñar UI y UX que cuenta con herramientas de diseño, creación de prototipos y generación de código. Actualmente es (posiblemente) la herramienta líder en la industria para diseñar interfaces y cuenta con características sólidas que respaldan a los equipos que trabajan en cada fase del proceso de diseño.

Figma está basado en el navegador, por lo tanto, no es necesario realizar ninguna descarga o instalación previa para su uso. Aunque si el usuario lo desea, hay versiones descargables para Windows y Mac OS.

Permite la colaboración en vivo y en tiempo real. Los miembros de un equipo pueden iniciar sesión a la vez y hacer cambios en algún diseño al mismo tiempo, y al estar todos los diseños guardados en línea todos estarán sincronizados con el proyecto. Los últimos cambios están siempre en el archivo, y no hay que transferir archivos entre miembros del equipo ni enviar archivos desde o hacia cualquier plataforma de almacenamiento de terceros.

Un detalle no menor a mencionar, es que a todos los archivos de un proyecto de Figma se les puede agregar comentarios directamente en el área de trabajo. Esta funcionalidad es de suma importancia para que miembros de un mismo equipo de trabajo puedan realizar colaboraciones en tiempo real, discutir los diseños actuales al mismo tiempo que los modifican. Sin ir más lejos el mismo cliente puede colaborar a la hora de tomar decisiones en los diseños, puede hacer sugerencias usando los comentarios y el diseñador puede implementar los cambios allí mismo.

Pero dentro de todas las funcionalidades que Figma pueda llegarnos a proveer desde el lado del diseño, la que más nos interesa para poder realizar este trabajo es la API que posee. La API de Figma permite el acceso, lectura e interacción con los archivos de la plataforma. Esto brinda la capacidad de ver y extraer cualquier objeto o capa, y sus propiedades, para poder representarlas fuera de Figma. Podemos presentar nuestros diseños o conectarlos a otras aplicaciones, pero nosotros le daremos una utilidad un poco más orientada al proceso de desarrollo de software.

La API de Figma está basada en una estructura REST, soporta autenticación vía tokens de acceso o OAuth2. Los pedidos se realizan por endpoints HTTP con funciones y códigos de respuesta claros, mediante estos endpoints se pueden solicitar archivos, imágenes, versiones de archivos, usuarios, comentarios, proyectos de equipo o archivos de proyectos.

Una vez conseguido acceso, se puede usar la API de Figma para inspeccionar la representación JSON del archivo. Cada capa y objeto en un archivo será representado dentro del archivo mediante un nodo. De esta forma tendremos acceso y podremos aislar el objeto y las propiedades asociadas a este.

2.4 Enfoque convencional de modificación de interfaz

Pensemos en el proceso que atraviesa una propuesta de cambio de interfaz de usuario hasta que se concreta y queda impactado en la aplicación dentro de nuestro ambiente productivo.

Cuando se dispone realizar un cambio en la interfaz de usuario debido a los resultados de un testeo de usabilidad o una solicitud del cliente, el diseñador debe trabajar en el cambio dentro de la plataforma de diseño que utilice hasta llegar a una versión final que se pueda delegar para su implementación.

Una vez concretado el diseño, el diseñador lo envía a un miembro del equipo de desarrollo. Éste deberá crear una tarea asociada al cambio solicitado siguiendo un formato especial [Cohn04] para escribirla, con el fin de poder asignarla a un desarrollador y darle seguimiento hasta su finalización.

Cuando la tarea está asociada a cambios visuales, se debe especificar el link al archivo de diseño actualizado. En otros casos no tan ideales podemos encontrarnos con que:

- Estos links no se proveen.
- Los archivos de diseño no están actualizados.
- Se adjuntan sólo imágenes relacionadas al cambio y no un link a la plataforma donde se encuentran detalles más específicos.
- No hay nada que indique al desarrollador el nuevo diseño asociado a la tarea.

Una vez escrita la tarea esta se ubicará en el Product Backlog para ser tomada por un desarrollador frontend en el próximo Sprint (si la tarea es urgente se puede ubicar en el Sprint Backlog para desarrollarse inmediatamente). Antes de comenzar la implementación por parte del desarrollador, la tarea debe pasar por el proceso de Sprint Planning [Schwaber02] donde se verificará, entre otras cosas, si la tarea es realizable o si necesita especificarse más información, y en base a esto se dispondrá de un plazo de entrega y un grado de dificultad generalmente asociado a un puntaje.

Debemos aclarar también, que por lo general, las tareas relacionadas con los cambios de diseño o usabilidad (salvo algunas excepciones) son las tareas menos priorizadas por los desarrolladores, ya que usualmente se opta por realizar primero las tareas que agregan funcionalidad al sistema; por lo tanto, esta tarea podría llevar más tiempo del necesario.

Una vez que la tarea es asignada a un desarrollador y estimada, se procede a realizar la implementación, creación del request del cambio al proyecto, revisión de código [Bacchelli13], testeo [Tian05] y despliegue en ambiente/s. Por lo general, en este proceso se pueden presentar ciertos problemas que pueden hacer que alguna de las etapas dure más de lo esperado o se deba retroceder a etapas anteriores:

- Construcción y modificación de tests del componente nuevo.
- Testeo manual por el desarrollador.
- Petición de cambio de código durante la revisión.
- Error en el testeo por QA.
- Error en el testeo por diseñador (el componente desarrollado no es idéntico al diseñado).
- Error en el testeo por usuario final (el componente desarrollado no es el que se esperaba o no cumple con lo solicitado).

Además puede ocurrir que el desarrollador no siga al pie de la letra el diseño del diseñador por falta de conocimiento del diseño, falta de conocimiento de la tecnología para lograr la implementación, distracción, error en la comunicación del cambio o por simplemente no encontrar una forma de realizar el diseño 100% idéntico al diseño y encontrar la solución más aproximada. Todas estas situaciones conllevan a que el desarrollo del componente no sólo lleve más tiempo del deseado, sino que el resultado no sea el esperado la mayoría de las veces.

Suponiendo el mejor de los casos donde no existe ningún error dentro del testeo del cambio, no se solicita un cambio en la revisión de código y los testeos por parte del equipo de QA y el diseñador pasan correctamente, la implementación se da como terminada y el cambio se materializa en el ambiente productivo. Aquí el diseñador puede verificar que la modificación se realizó correctamente y realizar nuevamente los testeos de usabilidad para verificar que la modificación tuvo el impacto deseado. Es decir, luego de todas estas actividades necesarias para implementar un cambio en la interfaz, quizás los resultados obtenidos no son los esperados y se debe repetir el mismo proceso hasta lograrlo.

Nuestra propuesta intentará presentar una alternativa a esta serie de pasos por una más simple y directa evitando muchos de los pasos anteriormente descritos y evitando la confusión en la comunicación de poca calidad entre las partes. Se le dará más protagonismo y herramientas al diseñador para resolver tareas UI a la vez que se le da más tiempo y libertad al desarrollador a la hora trabajar en otros aspectos de una aplicación.

CAPÍTULO 3

Integración con Figma

3.1 Integración de la Aplicación con Figma

En esta sección vamos a definir el contexto sobre el cual trabajaremos y la forma de trabajar que intentaremos aplicar.

El objetivo de este trabajo será encontrar una enfoque de trabajo en el cual podamos ahorrar tiempos a la hora de realizar y entregar tareas, y al mismo tiempo efectivizar el tiempo de trabajo de los integrantes de nuestro equipo. Creemos que delegando las tareas relacionadas con cambios de estilos e interfaz de usuario al diseñador y dejando todas las tareas relacionadas con la lógica de la aplicación al desarrollador, ayudará a ahorrar tiempos.

Por eso el objetivo inicial que tenemos es poder lograr que el diseñador pueda impactar en una aplicación productiva, cambios relacionados al diseño desde la plataforma de diseño Figma, sin necesitar del desarrollador de software para el trabajo.

3.1.1 Guía de estilo y Tema de la aplicación

Como ya mencionamos, la API de Figma permite conectar una aplicación con la plataforma de diseño, sólo necesitamos una estrategia para que el diseñador pueda definir y utilizar componentes dentro de Figma y que la aplicación los pueda leer y mostrar en la interfaz de usuario con poca o nula interacción de un desarrollador de software.

Esta estrategia se basa en dos conceptos importantes que necesitamos explicar primero:

- Guía de estilo (style guide).
- Tema (theme).

Una de las formas para asegurarse que el equipo esté en sintonía al momento de diseñar distintas partes de la aplicación, es crear una documentación de diseño o una guía de estilos de diseño. Es de mucha ayuda tener una guía de estilo para crear una experiencia consistente entre las diferentes páginas y además, ayuda a garantizar el futuro desarrollo o que la producción de terceros siga los mismos lineamientos de la marca y que se perciba como parte de esta.

Como podemos ver en la Figura 3.1.1.1, una guía de estilo es una colección de elementos prediseñados gráficos y reglas que diseñadores o desarrolladores web deben seguir para asegurarse que partes separadas del sitio web sean consistentes y creen una experiencia cohesiva al final.

Es extremadamente importante cuando varios diseñadores están trabajando juntos en aplicaciones web, que no existan errores de interpretación, subjetividades, o ajustes en los estilos basados en gustos personales. En el desarrollo, tener los elementos de la aplicación definidos hace más fácil la labor de los desarrolladores web al momento de utilizar estos elementos. Además, ellos sabrán qué elementos tienen que codificar y podrán ver exactamente lo que necesitan desde el comienzo.

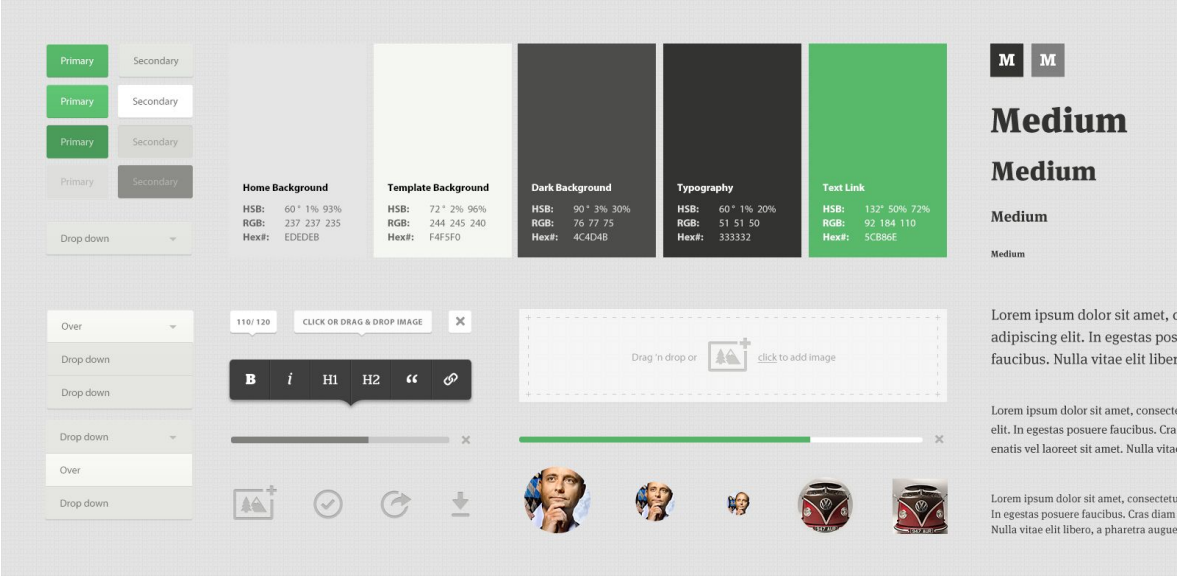


Figura 3.1.1.1: Kit-Ui para una aplicación web.

El concepto de “temas” está más relacionado a la parte técnica e implementación de la aplicación por parte del desarrollador. Los temas permiten separar los detalles de diseño de la estructura y comportamiento de la interfaz de una aplicación web. Así como los estilos son un conjunto de atributos que especifican apariencia de un componente, como lo son el color, tamaño de fuente, color de fondo, etc; el tema es un tipo de estilo que se aplica a toda la aplicación y no sólo a un componente.

Cuando se aplica un tema, cada vista de la aplicación tendrá acceso a los estilos que existan dentro él, y por lo tanto, se podrá reutilizar el mismo conjunto de estilos. Por consiguiente evitamos, desde el punto de vista de código de la aplicación, la repetición de archivos de estilos que especifiquen los mismos estilos para diferentes vistas. De esta forma podemos lograr que en la aplicación existan siempre los mismos tipos y estilos de botones, colores, tipografías, alertas, barras de progreso, íconos, inputs, etc. En esencia, el

tema de la aplicación es una representación directa de la guía de estilo dentro de la aplicación y tiene un impacto directo en la experiencia de los usuarios.

3.1.2 Acceso a los componentes de la interfaz

Podríamos intentar implementar una aplicación que pueda consumir información relacionada a su interfaz visual desde un proyecto de Figma utilizando la API que provee la plataforma, como bien describimos en la sección Introducción a la herramienta Figma. De esta forma tendríamos una aplicación con una estructura base que no obtenga los estilos de su interfaz mediante la compilación de sus archivos CSS en el código base, sino que los obtenga a partir de la información que tome de la API de Figma.

Y como la información que posee el proyecto Figma no es más que la representación de la interfaz de nuestra aplicación, podríamos obtener todos los datos relacionados a todos los componentes visuales llamando a la API. Por lo tanto, cuando el equipo de trabajo tenga que implementar un cambio de diseño, podría no tener que pasar por el proceso convencional de desarrollo para agregar un cambio en la interfaz.

Sólo se necesitaría conectar la aplicación a la API de Figma, el diseñador realizaría los cambios dentro de la plataforma de diseño, y la aplicación generará sus componentes visuales consumiendo la información de la API como podemos ver en la Figura 3.1.2.1.

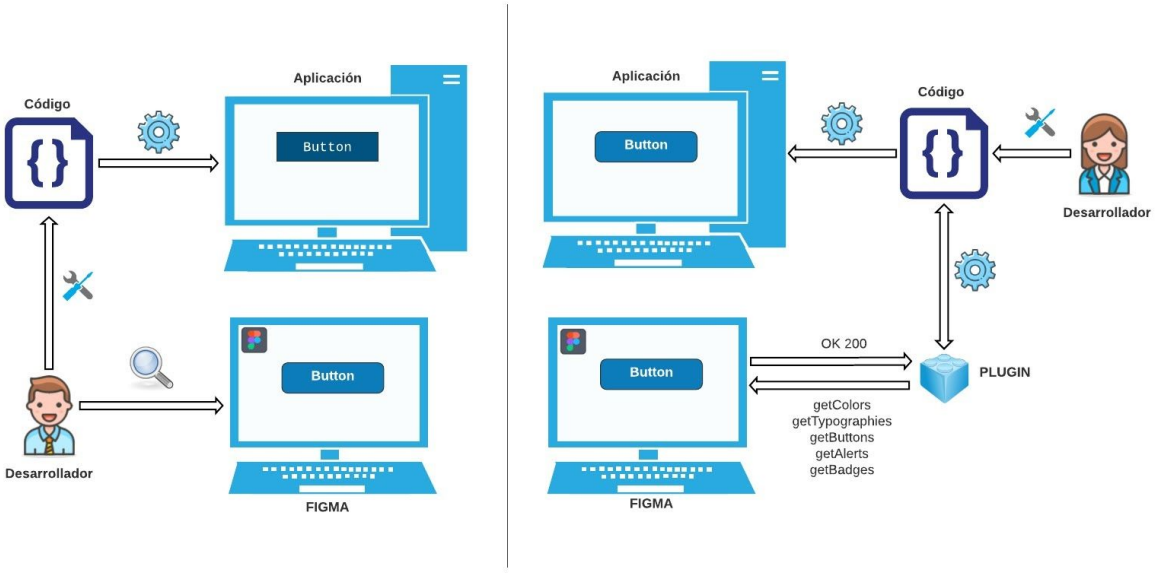


Figura 3.1.2.1: Comparación de enfoque convencional (izquierda) con enfoque utilizando la integración con Figma (derecha) para la creación de un elemento UI.

3.1.3 Enfoque de trabajo

Habiendo definido los conceptos de guía de estilos, tema y la forma de acceder a los componentes de la interfaz, nuestro enfoque será poder definir una guía de estilos estándar y reutilizable que se pueda usar en cualquier aplicación web y crear un tema dinámico que se pueda conectar a la plataforma Figma donde se encuentra esta guía de estilos.

La guía de estilos debe ser estándar para que no tenga que ser redefinida totalmente cada vez que se quiera crear una nueva aplicación. Para lograr esto, incluiremos aquí los componentes más básicos y usados por toda aplicación web (botones, tipografías, paleta de colores, imágenes, íconos, badges, alertas, etc). Estos componentes son los que luego serán referenciados por la aplicación, y como es de esperarse, la guía tiene que estar lo más completa posible. Así evitaríamos que el desarrollador agregue nuevos componentes visuales en vez de reutilizar y modificar los existentes.

El tema debe estar conectado a la guía de estilos mediante la API, para auto-modificarse si es que algún componente de la plataforma Figma cambia. Cuando el tema se conecte a la API, dentro de éste se creará una clase CSS por cada componente visual que exista en la guía de estilos del archivo de Figma. Esto permitirá al desarrollador de software utilizar todos los componentes definidos en Figma usando las clases que se definan en el tema de la aplicación.

Supongamos que tenemos la guía de estilos de nuestra aplicación en un archivo de un proyecto de Figma con los botones que se muestran en la Figura 3.1.3.1:

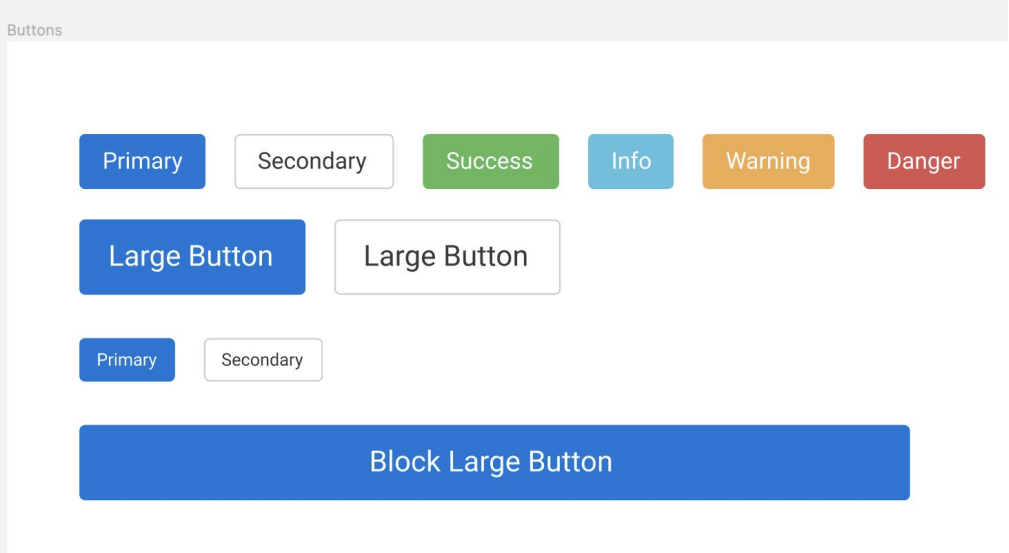


Figura 3.1.3.1: Botones dentro del Kit-Ui en un archivo de Figma.

Según lo explicado anteriormente, luego que el tema se haya conectado a Figma, el desarrollador dispondrá dentro del tema una clase CSS por cada botón dentro de la guía. Por ejemplo, si el desarrollador quiere incluir el botón “Primary” dentro de una de las páginas de la aplicación, tendrá que hacer uso de la clase que esté asociada al botón “Primary” dentro del tema.

Explicaremos más en detalle cómo el desarrollador hará uso de estas clases y cómo estas clases se crean y asocian a un componente de Figma en la sección Especificación de plugin a entregar. Este enfoque no sólo estará ligado a los botones, sino también a todos los componentes visuales que estén dentro de la integración. De esta forma lograremos que los componentes que existan en Figma, puedan ser incluídos de manera idéntica en la aplicación ya que los estilos CSS provienen del archivo de diseño. No existirán subjetividades, errores de comunicación ni falta de conocimiento para impactar el cambio. El componente se verá en la aplicación exactamente igual a como se ve en la plataforma de diseño.

Este enfoque no sólo aplica a la creación de componentes visuales sino también a su modificación una vez que estos fueron creados. En cuanto el diseñador haga algún cambio de estilos relacionado a un componente en particular en la plataforma de Figma, este cambio también se verá impactado en la misma aplicación. Recordemos que la aplicación estará conectada a la API de Figma, y si los estilos de un componente cambian, los datos que provengan de la API también cambiarán. Y como consecuencia de esto, los estilos asociados a la clase del componente dentro del tema, también cambiarán.

3.1.4 Comparación de estrategias de desarrollo

Tratemos de comparar los pasos que le lleva a un desarrollador para crear un nuevo botón de “Iniciar Sesión” dentro de la aplicación, utilizando ambos enfoques: el convencional y el propuesto por nosotros.

Enfoque convencional al agregar un componente nuevo

Si tenemos que crear un botón de “Iniciar Sesión” que sea idéntico al que se encuentra en los diseños, como desarrolladores debemos primero dirigirnos a la plataforma de diseño para conseguir todos los datos que conciernen al estilo del botón. De esta forma se evitan subjetividades y podemos confeccionar el botón tal cual lo diseñó el diseñador. Una vez que encontramos el botón dentro de los diseños recopilamos todos los datos que podamos relacionados a los estilos del botón:

- Alto y ancho.
- Color de fondo.
- Márgenes.
- Espaciado.

- Tipografía.
- Alineación de texto.
- Bordes.
- Comportamiento en hover.
- Comportamiento en click.

Además de estos datos, el desarrollador también deberá obtener o deducir datos del posicionamiento respecto de la pantalla, ya que no suelen aparecer directamente dentro del CSS del diseño.

Una vez encontrados todos estos datos debemos seguir con la línea de trabajo especificada en la sección Enfoque convencional de modificación de interfaz:

- Desarrollo de la tarea:
    - Desarrollo del código html.
    - Desarrollo del código css.
    - Implementación de los tests.
- Revisión de código (con posibles regresiones).
- Deploy del cambio a ambiente/s preproductivos.
- Testeo del cambio por equipo de QA y diseñador (con posibles regresiones).
- Deploy del cambio al ambiente productivo.
- Testeo del cambio por equipo de QA y diseñador (con posibles regresiones).
- Validación del usuario final (con posibles regresiones).

Enfoque con aplicación integrada a Figma al agregar un componente nuevo

El desarrollador trabajará en una aplicación que contendrá un plugin que conecta la aplicación con el archivo de Figma que posee los diseños. A su vez el plugin creará dentro de la aplicación una ruta a la cual sólo los desarrolladores pueden acceder. Esta ruta llevará a una página donde existirá la guía de utilización de los componentes visuales de la aplicación. Dentro de esta guía encontraremos, entre todos los componentes visuales de la aplicación, todos los botones existentes junto con el código que se debe escribir para poder utilizarlos.

Figura 3.1.4.1: Sección de botones de la guía de estilos proveída por el plugin a entregar.

El desarrollador sólo debe saber cuál botón debe utilizar de todos los presentes en la guía de estilos y utilizar el código asociado a éste como podemos en la Figura 3.1.4.1.. Una vez incluido el código, el botón se renderizará exactamente igual a como aparece diseñado en Figma.

A simple vista podemos notar algunas ventajas al utilizar esta estrategia de desarrollo. En primer lugar vemos una reducción en la cantidad de código que hay que escribir por parte del desarrollador, ya que evitamos escribir código relacionado a estilos CSS y HTML. Sólo se tiene que identificar el componente visual a incluir en la aplicación dentro de la guía de estilos, y copiar la línea de código asociada para hacer uso de éste.

En este caso, con el nuevo enfoque propuesto no podremos evitar las etapas de creación de request al proyecto, revisión de código y testeos porque el componente visual no existía en primer lugar, pero sí podemos afirmar que al haber menos código escrito y más certeza en los estilos de los componentes, estas etapas serán más cortas en tiempo.

Como segunda ventaja vemos que nos desligamos de las subjetividades y errores que pueda llegar tener el desarrollador en la implementación, el componente es exactamente igual al diseño que propuso el diseñador.

Como este ejemplo se da cuando el componente visual no existía previamente dentro de la aplicación, comparemos ahora el proceso de desarrollo cuando se debe realizar una modificación a un componente que ya existe en la aplicación.

Enfoque convencional al modificar un componente

El desarrollador debe dirigirse a la plataforma de Figma, identificar el componente visual a modificar, identificar los cambios en el diseño y aplicar los cambios necesarios en el código asociado a los estilos de la aplicación. Luego el proceso es exactamente igual al anterior con todas las etapas que requieren que se efectivice el cambio, con la existencia de posibles errores y retrocesos a etapas anteriores.

Enfoque con aplicación integrada a Figma al modificar un componente

El diseñador si quiere hacer algún cambio dentro del diseño de la aplicación, sólo tiene que hacer el cambio en el componente visual dentro de Figma, y el tema, al estar conectado a la plataforma, se actualizará con los nuevos cambios. Y al actualizarse el tema, la aplicación que hace uso de este, se terminará actualizando con solo actualizar la página dentro del browser.

Al haber descrito el segundo posible caso de uso de esta integración nos encontramos con la tercera ventaja de este enfoque. Al existir el componente y solo necesitar actualizar el diseño en Figma para que se actualice el tema, evitamos todas las etapas de desarrollo mencionadas en el enfoque convencional con el consecuente ahorro de costos de desarrollo, ya que no hay agregado de código nuevo sobre el que haya que hacer deploy ni testeos. Las únicas pruebas necesarias serían las pruebas de usuario, de las que nuevamente debería encargarse el diseñador. Más adelante hablaremos de cómo se pueden crear con nuestro enfoque una versión de la aplicación que el diseñador pueda usar para estas pruebas de usuario.

Si pudiésemos aplicar el mismo enfoque para la mayor cantidad de componentes visuales que tenga la interfaz de la aplicación, podríamos lograr incluir componentes visuales más rápido que con el enfoque convencional y las modificaciones de estos se lograrían casi instantáneamente.

Hasta aquí hemos descripto que con la integración propuesta entre Figma y la aplicación web podemos obtener importantes ventajas. Más allá de las mismas, podemos explorar qué otras cosas podemos lograr para que la integración sea más completa.

3.1.5 Aplicar estado a los componentes

Siguiendo nuestra idea principal de delegar tareas relacionadas al diseño de interfaz al diseñador para que el desarrollador pueda trabajar en tareas más funcionales, nos preguntamos qué otros aspectos de la aplicación relacionadas al diseño podemos sumar a la integración de Figma.

Dentro del diseño de interfaz de la aplicación no sólo nos encontramos con los estilos que presentan los componentes sino también con cómo estos se ven cuando interactúan con los usuarios. Por ejemplo, los botones pueden cambiar de color y tamaño cuando:

- El usuario hace click sobre ellos.
- El usuario pasa con el cursor por arriba de ellos.
- El botón se encuentra deshabilitado.
- El botón se encuentra enfocado.

Todos estos estados por los que puede pasar un botón, se pueden representar mediante diseños en Figma y, por lo tanto, también se podrían importar desde la plataforma. Como podemos ver en la Figura 3.1.5.1, cada botón (o componente visual) tendrá su diseño en su versión normal, cliqueada, deshabilitada, enfocada, pasando por arriba con el mouse, etc.

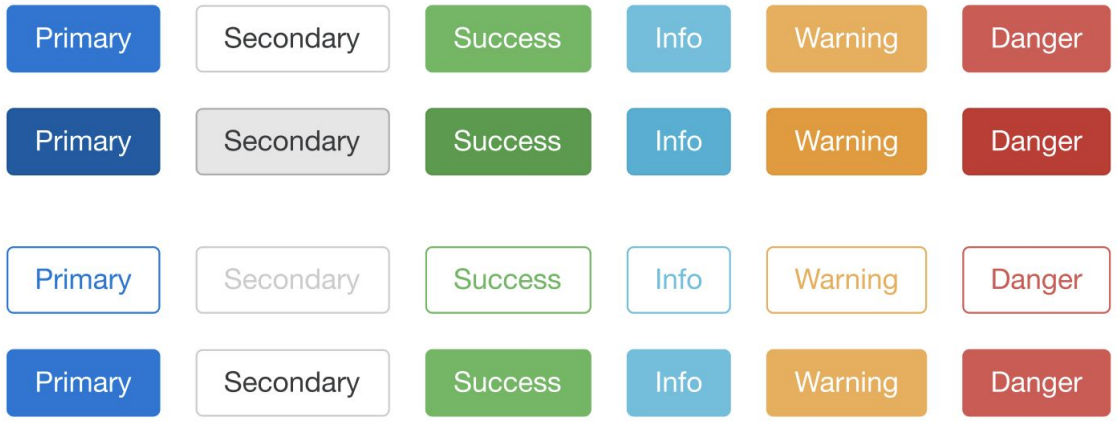


Figura 3.1.5.1: Sección de botones con todos sus estados representados dentro de un Kit-Ui hecho en Figma.

Todos estos estados por los que pasa un botón están representados en un diseño, por lo que tendremos múltiples diseños que representan al mismo botón en un estado en particular. En este caso, la integración propuesta se encargará de identificar cada botón, agrupar sus diferentes estados, procesarlos, y enviarlos para que el tema de la aplicación frontend lo pueda aplicar correctamente.

Esto se podría aplicar no sólo para los botones sino para todos los componentes visuales que presenten un cambio de estilo por cada estado que presente.

Dentro del enfoque inicial que tuvimos para con esta integración, nos propusimos investigar también sobre cómo usar esta integración para otros propósitos: el comportamiento que se podía aplicar a los componentes visuales, su diseño responsivo en diferentes resoluciones y tamaños de pantalla, el uso de comentarios que provee Figma para aplicar alguna de estas

integraciones u otras. Todas estas ramas de la investigación se encuentran en la sección Complejidades y Limitaciones.

3.1.6 Aplicar A-B testing

El concepto de A-B testing surge de la necesidad de aumentar y lograr constancia en el tráfico de usuarios dentro de una aplicación web. Dentro de sus objetivos, nos encontramos con uno de los más importantes que es mejorar la experiencia de usuario dentro de una aplicación. Las pruebas de A/B testing nos permiten ir perfeccionando el contenido que se le ofrece a los visitantes para que su experiencia de usuario sea cada vez más personalizada y acorde a sus expectativas y necesidades, buscando identificar los cambios que incrementan un resultado determinado (por ejemplo, la proporción de clics que recibe un banner publicitario, la proporción de ventas que genera un aviso promocional nuevo, la cantidad de suscripciones que genera un cambio en un componente visual).

La práctica consiste en comparar dos versiones de una misma aplicación web para comprobar cuál de las dos versiones es más eficiente [Kohavi07]. Estas variaciones, llamadas A y B, se muestran de forma aleatoria a los distintos usuarios de la aplicación web, una parte de ellos verá la versión A y la parte restante verá la versión B.

Una vez mostradas las distintas versiones a los usuarios, un análisis estadístico hace posible testear la efectividad de cada variación en base a distintos indicadores de rendimiento. En resumen, se puede comprobar qué versión genera más clics, suscripciones, ventas, etc. Los resultados determinarán la versión ganadora, es decir, la versión que genere más interacción y tráfico de usuarios y la que se debería utilizar de ahora en más.

Por ejemplo, en una página web de comercio electrónico, el proceso de compra es normalmente un buen candidato para realizar un test A/B, dado que, la variación de ciertas partes (disposición y cambio de contenido, cambio de tipografía, colores, componentes visuales, imágenes, etc) de la página que se esté mostrando puede generar más o menos ventas.

Dentro de esta sección ofreceremos una alternativa a la manera de aplicar A-B testing sobre una aplicación web que utilice la integración con Figma.

Dada la premisa de una prueba de A-B testing, se requieren dos versiones de una aplicación web para medir los resultados que genera una u otra en base a la interacción con el usuario. Supongamos que una versión presentará una versión de estilos para sus componentes visuales, y la otra versión utilizará otra versión de estilos con el fin de realizar la comparación.

A su vez necesitamos un poco de trabajo en manejo de manejo de servidores para poder persistir estas dos versiones de la misma aplicación en un mismo servidor, pero a su vez, se debe programar un balanceo de carga para enviar la mitad del tráfico de usuarios a una

versión, y la otra mitad a la otra versión. No es el objetivo de esta sección, explicar cómo se realiza esta configuración.

Pero cómo logramos que una aplicación integrada con Figma posea dos versiones diferentes de diseño visual ? La respuesta está en cómo se conecta la aplicación con un archivo de Figma.

Dentro de las variables de entorno de la aplicación web, se debe especificar el ID del archivo de Figma del cual se obtiene la información de los estilos. Por lo tanto, si una aplicación se despliega con el ID de un archivo de Figma y la otra aplicación se despliega con otro ID diferente, ambas versiones consumirán estilos diferentes también.

En conclusión, el diseñador podría trabajar en dos diseños diferentes sobre la plataforma Figma y luego enviar los IDs de cada archivo al equipo de desarrollo. Luego no queda más que hacer un despliegue de una versión con un ID y hacer el despliegue de la otra versión con el otro ID.

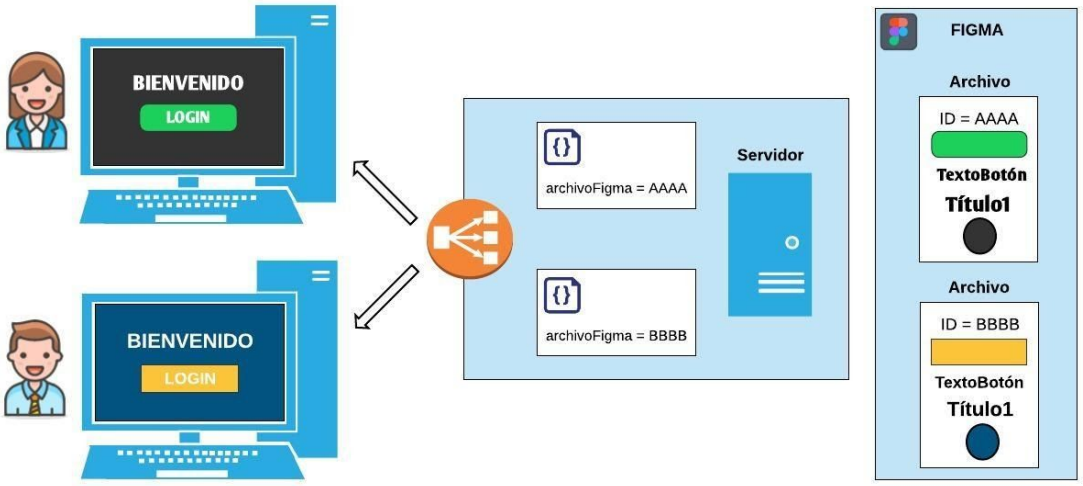


Figura 3.1.6.1: Representación de aplicación sometida a A-B testing utilizando la integración con Figma.

En este nuevo enfoque podemos notar que el código que se despliega de la aplicación es exactamente el mismo, lo único que varía es el ID de archivo dentro de las variables de entorno con las cuales se despliega.

Como vemos en el ejemplo de la Figura 3.1.6.1, el diseñador podría encargarse por sí solo de establecer un contexto real para realizar pruebas de A-B testing con el fin de obtener datos sobre usabilidad. Lo único que necesitaría es tener dos copias del mismo kit UI en Figma y realizar las modificaciones pertinentes en uno de los archivos.

Aquí tenemos una gran ventaja ya que el diseñador logra independencia del equipo de desarrollo y los tiempos para configurar el contexto sobre el cual se harán las pruebas de A-B testing son mucho menores ya que se omite tiempo destinado al proceso de desarrollo (configuración de las distintas versiones a someter las pruebas).

El diseñador se encargará de establecer el contexto, pero para que la prueba de A-B testing funcione también se necesita de un desarrollador encargado de hacer los respectivos despliegues al entorno de producción y setear la configuración del balanceador de carga. Luego la forma de medir los datos que se procesen en cada instancia quedará del lado del equipo de desarrollo.

Este enfoque está destinado a realizar pruebas de A-B testing que se relacionen con cambios de identidad visual o modificaciones de estilo de componentes visuales que ya estén dentro de la aplicación. No sería lo mismo si se quiere hacer un testeo con cambios más complejos como:

- Cambio en la disposición de información entre versiones.
- Mostrar cierto componente en una versión y en otra no.
- Cambio de flujos de navegación.
- Cambio de contenido de texto entre versiones.

En estos casos se podría seguir utilizando la integración de Figma pero las ventajas estarían relacionadas a la velocidad con la que los cambios llegan al ambiente de producción y no con la independencia del diseñador para setear ambientes de pruebas de A-B testing. A su vez, la forma de establecer dos versiones distintas de la misma aplicación sería de la forma convencional: se deben hacer dos despliegues de dos versiones distintas de la aplicación (historia, commits y código distinto).

3.1.7 Aplicar pruebas de concepto de diseño

Siguiendo la línea del enfoque anterior, el diseñador también podría tener preparadas para su uso, algunas instancias de las aplicaciones con las que esté trabajando desplegadas en distintos ambientes de prueba (sandbox). Cada una de estas instancias estaría ligada a archivos de Figma específicos.

Por lo tanto, si el diseñador quisiera hacer diferentes pruebas de concepto sobre alguna de las aplicaciones que tenga en su sandbox lo único que necesitaría es hacer el respectivo cambio de diseño dentro del archivo de Figma correspondiente.

Supongamos que el dueño de alguna de las aplicaciones quiere hacer un rebranding (cambio de marca o rediseño de identidad), el diseñador podría mostrar en el momento cómo quedaría la aplicación luego de aplicar los cambios que sean solicitados directamente en el archivo de Figma. Esto es de gran utilidad para poder ver con rapidez cómo quedará la aplicación luego de hacer los cambios. Hasta se podrían hacer los cambios dentro de Figma y

verlos impactados en la aplicación en tiempo real mientras el cliente los está solicitando.

Es común que se haga un rebranding de toda una aplicación que puede llevar días o semanas hasta obtener el resultado final utilizando el enfoque convencional de desarrollo, pero a la hora de ver los cambios aplicados en todos los componentes y pantallas y recibir una devolución del cliente, nos encontremos con alguna inconsistencia o error que no se había detectado previamente dentro de la etapa de diseño. No sólo pueden hacerse pruebas relacionadas a rebranding, sino a cualquier tipo de cambio que el diseñador quiera probar y no depender de un desarrollador para ver cómo quedará el resultado en la aplicación real.

Creemos que es de suma importancia poder dar esta independencia al diseñador para poder hacer pruebas de diseño y ver los resultados en tiempo real sobre aplicaciones reales ya que no sólo se logra independencia sino que se ahorra tiempo de desarrollo (en todos sus procesos) para lograr el cambio requerido.

3.2 Complejidades y Limitaciones

3.2.1 Aplicar comportamiento a los componentes

Si nos ponemos a pensar en componentes visuales como inputs, selects, checkboxes, radio buttons, etc; estos suelen tener algún tipo de comportamiento para ayudar al usuario a interactuar con ellos, por ejemplo:

- Presentan placeholders para especificar qué tipo de información debe ingresarse.
- Presentan algún tipo de máscara para limitar y ayudar al usuario a ingresar la información correctamente.
- Presentan íconos con mensajes de ayuda para dar más información al usuario.
- Poseen mensajes de error o éxito dependiendo lo que ingrese el usuario.
- Cambian de color en base a si lo que ingresó el usuario es correcto o no.

En un principio, se pensó en la idea de poder representar este tipo de información en Figma para que sea importada por la aplicación. Consultando esta problemática con una diseñadora profesional nos encontramos con diferentes limitaciones para hacer esto.

La información que está asociada a estos comportamientos, se presentan para el caso de un elemento, en un contexto y en un tiempo específico. No todos los inputs van a tener los mismos placeholders ni máscaras, no todos presentarán el mismo mensaje de éxito o error en caso de completarse o no satisfactoriamente, no siempre la información que se muestre será la misma si el usuario ha iniciado sesión o no, etc.

Es decir, no todos los componentes visuales presentan la misma información asociada en todos los contextos de la aplicación. Por lo tanto, no es factible para un diseñador encargarse de diseñar todos estos comportamientos para cada uno de los componentes

visuales con el fin de que éstos se apliquen dentro de la aplicación automáticamente mediante la integración propuesta.

A su vez, suponiendo el caso que el diseñador pueda especificar todos estos comportamientos dentro del diseño, existe una información de negocio asociada que el diseñador no puede ni debe manejar. El diseñador, si posee conocimientos de UX, puede identificar problemas y sugerir soluciones, pero no tiene las herramientas ni datos necesarios para impactar los cambios. Esto ya está ligado al trabajo que debe realizar el desarrollador de software.

3.2.2 Aplicar cambios relacionados al diseño responsivo

Dentro de los conceptos de diseño dentro del desarrollo de software, existe una filosofía de trabajo que ha tomado mucha popularidad en los últimos años debido al creciente uso de diferentes tipos de dispositivos móviles para navegar la Internet. Esta filosofía llamada diseño responsivo [Marcotte17] o adaptativo tiene el objetivo de adaptar las aplicaciones web al dispositivo que se esté usando para visitarlas.

Cada uno de estos dispositivos posee una diferente resolución de pantalla, lo que genera la necesidad de adaptar la experiencia de navegación. El diseño responsivo se encarga de satisfacer estas necesidades, mediante el uso de diferentes versiones de interfaz de usuario, utilizando los mismos archivos HTML y CSS, para cubrir todas las resoluciones de pantalla posibles.

En un principio se pensó en poder importar los diferentes diseños de Figma de un componente visual asociados a las diferentes resoluciones de pantalla. Es decir, si un botón se ve diferente dependiendo de si se está usando un celular, una tablet o un monitor para navegar la aplicación, entonces en Figma podría existir un diseño diferente del botón para cada resolución. Luego la aplicación importaría todos esos diseños y mostraría el correspondiente dependiendo el dispositivo que se esté usando.

Esta idea es aplicable a la integración pero descubrimos que no es de mucha utilidad por las siguientes razones. Consultamos con una diseñadora profesional las ventajas de incluir esta funcionalidad y nos encontramos con que son casi nulas, ya que los diseños responsivos no suelen variar en cuanto al diseño visual de los componentes, sino en cuanto a la disposición y cantidad de información que se muestra como podemos ver en la Figura 3.2.2.1.

Figura 3.2.2.1: Ejemplo de redistribución de información en un diseño responsive para una aplicación web.

Para aplicar cambios relacionados al diseño responsivo de la aplicación se debe trabajar con conceptos relacionados con la disposición de los elementos y el uso de grillas para presentar la información. Si bien es posible lograr una integración con fin responsivo sólo para los estilos visuales de los componentes, no se encuentra dentro del marco de trabajo que nos propusimos y tampoco sería de mucha utilidad. No obstante en la sección Trabajos futuros se explica cuáles serían los pasos a seguir para lograr la extensión para integrar diseños responsivos para componente visuales.

3.2.3 Uso de comentarios

Cabe destacar que no sólo podemos acceder a los archivos y capas mediante la API, sino que también podemos acceder y modificar mediante métodos GET y POST los comentarios relacionados con estos archivos y sus elementos. Esta capacidad creímos que nos sería de suma importancia para nuestra prueba de concepto ya que los comentarios no son más que información asociada a un elemento del diseño.

Dentro de esta información, el diseñador puede incluir no solo sugerencias o anotaciones relacionadas al propio trabajo, sino que también se podrían incluir instrucciones de comportamiento del elemento. Si el elemento es un input, por ejemplo, podríamos asociar a éste sus placeholders, máscaras, mensajes de error, restricciones, etc, dentro de los comentarios. Si el componente posee diseños responsivos (diferentes diseños para cada

pantalla de dispositivo), se podría especificar cuál utilizar en cada caso. Si el componente posee diferentes diseños por cada uno de sus estados esta información también podría especificarse aquí.

Como toda esta información puede ser consumida mediante la API, al igual que la información que se puede consumir relacionada a la parte “estética” de la aplicación, el diseñador podría asociar información valiosa relacionada a un elemento dentro de sus comentarios. Luego la aplicación podría consumir esta información de la API y aplicar el comportamiento, diseño responsivo o estado mediante la integración.

Como ya aclaramos anteriormente, la idea de poder aplicar comportamiento relacionado con información de negocio desde una plataforma de diseño y por un diseñador, no tuvo buen puerto por algunas restricciones o limitaciones. Aclaramos que el diseñador no debe ni puede manejar información o datos del negocio sobre componentes visuales, ya que estos se definen dentro de un contexto, momento y elemento en particular. Tampoco es algo que se encuentre dentro del marco de trabajo de un diseñador, no tiene las facultades o herramientas necesarias para realizar tal trabajo.

Por otro lado, también se pensó en la idea de poder especificar instrucciones relacionadas a comportamiento del componente en diferentes resoluciones de pantalla, una forma de aplicar diseños responsivos desde Figma. Pero luego de investigar en profundidad y entender que el diseño responsivo está basado en cambios de posicionamiento y cantidad de información, entendimos que los datos provenientes de Figma no iban a servir para incursionar en esa estrategia de trabajo. Por lo tanto, los comentarios tampoco nos iban a ser de utilidad en ese aspecto.

Otra alternativa posible era poder enviar información de los diseños relacionados a los estados de los componentes visuales dentro de los comentarios. Por ejemplo, si existe un componente visual como un texto, botón o alerta; era interesante la idea de poder enviar en los comentarios información de los estilos del componente cuando este está deshabilitado, seleccionado, en un contexto de error, etc.

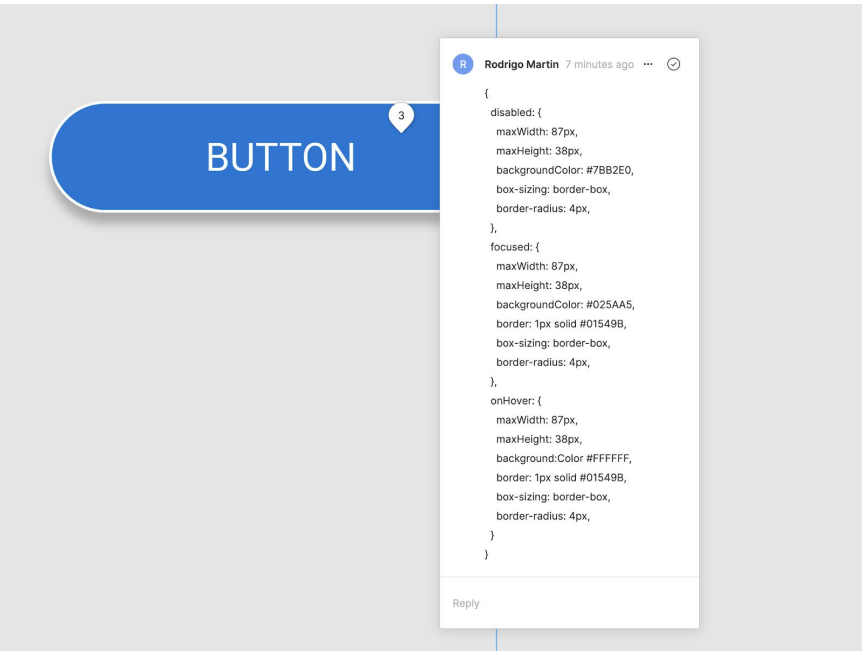


Figura 3.2.3.1: Ejemplo de especificación de estilos en diferentes estados visuales para un botón dentro de Figma.

Como podemos ver en la Figura 3.2.3.1, cada componente visual podría tener su comentario asociado en donde se especifique qué estilo CSS utilizar en cada estado que exista. Estos comentarios luego se podrían acceder desde la aplicación mediante un llamado a la API de Figma y transformado a código CSS (luego de un procesamiento adecuado) para ser utilizado por el tema. Esto presenta algunas problemáticas para el diseñador, ya que este será el encargado de crear estos comentarios para cada componente visual.

Primero y principal, se debe presuponer que el diseñador tiene conocimientos de CSS como para poder realizar este tipo de comentarios, lo cual puede no ser verdad. Aún así, la mayoría de las plataformas de diseño poseen un apartado CSS para cada componente que permite obtener este tipo de datos. Si el diseñador tuviese un diseño para cada estado de cada componente podría obtener esta información de su apartado CSS. Llegado a este punto, este tipo de integración generaría más problemas de los que resuelve ya que el proceso es muy tedioso y llevadero.

En segundo lugar, esta estructura de datos (que quizás pueda utilizarse de forma más amigable al diseñador y luego procesarse para que se obtenga algo similar a un JSON) es muy propensa a errores, ya que si el diseñador comete algún error en la sintaxis de la estructura, puede que la aplicación no la procese correctamente. A su vez, se debe educar al diseñador para que empiece a utilizar estas convenciones y comprenda las limitaciones y reglas que estas tienen.

En tercer lugar, y quizás entrando en un lugar un poco más técnico, nos encontramos con algunas limitaciones en cuanto a cómo se obtiene esta información desde la plataforma Figma. Para obtener los comentarios en Figma, se debe utilizar un endpoint diferente del que se utiliza para obtener la información relacionada con los estilos visuales. Esto significa que nuestra aplicación deberá hacer dos llamados (en realidad son más pero por simplicidad lo dejaremos en dos), uno para obtener los estilos de los componentes y otro para obtener la información relacionada a los estados de todos los componentes. Esta información no viene preparada para su consumo, debe ser procesada y organizada de forma que la aplicación los pueda utilizar y eso lleva tiempo de procesamiento. Por cada elemento visual se debe procesar la información de los estilos y luego buscar dentro de todos los comentarios cuál es el que está asociado a los estilos de sus estados. En archivos de diseño donde quizás existan decenas de componentes y cada uno con estados diferentes almacenados en sus comentarios, provocaría que este enfoque resulte en pérdida de performance para el sistema.

Igualmente, nos encontramos con una buena alternativa a estas problemáticas. Dentro del trabajo del diseñador, es muy común que existan diseños impactados en las plataformas de diseño no sólo para cada componente sino también para sus estados. Como podemos ver en la Figura 3.2.3.2, si existe un botón dentro de los diseños, por lo general, también existen diseños para ese botón en su estado deshabilitado, enfocado, cliqueado, etc.

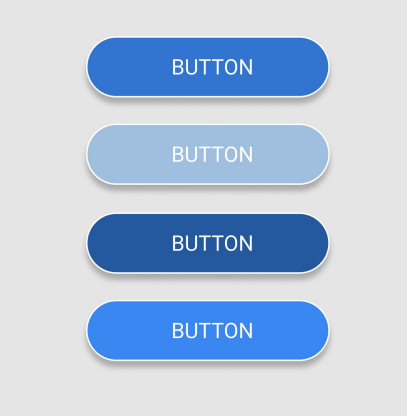


Figura 3.2.3.2: Estados de un botón representados en un diseño de Figma.

Entonces, desde el lado del diseñador se haría un diseño por cada estado del componente, en este caso un botón, y estos diseños se obtendrían llamando a un único endpoint, no necesitamos más utilizar el endpoint de comentarios. Luego la aplicación procesará e identificará a cada “grupo” de botones como un sólo botón. Este botón tendrá estilos CSS asociados para cada uno de sus con diferentes estados (normal, deshabilitado, cliqueado, enfocado, etc), y enviará los datos CSS al tema de la aplicación para que puedan ser procesados. Esto lo explicaremos en detalle más adelante en la sección Especificación de convenciones para construir diseños importables.

De esta forma, el diseñador no altera su manera de trabajar, ya que no le solicitaremos que maneje estructuras de datos ni comentarios por cada componente que tenga estados visuales diferentes. En todo caso, si el diseñador no suele trabajar de esta forma (lo cual es un error según los diseñadores que consultamos), empezar a realizar diseños para los estados de los componentes es una buena práctica, está dentro de su marco de trabajo y no genera mucho esfuerzo hacerlo.

En conclusión, los comentarios no fueron la mejor idea para alojar información relacionada al comportamiento, diseño responsivo o estado de los componentes, pero encontramos una buena alternativa para importar estilos CSS relacionados a los estados visuales utilizando un diseño por cada estado que posea cada componente.

CAPÍTULO 4

Guía de construcción de diseños auto-adaptativos

4.1 Especificación de convenciones para construir diseños importables

A la hora de diseñar los componentes en Figma, definimos una serie de pautas que debe seguir el diseñador para que la integración funcione correctamente. El salirse del marco de trabajo que definen estas normas, probablemente provoque la falta de impacto de los diseños en los componentes visuales de la aplicación.

Estas normas darán una referencia a los desarrolladores y diseñadores del alcance de la importación que posee cada componente, es decir, qué y cuántos estilos o especificaciones se pueden importar desde Figma por cada componente. En esta sección especificaremos estas normas a seguir por cada componente que se incluya dentro de la integración.

4.1.1 Botones

Figura 4.1.1.1: Sección de botones en Kit-Ui hecho en Figma para integración de este trabajo.

Como se muestra en la Figura 4.1.1.1, dentro de la sección de botones encontraremos:

- En la primera fila encontramos a los **botones en su estado normal**.
- En la segunda fila se muestra el diseño de estos botones cuando pasan a tener un estado **focus/hover.**
- En la tercera fila están los botones en su estado **desactivado.**
- Y las filas restantes son otros tipos de botones como **con borde**, **medianos**, **pequeños** y **en bloque.**

A la hora de poder extender estos botones o bien actualizarlos, el diseñador puede modificar una lista específica de atributos de cada uno. Y en el caso de querer agregar componentes nuevos, deberá hacerlo siguiendo un proceso específico.

Actualizar un botón

Este proceso es sencillo y confiable siempre y cuando se modifique la propiedad en el elemento correcto en Figma.

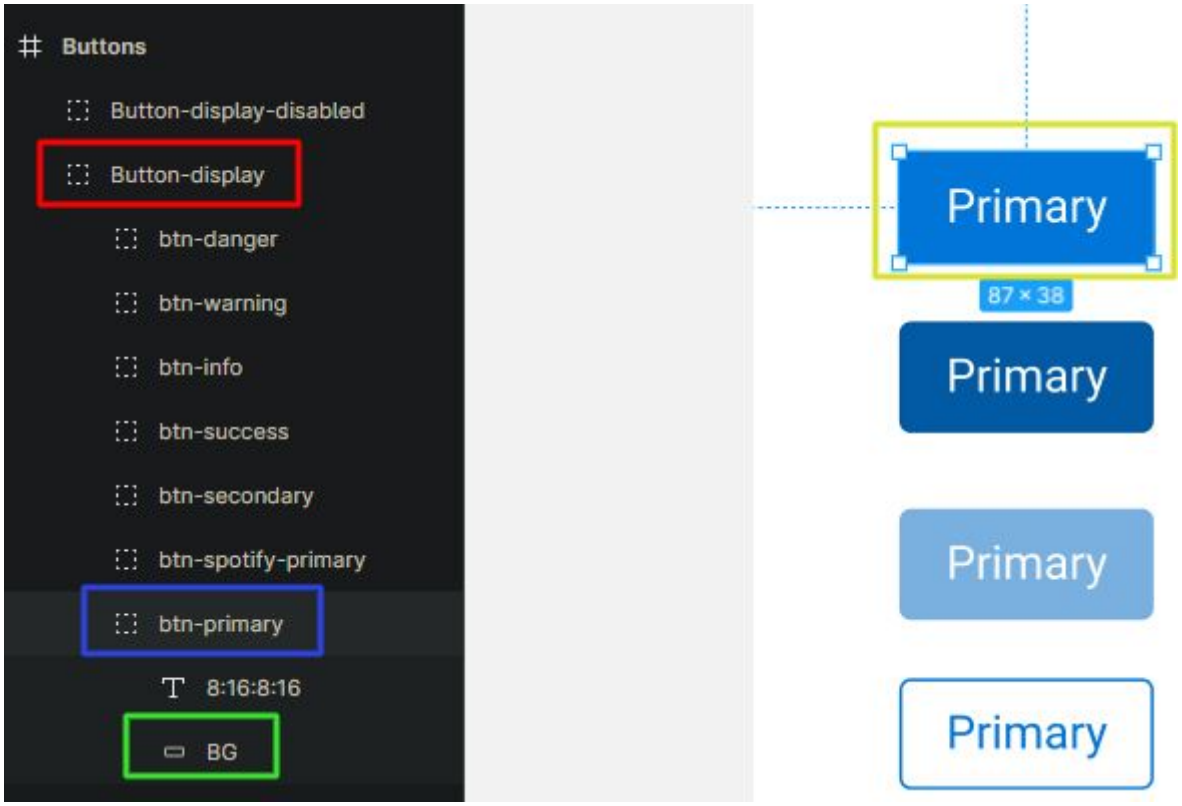


Figura 4.1.1.2: Estructura de datos de la sección de botones en el Kit-Ui de Figma.

Para ello debemos estar seguros que dentro de los **Buttons -> <Subgrupo>** quede seleccionado el elemento **btn-**<deseado>, en el ejemplo de la Figura 4.1.1.2 sería el

btn-primary. Ya entonces podemos modificar cualquiera de las siguientes propiedades dentro de la sección que se muestra en la Figura 4.1.1.3:

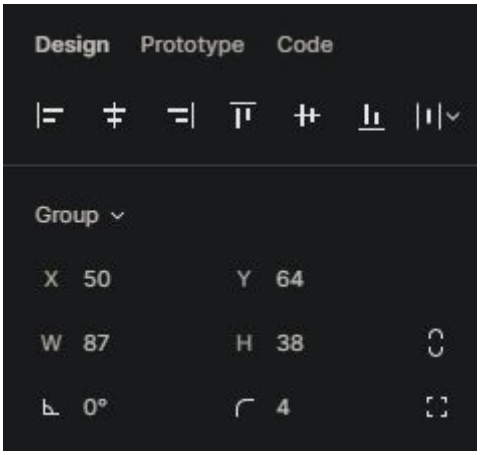


Figura 4.1.1.3: Sección 1 / 2 de propiedades a modificar en un botón hecho en Figma.

- W representa el largo del botón.
- H representa el alto del botón.
- representa el porcentaje del borde curvo.

Y por último dentro del **btn-<deseado>,** ejemplo **btn-primary**, podemos cambiar los colores actualizando algunas propiedades del componente **BG** como podemos ver en la Figura 4.1.1.4:

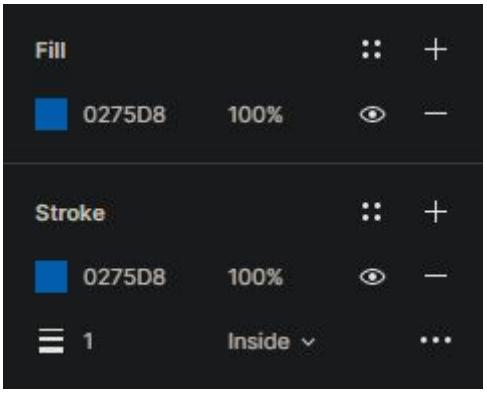


Figura 4.1.1.4: Sección 2 / 2 de propiedades a modificar en un botón hecho en Figma.

- **Fill** permite cambiar el color de fondo del botón.
- **Stroke** permite cambiar el color del borde del botón.

Este proceso es el mismo para todos los elementos en todos los subgrupos de botones. Es por este motivo que algunas de las convenciones a mantener cuando se quiera crear nuevos botones son:

1. Crear un nuevo subgrupo o bien crear el elemento **btn-<nombre>** dentro de un **subgrupo** ya existente.
2. Siempre los botones deben llamarse **btn-<nombre>.**
3. Dentro de un mismo subgrupo **no puede haber dos botones con el mismo nombre.**
4. Todos los botones deben tener un elemento llamado **BG** de tipo rectángulo como se muestra en la Figura 4.1.1.5 para definir los colores de fondo y border.

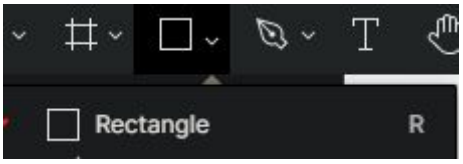


Figura 4.1.1.5: Botón para crear una figura de tipo rectángulo.

Crear un nuevo botón

Este proceso es tan simple como copiar un botón existente y pegarlo dentro de otro subgrupo o bien del mismo en que se encuentre. Al momento de renombrarlo o modificarlo, debe hacerse según lo especificado anteriormente.

Si fuese necesario crear un nuevo subgrupo, se puede copiar también todo un subgrupo y pegarlo y luego actualizar o borrar los botones deseados.

Cómo convenciones en este proceso se repiten las mismas explicadas anteriormente, pero además, **no pueden haber subgrupos con el mismo nombre**.

4.1.2 Colores

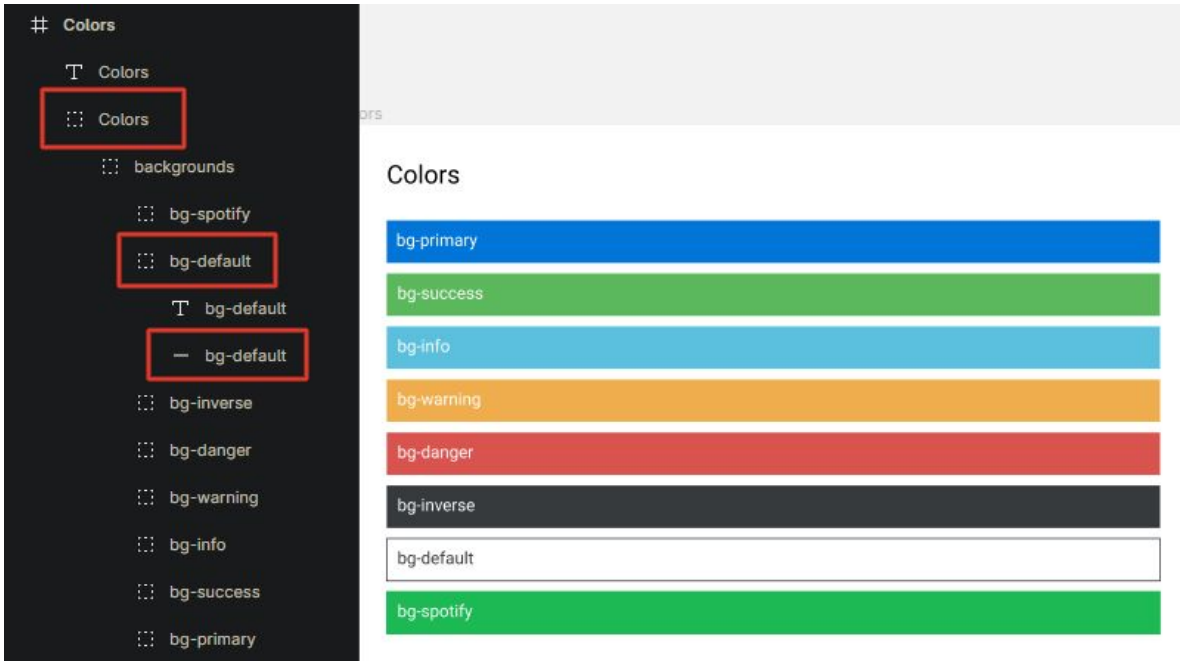


Figura 4.1.2.1: Estructura de datos de la sección de colores en el Kit-Ui en Figma.

Cada elemento dentro de **Colors -> backgrounds** representa un color en Figma como podemos ver en la Figura 4.1.2.1. Estos elementos **bg-<nombre>** tiene en cada uno un elemento adicional **con el mismo nombre**.

Actualizar un color

La única propiedad que se debe cambiar para actualizar el color sería la propiedad **Fill** dentro del elemento que se encuentra en la Figura 4.1.2.2:

Figura 4.1.2.2: Elemento y propiedad para modificar un color en Figma.

Crear un nuevo color

Este proceso es tan simple como copiar un color existente y pegarlo dentro **backgrounds** como podemos ver en la Figura 4.1.2.3. Luego si se quiere renombrar o modificar, se debe hacer según lo especificado anteriormente.

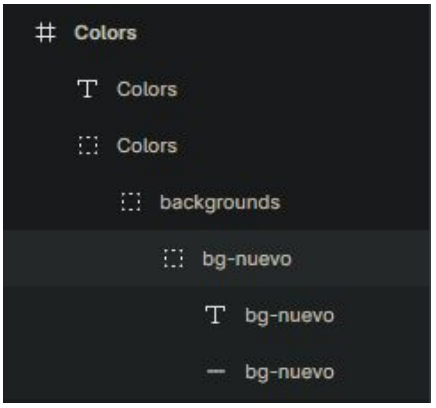


Figura 4.1.2.3: Color copiado dentro de la estructura de colores en Figma.

Como convención para la creación de colores, no puede haber **colores con el mismo nombre**.

4.1.3 Tipografías

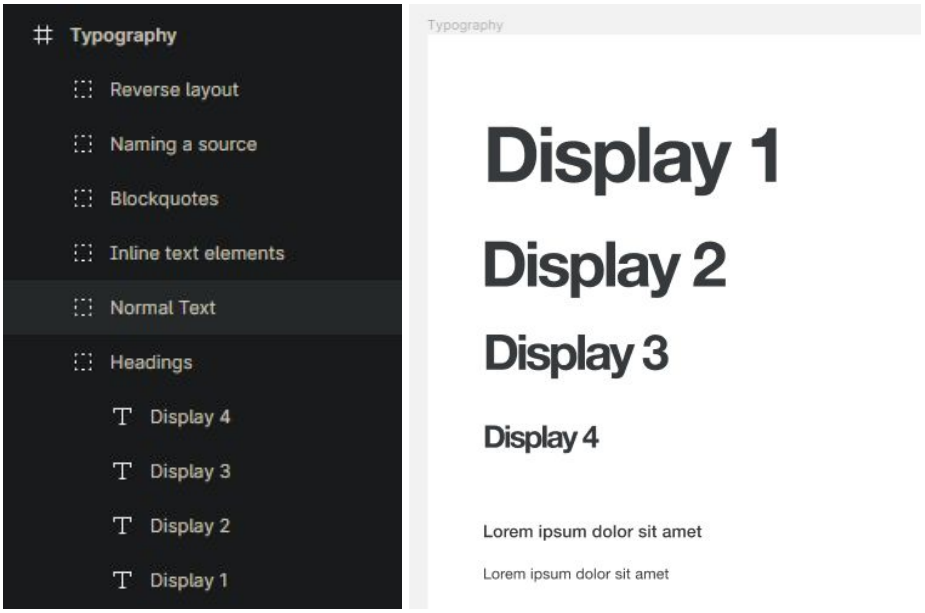


Figura 4.1.3.1: Estructura de datos de la sección de tipografías en el Kit-Ui en Figma.

En lo referente a **tipografías**, contamos con un **grupo** y un **tipo de tipografía**. Donde cada elemento  representa una tipografía diferente como podemos ver en la Figura 4.1.3.1.

Actualizar una tipografía

Se deberá seleccionar un elemento  dentro de un grupo, y cambiar las propiedades dentro de **Text**, como se puede ver en la Figura 4.1.3.2. Por ejemplo el **tipo de fuente, tamaño, espaciado, alineación, etc.**

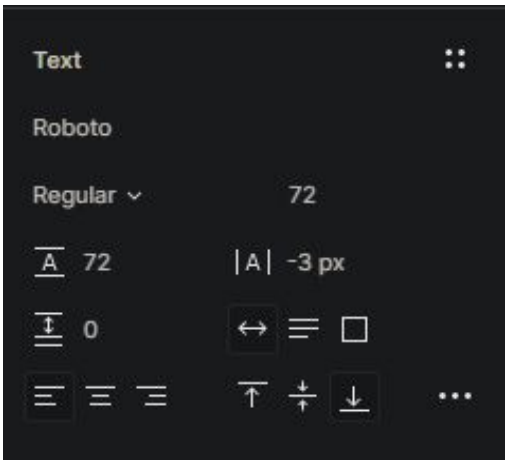


Figura 4.1.3.2: Propiedades a modificar de una tipografía dentro de Figma.

Crear una nueva tipografía

Este proceso, como los anteriores, consiste en cómo copiar una tipografía existente y pegarlo dentro **algunos de los grupos**:

1. Normal Text
2. Headings

Dentro del mismo, esta nueva tipografía deberá contener un **nombre único** como podemos ver en la Figura 4.1.3.3**:**

Figura 4.1.3.3: Tipografía agregada dentro de la estructura de datos de las tipografías.

4.1.4 Badges

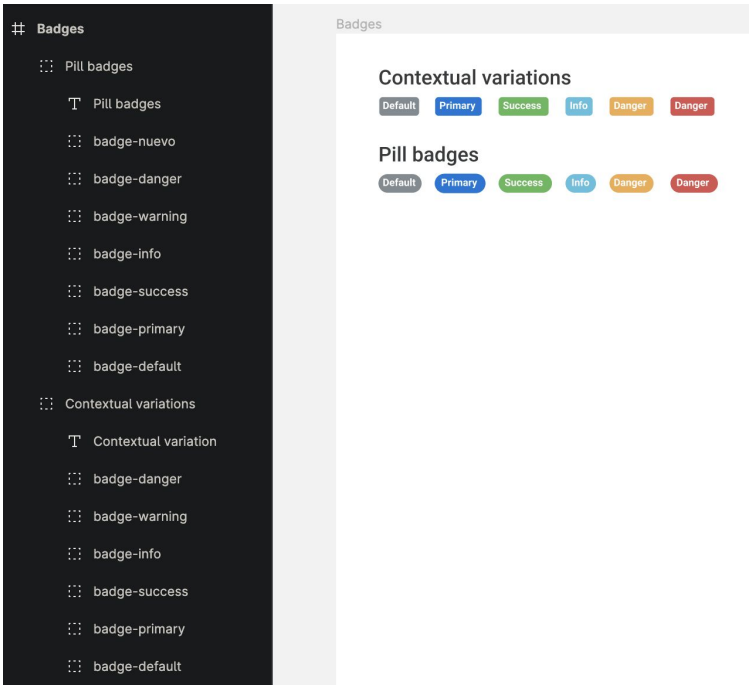
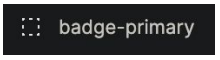


Figura 4.1.4.1: Estructura de datos de la sección de badges en el Kit-Ui en Figma.

Dentro de la Figura 4.1.4.1 podemos observar dos grandes grupos de badges: **Pill badges** y **Contextual variation.**

Cada uno de los badges se representa de manera sencilla, toman un alto y ancho dinámico en base al contenido del mismo. Por lo que el diseñador tiene a su alcance la posibilidad de modificar **únicamente** los atributos referentes al **color de fondo** y **borde**.

Actualizar un badge

Luego de haber seleccionado un elemento 

, es posible modificar el **color de fondo** y **borde** del mismo actualizando las propiedades del **BG** como podemos ver en la Figura 4.1.4.2.

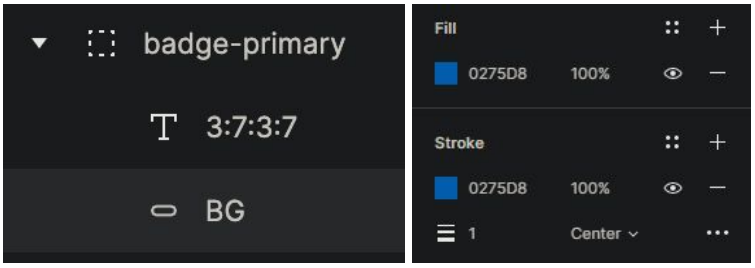


Figura 4.1.4.2: Elemento y propiedades para modificar un badge en Figma.

Crear un nuevo badge

En este punto debemos copiar un badge existente y pegarlo en algunos de los dos grupos mencionados anteriormente. Dentro del mismo, este nuevo badge deberá contener un **nombre único y un elemento** llamado **BG** de tipo ▭ como podemos ver en la Figura 4.1.4.3.

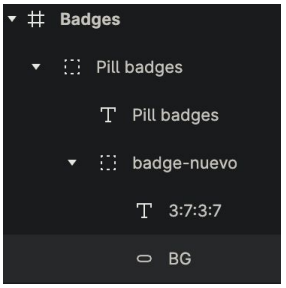


Figura 4.1.4.3: Badge agregado dentro de la estructura de datos de los badges.

4.1.5 Alerts

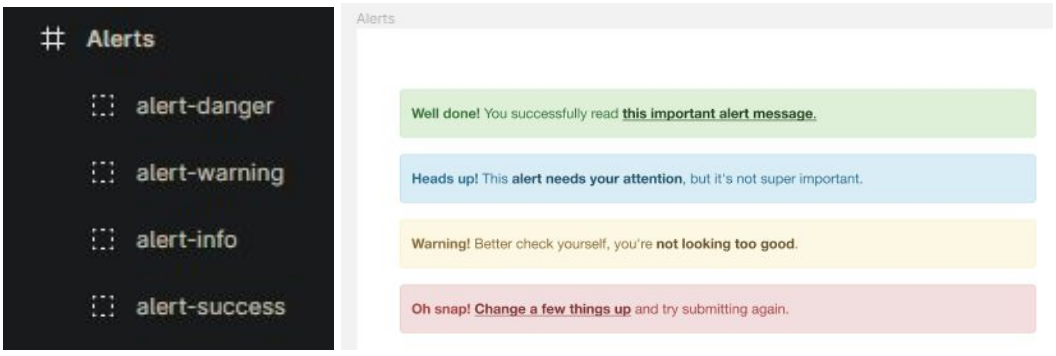


Figura 4.1.5.1: Estructura de datos de la sección de alerts en el Kit-Ui en Figma.

Como podemos ver en la Figura 4.1.5.1 los **alerts** constan de elemento **alert-<nombre>**, del cual es posible cambiar varias propiedades como:

1. Alto
2. Ancho
3. Color de fondo
4. Color de borde
5. Porcentaje del borde curvo
6. Color de texto

Actualizar un alert

Para modificar un alert, primero se debe seleccionar uno de ellos, por ejemplo alert-success. Como podemos ver en la Figura 4.1.5.2, hay dos elementos fundamentales T y BG. Cada unos de estos permite modificar atributos particulares del **alert**.

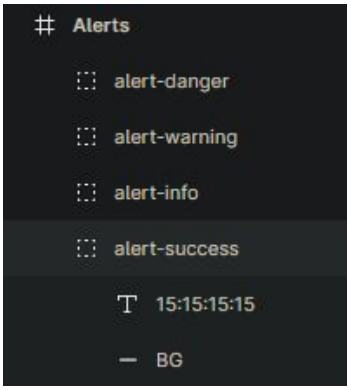


Figura 4.1.5.2: Elementos para modificar un alert en Figma.

Como podemos ver en la figura El elemento T, permite cambiar el **color del texto** del alert como podemos ver en la Figura 4.1.5.3.

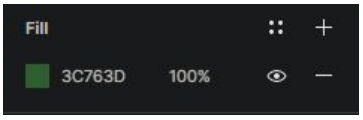


Figura 4.1.5.3: Propiedad para cambiar el color del texto en un alert.

Como podemos ver en la Figura 4.1.5.4 el elemento BG, nos permite cambiar el color de fondo como a su vez **alto, ancho y curvatura del borde.**

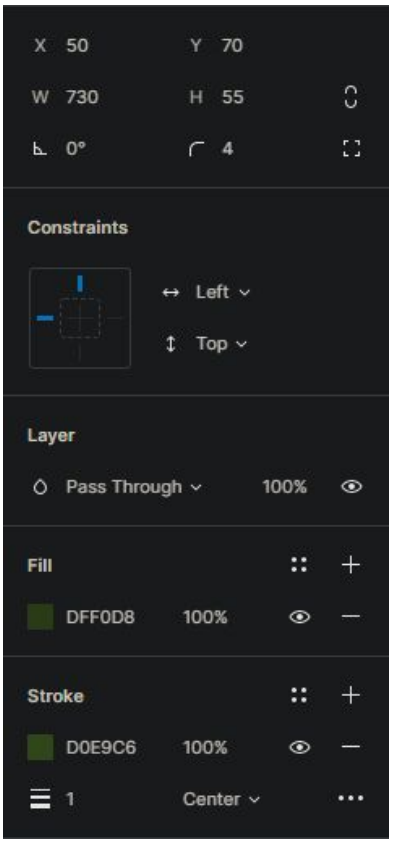


Figura 4.1.5.4: Propiedades para cambiar color, dimensiones y bordes de un alert.

Crear un nuevo alert

En este punto debemos clonar un alert existente. Dentro del mismo, este nuevo alert deberá contener un **nombre único y un elemento** llamado **BG** de tipo y un elemento tipo .

4.2 Especificación de plugin a entregar y diagrama de arquitectura elegida

En esta sección explicaremos qué tipo de entregable se le dará a un equipo de desarrollo, qué componentes lo conforman y cómo interactúan éstos entre ellos.

En primer lugar, lo que recibirá el equipo de desarrollo es un link a un repositorio público de Github, y dentro de este repositorio encontraremos un código boilerplate para poder utilizar la integración. Se denomina **boilerplate** a un conjunto de secciones de código, que suelen repetirse de proyecto en proyecto para que éste funcione según lo deseado.

Supongamos que necesitamos un stack de desarrollo que consiste en librerías como React, Babel, Express, Jest, Webpack, etc. Cada vez que se empiece un nuevo proyecto, se deben inicializar y configurar estas librerías para que funcionen correctamente entre ellas. Con cada proyecto nuevo que se empiece, esta inicialización y configuración se deberá repetir. También podríamos introducir, no intencionalmente, inconsistencias entre proyectos en cuanto a cómo estas librerías se configuran.

En estos casos es cuando necesitamos código boilerplate; éstos funcionan como una plantilla reutilizable que puede clonarse y reusarse en cada proyecto que necesite las configuraciones que el boilerplate presente.

Como podemos ver en la Figura 4.2.1, nuestro boilerplate presentará una aplicación desarrollada en React, Redux y Material-UI para el stack del front-end y Ruby on Rails para el stack del back-end. A su vez esta aplicación vendrá con la integración a un archivo Figma lista para su uso. Siguiendo una serie de pasos especificados en el archivo README del repositorio, se podrá tener la aplicación conectada a Figma funcionando en un entorno local.

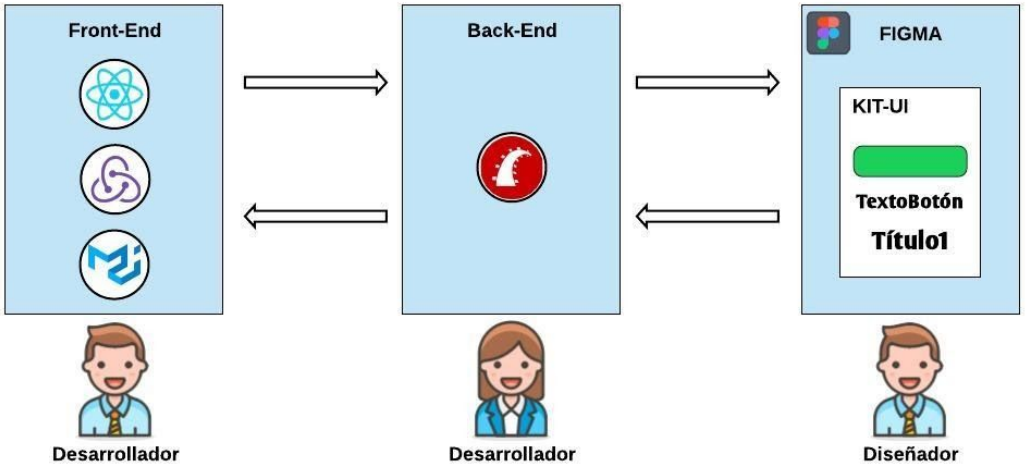


Figura 4.2.1: Estructura del boilerplate a entregar.

Luego será tarea del desarrollador y diseñador cambiar algunas configuraciones para que la aplicación se conecte al archivo de Figma que ellos escojan.

Dentro de este boilerplate la única pantalla, vista o página desarrollada será la guía de estilos. Como podemos ver en la Figura 4.2.2, esta guía contendrá la visualización y código asociado para su utilización de todos los componentes visuales que están actualmente importados desde Figma. Para acceder a ella se debe acceder mediante url a **/style_guide.** Si el archivo conectado a la aplicación cambia, la guía se actualizará con los nuevos componentes que se importen correctamente.

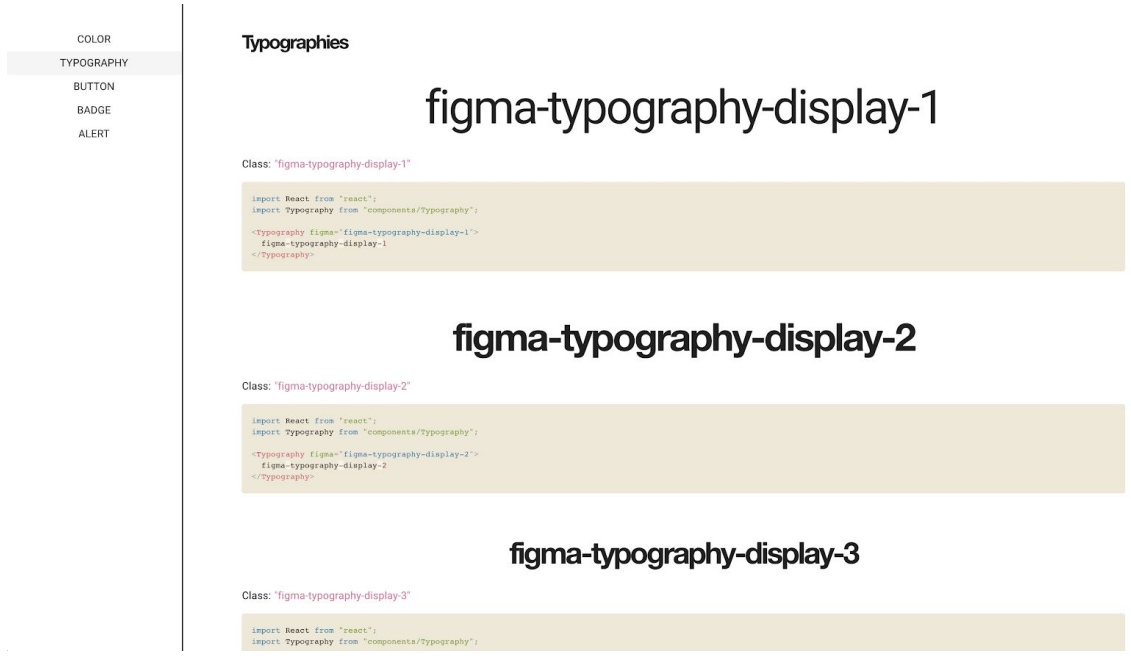


Figura 4.2.2: Guía de estilos del boilerplate a entregar.

4.2.1 Front-End

Para construir la sección de la aplicación que interactúa directamente con el usuario dispusimos de un conjunto de tecnologías para su correcto funcionamiento:

- ReactJS: es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página.
- Redux: es una pequeña librería con una API simple y limitada, diseñada para ser un contenedor del estado de la aplicación.
- Material-UI: es una librería UI, destinada a utilizarse dentro de una aplicación React, que permite reutilizar sus componentes construidos con la filosofía Material Design [MaterialDesign20].

La plataforma front-end será la encargada de solicitar al back-end los estilos asociados al archivo de Figma que se encuentre dentro de la configuración. Cuando el back-end le envíe los datos solicitados, se actualizará el tema de Material UI con la información y se actualizarán los componentes asociados al tema. Para que este flujo de información funcione correctamente existen ciertos componentes interconectados dentro de la aplicación front-end. No es el objetivo de esta sección ahondar en detalles técnicos, pero creemos que vale la pena mencionarlos y describir brevemente su función principal.

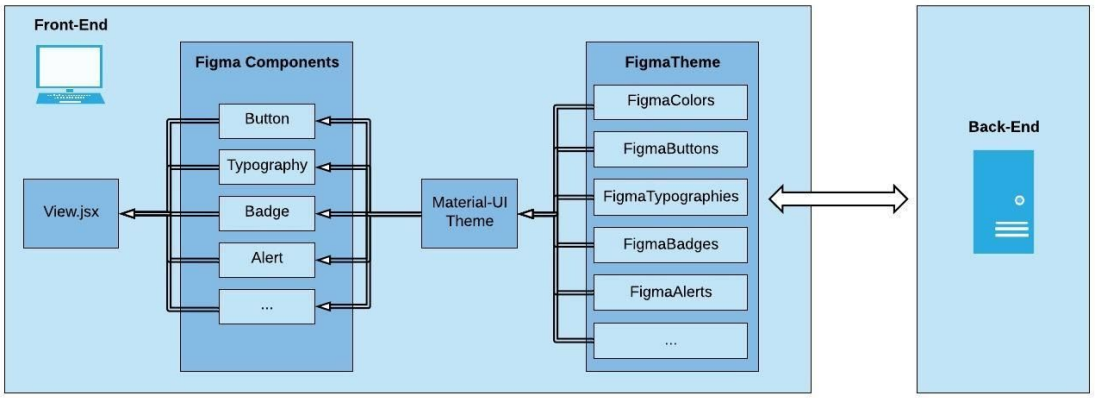


Figura 4.2.1.1: Estructura y flujo de información dentro de los componentes más importantes del Front-End.

Como podemos ver en la Figura 4.2.1.1 los componentes esenciales dentro del Front-End son:

- FigmaTheme: cada vez que se inicializa o se refresca la aplicación, este componente es el encargado de solicitar estilos CSS a cada uno de sus componentes internos. Cada uno de estos componentes internos se encargará de solicitar al Back-End los estilos asociados al componente que representa:

    - FigmaColors: solicitará estilos relacionados a la paleta de colores.
    - FigmaButtons: solicitará estilos relacionados a los botones de la aplicación.
    - FigmaTypographies: solicitará estilos relacionados a las tipografías de la aplicación.
    - FigmaBadges: solicitará estilos relacionados a las insignias de la aplicación.
    - FigmaAlerts: solicitará estilos relacionados a las alertas de la aplicación.

    Luego de que cada componente interno obtenga su respectiva información CSS, el FigmaTheme iterará por todos ellos actualizando el tema de Material-UI con la información CSS. Esta actualización consiste en crear nuevas clases CSS dentro del tema para que puedan ser usadas por la aplicación.

- Material-UI Theme: es el tema de la aplicación Front-End. Será actualizado por FigmaTheme con datos del Back-End cada vez que se inicialice o refresque la aplicación.

- Figma Components: son componentes React que utilizan las clases Figma del tema de Material-UI. Fueron creados para utilizar las clases creadas a partir de Figma de una manera fácil y rápida. Existen tantos componentes como componentes internos que posea el Figma Theme.

- View.jsx: representa todas las vistas de la aplicación React y dentro de éstas se importarán los Figma Components.

4.2.2 Back-End

El componente principal en nuestro **Back-End**, es el **FigmaService**. Este servicio se encarga de procesar los llamados que se realicen a cada **endpoint** del Back-End para luego obtener la información solicitada llamando a la **API de Figma**. Cuando se realice un llamado a algún endpoint, este servicio se encargará de obtener la información del nodo correspondiente y devolver la información al Front-End en un formato **JSON** más simplificado y listo para su transformación a clase CSS.

Tecnologías utilizadas

- Ruby On Rails: Es un framework de desarrollo basado en el lenguaje Ruby [RoR20].
- Heroku: Es una plataforma que permite integración de variables de entorno, despliegue y versionado de despliegues [Heroku20].
- Archivos .env para la definición de variables de entorno por entorno. Permite definir diferentes configuraciones para cada entorno.

Procesamiento de la información de Figma

A la hora de procesar un archivo en Figma, debemos obtener una autorización por parte de la plataforma para utilizar su API [FigmaAutenticación20]. Sin esta autorización no se podrá acceder a la información respecto a los diseños de un archivo determinado.

Por este motivo, nuestro Back-End no solo debe conocer qué archivo se va a procesar, sino también contar con una **API_KEY** generada desde el dashboard de Figma. Esta **API_KEY** se debe agregar a la configuración de la aplicación para usarla correctamente.

Nuestra aplicación Front-End solicitará a nuestro Back-End la información relacionada a los estilos de cada componente que esté integrado. Ese proceso consiste en la realización de diferentes llamadas del Front-End (una por cada componente visual integrado) a diferentes endpoints del Back-End (uno por cada componente visual integrado). Cada uno de estos

endpoints del Back-End, haciendo uso de la API de Figma, obtendrá los estilos del componente visual al que esté asociado como podemos ver en la Figura 4.2.2.1.

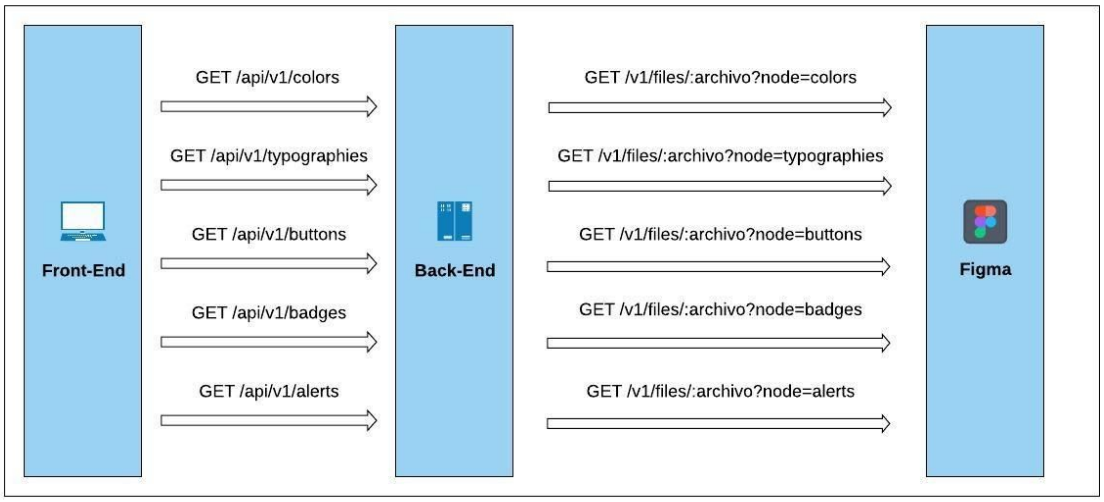


Figura 4.2.2.1: Uso de endpoints locales y de Figma para obtener información de estilos.

Cada llamada a la API de Figma permite obtener información respecto a un **nodo** dentro de un **archivo** dado:

- **Archivo**: Un archivo de Figma, es donde el diseñador puede representar componentes, paginas, etc. En nuestro boilerplate, por temas de simplicidad, usamos un único archivo para representar todos los componentes. Un archivo en Figma es identificado por una serie de caracteres luego de **"/file/..."**. Por ejemplo en la siguiente url, https://www.figma.com/file/FBjrbCOv6x3BV89QVfmy76/kit-UI---Tesis-(Production)?node-id=0%3A1, el **identificador** del archivo, sería **FBjrbCOv6x3BV89QVfmy76.**
- **Nodo:** Dentro de un archivo, podemos encontrar secciones o nodos, estos permiten identificar o acceder de manera rápida a una parte del archivo. Cada vez que se llame a la API de Figma, se puede obtener información precisa y detallada de un nodo o sección en particular.

A su vez, la API de Figma devuelve no sólo la información relacionada a los estilos de los componentes, sino también información innecesaria para nuestra integración (ids, posicionamiento absoluto dentro del archivo, tipo de forma, etc).
Procesar toda esta información de manera que nuestro Front-End pueda hacer uso de ella para "dibujar/representar" cada componente, conlleva un proceso particular y distinto para cada uno de éstos. No es la misma información que se debe procesar para los colores que

para los botones, alertas o tipografías. Por esta razón, decidimos alojar cada componente visual dentro de un nodo específico del archivo de Figma para luego acceder a cada uno de estos de manera independiente y hacer el procesamiento correspondiente.

Cuando se necesiten obtener los estilos de los componentes visuales de la aplicación, se hará una llamada al Back-End por cada componente integrado, ya que habrá un endpoint con un manejador específico para cada componente. Como podemos ver en la Figura 4.2.2.2, haciendo el uso de múltiples llamadas en paralelo utilizando endpoints independientes por cada componente en vez de una llamada única con un procesamiento único, podemos optimizar y reducir los tiempos de respuestas de cada operación.

Figura 4.2.2.2: Uso de endpoints para obtener estilos de componentes integrados en forma paralela.

Como podemos ver a continuación cada request tiene un tiempo independiente y permite hacer uso de la información de manera temprana en la aplicación. Luego el Front-End procesa las respuestas de cada endpoint y configura el tema de la aplicación acorde a los resultados.

Nuestro Back-End del boilerplate posee los siguientes endpoints creados y listos para usar:

- GET /api/v1/colors: como podemos ver en la Figura 4.2.2.3, este endpoint permite obtener información sobre los colores definidos en Figma y devuelve únicamente los nombres con sus respectivos valores para ser usados directamente.

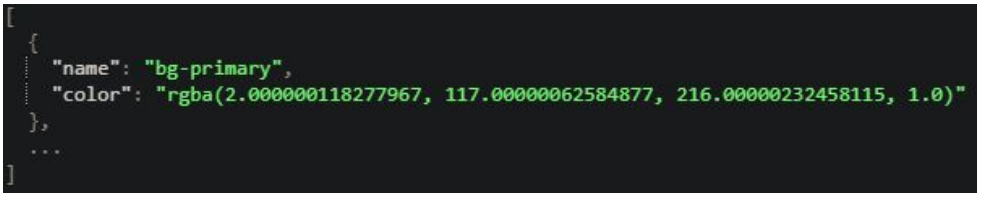


Figura 4.2.2.3: Resultado de endpoint GET /api/v1/colors.

- GET /api/v1/typographies: como podemos ver en la Figura 4.2.2.4, este endpoint permite obtener información sobre las tipografías definidas y los atributos de las mismas.

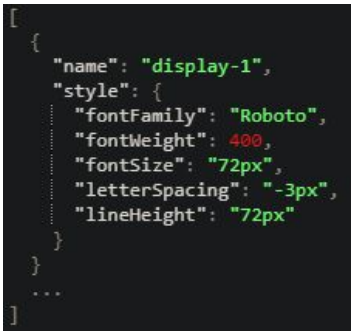


Figura 4.2.2.4: Resultado de endpoint GET /api/v1/typographies.

- GET /api/v1/buttons: como podemos ver en la Figura 4.2.2.5 devuelve al Front-End, todos los botones definidos en todos los grupos así como las características de los mismos.

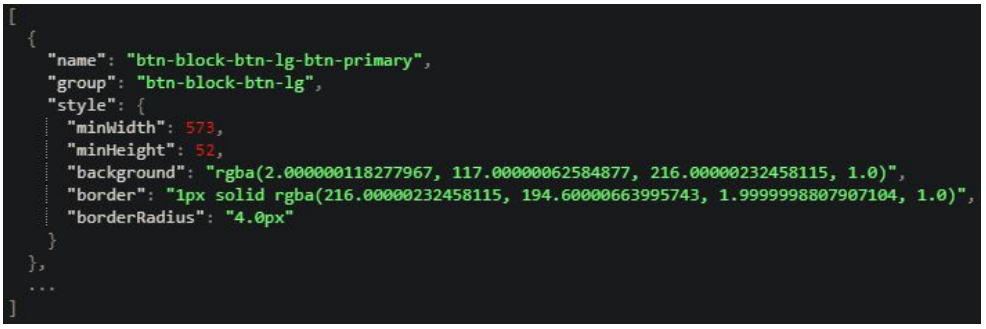


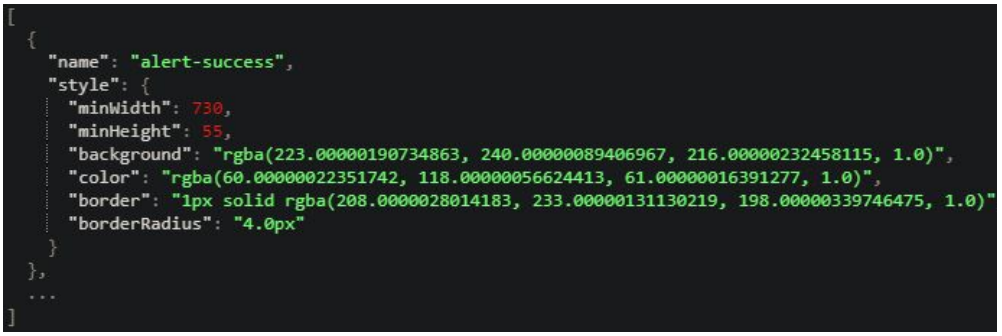
Figura 4.2.2.5: Resultado de endpoint GET /api/v1/buttons.

- GET /api/v1/badges: como podemos ver en la Figura 4.2.2.6, este endpoint procesa todos los badges en el nodo correspondiente, y devuelve los nombres y las especificaciones de cada uno.

```
[
  {
    "name": "contextual-variations-badge-default",
    "group": "contextual-variations",
    "style": {
      "minWidth": 20,
      "minHeight": 20,
      "width": "max-content",
      "height": "max-content",
      "background": "rgba(128.71032625436783, 137.90928304195404, 145.3563666343689, 1.0)",
      "borderRadius": "4.0px"
    }
  },
  ...
]
```


Figura 4.2.2.6: Resultado de endpoint GET /api/v1/badges.

- GET /api/v1/alerts: como podemos ver en la Figura 4.2.2.7, este endpoint permite obtener información sobre los alerts diseñados y sus características.



```
[
  {
    "name": "alert-success",
    "style": {
      "minWidth": 730,
      "minHeight": 55,
      "background": "rgba(223.00000190734863, 240.00000089406967, 216.00000232458115, 1.0)",
      "color": "rgba(60.00000022351742, 118.00000056624413, 61.00000016391277, 1.0)",
      "border": "1px solid rgba(208.0000028014183, 233.00000131130219, 198.00000339746475, 1.0)",
      "borderRadius": "4.0px"
    }
  },
  ...
]
```


Figura 4.2.2.7: Resultado de endpoint GET /api/v1/alerts.

CAPÍTULO 5

Experimento

5.1 Introducción

Con el fin de poder validar las ventajas que esta integración nos proveería, decidimos que la mejor opción era realizar un experimento para ver nuestro integración con Figma en acción. Necesitábamos poner a prueba todo lo investigado aplicándolo en una situación real, con un sistema real, desarrolladores y diseñadores reales, y tareas que aparezcan en la cotidianidad dentro de un proyecto de software.

De esta forma, al momento de resolver alguna tarea relacionada con la UI del proyecto, podremos utilizar el enfoque propuesto en nuestra integración y compararlo con una resolución que utilice el enfoque convencional. Al comparar ambos enfoques, nos concentramos en validar que el tiempo de espera entre una propuesta de cambio de UI y el momento en el que efectivamente se concreta, es menor utilizando el enfoque propuesto en este trabajo de investigación.

5.2 Objetivos

El experimento fue realizado con el objetivo de poder obtener resultados que efectivicen la reducción de tiempo al utilizar nuestro enfoque de trabajo. Este enfoque, como bien se explicó en la sección 3.1.3. Enfoque de trabajo, se basa en poder realizar modificaciones UI sin la necesidad de un desarrollador de software para escribir el código necesario. Y si entramos en detalle de lo que esto significa, no sólo se eliminaría el accionar de un desarrollador de software, sino también todas las tareas implicadas en un desarrollo de software que involucran a Business Analysts, Product Managers, Equipo de desarrollo, Testers, etc.

Este experimento produjo diferentes métricas las cuales fueron analizadas y utilizadas para respaldar nuestra conclusión final. Dentro de éstas podemos nombrar el tiempo utilizado por un desarrollador para realizar el cambio de código, el tiempo utilizado por el diseñador para realizar el cambio, el tiempo de utilizado para testear los cambios, el tiempo utilizado para la revisión de código, el tiempo “muerto” que se encuentra entre el pasaje tareas, el tiempo total desde que se solicita el cambio hasta que queda impactado en la aplicación real, la cantidad de tarjetas/tickets/tareas creadas, la cantidad de líneas de código escritas, etc.

A su vez, a partir de este experimento, obtuvimos un feedback de la utilización de la integración. A partir de este feedback, nos enfocamos en deducir ventajas y desventajas de

uso, complicaciones más comunes, puntos fuertes de la integración y sugerencias de mejoras para enfocar futuros desarrollos, factibilidad de uso cotidiano, estrategias para la educación inicial sobre la herramienta, entre otras. Todo este análisis permite concluir cuán posible es la adopción de este enfoque por parte de la industria del software.

5.3 Presentación del experimento

Dentro de la industria del software, existen diferentes procesos, metodologías, formatos y herramientas que han sido adaptados de manera casi universal para lograr el desarrollo efectivo de un producto. Con el fin de poder contextualizar este experimento en un marco lo más acercado a la realidad posible, utilizaremos dichos elementos en cada etapa del desarrollo con el fin de obtener resultados confiables y repetibles.

5.3.1 Equipo y metodología de trabajo

Para poder hacer una comparación entre el enfoque convencional y el enfoque propuesto en este trabajo utilizamos dos equipos, cada uno trabajó con un enfoque distinto. Llamamos equipo A al equipo que utilizó el enfoque nuevo y equipo B al equipo que utilizó el enfoque convencional. Cada uno de estos equipos estuvo formado por un diseñador y dos desarrolladores de software, no contamos con Product Managers ni Testers por simplicidad, pero las tareas que requerían de estos roles fueron realizadas por los desarrolladores de software y diseñador en caso de algún testeo UI.

Estos dos equipos trabajaron utilizando metodología Scrum, es decir, que realizaron una Sprint Planning para estimar las tareas que se propusieron dentro de este experimento, pero no realizaron Reuniones Diarias, Demos ni Retrospectivas ya que estas ceremonias no tienen relación con los objetivos que queríamos alcanzar ni las métricas que deseábamos obtener. Dentro de la Sprint Planning, participaron sólo los desarrolladores de software en el caso del equipo B y se sumó el diseñador en el caso del equipo A. Se utilizó Poker Planning [Haugen06] para realizar las estimaciones.

5.3.2 Formato de tareas

Ambos equipos crearon tareas/tickets dentro de Trello asociados a las problemáticas que se les dieron a resolver. Como no contamos con un Product Manager o Business Analyst, este trabajo fue realizado por los desarrolladores de software y diseñadores. Estos tickets incluyen:

- Título.
- Descripción.
- Puntaje de estimación.

- Prioridad (Alta, Media y Baja).
- Estado (Backlog, En Progreso, En Testeo, Finalizado).
- Criterio de aceptación.
- Comentarios.
- Diseños asociados.

5.3.3 Repositorios

La metodología de trabajo utilizada en este experimento está basada en el uso del repositorio Github como manejador de versionado de código. Ambos equipos clonaron el mismo repositorio dentro de Github donde se encuentra el sistema sobre el cual trabajaron.

Se trabajó con el patrón feature-branch como se muestra en la Figura 5.3.3.1. En este patrón, el desarrollador crea una nueva rama a partir de una rama base para trabajar en una nueva funcionalidad. Sobre esta rama se realiza el trabajo necesario y una vez terminado el trabajo, se integran los cambios dentro de la rama base.

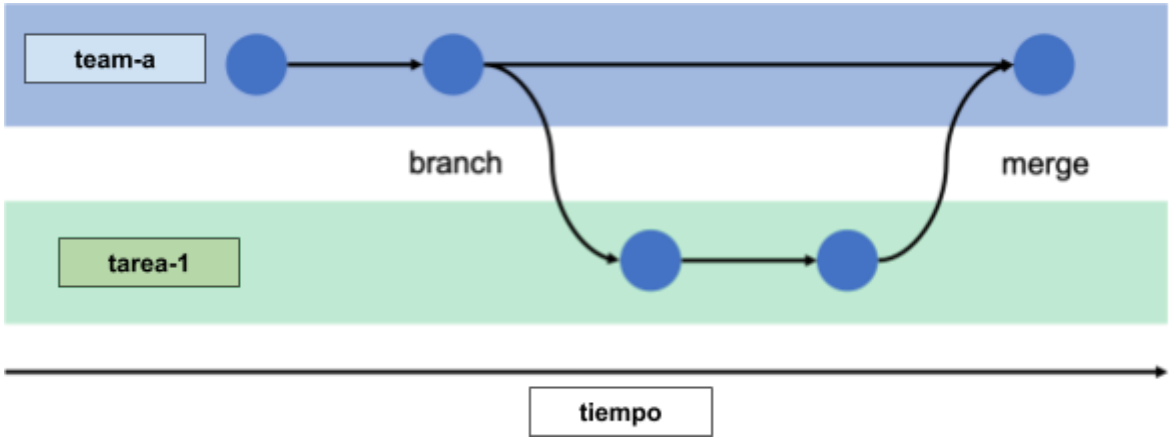


Figura 5.3.3.1: Metodología feature-branch para creación de features en el código.

Sobre estas ramas bases los desarrolladores trabajan e incluyen los cambios que se solicitan mediante el uso de Pull Requests. Los Pull Requests son peticiones que hace el desarrollador para incluir nuevo código (commits) dentro de una rama destino, por lo tanto, el equipo A y B crearon un Pull Request por cada tarea que hayan creado.

A su vez, estos Pull Requests son revisados y aprobados por otro desarrollador del equipo antes de incluirse en la rama destino, y en caso de encontrarse alguna posible mejora o error, se solicita la modificación del código del Pull Request. Este proceso se repite hasta que el Pull Request queda aprobado. Sólo en este punto el Pull Request puede ser “mergeado” a la rama destino.

Por lo tanto, cada rama tiene un historial de commits que se corresponde cada uno con una tarea en particular. Por ejemplo, si del experimento se hubiesen creado 5 tareas, se asume que cuando las 5 tareas estén terminadas, se habrán creado 5 Pull Requests, se habrán mergeado 5 Pull Requests a la rama del equipo y la rama del equipo tendrá en su historial:

- Tarea 5.
- Tarea 4.
- Tarea 3.
- Tarea 2.
- Tarea 1.
- Tarea N previa al experimento.

Dentro del repositorio de Github existen cinco ramas/branches: “master”, “team-a”, “team-b”, “team-a-stg” y “team-b-stg”. La rama “master” es el estado actual de la aplicación y la base de la cual se partió para realizar los cambios propuestos. De esta rama “master” salen las ramas “team-a” y “team-b” utilizadas como rama de “Producción”, y las ramas “team-a-stg” y “team-b-stg” como rama de “Staging”. En la siguiente sección explicaremos en detalle para qué sirve cada una.

5.3.4 Entornos de desarrollo

Ambos equipos dispusieron de ambientes de trabajo para realizar el desarrollo de las tareas. Un ambiente de trabajo lo definimos como hardware y software donde se ejecuta una aplicación. Dependiendo del proceso de desarrollo, existirá una cierta cantidad de ambientes por los cuales se irá propagando una aplicación desde su desarrollo hasta su salida a producción.

Cada ambiente tiene su propia copia del código de la aplicación de forma que no haya interferencias en los ambientes y entre los diferentes participantes en la construcción del software. Cada vez que un cambio se aplique en un entorno, se le realizarán los procesos de calidad asociados a éste (tests de unidad, tests de integración, tests de aceptación, etc). Así se irá propagando a los siguientes entornos e irá pasando por diferentes procesos de calidad que asegurarán que cuando el cambio llegue al último entorno, no habrá ninguna regresión. Todo este conjunto de procesos se le denomina proceso de QA (Quality Assurance).

Como podemos ver en la Figura 5.3.4.1, se dispuso de 3 ambientes de trabajo por equipo, y cada ambiente contiene una copia en un instante de tiempo de la rama team-a o team-b.

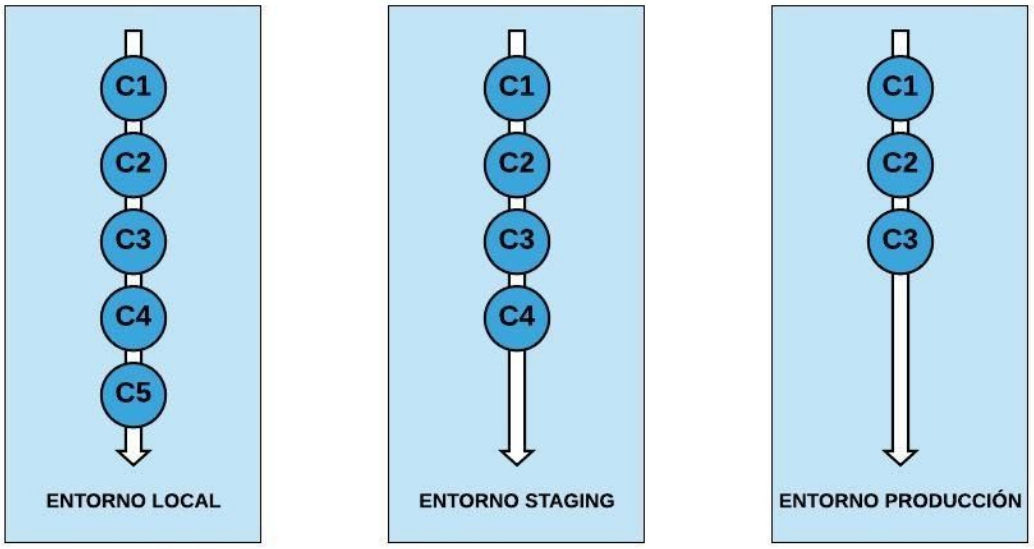


Figura 5.3.4.1: Conjunto de ambientes de desarrollo para utilizar en el experimento.

- Entorno local: es el entorno donde se realizan todos los desarrollos y pruebas del código antes de crearse el Pull Request a la rama destino del equipo. Es la máquina local del desarrollador y una vez que la tarea se considera terminada, se despliega al entorno siguiente.
- Entorno de staging: es el entorno en el cual se despliegan los cambios de un Pull Request una vez aprobados y mergeados. Aquí se realizará el proceso de QA para asegurar que los cambios incluídos en la rama, funcionan como se espera y la tarea asociada cumple con el criterio de aceptación. Si el cambio no funciona como se espera o la tarea no cumple con el criterio de aceptación, se debe solicitar la modificación correspondiente y repetir el proceso anterior hasta que llegue la corrección al entorno.
- Entorno de producción: es el entorno en el cual se despliega la rama que posea cambios luego de realizarse el proceso de QA en el entorno de staging. En este entorno se considera que todas las tareas asociadas a los commits existentes en la rama, cumplen con el criterio de aceptación. No existen cambios o commits que no se hayan testeado previamente. Es el entorno utilizado por el usuario final.

Para lograr automatizar los despliegues a los diferentes entornos, dispusimos de ramas “Producción” y ramas “Staging”. Las ramas “Producción” fueron configuradas para que cuando se les incluya nuevo código, éstas sean desplegadas automáticamente en el ambiente de “Producción”. Las ramas “Staging” tienen el mismo propósito pero el despliegue automático se realiza al ambiente de “Staging”.

Supongamos el caso de un desarrollador del equipo A, que quiere trabajar sobre la Tarea 1. En este caso, el desarrollador crea una nueva rama llamada “tarea-1” a partir de la rama

“team-a-stg”. En esta rama el desarrollador trabaja en lo solicitado y una vez terminado el trabajo, la rama “tarea-1” es integrada a la rama “team-a-stg”. Lo mismo hace el desarrollador del equipo B, pero con su rama “team-b-stg”.

Cuando se realiza la integración a la rama “team-X-stg” se realiza un despliegue automático y el código nuevo puede testearse en el ambiente de “Staging”. Una vez que se haga un testeo satisfactorio dentro de este ambiente, la rama “team-X-stg” es integrada a la rama “team-X” lo cual produce otro despliegue automático pero esta vez, al ambiente de “Producción”. Aquí se realizan los testeos por el usuario final.

5.3.5 Herramienta de gestión de proyecto

El desarrollo de software dentro del mercado suele manejarse con herramientas de administración y gestión de proyectos con el fin de dar visibilidad del estado de las tareas, mejorar la comunicación entre los integrantes del equipo, proyectar objetivos, identificar riesgos, entre otros. Por esta misma razón y con el fin de simular lo mejor posible la gestión de un desarrollo convencional, utilizamos la herramienta Trello para que cada equipo administre cada una de sus tareas.

Como podremos ver en la Figura 5.3.5.1, dentro de la herramienta Trello tenemos tantas columnas como estados pueda tener una tarea:

- Backlog: todavía la tarea no fue empezada.
- En Proceso: la tarea fue tomada por un desarrollador y está en proceso de desarrollo.
- En Revisión: la tarea fue terminada y está en espera de ser aprobada por otro desarrollador.
- En Testeo: la tarea fue aprobada por un desarrollador y se realizó un despliegue al ambiente de Staging para su testeo.
- Finalizado: la tarea cumplió con el criterio de aceptación y se considera terminada.

Figura 5.3.5.1: Board de tareas para utilizar en el experimento.

Durante el experimento cada tarjeta fue ubicada dentro de alguna de las 4 columnas, logrando que con sólo mirar el tablero de Trello se pudiese detectar el estado de todas las tareas y quién está trabajando en cada una.

5.3.6 Aplicación experimental

Para aplicar estos enfoques de trabajo, debíamos disponer de un producto de software sobre el cual poder realizar tareas cotidianas de desarrollo y así corroborar las mejoras que esperamos obtener.

Con este fin, desarrollamos una aplicación web construida con React, Redux y Material-UI en el frontend, y Ruby on Rails en el backend. Esta aplicación utiliza la API de Spotify para buscar artistas, canciones, discos, popularidad de los artistas, imágenes, cantidad de followers de los artistas, género musical de los artistas, entre otros datos. Se debe tener un usuario de Spotify para obtener acceso a todas las funcionalidades.

La aplicación cuenta con las siguientes páginas:

Página de Login (Figura 5.3.6.1)

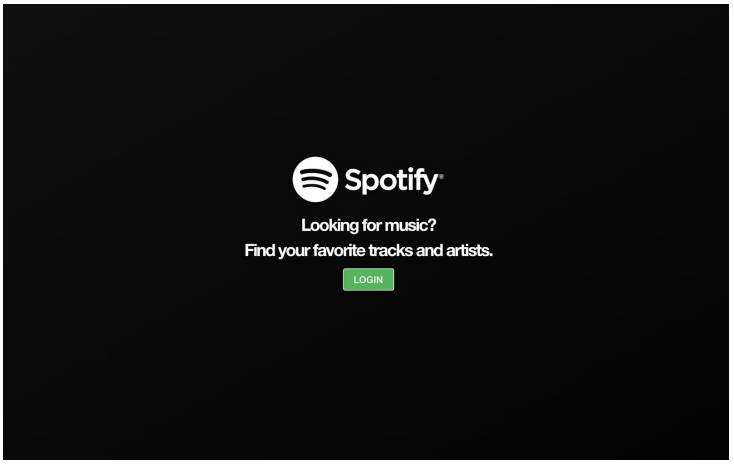


Figura 5.3.6.1: Página de Login de la aplicación experimental.

Para iniciar sesión se debe hacer clic en el botón Login que redirigirá al usuario a la pantalla de inicio de sesión de Spotify.

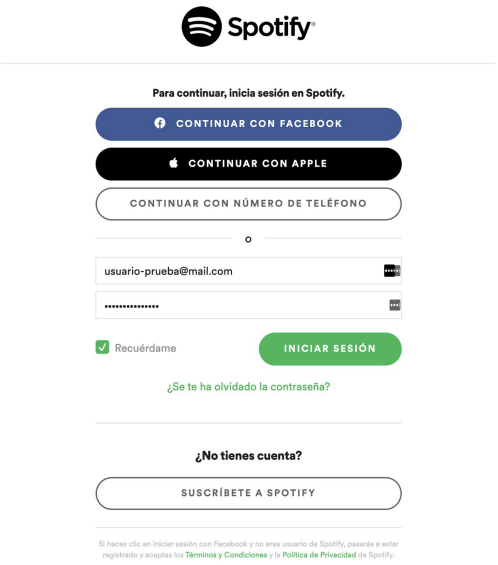


Figura 5.3.6.2: Página de Inicio de Sesión de Spotify de la aplicación experimental.

Como podemos ver en la Figura 5.3.6.2, aquí se deberá ingresar con el usuario y contraseña de una cuenta de Spotify o utilizar alguna de las opciones propuestas de inicio de sesión.

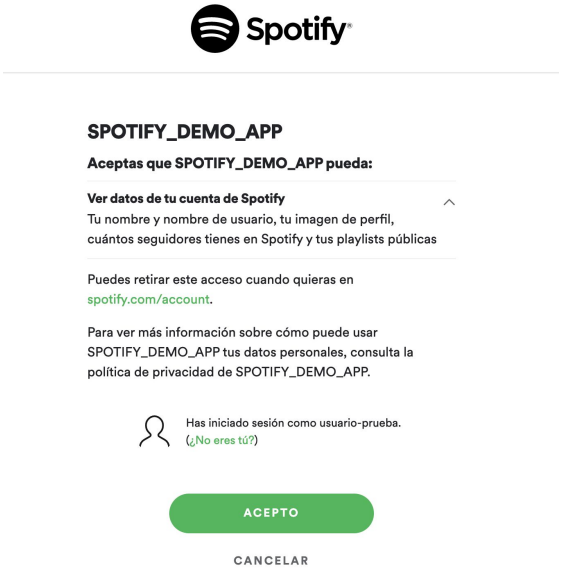


Figura 5.3.6.3: Página de confirmación de términos y condiciones de Spotify de la aplicación experimental.

Una vez ingresado un usuario correcto se mostrará una página de confirmación, como muestra la Figura 5.3.6.3, donde el usuario debe aceptar los términos y condiciones para que la aplicación pueda utilizar datos de su cuenta de Spotify.

Página de Inicio (Figura 5.3.6.4)

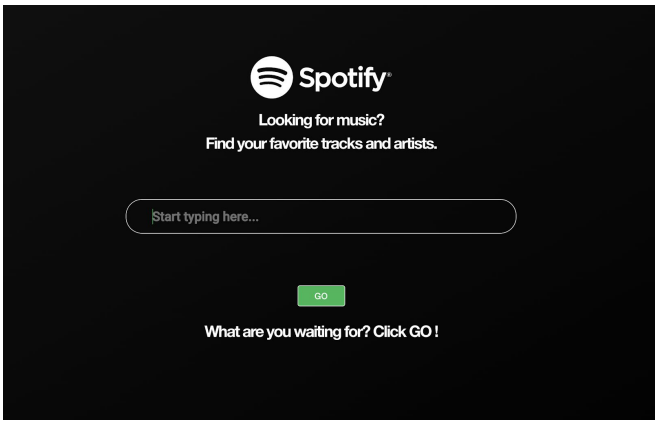


Figura 5.3.6.4: Página de Inicio de la aplicación experimental.

Una vez que el usuario haya iniciado sesión con su usuario de Spotify y haya aceptado los términos y condiciones, será redirigido a la pantalla de inicio donde puede realizar la búsqueda de artistas y canciones. Una vez ingresado el artista o la canción a buscar, se deberá hacer clic en el botón GO para acceder a la siguiente pantalla con los resultados de la búsqueda.

Página de Resultados de Canciones y Artistas

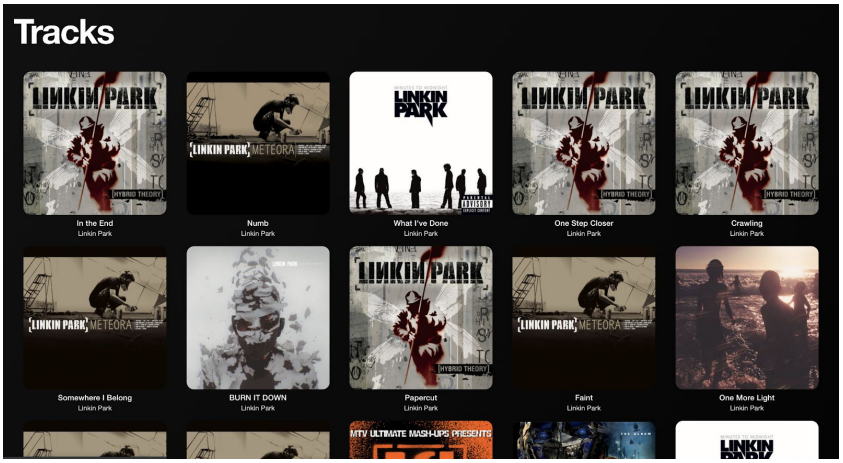


Figura 5.3.6.5: Sección de Resultados de Canciones de la aplicación experimental.

A partir de los resultados que se generen en la búsqueda se mostrarán todas las canciones y todos los artistas que hayan aparecido como resultado en esta página.

Como podemos ver en la Figura 5.3.6.5, dentro de la sección de canciones encontraremos todas las canciones asociadas a la búsqueda junto con la imagen del álbum al que pertenece, nombre de la canción y nombre del álbum. Si se hace clic en alguna de las canciones que aparecieron como resultado, el usuario será redirigido al Reproductor Web de Spotify y reproducirá automáticamente la canción seleccionada.

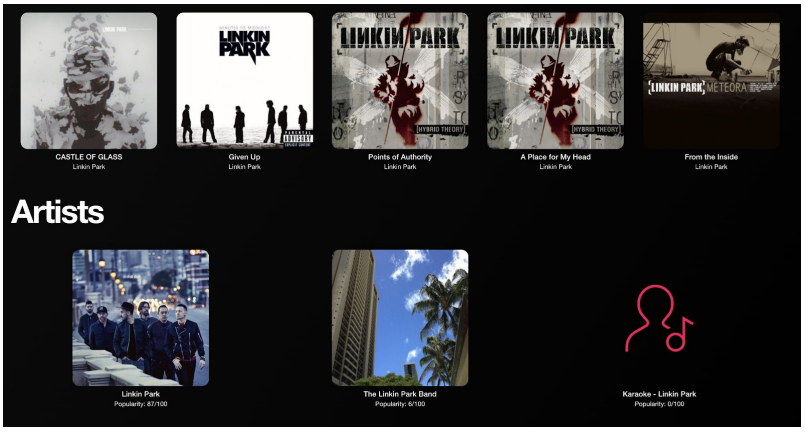


Figura 5.3.6.6: Sección de Artistas de la aplicación experimental.

Como podemos ver en la Figura 5.3.6.6, dentro de la sección de artistas encontraremos todas los artistas asociados a la búsqueda junto con una imagen del artista, nombre del artista y popularidad. Si se hace clic en alguna de las imágenes de los artistas que aparecieron en el resultado, el usuario será redirigido a la Página del Artista descrita a continuación.

Página del Artista

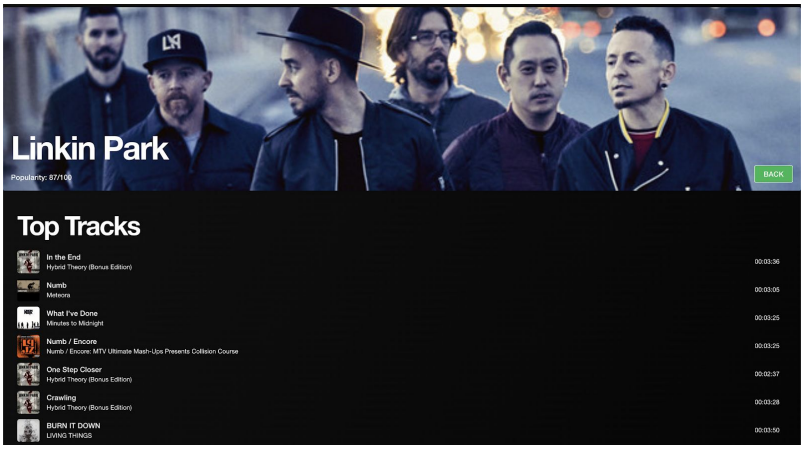


Figura 5.3.6.7: Página del Artista de la aplicación experimental.

Como podemos ver en la Figura 5.3.6.7, dentro de la página del Artista se puede ver un banner del artista, junto con su nombre, popularidad y un botón para volver a la página anterior. A su vez se pueden ver la sección de las canciones más reproducidas del artista, el álbum al cual pertenece la canción y la duración de la canción. Si se hace clic en alguna de las canciones, el usuario será redirigido al Reproductor Web de Spotify y reproducirá automáticamente la canción seleccionada.

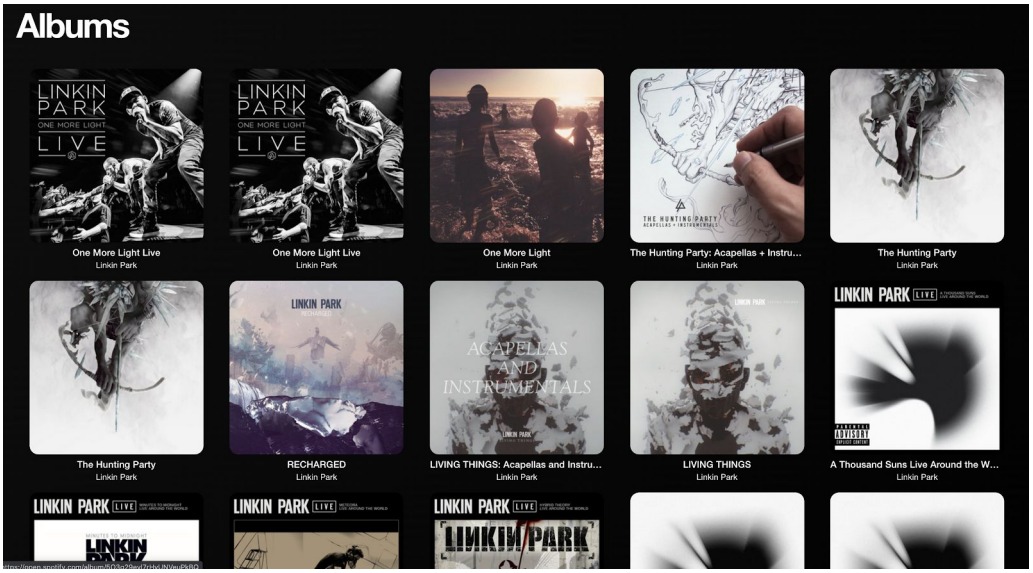


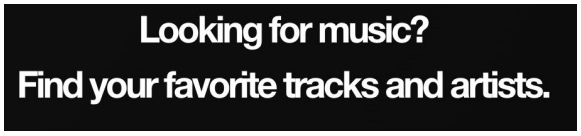
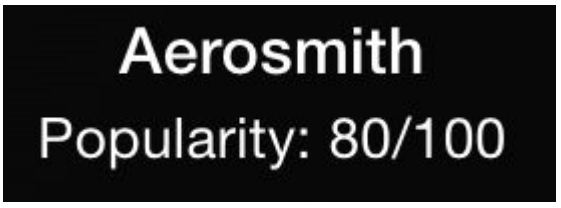
Figura 5.3.6.8: Sección de álbumes dentro de la página del artista de la aplicación experimental.

Dentro de la página del artista también podemos divisar la sección de los álbumes del artista, junto con su nombre y el nombre del artista como podemos ver en la Figura 5.3.6.8. Si se hace clic en alguno de los álbumes, el usuario será redirigido al Reproductor Web de Spotify con el álbum seleccionado listo para reproducir.

A su vez, incluímos dentro del sistema la integración que se especifica en la sección Especificación de plugin a entregar del Capítulo 3. Por lo tanto, la aplicación posee un tema de Material-UI que se conecta a un archivo de la plataforma Figma, y dentro de este archivo se encuentra el kit UI utilizado en la aplicación.

Mediante el uso de las especificaciones y convenciones mencionadas, el desarrollador puede incluir los componentes que estén diseñados dentro del archivo de Figma, así como también componentes nuevos que el diseñador cree.

En la siguiente tabla podemos ver la comparación de los componentes visuales en la aplicación y su correspondiente diseño dentro del archivo de Figma:

Componentes visuales en la aplicación	Componentes visuales en Figma
	
	
	
	

5.3.7 Problemas a resolver

El experimento consistió en plantear tareas a resolver dentro de esta aplicación, estas tareas consistieron en:

- Agregar funcionalidades nuevas.
- Agregar nuevos componentes visuales.
- Modificar componentes visuales existentes.
- Agregar un componente complejo que utilice componentes simples de Figma.

De esta forma pudimos obtener datos y métricas que nos ayudaron a deducir si esta integración provee mejoras en el desarrollo de la UI de un sistema. Estas tareas a resolver simulan la solicitud del cliente dueño de la aplicación que necesita agregar nuevas funcionalidades y agregar cambios visuales para aumentar el flujo de visitas y uso de la página:

1. Cada canción que aparezca dentro del resultado de una búsqueda o dentro de las canciones más escuchadas en la página del artista, debe tener un botón para “Agregar a Lista de Favoritos”. Al cliquear este botón, la canción será agregada en una “Lista de Favoritos”. Si la canción ya estaba agregada, el botón será para “Quitar de la Lista de Favoritos”.

2. Cuando se haga clic en el botón “Agregar a Lista de Favoritos” o en el botón “Quitar de la Lista de Favoritos”, debe aparecer un componente Modal con un mensaje solicitando la confirmación de la acción. El Modal tendrá un botón de “Cancelar” y de “Aceptar” que cancelará la acción o le dará curso respectivamente.

3. Crear la página “Lista de Favoritos”. Aquí aparecerán todas las canciones que fueron agregadas por el usuario (reutilizar el componente de Top Tracks). El usuario debe poder hacer clic en sus canciones de la lista, y ser redirigido a la página de Spotify para que se reproduzca la canción. A su vez cada canción tendrá el botón “Quitar de la Lista de Favoritos”.

4. Cuando la canción sea agregada o quitada de la Lista de Favoritos, debe aparecer un componente Alert que notifique que se ha agregado o quitado la canción.

5. Modificar los botones, tipografías y paleta de colores de la aplicación para darle otra identidad a ella (cambio de branding).

Para realizar estas tareas, el equipo A utilizará la integración con Figma propuesta en este trabajo y el equipo B utilizará el enfoque convencional de desarrollo (sin integración).

5.3.8 Datos a recopilar, métricas y SUS

Cada equipo realizó las tareas propuestas y por cada equipo se recopilaron los siguientes datos utilizados en un posterior análisis:

- Cantidad de tareas creadas.
- Estimación de cada tarea en puntos.
- Estimación total de todas las tareas en puntos.
- Cantidad de tareas asignadas al desarrollador.
- Cantidad de tareas asignadas al diseñador.
- Cantidad de líneas de código agregadas por tarea.
- Cantidad de líneas de código borradas por tarea.
- Cantidad de líneas de código resultantes por tarea.
- Cantidad de líneas de código agregadas en total.
- Cantidad de líneas de código borradas en total.
- Cantidad de líneas de código resultantes en total.
- Cantidad de revisiones de código.
- Cantidad de solicitudes de cambio en revisión de código.

- Cantidad de deploys a ambiente de Staging.
- Cantidad de deploys a ambiente de Producción.
- Cantidad de regresiones en testeo de ambiente de Staging.
- Cantidad de regresiones en testeo de ambiente de Producción.
- Tiempo que estuvo cada tarea en el estado “En Progreso”.
- Tiempo que estuvo cada tarea en el estado “En Revisión”.
- Tiempo que estuvo cada tarea en el estado “En Testeo”.
- Tiempo que le llevó a cada tarea pasar del estado “En Progreso” al estado “Finalizado”.

Comparar los resultados de estas métricas nos permitió generar conclusiones en base a si el uso del enfoque propuesto mediante la integración con Figma provee ventajas en el desarrollo de una aplicación sobre el enfoque convencional o no.

A su vez utilizamos el Sistema de Escalas de Usabilidad, también conocido como Escala de Usabilidad de un Sistema (EUS) o simplemente SUS (System Usability Scale) para medir la usabilidad de la integración entre la aplicación y Figma [Lewis09].

Diferentes pruebas y tests han demostrado que los resultados obtenidos a partir del uso de esta escala suelen ser muy confiables y acertados [Bangor08], razón por la cual es uno de los métodos de medición de usabilidad más utilizados en Experiencia de Usuario.

La escala en sí consiste en 10 preguntas, cada una de las cuales puede ser puntuada de 1 a 5, donde 1 significa Total desacuerdo y 5 significa Total acuerdo.

Para obtener los resultados, se suman los resultados promediados obtenidos de los cuestionarios realizados a los integrantes del equipo que utilizó la integración, considerando lo siguiente: las preguntas impares (1,3,5,7 y 9) tomarán el valor asignado por el usuario, y se le restará 1. Para las preguntas pares (2,4,6,8,10), será de 5 menos el valor asignado por nuestros entrevistados.

Una vez obtenido el número final, se lo multiplica por 2,5 .

Para ejemplificar, supongamos que los resultados obtenidos en las preguntas, en orden, fueron:

3, 4, 3, 5, 3, 2, 1, 2, 4, 5.

Asignamos los nuevos valores según el algoritmo de SUS, y nuestros nuevos valores serán:

((3-1)+(5-4)+(3-1)+(5-5)+(3-1)+(5-2)+(1-1)+(5-2)+(4-1)+(5-5))*2,5

simplificando:

(2+1+2+0+2+3+0+3+3+0)*2,5

16*2.5=40

Entonces el puntaje SUS es 40. La puntuación máxima es 100, un puntaje SUS superior a 68 se considera por encima del promedio y cualquier resultado por debajo de 68 está por debajo del promedio. Este resultado nos indica que es menos que el promedio y debemos trabajar mucho en él.

5.4 Análisis de los resultados obtenidos

En este experimento participaron 5 personas: 2 desarrolladores para cada equipo y una diseñadora que ocupó ese rol en ambos equipos.

Equipo A

- Juan Machado
    - Edad: 24 años.
    - Puesto actual: Desarrollador de software SR.
    - Experiencia: 4 años.
    - Tecnologías que usa actualmente: React, Angular, Serverless, JAVA, C#.
- María Guillermina Vescovo
    - Edad: 27 años.
    - Puesto actual: Desarrolladora de software SR.
    - Experiencia: 4 años.
    - Tecnologías que usa actualmente: Ruby on Rails, React/Tailwind + Stimulus, Docker, Postgresql, Mysql.
- Sofía Fernández Gavio
    - Edad: 29 años.
    - Puesto actual: Diseñadora de productos digitales (UX/UI) de manera independiente.
    - Experiencia: 8 años.
    - Plataformas que usa actualmente: Sketch, Studio by Invision, Invision, Illustrator, entre otras.

Equipo B

- Matías Pérez
    - Edad: 30 años.
    - Puesto actual: Desarrollador de software SR.
    - Experiencia: 6 años.
    - Tecnologías que usa actualmente: React, Django y NodeJS.
- Alan Toris
    - Edad: 29 años.
    - Puesto actual: Desarrollador de software SR.
    - Experiencia: 4 años.

- Tecnologías que usa actualmente: Python/Django, React, Docker, Google Cloud.

Nosotros, los autores de este informe, también colaboraron en el experimento pero sólo para la toma de métricas y para dar soporte a los equipos en cuanto a dudas técnicas relacionadas al código ya escrito del sistema experimental, dudas de requerimientos y dudas en cuanto al proceso de desarrollo.

Previo al comienzo del experimento se coordinó una videollamada con cada equipo con el objetivo de hacer un recorrido por la aplicación experimental y su código para conocer los componentes más importantes y cómo interactúan. Con el equipo que utilizó la integración de Figma también se profundizó en cómo funciona la integración, cuales eran los componentes principales, cómo acceder a la guía de estilos e interactuar con ella.

Luego de haber realizado el experimento obtuvimos un amplio conjunto de métricas que analizaremos a continuación. Dentro de estas métricas encontramos las generales, las relacionadas a líneas de código y las relacionadas a tiempos de desarrollo. Sólo analizaremos en esta sección las métricas y gráficos que en comparación resultan las más importantes. Todas las métricas y gráficos se encuentran en la sección Anexo de métricas y gráficos en caso de querer ver todos los datos recolectados.

Métricas Generales

Medición	Team-A	Team-B
Tareas creadas	12	15
Estimación	28	36
Tareas asignadas al desarrollador	9	15
Tareas asignadas al diseñador	3	0
Revisiones de código	9	15
Deploys a ambiente de Staging	9	15
Deploys a ambiente de Producción	3	4
Regresiones en testeo de ambiente de Staging	0	2
Regresiones en testeo de ambiente de Producción	0	0

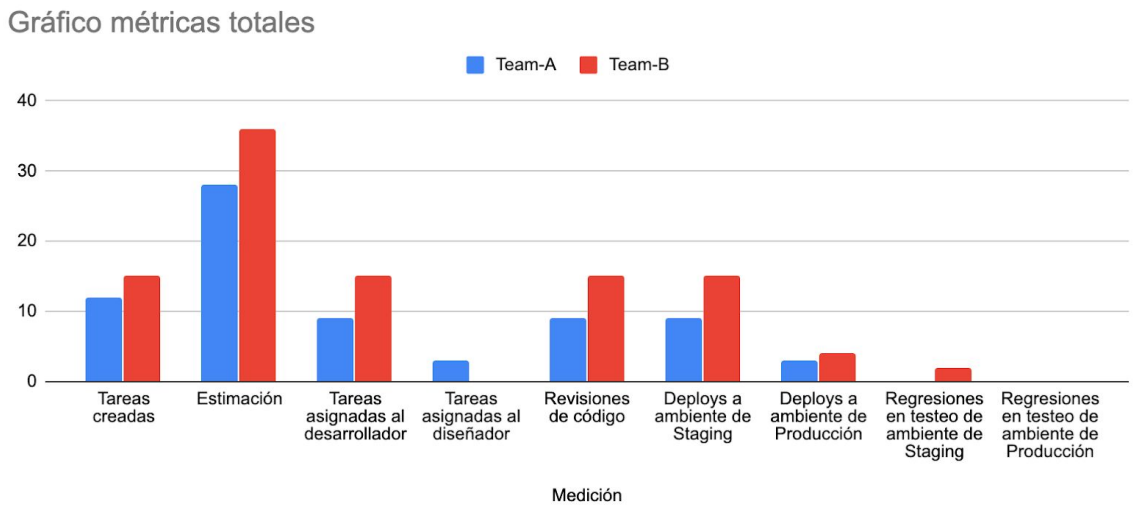


Figura 5.4.1: Gráfico comparativo de métricas totales.

Como podemos ver en la Figura 5.4.1, el equipo A creó menos tareas que el equipo B. Esto se debe a que el equipo B cometió dos errores de los cuales tuvo que generar dos tareas nuevas de categoría “error/bug” para resolverlos. A su vez, para la construcción de la “Página de Favoritos” el equipo A creó una sola tarea mientras que el equipo B creó dos tareas para resolver la funcionalidad.

En cuanto a las estimaciones, el equipo B consideró que la dificultad de las tareas en su totalidad era de 36 puntos mientras que el equipo A consideró que la dificultad total era de 28 puntos. Esto se debe a que en el equipo A, algunas tareas iban a ser realizadas por el diseñador dentro de Figma las cuales suponían poco trabajo en comparación a hacer las mismas tareas usando el enfoque convencional.

La distribución de tareas en el equipo A fueron 9 para los desarrolladores y 3 para el diseñador ya que éste debía encargarse de los cambios puramente visuales, mientras que en el equipo B las 15 tareas creadas fueron destinadas a los desarrolladores.

La cantidad de despliegues realizados a Staging por el equipo A son menores a las del equipo B ya que no sólo crearon menos tareas para trabajar, sino que las tareas que se hacían desde la plataforma Figma no requerían despliegues. Lo mismo ocurrió con los despliegues en el ambiente de Producción, el equipo A hizo un despliegue menos. Esto terminó impactando en los tiempos de desarrollo como veremos luego.

En cuanto a regresiones en los ambientes de desarrollo, el equipo B tuvo 2 regresiones pero no estuvieron asociadas a problemas de diseño o a la falta de integración con Figma, sino a equivocaciones de funcionalidad comunes y que surgieron de manera casual. No hubo ninguna regresión en los ambientes de producción.

Métricas relacionadas a cantidad de líneas de código

Medición	Team-A	Team-B
Líneas de código agregadas	720	802
Líneas de código borradas	145	209
Líneas de código resultantes	575	593
Líneas de código modificadas	865	1011

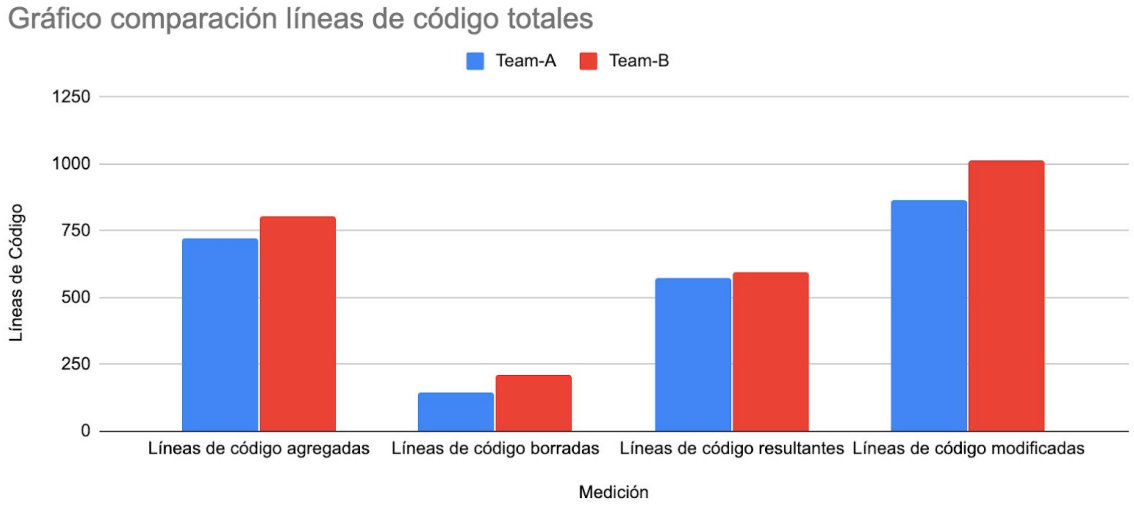


Figura 5.4.2: Gráfico comparativo de líneas de código totales.

Como podemos ver en la Figura 5.4.2, y contrario a lo que se esperaba en un principio, la cantidad de líneas trabajadas por cada equipo fueron muy similares. El equipo A agregó y borró una cantidad de líneas apenas menor a las que agregó y borró el equipo B. La cantidad de líneas resultantes terminaron siendo muy similares en consecuencia. Sí podemos resaltar que la cantidad de líneas con las que tuvo que interactuar el equipo A fueron menores que las del equipo B en un total de 146 líneas menos.

Dicho esto, podemos afirmar que la diferencia en cantidad de código escrito es casi insignificante entre los dos equipos ya que las tareas no eran muy diferentes entre sí y requerían modificaciones muy parecidas y en los mismos lugares del código. Por lo tanto, la integración con Figma no resultó en una ventaja importante en este aspecto dentro del experimento.

También cabe destacar que como muestra el siguiente gráfico, el equipo B cometió dos errores o bugs en el experimento, lo cual también terminó impactando en la cantidad de líneas agregadas, borradas, resultantes y modificadas. En caso que estos errores no

hubiesen ocurrido, la diferencia entre las líneas de cada equipo hubiese sido aún menor, reforzando la idea que la integración con Figma no impactó en la cantidad de líneas de código.

Métricas relacionadas a cantidad de líneas de código modificadas

Tarea	Team-A	Team-B
Agregar a Favoritos	541	485
Quitar de Favoritos	202	235
Página Favoritos	122	352
Errores	0	38
Cambiar identidad visual	0	70

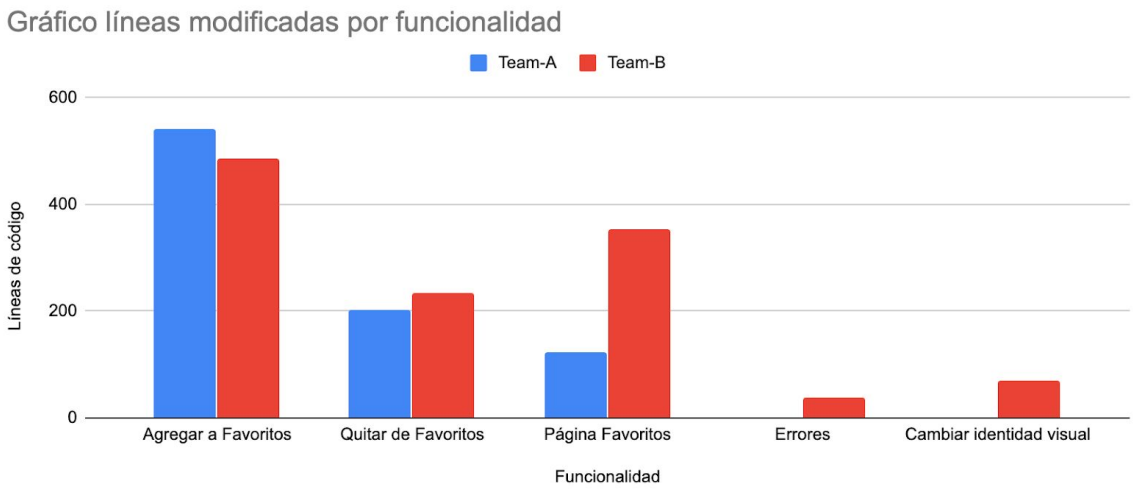


Figura 5.4.3: Gráfico comparativo de líneas de código modificadas por funcionalidad.

Como vemos en la Figura 5.4.3, en cuanto a las líneas de código modificadas o trabajadas por cada equipo, podemos destacar que los cambios que requirieron pura interacción del diseñador con Figma no requirió modificación de ninguna línea (equipo A). Esto supuso un ahorro de tiempo en cuanto al trabajo y proceso de desarrollo del que depende una tarea para ser desplegada, ya que en este caso no existió ningún despliegue.

Métricas relacionadas a tiempo en estado “En Progreso” por cada funcionalidad

Funcionalidad	Team-A	Team-B
Agregar a Favoritos	174	289
Quitar de Favoritos	62	77
Página Favoritos	49	129
Cambiar identidad visual	11	59

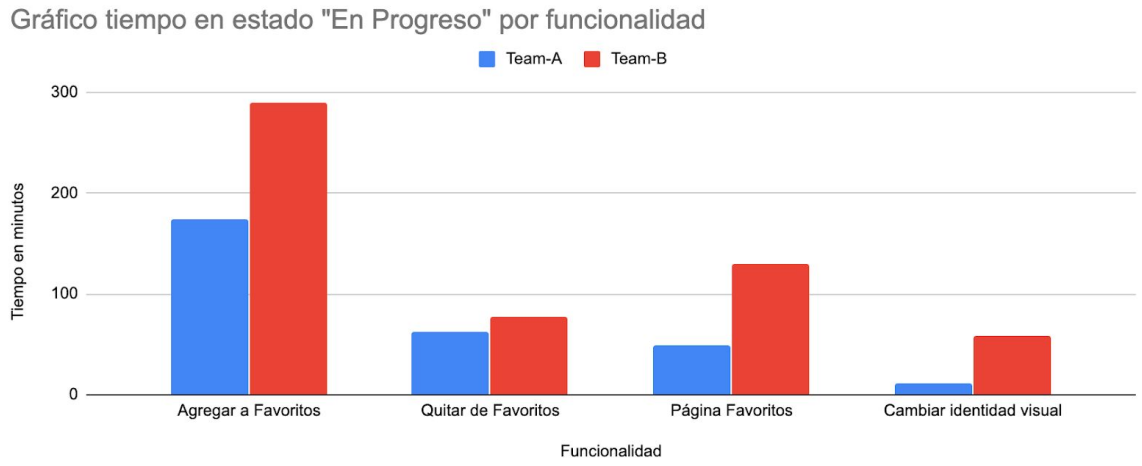


Figura 5.4.4: Gráfico comparativo de tiempo en estado “En Progreso” por funcionalidad.

Como podemos ver en la Figura 5.4.4, el tiempo que estuvo cada funcionalidad en el estado “En Progreso” varió entre equipos. El equipo A notoriamente tardó menos tiempo en terminar las funcionalidades que el equipo B. Habiendo presenciado y observado el experimento, pudimos deducir que esto se debió a dos razones importantes: la integración con Figma y la forma de programar de cada equipo.

Por un lado, la integración con Figma favoreció al equipo A ya que cada componente visual nuevo que se debía agregar en cada tarea, ya estaba listo para usar dentro de la guía de estilos. Por lo tanto, para agregar el componente en cada vista donde se lo requería, sólo se debía copiar el código especificado dentro de la guía de estilos para el componente deseado y copiarlo en el archivo correspondiente. No fue así para el equipo B, quien debía buscar los estilos del componente deseado en la plataforma Figma y desarrollarlo desde cero.

El segundo elemento que también afectó a esta métrica fue la manera de programar de cada equipo y esto quizás sea una opinión subjetiva y no muy tangible. El equipo A se lo notó más rápido y con falta de obstáculos al momento de programar cada funcionalidad,

mientras que el equipo B era más cauteloso y pensativo en cada tarea. Lo cual creemos que esto también influyó directamente en los tiempos de cada equipo.

Cabe destacar también que el equipo B cometió dos errores/bugs al realizar la funcionalidad “Página Favoritos”, por lo que el tiempo demorado en estado “En Progreso” para esa funcionalidad es la sumatoria de lo que tardaron las subtareas creadas más lo que tardaron en resolverse los errores.

Métricas relacionadas a tiempo en estado “En Revisión” por cada funcionalidad

Funcionalidad	Team-A	Team-B
Agregar a Favoritos	26	45
Quitar de Favoritos	16	36
Página Favoritos	7	19
Cambiar identidad visual	0	7

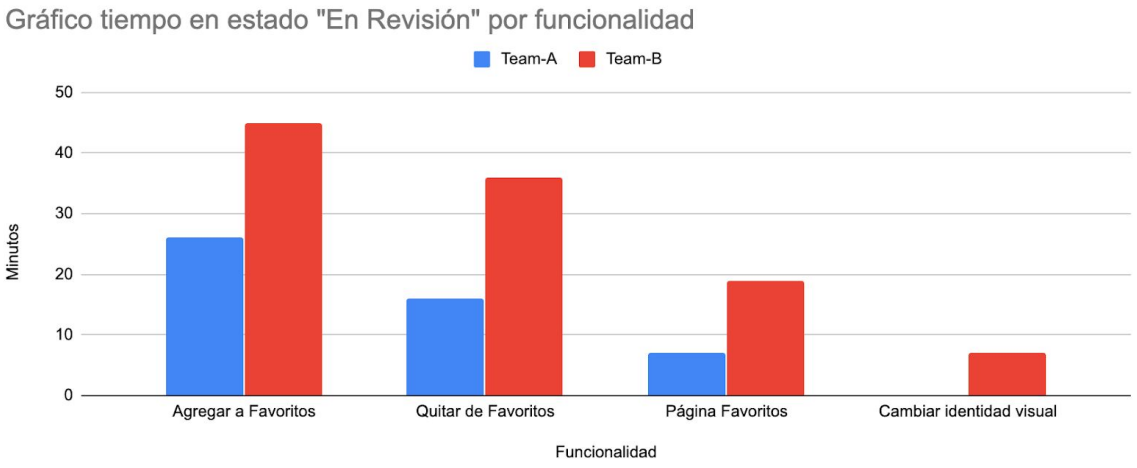


Figura 5.4.5: Gráfico comparativo de tiempo en estado “En Revisión” por funcionalidad.

Como podemos ver en la Figura 5.4.5, los tiempos de revisión también variaron entre equipos. El equipo A fue más rápido para revisar el código de todas sus tareas que el equipo B. Como podemos ver en los gráficos anteriores, la cantidad de líneas de código a revisar eran casi las mismas por lo que no atribuimos esos valores al resultado de esta métrica. Sí podemos afirmar nuevamente, que el equipo A se tomaba menos tiempo para revisar el código tanto en la etapa de “En Progreso” como en la etapa de “En Revisión”.

Nuevamente aclaramos, que el equipo B cometió dos errores/bugs al realizar la funcionalidad “Página Favoritos”, por lo que el tiempo demorado en estado “En Revisión”

para esa funcionalidad también es la sumatoria de lo que tardaron las subtareas creadas más lo que tardaron en resolverse los errores.

Métricas relacionadas a tiempo en estado “En Testeo” por cada funcionalidad

Antes de explicar los datos que obtuvimos relacionados a los tiempos en estado “En Testeo”, es necesario detallar la forma en la que ambos equipos trabajaron desde que una tarea llegaba al estado “En Testeo” hasta que terminaba en el estado “Finalizado”.

Cada equipo tomó cada funcionalidad que debía desarrollarse y la dividió en subtareas que luego fueron escritas en tickets de Trello. Por ejemplo, para la funcionalidad “Agregar a Favoritos” ambos equipos crearon 4 subtareas que se tradujeron en 4 tickets de Trello:

- Botón Agregar a Favoritos.
- Modal Agregar a Favoritos.
- Agregar a Favoritos.
- Alert Agregar a Favoritos.

Las especificaciones y detalles puntuales de cada tarea varió entre cada equipo pero todas englobaron los mismos requerimientos.

Ahora bien, para que una subtarea pueda pasar del estado “En Testeo” al estado “Finalizado” no sólo debía realizarse el proceso de QA, sino que una vez realizado se debía esperar a que las restantes tareas de la funcionalidad lleguen al estado “En Testeo” y también pasen el proceso de QA. Esto se debe a que el despliegue de una subtarea por sí misma sin las otras complementarias, resultaría en una fracción de la funcionalidad en el ambiente de Producción. Por ejemplo, tendríamos un botón o un modal sin ninguna acción en el ambiente donde interactúa el usuario final. Y como el experimento trata de acercarse lo más posible a la realidad decidimos tomar el camino que se toma en la industria de software: hacer despliegues sólo de funcionalidades completas dentro del ambiente de producción.

Por lo tanto, cada tarea debía esperar a las restantes de la funcionalidad para poder pasar al estado “Finalizado”, lo que supone que la primera tarea realizada de la funcionalidad es la que más tiempo estuvo en el estado “En Testeo”.

Ahora bien, si tuviésemos por ejemplo:

- La tarea 1 estuvo 30 minutos “En Testeo”.
- La tarea 2 estuvo 15 minutos “En Testeo”.
- La tarea 3 estuvo 10 minutos “En Testeo”.
- La tarea 4 estuvo 5 minutos “En Testeo”.

No sería real decir que la funcionalidad que engloba estas 4 tareas estuvo 30+15+10+5 = 60 minutos en el estado “En Testeo”. Sí sería más acertado tomar el tiempo “En Testeo” de

la primera tarea que fue la que realmente estuvo más tiempo en ese estado. Otra opción válida sería utilizar el tiempo promedio que estuvo la funcionalidad en el estado “En Testeo”.

Los siguientes gráficos mostrarán la comparación entre ambos equipos del tiempo que estuvo cada funcionalidad “En Testeo” utilizando estos dos enfoques.

Funcionalidad	Team-A	Team-B
Agregar a Favoritos	159	332
Quitar de Favoritos	74	92
Página Favoritos	13	172
Cambiar identidad visual	1	47

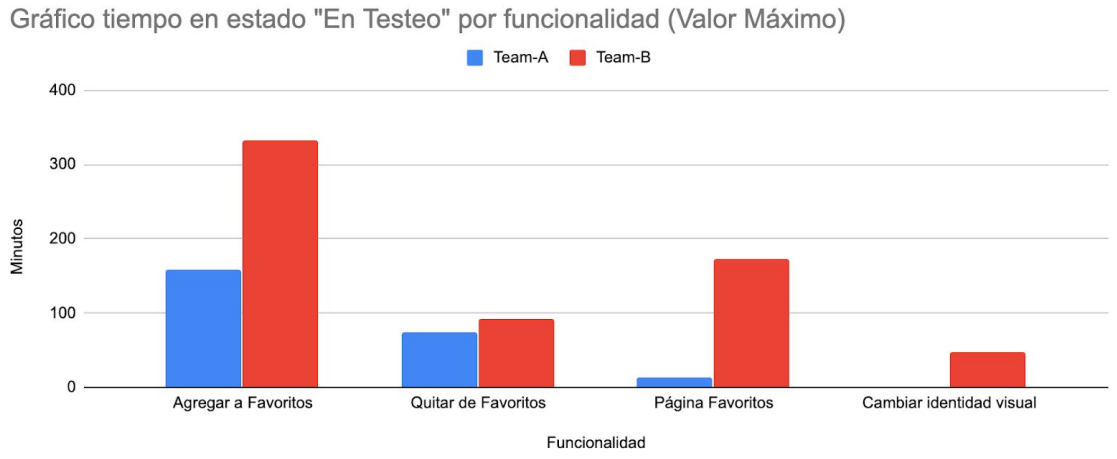


Figura 5.4.6: Gráfico comparativo de tiempo en estado “En Testeo” por funcionalidad utilizando enfoque de valor máximo.

Funcionalidad	Team-A	Team-B
Agregar a Favoritos	75.75	170.25
Quitar de Favoritos	40	50.75
Página Favoritos	13	89
Cambiar identidad visual	1	26.3

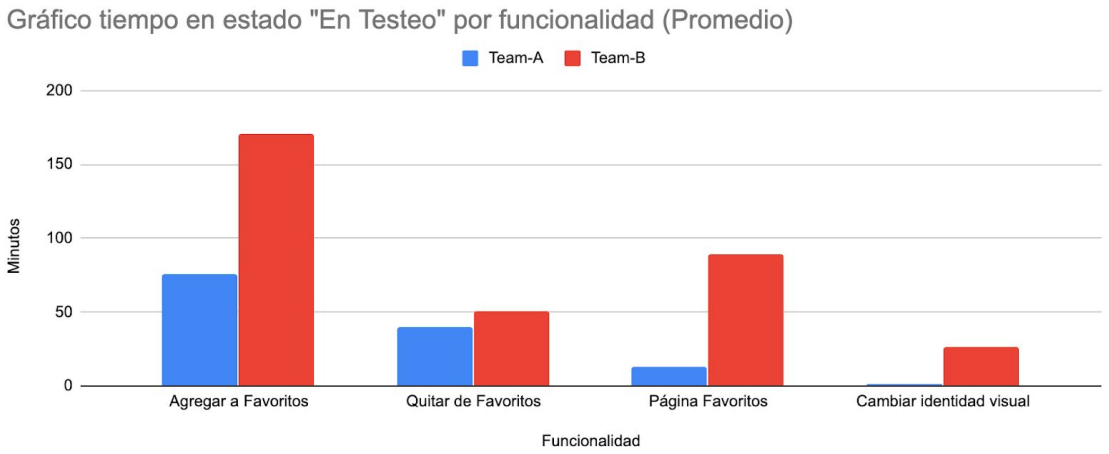


Figura 5.4.7: Gráfico comparativo de tiempo en estado “En Testeo” por funcionalidad utilizando enfoque de promedio.

Como podemos ver en la Figura 5.4.6 y la Figura 5.4.7, el tiempo en estado “En Testeo” fue menor en el equipo A utilizando ambos enfoques para la comparación. Esto se debe a que los tiempos que cada funcionalidad estuvo en el estado “En Progreso” y “En Revisión” en el equipo A fueron siempre menores que el equipo B.

No podemos asociar el tiempo que se utilizó pura y exclusivamente para el QA de cada tarea a esta diferencia, ya que este tiempo fue muy constante en cada equipo: ninguno tardó más de 5 minutos en hacer el proceso de QA en cada tarea.

Nuevamente recordamos que el equipo B cometió dos errores al realizar la funcionalidad “Página Favoritos”, lo cual también resultó en una demora de tiempo más amplia que la del equipo A.

Métricas relacionadas a tiempos totales

Tarea	Team-A	Team-B
Botón Agregar a Favoritos	211	410
Modal Agregar a Favoritos	159	299
Agregar a Favoritos	93	206
Alert Agregar a Favoritos	40	100
Botón Quitar de Favoritos	93	138
Modal Quitar de Favoritos	73	89
Quitar de Favoritos	53	61
Alert Quitar de Favoritos	19	28

Página Favoritos	69	187 + 170
Error 1	0	80
Error 2	0	67
Cambiar Botones	7	71
Cambiar Tipografías	5	48
Cambiar Colores	2	26

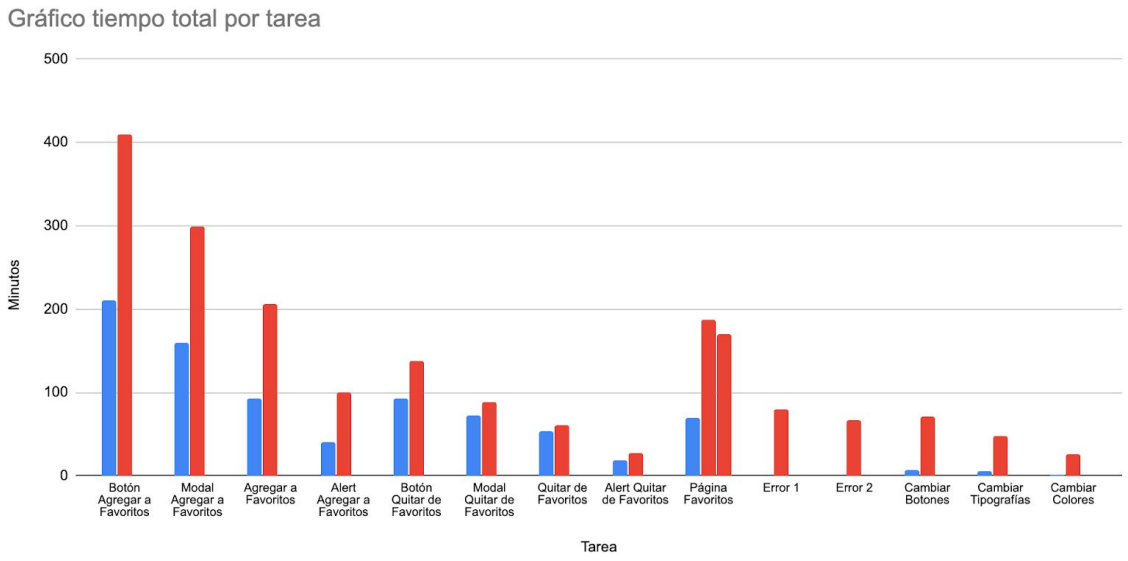


Figura 5.4.8: Gráfico comparativo de tiempo total por tarea.

Como podemos ver en la Figura 5.4.8, el tiempo que demoró cada tarea en pasar del estado “En Progreso” al estado “Finalizado” siempre fue menor en el equipo A y por amplia diferencia. Esto no es más que una sumatoria de los tiempos previamente analizados, y quizás, hay una diferencia bastante abultada en cada tarea ya que cada tiempo total tiene en cuenta el tiempo en que estuvo la tarea en el estado “En Testeo”. Como se explicó anteriormente, este tiempo se acumula en base a lo que tarden las tareas restantes de la funcionalidad. No es quizás un análisis muy justo pero son tiempos reales que pasan en la industria de software y tienen sentido al analizar las tareas por separado.

Funcionalidad	Team-A	Team-B
Agregar a Favoritos	359	666
Quitar de Favoritos	152	205
Página Favoritos	69	320
Cambiar identidad visual	12	113

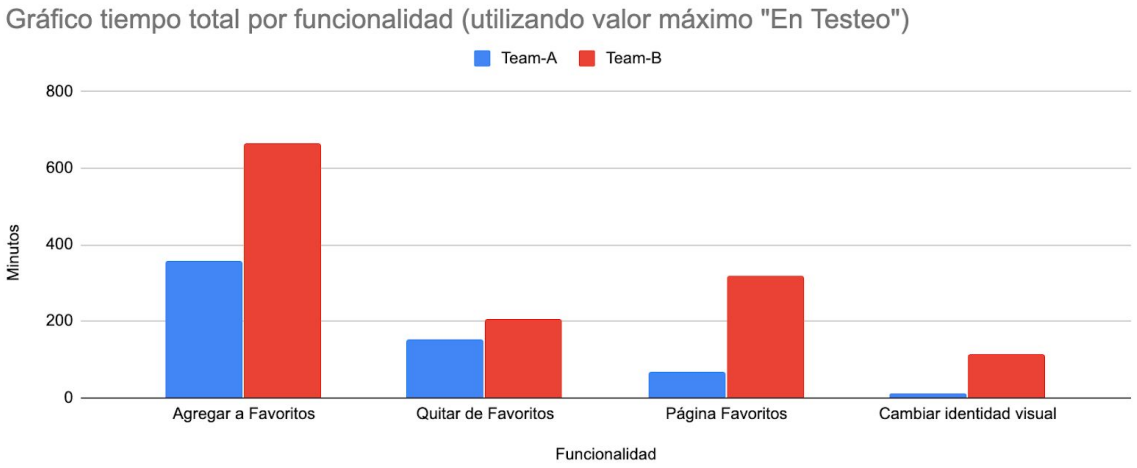


Figura 5.4.9: Gráfico comparativo de tiempo total por funcionalidad utilizando el enfoque de valor máximo “En Testeo”.

Funcionalidad	Team-A	Team-B
Agregar a Favoritos	275.75	504.25
Quitar de Favoritos	118	163.75
Página Favoritos	69	237
Cambiar identidad visual	12	92.3

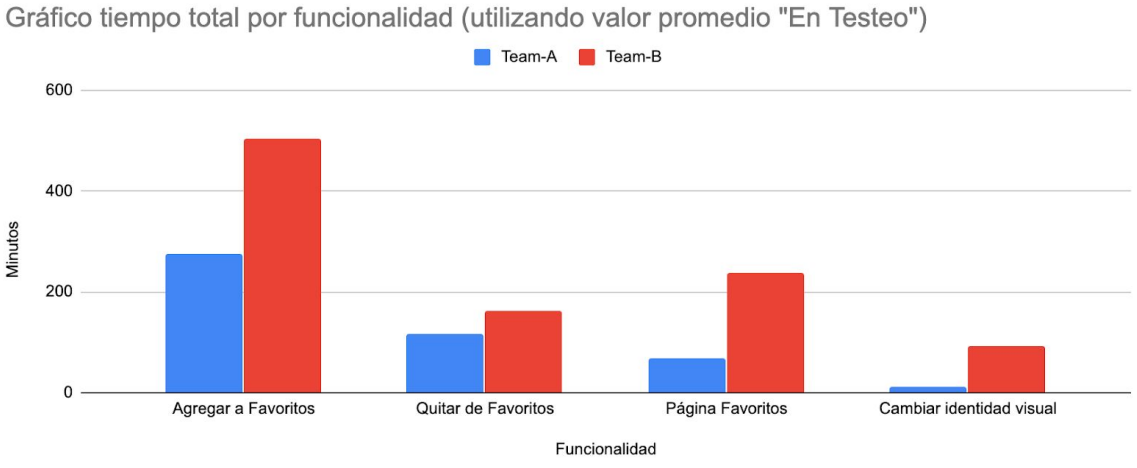


Figura 5.4.10: Gráfico comparativo de tiempo total por funcionalidad utilizando el enfoque de valor promedio “En Testeo”.

Como podemos ver en la Figura 5.4.9 y la Figura 5.4.10, utilizando los enfoques de valor máximo y valor promedio al hacer el análisis de tiempos totales por funcionalidad no hay mucha diferencia en comparación a los gráficos ya analizados.

Cabe destacar que la diferencia de los tiempos en relación a las tareas de identidad visual (que el equipo A sólo necesitó utilizar Figma para impactar los cambios) es una de las más marcadas y creemos que a mayor escala de cambios el resultado seguiría siendo igual de amplio.

Resultados del SUS

A continuación se encuentran los resultados y anotaciones que proporcionó el equipo A sobre la integración utilizada con Figma.

Fecha: 18/08/2020
Nombre y Apellido: Sofía Fernández Gavio.
Edad: 29 años.
Profesión: Diseñadora de productos digitales (UX/UI) de manera independiente.

(5-1)+(5-1)+(5-1)+(5-3)+(4-1)+(5-2)+(4-1)+(5-1)+(4-1)+(5-4)
(4+4+4+2+3+3+3+4+3+1)*2,5 = 77.5

Comentarios: Probando los cambios en el archivo de Staging podía ver los cambios reflejados al instante dentro del ambiente. Pero en el ambiente de Producción, un usuario podría entrar y ver los cambios a medias. Estaría bueno poder tener algún tipo de funcionalidad que permita no publicar cambios sin mi consentimiento previo.

Fecha: 19/08/2020
Nombre y Apellido: Juan Machado.
Edad: 24 años.
Profesión: Desarrollador de software SR.

(3-1)+(5-1)+(4-1)+(5-2)+(5-1)+(5-2)+(5-1)+(5-1)+(4-1)+(5-3)
(2+4+3+3+4+3+4+4+3+2)*2,5 = 80

Comentarios: La guía de estilos fue fundamental en el uso de la integración, sin ella hubiese sido muy difícil crear los componentes. El ramp-up para explicar eso era muy necesario. Para darle mi voto de confianza me gustaría que la integración tenga más componentes integrados para resolver problemas más variados.

Fecha: 19/08/2020
Nombre y Apellido: María Guillermina Vescovo.
Edad: 27 años.
Profesión: Desarrolladora de software SR.

(4-1)+(5-1)+(5-1)+(5-1)+(5-1)+(5-1)+(4-1)+(5-1)+(4-1)+(5-2)
(3+4+4+4+4+4+3+4+3+3)*2,5 = 90

Comentarios: Al tener los diseños integrados en la guía no necesitamos coordinarnos con la diseñadora y ahorramos tiempo en eso. Los despliegues automáticos fueron muy útiles porque era un problema menos para preocuparse. Me gustaría que la integración siga completándose.

Como podemos ver en los resultados del formulario SUS, todos los puntajes estuvieron arriba del promedio por lo que creemos que la usabilidad de esta integración fue muy buena.

A su vez nos encontramos con una buena sugerencia por parte de la diseñadora, en la que nos indicó que se podría buscar una forma de impactar los cambios de manera grupal y que se pida consentimiento al menos en el ambiente de Producción. Esto se debe a que en el proceso de cambio de componentes visuales, si los cambios no se aplican en bloque, un usuario final podría ver el trabajo en progreso del diseñador.

El equipo también recalcó que la guía de estilos fue esencial para que la integración sea fácil de usar y que se ahorró mucho tiempo en ese sentido ya que no se dependía de Figma o de la diseñadora para consultar los estilos.

Por último, ambos desarrolladores propusieron la ampliación de la integración a más componentes visuales para abarcar más posibilidades a la hora de trabajar en un proyecto real.

5.5 Conclusiones

Primero y antes de adentrarnos en conclusiones técnicas, debemos hacer una mención especial a los chicos que se comprometieron a realizar este experimento. Conseguir voluntarios para un objetivo enfocado a la investigación en la educación pública es algo muy difícil de conseguir y ayudó a que los datos obtenidos fueran mucho más acercados a la realidad de la industria.

A su vez, queremos recalcar que por más que hayamos contado con estas personas para realizar el experimento, se debería realizar un experimento más amplio y con mayor cantidad de personas para que los resultados sean más confiables. Creemos que las

diferencias personales a la hora de analizar y resolver los problemas, programar las soluciones y revisar el código escrito entre ambos equipos tuvo incidencia en las métricas obtenidas. Cada individuo tiene su manera de pensar y trabajar lo cual genera que, al tener una entrada de datos de sólo 5 personas, los datos obtenidos estén condicionados por las cualidades personales que cada uno posea.

Otro factor importante para señalar antes de elaborar conclusiones, es que el equipo B cometió dos errores casuales y no relacionados a la falta de integración con Figma. Esto terminó impactando, en menor medida, en la cantidad de líneas de código escritas y los tiempos de desarrollo totales. Si estos errores no hubieran ocurrido o si los hubiese cometido el otro equipo, la brecha entre ambos equipos se habría acortado. No obstante, esta variación no hubiese cambiado las conclusiones que mencionaremos, pero sí es justo mencionarlo.

Creemos que estos dos análisis son importantes para quizás hacer una crítica más honesta en cuanto a los datos que obtuvimos.

Ahora bien, en cuanto a las conclusiones técnicas que podemos sacar en base a los datos obtenidos observando el experimento y las respuestas del formulario SUS, podemos hacer hincapié en 3 puntos muy importantes.

1. Cantidad de líneas de código similar

En primer lugar, contrario a lo que se pensaba en un principio antes de realizar el experimento, es que la cantidad de líneas de código escritas no varió de manera significativa entre ambos equipos.

Se puede aclarar que las tareas a realizar eran las mismas para ambos equipos, requerían modificaciones no muy complejas y la cantidad de elementos visuales involucrados no era muy grande. Por lo tanto, el código necesario para resolver la funcionalidad de cada tarea era muy parecido entre ambos equipos. Las mayores diferencias se encontraron a la hora de cómo insertar nuevos componentes visuales en donde quedó favorecido el equipo con la integración en Figma.

También queremos recalcar que la cantidad de líneas modificadas (suma de líneas agregadas más líneas modificadas) totales fue mayor en el equipo B, lo que nos da un indicio que el equipo B tuvo más interacción con el código que el equipo A.

Lo que sí podemos afirmar con seguridad, es que el equipo A no tuvo que agregar ni quitar ninguna línea del código para poder impactar cambios puramente visuales ya que de eso se encargó la diseñadora del equipo utilizando la plataforma Figma. Esto no queda atado a las cualidades personales del equipo, esto será siempre así para cualquier cambio puramente visual reduciendo el trabajo del desarrollador.

2. Ahorro de tiempo para el equipo con integración a Figma

En segundo lugar, y quizás el más importante, el tiempo que estuvo cada tarea en el estado “En Progreso” y “En Revisión” fue siempre menor en el equipo A a excepción de una (“Quitar de Favoritos” que fue tarea puramente funcional). Esta métrica es tan importante al punto que, la suma de minutos que estuvo cada tarea del equipo A en el estado “En Progreso” (296 minutos) es casi la mitad que la suma de minutos que estuvo cada tarea del equipo B en el mismo estado (554 minutos).

Dentro de las causas que generó este resultado, creemos que la más importante fue la presencia de la guía de estilos que proveía la integración con Figma al equipo A. Fue muy notoria la facilidad con la que los desarrolladores sólo debían copiar el código que la guía les proporcionaba para cada elemento visual que necesitaban. Caso contrario, el equipo B debía modificar los archivos CSS e ir haciendo pruebas en tiempo real dentro del navegador para que los estilos queden igual a los diseños. A su vez, el equipo B debía ingresar a la plataforma Figma para encontrar los valores CSS necesarios para cada componente que debía usar, lo que también provocó más demora en los tiempos.

Por último, el tiempo que demoró cada tarea en pasar desde su estado “En Progreso” a su estado “Finalizado” fue siempre menor en el equipo A. Como explicamos anteriormente, en la industria del software se suelen desplegar funcionalidades completas dentro del ambiente de interacción con el usuario final. Al tener un ahorro de tiempo muy importante en el desarrollo de código en todas las tareas, el tiempo que debía esperar cada tarea para que sus complementarias terminaran también fue más corto en el equipo A. Esta sumatoria de ahorro de tiempos generó que el equipo A tardase mucho menos en terminar todas sus tareas en comparación con el equipo B.

No podemos asegurar que la integración de Figma proporcionará siempre un ahorro de tiempo considerable como se vio en el experimento debido a las pocas personas que participaron y la complejidad y cantidad de tareas a realizar, pero sí podemos afirmar que el ahorro de tiempo que se proporciona a la hora de realizar cambios puramente visuales debería ser ampliamente menor en la mayoría de los casos.

3. La integración tuvo un resultado positivo en cuanto a la usabilidad

Los resultados del formulario SUS fueron muy positivos, los 3 conjuntos de respuestas dieron un resultado por arriba del promedio: 77.5% , 80% y 90%. Creemos que al haber tenido reuniones con el fin de hacer un pasaje de conocimiento sobre la integración antes de realizar el experimento fue de suma importancia para que el equipo A no tenga problemas durante el uso real. También entendemos que las tareas a realizar dentro del experimento eran medianamente simples desde el aspecto de diseño lo cual no supuso complicaciones mayores.

Dentro de los comentarios que nos dejaron los integrantes del equipo A, los dos pensamientos más interesantes fueron:

- La necesidad de tener una funcionalidad dentro de la integración que permita enviar cambios desde la plataforma Figma de manera agrupada o en bloque para que el usuario final no tenga acceso a la aplicación en medio del trabajo del diseñador.
- La necesidad de ampliar la integración en cuanto a funcionalidad y componentes visuales para poder resolver tareas más variadas y abrir el campo de uso tanto para los desarrolladores como para el diseñador.

CAPÍTULO 6

Conclusión y Trabajos Futuros

6.1 Conclusiones

Actualmente las metodologías ágiles son la opción elegida por prácticamente todos los equipos de desarrollo para crear un producto de software. Este contexto permite a los equipos el poder ser reactivos a los cambios que se puedan presentar en cualquier etapa del proceso de desarrollo, sin perder performance y constancia en las entregas funcionales de software de cada sprint.

No obstante, los problemas de diseño y usabilidad son una constante dentro de estos sistemas los cuales dificultan una experiencia completa y amena para el usuario final. Y en este contexto ágil no es muy común que el equipo de desarrollo priorice tiempo y esfuerzo en la solución de estos problemas en contraste con la entrega de código que agrega valor funcional. A su vez, los tiempos que conllevan las prácticas de diseño que logran cambios positivos a la usabilidad y experiencia del usuario final no se condicen con los tiempos que requieren las metodologías ágiles, por lo tanto es muy difícil encontrar una sincronización.

Sumado a la diferencia de prioridades y tiempos, nos encontramos con el proceso que conlleva cada cambio hasta llegar a una versión final en un ambiente productivo. El diseñador debe esperar por una extensa lista de actividades sólo para poder probar nuevos diseños, componentes visuales o cambios en los estilos.

Y como si no fuese poco, es muy común que los sistemas que se encuentran en el ambiente productivo tengan inconsistencias en relación a los diseños que propuso el diseñador, y a su vez sea inconsistente dentro del mismo sistema (por ejemplo, que exista variación de colores, tipografías, tamaños para un mismo componente).

Nuestro objetivo fue lograr un nuevo enfoque de trabajo mediante el uso de la herramienta Figma donde el diseñador pueda tomar la responsabilidad de realizar los cambios pertinentes a interfaz de usuario de manera exacta y con más rapidez, dándole lugar al desarrollador para enfocar su trabajo a aspectos relacionados con la lógica y funcionalidad de un sistema. Para cumplir con estos objetivos construimos una integración con la herramienta de diseño Figma que permitiese impactar cambios de interfaz de manera exacta y en poco tiempo.

Pero aún así necesitábamos corroborar que la integración en realidad funcionaba y proveía ventajas en su uso dentro de un ambiente de trabajo real, con personas, sistemas y tareas reales. Con este fin, se realizó un experimento sobre un ambiente ágil utilizando la integración mencionada del cual se obtuvieron resultados muy positivos. Creemos que el

enfoque de trabajo aportado puede seguir evolucionando en el tiempo y podrá facilitar un ambiente de trabajo más ágil y consistente en cuanto al aspecto visual de un sistema.

6.2 Contribuciones

Se desarrolló un plugin para lograr la integración entre la API de Figma y una aplicación web el cual se encuentra detallado en el Capítulo 4. El plugin permite integrar diferentes componentes visuales que existan dentro de la plataforma Figma para ser incluídos de manera sencilla y exacta dentro de la aplicación funcional. A su vez, la integración permite el impacto de modificaciones en la interfaz de usuario dentro del ambiente productivo por parte del diseñador sin la necesidad de un desarrollador de por medio.

El objetivo de este trabajo no se orientó al desarrollo de un plugin de integración entre plataformas, sino a un cambio del enfoque de trabajo dentro de un equipo de desarrollo. Como podemos ver en el Capítulo 3, en este enfoque el diseñador no sólo provee diseños para la aplicación, sino también aplica cambios en la interfaz y usabilidad directamente en ambientes funcionales. A su vez se propuso la posibilidad de aplicación de esta integración para pruebas de A-B testing y pruebas de concepto en ambientes de prueba a cargo puramente del diseñador.

En el Capítulo 5 se detalla el experimento que realizamos del cual participaron desarrolladores y diseñadores profesionales que pusieron a prueba la integración con la API de Figma resolviendo tareas relacionadas a cambios visuales dentro de una aplicación experimental. El equipo que utilizó el plugin desarrollado obtuvo una mejora significativa en los tiempos de entrega de las tareas propuestas en relación a los tiempos obtenidos del equipo que utilizó el enfoque convencional sin el uso del plugin. También se realizó una evaluación de usabilidad, la cual dio un resultado muy positivo en cuanto a la integración. Se obtuvo feedback que apuntó a la posibilidad de agregar nuevas funcionalidades y a la ampliación del plugin para lograr resolver tareas más complejas y variadas.

6.3 Limitaciones

Si bien los resultados de este trabajo fueron muy positivos y crearon una gran variedad de ampliaciones y aplicaciones posibles, a medida que íbamos progresando en la investigación nos encontramos con algunas limitaciones.

Como mencionamos en la sección Complejidades y Limitaciones Capítulo 3, hubo algunos campos de aplicación de la integración que no pudimos lograr por diferentes razones:

- Aplicar diseños responsive: la posibilidad de poder lograr una integración la cual pudiese diferenciar entre diferentes tamaños de pantalla y resoluciones existentes no resultó imposible pero sí quizás un poco innecesario. Luego de una investigación

realizada en conjunto con una diseñadora profesional entendimos que los diseños responsivos no suelen variar en cuanto al diseño visual de los componentes, sino en cuanto a la disposición y cantidad de información que se muestra. No obstante, sí es posible realizar una integración orientada a diseños responsive en cuanto a componentes visuales aislados sin incursionar en posicionamientos o grillas. En la siguiente sección explicaremos cómo se debería realizar.

- Aplicar posicionamiento de componentes: ésta era una de las aristas de la integración que quisimos lograr en un principio. El poder no sólo integrar los estilos visuales de un componente sino también su posicionamiento respecto a la página donde se encuentra. Pero la API de Figma no provee información respecto al posicionamiento de los componentes dentro de los diseños, sólo posicionamiento absoluto dentro del archivo Figma en relación a los ejes *x* e *y*. Quizás en un futuro esta información pueda ser obtenida mediante la API, pero por ahora no existe esa funcionalidad.
- Aplicar comportamiento a los componentes visuales: con el fin no sólo de realizar cambios UI sino también UX, intentamos encontrar algún tipo de integración mediante el uso de comentarios. Pero luego de consultar la factibilidad de esta opción, nos encontramos que era algo imposible de lograr. El diseñador tiene las facultades de poder identificar problemas y sugerir soluciones, pero no posee las herramientas ni datos necesarios para impactar los cambios sobre un componente en particular en un momento y contexto en particular.

6.4 Trabajos futuros

Durante el desarrollo de este trabajo nos encontramos con diferentes temáticas, las cuales nos hubiera gustado profundizar pero no incursionamos en ellos con el fin de no salirnos del marco de trabajo que nos propusimos originalmente. A continuación describimos cada una de estas temáticas para que puedan ser revisadas en un futuro e incorporadas a este proyecto.

Cambio de tecnologías

Al iniciar con este trabajo nos propusimos desarrollar un plugin que logre integrar la API de Figma con una aplicación. Para lograr esto necesitábamos contar con una plataforma backend que se encargue de conectar con la API y procesar los datos, y otra plataforma frontend que se encargue de utilizar los datos procesados para renderizar los componentes visuales integrados. Además RoR presenta un sistema de despliegues dentro de Heroku lo cual nos permitía realizar despliegues rápidos sin mucha configuración de por medio mientras desarrollamos el plugin.

Como tecnología backend utilizamos Ruby on Rails ya que esta tecnología nos permite usar el server con claves privadas de Figma pero a la vez usar otra tecnología en el frontend en vez de tener dos aplicaciones/repositorios por separado.

Como tecnología frontend utilizamos ReactJS por dos grandes razones. En primer lugar, porque es la tecnología con la que nos sentíamos más seguros al encarar el proyecto siendo la que utilizamos todos los días en el ámbito laboral. En segundo lugar, existe un framework UI para React llamado Material-UI que incluye el concepto de tema dentro de sus funcionalidades. Éste concepto es una de las bases utilizadas en el plugin desarrollado. Ahora bien, ninguna de las razones propuestas genera una restricción entre las tecnologías utilizadas y el plugin desarrollado, por lo tanto, uno de los cambios propuestos para un futuro sería poder generar el mismo plugin pero con diferentes tecnologías.

Para poder reemplazar la tecnología backend utilizada por otra (Django, NodeJS, Spring, Symfony, etc) se debe re-implementar en la tecnología escogida el componente FigmaService**.** Este componente es el encargado de procesar los llamados que se realicen a cada endpoint del backend para luego obtener la información de la API de Figma y luego procesarla.

Para poder reemplazar la tecnología frontend utilizada por otra (AngularJS, VueJS, EmberJS, etc) se debe prestar especial atención a la reescritura de la definición de los FigmaComponents. A su vez al cambiar React por otra tecnología, quedaría inutilizado Material-UI, y en consecuencia se debería utilizar otro framework frontend que soporte el concepto de temas (Bootstrap, Semantic UI, Angular Material, Foundation, etc). Una vez escogido el framework se deberá reescribir el FigmaTheme utilizando las convenciones y restricciones que posea el framework en cuestión.

Mayor integración de componentes y conceptos

Siguiendo el lineamiento mencionado en la sección de Limitaciones y en base al feedback obtenido en las respuestas del formulario SUS podemos mencionar las siguientes propuestas:

- Mayor integración de componentes: el plugin desarrollado sólo integra alguno de todos los componentes que se pueden encontrar en un kit UI. Como el fin de este trabajo no era lograr una integración amplia sino descubrir si el enfoque utilizado ahorraba tiempos en el desarrollo, no se utilizó mucho tiempo para lograr una integración mayor. Por lo tanto, una de las posibles ampliaciones de este trabajo sería lograr mayor diversidad de componentes a integrar.
- Integración de posicionamiento: por el momento la API de Figma no provee información sobre el posicionamiento de los componentes en base a sus contenedores. Si en un futuro esto se incluyese dentro de la API, sería una temática en que valdría la pena incursionar.
- Integración responsive: como mencionamos en la sección Aplicar cambios relacionados al diseño responsivo Capítulo 3, es posible lograr una integración con fin responsivo sólo para los estilos visuales de los componentes. Para lograr esto, se debe tomar el mismo enfoque que se explica en la sección Aplicar estado a los componentes. Sólo que en vez de hacer un diseño por cada estado del componente,

se debería crear los diseños con estados para el mismo componente pero en diferentes dimensiones de pantalla. Luego el frontend debería poder diferenciar en base al dispositivo en que se está renderizando la aplicación, cuál diseño utilizar.

Aplicabilidad de A-B testing

Si bien en la sección Aplicar A-B testing del Capítulo 3 se dio un primer enfoque de cómo podría realizarse una prueba de A-B testing utilizando la integración con Figma y sin depender de un desarrollador (salvo en el seteo del contexto a trabajar), no se realizó ningún experimento que demuestre la factibilidad y las ventajas de dicho enfoque. Quedaría como trabajo futuro poder realizar un experimento que compare el enfoque convencional con el propuesto y verificar que las métricas obtenidas den un resultado positivo.

Draft entre Figma y aplicación

Dentro de las devoluciones que obtuvimos del formulario SUS al realizar el experimento detallado en el Capítulo 5, nos encontramos con una interesante sugerencia por parte de la diseñadora. Se propuso una funcionalidad dentro de la integración que permita enviar cambios desde la plataforma Figma en bloque para que el usuario final no tenga acceso a la aplicación en medio del trabajo del diseñador.

Referencias bibliográficas

- [AdobeXDContentApi20]: Adobe XD Cloud Content APIs https://adobexdplatform.com/cloud-content-api-docs/. Accedido el 30/03/20.
- [AdobeXDPluginApi20]: Adobe XD Plugin APIs. https://adobexdplatform.com/plugin-docs/. Accedido el 30/03/20.
- [AdobeXDPricing20]: Adobe XD. https://www.adobe.com/products/xd.html. Accedido el 30/03/20.
- [Bacchelli13] Bacchelli, Alberto, and Christian Bird. "Expectations, outcomes, and challenges of modern code review." In 2013 35th International Conference on Software Engineering (ICSE), pp. 712-721. IEEE, 2013.
- [Bangor08] Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An empirical evaluation of the system usability scale. Intl. Journal of Human–Computer Interaction, 24(6), 574-594.
- [Bačíková15] Bačíková, M., "User experience design: Contrasting academic with practice," 2015 13th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, 2015, pp. 1-6.
- [Beck04] Beck, K. “Extreme Programming”. Addison-Wesley 2004.
- [Beux18] Beux, Jucélia, Ericles Bellei, Laís Brock, Ana Carolina Bertoletti, and Carlos Hölbig. "Agile Design Process with User-Centered Design and User Experience in Web Interfaces: A Systematic Literature Review." Latin American Journal of Computing Faculty of Systems Engineering Escuela Politécnica Nacional Quito-Ecuador 5, no. 2 (2018): 53-60.
- [Chamberlain06] S. Chamberlain, H. Sharp, and N. Maiden, “Towards a framework for integrating agile development and user-centred design,” in Extreme Programming and Agile Processes in Software Engineering, P. Abrahamsson, M. Marchesi, and G. Succi, Eds. Berlin, Heidelberg: Springer. Berlin Heidelberg, 2006, pp. 143–153.
- [Cohn04] Cohn, Mike. User stories applied: For agile software development. Addison-Wesley Professional, 2004.
- [DaSilva11] Da Silva, T. S., Martin, A., Maurer, F., & Silveira, M. “User-centered design and agile methods: a systematic review”. In 2011 Agile Conference (pp. 77-86). IEEE. 2011.
- [DaSilva18] Da Silva, T. S., Silveira, M. S., Maurer, F., & Silveira, F. F. “The evolution of agile UXD”. Information and Software Technology, 102, 1-5. 2018.
- [Figma19] Figma: the collaborative interface design tool. www.figma.com. Accedido el 11/11/19.
- [FigmaApi20]: Figma Rest API. https://www.figma.com/developers/api. Accedido el 30/03/20.
- [FigmaAutenticación20]: Autenticación de Figma API. https://www.figma.com/developers/api#authentication. Accedido el 12/08/20.
- [FigmaPricing20]: Figma Pricing. https://www.figma.com/pricing/. Accedido el 30/03/20.
- [Fowler06] Fowler, Martin, and Matthew Foemmel. "Continuous integration." (2006).

- [Fowler18] Fowler, M. (2018). Refactoring: improving the design of existing code. Addison-Wesley Professional.
- [Fowler99]: Beck, K., Fowler, M., & Beck, G. (1999). Bad smells in code. Refactoring: Improving the design of existing code, 1, 75-88.
- [Garrett10] Garrett, Jesse James. Elements of user experience, the: user-centered design for the web and beyond. Pearson Education, 2010.
- [Garrido, Grigera 11] Garrido, A., Rossi, G., Distante, D., 2011. Refactoring for usability in web applications.IEEE Softw. 28 (3), 60–67.
- [Gaurav12] Gaurav, K., Pradeep, K. B. “Impact of Agile Methodology on Software Development Process”. In 2012 International Journal of Computer Technology and Electronics Engineering (IJCTEE). Volume 2, Issue 4, August 2012.
- [Giacomelli18] Giacomelli Beux, J., Ericles, A. B., Brock, L. A., Bertoletti De Marchi, A. C., Holbig, C. A. Agile Design Process with User-Centered Design and User Experience in Web Interfaces: A Systematic Literature Review. ISSN: 1390-9266 - LAJC. (2018).
- [Grigera17] Grigera, J., Garrido, A., Rivero, JM., Rossi, G. Automatic detection of usability smells in web applications. Int. J. Human–Computer Studies 97 (2017) 129-148.
- [Haugen06] Haugen, N. C. (2006, July). An empirical study of using planning poker for user story estimation. In AGILE 2006 (AGILE'06) (pp. 9-pp). IEEE.
- [Heroku20] Heroku. https://www.heroku.com/. Accedido el 17/08/20.
- [Kohavi07] Kohavi, R., Henne, R. M., & Sommerfield, D. (2007, August). Practical guide to controlled experiments on the web: listen to your customers not to the hippo. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 959-967).
- [Krug14] Krug, S. “Don’t Make Me Think, Revisited: A Common Sense Approach to Web Usability." Berkeley, CA: New Riders (2014).
- [Lewis09] Lewis, J. R., & Sauro, J. (2009, July). The factor structure of the system usability scale. In International conference on human centered design (pp. 94-103). Springer, Berlin, Heidelberg.
- [ManifiestoAgil20] Manifiesto Ágil: https://agilemanifesto.org/iso/es/manifesto.html. Accedido 12/08/20.
- [Marcotte17] Marcotte, E. (2017). Responsive web design: A book apart n 4. Editions Eyrolles.
- [MaterialDesign20] Material Design. https://material.io/design. Accedido el 12/08/20.
- [Mkrtchyan18] Mkrtchyan, R. (2018). Wireframe, Mockup, Prototype: What is What? https://uxplanet.org/wireframe-mockup-prototype-what-is-what-8cf2966e5a8b. Accedido el 17/08/20.
- [Nielsen06] Nielsen, J., & Loranger, H. (2006). Prioritizing web usability. Pearson Education.
- [RoR20]: Ruby on Rails. https://rubyonrails.org/. Accedido el 17/08/20.
- [Rubin12] Rubin, Kenneth S. Essential Scrum: A practical guide to the most popular Agile process. Addison-Wesley, 2012.
- [Schwaber02] Schwaber, Ken, and Mike Beedle. Agile software development with Scrum. Vol. 1. Upper Saddle River: Prentice Hall, 2002.

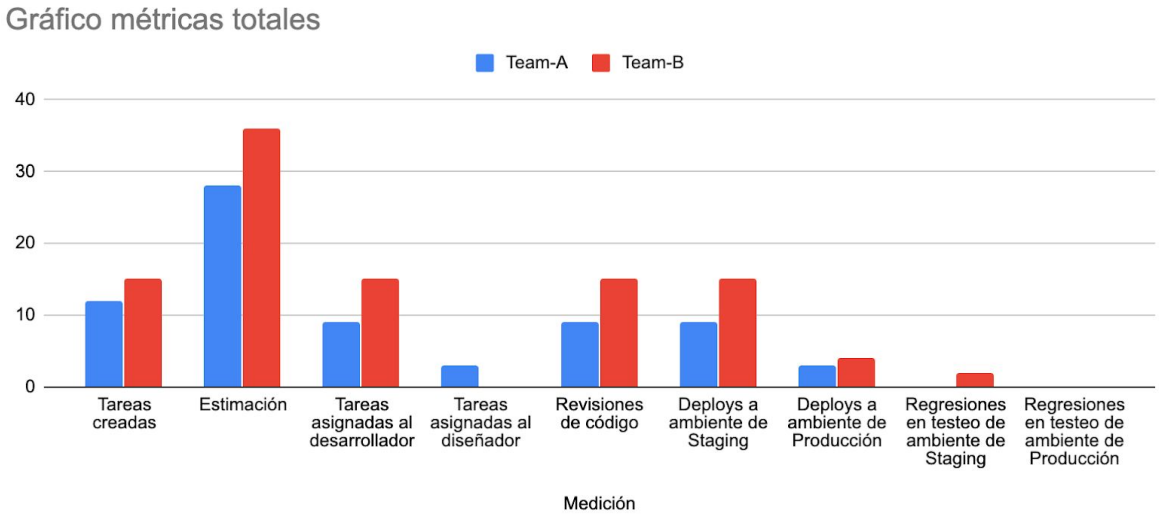
- [SketchApi20]: Sketch API Reference. https://developer.sketch.com/reference/api/. Accedido el 30/03/20.
- [SketchPricing20]: Sketch Pricing for individuals and teams. https://www.sketch.com/pricing/. Accedido el 30/03/20.
- [Sommerville11] Sommerville, Ian. "Software engineering 9th Edition." ISBN-10 137035152 (2011).
- [Tian05] Tian, Jeff. Software quality engineering: testing, quality assurance, and quantifiable improvement. John Wiley & Sons, 2005.

Índice de figuras

Anexo métricas y gráficos

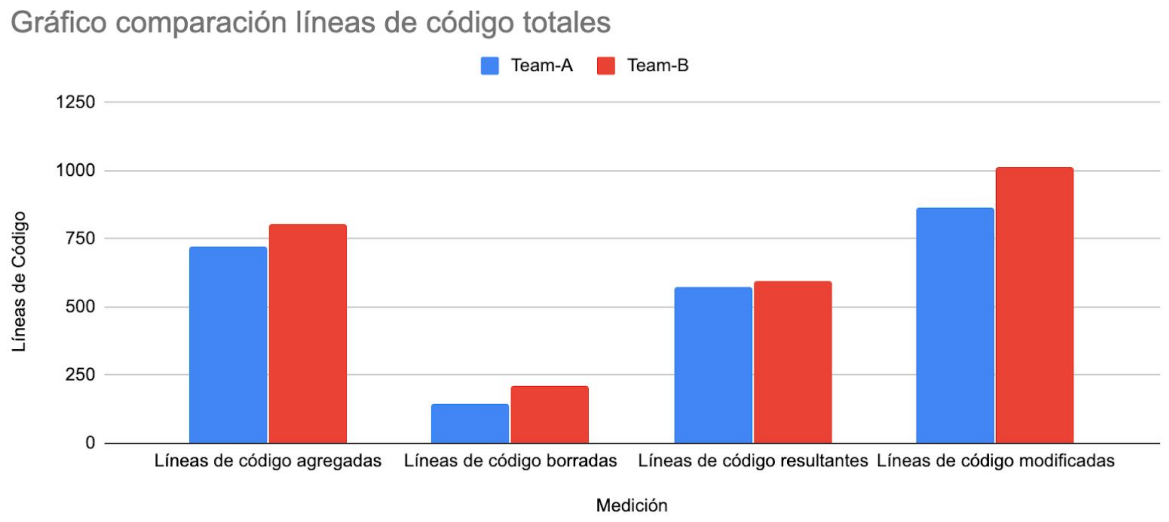
Métricas totales - Generales

Medición	Team-A	Team-B
Tareas creadas	12	15
Estimación	28	36
Tareas asignadas al desarrollador	9	15
Tareas asignadas al diseñador	3	0
Revisiones de código	9	15
Deploys a ambiente de Staging	9	15
Deploys a ambiente de Producción	3	4
Regresiones en testeo de ambiente de Staging	0	2
Regresiones en testeo de ambiente de Producción	0	0



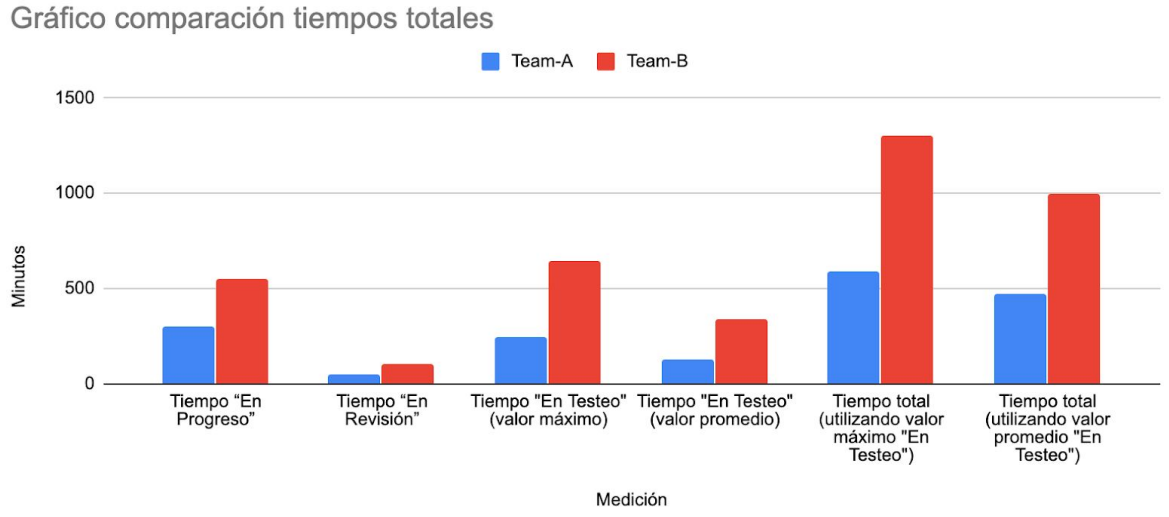
Métricas totales - Líneas de código

Medición	Team-A	Team-B
Líneas de código agregadas	720	802
Líneas de código borradas	145	209
Líneas de código resultantes	575	593
Líneas de código modificadas	865	1011



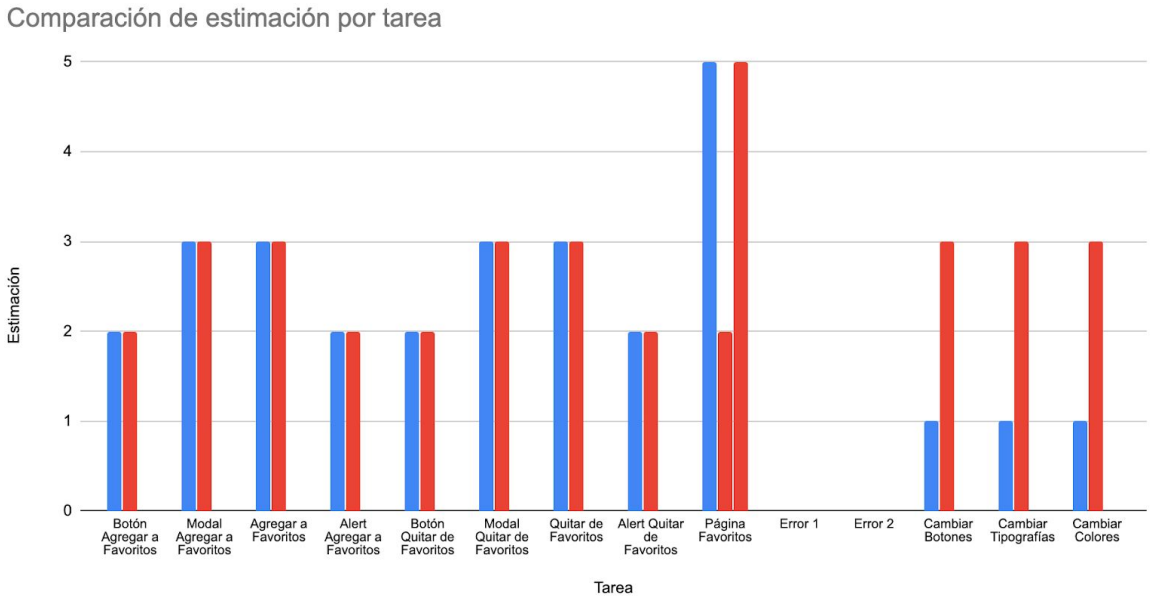
Métricas totales - Tiempos

Medición	Team-A	Team-B
Tiempo “En Progreso”	296	554
Tiempo “En Revisión”	49	107
Tiempo "En Testeo" (valor máximo)	247	643
Tiempo "En Testeo" (valor promedio)	129.75	336.33
Tiempo total (utilizando valor máximo "En Testeo")	592	1304
Tiempo total (utilizando valor promedio "En Testeo")	474.75	997.33



Métricas de estimación - Por tarea

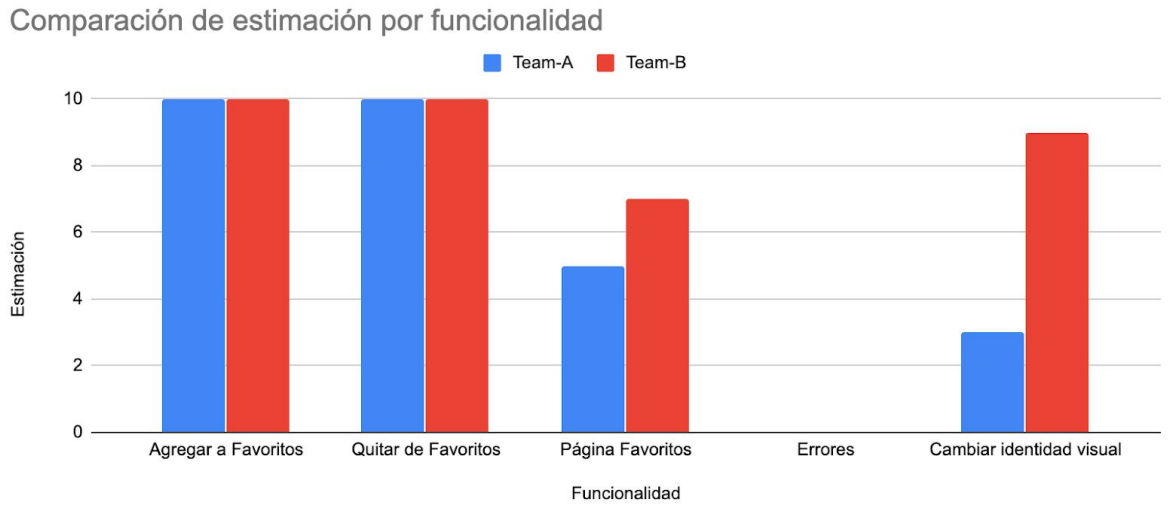
Tarea	Team-A	Team-B
Botón Agregar a Favoritos	2	2
Modal Agregar a Favoritos	3	3
Agregar a Favoritos	3	3
Alert Agregar a Favoritos	2	2
Botón Quitar de Favoritos	2	2
Modal Quitar de Favoritos	3	3
Quitar de Favoritos	3	3
Alert Quitar de Favoritos	2	2
Página Favoritos	5	2 + 5
Error 1	0	0
Error 2	0	0
Cambiar Botones	1	3
Cambiar Tipografías	1	3
Cambiar Colores	1	3



Métricas de estimación - Por funcionalidad

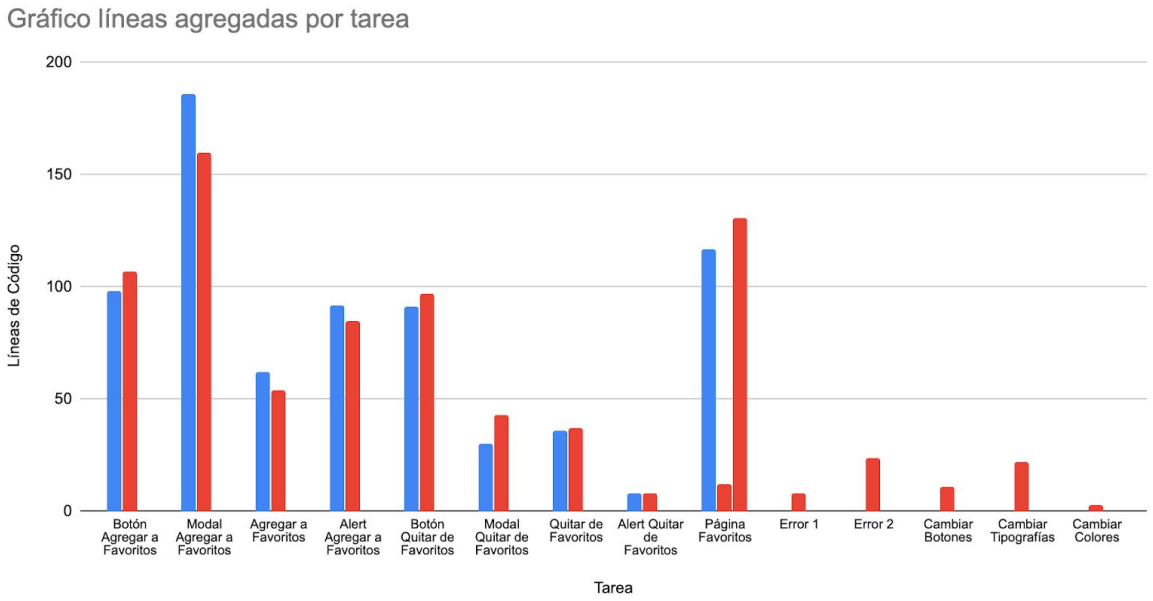
Funcionalidad	Team-A	Team-B
Agregar a Favoritos	10	10
Quitar de Favoritos	10	10

Página Favoritos	5	7
Errores	0	0
Cambiar identidad visual	3	9



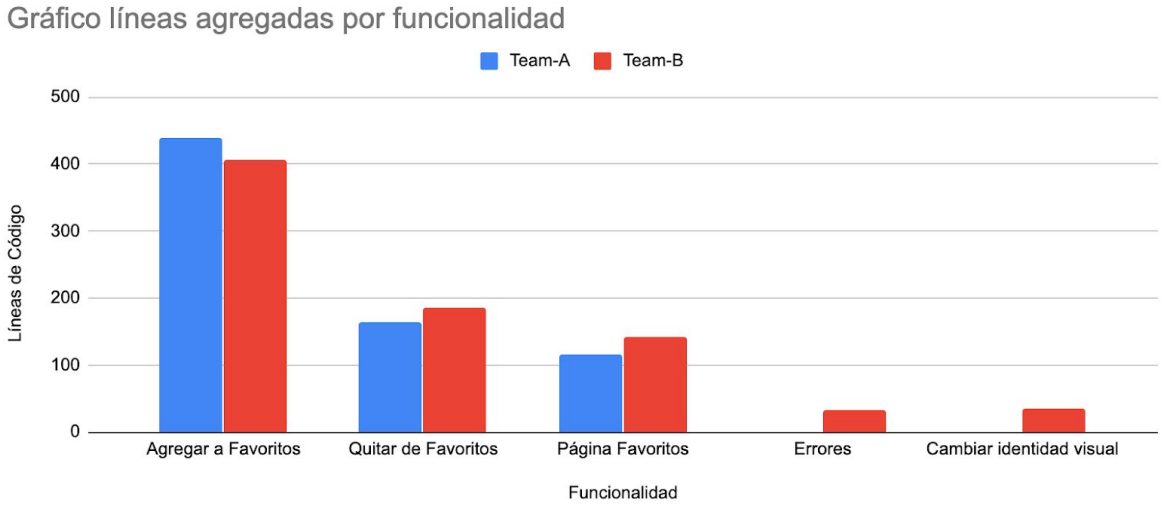
Métricas de líneas agregadas - Por tarea

Tarea	Team-A	Team-B
Botón Agregar a Favoritos	98	107
Modal Agregar a Favoritos	186	160
Agregar a Favoritos	62	54
Alert Agregar a Favoritos	92	85
Botón Quitar de Favoritos	91	97
Modal Quitar de Favoritos	30	43
Quitar de Favoritos	36	37
Alert Quitar de Favoritos	8	8
Página Favoritos	117	12 + 131
Error 1	0	8
Error 2	0	24
Cambiar Botones	0	11
Cambiar Tipografías	0	22
Cambiar Colores	0	3



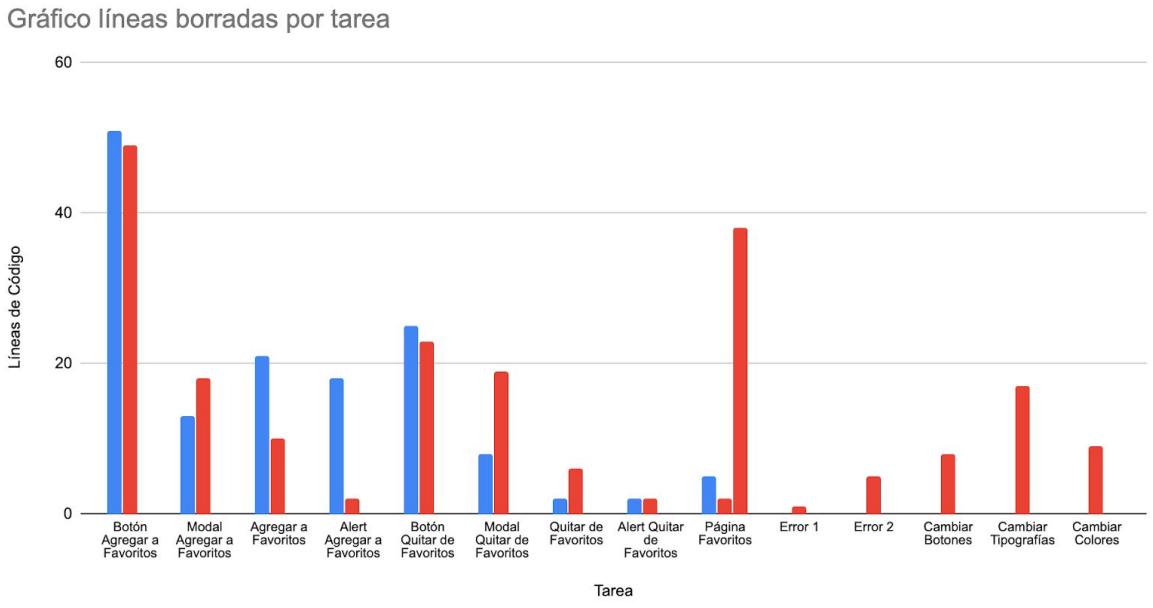
Métricas de líneas agregadas - Por funcionalidad

Funcionalidad	Team-A	Team-B
Agregar a Favoritos	438	406
Quitar de Favoritos	165	185
Página Favoritos	117	143
Errores	0	32
Cambiar identidad visual	0	36



Métricas de líneas borradas - Por tarea

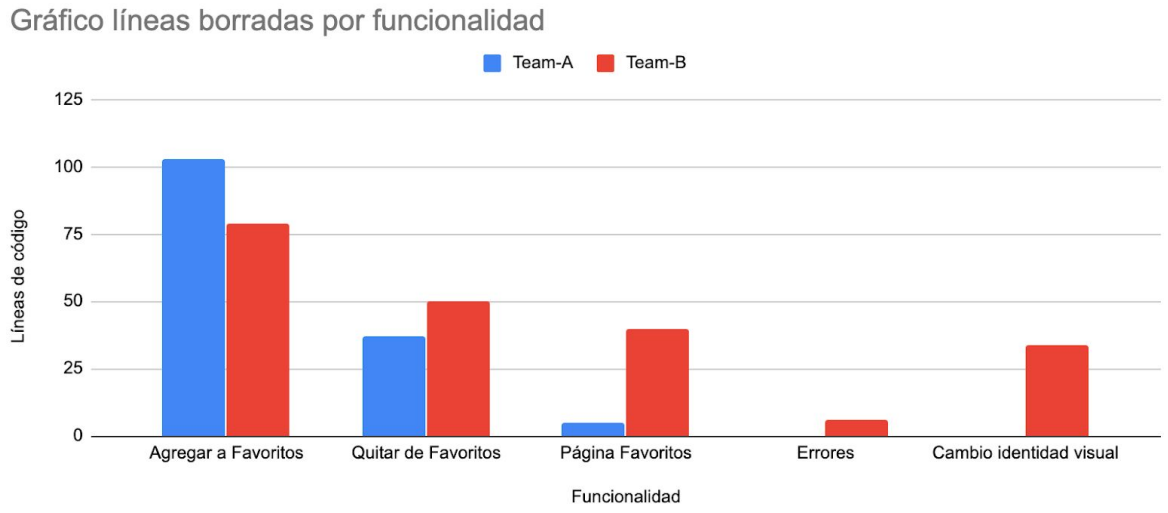
Tarea	Team-A	Team-B
Botón Agregar a Favoritos	51	49
Modal Agregar a Favoritos	13	18
Agregar a Favoritos	21	10
Alert Agregar a Favoritos	18	2
Botón Quitar de Favoritos	25	23
Modal Quitar de Favoritos	8	19
Quitar de Favoritos	2	6
Alert Quitar de Favoritos	2	2
Página Favoritos	5	2 + 38
Error 1	0	1
Error 2	0	5
Cambiar Botones	0	8
Cambiar Tipografías	0	17
Cambiar Colores	0	9



Métricas de líneas borradas - Por funcionalidad

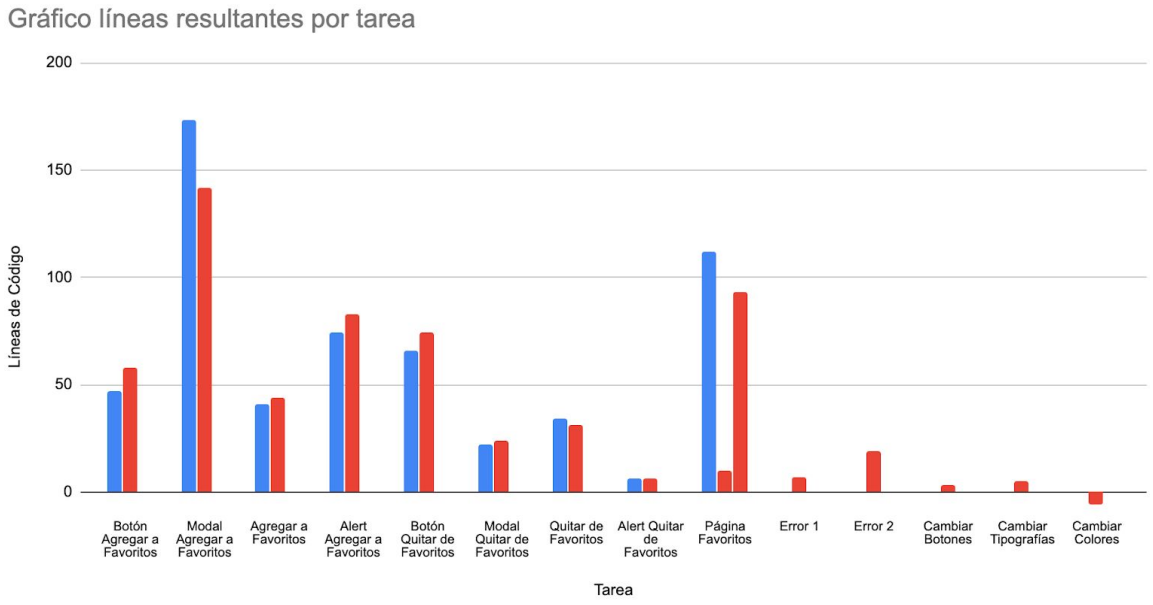
Funcionalidad	Team-A	Team-B
Agregar a Favoritos	103	79
Quitar de Favoritos	37	50

Página Favoritos	5	40
Errores	0	6
Cambio identidad visual	0	34



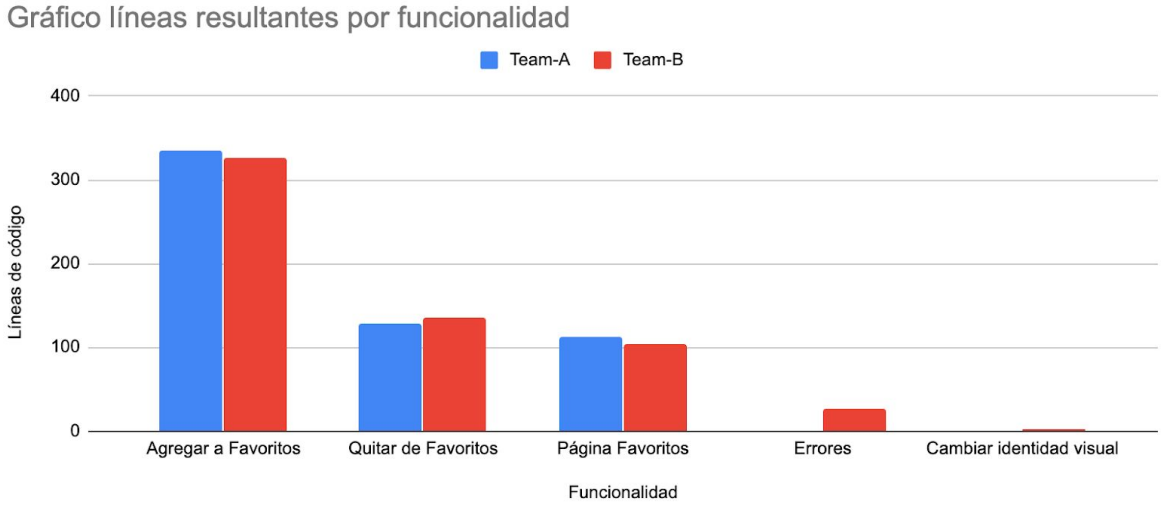
Métricas de líneas resultantes - Por tarea

Tarea	Team-A	Team-B
Botón Agregar a Favoritos	47	58
Modal Agregar a Favoritos	173	142
Agregar a Favoritos	41	44
Alert Agregar a Favoritos	74	83
Botón Quitar de Favoritos	66	74
Modal Quitar de Favoritos	22	24
Quitar de Favoritos	34	31
Alert Quitar de Favoritos	6	6
Página Favoritos	112	10 + 93
Error 1	0	7
Error 2	0	19
Cambiar Botones	0	3
Cambiar Tipografías	0	5
Cambiar Colores	0	-6



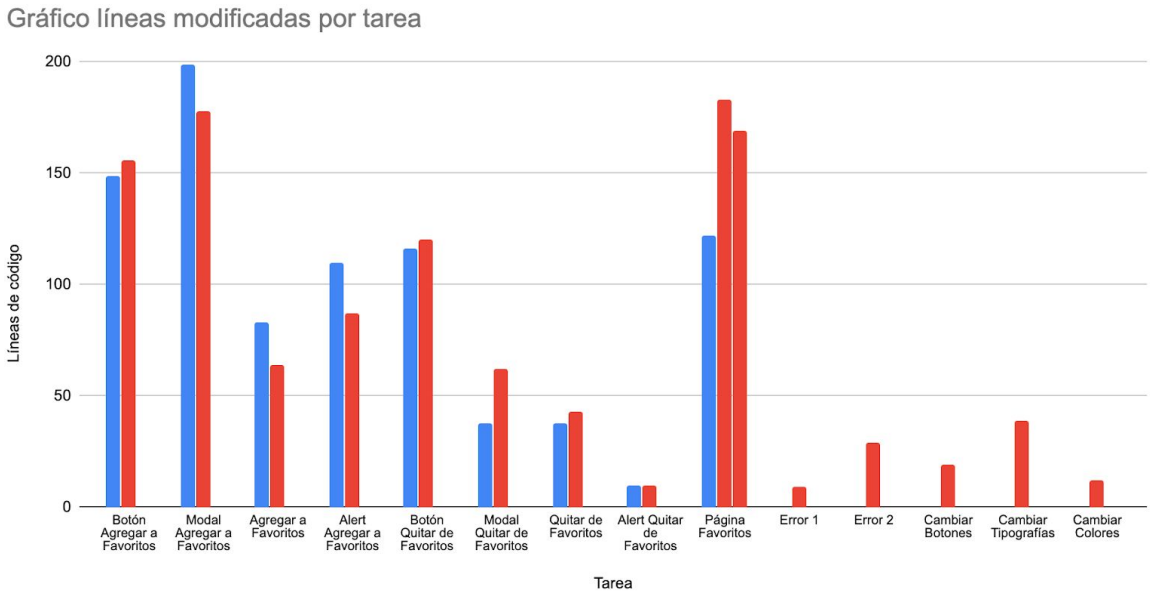
Métricas de líneas resultantes - Por funcionalidad

Funcionalidad	Team-A	Team-B
Agregar a Favoritos	335	327
Quitar de Favoritos	128	135
Página Favoritos	112	103
Errores	0	26
Cambiar identidad visual	0	2



Métricas de líneas modificadas - Por tarea

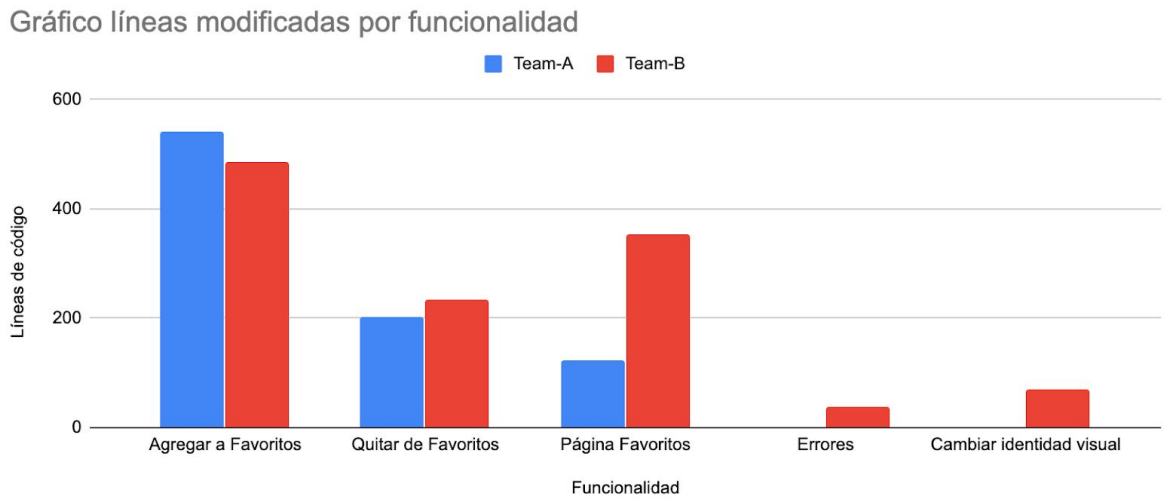
Tarea	Team-A	Team-B
Botón Agregar a Favoritos	149	156
Modal Agregar a Favoritos	199	178
Agregar a Favoritos	83	64
Alert Agregar a Favoritos	110	87
Botón Quitar de Favoritos	116	120
Modal Quitar de Favoritos	38	62
Quitar de Favoritos	38	43
Alert Quitar de Favoritos	10	10
Página Favoritos	122	183 + 169
Error 1	0	9
Error 2	0	29
Cambiar Botones	0	19
Cambiar Tipografías	0	39
Cambiar Colores	0	12



Métricas de líneas modificadas - Por funcionalidad

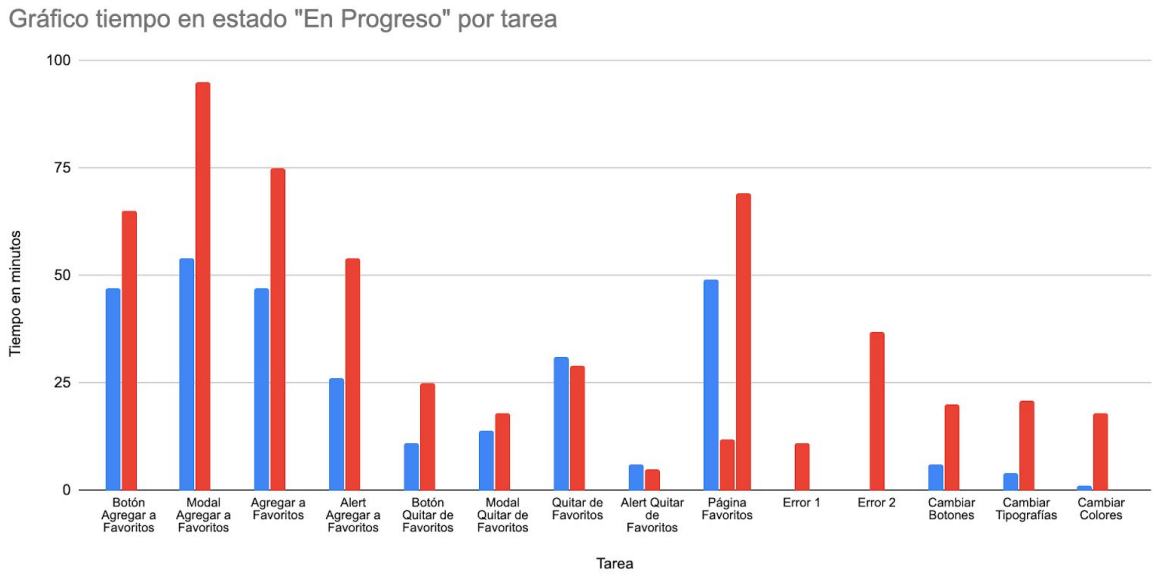
Tarea	Team-A	Team-B
Agregar a Favoritos	541	485
Quitar de Favoritos	202	235
Página Favoritos	122	352

Errores	0	38
Cambiar identidad visual	0	70



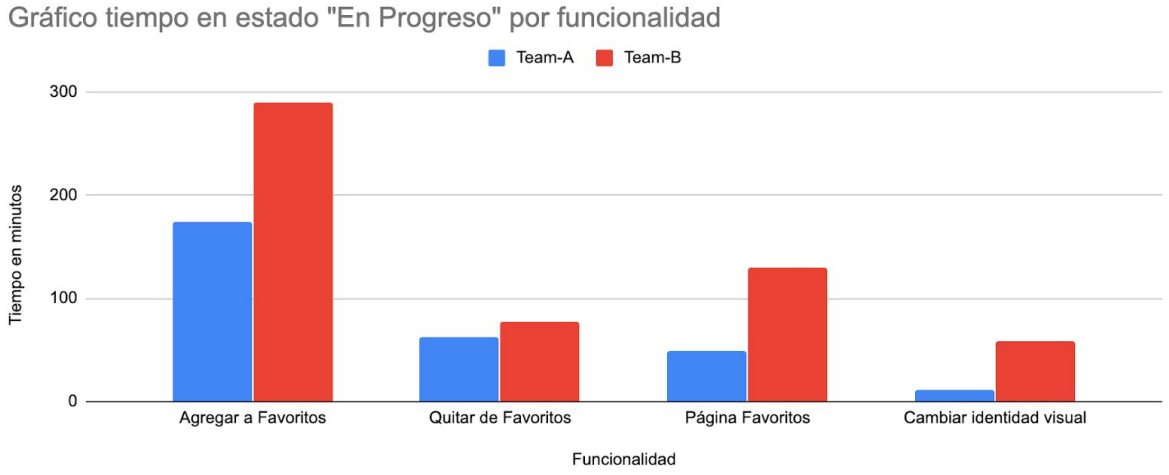
Métricas de tiempo en estado “En Progreso” - Por tarea

Tarea	Team-A	Team-B
Botón Agregar a Favoritos	47	65
Modal Agregar a Favoritos	54	95
Agregar a Favoritos	47	75
Alert Agregar a Favoritos	26	54
Botón Quitar de Favoritos	11	25
Modal Quitar de Favoritos	14	18
Quitar de Favoritos	31	29
Alert Quitar de Favoritos	6	5
Página Favoritos	49	12 + 69
Error 1	0	11
Error 2	0	37
Cambiar Botones	6	20
Cambiar Tipografías	4	21
Cambiar Colores	1	18



Métricas de tiempo en estado “En Progreso” - Por funcionalidad

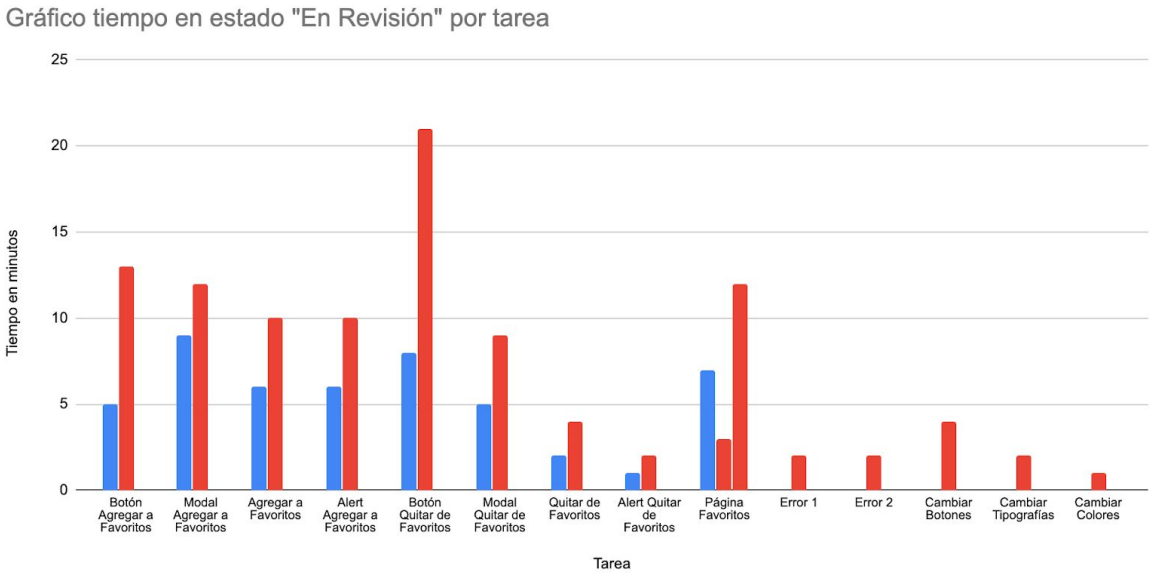
Funcionalidad	Team-A	Team-B
Agregar a Favoritos	174	289
Quitar de Favoritos	62	77
Página Favoritos	49	129
Cambiar identidad visual	11	59



Métricas de tiempo en estado “En Revisión” - Por tarea

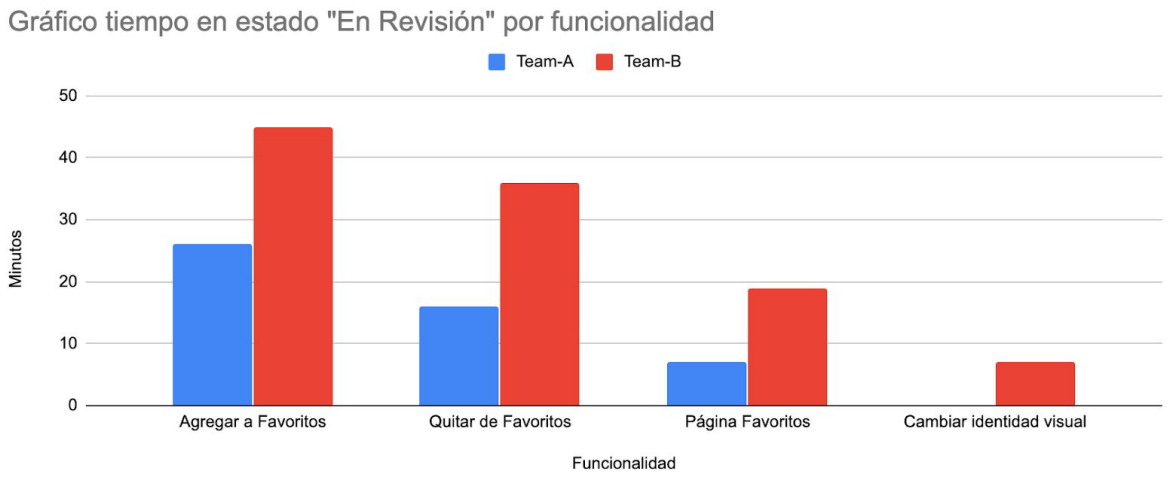
Tarea	Team-A	Team-B
Botón Agregar a Favoritos	5	13
Modal Agregar a Favoritos	9	12

Agregar a Favoritos	6	10
Alert Agregar a Favoritos	6	10
Botón Quitar de Favoritos	8	21
Modal Quitar de Favoritos	5	9
Quitar de Favoritos	2	4
Alert Quitar de Favoritos	1	2
Página Favoritos	7	3 + 12
Error 1	0	2
Error 2	0	2
Cambiar Botones	0	4
Cambiar Tipografías	0	2
Cambiar Colores	0	1



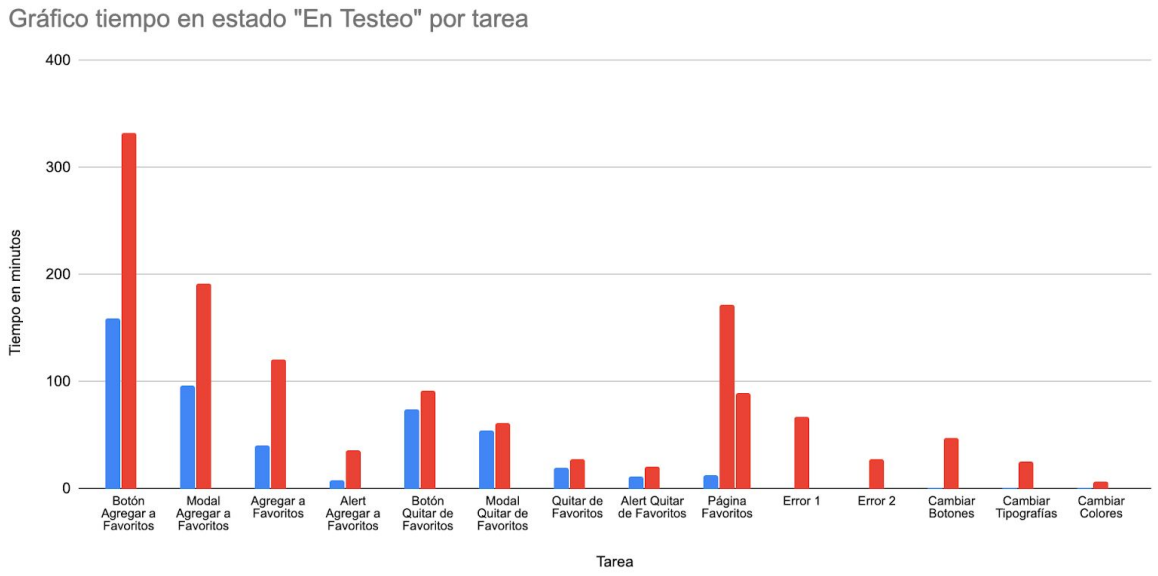
Métricas de tiempo en estado “En Revisión” - Por funcionalidad

Funcionalidad	Team-A	Team-B
Agregar a Favoritos	26	45
Quitar de Favoritos	16	36
Página Favoritos	7	19
Cambiar identidad visual	0	7



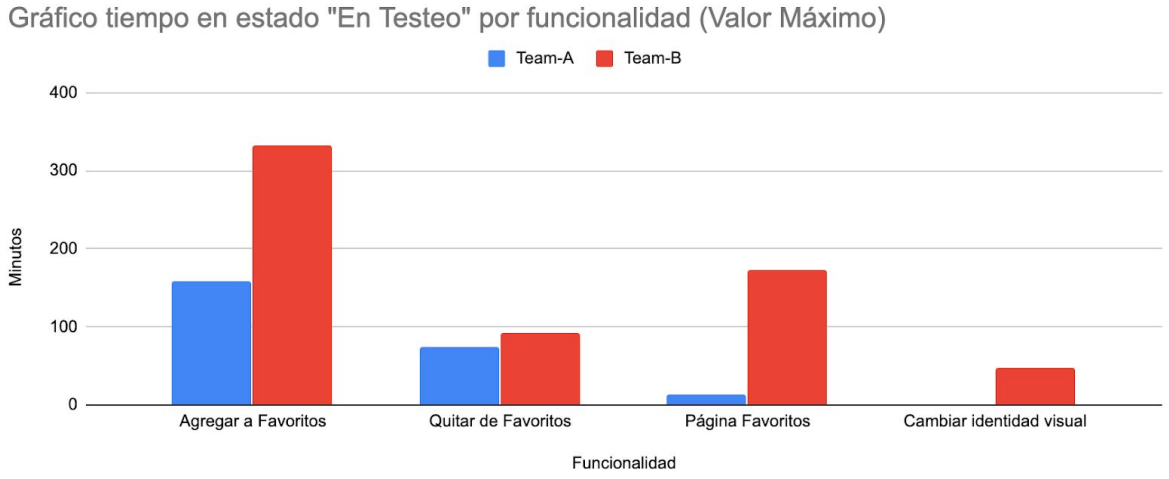
Métricas de tiempo en estado “En Testeo” - Por tarea

Tarea	Team-A	Team-B
Botón Agregar a Favoritos	159	332
Modal Agregar a Favoritos	96	192
Agregar a Favoritos	40	121
Alert Agregar a Favoritos	8	36
Botón Quitar de Favoritos	74	92
Modal Quitar de Favoritos	54	62
Quitar de Favoritos	20	28
Alert Quitar de Favoritos	12	21
Página Favoritos	13	172 + 89
Error 1	0	67
Error 2	0	28
Cambiar Botones	1	47
Cambiar Tipografías	1	25
Cambiar Colores	1	7



Métricas de tiempo en estado “En Testeo” - Por funcionalidad usando Valor Máximo

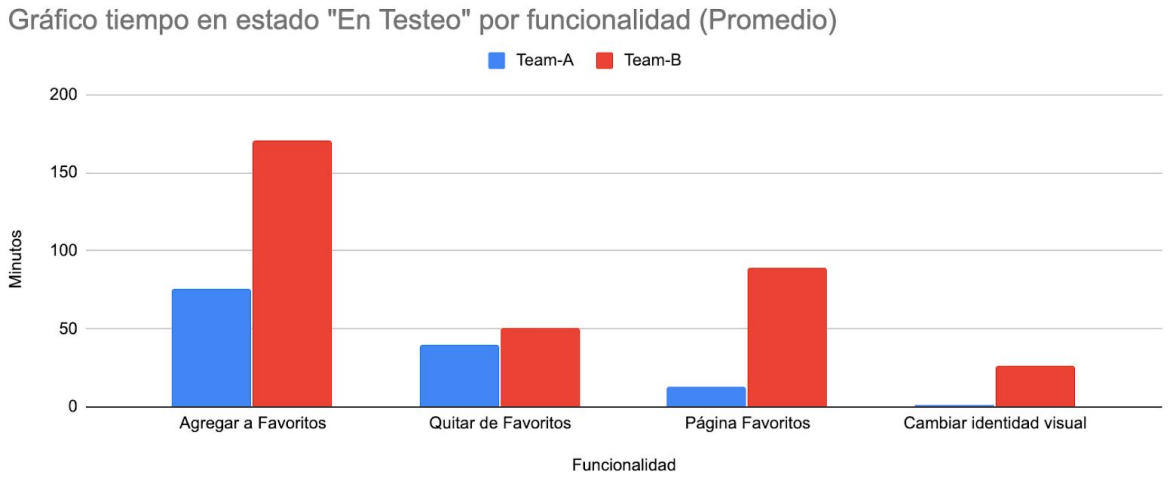
Funcionalidad	Team-A	Team-B
Agregar a Favoritos	159	332
Quitar de Favoritos	74	92
Página Favoritos	13	172
Cambiar identidad visual	1	47



Métricas de tiempo en estado “En Testeo” - Por funcionalidad usando Valor Promedio

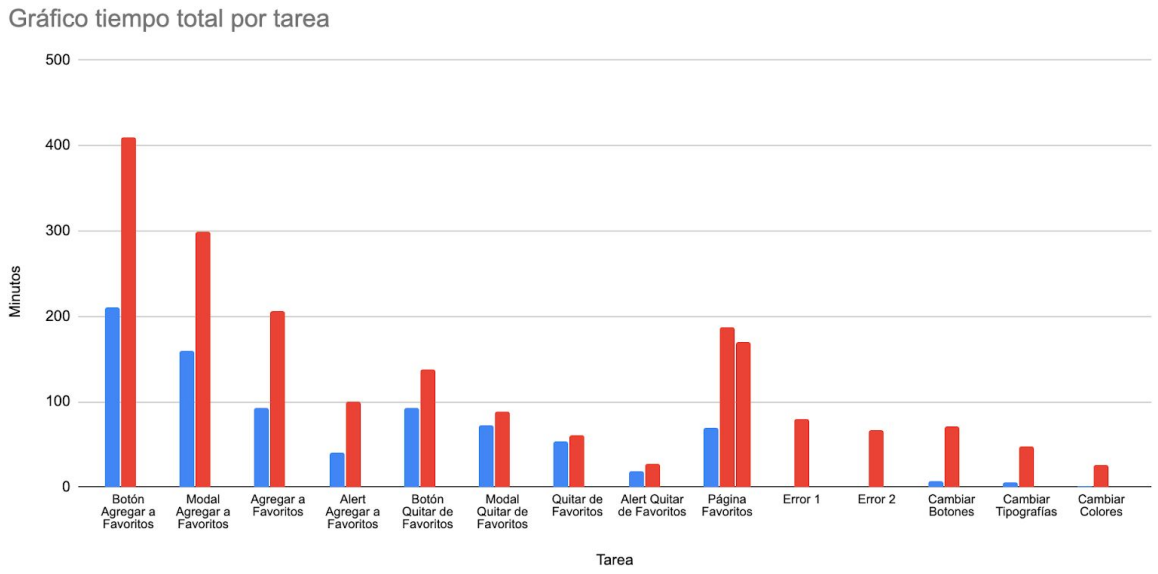
Funcionalidad	Team-A	Team-B
Agregar a Favoritos	75.75	170.25
Quitar de Favoritos	40	50.75

Página Favoritos	13	89
Cambiar identidad visual	1	26.33



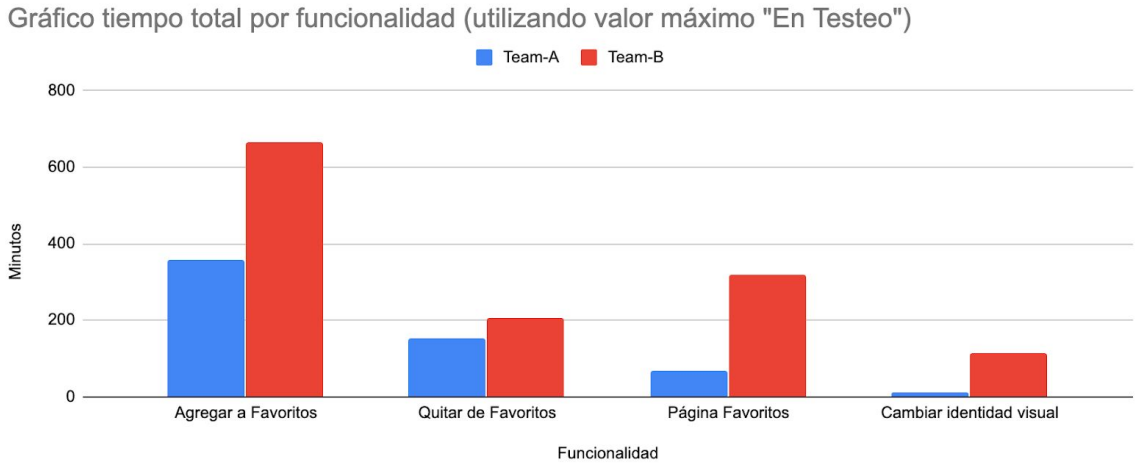
Métricas de tiempo total - Por tarea

Tarea	Team-A	Team-B
Botón Agregar a Favoritos	211	410
Modal Agregar a Favoritos	159	299
Agregar a Favoritos	93	206
Alert Agregar a Favoritos	40	100
Botón Quitar de Favoritos	93	138
Modal Quitar de Favoritos	73	89
Quitar de Favoritos	53	61
Alert Quitar de Favoritos	19	28
Página Favoritos	69	187 + 170
Error 1	0	80
Error 2	0	67
Cambiar Botones	7	71
Cambiar Tipografías	5	48
Cambiar Colores	2	26



Métricas de tiempo total - Por funcionalidad usando valor máximo en “En Testeo”

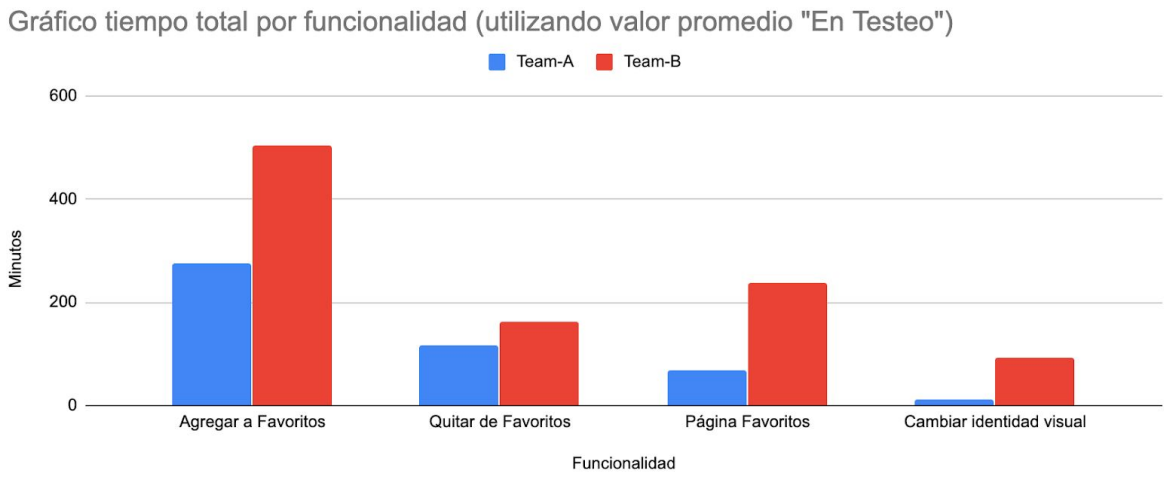
Funcionalidad	Team-A	Team-B
Agregar a Favoritos	359	666
Quitar de Favoritos	152	205
Página Favoritos	69	320
Cambiar identidad visual	12	113



Métricas de tiempo total - Por funcionalidad usando valor promedio en “En Testeo”

Funcionalidad	Team-A	Team-B
Agregar a Favoritos	275.75	504.25
Quitar de Favoritos	118	163.75

Página Favoritos	69	237
Cambiar identidad visual	12	92.3



Anexo plantilla SUS

Fecha:
Nombre y Apellido:
Edad:
Profesión:

Instrucciones: para cada una de las siguientes afirmaciones, marque la casilla que mejor describa su opinión sobre la integración actualmente.

	Totalmente en desacuerdo				Totalmente de acuerdo
1. Opino que me gustaría usar esta integración frecuentemente.	☐	☐	☐	☐	☐
2. Encuentro esta integración innecesariamente compleja.	☐	☐	☐	☐	☐
3. Pienso que esta integración fue fácil de usar.	☐	☐	☐	☐	☐
4. Creo que necesitaría ayuda para poder usar esta integración.	☐	☐	☐	☐	☐
5. Encontré que las diversas funciones de esta integración estaban bien integradas.	☐	☐	☐	☐	☐
6. Pensé que había demasiada inconsistencia en esta integración.	☐	☐	☐	☐	☐
7. Me imagino que la mayoría de la gente aprendería a utilizar esta integración muy rápidamente.	☐	☐	☐	☐	☐
8. Encontré esta integración muy engorrosa / incómoda de usar.	☐	☐	☐	☐	☐
9. Me sentí muy seguro/a al usar esta integración.	☐	☐	☐	☐	☐
10. Necesité aprender muchas cosas antes de poder comenzar a usar la integración.	☐	☐	☐	☐	☐

Proporcione cualquier comentario sobre esta integración: