UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Plataforma integrada para el Diseño UX con interacciones vibrotáctiles

AUTORES: Joaquin Cavenaghi, Lara Fons, Agustin Paturlanne

DIRECTOR/A: Dr. Andrés Rodríguez

CODIRECTOR/A: -

ASESOR/A PROFESIONAL: -

CARRERA: Licenciatura en Sistemas

Resumen

El tacto en UX mejora la interacción en pantallas y dispositivos hápticos, pero faltan herramientas accesibles para diseñadores. Esta tesina propone una extensión en plataformas de diseño UX que facilita la creación y personalización de vibraciones en móviles sin programar. Incluye patrones validados y almacenamiento persistente. Su utilidad fue comprobada en pruebas con usuarios, mejorando el diseño multisensorial.

Palabras Clave

Diseño UX - Interacciones hápticas - Retroalimentación táctil - Freesound - VibViz - Figma - Framer - Plugin de diseño - Prototipado interactivo - Biblioteca de vibraciones - Personalización de efectos - Conversión de audio a vibraciones - Sistema de almacenamiento persistente - Categorización de patrones - Etiquetas descriptivas - Integración de hardware móvil - Pruebas de usabilidad - System Usability Scale (SUS)

Conclusiones

La presente tesina aborda el desafío de integrar interacciones vibrotáctiles en el diseño UX mediante un plugin accesible para diseñadores. La herramienta, migrada de Figma a Framer, permite previsualizar efectos en dispositivos móviles y supera limitaciones técnicas. Las pruebas de usabilidad confirmaron su efectividad, reduciendo barreras y enriqueciendo la experiencia digital. Este trabajo sienta bases para futuras investigaciones en diseño sensorial.

Trabajos Realizados

Se diseñó y desarrolló un plugin para incorporar efectos hápticos en herramientas de diseño UX. Inicialmente creado en Figma, se migró a Framer para permitir la previsualización en dispositivos reales. Se incorporó un catálogo de vibraciones basado en VibViz, soporte para personalización y almacenamiento persistente. Pruebas con usuarios evaluaron su eficacia, logrando una alta puntuación en usabilidad y validando su impacto en la accesibilidad del diseño háptico.

Trabajos Futuros

Se buscará mejorar la conversión de audio a vibraciones, ampliar el catálogo de patrones y optimizar la categorización de efectos. También se evaluará la posibilidad de desarrollar una versión independiente del plugin o integrarlo con otras plataformas de diseño UX para extender su alcance y aplicabilidad.

Fecha de la presentación: Marzo 2025

Plataforma integrada para el Diseño UX con interacciones vibrotáctiles

Tesina de Licenciatura en Sistemas

Joaquin Cavenaghi
Lara Fons
Agustin Paturlanne

Director: Dr. Andrés Rodríguez

Facultad de Informática
Universidad Nacional de La Plata
2025

Agradecimientos

Expresamos nuestro más sincero agradecimiento a quienes nos acompañaron en este camino y contribuyeron a la realización de esta tesina.

A nuestras familias, por su amor, paciencia y apoyo incondicional, fundamentales a lo largo de toda la carrera.

A nuestros amigos, por su compañía y aliento en cada etapa, tanto en los logros como en los desafíos.

A nuestro director de tesis, Andrés Rodríguez, por su guía, dedicación y valiosas contribuciones a este trabajo.

A los profesores de la Facultad de Informática de la Universidad Nacional de La Plata, por su compromiso y enseñanzas, que han sido clave en nuestra formación.

A nuestros compañeros de carrera, por su apoyo y el espíritu de colaboración que enriquecieron esta experiencia académica.

A todos ellos, gracias. Sin su acompañamiento, consejos y confianza, este trabajo no hubiera sido posible. Su apoyo ha sido un pilar fundamental en nuestra formación y en la concreción de esta etapa tan importante.

Índice general

Índice de códigos

Índice de figuras

Resumen

La incorporación de la modalidad del tacto contribuye al diseño de mejores y más ricas Experiencias de Usuario (UX). Tanto los sistemas basados en pantallas que incluyen el tacto como mecanismo de input o proveen alguna retroalimentación vibrotáctil, como los dispositivos corporales que proveen diferentes estímulos hápticos, evidencian esas ventajas. Sin embargo, aún son limitadas las herramientas disponibles para que los diseñadores de UX incluyan en su caja de herramientas la modalidad táctil como material de su trabajo. Se plantea desarrollar en esta tesina una herramienta que permita a los diseñadores concebir, probar y ajustar interacciones vibrotáctiles en aplicaciones destinadas a dispositivos móviles sin necesidad de conocimientos avanzados de programación. Luego de un análisis de los trabajos existentes y los requerimientos del diseño de este tipo de interacciones, se planteó una extensión de las plataformas habituales de uso por los diseñadores de UX para facilitar creación, asignación y personalización de efectos de vibración en elementos de la interfaz. La plataforma incorpora una biblioteca de patrones de vibración validada por expertos en diseño háptico y un sistema de almacenamiento persistente. La solución propuesta fue validada mediante pruebas con usuarios, demostrando su utilidad para el proceso de diseño UX multisensorial en dispositivos móviles.

Capítulo 1

Introducción

El diseño de experiencias de usuario (UX) ha evolucionado notoriamente en las últimas décadas, incorporando nuevas modalidades sensoriales para enriquecer la interacción entre los usuarios y los productos digitales, más allá de la vista y el oído. En particular, la aparición de dispositivos con pantallas capaces de detectar acciones y gestos de las personas facilita la incorporación del tacto a la interacción. Sin embargo, la integración de interacciones vibrotáctiles en las herramientas de diseño sigue presentando desafíos. Actualmente, los diseñadores no cuentan con soluciones que les permitan prototipar y probar experiencias táctiles sin recurrir a desarrolladores para codificar los efectos deseados. Esta limitación no solo restringe la creatividad y la iteración en etapas tempranas del diseño, sino que también dificulta la implementación de experiencias multisensoriales efectivas. Como resultado, la exploración y adopción de estas tecnologías en entornos comerciales y productos digitales sigue siendo limitada. Este escenario constituye un freno a la implementación masiva de interacciones táctiles en aplicaciones móviles.

Surge esta tesina de la necesidad de cerrar esta brecha tecnológica en el diseño de experiencias hápticas mediante el desarrollo de una plataforma integrada que facilite la creación de interacciones vibrotáctiles dentro de uno de los entornos de diseño empleados habitualmente por los diseñadores de UX. El objetivo principal es proporcionar a los diseñadores una herramienta intuitiva que les permita experimentar y ajustar efectos hápticos en tiempo real, sin requerir conocimientos avanzados de programación o hardware adicional. De este modo, se busca ampliar el acceso al diseño háptico y promover su exploración dentro de un entorno familiar y accesible para la comunidad UX.

Con el fin de abordar este problema, se desarrolló un plugin que permite la integración y prueba de efectos vibrotáctiles directamente en una de las herramientas de diseño más utilizadas en la industria. Para ello, se analizaron plataformas y recursos existentes como los frameworks de sistemas operativos en móviles (Android Haptics API [1] e iOS Haptics Framework [2]) y herramientas de creación, edición y organización de efectos vibrotáctiles (VibViz [3] y de Richtap [4]). Se buscó identificar las buenas prácticas en el diseño de interacciones hápticas.

La solución implementada incluye una biblioteca de vibraciones predefinidas clasificadas mediante etiquetas sensoriales que permite la personalización de nuevos efectos. El plugin permite seleccionar uno o más efectos de la biblioteca y agregarlos al diseño en curso para testearlo en el dispositivo móvil asociado. Además, se incorporó un un sistema de persistencia que

facilita guardar y recuperar configuraciones. .

A lo largo del desarrollo, se enfrentaron limitaciones técnicas significativas. Entre ellas, la ejecución restringida de código dentro del entorno de Figma [5] y la falta de soporte para simulaciones hápticas en tiempo real. Como alternativa, se exploró la integración con Framer [6], una plataforma que admite código interactivo lo cual permitió validar y ajustar las experiencias vibrotáctiles en dispositivos reales. Esta combinación de herramientas dio lugar a un flujo de trabajo que favorece la iteración y evaluación de los efectos hápticos en el diseño UX.

Los resultados obtenidos muestran que la propuesta es viable y funcional. Se desarrolló un plugin que permite a los diseñadores explorar, ajustar y probar interacciones vibrotáctiles de forma sencilla e integrada a su flujo de trabajo habitual. La herramienta fue validada mediante pruebas con usuarios, quienes destacaron su facilidad de uso y la utilidad de contar con una solución accesible para experimentar con háptica desde la etapa temprana de diseño.

Además de la presente Introducción, el resto de la tesina se organiza en los siguientes capítulos:

- **Capítulo 2**: Antecedentes y trabajos relacionados
- **Capítulo 3**: Objetivos, metodología y resultados esperados
- **Capítulo 4**: Limitaciones y Resoluciones
- **Capítulo 5**: Desarrollo
- **Capítulo 6**: Pruebas de usuario
- **Capítulo 7**: Conclusiones

Capítulo 2

Antecedentes y trabajos relacionados

2.1. Antecedentes

El diseño de interacción (Interaction Design, IxD) se centra en la creación de productos y servicios interactivos, poniendo énfasis en cómo los usuarios interactúan con ellos. Este enfoque va más allá del simple desarrollo del producto, considerando detalladamente las necesidades, limitaciones y contextos de los usuarios para adaptar el diseño a demandas específicas [7]. Según John Kolko, autor de "Thoughts on Interaction Design", el diseño de interacción implica la creación de un diálogo entre una persona y un producto, sistema o servicio. Este diálogo es tanto físico como emocional y se manifiesta en la interacción entre forma, función y tecnología a lo largo del tiempo [8].

Para abordar este proceso de manera integral, los diseñadores de interacción consideran cinco dimensiones [7]:

1. Palabras (1D): Incluyen el texto, como las etiquetas de los botones, que proporcionan a los usuarios la información necesaria.

2. Representaciones visuales (2D): Elementos gráficos como imágenes, tipografía e íconos que facilitan la interacción del usuario.

3. Objetos físicos/espacio (3D): Medios a través de los cuales los usuarios interactúan con el producto o servicio, como un ordenador portátil mediante un ratón o un teléfono móvil con los dedos.

4. Tiempo (4D): Se refiere a medios que cambian con el tiempo, como animaciones, videos y sonidos.

5. Comportamiento (5D): Cómo las anteriores dimensiones definen las interacciones que un producto permite, es decir, cómo los usuarios pueden realizar acciones en un sitio web o cómo operan un vehículo. También abarca la reacción del producto a las entradas del usuario y la retroalimentación proporcionada.

Aunque el diseño de interacción es fundamental en la UX, esta última abarca un espectro más amplio. El diseño UX se ocupa de todo el recorrido del usuario, incluyendo aspectos como la marca, el diseño, la usabilidad y la funcionalidad. Mientras que los diseñadores de interacción se centran en el momento de uso y en mejorar la experiencia interactiva, los diseñadores UX consideran la experiencia completa del usuario con el producto o servicio [7].

En resumen, el diseño de interacción es esencial para crear productos que no solo sean funcionales, sino que también ofrezcan experiencias significativas y satisfactorias a los usuarios.

En la era digital actual, la narrativa ha evolucionado más allá de las formas tradicionales de contar historias, incorporando tecnologías que enriquecen la experiencia del usuario. Una de estas innovaciones es la retroalimentación háptica, que se refiere a las respuestas táctiles que los dispositivos pueden proporcionar en respuesta a las acciones del usuario [9]. Este tipo de retroalimentación no solo añade un elemento físico a la interacción, sino que también tiene el potencial de profundizar la conexión emocional del usuario con el contenido narrativo [10].

La retroalimentación háptica puede ser vista como una herramienta que complementa los estímulos visuales y auditivos, creando una experiencia multisensorial. Consiste en la activación de diversos mecanismos sensoriales para proporcionar información táctil al usuario, incluyendo vibración, cambios de temperatura, fuerzas resistivas, golpes y texturas simuladas [9]. Al integrar estos efectos en momentos clave de una historia, se intensifica la inmersión del usuario y se facilita una mayor comprensión del contenido. Por ejemplo, un momento dramático en una narrativa puede ir acompañado de una vibración intensa, mientras que una escena más suave puede utilizar patrones más sutiles. Asimismo, una sensación de tensión puede lograrse mediante un aumento en la resistencia al movimiento, y un ambiente frío o cálido puede reforzarse con cambios de temperatura. Esto permite que el usuario no solo vea y escuche la historia, sino que también 'sienta' lo que está sucediendo a través de diferentes estímulos hápticos, lo que puede llevar a una experiencia más rica y memorable [11, 9].

Se ha demostrado que los 'efectos de sensación', definidos como asociaciones explícitas entre frases significativas y patrones hápticos, pueden mejorar significativamente la narrativa. Estas asociaciones permiten que los usuarios experimenten un vínculo más profundo con los eventos narrativos, ya que la retroalimentación táctil puede intensificar el impacto emocional de las historias. Al proporcionar patrones hápticos en momentos clave, se facilita no solo la inmersión, sino también una mejor retención y comprensión del contenido presentado [11].

Además, la retroalimentación háptica tiene el potencial de evocar respuestas emocionales específicas. Al asociar diferentes patrones táctiles con emociones particulares dentro de una historia, se puede guiar al usuario hacia una experiencia emocional más matizada y profunda. Este enfoque no solo es aplicable en el ámbito del entretenimiento, sino que también puede ser beneficioso en contextos educativos y terapéuticos, donde el aprendizaje multisensorial puede potenciar la asimilación de conceptos complejos.

Los hallazgos indican que el uso estratégico de la retroalimentación háptica abre nuevas posibilidades para el diseño narrativo interactivo. Al permitir a los creadores contar historias no solo a través de palabras e imágenes, sino

también mediante sensaciones físicas, se transforma la forma en que los usuarios interactúan con las narrativas contemporáneas. Este enfoque innovador sugiere un futuro prometedor para la narrativa enriquecida por tecnología, donde la conexión emocional y cognitiva del usuario se ve profundamente afectada por las experiencias multisensoriales [11].

Dentro del mundo de diseño de UX, el diseño de experiencias hápticas (HaXD) implica planificar, desarrollar y evaluar interacciones que conectan deliberadamente la tecnología interactiva con sentidos del tacto [12]. Esto ha captado la atención de investigadores interesados en entender cómo las interacciones táctiles pueden enriquecer la UX en entornos digitales y multisensoriales. Sin embargo, el contenido háptico sigue siendo escaso y el conocimiento sobre su diseño es limitado, lo cual representa un obstáculo significativo para los diseñadores que buscan incorporar experiencias hápticas efectivas. En particular, se han propuesto modelos que describen factores esenciales para diseñar y evaluar estas experiencias, brindando a los diseñadores una guía para mejorar la efectividad y satisfacción del usuario en sistemas hápticos.

Uno de los modelos más destacados, desarrollado por Kim y Schneider [13], establece una serie de parámetros de diseño para definir una experiencia háptica ideal. Entre estos se incluyen la *oportunidad* (timing), *densidad* e *intensidad* de los estímulos táctiles, factores que permiten ajustar la percepción del usuario ante respuestas vibrotáctiles o de fricción en dispositivos interactivos [13]. Estos parámetros son esenciales para lograr una integración efectiva con otros elementos sensoriales, como los visuales o auditivos, que en conjunto crean una experiencia más envolvente.

Además de los parámetros de diseño, este modelo enfatiza la importancia de ciertos requisitos de usabilidad, como la *utilidad* y *causalidad* en la experiencia. Estos criterios aseguran que las interacciones hápticas no solo sean percibidas adecuadamente, sino que también transmitan un sentido de coherencia y lógica en el contexto de uso. Esto es crucial para ayudar al usuario a interpretar y responder al estímulo de forma intuitiva [13]. La claridad en estas interacciones permite que el usuario se sienta cómodo y seguro al interactuar con el sistema, lo cual es fundamental para mantener su atención y fomentar su participación activa.

Sin embargo, el diseño de experiencias hápticas en la práctica enfrenta numerosos desafíos debido a la complejidad inherente a combinar estímulos táctiles con otras modalidades sensoriales de manera sincronizada. Los diseñadores deben considerar cuidadosamente factores técnicos y contextuales, como la precisión en el *timing* y las limitaciones del hardware utilizado. Estas consideraciones son vitales para evitar experiencias fragmentadas o inconsistentes que puedan frustrar al usuario o disminuir su inmersión en la narrativa. La falta de sincronización adecuada entre los estímulos táctiles y otros elementos sensoriales puede resultar en una experiencia desarticulada que no logre el impacto emocional deseado.

Estos desafíos resaltan también la necesidad urgente de herramientas de diseño robustas que faciliten el proceso iterativo necesario para crear experiencias hápticas efectivas. Los diseñadores deben poder experimentar con diferentes configuraciones hápticas antes de implementar una versión final del producto. Esto incluye probar diversas combinaciones de patrones táctiles y su relación con otros estímulos sensoriales para encontrar las configuraciones más efectivas [12]. En este sentido, las herramientas de prototipado como Figma, Sketch y Framer desempeñan un papel crucial en el diseño de experiencias interactivas, ya que permiten visualizar, testear y ajustar elementos de la interfaz de usuario antes de su desarrollo final, optimizando así el proceso de iteración y refinamiento.

Figma es una de las plataformas de diseño más utilizadas en la actualidad debido a su enfoque en la colaboración en tiempo real. Al ser una aplicación basada en la web, permite que múltiples diseñadores trabajen simultáneamente en el mismo proyecto, facilitando la iteración rápida y la retroalimentación continua. Una de sus principales ventajas es el sistema de componentes reutilizables, que permite diseñar elementos interactivos como botones y menús de forma modular, asegurando coherencia en toda la interfaz. En el diseño de experiencias hápticas, Figma puede utilizarse para definir la estructura visual de la interfaz y representar visualmente la integración de la retroalimentación táctil, aunque su capacidad para simular efectos hápticos es limitada [5].

Sketch, por otro lado, es una herramienta de diseño vectorial que ha sido ampliamente adoptada en la industria por su enfoque en la creación de interfaces digitales de alta fidelidad. Sketch es una aplicación de escritorio, lo que le otorga mayor flexibilidad en cuanto a rendimiento y personalización. Sus características avanzadas de edición de imágenes y exportación a código permiten que los diseñadores creen prototipos más detallados y listos para ser implementados por desarrolladores. Sin embargo, al igual que Figma, su capacidad para modelar interacciones hápticas de manera directa es limitada, aunque puede utilizarse para definir visualmente cómo y cuándo deben activarse ciertos efectos táctiles dentro de una interfaz [14].

Framer, en contraste con las herramientas anteriores, está diseñado específicamente para la creación de prototipos interactivos con un alto nivel de dinamismo. A diferencia de Figma y Sketch, Framer permite desarrollar interacciones avanzadas utilizando código, lo que lo convierte en una herramienta poderosa para simular efectos hápticos dentro de una experiencia de usuario. Mediante el uso de JavaScript y su API de animaciones, los diseñadores pueden integrar respuestas táctiles teóricas dentro de sus prototipos, como vibraciones simuladas o cambios en la respuesta visual al interactuar con un elemento. Además, Framer facilita el diseño responsivo para múltiples dispositivos, asegurando que los efectos hápticos se adapten a diferentes formatos de pantalla y tipos de hardware [6].

Al aprovechar estas herramientas de prototipado, los diseñadores pueden

definir con mayor precisión cómo la retroalimentación háptica debe integrarse en una interfaz, validando su efectividad en las primeras etapas del desarrollo. Si bien ninguna de estas herramientas permite la simulación física de efectos táctiles, sí ofrecen un marco estructurado para planificar y visualizar la relación entre los estímulos hápticos y otros elementos sensoriales dentro de la experiencia de usuario [12].

Por otro lado, un aspecto crucial identificado en la literatura es la dimensión hedónica de la experiencia háptica; es decir, la capacidad de generar sensaciones que resulten agradables o placenteras para el usuario más allá de su función práctica. El modelo propuesto por Kim y Schneider incluye dimensiones como la *armonía* y *expresividad*, las cuales buscan fomentar una conexión emocional con el usuario. Esto permite lograr interacciones hápticas no solo útiles sino también satisfactorias desde un punto de vista estético [13]. Esta dimensión hedónica es especialmente importante en aplicaciones donde la experiencia sensorial se convierte en el núcleo mismo de la interacción; por ejemplo, en videojuegos o entornos inmersivos como realidad virtual o aumentada.

Adicionalmente, se ha explorado cómo ajustar las vibraciones para sintonizar emociones específicas dentro del contexto narrativo. La investigación sugiere que es posible desarrollar 'manejadores afectivos' para regular las vibraciones según las emociones deseadas durante una experiencia interactiva. Esto implica no solo comprender cómo diferentes patrones táctiles pueden afectar las emociones del usuario sino también cómo estos pueden ser ajustados dinámicamente para optimizar su impacto emocional [10]. Esta capacidad para afinar las vibraciones abre nuevas oportunidades para personalizar experiencias interactivas y mejorar aún más el compromiso emocional del usuario.

La implementación efectiva de estos manejadores afectivos podría transformar radicalmente cómo se diseñan las experiencias narrativas interactivas. Al permitir ajustes precisos basados en respuestas emocionales en tiempo real, los diseñadores podrían crear entornos donde cada interacción esté íntimamente alineada con el estado emocional del usuario. Esto no solo enriquecería las experiencias individuales sino que también podría fomentar un mayor nivel de empatía hacia los personajes y situaciones presentadas dentro de las narrativas.

En resumen, la literatura reciente sobre diseño de experiencias hápticas establece un marco comprensivo que combina parámetros técnicos, requisitos de usabilidad y dimensiones experienciales. Este marco proporciona una base teórica sólida para el desarrollo de interfaces hápticas inmersivas y efectivas. Ayuda a entender mejor las necesidades específicas tanto de los diseñadores como de los usuarios finales mientras enfrenta los desafíos prácticos relacionados con integración y sincronización sensorial en dispositivos interactivos [13, 12]. Este enfoque holístico promete avanzar significativamente en el campo del diseño interactivo al permitir experiencias más ricas e impactantes

tanto a nivel emocional como cognitivo.

2.2. Trabajos relacionados

Se describen a continuación algunos trabajos relacionados con la tesina. Primero se presentan cinco trabajos académicos sobre la definición, diseño, clasificacion y evaluación de patrones táctiles. Luego se describen herramientas y frameworks comerciales disponibles al momento de la escritura de esta tesina.

2.2.1. Parámetros en el diseño de tactones

Los tactones (del inglés tactons, abreviatura de tactile icons) son patrones táctiles estructurados diseñados para transmitir información a través del sentido del tacto, de forma análoga a los íconos visuales o auditivos. Generalmente se implementan mediante estímulos vibrotáctiles y pueden variar en parámetros como duración, intensidad, frecuencia, ritmo y forma de onda para codificar distintos significados perceptibles por el usuario.

Hoggan y Brewster [15] llevaron a cabo un estudio para identificar parámetros efectivos en el diseño de tactones, que son mensajes táctiles estructurados utilizados para comunicar información de manera no visual. Investigaron técnicas como la modulación de amplitud, frecuencia y forma de onda para crear sensaciones de textura en estímulos vibrotáctiles. Los resultados mostraron tasas de reconocimiento del 94 % para la forma de onda, 81 % para la frecuencia y 61 % para la modulación de amplitud, sugiriendo que el uso de diferentes formas de onda es más efectivo para representar texturas táctiles en el diseño de tactones.

2.2.2. VibViz: biblioteca de patrones de vibración

VibViz es una plataforma diseñada para la organización, visualización y navegación de bibliotecas de vibraciones, presentada por Hasti Seifi, Kailun Zhang y Karon E. MacLean en la conferencia IEEE World Haptics en 2015. Su propósito principal es facilitar a diseñadores y desarrolladores la integración de retroalimentación háptica en sus aplicaciones, mejorando así la experiencia del usuario mediante interacciones táctiles más ricas y significativas. A medida que la tecnología avanza, la necesidad de herramientas que permitan gestionar eficazmente las vibraciones se ha vuelto crucial, dado que la retroalimentación háptica puede intensificar la inmersión del usuario y enriquecer la narrativa interactiva.

Funcionamiento y Características

VibViz ofrece un entorno donde los usuarios pueden explorar una variedad de patrones de vibración organizados en bibliotecas. Esta organización permite una navegación intuitiva y eficiente, lo que facilita a los diseñadores seleccionar las vibraciones apropiadas para diferentes interacciones dentro de sus aplicaciones. La plataforma no solo permite la selección de vibraciones predeterminadas, sino que también ofrece la posibilidad de crear y personalizar vibraciones según las necesidades específicas del proyecto. Los usuarios pueden ajustar parámetros como la intensidad, duración y frecuencia de las vibraciones, lo que les proporciona flexibilidad para adaptar las interacciones táctiles a diferentes contextos y emociones. Una característica destacada de VibViz es su enfoque open source, lo que permite a otros desarrolladores acceder al código fuente, modificarlo y adaptarlo a sus necesidades. Esto fomenta un ecosistema colaborativo donde los diseñadores pueden compartir sus propias bibliotecas de vibraciones y contribuir al desarrollo continuo de herramientas hapticas. Además, esta apertura promueve la innovación dentro del campo del diseño haptico, al permitir que diversas comunidades experimenten con nuevas ideas y enfoques. La interfaz de VibViz está diseñada para ser intuitiva, lo que minimiza la curva de aprendizaje para nuevos usuarios. Al permitir la visualización en tiempo real de cómo se comportan las vibraciones dentro del contexto de diseño, VibViz ayuda a los diseñadores a tomar decisiones informadas sobre qué patrones utilizar. Este enfoque práctico es esencial para el proceso iterativo del diseño, donde los cambios pueden ser probados y ajustados rápidamente antes de llegar a una versión final.

Relevancia en el Diseño Háptico

La relevancia de VibViz en el ámbito del diseño háptico radica en su capacidad para transformar cómo se perciben e implementan las interacciones táctiles dentro de las aplicaciones digitales. La retroalimentación háptica se ha demostrado como un componente crítico para mejorar la experiencia del usuario al añadir una dimensión física a las interacciones digitales. Al integrar patrones de vibración en momentos clave durante el uso de una aplicación, se puede aumentar significativamente la inmersión del usuario, facilitando una conexión emocional más profunda con el contenido. Investigaciones previas han mostrado que los efectos sensoriales pueden mejorar notablemente la narrativa interactiva al permitir que los usuarios "sientan"lo que experimentan visualmente [11]. En este sentido, VibViz no solo actúa como una herramienta técnica, sino también como un facilitador para contar historias más efectivas mediante el uso estratégico de patrones hápticos. Al proporcionar un sistema organizado para gestionar estas vibraciones, VibViz permite a los diseñadores experimentar con diferentes configuraciones antes

de implementarlas en sus proyectos finales (ver Figura 2.1). La capacidad para asociar diferentes patrones táctiles con emociones específicas también es un aspecto crucial que VibViz aborda. Al permitir ajustes dinámicos en función del contexto narrativo o emocional deseado, los diseñadores pueden crear experiencias más personalizadas y resonantes para los usuarios. Esto es especialmente relevante en aplicaciones educativas o narrativas donde el aprendizaje multisensorial puede potenciar la comprensión y retención de información [10].

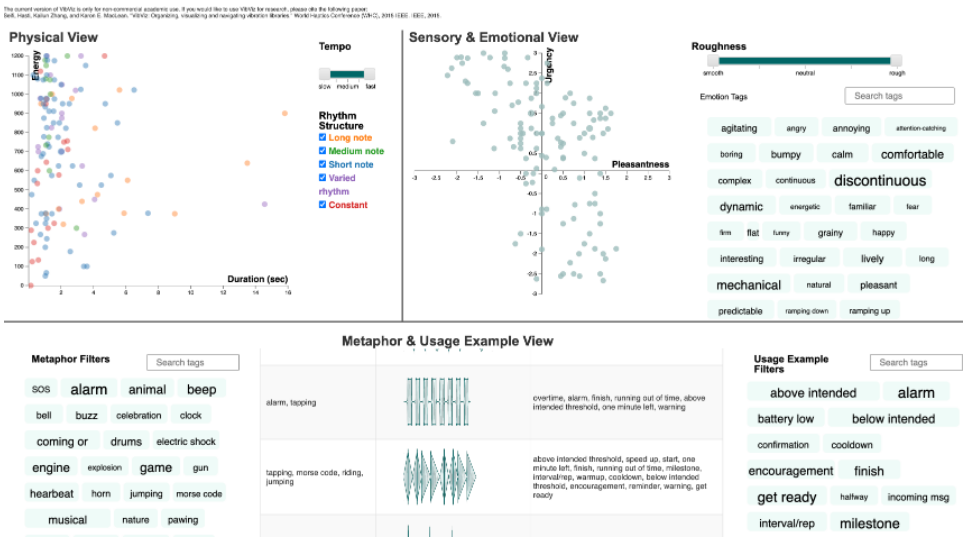

Figura 2.1: Interfaz visual VibViz

En resumen, VibViz representa un avance significativo en el campo del diseño haptico al ofrecer una plataforma accesible para organizar y manipular vibraciones. Su enfoque open source y su capacidad para personalizar interacciones táctiles proporcionan a los diseñadores las herramientas necesarias para enriquecer sus proyectos con retroalimentación háptica efectiva. La combinación de teoría sólida sobre retroalimentación háptica con una implementación práctica hace que VibViz sea una herramienta indispensable para aquellos interesados en mejorar sus diseños mediante interacciones táctiles innovadoras [3].

2.2.3. Macaron: un editor web para efectos vibrotáctiles

Schneider y MacLean [16] desarrollaron Macaron, un editor de efectos vibrotáctiles basado en la web que permite a los diseñadores crear, editar y compartir efectos hápticos de manera intuitiva. La herramienta incorpora ejemplos visualizados y editables, facilitando el proceso de diseño y permitiendo a los usuarios comprender y manipular parámetros vibrotáctiles sin necesidad de conocimientos técnicos avanzados. Este enfoque democratiza el

diseño háptico y promueve la experimentación y colaboración en la creación de experiencias táctiles.

2.2.4. Evaluación de diseños de mapeo para transmitir datos a través de tactones

Ferguson, Williamson y Brewster [17] exploraron cómo diferentes diseños de mapeo de datos pueden influir en la percepción y efectividad de los tactones. Mediante un experimento de estimación de magnitud, investigaron cómo parámetros vibrotáctiles como el tempo y la duración pueden transmitir información relacionada con variables como el error y el peligro. Los resultados indicaron que tanto el tempo como la duración se perciben de manera consistente al representar datos, proporcionando información valiosa para el diseño de tactones más intuitivos y efectivos.

2.2.5. Diseño de iconos hápticos rítmicos para dispositivos móviles

Ternes y MacLean [18] investigaron el uso del ritmo como parámetro principal en el diseño de iconos hápticos, conocidos como tactones, para dispositivos móviles. Su estudio demostró que el ritmo es un parámetro efectivo para diseñar conjuntos amplios de iconos hápticos que los usuarios pueden aprender y distinguir con facilidad. Este trabajo destaca la importancia del ritmo en la creación de vocabularios hápticos ricos y accesibles en interfaces móviles. Estos estudios académicos han contribuido significativamente al avance en el diseño de interacciones vibrotáctiles, proporcionando nuevas perspectivas y herramientas que facilitan la creación y evaluación de experiencias hápticas efectivas.

2.2.6. Android Haptics API

Android proporciona un marco versátil para la implementación de retroalimentación háptica que abarca desde efectos predefinidos simples hasta patrones personalizados diseñados para enriquecer la experiencia del usuario. Este apartado explora las capacidades, mejores prácticas y recomendaciones técnicas de la API de Android para la creación de efectos hápticos.

Clasificaciones de efectos hápticos

La API de Android organiza los efectos hápticos en tres categorías principales, cada una diseñada para cumplir con necesidades específicas:

- **Clear haptics**: Efectos discretos asociados a eventos puntuales, como la presión de un botón. Son predefinidos, consistentes y adecuados para

aplicaciones que requieren una interacción básica. Por ejemplo, los desarrolladores pueden utilizar constantes de `HapticFeedbackConstants` para representar este tipo de interacción.

- **Rich haptics**: Requieren actuadores hápticos avanzados con mayor ancho de banda para ofrecer una experiencia más expresiva, como simular texturas o movimientos complejos. Estos efectos ofrecen inmersión adicional, pero no todos los dispositivos son compatibles, por lo que es esencial implementar estrategias de respaldo.
- **Buzzy haptics**: Se caracterizan por vibraciones intensas y penetrantes que pueden generar sensaciones residuales en el usuario. Son útiles para eventos que demandan atención inmediata, como alarmas o notificaciones críticas.

Uso de efectos predefinidos

La API de Android ofrece una serie de efectos predefinidos a través de `VibrationEffect` y `HapticFeedbackConstants`, lo que garantiza una experiencia consistente en una variedad de dispositivos. Estos efectos se pueden implementar de manera sencilla, como se muestra en el siguiente ejemplo:

Código 2.1 Implementación de un efecto predefinido

```
1 Vibrator vibrator = context.getSystemService(Vibrator.class);
2 vibrator.vibrate(VibrationEffect.createPredefined(VibrationEffect.
      EFFECT_CLICK));
```

Además, los desarrolladores pueden verificar la compatibilidad del dispositivo con métodos como `Vibrator.areEffectsSupported` y `Vibrator.hasAmplitudeControl`, asegurando que los efectos seleccionados sean adecuados para el hardware.

Creación de efectos personalizados

La flexibilidad de la API permite crear patrones hápticos personalizados mediante primitivas definidas en `VibrationEffect.Composition`. Esto permite diseñar efectos específicos adaptados a las necesidades del usuario:

Código 2.2 Ejemplo de un patrón háptico personalizado

```
1 int delayMs = 100;
2 vibrator.vibrate(
3     VibrationEffect.startComposition()
```

```
4           .addPrimitive(VibrationEffect.Composition.PRIMITIVE_SPIN, 0.8f
    )
5           .addPrimitive(VibrationEffect.Composition.PRIMITIVE_CLICK, 1.0
    f, delayMs)
6           .compose());
```

Al trabajar con patrones personalizados, es importante considerar:

- **Compatibilidad del hardware**: Verificar la disponibilidad de primitivas en el dispositivo mediante `Vibrator.arePrimitivesSupported`.
- **Duración y amplitud**: Diseñar patrones con transiciones suaves para evitar efectos desagradables o ruidos no deseados.
- **Fallback**: Proveer patrones alternativos para dispositivos sin soporte completo.

Mejores prácticas y limitaciones

- Utilizar los efectos hápticos predefinidos por el sistema operativo siempre que sea posible para garantizar una experiencia de usuario consistente.
- Evitar el uso de métodos obsoletos como `createOneShot`, que no se alinean con los principios modernos de diseño háptico.
- Considerar la sincronización precisa entre estímulos hápticos y otros elementos de la interfaz, como audio y gráficos, para una experiencia inmersiva.

[1].

2.2.7. iOS Haptics Framework

El marco *Core Haptics* de iOS permite a los desarrolladores integrar retroalimentación háptica en sus aplicaciones, ofreciendo un control detallado sobre la intensidad, duración y sincronización de los efectos. Este apartado detalla las capacidades, componentes clave y ejemplos prácticos del uso de *Core Haptics*.

Capacidades integradas

Los dispositivos iOS compatibles ofrecen una gama de patrones hápticos predefinidos que se integran automáticamente en componentes como *switches*, *sliders* y *pickers*. Estos patrones aseguran una experiencia coherente para los usuarios, mientras que los desarrolladores pueden concentrarse en personalizar interacciones clave.

Creación de efectos personalizados

Para diseñar patrones hápticos personalizados, los desarrolladores deben usar `CHHapticEngine`, que conecta la aplicación con el hardware del dispositivo. El siguiente ejemplo muestra cómo inicializar un motor háptico y reproducir un patrón personalizado:

Código 2.3 Fragmento de inicialización de un motor háptico

```
let hapticEngine = try CHHapticEngine()
try hapticEngine.start()

let pattern = try CHHapticPattern(events: [event], parameters: [])
let player = try hapticEngine.makePlayer(with: pattern)
try player.start(atTime: 0)
```

El marco soporta dos tipos principales de eventos:

- **Eventos transitorios**: Representan pulsos breves, ideales para retroalimentación inmediata.
- **Eventos continuos**: Generan vibraciones sostenidas, como las que acompañan ringtones o efectos prolongados.

Formato AHAP

Los patrones hápticos también se pueden definir en archivos `.ahap`, un formato basado en JSON que simplifica la reutilización y el intercambio de efectos. Estos archivos contienen descripciones jerárquicas de eventos y parámetros, lo que facilita su gestión y edición.

Mejores prácticas

- Diseñar patrones coherentes y asociarlos claramente con acciones específicas dentro de la aplicación.
- Evitar el uso excesivo de retroalimentación háptica para prevenir la fatiga del usuario.
- Complementar los patrones hápticos con estímulos visuales y auditivos para una experiencia más inmersiva.

El marco *Core Haptics* también proporciona herramientas avanzadas para ajustar parámetros dinámicos como intensidad y nitidez, permitiendo una personalización precisa[2].

2.2.8. RichTap: un ecosistema comercial para háptica móvil

RichTap es una solución comercial desarrollada por la empresa AAC Technologies que ofrece un ecosistema completo para el diseño y prueba de experiencias vibrotáctiles en dispositivos móviles. Esta plataforma incluye motores hápticos avanzados, una biblioteca de efectos predefinidos y herramientas de edición diseñadas para facilitar la creación de interacciones táctiles realistas. Uno de sus principales aportes es la disponibilidad de un SDK y un motor de reproducción de alta fidelidad, lo que permite a desarrolladores y diseñadores integrar efectos hápticos sin necesidad de profundos conocimientos técnicos. Si bien su enfoque está orientado a la industria móvil, RichTap demuestra la viabilidad de ofrecer herramientas accesibles para el diseño háptico, aunque su naturaleza propietaria y su dependencia de hardware específico limitan su adopción en contextos académicos o de prototipado rápido [4].

2.2.9. Interhaptics: diseño multiplataforma de experiencias hápticas

Interhaptics es una plataforma que permite diseñar, editar y reproducir efectos hápticos en dispositivos con capacidades táctiles avanzadas, como controladores de realidad virtual o dispositivos móviles. Su editor visual facilita la creación de efectos hápticos personalizados y su SDK ofrece soporte para motores hápticos de distintas marcas. Una de sus principales características es la posibilidad de visualizar y ajustar curvas de vibración y parámetros como la frecuencia o la intensidad, lo que brinda un control fino sobre la experiencia táctil. Además, Interhaptics permite exportar efectos hápticos en distintos formatos, lo que facilita su integración en motores de juegos o aplicaciones móviles. Esta herramienta representa un modelo avanzado y accesible para el diseño háptico, aunque su orientación técnica puede representar una barrera para diseñadores sin experiencia en programación [19].

2.3. Investigaciones preliminares

Como parte de la investigación de antecedentes se analizaron los desafíos que plantea la creación de extensiones para dos de las herramientas más utilizadas por los diseñadores de UX (Figma[1] y Framer[2]) y la posibilidad de utilizar una plataforma de recopilación y reutilización de archivos de audio (FreeSound[3]) que aproveche el formato de patrones vibrotáctiles que provee VibViz [3].

[1] https://www.figma.com/es-la/
[2] https://www.framer.com/
[3] https://freesound.org/

2.3.1. Figma

Creación de un plugin

El desarrollo de un plugin en Figma implica un conocimiento detallado de su API y del entorno en el cual estos plugins interactúan con el editor. En esta sección, se analizan los aspectos técnicos y estructurales fundamentales para implementar funcionalidades avanzadas en Figma, desde la creación de interfaces de usuario hasta la manipulación del documento de diseño a través de su jerarquía de nodos.

Los plugins en Figma están basados en JavaScript y HTML, permitiendo que los desarrolladores extiendan la funcionalidad del editor mediante una API dedicada [20]. El entorno de ejecución de los plugins, o *sandbox*, proporciona un espacio seguro y optimizado donde el código del plugin se ejecuta en un hilo principal que no tiene acceso directo a las API del navegador, lo cual garantiza seguridad y rendimiento.

Para ilustrar este concepto, la Figura 2.2 muestra el modelo de comunicación entre el hilo principal y el *iframe* que contiene la interfaz de usuario. La comunicación entre el hilo principal y el *iframe* se realiza mediante el intercambio de mensajes, usando métodos como ‘postMessage‘ y ‘onmessage‘, los cuales se detallan a continuación.

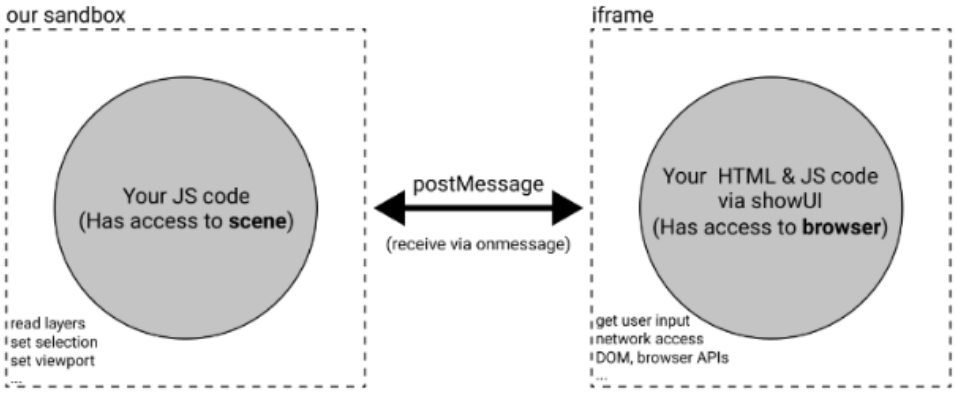

Figura 2.2: Modelo de comunicación entre el hilo principal y el iframe en el entorno de plugin de Figma.

Creación de la Interfaz de Usuario

Para mostrar una interfaz personalizada dentro de Figma, se utiliza la función ’figma.showUI()’. La configuración básica para la interfaz requiere tres pasos fundamentales [21]:

1. Crear un archivo HTML con el contenido de la interfaz.

2. Modificar el archivo *manifest.json* del plugin, indicando el archivo de la interfaz HTML.

3. Llamar a figma.showUI(html) desde el archivo principal del plugin ('code.ts'), como se muestra en el siguiente código:

```
1         figma.showUI(__html__);
2
```

Al implementar esta configuración, el contenido de 'ui.html' se muestra dentro del *iframe* de Figma, permitiendo que el usuario interactúe con la interfaz sin salir del editor.

Uno de los aspectos más importantes es la comunicación entre el código del plugin y la interfaz de usuario. Para enviar mensajes desde el *iframe* (HTML) al hilo principal del plugin, se usa el siguiente código en JavaScript:

```
1 <script>
2     parent.postMessage({ pluginMessage: 'mensaje' }, '*');
3 </script>
```

En el hilo principal, este mensaje se recibe mediante la función 'onmessage', que captura la información enviada desde la interfaz:

```
1 figma.ui.onmessage = (message) => {
2     console.log("Mensaje recibido desde la UI:", message);
3 };
```

De manera similar, el hilo principal puede enviar mensajes a la interfaz utilizando 'postMessage', y estos mensajes se reciben en la interfaz HTML como se muestra a continuación:

```
1 figma.ui.postMessage("mensaje desde el plugin");
```

```
<script>
    onmessage = (event) => {
        console.log("Mensaje recibido desde el plugin:",
        event.data.pluginMessage);
    }
</script>
```

Este sistema de intercambio de mensajes permite actualizar dinámicamente elementos de la interfaz o enviar información procesada, facilitando una interacción fluida entre la interfaz y el código del plugin.

Estructura de Documentos en Figma

La API de Figma organiza los archivos de diseño como árboles de nodos, donde cada elemento visual representa un nodo en la jerarquía. El nodo raíz es el 'DocumentNode', y de él derivan múltiples 'PageNodes', uno para cada página del archivo.

- **DocumentNode**: representa la raíz del archivo de Figma.
- **PageNode**: cada página del archivo es un 'PageNode' descendiente del 'DocumentNode'.

La Figura 2.3 ilustra esta estructura jerárquica, donde cada 'PageNode' contiene un conjunto de nodos que representan capas, grupos y otros elementos del diseño. Esta organización permite recorrer el documento de forma programática, accediendo a las capas y manipulando elementos específicos según las necesidades del plugin [22].

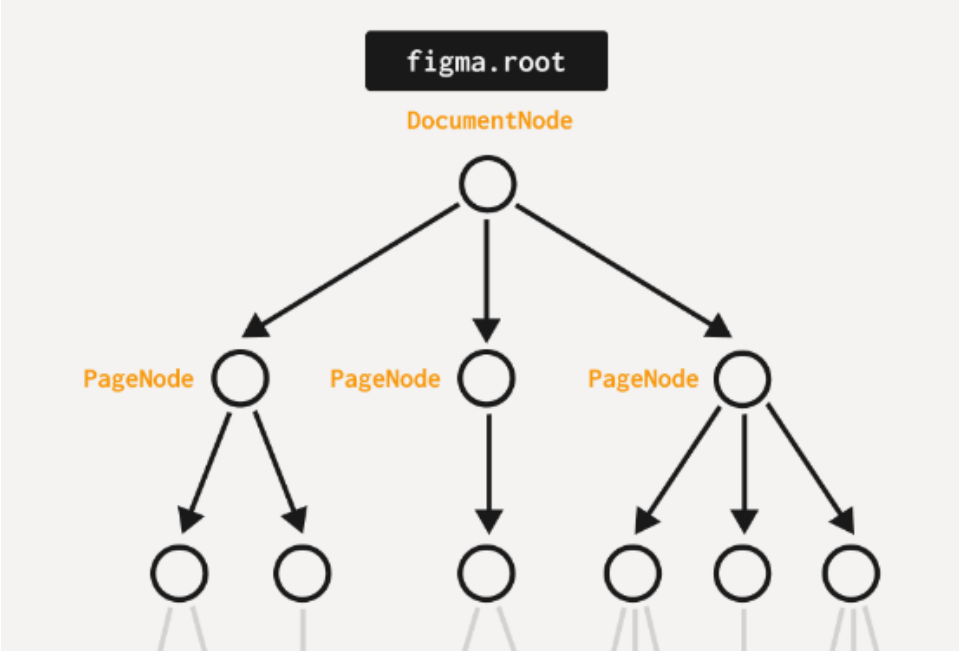

Figura 2.3: Estructura jerárquica de nodos en un archivo de Figma.

2.3.2. Framer

Framer es una herramienta poderosa y versátil orientada al diseño de interfaces interactivas, prototipos y experiencias de usuario altamente dinámicas. A diferencia de otras plataformas que se centran principalmente en el diseño visual estático y colaborativo, Framer permite integrar código directamente en los prototipos, lo que habilita funcionalidades avanzadas como la ejecución de lógica personalizada y la interacción con hardware o sistemas externos. Esta capacidad lo convierte en una solución ideal para proyectos que requieren un nivel de interactividad más profundo

Integración de Código

Framer permite a los usuarios escribir código en JavaScript, lo que brinda un nivel adicional de personalización y control sobre las interacciones dentro del prototipo. Esta capacidad es especialmente útil para aquellos que buscan implementar lógica compleja o integrar APIs externas en sus diseños. La posibilidad de ejecutar código directamente dentro del entorno de diseño facilita un flujo de trabajo más eficiente, ya que los diseñadores pueden experimentar con diferentes funcionalidades sin necesidad de pasar por un proceso de desarrollo separado.[23]

Uso en el Diseño UX

Debido a su capacidad para crear prototipos interactivos que reflejan con precisión la experiencia final del usuario, Framer se ha convertido en una herramienta popular entre los diseñadores UX. Al permitir la integración de animaciones y lógica personalizada, Framer ayuda a los diseñadores a comunicar sus ideas más efectivamente a los desarrolladores y partes interesadas. Esto es crucial en un entorno donde la colaboración entre equipos es fundamental para el éxito del producto final.

Además, Framer ha sido adoptado por muchas empresas como parte de su flujo de trabajo estándar para el diseño UX, gracias a su capacidad para facilitar pruebas rápidas e iteraciones basadas en la retroalimentación del usuario. Esto permite a los equipos ajustar sus diseños en función de las necesidades reales del usuario antes de pasar a la fase de desarrollo. [6]

2.3.3. FreeSound

FreeSound es una plataforma colaborativa diseñada para la recopilación, organización y reutilización de fragmentos de sonido, grabaciones y otros archivos de audio, los cuales se distribuyen bajo licencias Creative Commons. Esto permite a los usuarios de la plataforma acceder y utilizar los sonidos para diversos fines, promoviendo una cultura de reutilización y colaboración en el ámbito del audio [24].

Funcionalidades de la API de FreeSound

FreeSound proporciona una API robusta, que permite a los desarrolladores interactuar con su base de datos de sonidos de múltiples maneras. Las funcionalidades clave incluyen la posibilidad de explorar, buscar y recuperar sonidos, además de extraer características de los archivos de audio y realizar consultas avanzadas basadas en análisis de contenido y metadatos (como etiquetas). Adicionalmente, esta API permite registrar usuarios, cargar nuevos sonidos, así como calificar y comentar audios existentes [24].

La API de FreeSound facilita el almacenamiento y administración de sonidos personalizados, permitiendo a los usuarios subir sus propios audios y recuperarlos cuando deseen. La API también soporta la autenticación mediante OAuth2, lo cual permite implementar un sistema seguro y personalizado de acceso y gestión de archivos.

Autenticación OAuth2 en FreeSound

FreeSound ofrece dos métodos de autenticación: autenticación basada en token y OAuth2. La autenticación mediante OAuth2 sigue el flujo de concesión de código de autorización definido en el estándar RFC6749, lo

cual permite que los usuarios inicien sesión en FreeSound desde aplicaciones externas y autoricen las mismas a gestionar sus sonidos [25].

El flujo de autenticación implica los siguientes pasos:

1. El usuario es redirigido desde la aplicación externa a una página de autenticación en FreeSound, donde concede permiso para acceder a su cuenta (ver Figura 2.4).

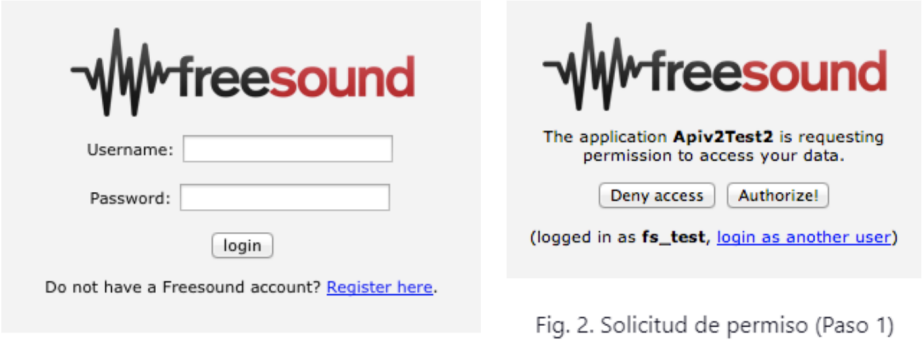

Fig. 1. Página de inicio de sesión (Paso 1)

Fig. 2. Solicitud de permiso (Paso 1)

Figura 2.4: Login y autorización

2. Tras la autorización, el usuario es redirigido a una URL con un código de autorización (ver Figura 2.5). Este código debe pegarse en la aplicación externa.

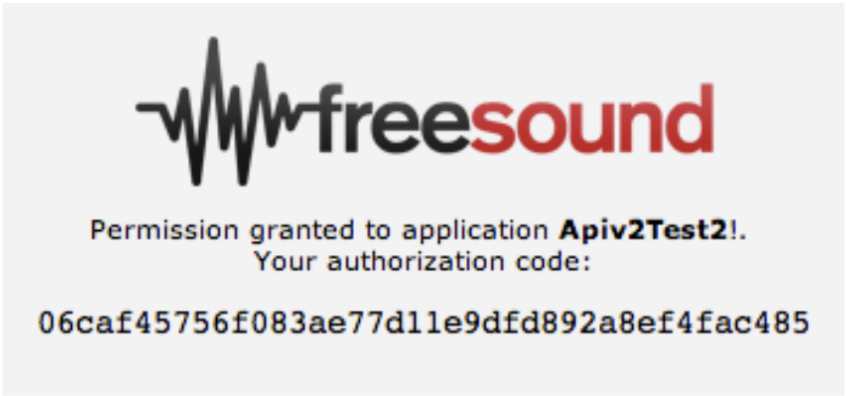

Fig 3. Using Freesound as redirect target (Step 2)

Figura 2.5: Código generado por Freesound

3. Finalmente, este código se intercambia por un token de acceso que permite a la aplicación externa interactuar con la API en nombre del usuario.

Este token de acceso tiene una validez de 24 horas. Además, la API proporciona un *refresh token* que permite obtener un nuevo token de acceso sin necesidad de repetir el flujo de autenticación completo, garantizando una experiencia continua para el usuario, sin necesidad de que el mismo deba ingresar sus credenciales cuando se vence el token [25].

Subida y Gestión de Archivos .wav

Una de las funcionalidades principales de FreeSound es la capacidad de subir archivos de audio. Para ello, existe el endpoint `POST /apiv2/sounds/upload/`, el cual permite a los usuarios cargar sus archivos .wav, proporcionando parámetros adicionales como una descripción del sonido, etiquetas y una licencia seleccionada. Estos detalles ayudan a organizar los sonidos en la plataforma, facilitando su recuperación y uso posterior [26].

Para subir un archivo, la solicitud debe incluir los siguientes parámetros:

- **audiofile**: el archivo de audio en formato admitido (.wav, .aif, .flac, .ogg o .mp3).
- **tags**: etiquetas descriptivas del sonido.
- **description**: una descripción textual del archivo de audio.
- **license**: la licencia seleccionada para el sonido (por ejemplo, "Atribución" o "Creative Commons 0").

Estos sonidos pasan luego por un proceso de descripción, procesamiento y moderación, en el cual el equipo de FreeSound verifica la conformidad del contenido antes de hacerlo accesible públicamente. Esta moderación puede llevar algunas horas o días. [27].

Relevancia en el Diseño Háptico

FreeSound, al permitir que los usuarios gestionen sus propios sonidos habilita un mayor grado de personalización. La posibilidad de etiquetar y describir los sonidos subidos facilita la organización y recuperación de estos recursos, permitiendo a los diseñadores iterar rápidamente en sus proyectos y experimentar con diversas configuraciones de audio [28].

Capítulo 3

Objetivos, metodología y resultados esperados

3.1. Objetivo General

El objetivo principal de esta tesis es desarrollar un plugin háptico que permita a los diseñadores integrar y probar interacciones vibrotáctiles de manera intuitiva y eficiente durante el proceso de diseño. Este plugin busca resolver una problemática común en el flujo de trabajo actual, donde los diseñadores no cuentan con herramientas para incorporar vibraciones directamente en sus prototipos. En su lugar, suelen delegar esta tarea a los programadores, lo que limita la iteración y experimentación en las etapas iniciales del diseño. La solución propuesta tiene como meta ofrecer una herramienta que simplifique este proceso, permitiendo a los diseñadores trabajar de manera autónoma y explorar nuevas posibilidades creativas en el ámbito del diseño háptico.

3.2. Objetivos Específicos

Para alcanzar el objetivo general, se plantean los siguientes objetivos específicos:

- **Investigación Teórica sobre Interacciones Hápticas**: Llevar a cabo una revisión de literatura relacionada con el diseño háptico y las interacciones vibrotáctiles. Esta investigación servirá como base para fundamentar las decisiones técnicas y conceptuales del desarrollo del plugin, asegurando que esté alineado con las mejores prácticas y conocimientos actuales en el campo.
- **Desarrollo de un Plugin Intuitivo**: Diseñar e implementar un plugin que ofrezca una biblioteca de vibraciones predefinidas, basadas en el catálogo de VibViz, junto con la funcionalidad para que los usuarios puedan agregar y personalizar sus propios sonidos. Este listado debe ser fácilmente accesible y organizado mediante etiquetas (*metáforas*) descriptivas que permitan identificar rápidamente las vibraciones según su propósito o contexto.
- **Edición y Personalización de Metáforas**: Incorporar herramientas dentro del plugin que permitan a los diseñadores modificar las metáforas asociadas a cada vibración. Esto permitirá que cada usuario ajuste

las descripciones para reflejar mejor su interpretación personal de las vibraciones, facilitando su uso en contextos específicos según las necesidades del usuario.

- **Integración con Botones Interactivos**: Implementar una funcionalidad que permita a los diseñadores añadir vibraciones a elementos interactivos (como botones) dentro de sus prototipos. Esto incluye traducir ondas sonoras predefinidas o personalizadas a patrones vibrotáctiles equivalentes, asegurando que la respuesta háptica sea coherente con la intención del diseño.
- **Metodología Incremental Basada en Control de Versiones**: Adoptar un enfoque incremental para el desarrollo del plugin, utilizando sistemas de control de versiones para gestionar el código fuente. Esto permitirá realizar mejoras continuas al plugin, mantener un registro claro del progreso y garantizar la estabilidad del proyecto durante todas sus etapas.

3.3. Metodología

Para alcanzar los objetivos planteados, se seguirá una metodología basada en los siguientes pasos:

- **Revisión Bibliográfica**: Se realizará un análisis detallado de artículos académicos relacionados con interacciones hápticas y vibrotáctiles para comprender mejor sus aplicaciones prácticas y teóricas.
- **Diseño e Implementación Iterativa**: El desarrollo del plugin seguirá un enfoque incremental basado en ciclos iterativos. Cada iteración incluirá fases de diseño, implementación y prueba para garantizar que las funcionalidades cumplan con los requisitos establecidos.
- **Uso de Herramientas Específicas**: Se utilizarán plataformas como Figma y Framer para desarrollar e integrar el plugin. Además, se emplearán repositorios para gestionar el código fuente y documentar los avances realizados.
- **Pruebas con Usuarios**: Una vez implementadas las funcionalidades principales, se realizarán pruebas con diseñadores reales para evaluar la usabilidad e identificar posibles mejoras o ajustes necesarios.
- **Documentación Continua**: Durante todo el proceso se mantendrá una documentación detallada sobre las decisiones técnicas tomadas, los problemas encontrados y las soluciones implementadas.

3.4. Resultados Esperados

Se espera obtener los siguientes resultados al finalizar esta tesis:

1. Un plugin funcional para plataforma de diseño UX que permita gestionar bibliotecas de vibraciones predefinidas y personalizadas, editar metáforas asociadas y facilitar la incorporación de sonidos propios.
2. Una interfaz intuitiva que facilite a los diseñadores agregar vibraciones a elementos interactivos dentro de sus prototipos sin necesidad de conocimientos técnicos avanzados.
3. Un flujo de trabajo optimizado dentro de la plataforma de diseño UX eliminando la necesidad de pasos adicionales entre plataformas.
4. Un mayor conocimiento teórico sobre interacciones vibrotáctiles y su impacto en la experiencia del usuario, así como una comprensión más profunda sobre el uso práctico de herramientas como Figma y Framer.
5. Una contribución significativa al campo del diseño háptico mediante la creación de una herramienta accesible e innovadora que fomente la experimentación creativa entre diseñadores.

Capítulo 4

Limitaciones y Resoluciones

Durante el desarrollo de la plataforma integrada para diseño UX con interacciones vibrotáctiles, se enfrentaron varias limitaciones que influyeron tanto en las decisiones tecnológicas como en la implementación del plugin. A continuación, se detallan las principales restricciones encontradas, las razones detrás de cada una y las soluciones adoptadas para mitigar su impacto.

4.1. Limitación 1: Elección de la Plataforma de Desarrollo del Plugin

Inicialmente, se evaluaron varias plataformas para desarrollar el plugin, considerando tanto Figma como Framer. Aunque Framer permite una mayor flexibilidad para ejecutar código directamente en sus proyectos, al momento de esta investigación no ofrecía soporte para la creación de plugins personalizados. Por este motivo, se optó por Figma al comenzar, una de las plataformas más populares y ampliamente utilizadas en el diseño UX, que proporciona una API específica para el desarrollo de plugins [20].

4.1.1. Ventajas de Usar Figma

Figma ofreció múltiples ventajas para nuestro desarrollo:

- Amplia adopción por parte de la comunidad de diseñadores, lo que asegura que la solución se integre en flujos de trabajo existentes.
- Soporte robusto para personalización mediante su API, incluyendo el acceso estructurado al árbol de nodos de los documentos [22].

4.1.2. Desventajas de Usar Figma

Sin embargo, el uso de Figma presentó ciertas limitaciones:

- No permite la ejecución de código directamente, lo que restringió las posibilidades de simular las vibraciones en el entorno de diseño.
- La API no incluye herramientas gratuitas para manejar archivos de audio o convertirlos a formatos compatibles con vibraciones.

Estas restricciones han llevado a implementar soluciones complementarias, como se explica en las siguientes secciones.

4.2. Limitación 2: Restricciones para Ejecutar Código en Figma

Figma no permite la ejecución de código directamente en su entorno, lo que imposibilitó incorporar la simulación de vibraciones durante la etapa de diseño. Esta restricción impacta negativamente en la capacidad de los diseñadores para iterar rápidamente y probar las interacciones vibrotáctiles sin depender de herramientas externas.

La ausencia de esta funcionalidad esencial plantea una dificultad significativa: los diseñadores no pueden verificar en tiempo real cómo las vibraciones afectarán la experiencia del usuario. Esto introduce un riesgo en el flujo de trabajo, ya que las vibraciones solo pueden validarse después de que el diseño haya sido implementado en una etapa posterior, dificultando la iteración rápida y aumentando los costos de desarrollo en caso de ajustes.

4.2.1. Solución: Integración con Framer

Para superar esta limitación, la etapa de prueba interactiva se trasladó a Framer, en donde si es posible la ejecución de código dentro de prototipos interactivos. Este enfoque permitió mantener el proceso de diseño en Figma mientras se aprovechaban las capacidades avanzadas de Framer para validar las vibraciones.

El flujo de trabajo final quedó estructurado de la siguiente manera:

- **Configuración en Figma:** Los diseñadores utilizan el *plugin-haptico* para crear, seleccionar y asignar patrones de vibración a elementos del diseño.
- **Exportación a Framer:** Una vez completado el diseño, este se exporta a Framer siguiendo un conjunto de instrucciones definidas.
- **Prueba en Framer:** En Framer, las vibraciones se activan según los eventos configurados, permitiendo a los diseñadores experimentar en tiempo real con un dispositivo móvil.

Aunque este proceso introduce un paso adicional en el flujo de trabajo, proporciona una solución funcional que permite validar las decisiones de diseño antes de la implementación final. Esto no solo asegura la calidad de las interacciones vibrotáctiles, sino que también reduce la posibilidad de errores o malentendidos durante la etapa de desarrollo.

4.2.2. Impacto en el Proceso de Diseño

El uso combinado de Figma y Framer equilibra las limitaciones técnicas de cada herramienta. Este enfoque asegura que los diseñadores puedan ajus-

tar sus prototipos basándose en pruebas reales, optimizando así la calidad del producto final. .

4.3. Limitación 3: Conversión de Archivos de Audio

Durante la etapa de desarrollo, no existía una herramienta integrada que permitiera convertir archivos de audio en los arreglos de datos necesarios para las vibraciones. Este proceso es esencial para mapear correctamente las vibraciones con los patrones establecidos, pero la falta de herramientas automatizadas aumentó la complejidad del desarrollo.

4.3.1. Solución: Implementación de Lógica Manual

Para resolver este problema, se desarrolló un script manual que convierte los archivos de audio a arreglos de datos. Aunque este enfoque es menos eficiente, permitió garantizar la funcionalidad básica del sistema. Además, se dejó abierta la posibilidad de integrar herramientas más avanzadas en el futuro a medida que la tecnología evolucione.

4.4. Limitación 4: Consultas bloqueadas desde entorno aislado de Figma

Al desarrollar el plugin para Figma, se presenta un problema relacionado con las restricciones de seguridad impuestas por el entorno aislado en el que se ejecutan los plugins. En particular, al intentar realizar una solicitud HTTP hacia una API externa para obtener recursos, como archivos de audio, se observa el siguiente error:

> Access to fetch at '...' from origin 'https://www.figma.com' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.

Este error surge debido a la política de Cross-Origin Resource Sharing (CORS), que restringe las solicitudes realizadas desde orígenes no autorizados. En el caso de los plugins de Figma, estas solicitudes provienen de un entorno restringido donde el valor de la cabecera Origin es 'null'. Como resultado, muchas APIs externas, que no aceptan este origen, bloquean las solicitudes.

Al investigar esta problemática, se hallaron comentarios de desarrolladores de la comunidad de Figma que confirmaban esta limitación. Dos enfoques destacados fueron los siguientes:

Desarrollador 1: *El plugin se ejecuta en un navegador web restringido que sólo puede aceptar origin: 'null' en la cabecera de la petición. Algunas APIs bloquean este tipo de cabecera, lo que produce un error. Una solución es llamar a un proxy CORS que acepte la cabecera origin: 'null' en lugar de llamar directamente a la API de destino.*

Desarrollador 2: *El servicio que está utilizando debe permitir todos los dominios de origen mediante Access-Control-Allow-Origin: '*' para completar esta solicitud. Puede probar a utilizar servicios proxy CORS si no le preocupa la privacidad de los datos.*

Ambos enfoques resaltan la dificultad inherente a trabajar con las políticas de CORS en un entorno restringido y sugieren el uso de un servicio proxy que actúe como intermediario entre el plugin y el recurso solicitado.

4.4.1. Solución: Proxy con plataforma ajena

Para resolver esta limitación, se implementó un proxy que actúa como intermediario para realizar las solicitudes HTTP. Se optó por utilizar la plataforma CORS Anywhere, un servicio público que reenvía las solicitudes HTTP y añade las cabeceras necesarias para sortear las restricciones de CORS.

El proxy intercepta las solicitudes del plugin y las reenvía al servidor destino con las cabeceras requeridas, modificando el valor de Origin a un formato aceptable. Este enfoque nos permitió continuar trabajando con APIs que de otro modo habrían rechazado las solicitudes provenientes del plugin.

A continuación, se presenta un fragmento del código utilizado para implementar esta solución:

Código 4.1 Parte del codigo Javascript

```
async function getSoundToArray(soundUrl) {
    const header = new Headers();
    header.append("Authorization", "Bearer " + t0k3n);
    // Agregar el X-Requested-With header para satisfacer
    // los requisitos de CORS Anywhere
    header.append("X-Requested-With", "XMLHttpRequest");

    const requestOptions = {
      method: "GET",
      headers: header,
    };

    // Se usa el proxy CORS Anywhere
    const request = await fetch(
      `https://cors-anywhere.herokuapp.com/${soundUrl}`,
      requestOptions
```

```
17      );
18
19      if (request.ok) {
20        const fileBlob = await request.blob();
21        console.log(fileBlob);
22        const array = await getArraySound(fileBlob);
23
24        parent.postMessage(
25          { pluginMessage: { type: "add-to-figma", array: array } },
26          "*"
27        );
28      } else {
29        console.error("Error fetching sound:", request.statusText);
30      }
31  }
```

En este fragmento de código, se utiliza la URL de CORS Anywhere (https://cors-anywhere.herokuapp.com/) como prefijo para todas las solicitudes a la API externa. Además, se incluyen cabeceras como X-Requested-With para cumplir con los requisitos del proxy.

Este enfoque resolvió el problema de manera efectiva, facilitando la recuperación de recursos externos sin modificaciones en las APIs objetivo. Sin embargo, cabe destacar que el uso de proxies como CORS Anywhere conlleva limitaciones significativas que deben ser consideradas. Además de la dependencia de un servicio externo y los potenciales problemas de privacidad si los datos transmitidos son sensibles, existe una restricción adicional que afecta directamente la usabilidad del plugin. Para que CORS Anywhere funcione correctamente en el contexto dado, el usuario debe dirigirse manualmente a la página de demostración del servicio (`https://cors-anywhere.herokuapp.com/corsdemo`) y hacer clic en el botón que solicita acceso temporal al servidor. Este paso, necesario para habilitar el proxy temporalmente, introduce una nueva limitación debido a una fricción considerable en la experiencia del usuario, quien idealmente esperaría que el plugin operara de manera transparente y sin requerir acciones externas.

4.5. Limitación 5: Interacción Manual del Usuario con CORS Anywhere

Originalmente, al desarrollar el plugin, surgió la necesidad de sortear las restricciones de CORS (Cross-Origin Resource Sharing) impuestas por el entorno aislado de Figma y Framer. Como se describió en la Sección 4.4, la solución inicial implicaba el uso de un servicio proxy, específicamente CORS Anywhere, para intermediar las solicitudes HTTP y añadir las cabeceras necesarias.

Sin embargo, esta solución introdujo una limitación significativa en la experiencia del usuario. Para que CORS Anywhere funcionara correctamen-

te, el usuario debía interactuar manualmente con la página de demostración del servicio (`https://cors-anywhere.herokuapp.com/corsdemo`) y hacer clic en el botón Request temporary access to the demo server". Este paso adicional, necesario para habilitar temporalmente el acceso al servidor proxy, representaba una barrera importante en la usabilidad del plugin, ya que requería que el usuario realizara una acción externa y comprendiera la necesidad de este paso.

4.5.1. Solución: Implementación con CORS Proxy

Afortunadamente, durante el desarrollo, se encontró una alternativa prometedora que elimina la necesidad de interacción manual del usuario. Se observó un servicio llamado CORS Proxy (`https://corsproxy.io/?url=https://example.com`), que permite evitar las restricciones de CORS directamente desde el entorno de Framer, sin requerir ninguna acción adicional por parte del usuario.

CORS Proxy funciona como un proxy inverso que añade las cabeceras CORS necesarias a las solicitudes, permitiendo que el plugin acceda a recursos externos sin violar las políticas de seguridad del navegador. Al adoptar esta herramienta, se logró eliminar la dependencia de CORS Anywhere y su requisito de acceso temporal, mejorando significativamente la usabilidad de nuestro plugin.

A continuación, se presenta un fragmento del código utilizado para implementar esta solución:

Código 4.2 Codigo modificado con nueva herramienta de CORS

```
1   async function getSoundToArray(soundUrl: string) {
2     const header = new Headers();
3     const authToken = localStorage.getItem("authToken");
4     header.append("Authorization", "Bearer " + authToken);
5
6     // Se usa el proxy corsproxy.io para evitar problemas de CORS
7     const request = await fetch(
8       `https://corsproxy.io/?url=${soundUrl}`
9     );
10
11    if (request.ok) {
12      // Get the file as a blob
13      const fileBlob = await request.blob();
14      console.log(fileBlob);
15      const array = await getArraySound(fileBlob);
16      return array;
17    } else {
18      console.error("Error fetching sound:", request.statusText);
19    }
20    }
```

En este fragmento de código, se utiliza la URL de CORS Proxy (`https://corsproxy.io/?url=`) como prefijo para todas las solicitudes a la API externa. Esto permite que las solicitudes se realicen sin problemas, sin requerir ninguna intervención manual por parte del usuario.

Al reemplazar CORS Anywhere con CORS Proxy, se ofrece una experiencia de usuario más fluida e intuitiva, eliminando la necesidad de pasos adicionales y simplificando el proceso de integración de vibraciones en los diseños. Esta mejora no solo facilita el uso del plugin, sino que también lo hace más accesible a diseñadores con menos conocimientos técnicos, promoviendo así una mayor adopción y experimentación con interacciones hápticas.

4.6. Limitación 6: Almacenamiento Persistente de Metáforas y Sonidos Personalizados

Figma no ofrece opciones integradas para el almacenamiento persistente, lo que planteó un desafío significativo al momento de garantizar que las metáforas y sonidos personalizados de cada usuario pudieran mantenerse disponibles entre sesiones. Este tipo de persistencia es esencial para brindar una experiencia consistente y personalizada, especialmente en herramientas orientadas al diseño iterativo.

4.6.1. Solución: Uso de *clientStorage*

Durante el desarrollo, se evaluaron varias alternativas para gestionar el almacenamiento de datos, incluyendo herramientas como Dexie, IndexedDB y LocalStorage. Sin embargo, cada una presentó limitaciones en términos de complejidad de integración o compatibilidad con el entorno de Figma.

Finalmente, se optó por *clientStorage*, una funcionalidad que permite almacenar datos en formato clave-valor. Aunque no es tan robusto como otras opciones, *clientStorage* ofreció una integración directa y una curva de aprendizaje reducida, lo que facilitó la implementación. Mediante esta herramienta, se logró garantizar la persistencia de metáforas y sonidos personalizados entre sesiones de uso, brindando a los diseñadores la capacidad de acceder a su contenido sin interrupciones.

4.7. Limitación 7: Hosting de Sonidos Personalizados

Otro desafío importante fue encontrar una solución adecuada para almacenar y gestionar los sonidos personalizados de los usuarios. Este aspecto era crucial para garantizar que los archivos subidos fueran accesibles y estuvieran protegidos, sin añadir una carga excesiva al flujo de trabajo del plugin.

4.7.1. Solución: Implementación con FreeSound

Inicialmente, se evaluaron varias opciones para el hosting de sonidos personalizados, entre las cuales SndUp parecía ser una solución adecuada. Sin embargo, esta herramienta presentó limitaciones que impedían satisfacer completamente los requisitos del proyecto, lo que llevó a explorar alternativas adicionales.

Tras una investigación más extensa, se decidió utilizar FreeSound, una plataforma diseñada para la gestión de archivos de audio. No obstante, esta solución requiere que los usuarios inicien sesión en FreeSound para asociar los sonidos a su cuenta. Este paso adicional garantiza la seguridad y persistencia de los archivos, al tiempo que permite aprovechar las funcionalidades avanzadas de la API de FreeSound para integrar estos sonidos en el flujo de trabajo del plugin.

4.8. Consideraciones Finales

Las limitaciones encontradas durante el desarrollo de esta plataforma subrayan los desafíos inherentes al diseño e implementación de herramientas innovadoras. Sin embargo, las soluciones adoptadas demostraron ser funcionales y permitieron cumplir con los objetivos establecidos. Este proceso también abre el camino para futuras mejoras, particularmente en la automatización de tareas clave y la integración de funcionalidades más avanzadas en las plataformas utilizadas.

Capítulo 5

Desarrollo

En este capítulo se detalla el proceso llevado a cabo para implementar el *plugin-haptico*, una herramienta que facilita la integración de interacciones vibrotáctiles en flujos de diseño UX. Este desarrollo abordó diversos desafíos tecnológicos relacionados con la limitación de plataformas actuales como Figma para incluir comportamientos avanzados como la ejecución de vibraciones, así como otros obstáculos surgidos durante el desarrollo y cómo estas limitaciones se superaron.

El desarrollo del *plugin-haptico* puede dividirse por etapas:

1. Diseño y conceptualización del plugin.
2. Implementación del plugin en Figma.
3. Desarrollo de la interacción con Framer.
4. Migración del plugin a Framer.
5. Pruebas y puesta en escena en dispositivos móviles.

5.1. Diseño y conceptualización del plugin

La conceptualización del *plugin-haptico* inició con la identificación de una necesidad en los flujos de diseño UX: la falta de herramientas que permitan integrar y probar interacciones vibrotáctiles durante el diseño. Actualmente, los diseñadores suelen delegar estas interacciones a los desarrolladores, lo que complica la iteración rápida y consistente.

Para solucionar esto, se definieron los objetivos del plugin:

- Facilitar la selección y personalización de patrones vibrotáctiles: Permitiendo a los diseñadores elegir vibraciones predefinidas o crear patrones personalizados.
- Asignación de patrones a elementos específicos del diseño: A través de una interfaz que simplifique esta asociación.
- Exportación y prueba de las vibraciones en prototipos interactivos: Habilitando su simulación en dispositivos móviles mediante integración con Figma y Framer.

5.1.1. Conceptualización de la Interfaz

El diseño de la interfaz se enfocó en la usabilidad y la integración natural con Figma. Se establecieron las siguientes características para garantizar una experiencia fluida:

- Panel Desplegable en Figma: Un panel que permitiera a los diseñadores seleccionar patrones vibrotáctiles directamente desde la interfaz del plugin, con opciones de previsualización.
- Modificación de Vibraciones: Edición de las categorías y tags asociadas a cada vibración

5.1.2. Diseño Técnico

En paralelo, se desarrolló la arquitectura técnica del plugin. Esto incluyó la definición de cómo interactuaría el plugin con la API de Figma y cómo se manejarían los datos asociados a los patrones vibrotáctiles, tanto su almacenamiento en algún servicio externo como su reproducción y edición.

5.1.3. Metodología de Trabajo

Para garantizar un desarrollo ordenado y eficiente, se definió una metodología de trabajo basada en control de versiones. Se creó un repositorio en GitHub, estructurado con las siguientes ramas:

- `main`: Rama principal utilizada para versiones estables del plugin.
- `develop`: Rama de desarrollo en la cual se integraron las funcionalidades antes de su consolidación.
- `feature`: Ramas derivadas de `develop` para implementar funcionalidades específicas.
- `fix`: Ramas dedicadas a solucionar errores detectados durante el desarrollo o pruebas.

El desarrollo avanzó de manera incremental, comenzando por las funcionalidades básicas y avanzando hacia las más complejas. Esta estructura permitió la colaboración en paralelo, asegurando que el código nuevo no afectara las partes estables del proyecto.

5.2. Implementación del plugin en Figma

El proceso de implementación del *plugin-haptico* comenzó con la creación de un plugin base utilizando la API de Figma. Este punto inicial fue

esencial para garantizar que el plugin pudiera integrarse sin problemas en la plataforma y sirviera como base para el desarrollo de funcionalidades más avanzadas, como la gestión de vibraciones desde el catálogo de VibViz.

5.2.1. Creación del Plugin Base

Para crear el plugin base en Figma, se siguieron los pasos recomendados en la documentación oficial de la API [29]:

1. Acceso al entorno de desarrollo: Desde Figma, se accedió al menú contextual seleccionando **Plugins** → **Development** → **New Plugin...**.

2. En el modal **Create plugin**, se seleccionó Diseño Figma y se le dió el nombre **haptic-plugin**, como se muestra en la Figura 5.1.

3. Definición de la estructura del proyecto: Se utilizó la plantilla básica proporcionada por Figma llamada **Custom UI**, generando automáticamente un archivo `manifest.json` para configurar el plugin. Este archivo define las propiedades esenciales como el nombre, la descripción, y los permisos necesarios.

4. Ejecución inicial del plugin: Con la estructura básica creada, se realizó una primera ejecución para verificar su correcta integración en el entorno de Figma. Esto consistió en mostrar una ventana emergente con el contenido por defecto del plugin creado y sus iteracciones, asegurando que los elementos clave del plugin respondían de manera adecuada a la interacción del usuario (ver Figura 5.2).

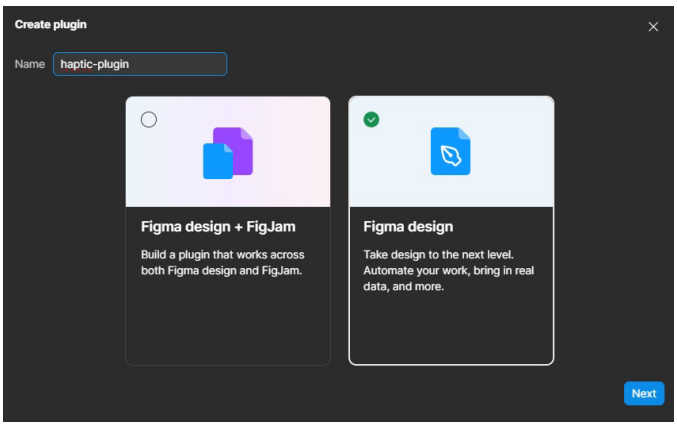

Figura 5.1: Opciones para crear un nuevo plugin en Figma.

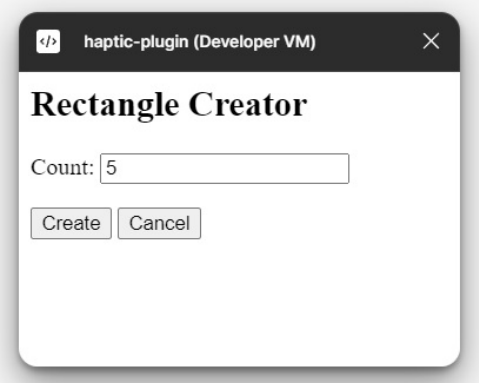

Figura 5.2: Primera ejecución del plugin en figma.

5.2.2. Integración del Catálogo de VibViz

Una vez obtenido el conocimiento y una base para comenzar a crear el plugin, el siguiente paso en la implementación fue integrar el catálogo de vibraciones de VibViz. Este catálogo, diseñado para facilitar la exploración y selección de patrones vibrotáctiles predefinidos, se consideró una pieza fundamental del plugin, ya que proporcionaría a los diseñadores una base amplia de vibraciones para utilizar en sus flujos de diseño. Este catálogo no solo aportó una base sólida para el diseño, sino que también se plantearon varios desafíos relacionados con la extracción y el enriquecimiento de datos, los cuales se resolvieron mediante herramientas automatizadas y un análisis detallado.

Uno de los archivos que proporciona la documentación abierta de VibViz, denominado `vibrationAnnotations-July24th2016.csv`, contenía una gran cantidad de información útil para el desarrollo del plugin, y otra redundante o innecesaria para los objetivos deseados (ver Figura 5.3).

id	index	energy	tempo	roughness	pleasantness	arousal	sensationTags	emotionTags
v-09-09-8-11	11	-2	0.769230769	-2.615384615	0.692307692	-2.461538462	simple,discontinuous,soft,g	comfortable,calm,natural,familiar,p
v-09-09-8-20	12	-1	0.538461538	-0.692307692	1.153846154	-0.384615385	smooth,simple,regular,bum	lively,pleasant,calm,natural,familia
v-09-09-8-24	13	0.285714	-0.571428571	0.071428571	-2	1	ramping up,dynamic,grainy,smooth transition,complex,long	annoying,predictable,agitating,me
v-09-10-11-55	14	2.230769	-0.538461538	1.230769231	-1.076923077	1.846153846	dynamic,discontinuous,rou up,long,complex	uncomfortable,surprising,annoying
v-09-10-11-58	10	0.5	-2	0.5	0.5	0	continuous,regular,long,sof up,changing	natural,boring,predictable,annoyin
v-09-10-12-11	15	2.076923	2.307692308	2.076923077	-1.692307692	2.230769231	rough,regular,discontinuou	annoying,angry,uncomfortable,urg

Figura 5.3: Pequeña muestra del listado de VibViz.

Tras un análisis inicial, se identificó que los datos clave que debían ser extraídos eran los identificadores únicos (**id**) y las metáforas asociadas a cada patrón (**metaphors**). Para procesar esta información, se desarrolló un script en Python que se encargó de filtrar y depurar el archivo original,

eliminando columnas superfluas y generando un conjunto de datos reducido que contenía únicamente estos dos campos esenciales. Esta etapa marcó el inicio de la estructuración del catálogo necesario para el plugin (Figura 5.4).

Código 5.1 Codigo Python de filtrado

```
import csv

input_file = "vibrationAnnotations-July24th2016.csv"
output_file = "filtered_vibrationAnnotations.csv"

with open(input_file, mode="r", encoding="utf-8") as infile:
    reader = csv.DictReader(infile)

    with open(output_file, mode="w", encoding="utf-8", newline="") as outfile:
        writer = csv.DictWriter(outfile, fieldnames=["id", "metaphors"])
        writer.writeheader()

        for row in reader:
            writer.writerow({"id": row["id"], "metaphors": row["metaphors"]})
```

Sin embargo, todavía faltaba información crucial para completar el esquema de datos requerido. Por un lado, era necesario obtener los enlaces a los archivos de audio correspondientes a los patrones de vibración; por otro, era indispensable contar con una representación visual asociada a cada vibración. Estas dos piezas faltantes no estaban incluidas en el archivo inicial de VibViz, lo que llevó a profundizar en el análisis del flujo de datos de su plataforma.

A través de herramientas de desarrollo web, como las DevTools del navegador, se rastrearon las solicitudes realizadas por la página de VibViz al seleccionar un patrón específico. Este análisis reveló que los archivos de audio estaban alojados en la URL base **https://www.cs.ubc.ca/~ seifi/VibViz/vteffects/[sound_id].wav**, mientras que las imágenes representativas podían encontrarse en **https://www.cs.ubc.ca/~ seifi/VibViz/PNGglyph/[sound_id].png**, donde `[sound_id]` corresponde al identificador único de cada vibración. Este descubrimiento permitió reproducir de manera automatizada este flujo para completar los datos faltantes.

filtered_vibrationAnnotations

id	metaphors
v-09-09-8-11	hearbeat,pulsing,tapping,poking
v-09-09-8-20	hearbeat,tapping,animal
v-09-09-8-24	engine,coming or going
v-09-10-11-55	alarm,jumping,sliding,phone
v-09-10-11-58	purring,snoring,animal,coming or going
v-09-10-12-11	alarm,electric shock
v-09-10-12-13	alarm,tapping,musical instruments,drums,beep,phone,celebration
v-09-10-12-16	musical instruments,beep
v-09-10-12-2	snoring,buzz
v-09-10-12-6	snoring,bell,walking,alarm,coming or going
v-09-10-12-9	tapping,alarm,musical instruments,drums,animal
v-09-10-3-52	tapping,morse code,phone,alarm
v-09-10-3-56	gun
v-09-10-4-2	tapping,pulsing
v-09-10-4-20	hearbeat,pawing,animal,tapping
v-09-10-4-23	hearbeat,tapping,pawing,clock
v-09-10-4-25	engine,hearbeat,tapping
v-09-10-4-6	tapping,coming or going,animal
v-09-10-6-16	tapping,beep,hearbeat
v-09-10-6-22	gun,electric shock,engine,riding
v-09-10-6-27	tapping,beep,alarm,musical instruments,drums
v-09-10-6-38	getting close,engine
v-09-10-6-43	pulsing,game,tapping,snoring,sliding

Figura 5.4: Pequeña muestra del listado de VibViz filtrado.

Mediante un segundo script en Python, se logra recorrer estas rutas y construir un dataset enriquecido que contiene los cuatro elementos fundamentales para cada patrón: el identificador único, las metáforas descriptivas, la URL del archivo de sonido y la imagen representativa (ver Figura 5.5).

Código 5.2 Codigo Python para Json

```
import csv
import json

input_file = "filtered_vibrationAnnotations.csv"
output_file = "sounds.json"

```

```python
base_url = "https://www.cs.ubc.ca/~seifi/VibViz/vteffects/"
base_image_url = "https://www.cs.ubc.ca/~seifi/VibViz/PNGglyph/"

sounds = []

with open(input_file, mode="r", encoding="utf-8") as infile:
    reader = csv.DictReader(infile)

    for row in reader:
        sound = {
            "name": row["id"],
            "url": f"{base_url}{row['id']}.wav",
            "image": f"{base_image_url}{row['id']}.png",
            "metaphors": row["metaphors"]
        }
        sounds.append(sound)

with open(output_file, mode="w", encoding="utf-8") as outfile:
    json.dump(sounds, outfile, indent=2, ensure_ascii=False)
```


```json
[
  {
    "name": "v-09-09-8-11",
    "url": "https://www.cs.ubc.ca/~seifi/VibViz/vteffects/v-09-09-8-11.wav",
    "image": "https://www.cs.ubc.ca/~seifi/VibViz/PNGglyph/v-09-09-8-11.png",
    "metaphors": "hearbeat,pulsing,tapping,poking"
  },
  {
    "name": "v-09-09-8-20",
    "url": "https://www.cs.ubc.ca/~seifi/VibViz/vteffects/v-09-09-8-20.wav",
    "image": "https://www.cs.ubc.ca/~seifi/VibViz/PNGglyph/v-09-09-8-20.png",
    "metaphors": "hearbeat,tapping,animal"
  },
  {
    "name": "v-09-09-8-24",
    "url": "https://www.cs.ubc.ca/~seifi/VibViz/vteffects/v-09-09-8-24.wav",
    "image": "https://www.cs.ubc.ca/~seifi/VibViz/PNGglyph/v-09-09-8-24.png",
    "metaphors": "engine,coming or going"
  },
  {
    "name": "v-09-10-11-55",
    "url": "https://www.cs.ubc.ca/~seifi/VibViz/vteffects/v-09-10-11-55.wav",
    "image": "https://www.cs.ubc.ca/~seifi/VibViz/PNGglyph/v-09-10-11-55.png",
    "metaphors": "alarm,jumping,sliding,phone"
  },
  {
    "name": "v-09-10-11-58",
    "url": "https://www.cs.ubc.ca/~seifi/VibViz/vteffects/v-09-10-11-58.wav",
    "image": "https://www.cs.ubc.ca/~seifi/VibViz/PNGglyph/v-09-10-11-58.png",
    "metaphors": "purring,snoring,animal,coming or going"
  },
```


Figura 5.5: Pequeña muestra del JSON de sonidos previo a la traducción

Para garantizar la calidad del catálogo, se realizó una validación manual que confirmó la integridad y la coherencia de los datos obtenidos. Finalmente, se tradujeron las metáforas al español y luego esta información se integró

en el *plugin-haptico*, permitiendo un acceso fácil a un catálogo organizado y completo de vibraciones.

5.2.3. Almacenamiento y Recuperación de Tags/Metáforas personalizadas

El almacenamiento y recuperación de tags o metáforas asociadas a las vibraciones fueron diseñados para garantizar eficiencia, persistencia de datos y simplicidad en la estructura del plugin. Debido a la necesidad de evitar redundancias,se eligió una estructura centralizada basada en un único repositorio de información persistente.

En un primer momento se tuvieron en cuenta diversas opciones como Dexie o IndexedDB, pero a causa de las restricciones con las que cuenta Figma, la implementación de estas opciones se tornaba demasiado compleja. Por ende, para almacenar tags personalizadas se utilizó un modelo de datos orientado a claves, con cada vibración identificada por el *id* único del sonido. Las metáforas personalizadas se asociaron directamente a este identificador en una base de datos local, *localStorage*, que permite realizar operaciones de almacenamiento y recuperación de datos de manera eficiente.

La estructura del almacenamiento se definió de la siguiente forma:

Código 5.3 Ejemplo de almacenamiento key-value de tags personalizados

```
1 key: 'v-09-09-8-24',
2 value: 'latido del corazon,animal,palpito'
```

El almacenamiento de las tags se implementó mediante funciones especializadas que permiten realizar las operaciones de agregado, modificación y recuperación de datos. Estas funciones interactúan con *localStorage* para garantizar un manejo eficiente y estructurado de la información. Una de las funciones está diseñada para agregar nuevas metáforas asociadas a una vibración específica. En caso de que la clave correspondiente no exista en *localStorage*, se crea un nuevo registro; si la clave ya está presente, el valor asociado se actualiza adecuadamente. Adicionalmente, otra función se encarga de recuperar las metáforas almacenadas, permitiendo obtener las asociadas a cada vibración según el usuario.

Código 5.4 Agregar tags

```
1 async function updateSound(sound: { name: string; metaphors: string }) {
2   await figma.clientStorage.setAsync(sound.name, sound.metaphors);
3 }
```

Código 5.5 Recuperar tags personalizados de los sonidos

```
1 const data = await figma.clientStorage.getAsync(msg.key);
```

```
2     const url = await figma.clientStorage.getAsync(msg.key + "-url");
3     const image = await figma.clientStorage.getAsync(msg.key + "-image
      ");
4     const external = url && image;
5     figma.ui.postMessage({
6       type: "sound",
7       key: msg.key,
8       value: data,
9       url,
10      image,
11      external,
12    });
```

En la interfaz del plugin, las tags personalizadas se presentan de manera intuitiva junto con las vibraciones predefinidas. La comunicación entre la interfaz de usuario (UI) y la lógica del plugin se gestiona a través de mensajes intercambiados entre los archivos `code.ts` y `ui.html`, lo que garantiza una actualización en tiempo real de las tags visualizadas.

Para garantizar que el almacenamiento y la recuperación de tags funcionan correctamente, se realizaron pruebas centradas en la persistencia de los datos, asegurando que las tags agregadas permanecen en la base de datos incluso después de cerrar y reabrir el plugin. También se verificó la consistencia de las operaciones, confirmando que las acciones de agregar, eliminar y recuperar tags se reflejan correctamente en la interfaz del usuario. Además, se validó el manejo adecuado de errores, comprobando que las funciones responden correctamente ante situaciones como IDs de vibraciones inexistentes o intentos de agregar tags duplicadas. Con estas pruebas, se asegura una experiencia de usuario consistente y fluida para los diseñadores que utilizan el plugin para gestionar interacciones vibrotáctiles.

5.2.4. Integración de un Filtro para Sonidos y Metáforas

Contando con las metáforas personalizadas, y con el objetivo de optimizar la experiencia de los diseñadores además de proporcionar una herramienta útil para la selección de sonidos en proyectos vibrotáctiles, se implementó un filtro avanzado. Este filtro permite a los usuarios buscar sonidos mediante su identificador único o filtrar resultados por metáforas/tags asociadas. La funcionalidad del filtro está diseñada para ofrecer flexibilidad y rapidez, permitiendo a los diseñadores centrarse en categorías específicas o identificar sonidos individuales de forma eficiente.

El filtro consiste en un campo de entrada (input) donde el usuario puede introducir una cadena de texto. A partir de este campo se analizan los datos almacenados en la base de datos, comparando tanto los identificadores de los sonidos como las metáforas asociadas. Al escribir en el campo de búsqueda, el sistema realiza una consulta dinámica, actualizando los resultados en tiempo real y mostrando solo los elementos que coinciden con los criterios especificados.

La lógica subyacente del filtro está optimizada para admitir coincidencias parciales y completas, facilitando la identificación de sonidos incluso cuando el usuario no recuerda el identificador completo o solo tiene en mente una palabra clave relacionada con una metáfora.

Código 5.6 Implementación del filtro

```
function filterSounds() {
      const filter = document
        .getElementById("metaphorFilter")
        .value.toLowerCase();
      const filteredSounds = sounds.filter((sound) =>
        sound.metaphors
        .split(",")
        .some((metaphor) => metaphor.trim().toLowerCase().includes(
    filter))
      );
      displayFilteredSounds(filteredSounds);
    }
```

Esta función garantiza que el filtro sea flexible y eficiente, adaptándose a las necesidades del usuario final.

La integración de este filtro aporta varios beneficios clave a la experiencia del diseñador. En primer lugar, al permitir filtrar por metáforas, el diseñador puede agrupar sonidos relacionados, lo que facilita la comparación y selección del más adecuado para una funcionalidad específica. Además, la posibilidad de buscar por identificador acelera el proceso de localización de sonidos específicos dentro de una colección extensa, optimizando así el flujo de trabajo. Finalmente, la actualización dinámica de los resultados mejora la interacción del usuario al proporcionar una experiencia fluida y sin interrupciones, lo que refuerza la interactividad y la eficacia del plugin.

5.2.5. Implementación de la subida de sonidos personalizados

Como primer paso luego de la incorporación del catalogo al *plugin-haptico*, se incorporó la posibilidad de que los usuarios pudieran añadir sonidos personalizados al catálogo. Esto se logró mediante la integración de la API de FreeSound, una plataforma colaborativa que permite la subida, organización y reutilización de archivos de audio bajo licencias Creative Commons [24]. A través de esta integración, los usuarios pueden cargar sus propios sonidos, gestionarlos mediante etiquetas descriptivas y utilizar estos recursos para enriquecer sus diseños con interacciones vibrotáctiles personalizadas.

El proceso de carga comienza cuando el usuario hace clic en el botón de agregar sonido dentro del plugin. Este botón activa un modal donde el usuario puede ingresar los detalles del sonido que desea subir, como el nombre, la descripción, las etiquetas y, por supuesto, el archivo de audio.

La lógica de la interfaz se encarga de validar los campos y luego enviar los datos al backend, donde se interactúa con la API de FreeSound para realizar la subida del archivo.

El botón de agregar sonido, representado por un ícono de un círculo con un símbolo de más, activa el modal correspondiente en la interfaz. Su implementación en HTML es la siguiente:

Código 5.7 Código del botón HTML

```
<button id="add-sound" title="Add sound">
  <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill
    ="currentColor" class="bi bi-plus-circle-fill" viewBox="0 0 16
    16">
    <path d="M16 8A8 8 0 1 1 0 8a8 8 0 0 1 16 0M8.5 4.5a.5.5 0 0 0-1 0
    v3h-3a.5.5 0 0 0 0 1h3v3a.5.5 0 0 0 1 0v-3h3a.5.5 0 0 0 0-1h-3z"/>
  </svg>
</button>
```

Al hacer clic en este botón, se ejecuta un evento en JavaScript que verifica si el usuario está autenticado. Si el usuario no ha iniciado sesión, se muestra un modal de registro; de lo contrario, se muestra el modal para la carga del sonido:

Código 5.8 Codigo que verifica si el usuario esta o no registrado

```
document.getElementById("add-sound").onclick = () => {
  parent.postMessage({
    pluginMessage: {
      type: "check-token"
    }
  }, "*");

  if (!registered) {
    document.getElementById("registrationModal").style.display = "
    block";
  } else {
    document.getElementById("uploadModal").style.display = "block";
    document.getElementById("registrationModal").style.display = "none
    ";
  }
};
```

El plugin contiene dos modales esenciales: uno para la autenticación del usuario y otro para la carga del sonido. Si el usuario ya está autenticado, el modal de subida de archivo le permite ingresar el nombre, la descripción, las etiquetas y seleccionar el archivo de audio. Para mejorar la usabilidad, se implementó una función que permite cerrar estos modales fácilmente al hacer clic en un botón de cierre:

Código 5.9 Codigo que cierra los modales.

```
// Cerrar el modal de subida
document.querySelector(".close-upload").onclick = () => {
  document.getElementById("uploadModal").style.display = "none";
```

```
};

// Cerrar el modal de registro
document.querySelector(".close-register").onclick = () => {
  document.getElementById("registrationModal").style.display = "none";
};
```

La función principal que maneja la lógica de subida del archivo se activa cuando el usuario hace click en el botón de "subir sonido". Esta función valida que todos los campos estén completos (nombre, descripción, etiquetas y archivo de audio) y ciertos requisitos necesarios como por ejemplo que se ingresen como mínimo 3 etiquetas. Posteriormente, envía la información a la API de FreeSound.

Código 5.10 Funcion principal del boton

```
document.getElementById("uploadSound").onclick = async () => {
  document.getElementById("uploadSound").disabled = true; // Bloquear
    el boton

  const name = document.getElementById("name").value.trim();
  const description = document.getElementById("description").value.
    trim();
  const tags = document.getElementById("tags").value.trim();
  const fileInput = document.getElementById("soundFile");

  try {
    if (description === "" || tags === "" || fileInput.files.length
    === 0) {
      alert("Todos los campos son obligatorios");
      return;
    }

    if (tags.split(" ").length < 3) {
      alert("Por favor, ingrese al menos 3 tags.");
      return;
    }

    if (sounds.some((sound) => sound.name === name)) {
      alert("El nombre ya existe en la lista de sonidos. Por favor,
    elija otro.");
      return;
    }

    if (fileInput.files.length > 0) {
      const file = fileInput.files[0];
      const reader = new FileReader();

      const succeded = await uploadSound(file, tags, description, name
    );

      if (succeded) {
        document.getElementById("name").value = "";
        document.getElementById("description").value = "";
        document.getElementById("tags").value = "";
        document.getElementById("soundFile").value = "";
        alert("Sonido subido exitosamente! Aguarde unos minutos para
    que aparezca en la lista de sonidos.");

```

```
38          setTimeout(() => {
39            displaySounds();
40          }, 3000);
41        } else {
42          alert("Error al cargar el sonido. Por favor, intentelo de
      nuevo.");
43        }
44
45        document.getElementById("uploadModal").style.display = "none";
46      } else {
47        alert("Seleccione un archivo para cargar.");
48      }
49    } catch (error) {
50      console.error("Error uploading sound:", error);
51    } finally {
52      document.getElementById("uploadSound").disabled = false;
53    }
54  };
```

Para la subida del archivo a FreeSound, se utiliza la función uploadSound, que maneja el envío de la información a la API. Esta API requiere un formulario de tipo POST que incluye tanto el archivo de audio como los metadatos (etiquetas, descripción, nombre y licencia). La respuesta de la API devuelve un ID único para el sonido, que luego se utiliza para generar la URL de reproducción y la imagen asociada.

Código 5.11 Función uploadSound

```
1  async function uploadSound(file, tags, description, name) {
2    const data = new FormData();
3    data.append("name", name);
4    data.append("tags", tags);
5    data.append("description", description);
6    data.append("license", "Creative Commons 0");
7    data.append("audiofile", file);
8
9    const header = new Headers();
10   header.append("Authorization", "Bearer " + t0k3n);
11
12   const requestOptions = {
13     method: "POST",
14     headers: header,
15     body: data,
16   };
17
18   const request = await fetch("https://freesound.org/apiv2/sounds/
       upload/", requestOptions);
19
20   if (!uniqueId) {
21     await getUniqueId();
22   }
23
24   const response = await request.json();
25   if (response.id) {
26     const key = response.id.toString();
27     sound_data = {
28       key: key,
29       metaphors: tags,
30       url: `https://cdn.freesound.org/previews/${key.slice(0, 3)}/${
```

```
      key}_${uniqueId}-lq.mp3`,
31      image: `https://cdn.freesound.org/displays/${key.slice(0, 3)}/${
      key}_${uniqueId}_wave_bw_M.png`,
32    };
33
34    sounds.push({
35      name: key,
36      url: sound_data.url,
37      image: sound_data.image,
38      metaphors: tags,
39    });
40
41    parent.postMessage({
42      pluginMessage: {
43        type: "sound-uploaded",
44        sound: sound_data,
45      }
46    }, "*");
47
48    return true;
49  } else {
50    console.log("Error en la subida del archivo");
51    return false;
52  }
53 }
```

Con esta implementación, el usuario puede subir sonidos personalizados y gestionar su biblioteca de manera sencilla dentro del plugin, ampliando las posibilidades de personalización y exploración de interacciones vibrotáctiles.

Código 5.12 función playSound para reproducir un sonido

```
1 } else if (msg.type === "sound-uploaded") {
2   await figma.clientStorage.setAsync(msg.sound.key, msg.sound.
      metaphors);
3   await figma.clientStorage.setAsync(msg.sound.key + "-url", msg.sound
      .url);
4   await figma.clientStorage.setAsync(msg.sound.key + "-image", msg.
      sound.image);
5 }
```

5.2.6. Implementación de la Reproducción de Sonido

Con el catálogo de VibViz integrado al plugin de manera funcional, y la funcionalidad de añadir un sonido creada, el siguiente paso consistió en implementar la funcionalidad de reproducción de sonido al presionar un botón correspondiente a un sonido del catálogo. Esta característica permite a los usuarios escuchar el sonido seleccionado antes de asignarlo a una interacción vibrotáctil en forma de vibración, facilitando la toma de decisiones en el diseño. Para lograr esto, se desarrolló un mecanismo que muestra los sonidos disponibles en la interfaz del plugin y permite su reproducción de manera sencilla e intuitiva. La función `displaySounds()` es la encargada de renderizar la lista de sonidos, presentándolos con su imagen representativa, nombre y metáforas asociadas, junto con un botón de reproducción.

Código 5.13 parte de la función displaySounds

```
function displaySounds(soundsToDisplay = sounds) {
const soundList = document.getElementById("sound-list");
soundList.innerHTML = "";
soundsToDisplay.forEach((sound) => {
soundList.innerHTML +=
            <div class="sound-item" id="sound-${sound.name}">
                <div class="sound-image">
                    <img src="${sound.image}" alt="${sound.name}"
    width="50" height="50">
                </div>
                <div class="sound-info" style="width: 40%">
                    <span>Nombre: ${sound.name}</span>
                    <span>Met\'aforas: ${sound.metaphors}</span>
                </div>
                <button onclick="playSound('${sound.url}')" title="
    Reproducir sonido">
                    <svg xmlns="http://www.w3.org/2000/svg" width="16"
     height="16" fill="currentColor" class="bi bi-play-circle-fill"
    viewBox="0 0 16 16">
                        <path d="M16 8A8 8 0 1 1 0 8a8 8 0 0 1 16 0M6
    .79 5.093A.5.5 0 0 0 6 5.5v5a.5.5 0 0 0 .79.407l3.5-2.5a.5.5 0 0
     0-.814z"/>
                    </svg>
                </button>
            </div>;
            // mas
});
}
```

Para la reproducción del sonido, se utilizó la API `Audio` de JavaScript, que permite cargar y ejecutar archivos de audio directamente desde una URL. La función `playSound()` toma la URL del sonido como argumento y lo reproduce al instante.

Código 5.14 función playSound para reproducir un sonido

```
function playSound(url) {
const audio = new Audio(url);
audio.play().catch(error => console.error("Error al reproducir el
    sonido:", error));
}
```

Dado que los sonidos pueden provenir de diferentes fuentes, como FreeSound o el propio catálogo de VibViz, esta solución garantiza la flexibilidad necesaria para manejar distintos orígenes sin afectar la funcionalidad del plugin.

5.2.7. Implementación de la conversión de sonido a vibración

Una vez implementada la funcionalidad de reproducción de sonidos en el plugin, el siguiente paso consistió en transformar estos sonidos en vibraciones para ser utilizadas en las interacciones vibrotáctiles dentro del diseño UX. Esta conversión es crucial, ya que permite a los diseñadores explorar

las correspondencias entre los patrones sonoros y la retroalimentación háptica. Uno de los principales retos al implementar esta función fue la ausencia de un método estandarizado que permitiera convertir un archivo de sonido en una secuencia de vibraciones. Además de esto, no se halló una manera de conectar la API de Figma con los métodos de la API de Android para ejecutar sonidos directamente en dispositivos móviles, debido a las numerosas limitaciones ya mencionadas por trabajar en el entorno de Figma. A raíz de lo mencionado, se optó por usar window.vibrate() como la solución más viable, ya que permite generar patrones de vibración sin depender de integraciones adicionales con sistemas operativos específicos.

Para realizar la conversión de un sonido (archivo .wav) en un arreglo compatible con window.vibrate(), se implementó la función getSoundToArray(soundUrl), que se encarga de obtener el archivo de audio desde su URL, procesarlo y enviarlo a Figma en el formato de un arreglo de vibración.

Código 5.15 Funcion principal para convertir el sonido seleccionado en un arreglo

```
async function getSoundToArray(soundUrl) {
  const header = new Headers();
  header.append("Authorization", "Bearer " + t0k3n);
  header.append("X-Requested-With", "XMLHttpRequest");
  const requestOptions = {
    method: "GET",
    headers: header,
  };

  const request = await fetch(
    `https://cors-anywhere.herokuapp.com/${soundUrl}`,
    requestOptions
  );
  if (request.ok) {
    const fileBlob = await request.blob();
    const array = await getArraySound(fileBlob);
    parent.postMessage(
      { pluginMessage: { type: "add-to-figma", array: array } },
      "*"
    );
  } else {
    console.error("Error fetching sound:", request.statusText);
  }
}
```

Dado que el entorno de Figma impone restricciones de seguridad en las solicitudes HTTP, fue necesario utilizar un servicio intermediario para realizar las peticiones. Se empleó CORS Anywhere [30], un servicio basado en Heroku, que permite superar restricciones de CORS al actuar como un proxy entre la aplicación y la fuente de datos. Sin embargo, este enfoque presenta limitaciones, como disponibilidad y posibles restricciones en la cantidad de peticiones permitidas.

Por su parte, la función getArraySound(file, threshold, sampleStep) realiza el análisis del archivo de audio y extrae un patrón de vibración con base

en la amplitud de la señal. Los principales parámetros de esta función son:

1. file: archivo de audio en formato Blob.
2. threshold: umbral de amplitud para determinar si un segmento corresponde a vibración o pausa.
3. sampleStep: intervalo de muestreo en el que se toman valores del audio.

Código 5.16 Funcion que convierte en sonido en arreglo

```
async function getArraySound(file, threshold = 0.5, sampleStep = 100) {
  const context = new(window.AudioContext || window.webkitAudioContext)();
  const arrayBuffer = await file.arrayBuffer();
  const audioBuffer = await context.decodeAudioData(arrayBuffer);
  const data = audioBuffer.getChannelData(0);
  const sampleRate = audioBuffer.sampleRate;

  const pattern = [];
  let vibrate = false;
  let currentDuration = 0;
  const msPerStep = (1000 / sampleRate) * sampleStep;

  for (let i = 0; i < data.length; i += sampleStep) {
    const amplitude = Math.abs(data[i]);
    if (amplitude > threshold) {
      if (vibrate) {
        currentDuration += msPerStep;
      } else {
        if (currentDuration > 0) pattern.push(Math.round(currentDuration));
        vibrate = true;
        currentDuration = msPerStep;
      }
    } else {
      if (vibrate) {
        if (currentDuration > 0) pattern.push(Math.round(currentDuration));
        vibrate = false;
        currentDuration = msPerStep;
      } else {
        currentDuration += msPerStep;
      }
    }
  }

  if (currentDuration > 0) pattern.push(Math.round(currentDuration));
  console.log(pattern);
  return pattern;
}
```

Este algoritmo procesa la señal de audio y traduce las variaciones de amplitud en una secuencia de vibración. Se recorre el audio a intervalos regulares (sampleStep), y si la amplitud supera el umbral (threshold), se considera una fase de vibración; de lo contrario, se interpreta como una

pausa. Esto es así dado que window.vibrate() no permite que la intensidad de la vibración varíe. La duración de cada fase se acumula y se almacena en el arreglo final que luego es compatible con window.vibrate().

El siguiente botón dentro de la interfaz del plugin es el que activa la función de conversión:

Código 5.17 Código del botón de conversión

```
<button onclick="getSoundToArray('${sound.url}')" title="Agregar al
    prototipo">
  <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16"
  fill="currentColor" class="bi bi-plus" viewBox="0 0 16 16">
    <path d="M8 1a.5.5 0 0 1 .5.5v6h6a.5.5 0 0 1 0 1h-6v6a.5.5 0 0
  1-1 0v-6h-6a.5.5 0 0 1 0-1h6v-6A.5.5 0 0 1 8 1z"/>
  </svg>
</button>
```

Como también puede apreciarse, se realiza el envío del mensaje *add-to-figma*, mediante parent.postMessage, el cual es recibido en el archivo code.ts, donde se encarga de procesar el arreglo y agregarlo al lienzo de Figma en un Frame junto con un mensaje explicativo.

Código 5.18 Código del botón de conversión

```
if (msg.type === "add-to-figma") {
    const array = msg.array;

    await figma.loadFontAsync({ family: "Inter", style: "Regular" });

    const frame = figma.createFrame();
    frame.resize(400, 300);
    frame.fills = [{ type: "SOLID", color: { r: 1, g: 1, b: 1 } }];
    frame.name = "Haptic Interaction Frame";

    const text = figma.createText();
    text.characters = "Cree una interaccion 'OnClick' con la accion '
    Open Link' en el boton para abrir nuestra web externa: https://
    ourapi.com";
    text.fontSize = 16;
    text.resize(380, 50);
    text.x = 10;
    text.y = 10;
    text.fills = [{ type: "SOLID", color: { r: 0, g: 0, b: 0 } }];
    frame.appendChild(text);

    const buttonFrame = figma.createFrame();
    buttonFrame.resize(100, 40);
    buttonFrame.cornerRadius = 10;
    buttonFrame.fills = [{ type: "SOLID", color: { r: 0, g: 0, b: 1 }
    }];
    buttonFrame.x = (frame.width - buttonFrame.width) / 2;
    buttonFrame.y = 70;

    const buttonText = figma.createText();
    buttonText.characters = "Vibrar!";
    buttonText.fontSize = 16;
    buttonText.resize(80, 30);
    buttonText.x = 10;
```

```
32      buttonText.y = 5;
33      buttonText.fills = [{ type: "SOLID", color: { r: 1, g: 1, b: 1 }
        }];
34      buttonFrame.appendChild(buttonText);
35
36      frame.appendChild(buttonFrame);
37      figma.currentPage.appendChild(frame);
38  }
```

5.2.8. Test de la vibración seleccionada

Una vez que el usuario ha seleccionado un sonido de la biblioteca disponible y luego de que se haya generado el arreglo correspondiente para la ejecución de la vibración, el siguiente paso en el flujo de trabajo consiste en integrar dicho patrón al diseño dentro de Figma. Para facilitar este proceso, se implementó un botón para agregar el sonido seleccionado al diseño. Al hacerlo, se despliega automáticamente un cartel informativo que confirma la correcta generación del arreglo de vibración y su correcta asociación con el diseño en curso, proporcionando así una validación visual del proceso (ver Figura 5.6).

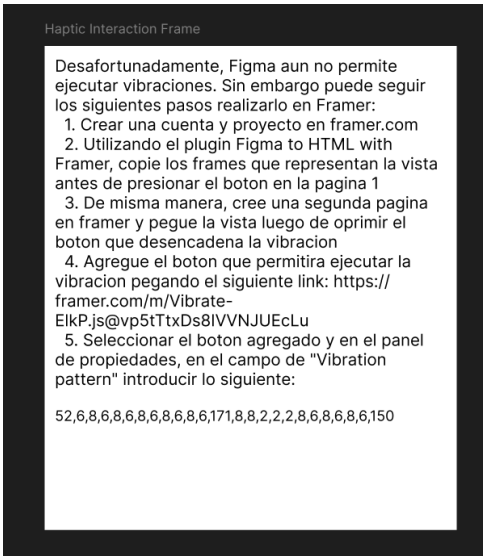

Figura 5.6: Cartel informativo con los pasos a seguir

Sin embargo, esta etapa del desarrollo reveló una de las principales limitaciones técnicas de la plataforma, especialmente en lo que respecta a la ejecución de vibraciones en dispositivos móviles. Figma, debido a su naturaleza altamente restrictiva en términos de interacción con hardware, no permite la activación de vibraciones dentro de su aplicación móvil ni desde navegadores en dispositivos celulares. Esta restricción impacta negativamen-

te en la experiencia del diseñador, ya que imposibilita la prueba en tiempo real de la retroalimentación háptica dentro de los prototipos, afectando directamente la fluidez del flujo de trabajo iterativo y a la efectividad del proceso de diseño.

Ante esta situación, se realizó un análisis detallado de posibles soluciones alternativas que pudieran brindar soporte para la ejecución de vibraciones en dispositivos móviles sin depender exclusivamente de Figma. Como resultado de esta investigación, se identificó a Framer como una plataforma viable, ya que permite la integración de código JavaScript personalizado dentro de los prototipos, ofreciendo una mayor flexibilidad en comparación con Figma. A partir de esta posibilidad, se desarrolló un procedimiento estructurado mediante el cual los usuarios pueden migrar sus diseños desde Figma a Framer y, posteriormente, habilitar la ejecución de vibraciones en entornos móviles.

Framer permite a los diseñadores integrar código personalizado en sus prototipos, lo que la convierte en una herramienta significativamente más flexible en comparación con Figma. Su capacidad para ejecutar scripts y manejar interacciones avanzadas posibilita una mayor personalización en el comportamiento de los prototipos, incluyendo la integración de efectos hápticos en dispositivos móviles. A partir de esta funcionalidad, se desarrolló un procedimiento mediante el cual los usuarios pueden trasladar sus diseños de Figma a Framer y, posteriormente, habilitar la ejecución de vibraciones dentro del prototipo.

El procedimiento propuesto consta de los siguientes pasos:

1. Creación de una cuenta en Framer: Dado que la plataforma requiere autenticación para la gestión de proyectos, el usuario debe registrarse e iniciar sesión en Framer.

2. Migración del diseño desde Figma: Se proporciona un método estructurado para transferir los diseños creados en Figma hacia Framer, asegurando la conservación de la jerarquía de elementos y estilos aplicados. Esto se lleva a cabo utilizando un plugin llamado Figma to HTML, donde simplemente se copia y pega el diseño.

3. Incorporación de un botón de activación de vibraciones: Se provee al usuario un botón preconfigurado a través de un enlace, el cual debe ser insertado en el diseño dentro de Framer. Este botón actúa como disparador para la ejecución de la vibración.

4. Inserción del arreglo de vibraciones: Finalmente, el usuario debe pegar el arreglo generado previamente en el código del prototipo en Framer, lo que permitirá que, al interactuar con el diseño en un dispositivo móvil, se ejecute la vibración correspondiente.

Si bien este proceso puede resultar engorroso para el diseñador, ya que implica la utilización de múltiples herramientas y la ejecución de varios pa-

sos adicionales en comparación con un flujo de trabajo convencional dentro de Figma, representa una solución práctica ante la limitación impuesta por la plataforma. Gracias a esta estrategia, es posible experimentar la retroalimentación háptica en un entorno más cercano a la realidad, lo que permite evaluar de manera más precisa cómo se sentirán las vibraciones en un dispositivo físico antes de su implementación final. No obstante, se tiene plena conciencia de que esta transición entre herramientas puede generar cierta fricción en el usuario final, especialmente en aquellos diseñadores que están acostumbrados a trabajar exclusivamente dentro de un único ecosistema y prefieren evitar cambios en su metodología de trabajo.

El hecho de tener que migrar de Figma a Framer no solo introduce una capa adicional de complejidad, sino que también implica una curva de aprendizaje para quienes no están familiarizados con esta segunda plataforma. Además, la necesidad de exportar los diseños, configurar los elementos interactivos y asegurarse de que la vibración se ejecute correctamente puede resultar una tarea tediosa, sobre todo en las primeras iteraciones del proyecto. Sin embargo, dadas las restricciones técnicas actuales y la ausencia de soporte nativo para vibraciones en Figma, esta metodología se presenta como la alternativa más viable y efectiva para validar e iterar sobre experiencias hápticas dentro del proceso de diseño de interfaces. A largo plazo, esta estrategia no solo permite una mayor fidelidad en la simulación de las interacciones táctiles, sino que también abre la posibilidad de seguir explorando nuevas herramientas y enfoques que optimicen la integración de háptica en entornos de diseño digital.

5.2.9. Unificación del flujo de trabajo: migración del plugin a Framer

Desde un principio, se tuvo en claro que el flujo de trabajo resultante, basado en la combinación de Figma y Framer para la validación de vibraciones, sería poco práctico y engorroso para los usuarios finales. Esta fragmentación en el proceso no solo afectaba la fluidez del diseño, sino que también comprometía la adopción y efectividad del plugin dentro de un entorno profesional. Ante esta problemática, se concluyó que el proyecto, en su estado actual, no cumplía con las expectativas establecidas en términos de usabilidad e integración, lo que llevó a replantear la estrategia y buscar una solución más eficiente.

Por este motivo, se decide migrar completamente el plugin a Framer, unificando así el flujo de trabajo en una única plataforma que permitiera diseñar, probar y ajustar las interacciones hápticas sin necesidad de herramientas intermedias. Sin embargo, es importante destacar que Framer no fue la primera elección al inicio del desarrollo de esta tesina, ya que en aquel momento la plataforma no contaba con una herramienta específica para la creación de plugins. Esta limitación condujo a optar inicialmente por Fig-

ma, que ofrecía una API estable y ampliamente utilizada en la comunidad de diseño. No obstante, con la posterior incorporación de funcionalidades avanzadas en Framer, su adopción se convirtió en la mejor alternativa para garantizar una experiencia de usuario más intuitiva y eficiente.

En primera instancia, se investigó el funcionamiento de los plugins en Framer, dado que se trata de una característica completamente nueva para la plataforma. Para ello, se realizó un análisis de la documentación oficial y se exploraron distintos recursos que permitieran comprender la arquitectura y las herramientas disponibles para el desarrollo. Una vez reunida y asimilada la información relevante, se dio inicio al proceso de implementación del plugin, siguiendo las mejores prácticas recomendadas por Framer.

Tras este período de exploración teórica, se configuró el entorno de desarrollo para comenzar con la implementación del plugin. Para ello, el siguiente comando fue ejecutado en la terminal del entorno de desarrollo:

Código 5.19 Comando para crear un nuevo plugin en Framer

```
1 npm create framer-plugin@latest
```

Este comando inicializó la estructura base del proyecto, generando automáticamente una carpeta con la configuración necesaria para el desarrollo del plugin (ver Figura 5.7).

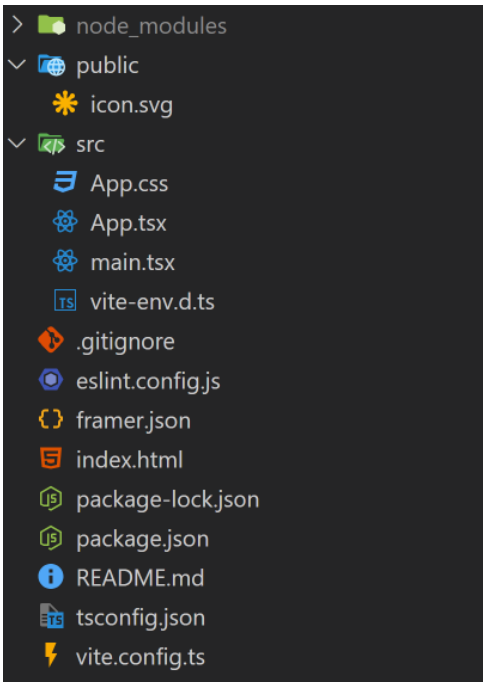

Figura 5.7: Estructura inicial del plugin en Framer

Una vez creada esta estructura, se accede a la carpeta del proyecto y, para

iniciar el entorno de desarrollo y permitir que Framer detectara el plugin en tiempo real, se ejecuta:

Código 5.20 Ejecutar el plugin en modo desarrollo

```
npm run dev
```

Este proceso permite realizar modificaciones en el código y visualizar instantáneamente los cambios dentro de Framer, facilitando la iteración y depuración del proyecto.

Framer estructura la interfaz principal del plugin dentro del archivo *App.tsx*, donde se gestionan los elementos interactivos y la lógica del sistema. En este archivo, la función framer.showUI() se emplea para abrir la interfaz gráfica y definir su posición y dimensiones dentro del entorno de Framer. La siguiente línea de código ejemplifica cómo posicionar la ventana en la esquina superior izquierda con un ancho de 240 píxeles y una altura de 245 píxeles:

Código 5.21 Código para mostrar la interfaz gráfica del plugin en Framer

```
framer.showUI({ position: "top left", width: 240, height: 245 })
```

Con el entorno de desarrollo correctamente configurado, se da por iniciada la implementación del plugin en Framer. Uno de los aspectos que facilitó esta transición fue la similitud del proceso de desarrollo en Framer con el que ya se había experimentado en Figma. Esto permitió aprovechar los conocimientos previos y reducir la curva de aprendizaje, haciendo foco directamente en la adaptación del código y la integración de las funcionalidades existentes.

Afortunadamente, el proceso de migración no representó un esfuerzo excesivo en términos de desarrollo, ya que el código que había sido implementado en Figma se encontraba completamente funcional y probado. La principal tarea consistió en reformular la lógica existente para ajustarla a la infraestructura de Framer, lo que implicó reestructurar el código desarrollado en TypeScript y adaptarlo al paradigma de React, utilizado en el desarrollo de plugins con Framer.

Durante esta transición, el foco se mantuvo principalmente en reorganizar la arquitectura del plugin para garantizar una integración óptima en el nuevo entorno. Se realizaron ajustes en la forma en que se manejaban los eventos, se optimizó la interacción entre los componentes y se implementaron mecanismos para mejorar la eficiencia del procesamiento de datos. Gracias a la modularidad del código original y a las similitudes entre ambas tecnologías, esta conversión pudo llevarse a cabo de manera eficiente, minimizando los tiempos de implementación y asegurando que todas las funcionalidades se mantuvieran intactas en el nuevo entorno.

Finalmente, con la migración completada, fue posible consolidar un flujo de trabajo más ágil y unificado dentro de Framer, eliminando la necesidad de depender de múltiples herramientas y simplificando el proceso de validación de interacciones hápticas en el diseño de interfaces.

5.3. Storyboards

Para ilustrar visualmente el proceso de diseño y desarrollo del plugin, una serie de storyboards han sido creadas representando las etapas clave de su evolución. Estos storyboards ofrecen una visión general de cómo se concibieron y se implementaron las diferentes funcionalidades del plugin, desde su creación inicial en Figma hasta su migración completa a Framer. A continuación, se presentan organizados en dos subsecciones: la primera, centrada en el diseño inicial en Figma, y la segunda, que muestra el proceso de adaptación y desarrollo final en Framer. Estos storyboards proporcionan un contexto visual para comprender mejor las decisiones tomadas durante el desarrollo del plugin y cómo estas decisiones influyeron en su funcionalidad y usabilidad.

1. Agregar haptic plugin
2. Reproducir patrón de audio
3. Modificar etiquetas
4. Añadir patrón de audio al plugin
5. Historia completa de uso del plugin hasta prototipo con vibración móvil

5.3.1. Figma

1. Agregar haptic plugin

Página inicial de un proyecto existente en Figma (ver Figura 5.8):

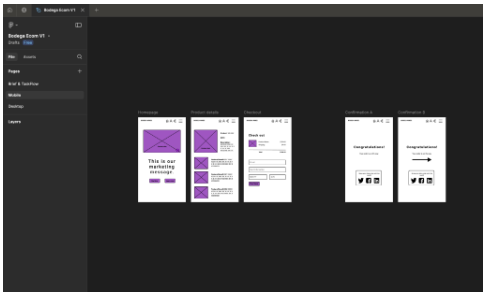

Figura 5.8: Paso 1

Búsqueda del plugin (ver Figura 5.9 y Figura 5.10):

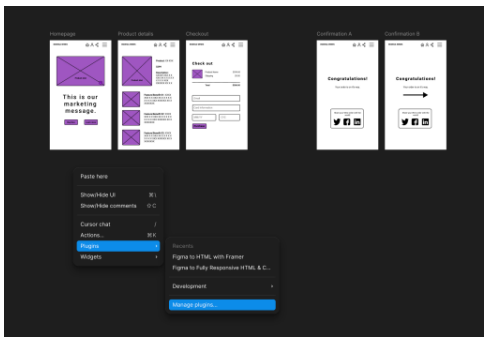

Figura 5.9: Paso 2.1

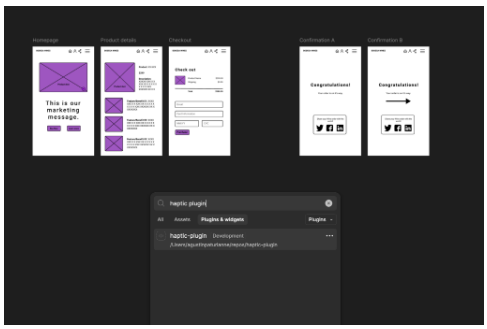

Figura 5.10: Paso 2.2

Plugin incorporado al workspace (ver Figura 5.11):

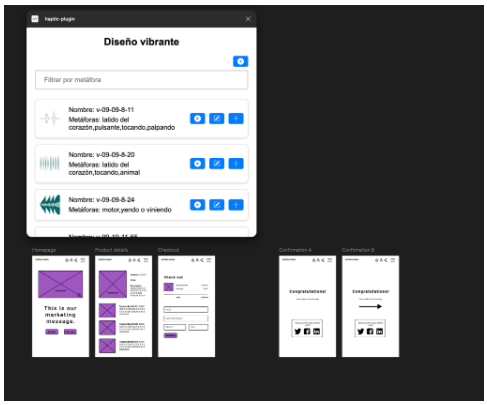

Figura 5.11: Paso 3

2. Reproducir patrón de audio

Reproducción del patrón de audio (ver Figura 5.12)

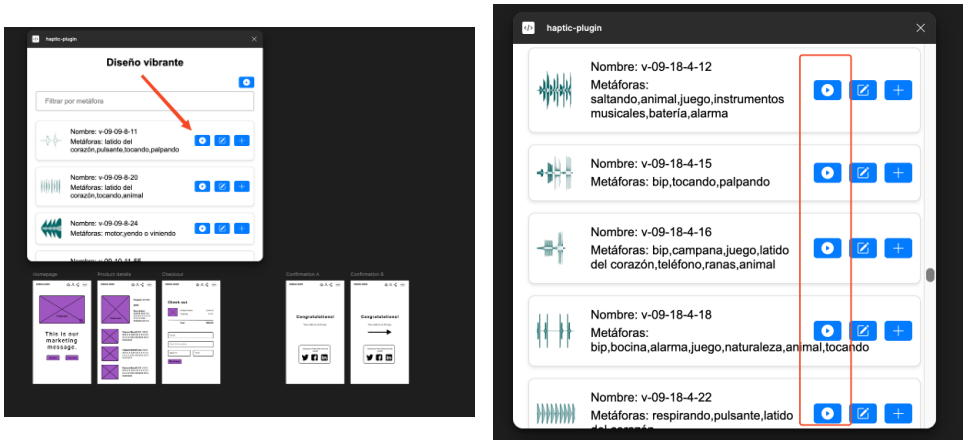

Figura 5.12: Botón de reproducción

3. Modificar etiquetas

Página inicial de sonidos (ver Figura 5.13):

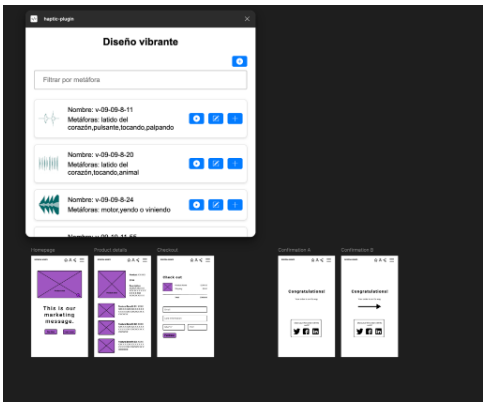

Figura 5.13: Estado inicial

Una vez reproducido el sonido que se desea editar, se presiona el botón “Modificar etiquetas” (ver Figuras 5.14, 5.15, 5.16 y 5.17):

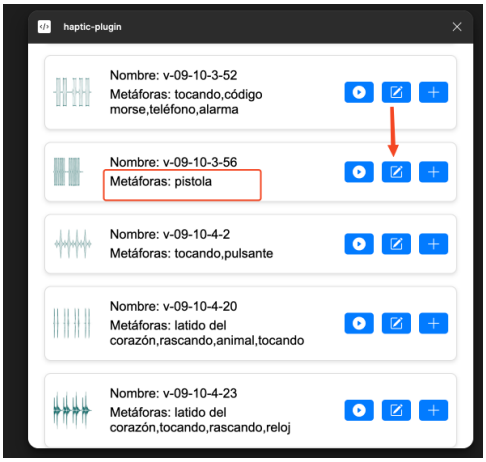

Figura 5.14: Botón para modificar etiquetas

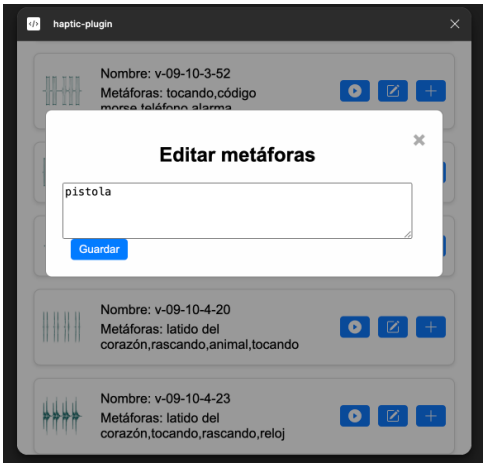

Figura 5.15: Modal de modificación desplegado

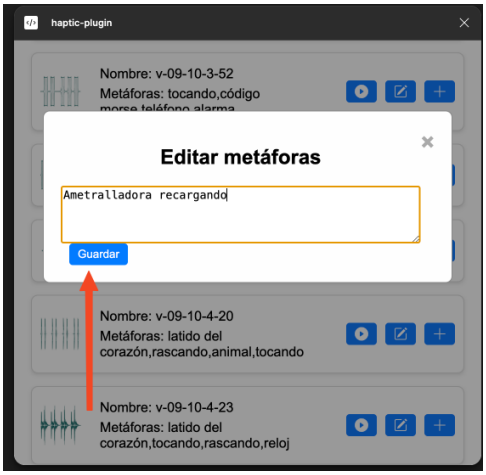

Figura 5.16: Botón para guardar las modificaciones realizadas

Metáforas modificadas:

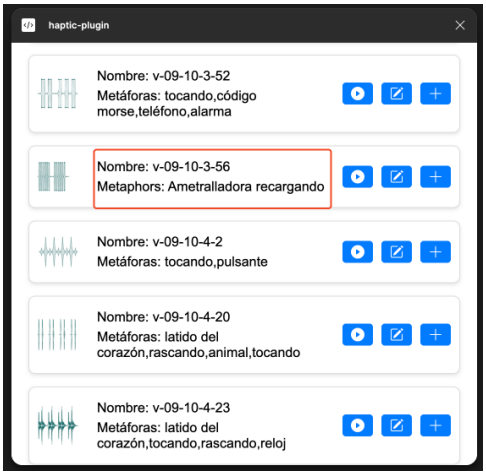

Figura 5.17: Cambios reflejados al instante

4. Añadir patrón de audio al plugin

Página inicial de sonidos (ver Figura 5.18):

Figura 5.18: Botón para añadir un nuevo patrón

Campos a completar (ver Figura 5.19 y 5.20):

Figura 5.19: Modal desplegado a completar

Figura 5.20: Campos rellenados como ejemplo

Una vez completados, se clickea el botón 'Subir' (ver Figura 5.21):

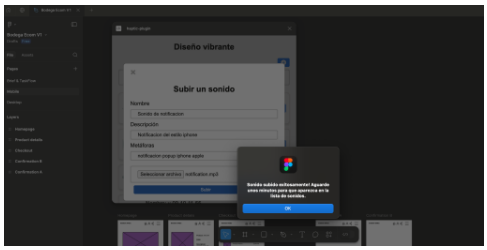

Figura 5.21: Patrón subido

Sonido propio agregado al listado (ver Figura 5.22):

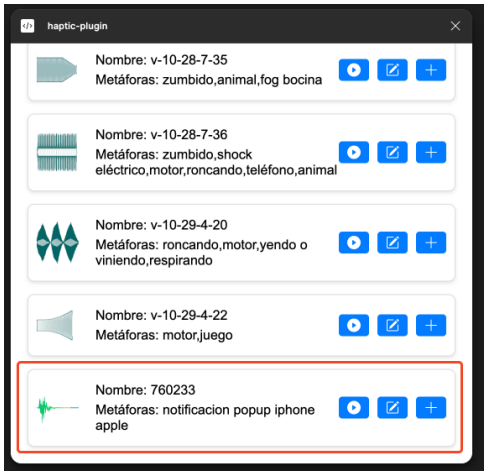

Figura 5.22: Resultado final

5. Historia completa de uso del plugin hasta prototipo con vibración móvil

Página inicial del plugin (ver Figura 5.23):

Figura 5.23: Estado inicial

Al hacer clic en el + de cualquier sonido, se generará el siguiente frame donde se encuentran detallados los pasos a seguir (ver Figura 5.24):

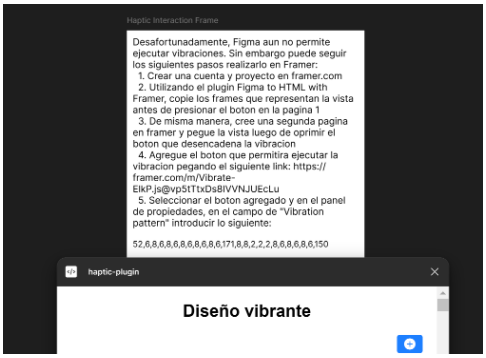

Figura 5.24: Pasos a seguir

Ingresar a framer.com y clickear 'Sign up for free' (ver Figura 5.25)

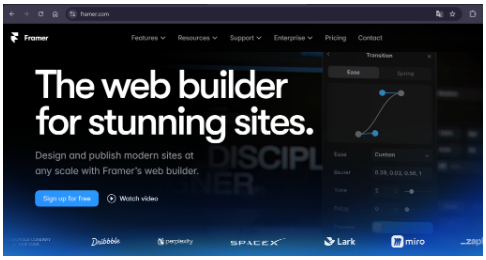

Figura 5.25: Inicio de framer

Completar los pasos a seguir para crear una cuenta (ver Figura 5.26).

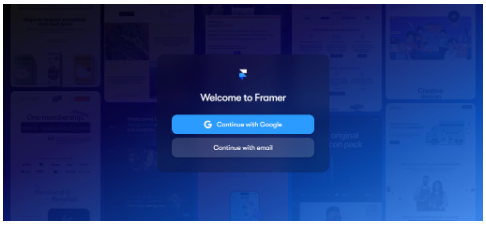

Figura 5.26: Seguir los pasos dentro de Framer

Una vez creada la cuenta, clickear 'New Project' (ver Figura 5.27).

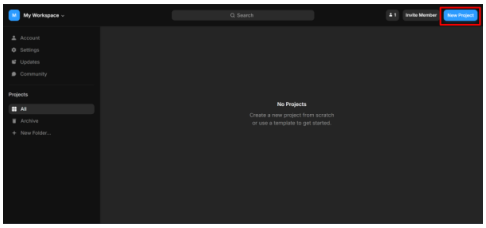

Figura 5.27: Crear un nuevo proyecto

Volver a Figma y buscar el plugin "Figma to HTML with Framer" desde 'Manage plugins' (ver Figura 5.28).

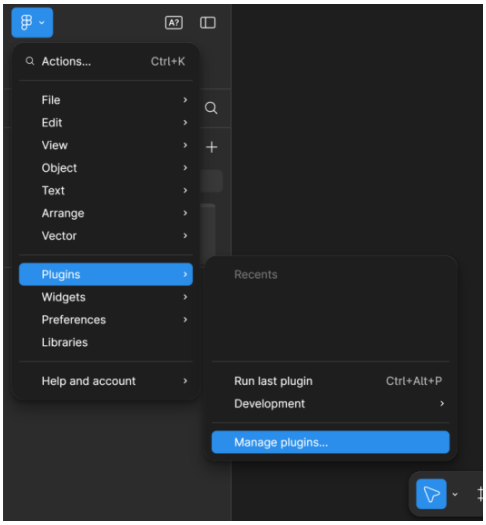

Figura 5.28: Búsqueda del plugin que traslada el diseño a Framer

Seleccionar el plugin (ver Figura 5.29).

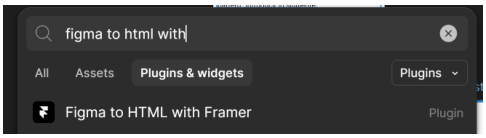

Figura 5.29: Añadir el plugin

Una vez añadido el plugin, seleccionar la primera vista (previa a la interacción) (ver Figura 5.30).

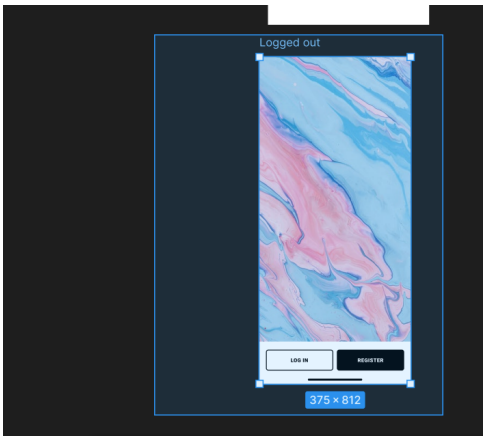

Figura 5.30: Selección de la vista

Luego clickear sobre el plugin agregado (ver Figura 5.31).

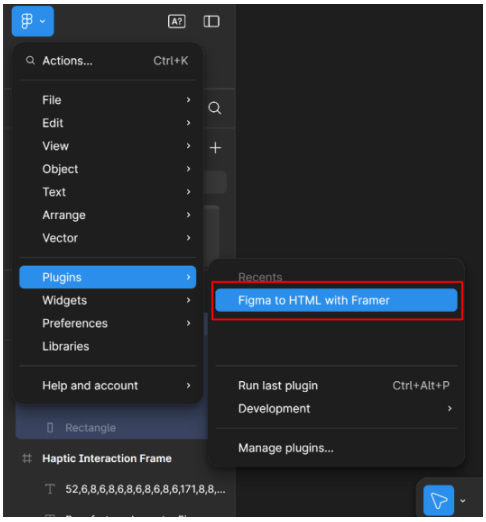

Figura 5.31: Seleccionar el plugin

Una vez hecho esto, aparecerá el siguiente cartel (ver Figura 5.32).

Figura 5.32: Resultado de copiar la vista

Al volver a Framer, y sobre la página 'Home' pegar la vista previa a la interacción (ver Figura 5.33).

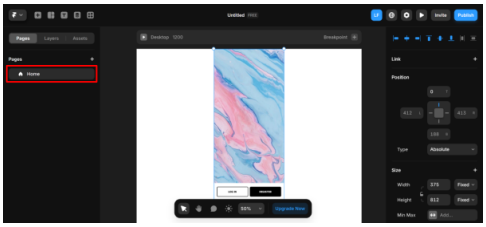

Figura 5.33: Vista pegada en Framer

Luego, copiar utilizando el plugin la vista que debe visualizarse posterior a la interacción (ver Figuras 5.34 y 5.35).

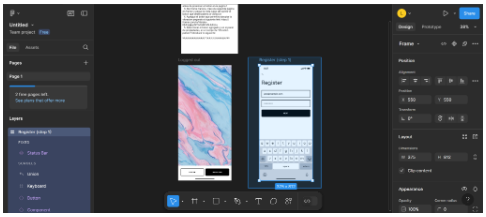

Figura 5.34: Repetición del proceso con la segunda vista

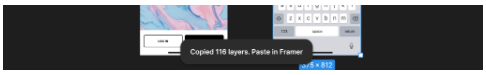

Figura 5.35: Vista copiada exitosamente

Volviendo a Framer, crear una nueva página (ver Figura 5.36).

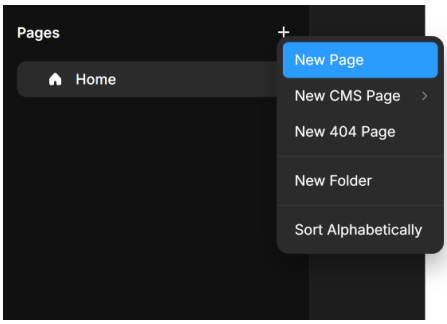

Figura 5.36: Nueva pagina en Framer

En la nueva página pegar la vista a mostrar posterior a la interacción (ver Figura 5.37).

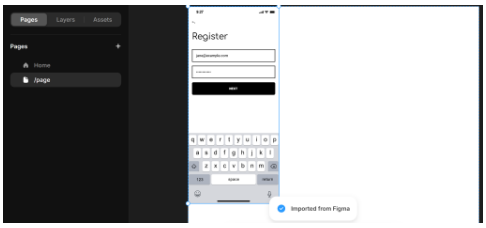

Figura 5.37: Segunda vista en la nueva página

Siguiendo con los pasos, copiar el botón que va a ejecutar la vibración (ver Figura 5.38).

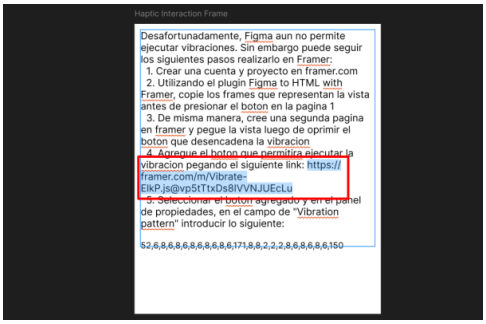

Figura 5.38: Enlace del botón a utilizar para disparar la vibración

Volver a Framer y pulsar las teclas ctrl + v para pegar el botón (ver Figura 5.39).

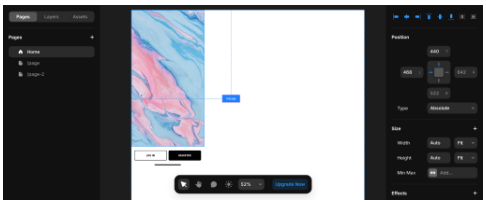

Figura 5.39: Pegar el boton en Framer

En Figma, copiar el arreglo generado (ver Figura 5.40).

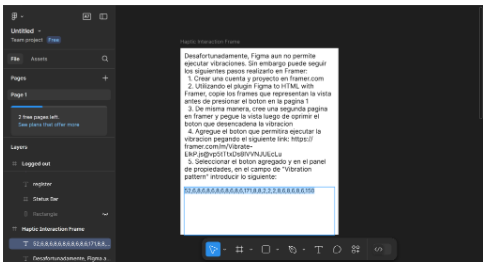

Figura 5.40: Arreglo que representa la vibración seleccionada

Volviendo a Framer, seleccionar el botón agregado, deslizando hacia abajo en la sección derecha de la página (donde se muestran las propiedades del botón) y pegar la cadena copiada en el campo “Vibration pattern” del botón añadido (ver Figura 5.41).

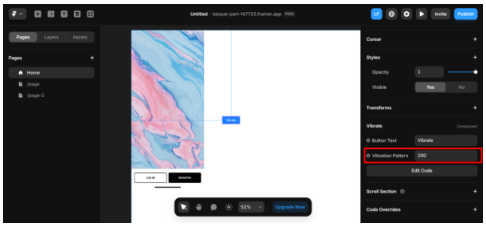

Figura 5.41: Campo donde se agrega la cadena copiada

Editar el botón 'Vibrate' para que siga los estilos de la vista, y linkear el botón a la página post interacción para que además de vibrar, se pase a la vista siguiente (ver Figura 5.42).

Figura 5.42: Edición del botón

Por último, hacer click en 'Publish' para luego acceder a la demo desde un celular (ver Figura 5.43).

Figura 5.43: Publicar los cambios realizados

Ahora, solo queda copiar la url generada y acceder a la misma desde un celular Android para poder percibir la vibración (ver Figura 5.44).

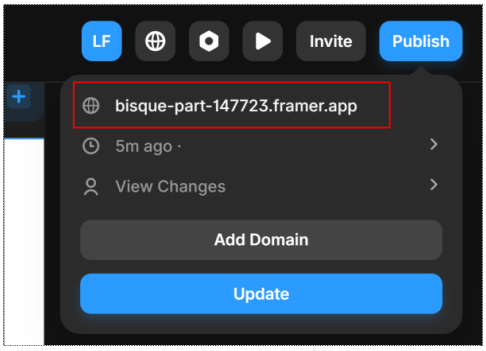

Figura 5.44: Enlace al diseño final

5.3.2. Framer

1. Agregar haptic plugin

Página inicial de un proyecto existente en Framer (ver Figura 5.45).

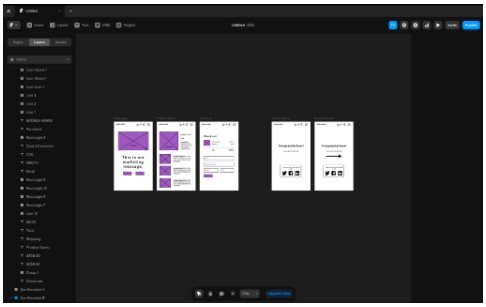

Figura 5.45: Inicio de Framer

Búsqueda del plugin (ver Figura 5.46).

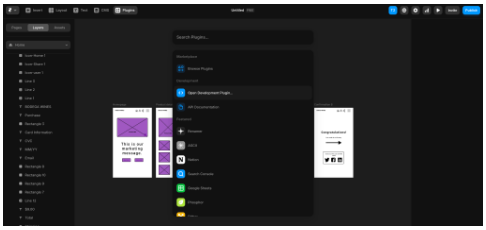

Figura 5.46: Agregar el plugin

Plugin incorporado al workspace (ver Figura 5.47).

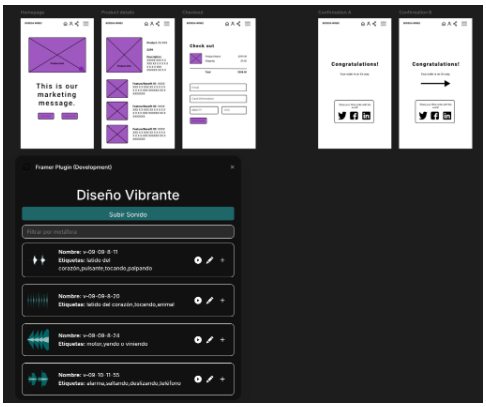

Figura 5.47: Plugin incorporado en Framer

2. Reproducir patrón de audio

Página inicial de un proyecto existente en Framer (ver Figuras 5.48).

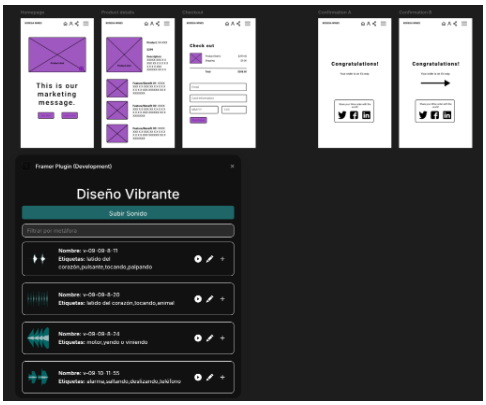

Figura 5.48: Estado inicial dentro de Framer

Reproducción del patrón de audio (ver Figuras 5.49 y 5.50).

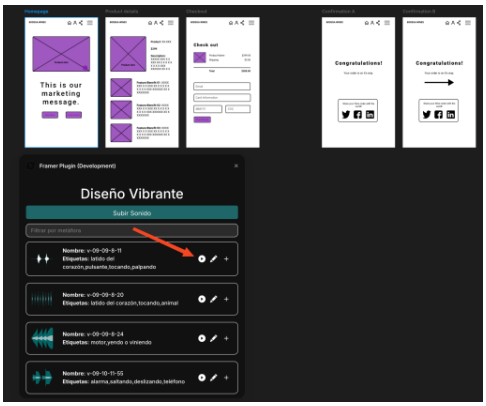

Figura 5.49: Botón de reproducción

Figura 5.50: Botones de reproducción

3. Modificar etiquetas

Página inicial de sonidos (ver Figura 5.51).

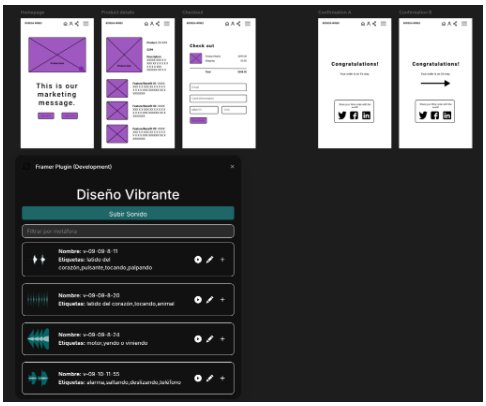

Figura 5.51: Estado inicial en Framer

Una vez reproducido el sonido que se desea editar, presionar el botón "Modificar etiquetas" (ver Figuras 5.52, 5.53 y 5.54).

Figura 5.52: Botón para modificar etiquetas

Figura 5.53: Modal de edición de etiquetas

Figura 5.54: Botón para guardar los cambios

Etiquetas modificadas (ver Figura 5.55).

Figura 5.55: Cambios reflejados

4. Añadir patrón de audio al plugin

Página inicial de sonidos (ver Figura 5.56).

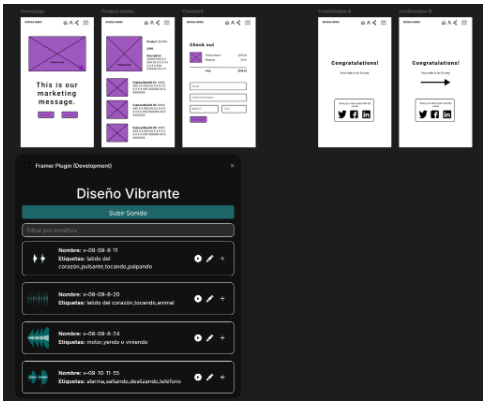

Figura 5.56: Estado inicial en Framer

Seleccionar el botón 'Subir sonido' (ver Figura 5.57).

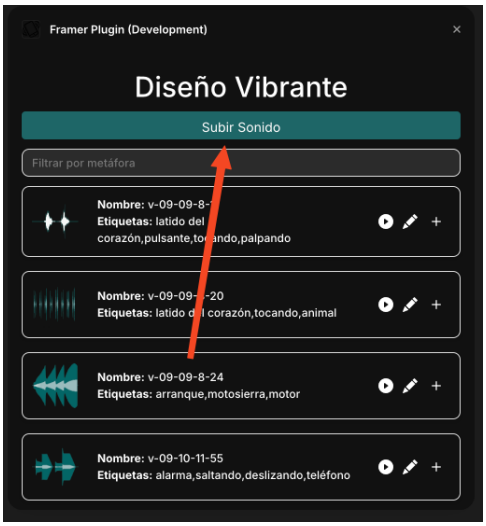

Figura 5.57: Botón de subida de un sonido

Aparecen los campos a completar (ver Figuras 5.58 y 5.59).

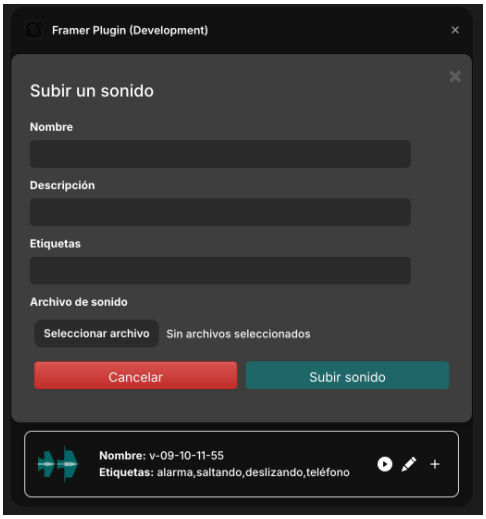

Figura 5.58: Modal para subir un sonido desplegado

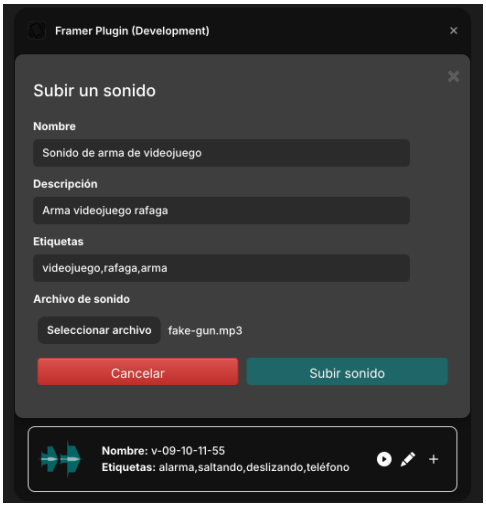

Figura 5.59: Campos rellenados a modo de ejemplo

Una vez completados, se clickea el botón 'Subir' (ver Figura 5.60).

Figura 5.60: Botón de confirmación para subir el sonido

Sonido propio agregado al listado (ver Figura 5.61).

Figura 5.61: Resultado reflejado en el listado

5. Historia completa de uso del plugin hasta prototipo con vibración móvil

Página inicial del plugin (ver Figura 5.62).

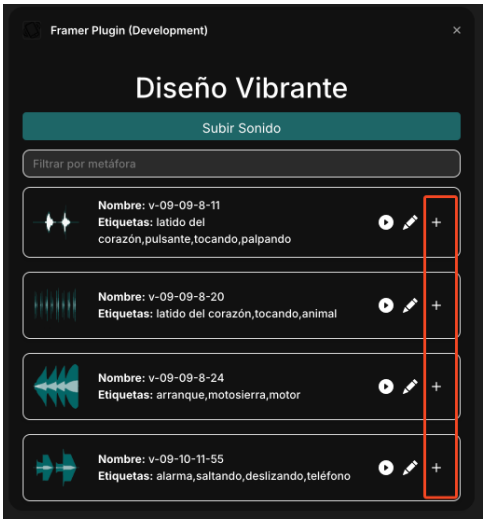

Figura 5.62: Estado inicial en Framer

Al clickear el + de cualquier ítem del catálogo, se generará el botón con la vibración relacionada al patrón de sonido (ver Figura 5.63).

Figura 5.63: Botón para agregar el sonido seleccionado

Como se ve en la imagen, el botón se crea exitosamente y ya posee el valor de la vibración (ver Figura 5.64).

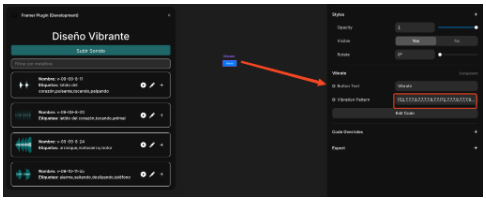

Figura 5.64: Botón con sonido agregado al diseño

Por último, agregar el botón en el frame deseado y hacer click en Publish para luego poder acceder a la demo desde un celular (ver Figura 5.65).

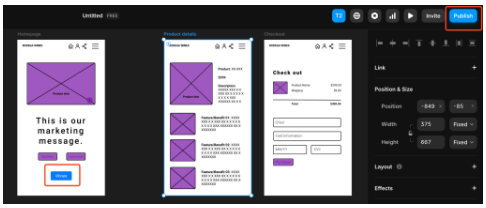

Figura 5.65: Publicar los cambios realizados

Ahora, solo queda copiar el enlace generado y acceder a la misma desde un celular Android para poder percibir la vibración (ver Figura 5.66).

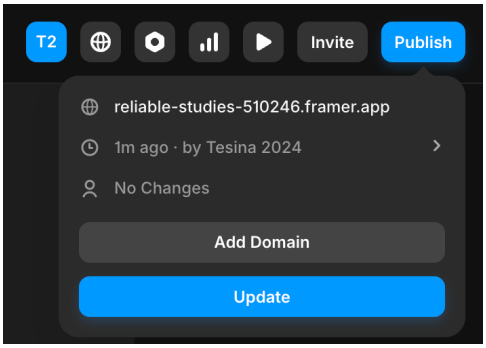

Figura 5.66: Enlace generado para acceder desde un celular

Capítulo 6

Pruebas de usuario

En este capítulo, se presentan en detalle las pruebas diseñadas específicamente para los usuarios finales, cuyo propósito principal es recopilar su retroalimentación sobre lo desarrollado. Estas pruebas permiten evaluar la experiencia de uso, identificar posibles mejoras y detectar cualquier inconveniente que pueda afectar la funcionalidad o usabilidad del plugin. A través de la interacción directa con los usuarios, se busca obtener comentarios valiosos que contribuyan a optimizar el desarrollo y asegurar que el resultado final cumpla con sus expectativas y necesidades.

6.1. Diseño de las pruebas

Esta guía está diseñada para evaluar la usabilidad del plugin háptico en Framer. El objetivo es observar cómo los diseñadores interactúan con el plugin en escenarios comunes de diseño de aplicaciones móviles para comercio electrónico y juegos, y recopilar información sobre su facilidad de uso, eficiencia y satisfacción general.

A continuación, se presentan las pruebas que simulan diferentes tareas de un proceso de diseño de interacciones móviles en las que un diseñador podría utilizar el plugin.

6.1.1. Contexto

Participantes: Diseñadores con experiencia en diseño de interfaces para dispositivos móviles.

Duración: Cada prueba tendrá una duración máxima de 1 hora por participante.

Consentimiento: Se solicitará el consentimiento explícito de cada participante para utilizar los datos recopilados en la prueba en el documento de tesis.

Software y Hardware:

- Computadora con acceso a Framer y al plugin háptico instalado.
- Teléfono móvil Android para probar las vibraciones en tiempo real.
- Software de grabación de pantalla para registrar las interacciones del usuario: Google Meet.

Entorno de Prueba: Remoto.

6.1.2. Tareas asignadas y objetivos

Tarea 1: Diseñar una Interacción de Confirmación de Compra en una Aplicación de Comercio Electrónico.

Objetivo: Evaluar la capacidad del diseñador para agregar efectos de vibración a una interacción de confirmación de compra y la facilidad de uso de la biblioteca de efectos predefinidos.

Descripción de la tarea:

- En Framer, cree un nuevo proyecto de diseño para una aplicación móvil de comercio electrónico.
- Diseñe una pantalla de confirmación de compra que muestre un resumen del pedido y un botón de `"Confirmar"`.
- Utilice el plugin háptico para agregar un efecto de vibración al botón de `"Confirmar"`. Elija un patrón de vibración de la galería predefinida que considere apropiado para indicar que la compra se ha realizado con éxito.
- Pruebe la interacción en su teléfono móvil para verificar el efecto de vibración.
- Comente sobre la facilidad de encontrar y seleccionar el efecto de vibración adecuado. ¿Le resultó intuitivo el proceso? ¿Qué mejoras sugeriría?

Tarea 2: Buscar un Patrón de Vibración Específico para una Acción en un Juego Móvil

Objetivo: Evaluar la capacidad del diseñador para navegar y buscar patrones de vibración específicos dentro de la biblioteca del plugin, en el contexto de un juego móvil.

Descripción de la tarea:

- Imagine que está diseñando un juego móvil y necesita un patrón de vibración específico para indicar que el jugador ha recolectado un objeto valioso. Las características del patrón deben ser las siguientes: Vibración corta y suave que transmita una sensación de recompensa.
- Utilice la función de búsqueda del plugin para encontrar un patrón que coincida con estas características.
- Describa los pasos que siguió para buscar el patrón. ¿Le resultó fácil encontrar lo que buscaba? ¿Qué términos de búsqueda utilizó?
- Si encuentra un patrón que se ajuste a sus necesidades, agréguelo a la acción de recolectar el objeto valioso. Si no, mencione por qué le parece que no encontró lo que buscaba.

- Comente sobre la eficacia de la función de búsqueda y la organización de la biblioteca de efectos. ¿Qué mejoras sugeriría para facilitar la búsqueda de patrones específicos?

Tarea 3: Adaptación o Creación de un Nuevo Patrón de Vibración para un Feedback Personalizado en un Juego Móvil

Objetivo: Evaluar la capacidad del diseñador para adaptar un patrón de vibración existente o crear uno nuevo que se ajuste a sus necesidades, proporcionando un feedback único en un juego móvil.

Descripción de la tarea:

- Imagine que ha buscado en el catálogo y no ha encontrado ningún patrón de vibración que se ajuste a las características que busca (por ejemplo, una vibración intensa y rítmica para indicar que un ataque especial está listo para ser utilizado).
- Utilice las herramientas del plugin para crear un nuevo patrón de vibración con las características deseadas. Si lo prefiere, puede adaptar un patrón existente.
- Describa el proceso de creación o adaptación del patrón. ¿Le resultó fácil utilizar las herramientas de edición? ¿Qué limitaciones encontró?
- Agregue el patrón creado o adaptado a la acción de activar el ataque especial en su juego móvil y pruébelo en su teléfono móvil.
- Comente sobre la flexibilidad y facilidad de uso de las herramientas de edición. ¿Qué mejoras sugeriría para facilitar la creación y personalización de patrones de vibración?

6.1.3. Instrucciones Generales

Se entrega a cada sujeto la siguiente lista de instrucciones luego de su consentimiento informado.

1. Lea detenidamente las instrucciones de cada prueba antes de comenzar.
2. Realice las tareas de la forma más natural posible, como si estuviera trabajando en un proyecto real.
3. Comente en voz alta sus pensamientos, decisiones y cualquier problema que encuentre durante la prueba.
4. No dude en hacer preguntas si algo no está claro.
5. El evaluador observará y tomará notas durante la prueba.

6.2. Ejecución de las pruebas

Para la ejecución se reclutaron 3 sujetos con los siguientes datos demográficos;

ID	Edad	Género	Nacionalidad	Actividad	Años de experiencia
1	23	Masculino	Argentina	Desarrollador Full-Stack	3
2	22	Femenino	Argentina	Desarrollador Web	2
3	26	Masculino	Argentina	Desarrollador y Diseñador UX	5

Cuadro 6.1: Datos demográficos de los participantes

A continuación se incluye una descripción completa de la ejecución de las tres tareas por cada uno de los sujetos. Más adelante se discuten los hallazgos correspondientes.

6.2.1. Usuario 1

Resultados de la Tarea 1: Diseño de una Interacción de Confirmación de Compra en una Aplicación de Comercio Electrónico

Durante la primera prueba de usabilidad, el participante, un programador con experiencia avanzada en diseño, completó exitosamente las tareas propuestas de manera remota y dio su consentimiento a publicar lo observado. La prueba le llevó aproximadamente 15 minutos.

A continuación, se detallan sus acciones:

Desarrollo de la Prueba

- **Apertura del Plugin:** El usuario inició correctamente el plugin háptico dentro de Framer.
- **Exploración del Catálogo de Sonidos:** Inicialmente, intentó registrarse en Freesound, pero identificó que este paso no era necesario para la prueba. Posteriormente, exploró la biblioteca de sonidos, reproduciendo varias opciones antes de seleccionar una que consideró adecuada para la interacción (ver Figura 6.1).

Figura 6.1: Exploración inicial del usuario

- **Selección y Aplicación del Efecto:** Tras elegir un sonido, lo agregó correctamente al diseño de la pantalla de confirmación de compra. Luego, adaptó el componente para que se ajustara a su diseño específico (ver Figura 6.2).

Figura 6.2: Diseño realizado

- **Prueba en Dispositivo Móvil:** Publicó el proyecto en la web para probar la interacción en su teléfono móvil Android. Al evaluar la vibración, notó que el efecto funcionaba correctamente, aunque le pareció demasiado suave para transmitir de manera efectiva la confirmación de compra.

Resultados de la Tarea 2: Búsqueda de un Patrón de Vibración Específico para una Acción en un Juego Móvil

En esta segunda prueba, el participante diseñó una interacción para representar la obtención de un logro en un juego móvil, utilizando el plugin háptico. La prueba le llevó aproximadamente 20 minutos.

A continuación, se detallan sus acciones:

Desarrollo de la Prueba

- **Diseño de la Interfaz:** Antes de comenzar la búsqueda del patrón de vibración, el usuario diseñó una interfaz para estructurar adecuadamente la prueba.
- **Exploración Inicial:** El usuario inicialmente intentó encontrar un patrón de vibración adecuado deslizándose manualmente por el catálogo de VibViz, sin notar la existencia de la función de búsqueda (ver Figura 6.3).

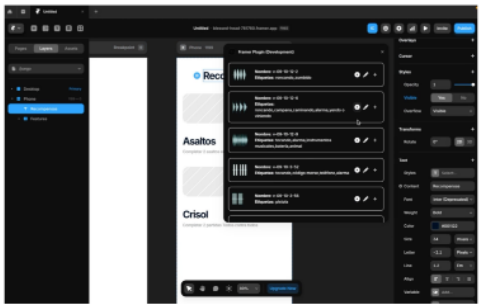

Figura 6.3: Exploración inicial del usuario para la segunda prueba

- **Descubrimiento de la Búsqueda:** Luego de preguntar acerca de la funcionalidad, se le informó que la opción de búsqueda estaba disponible. A partir de ese momento, comenzó a utilizar filtros con palabras clave como `"corta" ,"suave" y "vibracion corta"` hasta encontrar un patrón que se adaptaba a sus necesidades.
- **Selección y Prueba del Patrón:** El usuario seleccionó el patrón de vibración y lo incluyó en su diseño. Publicó la interacción para acceder a ella desde un dispositivo móvil Android y probar la vibración en tiempo real
- **Identificación de Problema:** Al probar el efecto, notó que la vibración era muy similar a la utilizada en la Prueba 1. Tras revisar, identificó que había seleccionado involuntariamente el mismo patrón.
- **Repetición del Proceso:** Para corregir esto, realizó nuevamente la búsqueda y seleccionó un nuevo patrón, el cual probó en el dispositivo móvil y consideró adecuado para la interacción.

Resultados de la Tarea 3: Proceso de Autenticación y Subida de Sonidos Personalizados

En esta tercera prueba, el participante evaluó la experiencia de autenticación en FreeSound y la funcionalidad de subida de sonidos personalizados dentro del plugin háptico. Tardó un total de 15 minutos.

A continuación, se detallan sus acciones:

Desarrollo de la Prueba

- **Registro e Inicio de Sesión en FreeSound:** El usuario comenzó registrándose en FreeSound y luego inició sesión en la plataforma.
- **Dificultad para Obtener el Token de Autenticación:** Tras iniciar sesión, no identificó de inmediato cómo obtener el token necesario para la autenticación con el plugin. Descubrió por su cuenta que debía volver a hacer clic en `"registrarse/iniciar sesión"` dentro del plugin para ser redirigido al enlace de autorización y obtener el token.
- **Subida de un Sonido Personalizado:** Procedió a subir un sonido con los siguientes datos:
  - Nombre: "SuperPoder"
  - Descripción: "Super"
  - Etiquetas: "juego, ataque, fatality"
  - **L**a subida se realizó exitosamente en el primer intento (ver Figura 6.4 , 6.5 y 6.6).

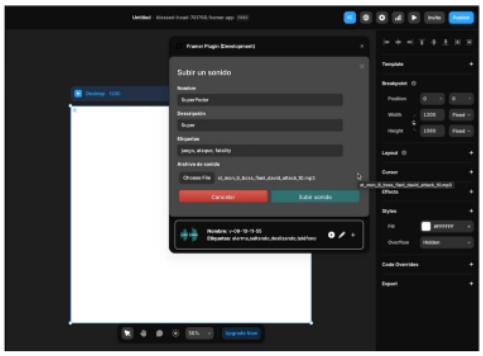

Figura 6.4: Datos del sonido a subir

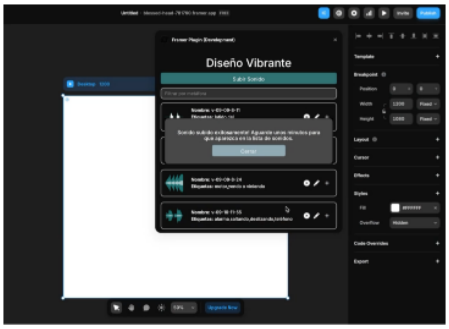

Figura 6.5: Notificación de subida exitosa

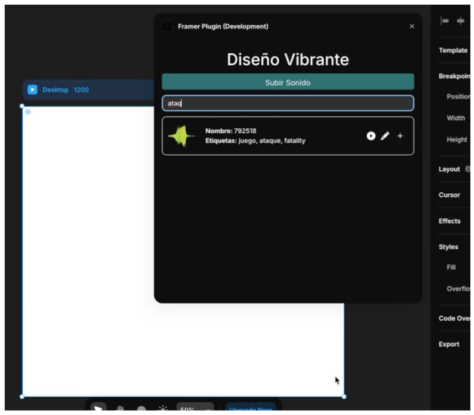

Figura 6.6: Sonido accedido desde la biblioteca

6.2.2. Usuario 2

Resultados de la Tarea 1: Diseño de una Interacción de Confirmación de Compra en una Aplicación de Comercio Electrónico

Resultados: Durante la primera prueba de usabilidad, la participante completó exitosamente las tareas propuestas sin dificultades significativas. La prueba le llevó aproximadamente 10 minutos. A continuación, se detallan sus acciones:

Desarrollo de la Prueba

- **Apertura del Plugin:** La usuaria inició correctamente el plugin en Figma sin inconvenientes.
- **Exploración del Catálogo de Sonidos:** Navegó por la biblioteca de sonidos, reproduciendo varias opciones antes de seleccionar una. Le pareció útil que el catálogo incluya gráficos, ya que considera que esto complementa la experiencia auditiva (ver Figura 6.7).

Figura 6.7: Exploración inicial del usuario 2

- **Selección y Aplicación del Efecto:** Logró agregar un sonido correctamente, pero se demoró unos minutos en modificar el tamaño y aspecto del botón generado por el plugin hasta modificarlo de manera deseada (ver Figura 6.8).

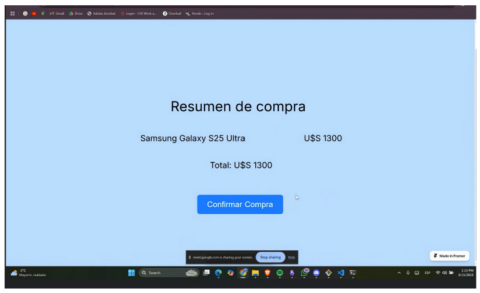

Figura 6.8: Diseño realizado por el usuario 2 para la tarea 1

Vista desde el celular (ver Figura 6.9).

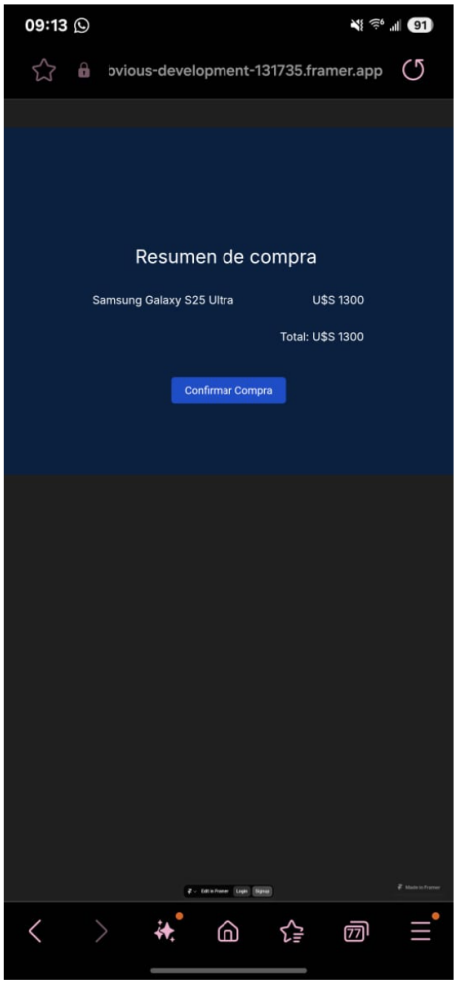

Figura 6.9: Resultado final en el celular del usuario 2

Resultados de la Tarea 2: Búsqueda de un Patrón de Vibración Específico para una Acción en un Juego Móvil

Desarrollo de la Prueba

- **Diseño de la Interfaz:** Antes de comenzar la búsqueda del patrón de vibración, la usuaria diseñó una interfaz sencilla para evaluar la integración del efecto dentro del juego móvil.
- **Exploración Inicial:** Comenzó explorando el catálogo de vibraciones, pero notó que las etiquetas predeterminadas no eran del todo útiles para encontrar un patrón adecuado en este caso. Sin embargo, le gustó la posibilidad de modificarlas según su criterio.
- **Selección y Prueba del Patrón:** Tras probar varias opciones, selec-

cionó un patrón de vibración que consideró apropiado para la acción del juego. Observó que el gráfico incluido en la interfaz fue útil para visualizar la intensidad del efecto antes de probarlo en el dispositivo (ver Figura 6.10).

Figura 6.10: El cursor del usuario 2 señala el sonido seleccionado para esta tarea

Resultados de la Tarea 3: Proceso de Autenticación y Subida de Sonidos Personalizados

Desarrollo de la Prueba

- **Registro e Inicio de Sesión en FreeSound:** La usuaria inició sesión sin inconvenientes.
- **Exploración de la Plataforma:** Navegó por la plataforma y observó las opciones disponibles para la subida de sonidos personalizados.
- **Subida de un Sonido Personalizado:** Procedió a subir un sonido con etiquetas personalizadas, destacando la facilidad del proceso. El único error que cometió, fue no haber puesto 3 etiquetas pero lo corrigió en cuestión de segundos
- **Visualización del Sonido en el Catálogo:** Confirmó que el sonido aparecía en el catálogo, pero le pareció raro que aparezca al final de la lista.

6.2.3. Usuario 3

Resultados de la Tarea 1: Diseño de una Interacción de Confirmación de Compra en una Aplicación de Comercio Electrónico

Resultados: Durante la primera prueba de usabilidad, el participante, un diseñador UX con experiencia en Framer, exploró el plugin háptico para integrar una vibración en una interacción de juego. La prueba le llevo 10 minutos. A continuación, se detallan sus acciones:

Desarrollo de la Prueba

- **Creación interfaz:** El participante creo una interfaz de compra de un plan para un comercio (ver Figura 6.11).

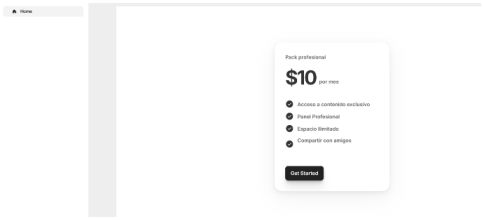

Figura 6.11: Interfaz realizada por el usuario 3

- **Apertura del Plugin:** El participante ingresó al plugin sin inconvenientes dentro de Framer. La interfaz del plugin fue identificada rápidamente (ver Figura 6.12).

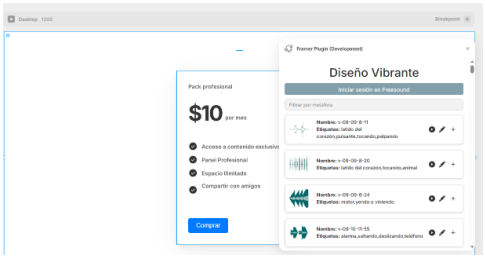

Figura 6.12: Plugin agregado exitosamente

- **Exploración del Catálogo de Sonidos:** Exploró las opciones predefinidas, reproduciendo varias opciones antes de seleccionar una que consideró adecuada para la interacción. El participante considera que la falta de una descripción detallada de cada vibración dificulta la selección adecuada (ver Figura 6.13).

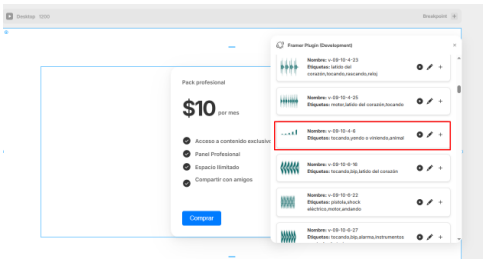

Figura 6.13: Selección del usuario 3

- **Selección y Aplicación del Efecto:** Tras elegir un sonido, lo agregó sin inconvenientes al diseño de la pantalla de confirmación de compra. Luego, adaptó el componente para que se ajustara a su diseño específico (ver Figura 6.14).

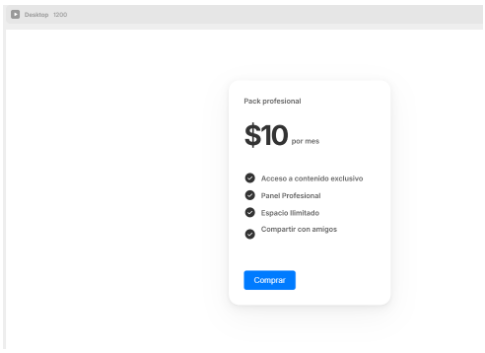

Figura 6.14: Resultado de la tarea 1 realizada por el usuario 3

- **Prueba en Dispositivo Móvil:** Publicó el proyecto en la web para probar la interacción en su teléfono móvil Android. Al evaluar la vibración, notó que el efecto funcionaba correctamente, aunque no le pareció correcta la exactitud de la vibración respecto al sonido que él había seleccionado (ver Figura 6.15).

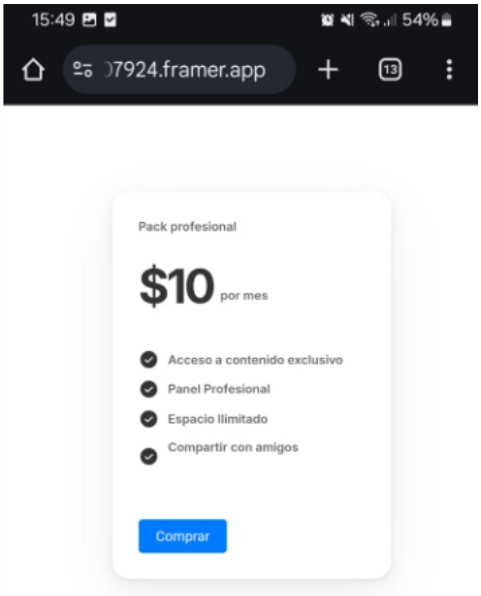

Figura 6.15: Resultado en el celular del usuario 3

Resultados de la Tarea 2: Búsqueda de un Patrón de Vibración Específico para una Acción en un Juego Móvil

Resultados: En esta segunda prueba, el participante diseñó una interacción para representar la obtención de un objeto importante en un juego móvil, utilizando el plugin háptico. La prueba le llevo aproximadamente 15 minutos. A continuación, se detallan sus acciones:

Desarrollo de la Prueba

- **Diseño de la Interfaz:** Antes de comenzar la búsqueda del patrón de vibración, el usuario diseñó una interfaz para estructurar adecuadamente la prueba (ver Figura 6.16).

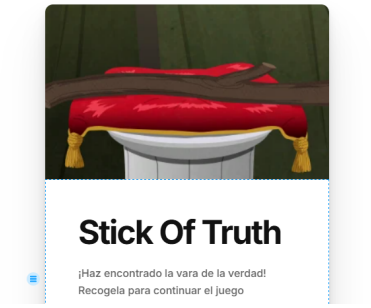

Figura 6.16: Interfaz realizada para la prueba

- **Exploración Inicial:** El usuario al instante de abrir el plugin utilizó la función de búsqueda.
- **Selección y Prueba del Patrón:** El usuario seleccionó el patrón de vibración y lo incluyó en su diseño como "Recolectar". Publicó la interacción para acceder a ella desde un dispositivo móvil Android y probar la vibración en tiempo real. Luego, se probó la interacción en y a diferencia de la prueba anterior esta si le pareció acorde aunque noto falta de intensidad en la vibración (ver Figura 6.17).

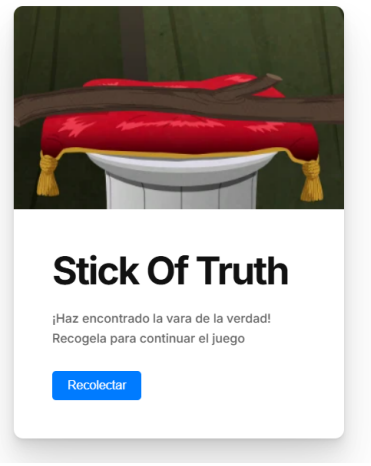

Figura 6.17: Botón agregado al diseño

Resultados de la Tarea 3: Proceso de Autenticación y Subida de Sonidos Personalizados

Resultados: En esta tercera prueba, el participante evaluó la experiencia de autenticación en FreeSound y la funcionalidad de subida de sonidos

personalizados dentro del plugin háptico. Tardo un total de 20 minutos. A continuación, se detallan sus acciones:

Desarrollo de la Prueba

- **Registro e Inicio de Sesión en FreeSound:** El usuario comenzó registrándose en FreeSound y luego inició sesión en la plataforma. Una vez hecho esto, no tuvo ni menciono problemas para obtener el token (ver Figura 6.18).

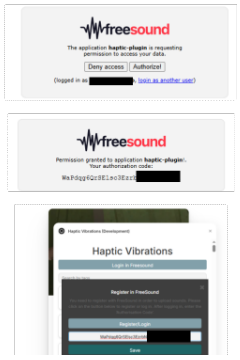

Figura 6.18: Registro y token del usuario 3

- **Búsqueda del sonido:** El usuario no contaba con algún sonido predefinido para subir, por lo que procedió a buscar un sonido acorde a sus necesidades para subirlo al plugin. Una vez lo encontró, procedió a subirlo al plugin (ver Figura 6.19).

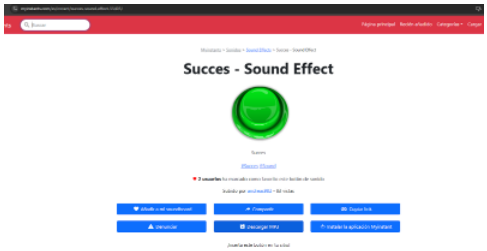

Figura 6.19: Obtención de un sonido

- **Subida de un Sonido Personalizado:** Procedió a intentar subir un sonido con los siguientes datos (ver Figura 6.20).:
  - Nombre: `"Success"`
  - Descripción: `"Sonido satisfactorio al obtener algo"`
  - Etiquetas: `"Satisfactorio, avance"`

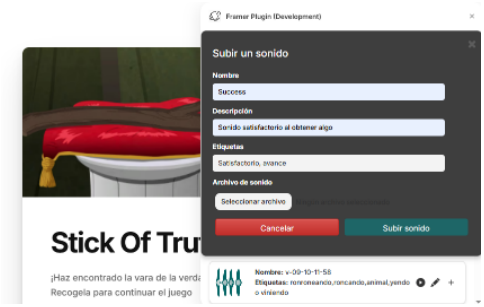

Figura 6.20: Datos del sonido a subir

Se observó que no pudo subirlo ya que al menos se necesitan 3 etiquetas (ver Figura 6.21).

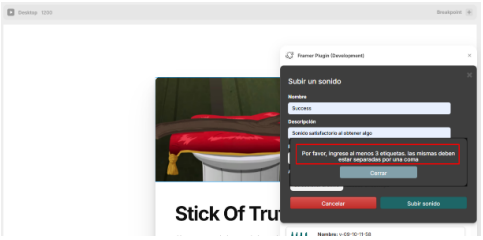

Figura 6.21: Error en el proceso por no agregar como minimo 3 etiquetas

Vuelve a intentarlo esta vez con 3 etiquetas, pero lanza error al subirlo, esto es porque el sonido ya se encuentra cargado en FreeSound (ver Figura 6.22).

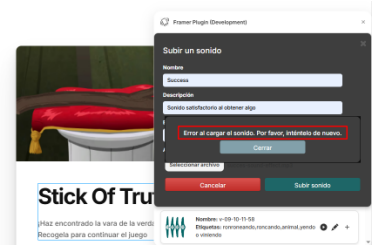

Figura 6.22: Error debido a la existencia en Freesound del sonido seleccionado

La subida se realizó exitosamente en el tercer intento, con un nuevo sonido (ver Figura 6.23).

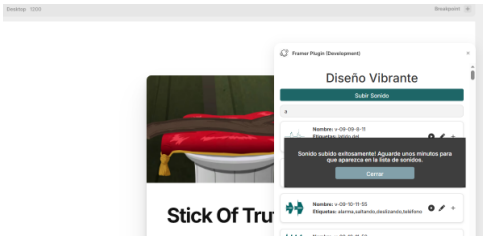

Figura 6.23: Éxito al subir el sonido nuevo

- **Visualización del Sonido en el Catálogo:** Tras agregar el sonido, observó que este aparecía en el catálogo (ver Figura 6.24).

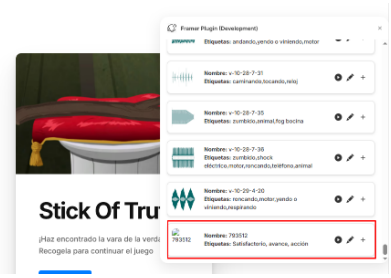

Figura 6.24: Sonido finalmente incluido en la biblioteca del plugin

6.3. Resultados de las Pruebas

Las pruebas realizadas con tres diseñadores permitieron identificar errores comunes y dificultades en el uso del *plugin-haptico*, así como recopilar sugerencias valiosas para mejorar su funcionalidad y experiencia de usuario. A continuación, se presentan los resultados generales organizados en dos categorías: errores/dificultades encontradas y mejoras sugeridas.

Errores y Dificultades Encontradas Durante las pruebas, se identificaron las siguientes dificultades comunes:

- **Claridad en la interfaz:** Algunos íconos y botones, como el `"+"` para agregar un sonido o el lápiz para editar etiquetas, no fueron percibidos como intuitivos por los usuarios. Esto generó confusión sobre sus funciones y ralentizó el flujo de trabajo.
- **Navegación en el catálogo de vibraciones:** La organización del catálogo no resultó completamente intuitiva. Los usuarios señalaron que encontrar patrones específicos era complicado debido a la falta de una categorización más clara y etiquetas relevantes.

- **Falta de retroalimentación visual:** En acciones clave, como la subida de sonidos personalizados o la aplicación de patrones a componentes, los usuarios notaron la ausencia de confirmaciones visuales que indicaran que la acción se había completado con éxito.
- **Funcionalidad del filtro:** Aunque el filtro funcionaba correctamente, algunos usuarios tuvieron problemas para comprender cómo utilizarlo eficazmente. Esto se debió a la falta de ejemplos o placeholders informativos que explicaran cómo buscar por etiquetas.
- **Identificación de vibraciones:** La identificación de patrones mediante códigos numéricos en lugar de nombres descriptivos dificultó la selección rápida y precisa de vibraciones.
- **Edición limitada del componente generado:** Los usuarios reportaron que no podían modificar atributos visuales (como el color) del botón generado automáticamente al aplicar un patrón. Esto limitó su capacidad para personalizar elementos según sus diseños.
- **Formulario de subida poco claro:** El formulario para subir sonidos personalizados no especificaba claramente qué eran los `"tags"` ni cuántos debían incluirse como mínimo. Esto generó confusión durante el proceso.
- **Vibración con baja intensidad:** Algunos usuarios percibieron que las vibraciones eran menos potentes de lo esperado, lo que podría limitar su efectividad en ciertos contextos.
- **Visualización del nombre asignado:** Los sonidos subidos no mostraban correctamente el nombre asignado por el usuario durante el proceso de subida, lo que generó inconsistencias entre lo esperado y lo mostrado.
- **Error al subir sonidos existentes en FreeSound:** Cuando un usuario intentó subir un sonido ya existente en la base de datos de FreeSound (subido previamente por otro usuario), el plugin mostró un mensaje genérico indicando que ocurrió un error, sin especificar la causa. Esto generó confusión sobre cómo proceder o qué alternativa tomar.

Mejoras Sugeridas Con base en las observaciones y comentarios recopilados durante las pruebas, se proponen las siguientes mejoras para optimizar el plugin:

- **Categorización del catálogo:** Implementar una estructura organizada que agrupe los patrones por categorías específicas, tales como

`"notificaciones"`, `"confirmaciones"` o `"errores"`. Esto facilitaría la navegación y selección.

- **Etiquetas descriptivas:** Reemplazar los identificadores numéricos por nombres descriptivos que reflejen mejor las características o sensaciones asociadas a cada vibración. Además, enriquecer las etiquetas predeterminadas para hacerlas más relevantes al contexto del diseño.
- **Placeholder informativo en el filtro:** Incluir un texto placeholder en el campo de búsqueda que explique claramente cómo buscar por etiquetas o palabras clave. También se podrían añadir ejemplos prácticos para guiar al usuario.
- **Retroalimentación visual:** Incorporar mensajes emergentes o animaciones que confirmen acciones importantes realizadas por el usuario, como la subida exitosa de un sonido personalizado o la aplicación correcta de un patrón a un componente.
- **Historial de selección:** Añadir una sección con los patrones recientemente utilizados para permitir un acceso rápido sin necesidad de buscarlos nuevamente.
- **Claridad en los íconos:** Rediseñar los íconos y botones clave para que sean más intuitivos. Por ejemplo, reemplazar el lápiz por un texto explícito como `"Editar etiquetas"` o usar un ícono más representativo para agregar sonidos.
- **Edición del componente generado:** Permitir a los diseñadores modificar atributos visuales (como color, tamaño o forma) del botón generado automáticamente al aplicar un patrón.
- **Formulario más claro para subida de sonidos personalizados:** Incluir instrucciones detalladas sobre qué son los `"tags"`, cuántos deben añadirse como mínimo y ejemplos prácticos para facilitar su comprensión.
- **Ajuste de intensidad de vibración:** Incorporar opciones para ajustar la intensidad de las vibraciones generadas, permitiendo una mayor personalización según las necesidades del diseño.
- **Ejemplos prácticos y descripciones:** Incluir descripciones breves o ejemplos prácticos para cada patrón dentro del catálogo, ayudando a los diseñadores a comprender mejor su uso potencial.
- **Mensaje claro al subir sonidos ya existentes en FreeSound:** Mejorar los mensajes de error relacionados con la subida de sonidos personalizados. Por ejemplo, si el sonido ya existe en FreeSound pero

no fue subido por el usuario actual, mostrar un mensaje explicativo que indique esta situación y sugiera buscar otro sonido diferente.

6.3.1. Evaluación de Usabilidad con el System Usability Scale (SUS)

Para evaluar la usabilidad del *plugin-haptico*, se aplicó el System Usability Scale (SUS), un cuestionario ampliamente utilizado y validado para medir la percepción subjetiva de la usabilidad de un sistema [31]. El SUS consta de diez preguntas con opciones de respuesta en una escala Likert de 5 puntos, que van desde "Totalmente en desacuerdo"hasta "Totalmente de acuerdo". Las preguntas están diseñadas para capturar la satisfacción general del usuario, la facilidad de uso, la complejidad percibida y la confianza al utilizar el sistema.

Una vez completado el cuestionario por los participantes, las respuestas se convierten en una escala de 0 a 100, donde una puntuación más alta indica una mejor usabilidad. La puntuación SUS se calcula siguiendo una fórmula específica que invierte las puntuaciones de algunas preguntas y luego suma todos los valores multiplicando el resultado por 2.5 [32].

Los resultados obtenidos fueron los siguientes (ver Figura 6.25).:

- **Participante 1:** 92.5 (A)
- **Participante 2:** 75 (B)
- **Participante 3:** 82.5 (A)
- **Puntuación promedio:** 83.3 (A)

Promedio PUNTUACIÓN	Promedio NOTA
83,3	A

TEACUP LAB

	PREGUNTA 1	PREGUNTA 2	PREGUNTA 3	PREGUNTA 4	PREGUNTA 5	PREGUNTA 6	PREGUNTA 7	PREGUNTA 8	PREGUNTA 9	PREGUNTA 10	SUS Puntuación	SUS Nota
	Me gustaría usar este plugin con frecuencia.	Encontré el plugin innecesariamente complejo.	Creo que el plugin es fácil de usar.	Creo que necesitaría la ayuda de un experto para usar el plugin.	Encontré que las diversas funciones del sistema estaban bien integradas	Pensé que había demasiada inconsistencia en el plugin.	Creo que la mayoría de las personas aprenderían a usar el plugin rápidamente.	Encontré el plugin muy incomodo de usar.	Me sentí muy seguro(a) usando el plugin.	Necesité aprender muchas cosas antes de poder empezar a usar el plugin.		
Participante 1	4	1	5	1	5	1	5	2	4	1	92,5	A
Participante 2	3	2	4	2	3	1	4	2	4	1	75	B
Participante 3	3	1	4	1	4	2	4	1	4	1	82,5	A

Figura 6.25: Gráfico con resultados SUS

Estos resultados indican que, en general, los usuarios perciben el *plugin-haptico* como un sistema fácil de usar. La puntuación promedio de 83.3 se sitúa en el rango `"Excelente usabilidad"` según las métricas de interpretación del SUS, lo que sugiere que el plugin cumple con los estándares de usabilidad y es bien recibido por los diseñadores. Sin embargo, es importante tener en cuenta que la puntuación del Participante 2 fue ligeramente inferior

(75), lo que sugiere que podría haber áreas de mejora para abordar las necesidades y expectativas de todos los usuarios. Ademas la prueba de testeo arrojó una muy buena devolución de los usuarios dado que las métricas en las respuestas se vieron reflejadas en la buena nota final, y demuestran el entendimiento y la rápida internalización del mismo.

Estos resultados reflejan áreas clave donde el plugin puede ser optimizado para mejorar su usabilidad y satisfacer mejor las necesidades de los diseñadores. Las sugerencias recopiladas serán fundamentales para futuras iteraciones del desarrollo del plugin.

Capítulo 7

Conclusiones

En esta tesina, se ha desarrollado una plataforma integrada para el diseño de experiencias de usuario (UX) que incorpora interacciones vibrotáctiles, materializada en un plugin inicialmente creado para Figma y posteriormente migrado a Framer. El objetivo principal fue proporcionar a los diseñadores una herramienta intuitiva y accesible que les permitiera integrar y experimentar con vibraciones hápticas sin requerir conocimientos avanzados de programación o hardware especializado. A lo largo del proyecto, se realizaron diversas pruebas de funcionalidad y usabilidad que validaron la utilidad de la herramienta y ofrecieron retroalimentación valiosa para su mejora. Esta sección presenta un análisis de los logros alcanzados, el impacto en el diseño UX, las lecciones aprendidas y las propuestas para trabajo futuro, integrando los resultados obtenidos.

El desarrollo del plugin comenzó en Figma, aprovechando su popularidad y capacidades colaborativas. Sin embargo, las limitaciones técnicas de Figma, como la imposibilidad de simular vibraciones en tiempo real y los pasos manuales que afectaban la usabilidad, motivaron la migración a Framer. Este cambio permitió superar dichas restricciones gracias a la capacidad de Framer para ejecutar código JavaScript y conectar con hardware móvil, posibilitando que los diseñadores previsualizaran vibraciones en dispositivos reales. Las pruebas finales demostraron una mejora significativa en la experiencia de diseño, destacando la flexibilidad de Framer para este propósito.

Un logro clave fue la implementación de un catálogo de vibraciones predefinidas basado en la biblioteca VibViz, con patrones como `"pulso corto"` y `"latido"`. En las pruebas de usabilidad, los participantes señalaron que este catálogo aceleraba el proceso de selección de efectos hápticos, aunque sugirieron una mejor categorización por tipo de emoción o uso. Además, se incorporó la posibilidad de personalizar efectos mediante la edición de etiquetas (*tags*) y la subida de sonidos propios, permitiendo a los diseñadores asignar nombres y etiquetas para facilitar su búsqueda en el listado. La conversión de archivos de audio en patrones de vibración fue implementada, pero las pruebas mostraron inconsistencias: en algunos sonidos la traducción a vibración era más efectiva que en otros, lo que los participantes señalaron como un área a mejorar. El sistema de almacenamiento persistente, basado en *local storage*, aseguró que las configuraciones se guardaran y recuperaran correctamente, optimizando el flujo de trabajo.

La efectividad del plugin se validó mediante pruebas de usabilidad con tres diseñadores, quienes completaron tareas relacionadas con el diseño de interacciones vibrotáctiles en aplicaciones de comercio electrónico y juegos

móviles. Los resultados indicaron que los usuarios podían emplear la herramienta con éxito, destacando su facilidad de uso y su integración con el flujo de diseño. La evaluación con el *System Usability Scale* (SUS) arrojó una puntuación promedio de 83.3, indicando una excelente usabilidad. Esto demuestra que la herramienta no solo es viable, sino que también reduce la barrera técnica para diseñadores sin experiencia en hápticas, enriqueciendo las interfaces multisensoriales.

El proceso de desarrollo dejó lecciones valiosas. La experiencia con Figma subrayó la importancia de evaluar las capacidades técnicas de una plataforma desde el inicio, mientras que Framer se consolidó como una opción más adecuada. Las pruebas de usabilidad resaltaron la necesidad de una interfaz clara y un catálogo bien organizado, lo que llevó a ajustes en su presentación visual. Además, la retroalimentación de usuarios reales identificó áreas de mejora que no se habrían detectado de otro modo, reforzando el valor de las iteraciones con pruebas.

De cara al futuro, se plantea optimizar la conversión de audio a vibraciones para mejorar su precisión, ampliar el catálogo de patrones predefinidos y perfeccionar las herramientas de personalización con una categorización más intuitiva. También se podría explorar una versión independiente del plugin o su integración con otras plataformas de diseño, consolidando su utilidad en el diseño UX multisensorial.

En resumen, este trabajo ha logrado cerrar la brecha entre el diseño UX y las interacciones hápticas con una herramienta práctica y accesible. Los resultados obtenidos sientan una base sólida para futuras investigaciones en este campo emergente, con un impacto positivo demostrado en la eficiencia y creatividad de los diseñadores.

Bibliografía

[1] Android Developers: *Haptics Design Guidelines*. `https://developer.android.com/develop/ui/views/haptics`, 2024. [Último acceso: 26 de noviembre de 2024].

[2] Apple Developers: *Core Haptics Documentation*. `https://developer.apple.com/documentation/corehaptics`, 2024. [Último acceso: 26 de noviembre de 2024].

[3] Seifi, Hasti, Kailun Zhang y Karon E MacLean: *VibViz: Organizing, visualizing and navigating vibration libraries*. En *2015 IEEE World Haptics Conference (WHC)*, páginas 254–259. IEEE, 2015.

[4] Technologies, AAC: *RichTap®: High-Fidelity Haptic Solutions for Mobile Devices*, 2022. `https://www.richtap-haptics.com/`, [Último acceso: 26 de febrero de 2025].

[5] Figma Inc.: *Figma - The Collaborative Interface Design Tool*, 2023. `https://www.figma.com/`, [Último acceso: 23 de febrero de 2025].

[6] Framer: *Framer: Interactive Design Platform*. `https://www.framer.com/`, 2024. Último acceso: 19 de noviembre de 2024.

[7] Interaction Design Foundation - IxDF. (2016, June 2).: *What is Interaction Design (IxD)?*., 2016. `https://www.interaction-design.org/literature/topics/interaction-design`, [Último acceso: 23 de febrero de 2025].

[8] Kolko, Jon: *Thoughts on interaction design*. Morgan Kaufmann, 2010.

[9] Jones, Lynette A.: *Haptics*. The MIT Press, Cambridge, MA, 2018, ISBN 9780262038340.

[10] Seifi, Hasti, Mattew Chun y Karon E Maclean: *Toward affective handles for tuning vibrations*. ACM Transactions on Applied Perception (TAP), 15(3):1–23, 2018.

[11] Israr, Ali, Siyan Zhao, Kaitlyn Schwalje, Roberta Klatzky y Jill Lehman: *Feel effects: enriching storytelling with haptic feedback*. ACM Transactions on Applied Perception (TAP), 11(3):1–17, 2014.

[12] Schneider, Oliver, Karon MacLean, Colin Swindells y Kellogg Booth: *Haptic experience design: What hapticians do and where they need help*. International Journal of Human-Computer Studies, 107:5–21, 2017.

[13] Kim, Erin y Oliver Schneider: *Defining haptic experience: foundations for understanding, communicating, and evaluating HX*. En *Proceedings of the 2020 CHI conference on human factors in computing systems*, páginas 1–13, 2020.

[14] Sketch: *Sketch - The Digital Design Toolkit*, 2023. `https://www.sketch.com`, [Último acceso: 23 de febrero de 2025].

[15] Hoggan, Eve y Stephen Brewster: *New parameters for tacton design*. En *CHI '07 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '07, página 2417–2422, New York, NY, USA, 2007. Association for Computing Machinery, ISBN 9781595936424. `https://doi.org/10.1145/1240866.1241017`.

[16] Schneider, Oliver S. y Karon E. MacLean: *Studying design process and example use with Macaron, a web-based vibrotactile effect editor*. En *2016 IEEE Haptics Symposium (HAPTICS)*, páginas 52–58, 2016.

[17] Ferguson, Jamie, John Williamson y Stephen Brewster: *Evaluating mapping designs for conveying data through tactons*. En *Proceedings of the 10th Nordic Conference on Human-Computer Interaction*, NordiCHI '18, página 215–223, New York, NY, USA, 2018. Association for Computing Machinery, ISBN 9781450364379. `https://doi.org/10.1145/3240167.3240175`.

[18] Ternes, David y Karon E. MacLean: *Designing Large Sets of Haptic Icons with Rhythm*. En Ferre, Manuel (editor): *Haptics: Perception, Devices and Scenarios*, páginas 199–208, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg, ISBN 978-3-540-69057-3.

[19] Interhaptics: *Interhaptics - Haptic Studio and SDK for Haptic Feedback Design*, 2021. `https://www.interhaptics.com/`, [Último acceso: 26 de marzo de 2025].

[20] Figma Inc.: *Figma Plugin API Documentation*, 2023. `https://www.figma.com/plugin-docs/`, [Último acceso: 5 de noviembre de 2024].

[21] Figma Inc.: *Figma Plugin API Creating a User Interface*, 2023. `https://www.figma.com/plugin-docs/creating-ui/`, [Último acceso: 5 de noviembre de 2024].

[22] Figma Inc.: *Figma Plugin API Documentation: Accessing the Document*, 2023. `https://www.figma.com/plugin-docs/accessing-document/`, [Último acceso: 5 de noviembre de 2024].

[23] Developers, Framer: *Introduction to Components*. `https://www.framer.com/developers/components/introduction`, 2024. [Último acceso: 19 de noviembre de 2024].

[24] FreeSound: *FreeSound API Documentation*. https://freesound.org/docs/api/overview.html. [Último acceso: 8 de noviembre de 2024].

[25] FreeSound: *FreeSound OAuth2 Authentication Guide*. https://freesound.org/docs/api/authentication.html. [Último acceso: 8 de noviembre de 2024].

[26] FreeSound: *FreeSound API - Upload Sound Endpoint*. https://freesound.org/docs/api/resources_sounds.html#upload-sound. [Último acceso: 8 de noviembre de 2024].

[27] FreeSound: *FreeSound API - Pending Uploads*. https://freesound.org/docs/api/resources_sounds.html#pending-uploads. [Último acceso: 8 de noviembre de 2024].

[28] FreeSound: *FreeSound - Collaborative Database for Audio Files*. https://freesound.org/help/about/. [Último acceso: 8 de noviembre de 2024].

[29] Figma Inc.: *Figma Plugin API Documentation*, 2023. https://www.figma.com/plugin-docs/plugin-quickstart-guide/, [Último acceso: 5 de noviembre de2024].

[30] Wu, Rob: *CORS Anywhere*, 2025. https://github.com/Rob--W/cors-anywhere, [Último acceso: 26 de marzo de 2025].

[31] Brooke, John *y cols.*: *SUS-A quick and dirty usability scale*. Usability evaluation in industry, 189(194):4–7, 1996.

[32] Bangor, Aaron, Philip T Kortum y James T Miller: *An empirical evaluation of the system usability scale*. Intl. Journal of Human–Computer Interaction, 24(6):574–594, 2008.