Diseño y desarrollo de una página web para la venta de prompts utilizados en inteligencias artificiales

Memoria

Miquel Plana Ballber

Máster Universitario en Desarrollo de sitios y aplicaciones web

Nombre Tutor/a de TF
Bruno Francisco Orcha Garcia

Profesor/a responsable de la asignatura
César Pablo Córcoles Briongos

Fecha Entrega
16-01-2023

Universitat Oberta de Catalunya

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño y desarrollo de una página web para la venta de prompts utilizados en inteligencias artificiales.*
Nombre del autor:	*Miquel Plana Ballber*
Nombre del consultor/a:	*Bruno Francisco Orcha*
Nombre del PRA:	*César Pablo Córcoles*
Fecha de entrega (mm/aaaa):	*01/2023*
Titulación o programa:	Máster Universitario en Desarrollo de sitios y aplicaciones web.
Área del Trabajo Final:	*Desarrollo web*
Idioma del trabajo:	*Castellano*
Palabras clave	*Prompt, Marketplace, web*

Resumen del Trabajo

En este proyecto se detalla el desarrollo de una página web dedicada a la compraventa de “prompts”, las frases de lenguaje natural con las que las inteligencias artificiales pueden generar texto e imagen. La página sirve para las principales inteligencias como DALL-E, Midjourney, Stable Diffusion o GPT3.

Abstract

This project details the development of a website for buying and selling prompts, the natural languaje sentences artificial intelliegences use to generate text and image. This page will be focused on the main AI’s such as DALL-E, Midjourney, Stable Diffussion or GPT3.

Índice

Lista de figuras

1. Introducción

En este apartado vamos a introducir el proyecto, justificarlo y planificarlo.

1.1. Contexto y justificación del Trabajo

En 2017 salió un artículo académico llamado “Attention is all you need”, donde se presentaba una manera más optimizada de leer el lenguaje natural para las inteligencias artificiales. Esto dio lugar a una revolución en este sector, ya que, por primera vez, las IAs entrenadas con suficiente texto podían entender con una precisión increíble lo que nosotros con nuestro lenguaje les queríamos decir.

Gracias a esta nueva metodología, surgieron inteligencias como GPT de Open-AI[1], las cuales podían interpretar nuestro texto y darnos respuestas adecuadas. Eso sí, mucho más precisas que cualquier cosa que habíamos visto antes.

Gracias a estos modelos de comprensión y generación de texto tan avanzados, en enero de 2021, Open-AI lanzó al mercado la fase beta de DALL·E, una inteligencia artificial que permitía, a través del lenguaje natural, generar imágenes de alta calidad como si hubiesen sido creadas por humanos. Un año más tarde, en abril de 2022, lanzó la versión DALL·E 2, entrenada con muchas más imágenes y dando unos resultados de mucha más calidad.

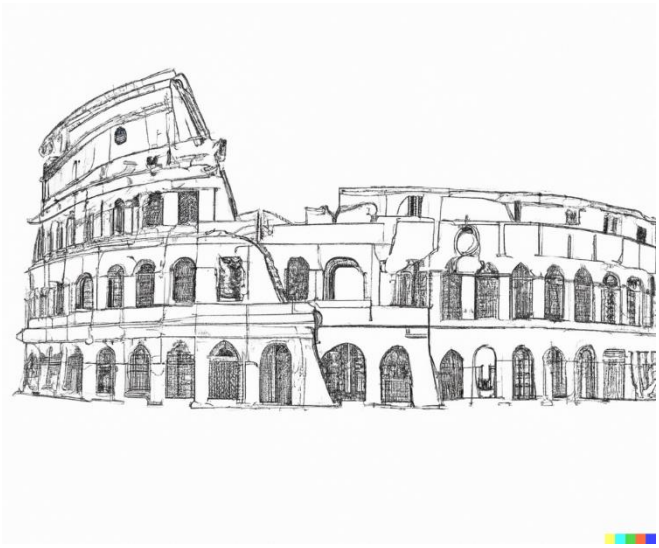


Figura 1. Imagen generada por el autor con DALL-E

Esta última versión tuvo mucha repercusión mediática, no solo por la facilidad con la que se podía generar una imagen de cualquier tipo, incluyendo imágenes que parecían hechas por cámaras fotográficas, sino también por las cuestiones éticas que surgían: ¿Es ético que la inteligencia se nutra de imágenes hechas por artistas vivos? ¿Qué pasará con el trabajo de los artistas? ¿Es segura esta tecnología? Independientemente de estas cuestiones, a partir de ese momento las inteligencias artificiales de generación de imágenes ganaron mucha

[1] https://openai.com/

popularidad. Las más conocidas son Stable Diffusion (de Hugging Face[2]), Midjourney[3] y DALL-E.

A partir de este punto, donde las inteligencias dan resultados de una calidad alta, se crea una nueva figura o "puesto de trabajo" llamada **prompt engineer**. El prompt engineer es una persona que conoce qué formas del lenguaje natural funcionan mejor en estos generadores de texto e imagen, es decir, saben cómo obtener realmente lo que quieren. Y es que, aunque la generación sea de alta calidad, si no sabes describírselo a la IA de una manera que lo entienda, los resultados que vas a obtener serán mediocres.

Si, por ejemplo, una empresa o un autónomo requieren del diseño de un logo, tal vez ya no acudirá al artista, sino que contratarán a un prompt engineer que, con muchos menos recursos y tiempo, obtendrá un resultado similar y de calidad. Pero claro, también podríamos querer utilizar un generador sin necesidad de esta figura intermedia, y además obtener imágenes de calidad. Por ejemplo, en el caso de una empresa de videojuegos pequeña, podrían querer hacer muchas pruebas para ver que aspecto se les puede dar a los ítems dentro del juego (armaduras, espadas…), pero haciendo pequeños cambios en cada iteración. ¿Es necesario pagar al prompt engineer cada vez? Por suerte, no.

Existe otra manera de obtener imágenes o textos de calidad de estos generadores, y es comprando prompts (frases) que han sido creados y verificados por otros usuarios, y que sabemos que dan buenos resultados. De esta manera solo tendríamos que introducir el prompt en el generador, e ir haciendo retoques a medida que iteramos sobre el producto. Así nace el concepto de **prompt marketplace**, un mercado online donde se pueden vender y comprar prompts. Los pioneros en esta tendencia son Promptbase[4].

Este trabajo está basado en esa misma idea, y se busca crear un marketplace de prompts con un enfoque más realista, en el que los prompts sirvan para emprendedores, artistas, empresas y proyectos del mundo real.

1.2. Objetivos del Trabajo

El objetivo de este proyecto es obtener una página web funcional y estética, preparada para vender prompts de cualquier usuario, validarlos internamente y comprarlos. No se espera tener usuarios activos, sino un producto que pueda ser utilizado.

Si entran en los objetivos de este trabajo:

- Diseñar la primera versión página web.
- Desarrollar un Minimum Viable Product (MVP) con la tecnología escogida.
- Que la página web sea accesible desde cualquier navegador a través de una URL.

[2] https://huggingface.co/spaces/stabilityai/stable-diffusion
[3] https://www.midjourney.com/home/
[4] https://promptbase.com/marketplace

- Que se pueda realizar cualquier proceso de compra y venta, registro y validación de prompts dentro de la página.
- Un pequeño análisis económico y del estado del mercado actual, sin entrar en detalle.

No entran en los objetivos:

- Estudio de mercado extenso y análisis de viabilidad del producto.
- Validar con usuarios reales el diseño ni el resultado final.
- Desarrollo de un back-end propio, se utilizará un Back-end as a Service (BaaS) en su lugar.
- Promoción o publicidad de cualquier tipo.
- Definición de métricas que indican el éxito o fracaso de la página.

Como se puede ver, el proyecto está muy centrado en la parte de desarrollo, que es la aptitud que se ha practicado durante el máster.

La página deberá cumplir, además, con todo lo establecido en el documento de requerimientos PEC1_rec_Plana_Ballber_Miquel entregado en la PEC-1.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

Dimensión sostenibilidad

A nivel de sostenibilidad, tenemos que tener en cuenta el factor el uso de servidores. Los datos en la nube gastan cada vez más electricidad y agua (para enfriar los servidores), por lo que es importante que durante todo el proyecto se mida el impacto que tiene seleccionar un proveedor u otro, prioritariamente escogiendo uno neutro en emisiones de carbono o que tenga como objetivo serlo en los próximos años.

Dimensión diversidad, género y derechos humanos

Como ya se ha mencionado en el apartado 1.1, la aparición de estas inteligencias artificiales pone en peligro trabajos creativos hasta ahora realizados únicamente por seres humanos. Es por ello que la creación del marketplace podría tener un impacto negativo en esta comunidad de artistas, lo cuál va a ser inevitable, pero controlable.

Las regulaciones gubernamentales de todo el mundo van años atrás en lo que a las inteligencias artificiales se refiere (e incluso Internet), siendo sus legislaciones muy laxas y retrasadas respecto al progreso real de estas tecnologías. Esto propicia que existan áreas grises donde se pueda crear contenido peligroso, ofensivo o similar, y solo queda en manos de las empresas que el uso de estas tecnologías se haga respetando unas leyes mínimas.

Con todo esto en mente, se tomarán diversas acciones en este trabajo para evitar las consecuencias negativas e intentar encauzar este movimiento como algo positivo:

- No se permitirá en la web la venta de prompts que generen imágenes a partir del trabajo de artistas vivos. Si se permitirá, por ejemplo, el de artistas ya fallecidos como Van Gogh o Picasso.
- No se permitirá contenido ofensivo, peligros, sexual o de apología al terror.
- No se permitirá la copia de prompts de otras páginas webs o creadores.
- No se permitirá la subida de prompts que utilicen la identidad de otras personas.

Esto implicará una revisión de cada uno de los prompts que se vayan a vender, como ya se ha anticipado en el apartado anterior.

1.4. Enfoque y método seguido

Para este proyecto solo existe una estrategia posible, y es desarrollar esta página web desde cero. No es posible basarse cien por cien en PromptBase (el competidor), puesto que podría incurrir en problemas de plagio, y además tiene sentido diferenciarse de la competencia con un producto un tanto distinto.

Para alcanzar el éxito se seguirá la metodología Lean, y lo que se construirá es un MVP, que servirá para rápidamente validar con usuarios reales si la página web es entendible, funcional y útil. Esta validación se haría después de finalizar el proyecto, por lo que no entrará en el mismo.

El porqué de seguir la metodología Lean es clara: En el software se tiene la capacidad de iterar muy rápidamente sobre un producto, pivotando si es necesario y corrigiendo errores en cada iteración. Es el único producto en el mundo donde se puede tomar este enfoque tan rápido y eficaz, y prueba de ello son todas las empresas de software que utilizan esta metodología o sus derivados.

Al seguir la metodología Lean, lo lógico es utilizar un framework de desarrollo que permita trabajar lo más rápido posible (por ejemplo, con componentes reutilizables). La selección de este framework se realiza más adelante en el proyecto.

1.5. Planificación del Trabajo

El trabajo se planifica acorde con el documento de requerimientos del proyecto entregado, y se hace uso del software Gantt Project. En la planificación también se ha tenido en cuenta el propio proceso de planificar y definir el proyecto.

Antes de mostrar la planificación, se definen las tareas a realizar, así como su interdependencia:

Id	Tarea	Descripción	Depende de
1	Definición del proyecto	Todos los puntos definidos en esta memoria sin contar la planificación.	-
2	Planificación del proyecto	Planificar las diferentes tareas del proyecto y sus tiempos.	1
3	Diseño de la página web.	Diseñar una primera versión de la página web para tenerla como referencia.	2
4	Definición de las tecnologías a utilizar	Se definirá el framework a utilizar, el lenguaje, el servicio de back-end y cualquier otra tecnología relevante.	2
5	Setup inicial del proyecto	Inicialización del proyecto en local, así como en Github, y realizar las instalaciones necesarias y crear los archivos correspondientes a páginas y componentes.	4
6	Header y Footer	Desarrollo de los componentes reutilizables de header y footer que irán en todas las páginas.	3, 5
7	Página principal (front-end)	Desarrollo del front-end de la página principal, incluidos todos los subcomponentes utilizados.	3, 5
8	Página de cuenta (front-end)	Desarrollo del front-end de la página de cuenta, tanto con usuario autenticado como sin autenticar, incluyendo subcomponentes utilizados.	3, 5
9.1	Página de venta (front-end)	Desarrollo del front-end de la página de venta, incluidos todos los subcomponentes utilizados.	3, 5
9.2	Página de compra (front-end)	Desarrollo del front-end de la página de compra, incluidos todos los subcomponentes utilizados.	3,5
10	Página de validaciones (front-end)	Desarrollo del front-end de la página de validaciones para el/los administradores de la página.	3, 5
11	Página de contacto (front-end)	Desarrollo del front-end de la página de contacto.	3, 5
12	Estructura de la base de datos	Definir la estructura de la base de datos, y teniendo en cuenta del servicio de back-end utilizado (SQL o NoSQL).	6, 7, 8, 9.1, 9.2, 10, 11

13	Setup del servicio de back-end	Registro, inicialización y configuración del servicio de back-end, tanto en su portal como en nuestro proyecto.	4, 5
14	Desarrollo de la autenticación	Conexión con el back-end para autenticar usuarios.	12, 13
15	Desarrollo de los pagos	Conexión con la API escogida de pagos, tanto para ventas como para compras.	12, 13
16.1	Subida de datos al back-end	Desarrollar las funcionalidades que sirven para enviar datos al back-end y conectarlas con los procesos de front-end.	12, 13
16.2	Mostrar datos de back-end en front-end.	Desarrollar las funcionalidades que sirven para poblar todo el front-end	12, 13
17	Envío del formulario de contacto	Conectar el formulario de contacto con el sistema que permitirá enviar al administrador de la página el correo.	11
18	Categorías	Definir que categorías de prompts.existen.	-
19	Términos y condiciones y FAQ	Crear los términos y condiciones y enlazarlos en el front-end, así como los FAQ.	7
20	Documentación	Documentar todo el desarrollo en la memoria. Se hace durante todo el proyecto.	-
20	Presentación	Presentación en formato video del proyecto según las especificaciones de la UOC.	Todo

Como se puede observar, el desarrollo toma un enfoque front-end first, en el que primero se desarrollará toda la parte visual, para luego conectarla con el back-end. Esto tiene diversos motivos de ser:

- Es más fácil mostrar el progreso, especialmente en este proyecto que va atado a unas PEC entregables con fecha de fin.
- Es mucho más fácil iterar si algo no acaba de encajar o se quiere cambiar.
- El diseño de back-end se hace más fácil una vez terminado el front-end.
- A los usuarios finales no les importa el back-end, sino el front-end y su experiencia de usuario.
-

Además, las tareas ya han sido ordenadas según la prioridad. El criterio que se ha seguido es el siguiente:

1. Definir, planificar y diseñar siempre irán antes de las tareas que se nutren de ellas. Primera se hace una definición del front-end en el diseño, para

poder desarrollarlo bien. Después se define la estructura de base de datos, para
2. Antes de iniciar los desarrollos, se debe hacer un setup de las herramientas que vamos a utilizar.
3. Se ha priorizado también por importancia. La parte más relevante de la app es la página principal, la cuenta para hacer compra ventas y el sistema de compras, mientras que el formulario de contacto, los términos y condiciones o las categorías no son prioritarios.

Esta planificación podría cambiar durante el desarrollo del proyecto, puesto que está sujeta a desarrollos no previstos o cambios en el diseño inicial. Durante todo el proceso de desarrollo estará presente la documentación.

Se asume un parón de 9 días en las fiestas de Navidad con motivo de la ausencia del autor.

En el siguiente Gantt separado por PEC’s se detalla el timeline del proyecto:

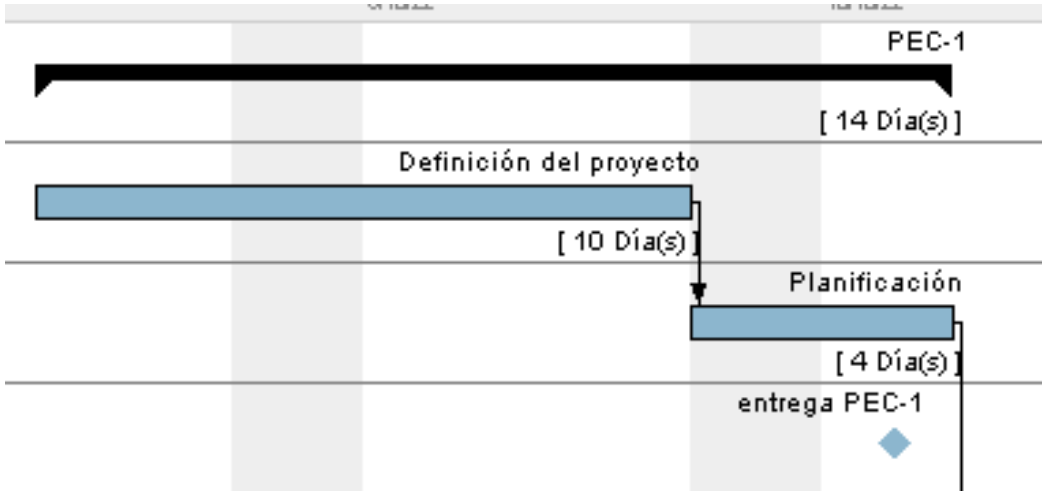


Figura 2. Planificación de la línea temporal de la PEC-1 (28 de septiembre - 11 de octubre)

En esta fase del proyecto simplemente definimos el proyecto y planificamos, lo que hay plasmado en el apartado 1 a fecha de entrega de esta memoria.

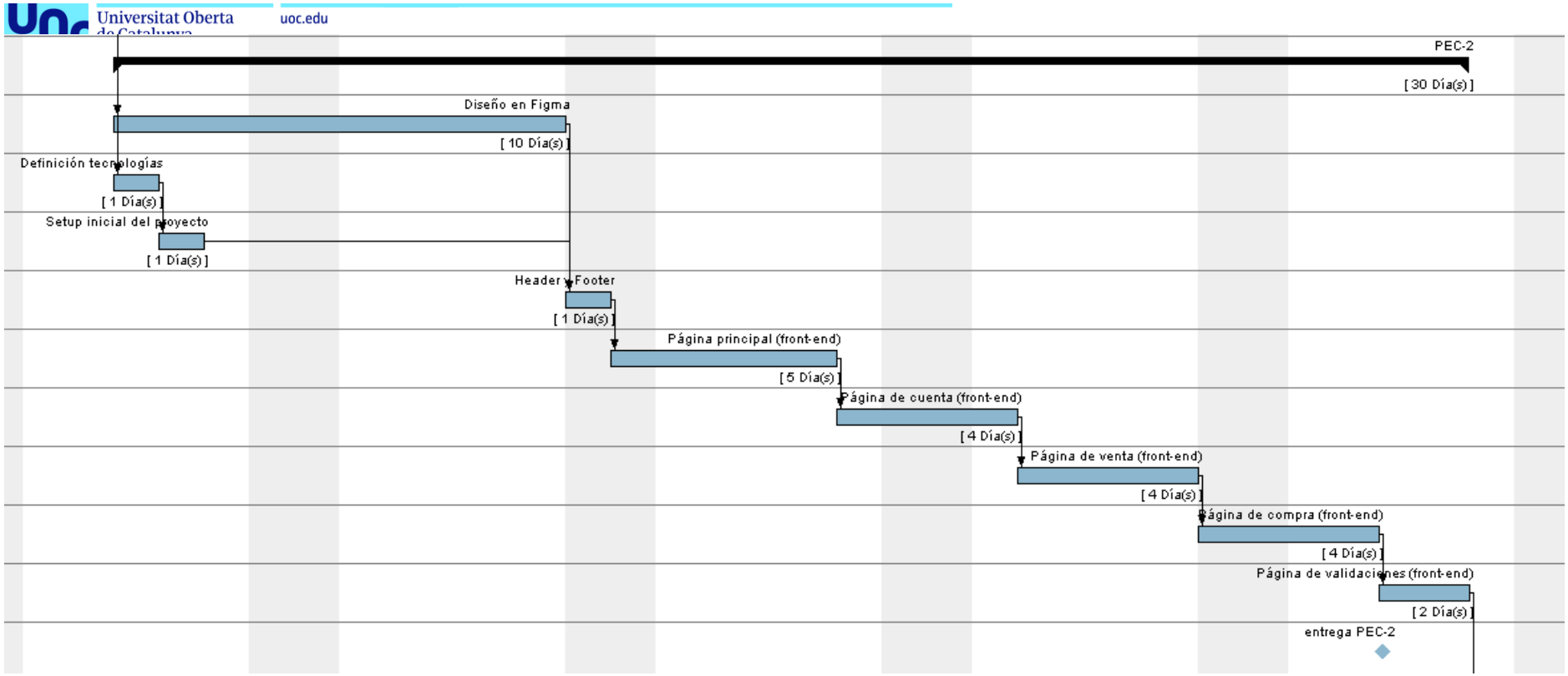


Figura 3. Planificación de la línea temporal de la PEC-2 (12 de octubre - 09 de noviembre)

En este periodo de tiempo se trabajará principalmente en la UX/UI con el diseño en Figma y el desarrollo del front-end. Finalizar esto a tiempo nos permitirá mostrar progresos a los stakeholders del proyecto.

Al inicio de la fase se definirán las tecnologías utilizadas durante todo el proyecto, y se hará el setup inicial.

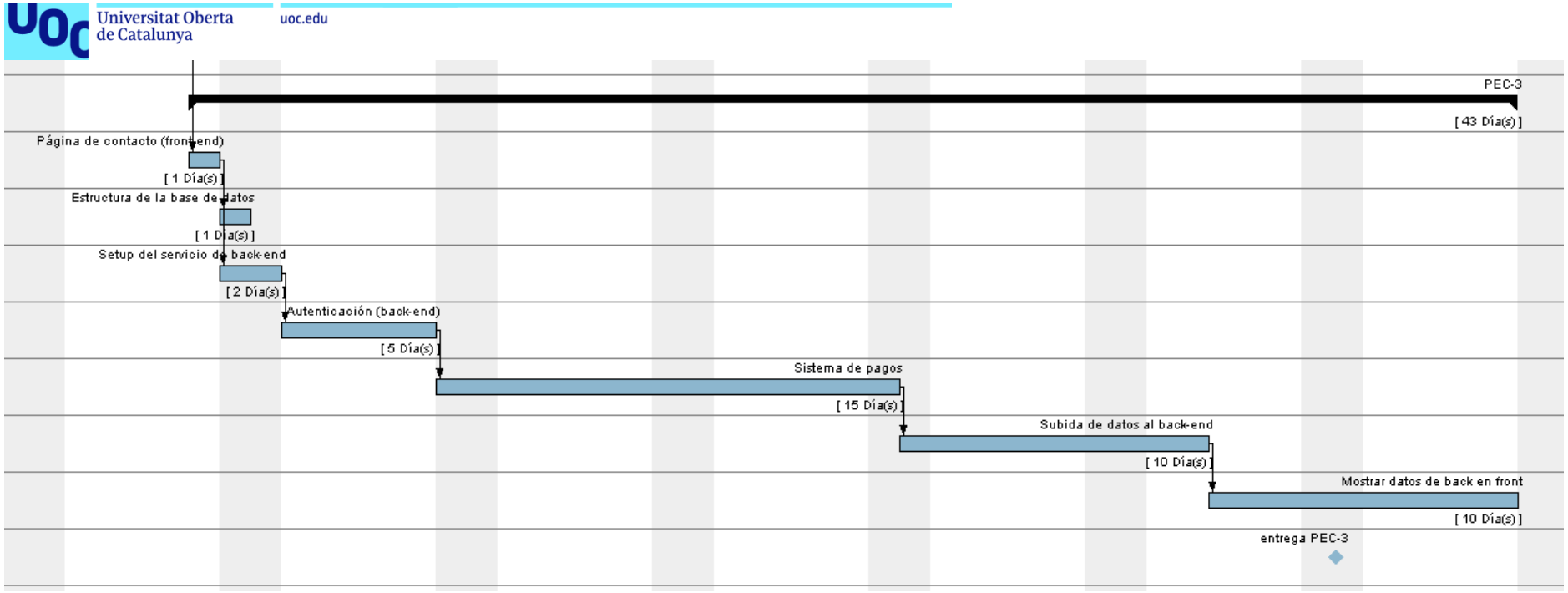


Figura 4. Planificación de la linea temporal de la PEC-3 (10 de noviembre - 18 de diciembre)

En este periodo de finalizarán los detalles del front-end menos relevantes y se empezará con el groso del back-end (sin poder llegar a terminarlo). Se ve como las fases del back-end son mucho más pesadas, dando especial importancia al sistema de pagos ya que es una parte crítica del funcionamiento de la página web.

La autenticación será más rápida que el resto ya que se utilizará el sistema dado por el proveedor de back-end.

La parte que no se visualiza al final es la inicialización de la tarea “Subir datos a back-end”.

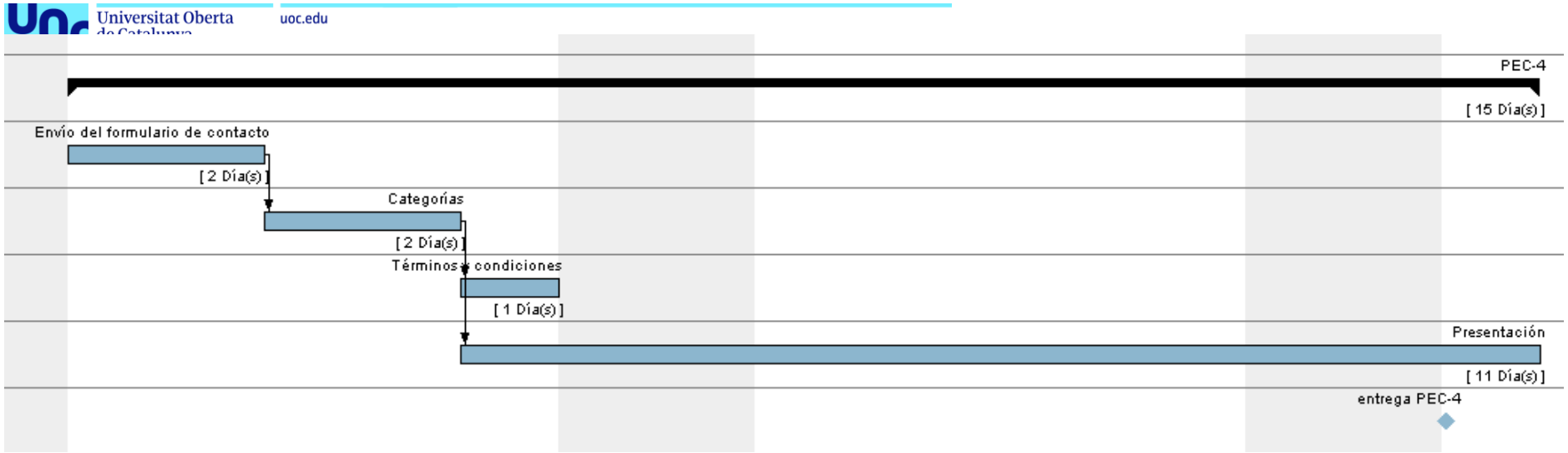


Figura 5. Planificación de la línea temporal de la PEC-4 (19 de diciembre - 16 de enero)

En esta línea temporal se muestra el final del proyecto, donde se acabará de dar los retoques finales al back-end para mostrar los datos en el front-end, y se finiquitarán tareas menores.

Gran parte del tiempo estará dedicado a la presentación final y a la documentación.

Esta planificación se puede ver más en detalle en el archivo adjunto “PEC2_planning_Plana_Ballber_Miquel.gan” entregado en la PEC-2, pero se tendrá que abrir con el software Gantt Project.

Cabe destacar que el desarrollo es en modo cascada, aunque, por ejemplo, las cards de desarrollo de front-end solo dependan de la de diseño y setup. Esto es debido a que solo hay un único desarrollador (yo) y por lo tanto solo me podré centrar en una tarea a la vez (dos como mucho si no son complejas).

Por este motivo se han creado algunas dependencias ficticias en el archivo del Gantt, ya que a nivel organizativo es más fácil gestionar un cambio si la tarea siguiente depende de la anterior.

Los recursos utilizados en este proyecto son:

- El tiempo del desarrollador (yo).
- Un ordenador portátil personal.
- Un ordenador portátil del trabajo con uso permitido por parte de la empresa para este proyecto como soporte en caso de fallar el primero.

2. Definición de tecnologías

Previamente a definir las tecnologías utilizadas en el proyecto, es necesario recordar que en los requerimientos del proyecto se ha establecido que se utilizará un Back-end as a Service (BaaS), que se escogerá **una vez finalizado el front-end**. Se especifican más detalles en el apartado de planificación del proyecto.

Por lo tanto, en este apartado se definirán solamente:

- Lenguaje a utilizar, juntamente con un framework de desarrollo.
- Framework de estilos, y guía de código de estilo a seguir.
- Otras tecnologías útiles durante el desarrollo.
- El hosting y el dominio.

Se dejará para el apartado de definición de back-end:

- El proveedor de Back-end as a Service.
- La API de gestión de pagos.
- La API o servicio de envío de emais.

2.1 Lenguaje

Para desarrollo web el estándar es utilizar Javascript, o su sucesor tipado, Typescript, por lo que esta decisión es sencilla.

En este proyecto se hará uso de **Typescript**, ya que el lenguaje tipado presenta una gran ventaja en el desarrollo y organización de proyectos. Poder definir el tipo de una variable u objeto, el tipo de los argumentos de una función y su tipo de retorno, entre otros, es la mejor manera de mantener un código organizado, lógico, y fácilmente leíble para otros desarrolladores en el caso de que el proyecto se expandiera. Por si fuera poco, los grandes frameworks de desarrollo dan soporte para Typescript (o incluso lo utilizan como lenguaje, como Angular).

2.2 Framework de desarrollo

En este paso será necesario detenerse más a valorar los diferentes frameworks de desarrollo existentes. Se van a evaluar brevemente los tres más grandes actualmente, React, Angular y Vue.

Esta evaluación no va a ser cuantitativa, por ejemplo, midiendo la performance de los tres frameworks, sino cualitativa, ya que no importa tanto el framework sino la comunidad y la comodidad del desarrollador con el framework.

Angular

La ventaja principal de Angular es que el uso de Typescript viene integrado, y encaja con nuestra necesidad de lenguaje. Además, es desarrollado por Google, por lo que cuenta con el respaldo de una gran empresa y varios años de rodaje en el mercado.

Su popularidad y uso, pero, han decaído respecto a sus dos competidores, y además su curva de aprendizaje es más alta por lo que nuevos desarrolladores tenderán a preferir, a la larga, otros frameworks. Esto implica que la comunidad existente tenderá a quedar reducida respecto a sus competidores.

Otro punto negativo de Angular, es que requiere de muchos más ficheros para un simple one-pager, mientras que los otros dos permiten tener una estructura de carpetas más sencilla y ordenada. Este es un punto clave, y por el cual se va a descartar el uso de Angular, debido a preferencias personales del autor.

React

React cuenta con el respaldo de Facebook (ahora Meta), y es utilizado por gigantes como Airbnb, Dropbox o Instagram. Además, es el framework con más popularidad del momento, y el que cuenta con mayores ofertas de trabajo y el mayor número de descargas. Cuenta con soporte para Typescript, aunque se requieren de algunas configuraciones extra.

Vue

Vue es el framework más nuevo, y aunque no está respaldado por una gran compañía, ha crecido mucho en popularidad en los últimos años. El problema de no estar respaldado es que es más fácil que quede desactualizado y caiga en el olvido.

Decisión final

Conociendo un poco lo que hay detrás de cada uno de los frameworks, solo resta tomar la decisión. Primero, basándonos en la popularidad, según Google Trends, React supera con creces a los otros frameworks:

Figura 6. Comparativa de popularidad de frameworks de desarrollo según Google Trends

Según npm trends, esto también se confirma, con una aplastante victoria por parte de React:

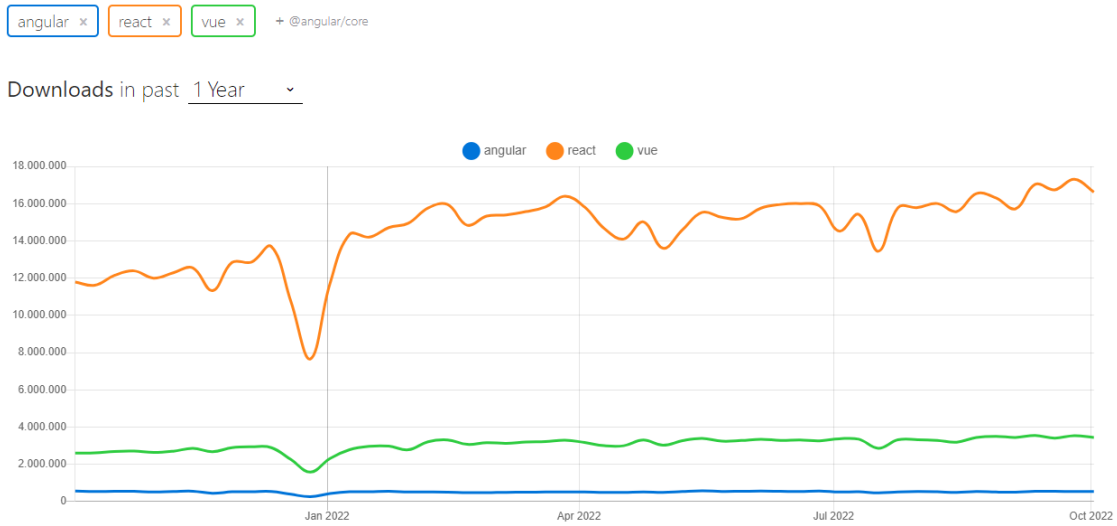


Figura 7. Comparativa de descargas de frameworks de desarrollo según npmtrends

Tiene sentido entonces que se utilice el framework más utilizado de la web en este proyecto. Además, a modo personal, he utilizado React en varios de mis proyectos y ese conocimiento me va a servir para entregar un mejor proyecto.

Como nota final, al usuario final no le interesa saber que framework es más rápido, o el que mejor gestiona componentes, sino que la página web funcione, por lo que decidirse por un framework que ya se conoce (y además es el más popular) es lo que tiene más sentido.

2.3 Framework de estilos

Existen dos grandes grupos en los frameworks de estilo, los UI kits, como **Bootstrap**, que permiten fácilmente crear una página con estilos predefinidos y menos código, y los utility first, como **Tailwind CSS**, que son mucho más flexibles, pero por el contrario requieren más código.

UI Kits

Las ventajas de un UI Kits son:

- Rapidez en el código.
- Estilos probados a los que todo el mundo está acostumbrado.
- Compatibilidad cross-browser testada.
- Estilo responsivo integrado.

Y sus desventajas:

- Si el estilo se empieza a desviar del kit original, requerirá de muchas sobreescrituras de estilo, resultando en un código mucho más grande y pesado, además de complicado de mantener.
- El look-and-feel de la página web será exactamente igual que muchas otras páginas web.

Utility first

Las ventajas de los utility first son:

- El tamaño de los archivos CSS es mucho menor, ya que contamos con muchísimas clases reutilizables.
- Mucho más personalizable que los UI Kits, para dar un look-and-feel único.

Sus desventajas:

- Para tareas simples, tendremos que escribir más código CSS, mientras que con un UI Kit lo tendríamos de un solo golpe.
- El estilo y el HTML están mezclados en un mismo archivo que puede llegar a ser muy grande.

Decisión final

Sabiendo que el framework de desarrollo, React, pone su foco en la creación de componentes reutilizables, esto da una ventaja a los utility first, ya que, para la mayoría de elementos, solo se tendrá que escribir el código de estilo una vez, por lo que se obtendrá prácticamente el mismo código que se podría tener con un UI-Kit, con mayor control sobre él.

Es por ello que el framework de estilos utilizado será **Tailwind CSS**, que cuenta con soporte para React en su documentación.

2.2 Guía de estilos

Aun utilizando Tailwind CSS como framework, es posible que sea necesario utilizar código CSS propio en determinados momentos, y es importante que siga una cierta convención. Por preferencia de uso, se utilizará BEM.

Se utilizará el paquete de iconos "Feather icons", que son sencillos y estilizados, y además tienen un paquete específico creado para React llamado "react-feather".

2.3 Otras tecnologías útiles

Aquí se listan otras tecnologías de las que se ha hecho uso durante todo el proyecto.

Github Copilot[5]

Un asistente de código por inteligencia a artificial, a través de una entrada en lenguaje natural. Gracias a Github Copilot, las tareas más repetitivas y estándar ahora se pueden hacer rápidamente a través de una línea de lenguaje natural.

Git y GitHub

Se utiliza git para el control de versiones, y GitHub como plataforma para almacenar este control de versiones en la nube. Al ser un proyecto con copyright, el repositorio es privado, por lo que en vez de ello se compartirá el código en un archivo zip.

Visual Studio Code

Se hará uso de la IDE de Visual Studio Code, que además permite la implementación de extensiones que nos ayudan a codificar más rápidamente.

Netlify

Se utilizará Netlify para hostear la página en www.promptmarkt.com. El dominio será comprado en Google Domains.

[5] https://github.com/features/copilot

3. Diseño y prototipado

En este apartado se definirá lo que esperamos a nivel de UX/UI del MVP, para después transportar estos requerimientos a un prototipo **no funcional** hecho con Figma.

3.1 Planteamiento del diseño

Fuera del listado de requerimientos que se detallan en el documento PEC1_rec_Plana_Ballber_Miquel entregado el 11 de octubre de 2022 junto con la memoria inicial, se debe plantear una primera idea o esbozo que facilitará el diseño del prototipo. Más adelante se detallan las herramientas que se han utilizado en el proceso de diseño.

Para plantear el diseño se ha utilizado la metodología mobile-first, en la que primero se diseña el estilo de la página vista en un dispositivo móvil, y se escala a dispositivos más grandes.

3.2 Prototipo de baja fidelidad

Los wireframes o prototipos de baja fidelidad están hechos con la herramienta Excalidraw, y se pueden abrir accediendo a https://excalidraw.com/ y cargando el archivo PEC2_Mocks.excalidraw adjunto en la PEC-2.

Para evitar sobrecargar la lectura de la memoria, se presentan estas imágenes horizontales que contienen todos los wireframes de móvil, tablet y ordenador:

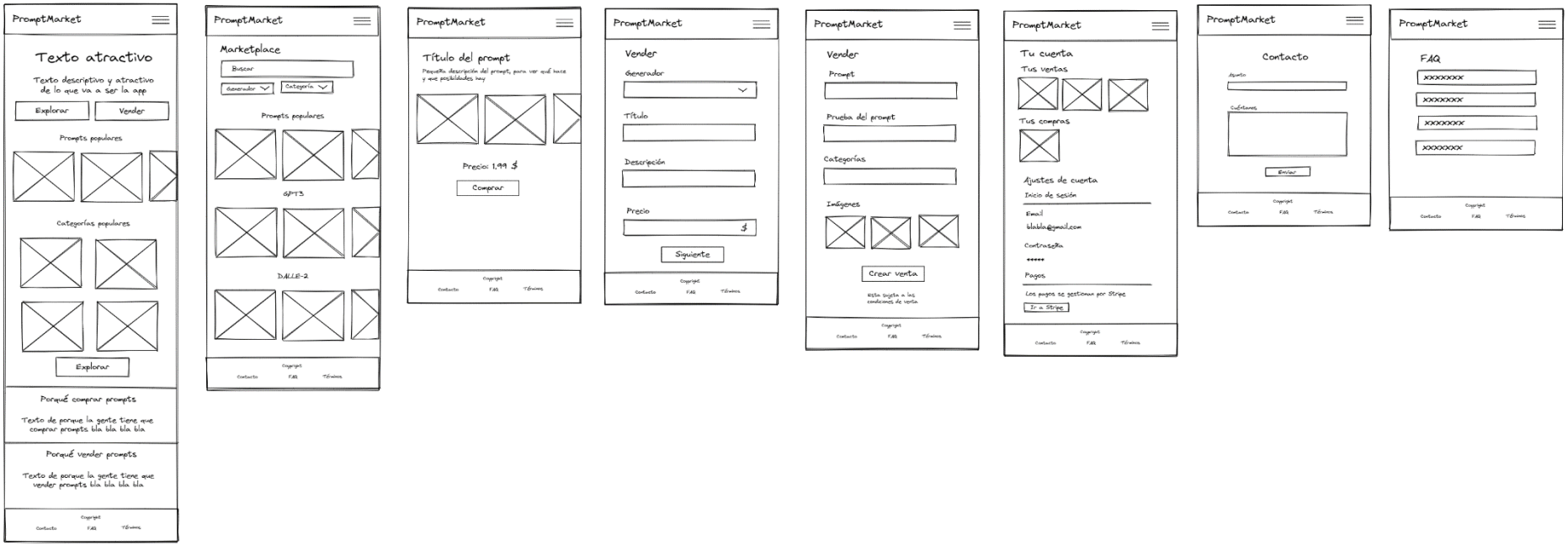


Figura 8. Prototipo de baja fidelidad de móvil

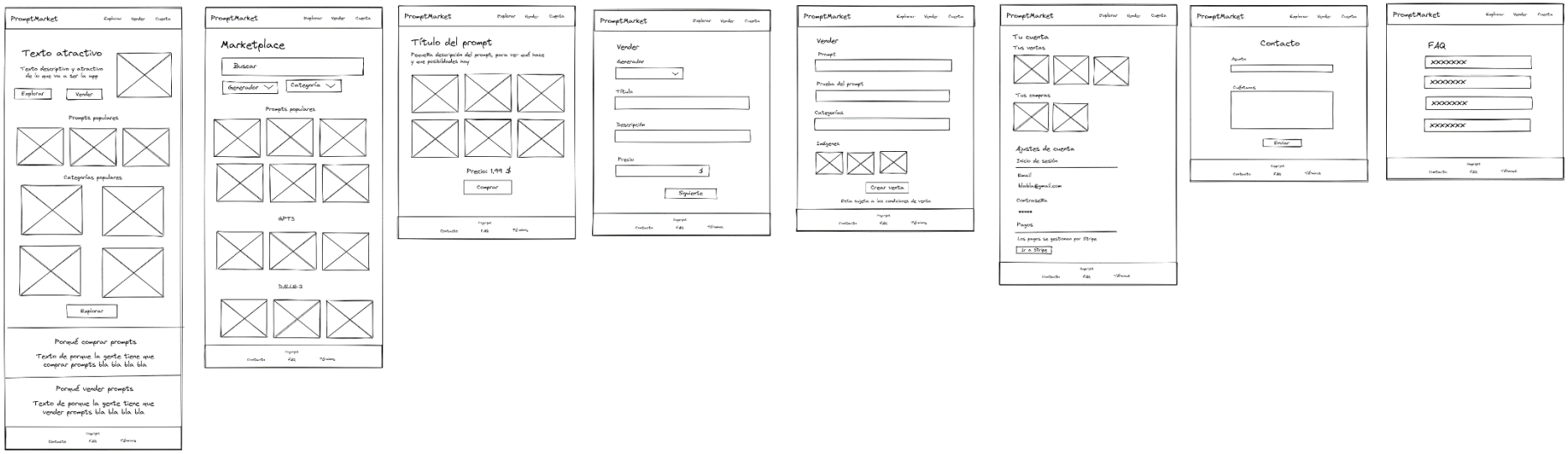


Figura 9. Prototipo de baja fidelidad de tablet

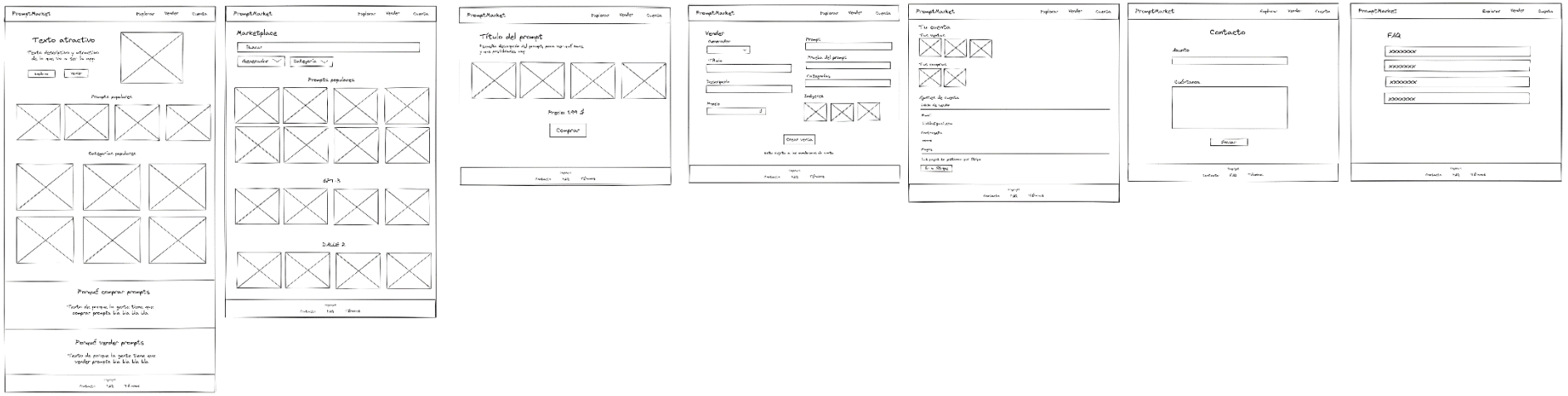


Figura 10. Prototipo de baja fidelidad de ordenador

Se puede ver el detalle de cada imagen en el **Anexo 1: Prototipo de baja fidelidad**. A continuación, se explican brevemente los wireframes:

El primer wireframe (izquierda) es el de la landing page o pantalla principal, donde se puede observar un mensaje que aclarará para qué sirve la página al cliente, además de dos botones call-to-action.

En la parte superior de la versión móvil se observa que en el header aparecerá el icono de las tres barras horizontales, simbolizando que hay un menú escondido. Ese menú no se ha diseñado puesto que serán cuatro textos en el centro de la pantalla. Además, se observa que se podrán ver prompts y categorías más populares, que también incitan al cliente a la acción.

En la parte inferior se encuentran dos textos sobre porqué comprar o vender prompts. Esto es típico en muchas páginas web, donde se explican los beneficios del servicio que se oferta. Finalmente se encuentra un footer muy sencillo con los enlaces a páginas secundarias de contacto, FAQ y Términos.

Tanto en la versión tablet como en la de ordenador, los únicos cambios notables son el tamaño de los elementos y la aparición de una imagen al lado del texto del título, que le dará cierto estilo a la página. Es una práctica muy común. También el cambio del header, que ahora tiene tres botones para navegar en vez de un icono.

Para un cliente comprador, la siguiente página sería ser la de explorar, para poder encontrar su prompt. Esta página contará con un buscador, así como con filtros de generador y de categoría. En su estado “untouched”, sin buscar nada, mostrará una colección de prompts populares y/o por categoría. Para adaptar esto a formato tablet y ordenador, lo único que se hace es aumentar el tamaño de los elementos, el resto queda igual.

La siguiente página es la del detalle de un prompt para cuando vas a comprarlo. En ella aparece la información más relevante, como el precio, título, descripción e imágenes de muestra, ya que el cliente requiere de esta información para poder comprarlo. También cuenta con un botón de comprar que redirigirá a la API de pagos, pero esta pantalla no tendrá diseño puesto que vendrá dada por la propia API. Para adaptar esto a formato tablet y ordenador, lo único que se hace es aumentar el tamaño de los elementos, el resto queda igual.

La siguiente página es la de poner un prompt a la venta. Para una venta, lo mínimo será rellenar un título, una descripción, un precio, para que generador es (DALLE, Midjourney…), el propio prompt, la prueba de que el prompt funciona (que será un enlace a una imagen pública generada con la inteligencia), la categoría o categorías a las que pertenece y unas imágenes que demuestren de qué es capaz el prompt. Como son muchos campos y en formato móvil se haría complicado rellenar todo a la vez, una buena práctica de UX es dividir el workflow en varios pasos, en este caso dos.

Se hará lo mismo para el workflow de tablet, que no dista mucho del de móvil excepto por su tamaño.

En cambio, para la versión ordenador, como todo cabe perfectamente en la misma pantalla, se ha decidido mantenerlo en una única para facilitar los cambios que pudiesen surgir al crear la venta:

La tercera página importante es la de cuenta, a la cual se accede a través del botón “Cuenta”, pero si no se ha iniciado sesión se deberá hacer a través de un modal que aparecerá en pantalla. Este modal no se ha diseñado porque es probable que el servicio de back-end provea de una manera de iniciar sesión sin codificar. Si hiciera falta, se codificaría una versión muy básica en la parte planificada para la autenticación.

Esta página de cuenta contará con los prompts comprados y vendidos, y con un set de ajustes para la cuenta como el cambio de email, contraseña o eliminación de cuenta. La gestión de pagos se hará con una API externa por lo que solo se redireccionará allí. Para la versión tablet y ordenador, no existen cambios, simplemente se ha adaptado el tamaño a la pantalla.

La siguiente página a la que podrán acceder los usuarios es la de contacto. Será una página muy sencilla con un formulario con un asunto y un cuerpo. La página se mantendrá igual tanto para móvil, tablet y ordenador:

Finalmente queda la página de las FAQ, que consistirá en unos desplegables muy básicos con un título y un texto en su interior. Se seguirá manteniendo la estructura tanto para los tres tamaños.

Estos son todos los wireframes diseñados para la página web. Es muy probable que durante la fase de prototipado y desarrollo, se encuentren partes de la página que no se habían pensado inicialmente pero que son necesarias.

Dado ese caso, los wireframes no se reharán, puesto que conllevaría mucho más tiempo rehacerlos y anotarlo en la documentación que directamente desarrollar la parte faltante. Puesto que se está construyendo un MVP, tiene sentido que el objetivo sea sacar un producto lo antes posible al mercado para validarlo con usuarios reales, antes que hacer un diseño perfecto de los wireframes o del prototipo.

3.3 Prototipo de alta fidelidad

A continuación, se muestra el prototipo de alta fidelidad realizado con la herramienta Figma[6] a partir de los wireframes del apartado anterior. Este prototipo presenta pequeñas variaciones y mejoras respecto a los wireframes, y permitirá definir la arquitectura de la aplicación, así como servir de modelo para el desarrollo del front-end.

[6] www.figma.com

Se puede ver el documento de Figma donde se ha prototipado en este enlace. Allí se encuentran los tres modelos: móvil, tablet y ordenador. El botón de "presentar" para verlo en un dispositivo no funciona ya que es un prototipo **no funcional**, es decir, que no tiene interactividad. Esto se debe a que no van a ser necesarios usability tests en esta fase del proyecto, puesto que estamos construyendo un producto cuya finalidad principal es testear con usuarios reales.

El motivo de no hacer usability tests radica en la propia naturaleza del MVP. Es un producto que construimos de manera rápida para poder validar con usuarios reales y un producto **real** si estamos respondiendo a las necesidades del mercado. Es por ello que los usability tests se harán una vez terminado el proyecto, como futuros pasos.

Entrando en diseño, lo primero es escoger los colores. Se ha escogido una paleta de colores neutra, con un negro para los elementos de contraste (#22242C) y un gris neutro para el fondo (#F6F6F6). Además, se hace uso de la paleta de colores de iOS, más concretamente el verde (#53D769), el azul (#007AFF) y el rojo (#C7372F).

Lo segundo es la fuente del texto. Una de las más comunes y que mejores resultados da es la Montserrat, que es la que se utilizará en el proyecto.

En las siguientes páginas se muestra el resultado final de traspasar los wireframes a un modelo de alta fidelidad. Si se desea ver el detalle de cada página, se puede hacer en el **Anexo 2: Prototipo de alta fidelidad**.

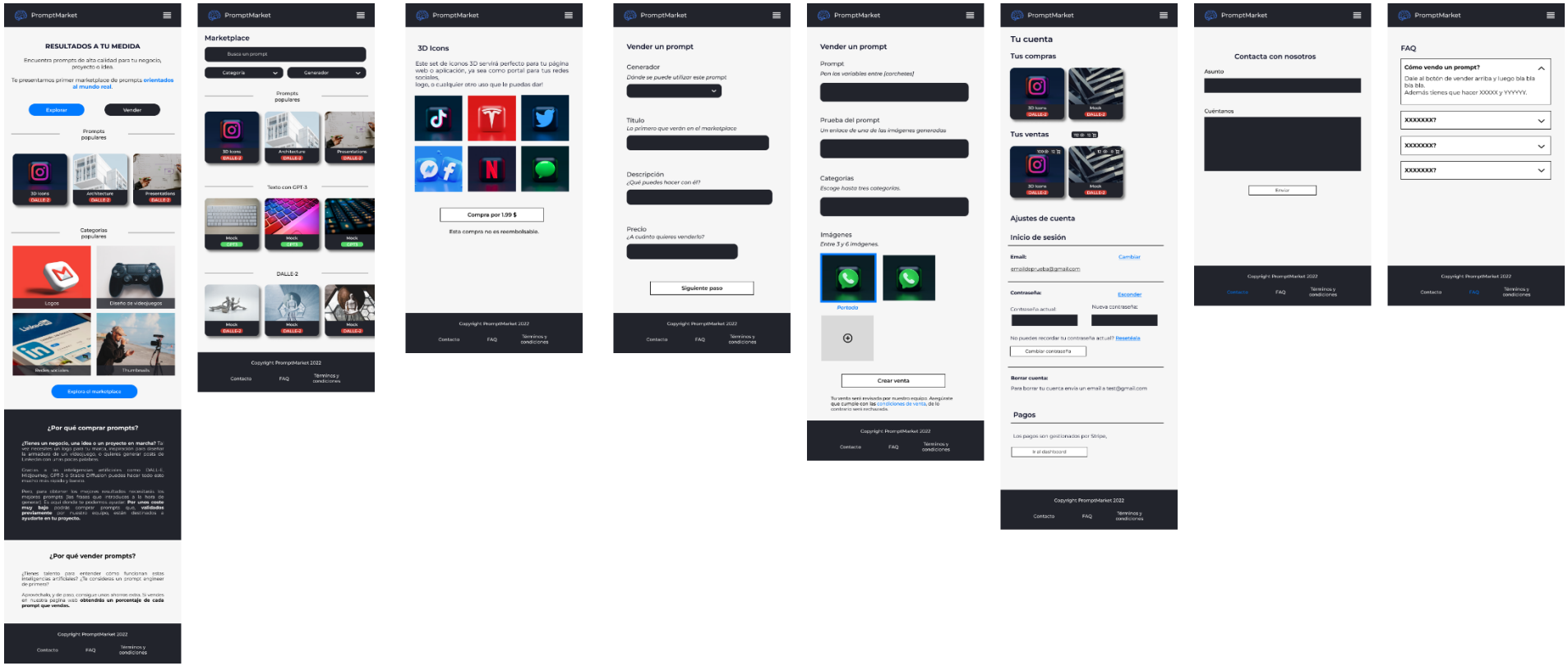


Figura 11. Prototipo de alta fidelidad de móvil

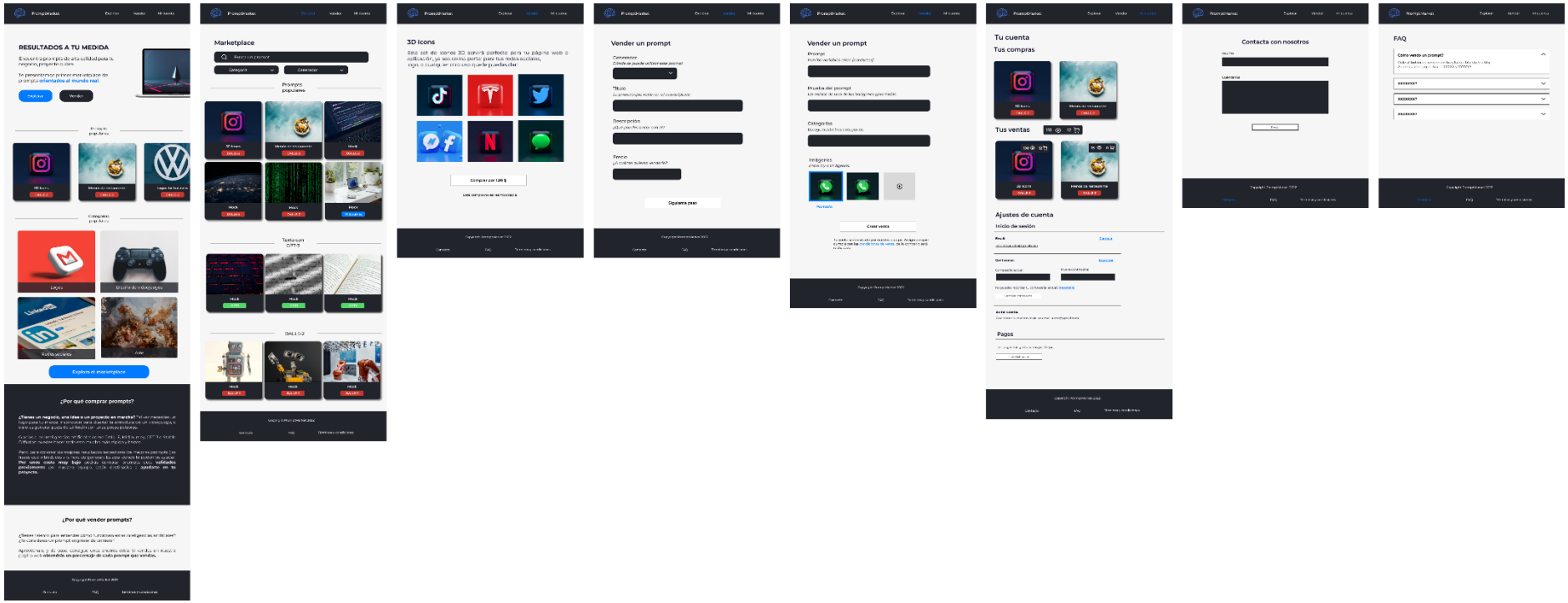


Figura 12. Prototipo de alta fidelidad de tablet

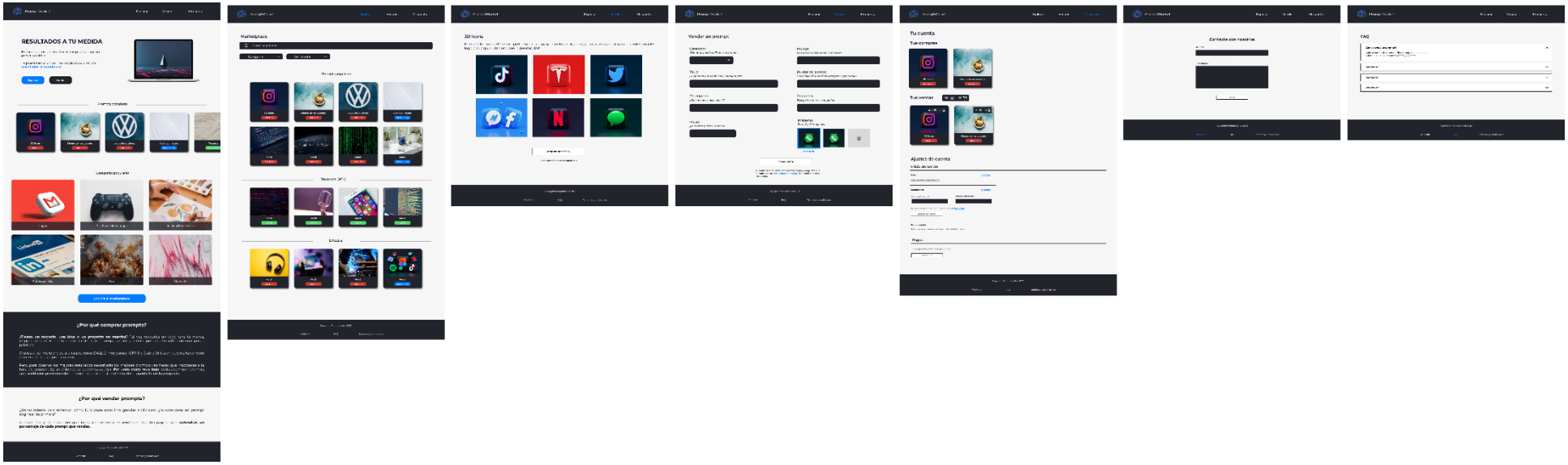


Figura 13. Prototipo de alta fidelidad de ordenador

Como se puede observar, la estructura se ha mantenido igual, aunque ahora es mucho más estética, con imágenes reales y la paleta de colores.

Se han introducido los mensajes reales que aparecerán en la web. También se ha introducido un logo en el header, que, como dato interesante, se ha diseñado con DALLE-2. Este será el logo de la página:

Figura 14. Logo de la página, diseñado con DALLE-2 por el autor

En el header también se ha marcado de color azul la página en la que se encuentra el usuario en ese momento.

También se ha diseñado el componente “card” de los prompts, donde antes simplemente aparecía un placeholder, y lo mismo para las categorías:

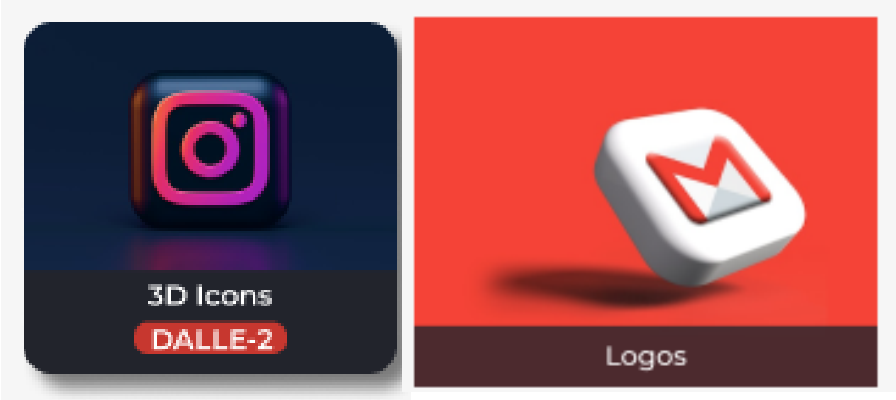


Figura 15. Componentes card de la página principal, para un prompt específico (izquierda), y para una categoría (derecha)

Las imágenes utilizadas son provistas por el plugin de Unsplash integrado en Figma, y son imágenes de uso permitido para uso académico. En la página web real no se utilizarán estas imágenes sino las generadas por las inteligencias, que no tienen copyright.

En la página de compra hay una excepción notable, y es que en el wireframe aparecía el precio encima del botón de comprar, mientras que en el prototipo de alta fidelidad aparece dentro de este. Esto es debido a que, una vez puesto en práctica, el precio encima no quedaba estético, y dentro sí.

Además, se ha añadido el texto de debajo de botón de comprar, que aparecerá en la página real, puesto que un prompt no debería ser reembolsable bajo ningún concepto (a menos que sea erróneo) al ser un concepto tan etéreo.

El resto de páginas es prácticamente igual al prototipo de baja fidelidad, solo que estilizado. Este modelo de alta fidelidad ya puede ser utilizado como referencia en el desarrollo.

4. Análisis de mercado y viabilidad económica

En este apartado se analiza el estado del mercado, así como la viabilidad económica del proyecto.

4.1 Mercado actual

La inteligencia artificial no es algo nuevo, sin embargo, el uso de prompts de lenguaje natural sí lo es, al menos al nivel al que se presentan ahora. Es por ello que el mercado de venta de prompts está aún en una fase “early adopters” en la curva de adopción de un producto:

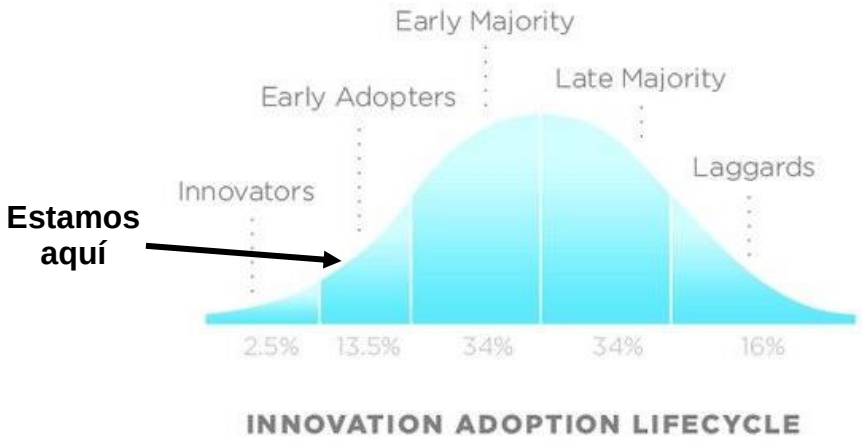


Figura 16. Curva de adopción de un producto

En el apartado de análisis DAFO se verán las ventajas e inconvenientes que tiene entrar en un mercado tan joven.

Es importante que, estando en un mercado joven, exista un crecimiento de interés en la temática, de lo contrario el proyecto fracasará a largo plazo. Observando las tendencias actuales se puede ver claramente que el interés está aumentando:

Figura 17. Tendencias de "inteligencia artificial" en los últimos 5 años

Se observa claramente que el interés por “inteligencia artificial” ha tomado un crecimiento exponencial en el último año. Esto es un muy buen indicador de que el mercado va a seguir creciendo.

Para DALL-E, en cambio, la tendencia es diferente:

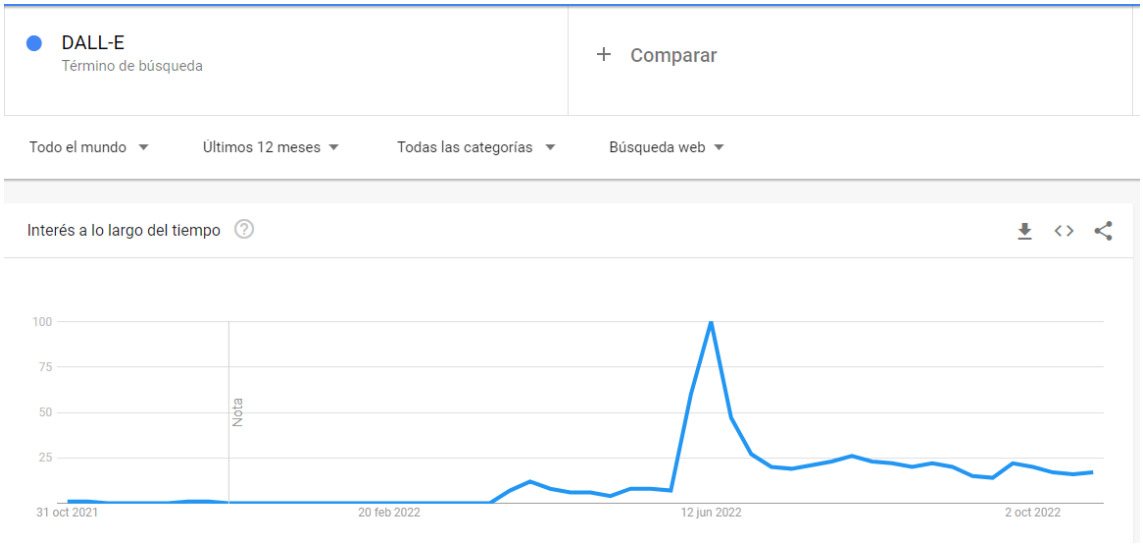


Figura 18. Tendencias de "DALL-E" en los últimos 12 meses

Se observa una tendencia muy alcista justo en el momento de salida de DALL-E 2, y después una bajada que corresponde a la perdida de interés de la mayoría, como pasa con todos los productos. Lo importante es que la tendencia no ha vuelto a cero, es decir, que se han afianzado usuarios. Como el producto tiene prospectos de mejora, es lógico esperar que este nivel de usuarios crezca.

Pasa exactamente lo mismo con los demás generadores de imágenes, por lo que no se mostrará su gráfico aquí.

Otro indicador de interés son los eventos y conferencias que surgen sobre inteligencia artificial. En el último año se han celebrado muchos de ellos, en esta página se pueden ver algunos de los más importantes.

La pregunta más importante: ¿Se puede dar una estimación del número potencial de clientes que podría tener una página web como la que se está desarrollando el proyecto?

Por desgracia, la venta de prompts es un mercado tan novedoso que no hay estudios de mercado disponibles, ni tan solo de campos más grandes como la inteligencia artificial en general, que los mencionen. Esto es muy usual en productos que entran en mercados jóvenes, el propio mercado se va definiendo a medida que más usuarios van entrando en él. Como hacer un estudio de mercado detallado no entra en el scope de este proyecto, no será posible dar estos datos.

Como nota final, es importante notar que la página web que está siendo desarrollada solo contempla la generación de imágenes, pero cada vez aparecen

más inteligencias artificiales con diferentes usos, y este es un mercado al que será posible expandirse con relativa facilidad. Esto implicaría nuevas fuentes de ingresos y nuevos posibles clientes.

4.2 Análisis de la competencia

La única competencia actual es PromptBase, aunque es muy probable que en los próximos meses aparezcan más mercados del estilo.

Aunque la idea la misma, en PromptMarket se contarán con dos diferencias muy claras:

1. No se generará nada con IA en la propia página web, a diferencia de PromptBase donde tienen un generador. Desde el punto de vista de rentabilidad es una pérdida de tiempo y dinero, puesto que el generador integrado en la página será seguramente mucho peor que los ya existentes y cien por cien dedicados a ello.

2. Los prompts vendidos en PromptBase son muy genéricos, y no tienen un objetivo claro. Los usuarios generan prompts de cualquier cosa, y si quedan bien, se pueden vender. En PromptMarket se optará por un enfoque distinto, donde todos los prompts deben tener una utilidad en el mundo real, orientada a emprendedores, artistas, negocios, empresas… cualquier cosa, pero que sea útil. Para ello, se pedirá al usuario que rellene un campo en la venta donde se explique la utilidad del prompt.

4.3 Análisis DAFO

Una práctica muy común es realizad un análisis DAFO (Debilidades, Amenazas, Fortalezas, Oportunidades). El de este proyecto luce así:

Figura 19. Análisis DAFO

Analizando en más detalle cada punto:

Debilidades

- Dinero limitado: Al no tener entrada de dinero asegurada ni inversión respaldando el proyecto, es posible quedarse sin fondos. Aun así es un caso improbable ya que los costos de una web son muy bajos actualmente.
- Dependencia de usuarios externos: Aunque se puedan crear prompts “in-house”, un punto clave de la página es que usuarios la utilicen para vender sus prompts. Sin ellos, el proyecto muy probablemente no funcionará.
- Oferta limitada de generadores: Al principio del proyecto solo se ofrecerán tres generadores de imagen, lo que podría llevar a usuarios a no vender en la página.

Amenazas:

- Nuevos competidores: La idea no es nueva, y es posible que entren más competidores al mercado, haciendo más difícil el éxito.
- Pérdida del interés en el producto: Es posible que esto solo sea una moda y que a lo largo del tiempo se pierda el interés y los usuarios de la página.
- Oferta gratuita del mismo producto: Puede haber creadores que promuevan su contenido de forma gratuita, haciéndonos perder usuarios. Esto es fácilmente solucionable teniendo mucha oferta de producto.

Fortalezas:

- Rápida adaptabilidad a cambios: Se pueden añadir nuevos generadores a la página web con mucha facilidad.
- Producto nuevo e innovador: Es una nueva tendencia con mucho mercado por delante y en un estado muy joven.

Oportunidades:

- Apertura de nuevas inteligencias: Cuantas más inteligencias salgan al mercado, más prompts se pueden crear y vender. Y en el mercado cada vez más aparecen nuevas inteligencias para hacer cosas muy variadas a través de prompts.
- Poca competencia actualmente: Al ser un mercado nuevo, aún no ha entrado toda la competencia que tendrá en el futuro.
- Nicho de mercado sin explorar: Al ser un mercado tan joven, se tiene la oportunidad de abrirlo, explorarlo e incluso modelarlo.

4.4 Viabilidad económica

El último paso es realizar un estudio de la viabilidad económica del proyecto. Primero se determinan los gastos fijos y variables:

- Dominio página web: En Google Domains encontramos dominios por 12 € al año. En concreto, el dominio que va a ser utilizado, **promptmarkt.com**, ya que promptmarket.com no estaba disponible, vale eso.
- Correo electrónico personalizado con el dominio de la página. Esto dará un aspecto más profesional al contacto por 4,68 €/mes por usuario durante los primeros 12 meses. Después costará 5,20 €/mes por usuario. De momento solo habrá un usuario.
- Back end as a service: Como aún no está definida la base de datos, y por lo tanto su coste, se hará una estimación al alza. Teniendo en cuenta que se almacenarán imágenes, que será lo más costoso, y que durante los primeros meses se estima tener un número de usuarios bajo, se considerarán 25 €/mes inicialmente. A medida que suba la carga de usuarios, mayor será este valor. Se estima 10 € más por cada mes, tirando al alza.
- Envío de emails, tanto de contacto (de fuera hacia dentro) como de información a los usuarios (de dentro hacia fuera, por ejemplo, al aprobar o rechazar un prompt). Los precios son muy asequibles, y contando que se superaría la capa gratuita, puede costar aproximadamente 20€/mes[7].
- Publicidad: Se aproximan unos gastos publicitarios de 200€/mes empezando el primer mes de lanzamiento, después de haber hecho las pruebas con el MVP, y con una duración de 4 meses para posicionar bien la página.
- API de pagos: La API de pagos, asumiendo un funcionamiento similar a Stripe, no tiene costes de entrada, pero si se llevan una parte por cada transacción de 0,25€ + un 1,4% en la unión europea, y un 2,9% en el resto del mundo. Esta cantidad se tendrá que descontar del posible beneficio recibido.
- Términos y condiciones: Por desconocimiento legal, lo mejora será subcontratar la creación de estos términos y condiciones. Por suerte, hoy en día existen muchas aplicaciones online que permiten crearlos al momento por un precio muy asequible. Se van a asumir 25€ de costes una sola vez.

Después se determina el rendimiento esperado a nivel de ventas y a nivel de usuarios activos en la página. Estos datos son muy difíciles de predecir, por lo que se ha intentado ser realista considerando la inversión en publicidad que se hará, el competidor actual, y los prospectos de mercado:

- Se estiman un crecimiento de ventas iniciando por la venta de 100 prompts al mes el segundo mes, hasta 10.000 prompts al mes al cabo de un año.

[7] Datos obtenidos de https://sendgrid.com/pricing/, API de referencia.

- En un caso malo, las ventas irían de 50 prompts mensuales hasta los 5.000 mensuales al cabo de un año.
- En un caso bueno, irían de 200 prompts mensuales a 15.000 mensuales al cabo de un año.
- Después de un año en todos los casos se aproxima a un 2% de crecimiento mensual manteniendo las ventas anteriores.
- El precio medio del prompt será de 2 €.
- El usuario final se llevará un 70% de la venta, y una parte del beneficio irá a la plataforma de pagos según lo definido.
- Se pagarán impuestos:
    - 21% de IVA.
    - IRPF dependiendo del beneficio después de impuesto y cuotas (19% hasta 6.000 €, 21% entre 6.001 € y 50.000 € y 23% a partir de 50.000 €). Los gastos deducibles ayudan a no pagar IRPF hasta un total de 2000 € al año.
- Cuota de autónomo (asumimos que de momento se hará por esta vía y no a través de una sociedad): Existe una regla no escrita pero aplicable que dice que si los ingresos son menores al SMI (1000 €/mes actualmente), no se debe pagar la cuota. Además, en 2023, el año de lanzamiento, se ha adaptado la cuota de manera que los primeros doce meses se paga una tarifa plana de 80 euros.

Siempre teniendo en cuenta que estos números son una estimación, se hará una predicción de los próximos 12 meses para los 3 casos, y se estudiará cada uno de ellos por separado.

La hoja de cálculo Excel donde se han realizado los gráficos y cálculos correspondientes se encuentra adjunta con el nombre PEC2_econom_Plana_Ballber_Miquel.xlsx entregado en la PEC-2. El resultado del estudio es el siguiente:

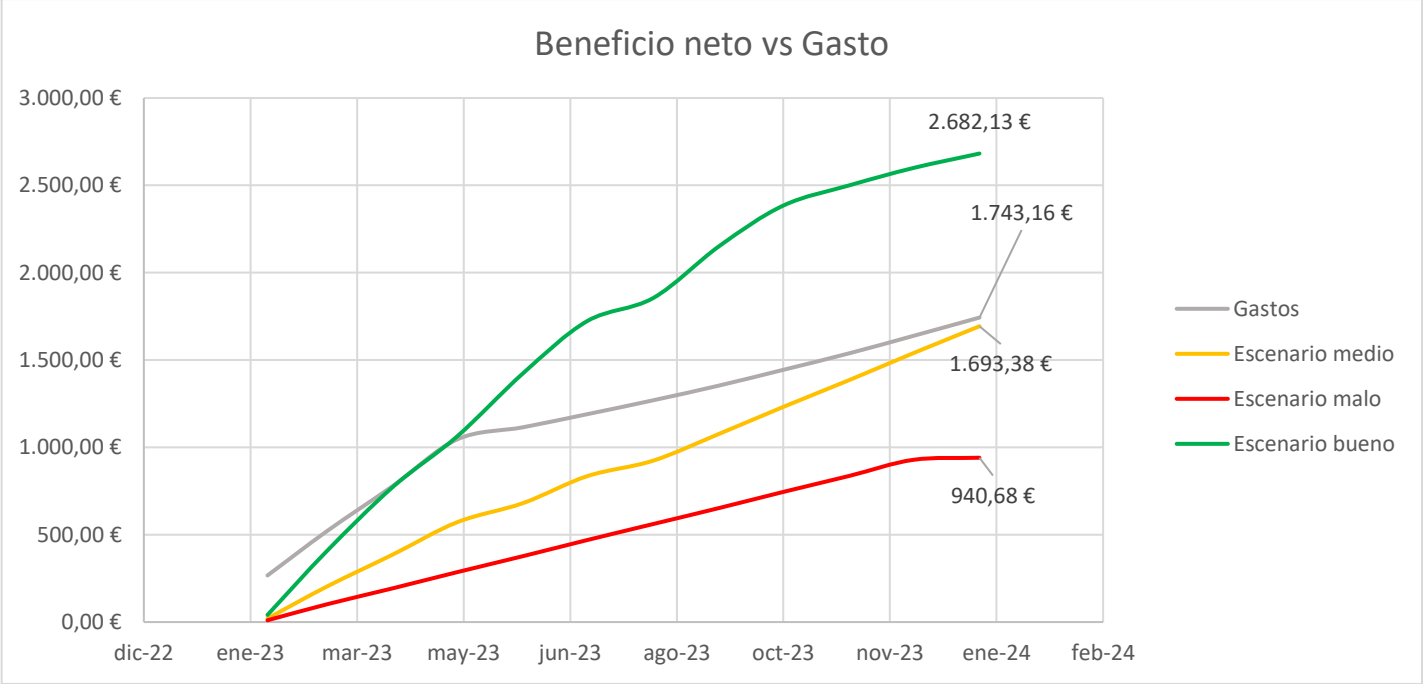


Figura 20. Estudio económico de diferentes escenarios

Los escenarios estudiados son desde febrero de 2023, la presunta salida de la página web, hasta enero de 2024 (incluido). En cualquier caso, aplicaría exactamente lo mismo en diferentes meses, ya que no es un producto estacional.

Es importante remarcar que los escenarios ya han tenido en cuenta impuestos, cuotas y otras tasas.

Se puede observar que tanto el escenario malo como el escenario medio están por debajo de los gastos. El escenario malo está muy por debajo, y se esperaría un resultado mejor de un escenario medio. Aun así, no es preocupante, puesto que se encuentra muy cerca de la rentabilidad, y en los próximos meses la superaría si las ventas siguen el curso descrito de un aumento del 2%.

En el caso del escenario malo se debería replantear la viabilidad del proyecto, y cerrar la página en caso de ser necesario.

El escenario bueno supera con creces los gastos a partir de abril de 2022, lo cual implicaría que se tendrían 9 meses de beneficio, el cual se podría reutilizar para mejorar la página o invertir en marketing.

En segundo lugar, también es interesante saber cuál sería el mínimo de ventas cada mes para, al menos, no entrar en pérdidas. Este cálculo, debido a que en España los impuestos varían según los gastos deducibles, pero también el rango de ingresos, es más complicado hacer el cálculo a la inversa. Es por ello que se ha simplificado y se han tenido en cuenta más impuestos y cuotas de las que se han tenido en cuenta para el cálculo de los escenarios.
Esto implica que el siguiente gráfico está hinchado, y que las ventas necesarias reales serán un poco inferiores a las que se muestran aquí. Esto en realidad es bueno, puesto que podría haber gastos imprevistos. El gráfico queda así:

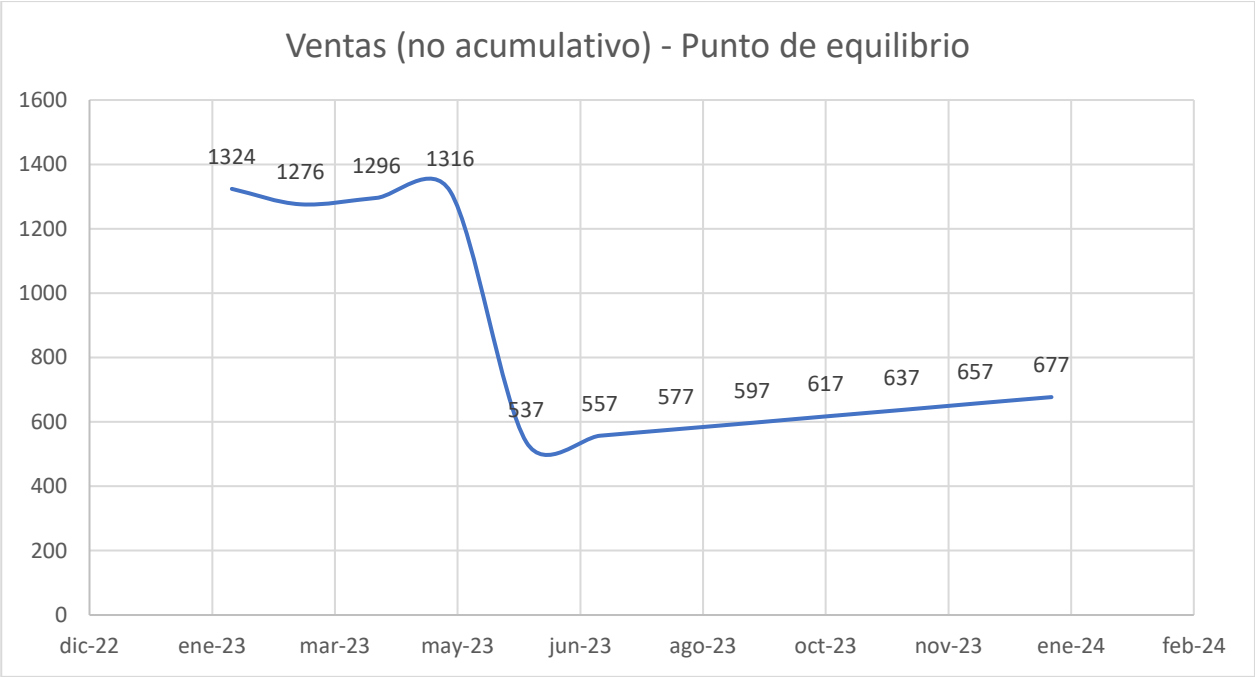


Figura 21. Ventas necesarias mes a mes (no acumulativo) para no tener pérdidas.

El total de ventas necesarias para alcanzar un punto de equilibrio durante los primeros doce meses sería de aproximadamente 10.068 en total.

Como se puede observar, los primeros doces cuatro meses las ventas necesarias para mantenerse sin pérdidas son muy elevadas. Esto es debido a la inversión en marketing, que después se deja de hacer y por ende baja el número necesario de ventas.

Si se desarrolla una tabla, se pueden ver las “milestones” de ventas mes a mes:

Mes	Ventas acumulativas
Febrero 2023	1324
Marzo 2023	2600
Abril 2023	3896
Mayo 2023	5212
Junio 2023	5749
Julio 2023	6306
Agosto 2023	6883
Septiembre 2023	7480
Octubre 2023	8097
Noviembre 2023	8734
Diciembre 2023	9391
Enero 2024	10068

Por supuesto, los primeros meses será imposible llegar a las milestones, pero es fácilmente remontable una vez se haya cogido tracción. Es también bueno que el número de ventas final esté cerca del número de ventas estimado para el escenario medio, quiere decir que no se iba tan desencaminado en la estimación.

5. Desarrollo del front-end

En este apartado se comentarán los detalles más relevantes del desarrollo de front-end, no el código en sí. Para ver extractos de código de las partes más importantes del front-end, se puede ver el **Anexo 3: Extractos de código del front-end.**

5.1 Estructura de carpetas

Una vez inicializado el proyecto con react-create-app, es de mucha utilidad organizarlo en carpetas que representen lo que contienen. De momento, solo existen las carpetas necesarias para el front-end:

src:
- assets
  - images (se guardan las imágenes del proyecto)
- components (archivos tsx que contienen componentes reutilizables)
- constants (constantes del proyecto, como categorías o generadores)
- modules (módulos reutilizables, como autenticación)
- screens (pantallas que verá el usuario)

Con esta estructura tan sencilla se puede organizar todo el front-end.

Es muy interesante remarcar que no se han utilizado archivos de estilo, sino que gracias a Tailwind se ha implementado todo el estilo directamente en los archivos .tsx de React, reduciendo el número de archivo necesarios y por lo tanto la facilidad para navegar por el proyecto.

5.2 Diferencias entre el diseño inicial y el desarrollo

A medida que se avanzaba en el desarrollo del front-end, iban apareciendo elementos que no se habían diseñado en la fase previa, pero eran necesarios, así como pequeños cambios que quedaban mejor de otra manera una vez visto en dispositivos reales. En este apartado se listan estos cambios.

Tarjeta del prompt

Se había planteado la tarjeta del prompt con el generador abajo y el título arriba, pero después de varias pruebas, los mejores resultados se obtenían con el generador en la parte superior, entre la imagen y el texto.

Figura 22. Tarjeta del prompt, antes (izquierda) y después (derecha)

Página de venta: Estado sin inicio de sesión

Durante el desarrollo se vio la necesidad de que el usuario estuviese registrado y con la sesión iniciada para vender, pero no se había pensado el diseño así inicialmente. Por ello se desarrolló un módulo muy sencillo de autenticación, aunque queda pendiente ver si el Back end as a Service proveerá el suyo propio (de no ser el caso se podrá usar este).

Este estado sin inicio de sesión consistirá en mostrar un texto llamativo y el módulo de autenticación para poder iniciar sesión, tal y como se muestra en esta imagen:

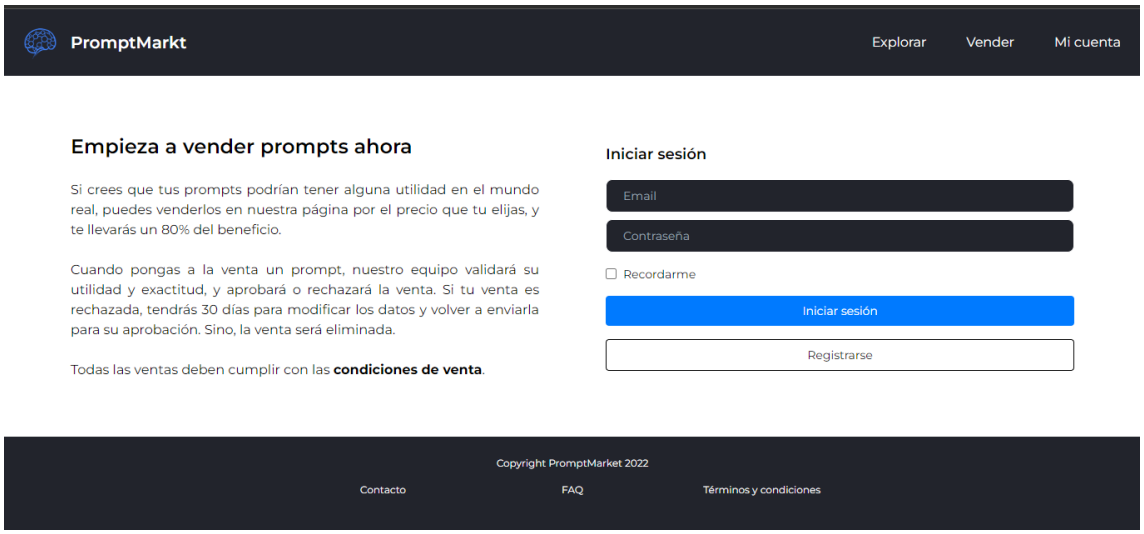


Figura 23. Página de venta, estado sin inicio de sesión

También se ha adaptado a diferentes tamaños de pantalla.

Página de venta: Campos extra y eliminación temporal de GPT-3

A medida que se iba aprendiendo sobre este nuevo mercado, quedaba claro que faltaban tres campos en la venta:

- El uso del prompt, donde el autor de este último explica la mejora manera de utilizarlo y sus variaciones.
- La razón por la que es válido para el mundo real. Este campo no es obligatorio y es solo para la validación interna.

Además, para facilitar el sistema de venta y compra, cada prompt solo pertenecerá a una categoría.

También se ha eliminado GPT-3 del MVP ya que funciona diferente al resto: No contiene imágenes, solo una de portada, y sus prompts son un poco más largos. La intención es que después de validar el MVP, se modifique el workflow de venta para que vaya por pasos en todos los dispositivos, y dependa del generador escogido, pero para el MVP y este proyecto con los generadores de imágenes es suficiente.

Estado de los prompts vendidos

Durante el desarrollo también quedó patente que el estado de un prompt en venta no estaba claro, faltaba información sobre si estaba pendiente, rechazado o aprobado.

Para ello, se ha desarrollado un modal que queda por encima de la tarjeta del prompt (solo en la pantalla de cuenta aparecería), que indica su estado, como se puede ver en esta imagen:

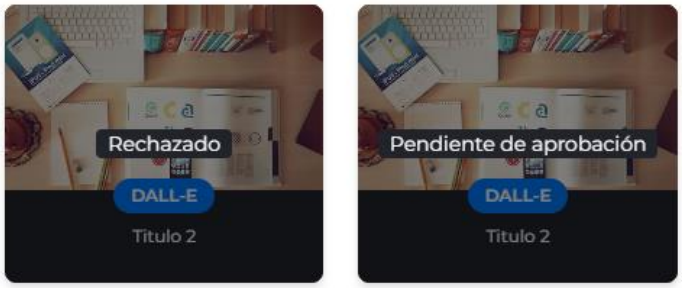


Figura 24. Estado de un prompt vendido, pendiente o rechazado

Página de un prompt sin iniciar sesión

Igual que en el caso de las ventas, quedó patente que para comprar un prompt se necesitaría una cuenta, pero no se había diseñado un estado donde el usuario no hubiese iniciado sesión. Para ello simplemente se ha añadido un texto pidiendo que se inicie sesión en vez del botón de compra:

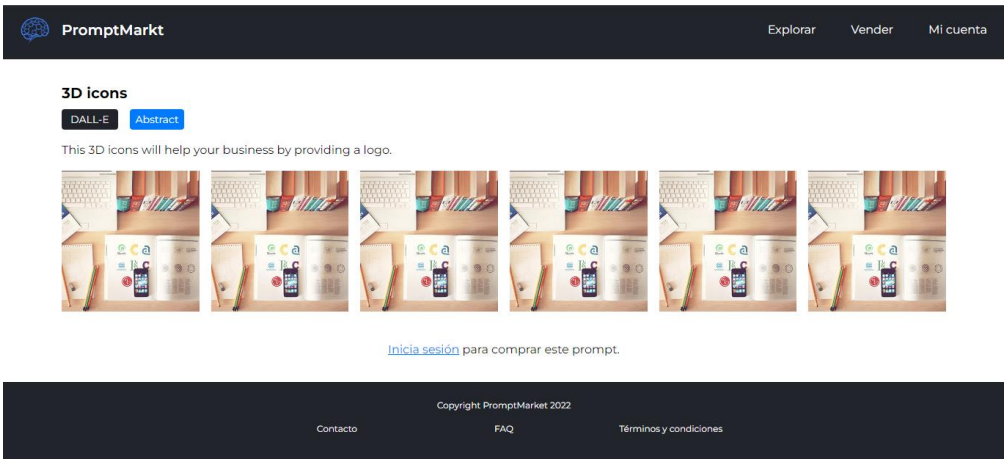


Figura 25. Página de prompt sin iniciar sesión

Página de un prompt una vez comprado

Lo mismo ocurría cuando el usuario había comprado el prompt, no se había diseñado un estado para ello. Simplemente se trataba de añadir el prompt y su uso debajo de las imágenes:

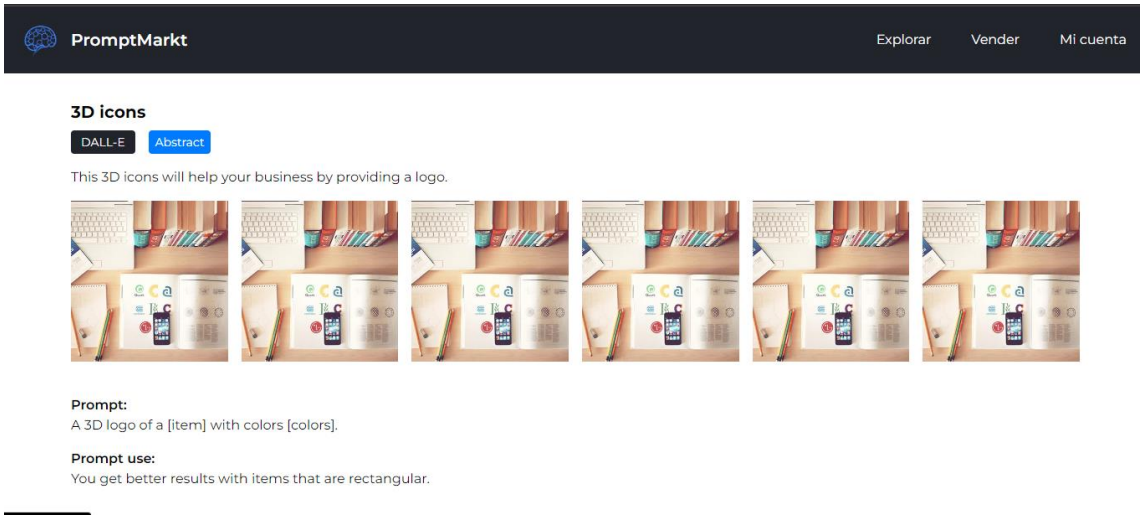


Figura 26. Página de prompt ya comprado

Página de validaciones

Se ha creado la página de validaciones en la siguiente url: www.promptmarkt.com/#/validate. Esta página solo será accesible para los super usuarios de la cuenta, por lo que es posible que en el momento de acceso ya no sea visible a menos que se tenga una cuenta de super usuario. En cualquier caso, este es su aspecto:

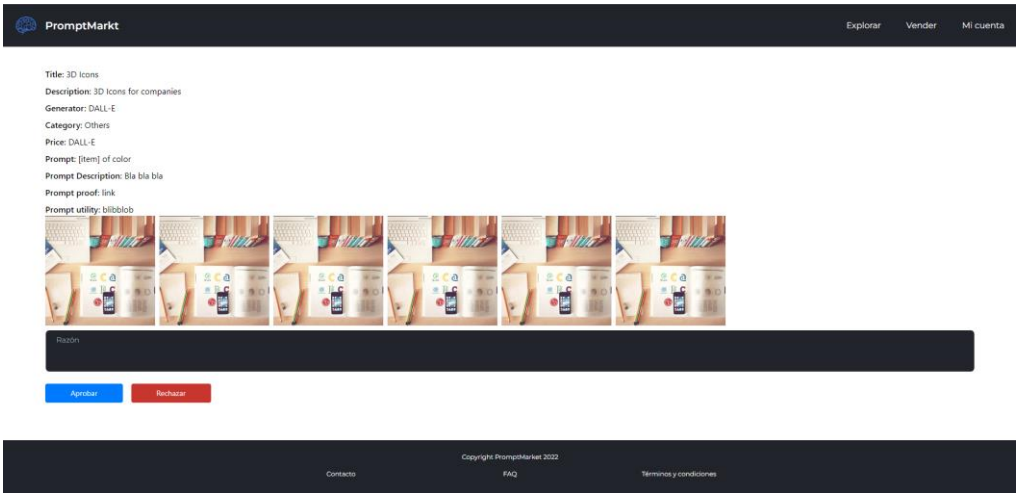


Figura 27. Página de validación

Se puede apreciar que su estilo es mínimo, ya que es una página de uso solo para super usuarios que validan prompts, por lo que no es necesario que sea bonita, sino funcional. Además, se pide añadir la razón de rechazo.

El resto del front-end es prácticamente igual al diseño hecho en Figma.

6. Definición del back-end

Una vez terminado el front-end, ya se tiene mucho más claro las necesidades del back-end, y por ello es más fácil decidir los servicios utilizados.

6.1 SQL o NoSQL

El primer paso para definir el back-end es saber cómo será la estructura de datos. Exsiten dos, SQL (relaciona) y NoSQL (no relacional), cada una con sus ventajas e inconvenientes:

SQL

La base de datos relacional es la base de datos que se ha utilizado desde hace más tiempo, y es aun la que prevalece en muchas empresas. Hacen uso del lenguaje SQL.

Estas bases de datos tienen tablas con una estructura definida, y las tablas tienen relaciones también claramente definidas. Esto es bueno para aplicaciones con una data altamente estructurada y pocos cambios, pero malo para aplicaciones con data no estructurada o con alta probabilidad de cambio en el futuro. Al ser tan rígidas, aseguran que la base de datos tendrá una integridad en datos muy alta. Esto es importante en aplicaciones con una alta necesidad de consistencia, como podrían ser las de transacciones financieras.

Las queries son muy consistentes, haciendo muy fácil la unión de dos tablas de datos, por ejemplo, y además permiten hacer queries mucho más complejas que las NoSQL.

NoSQL

Las bases de datos no relacionales son un poco más jóvenes que las SQL, pero no por ello peores. Es posible que en aplicaciones nuevas surja la necesidad de tener grandes cantidades de datos sin estructura fija, y para ello las NoSQL son perfectas.

Hay diferentes tipos, en los cuales no se entrará, pero todos ellos tienen algo en común, y es que su estructura de datos es muy flexible, cada “paquete” de datos puede tener una estructura diferente y no habría ningún problema en la base de datos. Esto a su vez es un problema si la fiabilidad de los datos es necesaria.

Una ventaja es que son capaces de escalar muy rápidamente y a través de diferentes localizaciones, lo que las hace perfectas para tener grandes cantidades de datos distribuidos por el mundo, y para gestionar mucho tráfico.

Un problema que tienen es que las queries se pueden volver muy complejas (o imposibles) si se requiere algo que salga fuera de un simple Update o Retrieve. También puede ser necesario actualizar diferentes puntos de datos en vez de uno solo dependiendo de nuestro modelo.

Decisión final

Conociendo las ventajas e inconvenientes, se va a evaluar cuál sería la mejor opción. Para ello, se listan una serie de características y requerimientos que tiene la página web:

- Data poco cambiante. Siempre va a ser la misma ya que se focaliza en la venta de un único producto.
- Necesidad de poder relacionar múltiples datos, como cuenta de usuario, prompt, número de ventas e imágenes.
- Cantidad de datos limitada. Se irán eliminando datos de prompts rechazados al cabo de 30 días de su rechazo si el prompt no se ha corregido.

Con esto, la decisión es clara, una base de datos **SQL**.

6.2 Back end as a service

Es momento de decidir el back end as a service que se utilizará. Esto es un punto muy importante, ya que una mala decisión puede acarrear problemas en el futuro.
Para hacer la toma de decisión, se debe hacer una evaluación de los mejores proveedores de este servicio y que cumplan con estos requerimientos:

- Uso de base de datos SQL o PostgreSQL.
- Sistema de autenticación integrado, no siendo estrictamente necesario un módulo front-end de autenticación.
- Precio asequible para páginas de nueva creación.
- Storage de imágenes integrado.
- Funciones de servidor. Este punto no es obligatorio, pero si deseable, ya que a veces es interesante lanzar funciones que se ejecuten en el lado del servidor en vez de en el lado del usuario, para evitar problemas con usuarios malintencionados, o simplemente para automatizar procesos como podría ser enviar un email o un SMS, o integrarse con servicios externos como podría ser la API de pagos.

Con esto en mente, los mejores proveedores que cumplen estos requisitos son:

AWS Amplify

Con el soporte del gigante de AWS detrás, AWS amplify es uno de los back-ends más completos de todo el mercado. Además de contar con el soporte de AWS, también permite usar todos sus servicios y las capas gratuitas de cada uno.

El mayor problema es que si se desconoce AWS, la barrera de entrada es complicada, y la documentación se hace difícil de navegar.

Supabase

Es una alternativa a Firebase que recientemente ha recibido una ronda de financiación para seguir creciendo. Es un competidor que ha ganado mucha tracción en los últimos años y un candidato a tener en cuenta.

Como ventajas, gestionan todo lo necesario para el proyecto, y su base de datos está en PostgreSQL, un plus para futuras migraciones a otros servicios. Cada vez más se está usando como una alternativa a Firebase y convirtiéndose en un nuevo estándar, siendo además una alternativa de la que se puede salir fácilmente si es necesario migrar (mientras que de Firebase no lo es).

Como desventajas, aun teniendo una versión gratuita, el precio es un poco más alto que el de otros competidores como AWS Amplify o Firebase, y la capa gratuita más pequeña, además de pausarse al cabo de 1 semana de inactividad en este modo. Además, es un servicio relativamente joven.

8base

Uno de los proveedores principales de back-end en low-code, aunque no tiene tanta tracción como los anteriores.

Aun así, es muy avanzado y permite a gente sin conocimientos técnicos (o apenas conocimientos técnicos) crear una back-end a medida. Ofrece además muchos servicios que pueden ser útiles.

Como puntos negativos, el cobro funciona por llamadas y por líneas de base de datos guardadas, en vez de por tamaño transferido o por tamaño guardado en base de datos. Esto representa una diferencia grande respecto al resto de la industria, además de un cambio negativo en el caso de tratarse de datos pequeños, como sería el caso de esta página. En general pierde en precio contra sus competidores.

Decisión final

El proveedor que se ha elegido será **Supabase**, ya que en los próximos meses y años se podrán ver nuevas features que afectarán muy positivamente a su uso, e incluso podrán reducir sus precios para competir con los grandes.

Además, mirando tendencias en Google Trends, se puede ver el aumento de interés en el último año por este proveedor, lo cual es un último indicador de que la decisión es buena:

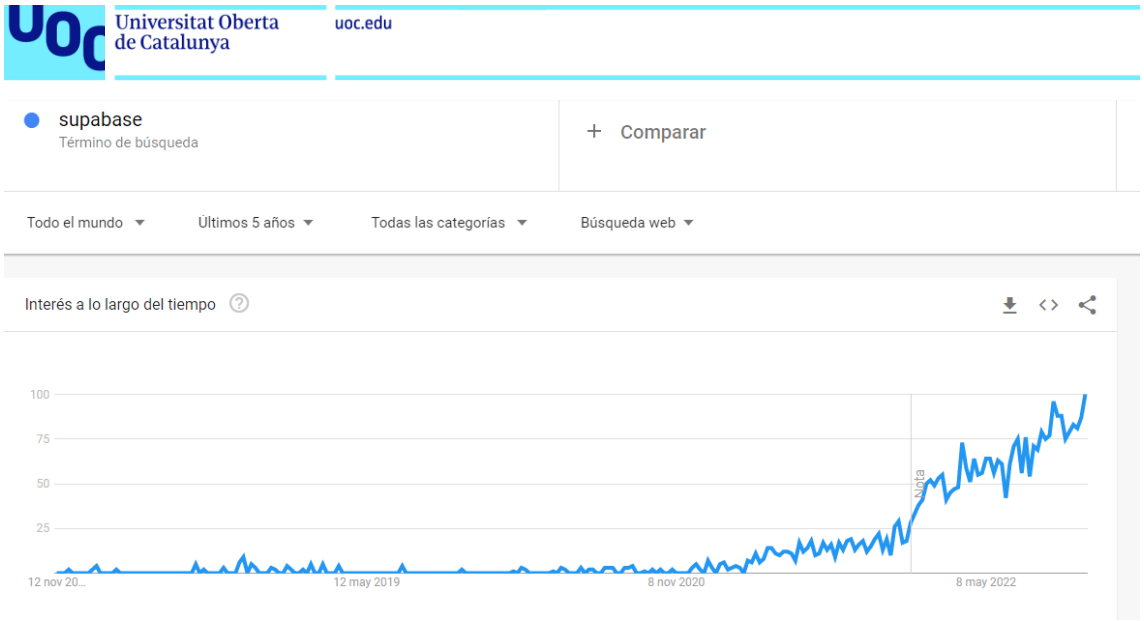


Figura 28. Búsqueda de Supabase en Google Trends

Con esto decidido, restan las APIs que se van a utilizar.

6.3 API de pagos

En este apartado se será más breve, y no se entrará en detalle de los diferentes proveedores de API de pagos. Se han considerado Stripe, Square API y PayPal, y se ha tomado la decisión de seguir con **Stripe**.

Esta decisión viene tomada porque Stripe ofrece precios competitivos, con los cuales además se hizo la predicción de costes, y además ofrece muchas alternativas para gestionar los pagos. La que va a ser utilizada será Stripe Connect, que es perfecta para un e-commerce del estilo de este proyecto.

6.4 API de envío de emails

Se utilizará MailerSend para poder hacer el envío de correos transacionales. La razón es puramente económica, ofrecen una capa gratuita de 12.000 emails, con conexión tanto por API como por SMTP, y sus precios de suscripción son muy económicos y más que suficientes para el nivel esperado de emails en la página.

6.5 Gestión del estado global de la aplicación

En algún momento será necesario que guardar valores globalmente de manera que puedan ser accedidos en cualquier momento por el usuario sin necesidad de hacer muchas queries o traspasar datos entre componentes de manera ineficaz.

Para ello se utilizará Redux, que es la librería más conocida y utilizada para el manejo del estado global.

6.6 Arquitectura de la aplicación

Ya definido cómo queremos el back-end, es necesario definir la estructura de la aplicación.

6.6.1 Datos necesarios desnormalizados

El primer paso es crear las entidades que representan los datos sin dividir en diferentes tablas (es decir, **sin normalizar**), simplemente definiendo que datos son necesarios. Después de esto se afinará más para ver si se pueden dividir las tablas y construir relaciones. Entre “<>” se encuentra el tipo del dato.

Datos de los prompts

- ID único <uuid>
- Título <string>
- Descripción <string>
- Categoría <string>
- Generador <string>
- Precio <number>
- Unidades del precio <string>
- Prompt <string>
- Uso del prompt <string>
- Prueba del prompt <string>
- Utilidad del prompt <string>
- Autor <uuid>
- Imágenes del prompt (entre 3 y 6) <Array<string>>
- Imagen de portada <string>
- Estado (aprobado, rechazado, pendiente…) <string>
- Razón del rechazo (si existe) <string>
- Momento de creación <timestamp>
- Ventas <number>
- Visitas <number>

Datos de Stripe

- Stripe Account ID <uuid>
- Product ID <string>
- Price ID <string>
- Autor <uuid>
- Prompt ID <uuid>

Estos serían todos los datos necesarios para el funcionamiento. Lo primero que se aprecia es que no hay tabla de usuarios, ya que Supabase lo maneja internamente a través de una base de datos escondida, para que el usuario no tenga que hacerlo.

Como se puede apreciar esta estructura tiene **muchísimos puntos de mejora**, ya que no tiene sentido trabajar en SQL con una estructura tan grande. Se puede subdividir en tablas que luego se relacionarán.

6.6.2 Estructurar en tablas más pequeñas

Ahora se estructurarán las tablas tal y como van a esta en la base datos. A continuación, se muestran los datos que contendrán y una pequeña explicación de porqué existen.

Prompt_info

Esta tabla existe para guardar toda la información que pueden ver los usuarios tanto si han comprado el prompt como si no. Es importante diferenciar entre la información que pueden ver dependiendo de si la han comprado o no, ya que un usuario listo podría obtener con una query a esta tabla el prompt, lo cuál sería contraproducente. Contendrá:

- Prompt_ID <uuid>
- Título <string>
- Descripción <string>
- Categoría <string>
- Generador <string>
- Precio <number>
- Unidades del precio <string>
- Momento de creación <timestamp>

Prompt_data

En contraposición a la anterior, aquí se guardarán los datos a los que solo se podrá acceder si se ha comprado el prompt. Contendrá:

- Prompt ID <uuid>
- Prompt <string>
- Uso del prompt <string>
- Prueba del prompt <string>
- Utilidad del prompt <string>

Como se puede observar, se comparte el prompt ID con la tabla anterior. De esta manera, a través de Joins se podrá relacionar la información. Lo mismo pasa en muchas de las otras tablas.

Prompt_authors

Es útil guardar el autor en otra tabla, para solo tener que acceder a este dato cuando realmente es necesario, por ejemplo, en el momento de la compra o en el momento de acceso a la cuenta del usuario.

- Prompt ID <uuid>
- Author ID <uuid>

El ID del autor es el ID de autenticación que dará Supabase.

Prompt_status

El estado del prompt, sea pendiente, rechazado o aprobado. Queremos que solo los admins puedan modificarla, y por lo tanto en el momento de creación del prompt se tendrá que lanzar una función en la nube para actualizar la tabla y que el usuario no apruebe el prompt sin permiso mandando un payload incorrecto.

- Prompt ID <uuid>
- Status <string>
- Rejected_reason <string>
- Last_updated <timestamp>

Esta tabla se actualizará a la hora de validar el prompt, ya se aprobándolo o rechazándolo.

Ventas

Las ventas tienen que ir separadas en una tabla segura que solo se pueda modificar desde el servidor. De esta manera se asegura que los usuarios maliciosos no obtienen acceso a prompts con un simple INSERT o UPDATE a la base de datos.

- Id <int8>
- Prompt ID <uuid>
- buyer_id <uuid>

De esta manera se controla que usuarios tienen que prompts en una sola tabla. Como los prompts se pueden comprar muchas veces, es necesario otro campo Id.

Stripe

La tabla de stripe contiene los datos de stripe de los usuarios, para poder crear después productos y venderlos si fuese necesario.

- User ID <uuid>
- Email <string>
- Stripe ID <string>

Stripe_products

Y en esta tabla se guardan los diferentes “productos” de Stripe, que son en realidad los prompts que vendemos. Stripe requiere que se tenga un ID para cada producto, y un ID para cada precio.

- Prompt ID <uuid>
- Stripe_product_id <string>
- Stripe_price_id <string>
- Stripe_account_id <string>

Estas serían las tablas mínimas necesarias para hacer funcionar la página, que ya tienen mucho mejor aspecto al haber sido normalizadas. Aun así, se pueden implementar mejoras en algunas de ellas. Además, **falta integrar las imágenes en este esquema**.

6.6.3 Mejoras implementables

Aquí se listan las mejoras que se pueden realizar a las tablas.

Outsourcing de las categorías y generadores

Guardar el string del nombre de las categorías y los generadores directamente en la tabla de prompts es posible, sin embargo, dado el caso de que uno de estos nombres cambiase, se tendría que hacer una refractorización masiva de la base de datos, con el coste que ello conlleva. Para evitar esto, se pueden crear dos tablas aparte, una para cada tipo. En ellas se definirá un ID y un título, y el ID se utilizará para relacionar la tabla de prompt_info con la categoría correspondiente, por lo que en la tabla de prompt_info se guardarán estos ID's en formato número.

Además, estas tablas tendrán también las traducciones en caso de ser necesario.

Tabla de administradores

Tiene sentido tener una tabla de administradores para luego poder ver si el usuario es administrador o no, y por ende si puede validar. La tabla simplemente contendrá el user ID de la autenticación, y servirá para las "security policies" más adelantes.

Imágenes

Como se ha comentado antes, no se han integrado las imágenes en el esquema. Esto es porque las imágenes se guardarán en el **storage**, que no es una tabla sino un bucket para contener archivos.

Para hacer su obtención muy sencilla, lo que se hará es que, para cada prompt existirá una carpeta, el nombre de la cual será el ID del prompt. De esta manera, hacer una query será simplemente buscar esa dirección en el storage. Un ejemplo sería esto:

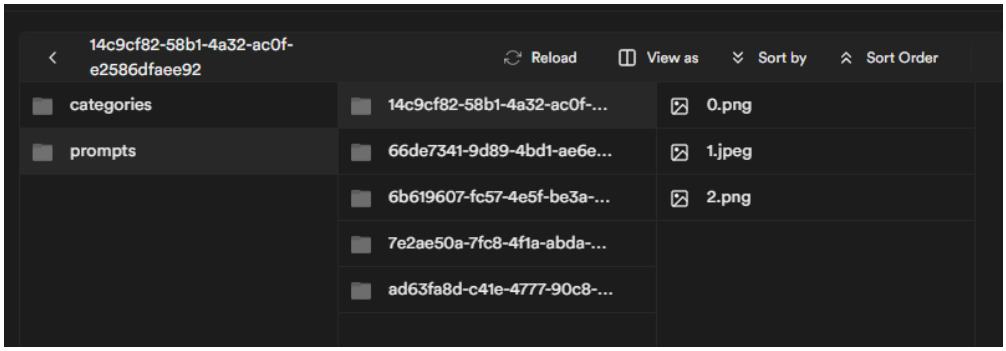


Figura 29. Ejemplo de las imágenes de un prompt en storage.

¿Y qué pasa con las imágenes de portada?

Para no tener que ir a buscar cada vez el storage en el momento de mostrar la card del prompt, se ha decidido guardar el Prompt ID y el path (el URL de la imagen) para tenerlo accesible fácilmente con un Join. Para las imágenes de categoría, lo mismo.

Por último, queda definir los niveles de seguridad de cada tabla, ya que las relaciones se muestran directamente en el esquema final.

6.6.4 Seguridad

La seguridad de las tablas es muy importante para poder protegerse de ataques, así como para poder mantener unas ciertas “reglas” en la página web, como por ejemplo solo poder obtener un prompt si lo has comprado.

Para ello se hace uso de la “Row Level Security” que ofrece Supabase, así como de sus “Policies”, que ofrecen seguridad para cada tipo de operación y nivel de usuario. Aquí se muestra el desglose de seguridad según las tablas:

Administradores

Todo el mundo podrá leer esta tabla para confirmar si es administrador. El resto de operaciones no está permitdo y solo se puede editar desde el dashboard de supabase.

Categorías y Generadores

Todo el mundo podrá leer esta tabla para confirmar si es administrador. El resto de operaciones no está permitdo y solo se puede editar desde el dashboard de supabase.

Imágenes de las categorías

Todo el mundo podrá leer esta tabla para confirmar si es administrador. El resto de operaciones no está permitdo y solo se puede editar desde el dashboard de supabase.

Imágenes de portada de prompts

Solo los usuarios autenticados podrán insertar. Todos los usuarios podrán leer. Solo podrá haber una línea por prompt ID. El resto de operaciones no está permitido.

Prompt_info

Solo los usuarios autenticados podrán insertar. Todos los usuarios podrán leer. Solo podrá haber una línea por prompt ID. El resto de operaciones no está permitido.

Prompt_data

Se podrán leer y editar los prompts siempre que tú seas el creador. Cualquier usuario autenticado podrá insertar un nuevo prompt. Solo se podrán leer datos si tienes el prompt (si en la tabla de ventas está registrada esa venta).

Los admins podrán leer los prompts si eres admin.

Prompt_authors

Solo los usuarios autenticados podrán insertar. Todos los usuarios podrán leer. Solo podrá haber una línea por prompt ID. El resto de operaciones no está permitido.

Prompt_status

Solo los adminstradores podrán insertar o hacer update de la tabla. También podrán leer.

Todos los usuarios podrán leer los prompts que tenga el estado aprobado y las líneas de los prompts que han creado.

Ventas

La tabla de ventas solo podrá ser leída por todos los usuarios autenticados y que posean un prompt en esa tabla. Solo podrán leer las líneas que posean. Ninguna operación más es permitida (se realizará acceso a través de funciones en el servidor).

Stripe

La tabla de datos de stripe solo podrá ser leída por los usuarios autenticados, y solo podrán leer la línea que corresponda a su ID de autenticación. Ninguna operación más es permitida (se realizará acceso a través de funciones en el servidor).

Stripe_products

Esta tabla no permitirá ninguna operación directa (se realizará acceso a través de funciones en el servidor).

Edge Functions

Estas son las funciones en el servidor que se vienen mencionando anteriormente. Son muy útiles, puesto que permiten lanzar funciones sin que el usuario pueda manipularlas antes de mandar, y además permiten guardar secretos dentro del servidor para que los usuarios maliciosos no puedan acceder a ellos de ninguna manera.

Se utilizarán estas funciones para diferentes tareas:

- Creación de cuentas de Stripe, ya que para cualquier operación con Stripe es necesaria la clave secreta.
- Creación de productos y precios de Stripe.
- Compras a través de Stripe.
- Actualización de tablas de datos sensibles de Stripe una vez completado cualquiera de los pasos anteriores.
- Actualización de la tabla de ventas.
- Actualización de la tabla de estado de los prompts.

Con esto, se puede decir que la aplicación es segura de ataques externos y manipulaciones no deseadas.

6.6.5 Esquema final

Una vez detallada la arquitectura, solo queda plasmar esto en un diagrama de bases de datos SQL. En este caso se ha utilizado la herramienta Lucidchart para ello.

El esquema se ve en la página siguiente.

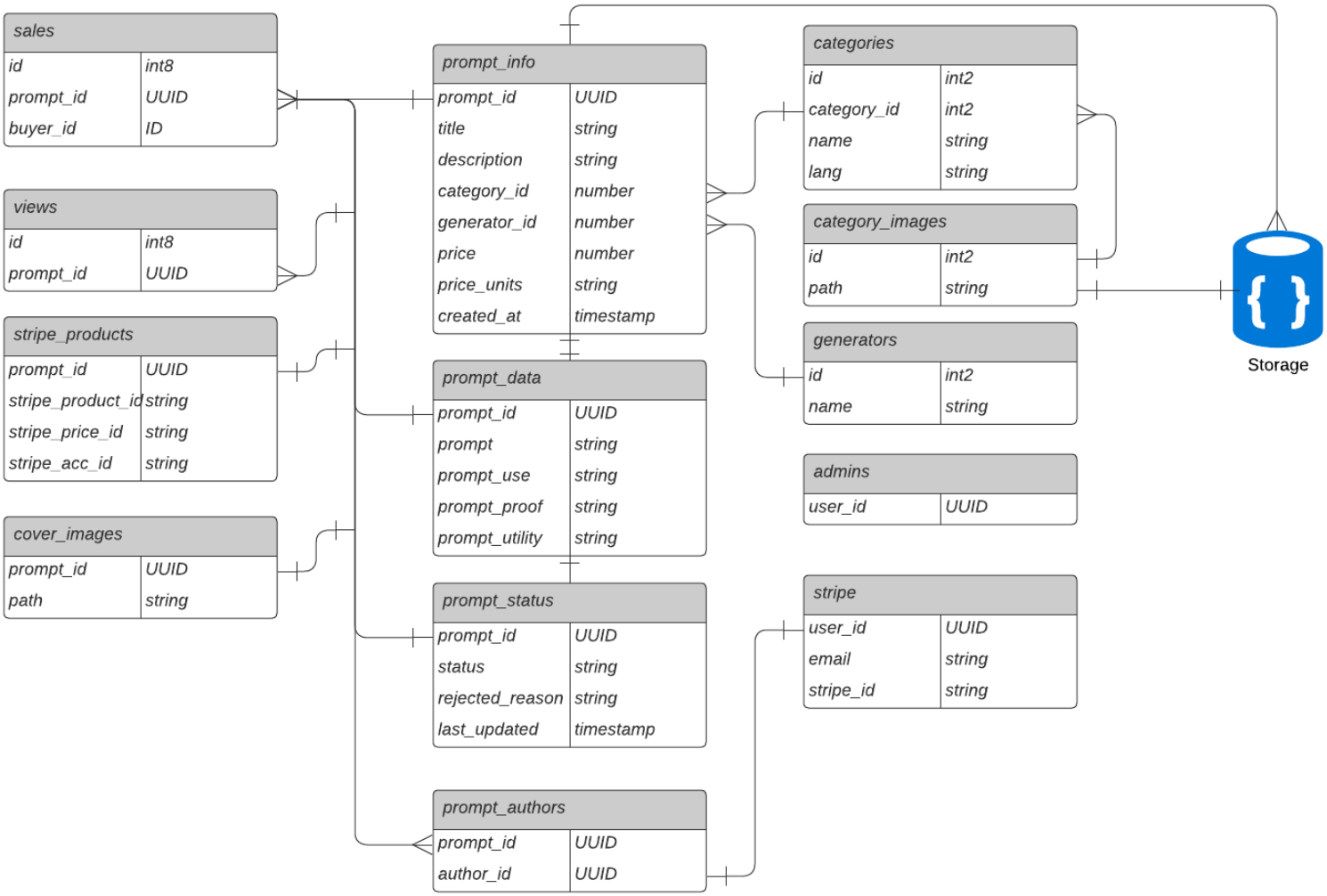


Figura 30. Esquema de la base de datos

Importante notar que la relación entre las tablas prompt_info, prompt_data y prompt_status es 1 to 1 en todas,.

6.6.6 Relación entre la página y los servicios externos

Además de tener relaciones internas en la base de datos, también existen relaciones entre la base de datos y las APIs externas. Constan de las siguientes partes:

- Supabase database: La propia base de datos de Supabase, teniendo en cuenta el servicio de autenticación.
- Supabase Edge Functions: Las funciones en la nube. Es importante hacer esta distinción para clarificar el diagrama ya que las Edge Functions actúan como intermediario en muchas ocasiones.
- Stripe: La entidad de Stripe en su totalidad.
- MailerSend: La entidad de MailerSend en su totalidad.
- Google Workspace: Donde está el email con el dominio comprado.
- Promptmarkt: La propia página web.

Todas ellas se relacionan entre si para conseguir una experiencia completa y fluida para el usuario.

El diagrama se puede ver en la siguiente página:

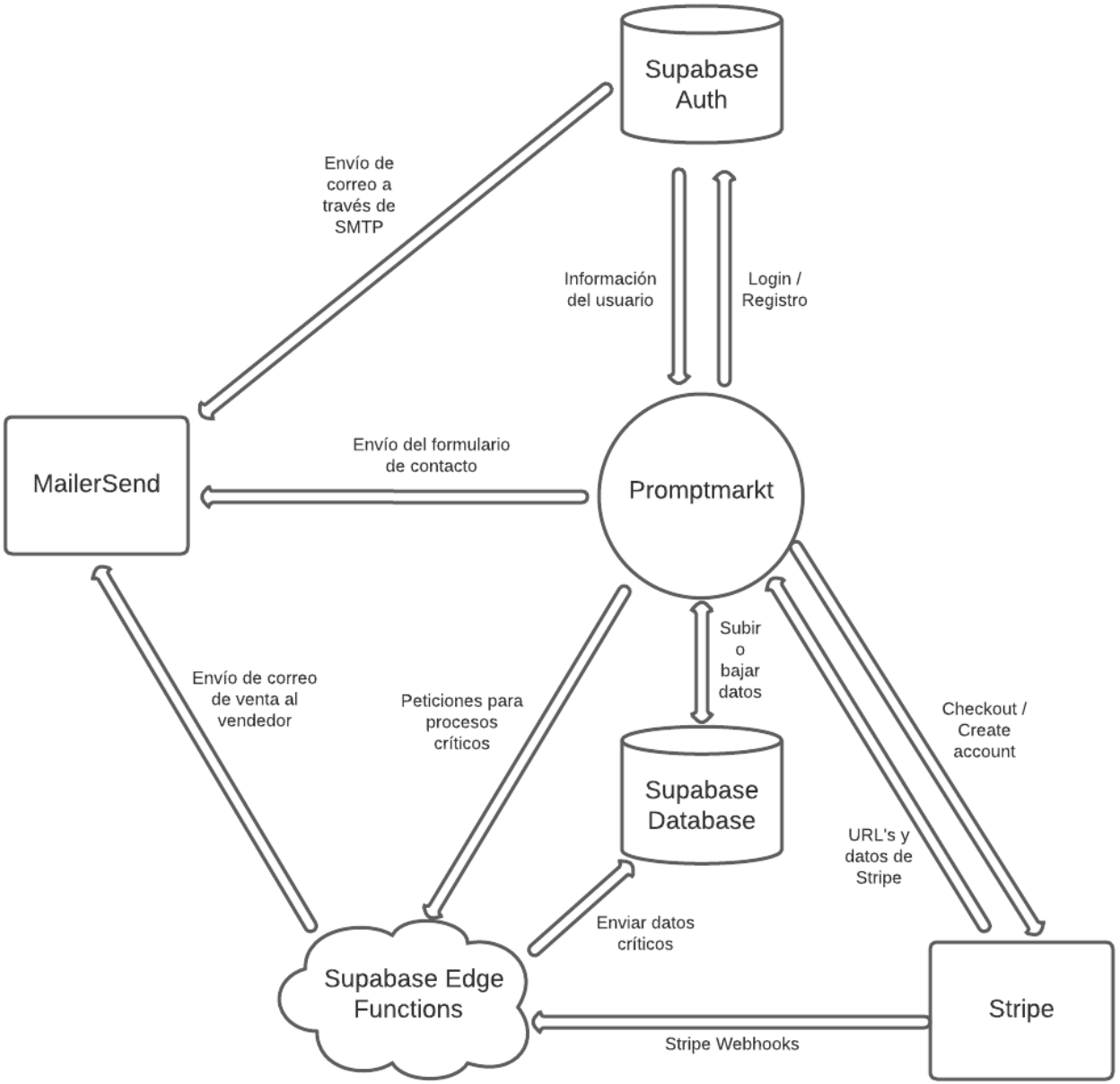


Figura 31. Diagrama de relaciones entre los diferentes servicios y la página

7. Desarrollo del back-end

Para ver las partes más importantes y/o interesantes del código de back-end, se puede ir al Anexo 4 – Extractos de código del Back End.

7.1 Autenticación

La autenticación está completamente gestionada por Supabase, y, además, contiene sus propias tablas por lo que no será necesario crear tablas nuevas para gestionar los datos de los usuarios. Las funciones ya vienen integradas en el paquete de Supabase, por lo que ni siquiera se tienen que crear en un servicio.

Cada usuario obtiene un UUID único que se utiliza en toda la página para validar u obtener resultados de la base de datos.

Además, tiene un set grande de distintos proveedores de autenticación (Google, Apple, Github…), con el que es muy fácil ampliar el rango de posibilidades existente.

Finalmente, ofrece la opción de mandar un correo de verificación desde un email personalizado. En este proyecto el email se enviará desde la dirección de Google Workspaces: promptmarkt@promptmarkt.com.

7.2 Servicios

Lo esencial para desarrollar el back-end es tener la posibilidad de realizar una conexión entre el front y el back. Para ello se utilizará un “service” o servicio, que es un archivo donde se guardan todas las posibles conexiones para poder reutilizarlas en todo el código. De esta manera, no se tienen que escribir las queries de nuevo en cada archivo de la página. Esta es una práctica estándar.

En el caso de este proyecto, existen solamente dos servicios, el de las tablas y el del storage, ambos de conexión a Supabase.

Para poder hacer la conexión a Supabase, se tiene que inicializar primero la conexión en un archivo supabase.js:

```
import { createClient } from "@supabase/supabase-js";

const supabaseUrl = process.env.REACT_APP_SUPABASE_URL;
const supabaseKey = process.env.REACT_APP_SUPABASE_ANON_KEY;

const supabase = createClient(supabaseUrl, supabaseKey, {
  autoRefreshToken: true,
  persistSession: true,
  detectSessionInUrl: false
});

export { supabase };
```

Figura 32. Código de conexión con supabase

Este archivo hace uso de la llave “ANON” y de la URL de la base de datos. Ambos valores están en el archivo “.env”, pero en producción se guardarán en el servicio de variables de entorno de Netlify para mayor seguridad. Estas claves, aun así, no son tan críticas como otras claves secretas de Supabase.

Una vez hecha esta conexión, ya se pueden hacer operaciones en la base de datos de Supabase. Un ejemplo muy sencillo de operación sería esta query donde se hace un “select” de la información de un prompt según su ID:

```
export const getPromptInfoByPromptId = async (promptId: string) => {
  const { data, error } = await supabase
    .from("prompt_info")
    .select("*")
    .eq("prompt_id", promptId)
    .single();

  return { data, error };
};
```

Figura 33. Ejemplo de query en el servicio de base de datos.

7.3 Edge Functions de Supabase (funciones en el servidor)

Las “Edge Functions” de Supabase son funciones que se lanzan de manera segura en el lado del servidor, y pueden servir para una funcionalidad que requiera un paso de seguridad extra o en la que no podemos dejar que el usuario manipule el payload enviado al servidor (por ejemplo, la gestión de un pago), o para gestionar eventos que tienen una repercusión (por ejemplo, se hace una venta y el vendedor debe recibir un correo conforme se ha hecho).

En el caso de este proyecto, las Edge Functions se utilizarán para:

- Gestionar la conexión con Stripe, ya que requiere de claves secretas que no pueden aparecer en front-end.
- Crear un punto de conexión para el webhook de Stripe que se lanza una vez completada una compra.
- Actualizar la tabla de ventas, ya que no se quiere permitir que los usuarios modifiquen libremente esa tabla.

Un ejemplo de función muy sencilla, es este:

```
import { serve } from "https://deno.land/std@0.131.0/http/server.ts";
import Stripe from "https://esm.sh/stripe@11.0.0?target=deno&no-check";

const stripe = Stripe(Deno.env.get("REACT_APP_STRIPE_TEST_SECRET_KEY") ?? "", {
  // This is needed to use the Fetch API rather than relying on the Node http
  // package.
  httpClient: Stripe.createFetchHttpClient()
});

const corsHeaders = {
  "Access-Control-Allow-Origin": "*",
  "Access-Control-Allow-Headers": "authorization, x-client-info, apikey"
};

serve(async (req) => {
  if (req.method === "OPTIONS") {
    return new Response("ok", {
      headers: corsHeaders
    });
  }

  const { stripeId } = await req.json();

  try {
    return await stripe.accounts.retrieve(stripeId).then((result: any) => {
      console.log("Stripe account retrieved result: ", result);
      return new Response(JSON.stringify(result), {
        headers: corsHeaders,
        status: 200
      });
    });
  } catch (error) {
    console.error("Error on checking stripe OB: ", error);
    return new Response(JSON.stringify(error), {
      headers: corsHeaders,
      status: 400
    });
  }
});
```

Figura 34. Edge Function para comprobar el estado de Stripe

En este caso simplemente se comprueba el estado de una cuenta de Stripe (se ha creado, está en proceso, etc.). Para ello, se utiliza el paquete de Stripe de Deno, y se hace una llamada a la llave secreta (en este ejemplo, de testeo) que esta guardada en Supabase, segura de ataques externos.

Con un stripeId que se le pasa a la función, se retorna un resultado u otro, y además en el Dashboard de Supabase se muestran los logs que se hayan ido introduciendo a lo largo del proceso, para poder revisar si hay errores:

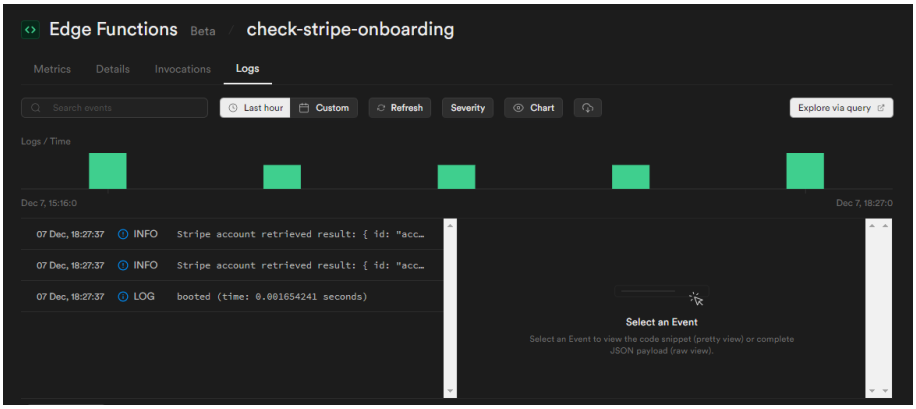


Figura 35. Dashboard de Supabase de Edge Functions

7.4 Stripe

Stripe es el servicio seleccionado para la gestión de pagos, y cuenta con un sistema automatizado muy completo y fácil de configurar. Aquí se resumen las partes más importantes para el proyecto, y se pueden ver ejemplos de algunos en el Anexo 4.

7.4.1 Connect

Sirve para poder conectar cuentas de usuarios externos que quieren vender en la plataforma. De esta manera, los pagos se gestionan de manera automática y cada parte interesada (Stripe, el usuario y promptmarkt) recibe su parte.

Las cuentas conectadas son de tipo “Express”, lo que implica un setup más rápido de la cuenta para el usuario.

7.4.2 Prices y Products

Todo funciona a través de productos que tienen un precio asociado. Ambos son objetos que contienen diferentes datos interesantes y a los cuales se debe hacer referencia para el momento de compra.

7.4.3 Checkout

El sistema de checkout permite que las compras sean seguras, ya que se gestionan directamente en Stripe en vez de tener que gestionarlas en la propia página web.

Existe también la posibilidad de gestionar uno mismo el checkout, pero se complica el código y la seguridad de la compra.

Además, la interfaz ya viene diseñada:

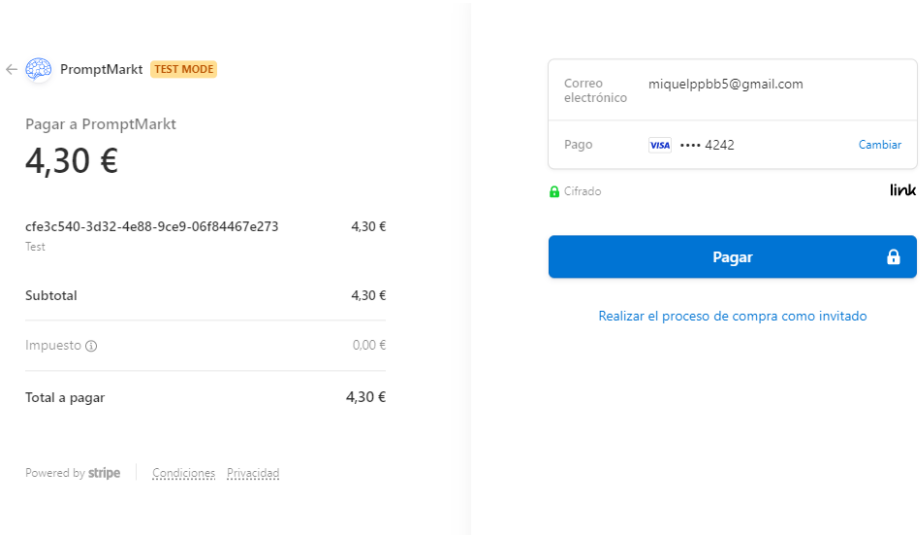


Figura 36. Ejemplo de prueba de Stripe Checkout

7.4.4 Webhooks

El “problema” de checkout es que, al no ejecutarse en la página web propia sino en Stripe, no es posible saber cuándo se ha ejecutado un pago correctamente o erróneamente, y no se pueden ejecutar las acciones de actualización de la base de datos.
Este problema es fácilmente resoluble con un webhook. Para configurarlo en stripe, simplemente se tiene que insertar la URL del webhook (que en el caso de esta página, es una Edge Function de Supabase), para que en el caso de que se dé un evento de los establecidos (por ejemplo, pago exitoso), se lance cierto código:

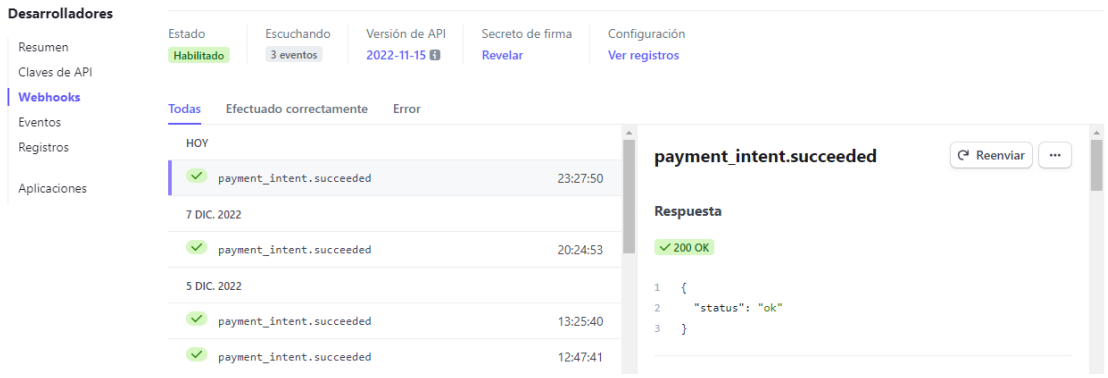


Figura 37. Webhooks de Stripe

7.5.5 Tax Automation

Aunque no se ha necesitado código, es importante mencionar que este servicio sirve para poder sacar las declaraciones de impuestos de manera rápida en el momento de presentación de declaraciones.

7.6 MailerSend

Para enviar correos transaccionales desde MailerSend, lo único que hay que hacer es enviar (desde el servidor, con un Edge Function) ciertos detalles del correo a un endpoint, nada más. El código es este:

```
return await fetch("https://api.mailersend.com/v1/email", {
  method: "POST",
  headers: {
    ...corsHeaders,
    "Content-Type": "application/json",
    Authorization: `Bearer ${MAILERSEND_KEY}`
  },
  body: JSON.stringify({
    from: {
      email: "promptmarkt@promptmarkt.com"
    },
    subject: subject,
    text: body,
    to: [
      {
        email: receiver
      }
    ]
  })
```

Figura 38. Código para enviar un email desde Mailersend

Las respuestas recibidas son diferentes códigos de estado de tipo 2XX, 4XX o 5XX, siendo los primeros los de éxito y los otros de fallo. En las Edge Functions se loguea la respuesta para poder debuguear en caso de error.

8. Seguridad

Ya se ha mencionado anteriormente la importancia de la seguridad cuando se trabaja con datos en el servidor, ya que existen muchas maneras de acceder a una base de datos si no se ha configurado bien sus políticas de acceso.

Lo mismo pasa con el uso de APIs, cualquier persona que tenga la clave secreta de tu API puede utilizar el servicio a tu costa, o peor, puede hacer uso malicioso de ella para bloquearte la página u obtener datos de los usuarios. Es algo que se debe evitar a toda costa.

Este apartado se dedica a ver en detalle la seguridad de la página web desarrollada.

8.1 Políticas de seguridad de Supabase

Antes, en el apartado 6.6.3, se han mencionado las políticas de seguridad de las diferentes tablas de la base de datos. Es en este punto donde se aplican estas políticas. Supabase da la posibilidad de aplicarlas de manera muy sencilla y con poco código, a través de sus “Policies”, que significa políticas en inglés.

Por ejemplo, si se quieren que todos los usuarios tengan acceso de lectura a una cierta tabla, el proceso es tan sencillo como rellenar este formulario a través del Dashboard:

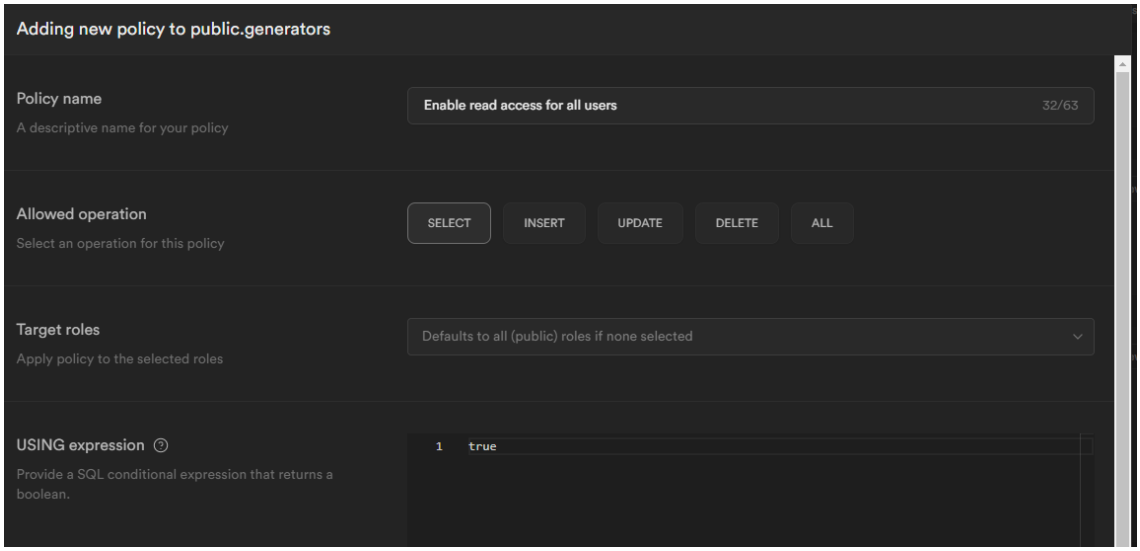


Figura 39. Supabase Policy de base de datos

Esta política es muy sencilla, pero hay algunas más complicadas que se pueden ver en el Anexo 4.

Es importante notar que estas políticas solo aplican si en la tabla en cuestión está aplicado el “Row-Level Security”, que protege cada fila con estas políticas. De lo contrario, la tabla está expuesta a todo el mundo.

Si, por el contrario, está activado el “Row-Level Security” pero no existen políticas, esa tabla será inaccessible, es decir, que son necesarias siempre, aunque sea una política de lectura simple como la anterior.

Como nota final, es interesante ver que las políticas se pueden dividir según el tipo de operación, y también pueden aplicar al storage donde se guardan las imágenes. Las políticas del storage son un poco más complejas a nivel de sintaxis, por ejemplo, esta política que se muestra a continuación permite que acceso de subida a los usuarios autenticados en la carpeta “prompts”, a que no se quiere que puedan subir imágenes en la carpeta “categories”:

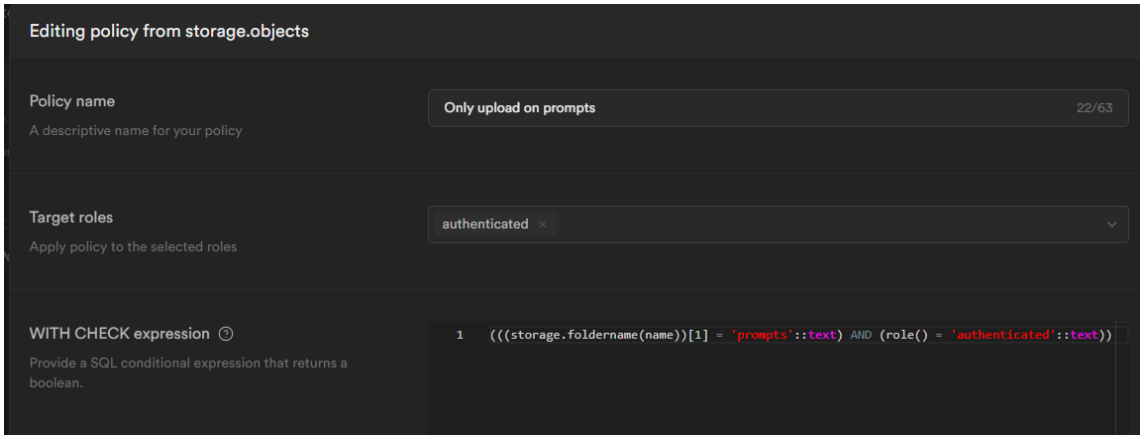


Figura 40. Supabase Policy de storage

8.2 Claves API

Como se ha mencionado antes, tener seguras las claves de API es esencial para asegurar que ningún usuario malicioso utiliza nuestro servicio gratuitamente, o modifica datos sin permiso.

Para ello, es de vital importancia seguir estas directrices:

1. Nunca subir las claves al repositorio de Github. Una buena práctica es tenerlas en un archivo .env, y este archivo añadirlo al gitignore.
2. Nunca tener las claves en un archivo local que después vamos a mandar al servidor. Para esto existen las funciones en el servidor, que permiten ejecutar código en el servidor sin necesidad de comprometer las claves. En este proyecto son las “Edge Functions” de Supabase.
3. Y, en relación al punto anterior, es siempre bueno tener las claves en el servidor, encriptadas. En nuestro caso, Supabase tiene un gestor de claves que se relaciona directamente con las “Edge Functions”.

Hay un punto importante a comentar, y es que Supabase, para incializarse, requiere de una key que no es secreta, pero mejor mantenerla fuera de los ojos de los usuarios. Para este problema, desde Netlify (el host de la página) se permiten guardar variables de entorno de manera más segura, con unos pocos clicks, y así poder acceder a ellas en el momento de entrar a la página.

9. Otros elementos

Después de realizar el groso de la página, quedan algunos elementos que deben ser tenidos en cuenta:

9.1 Traducciones

Aunque los prompts serán en su mayoría en inglés y no se traducirán, la página web existirá también en español para poder llegar al máximo de usuarios posibles. Esto quiere decir que será necesario establecer un sistema de traducciones para los textos estáticos dentro de la página.

Para ello, el paquete estándar es **react-i18next**, que permite gestionar las traducciones de manera muy sencilla, entre otros. Para ello, simplemente se tiene que instalar el paquete e inicializarlo:

```
import i18next from "i18next";
import { initReactI18next } from "react-i18next/initReactI18next";
import Backend from "i18next-http-backend";
import LanguageDetector from "i18next-browser-languagedetector";

i18next
  .use(LanguageDetector)
  .use(Backend)
  .use(initReactI18next)
  .init({
    interpolation: { escapeValue: false },
    // DEBUG MODE ONLY
    // lng: "en",
    fallbackLng: "en"
  });

export default i18next;
```

Figura 41. Inicialización i18next

En este caso particular, se quiere poder detectar el lenguaje automáticamente para luego retornar los valores de las traducciones que residen en la carpeta “public” del proyecto. Como solo hay español o inglés, si se recibe cualquier otro lenguaje va directo al inglés.

Los archivos de traducción son en formato JSON, y hay uno por cada lenguaje. Por ejemplo:

titles.json

public > locales > en > titles.json > ...

```
{
  "app": "Buy and sell AI prompts with real world applications",
  "home": "Buy and sell AI prompts with real world applications",
  "contact": "Contact PromptMarkt",
  "explore": "Find prompts on PromptMarkt",
  "sell": "Sell your prompts on PromptMarkt"
}
```

Figura 42. Ejemplo de JSON de traducción

Finalmente, para poder traducir, se hace uso del hook “useTranslation”, de esta manera:

```
const { t } = useTranslation(["home", "common", "titles"]);
```

Figura 43. Hook useTranslation

La parte de dentro del array indica de que JSONs queremos coger las traducciones. Para poner una traducción en la página simplemente hacemos esto:

```
<h1 className="text-2xl 2xl:text-3xl 2xl:mb-4 font-bold">
  {t("title", { ns: "home" })}
</h1>
```

Figura 44. Insertar una traducción de i18next

9.2 Términos y Condiciones

Para los términos y condiciones se ha utilizado el generador https://www.termsandconditionsgenerator.com/ que permite generar tus términos y condiciones según tus necesidades.

Además, se han adaptado al servicio que damos en la web, haciendo hincapié en que los prompts no son reembolsables y PromptMarkt se reserva en su totalidad el derecho de rechazarlos o eliminarlos.

10. Instrucciones de instalación

Aunque el proyecto no está terminado, ya se puede realizar tanto una instalación local del proyecto, como verlo en cualquier navegador o dispositivo.

Instalación local

**** Atención **** La instalación local ya no funcionará correctamente, debido a que son necesarias las diferentes claves de Supabase, Stripe y MailerSend para hacerla funcionar, y no es una práctica recomendada el ir distribuyéndolas a diferentes usuarios desconocidos.

Es por ello que se anima al lector a que utilice el punto de acceso por navegador.

Si aun así se quiere probar localmente, para poder instalar el proyecto es necesario tener node.js y npm instalados en nuestro ordenador.

Los pasos a seguir son los siguientes:

1. Extraer el archivo zip llamado “PAC_FINAL_código_PlanaBallber_Miquel.zip” en cualquier carpeta.
2. Navegar por terminal a dicha carpeta hasta estar en el root del proyecto. No es necesario entrar en ninguna subcarpeta.
3. Ejecutar **npm install** para instalar las dependencias.
4. Una vez instaladas, ejecutar el comando **npm start**, el cual abrirá automáticamente la página localhost en nuestro navegador, y se podrá ver la página.

Acceso por navegador

Si lo único que se quiere es ver la página web, se puede acceder a través del enlace www.promptmarkt.com.

11. Conclusiones y trabajos futuros

Una vez finalizado el proyecto, se plasman aquí sus conclusiones, así como los pasos futuros.

11.1 Conclusiones

Durante el desarrollo de este proyecto se ha conseguido un MVP listo para testear con usuarios reales, con las funcionalidades mínimas necesarias para ello.

Este MVP consta de una página web con el dominio www.promptmarkt.com, en la cual se puede, a líneas generales:

- Crear una cuenta en la página, y modificar sus datos.
- Subir prompts para venderlos.
- Comprar prompts.
- Buscar y ver prompts.
- Enviar emails de contacto.

El tech stack que utiliza es:

- React para el desarrollo.
- Tailwind CSS para el estilo.
- Netlify como host de la página web.
- Google Domains para el dominio.
- Supabase para la gestión del back-end. Supabase utiliza PostgreSQL para la base de datos. También se utilizan el servicio de Storage y Edge Functions.
- Stripe para la gestión de pagos.
- MailerSend para el envío de emails.

Se considera que el resultado obtenido es satisfactorio, ya que se ha podido completar el proyecto a tiempo siguiendo el Gantt planteado al inicio de este. Además, la página web, aun siendo un MVP, es perfectamente utilizable por cualquier usuario.

En cuanto al proyecto en sí se refiere, se considera un éxito puesto que el alumno ha adquirido conocimientos nuevos y esenciales en el mundo del desarrollo web, siendo estos:

- Uso de bases de datos SQL.
- Hostings de páginas web.
- Uso de APIs externas.
- Seguridad en páginas web.
- Y el propio desarrollo de una página web.

A modo personal, ha sido un reto en el que he trabajado mucho, pero con el que he disfrutado aprendiendo.

11.2 Pasos futuros

Se listan aquí los pasos futuros a realizar una vez terminado el proyecto.

A corto plazo:

- Testeo del MVP con usuarios reales.
- Mejoras de rendimiento para la escalabilidad de la página, en concreto tomar acciones para reducir el número de peticiones que se hacen al servidor y así reducir su coste.
- Implementación de un nuevo workflow de venta mucho más estilizado y claro, así como de la venta de prompts de GPT-3 y ChatGPT como añadidos a los genaradores de imagen. Esto es esencial para poder vender prompts de calidad y hacer que añadir un nuevo generador a la plataforma sea una tarea fácil.
- Editar un prompt rechazado. Actualmente si un prompt es rechazado se tiene que volver a enviar uno nuevo. Este proceso no es bonito para el vendedor, y sería más fácil si se pudiese editar.
- Mejoras de seguridad. Hay dos puntos identificados pendientes antes de poder lanzar a cliente el producto: Validar que el usuario que lanza ciertas Edge Functions sea administrador, y eliminar la propiedad de ser administrador de Redux para que no pueda ser manipulada.

Una vez cerrados estos pasos a corto plazo, se lanzaría el MMP o Minimum Marketable Product, un producto que ya puede ser utilizado por el usuario final. Una vez finalizado el MMP, los pasos son:

- Iniciar una campaña de marketing a través de Google Ads, y tal vez otras redes sociales.
- Mejoras sustanciales de la UI de la página para hacerla más atractiva.
- Mejora de las templates de los emails enviados.
- Mover a NextJS la página, ya que proporciona varias ventajas que una simple React App no tiene.
- Implementar otros tipos de login, como Google.
- Poder vender en otras divisas como dólares.

Y, a largo plazo, las metas de la página son:

- Poder hacer entender a un usuario sin conocimientos lo que es un prompt, y porqué es útil para él.
- Ser un marketplace de referencia de prompts que tienen aplicación en el mundo real.
- Tener un producto estilizado y útil que resulte agradable para el usuario.

12. Glosario

Prompt: Frase en lenguaje natural utilizada para indicar a una inteligencia artificial qué debe generar.

Prompt engineer: Especialista en crear prompts que generen los mejores resultados.

DALL-E, Midjourney, Stable Diffusion: Inteligencias artificiales de generación de imágenes.

GPT-3: Inteligencia artificial de generación de texto.

Early adopters: Usuarios que adoptan un producto o servicio en la primera fase de su ciclo de vida.

Minimum Viable Product: Versión mínima de un producto que presenta sus funcionalidades más básicas. Funciona como prueba para validar una idea o servicio en el mercado sin gastar muchos recursos.

Look-and-feel: Se dice del aspecto estético que tiene una página.

Landing page: Página donde aterriza el usuario nada más entrar en la página web.

Call-to-action: Un elemento que llama al usuario a que realice una acción, por ejemplo, un botón que diga “Guardar”.

Debuguear: Resolver incidencias de la página web.

13. Bibliografía

[1] «Edge Functions | Supabase». https://supabase.com/docs/guides/functions (accedido nov. 16, 2022).

[2] «Supabase Authentication and create Stripe customer - Subscription with Supabase and Stripe Billing | Part 1 | Sandro Maglione». https://www.sandromaglione.com/techblog/supabase-auth-create-stripe-customer-subscription-supabase-stripe-billing-part-1 (accedido nov. 16, 2022).

[3] «Collect payments then pay out | Documentación de Stripe». https://stripe.com/docs/connect/collect-then-transfer-guide (accedido nov. 11, 2022).

[4] «Cannot find namespace Context error in React (TypeScript) | bobbyhadz». https://bobbyhadz.com/blog/react-cannot-find-namespace-context (accedido nov. 07, 2022).

[5] «10 Minute Tutorial - Full Stack GitHub Authentication with Supabase & React - DEV Community 👨‍💻👩‍💻». https://dev.to/dabit3/10-minute-tutorial-full-stack-github-authentication-with-supabase-react-3c6b (accedido nov. 07, 2022).

[6] «Authentication with Supabase and React». https://ruanmartinelli.com/posts/supabase-authentication-react (accedido nov. 07, 2022).

[7] «sql server - Restrict update on certain columns. Only allow stored procedure to update those columns - Database Administrators Stack Exchange». https://dba.stackexchange.com/questions/24734/restrict-update-on-certain-columns-only-allow-stored-procedure-to-update-those (accedido nov. 06, 2022).

[8] «Allow users to modify only some (but not all) fields in a PostgreSQL table with row level security - Database Administrators Stack Exchange». https://dba.stackexchange.com/questions/298931/allow-users-to-modify-only-some-but-not-all-fields-in-a-postgresql-table-with (accedido nov. 06, 2022).

[9] «Creating multiple tables and table relationships». https://launchschool.com/books/sql/read/table_relationships (accedido nov. 06, 2022).

[10] «How to choose between SQL and NoSQL databases - Simple Talk». https://www.red-gate.com/simple-talk/databases/nosql/how-to-choose-between-sql-and-nosql-databases/ (accedido nov. 06, 2022).

[11] Shopify blog, «¿Tengo que hacerme autónomo si tengo una tienda online?», ene. 11, 2022. https://www.shopify.com/es-es/blog/113361733-tengo-que-hacerme-autonomo-si-tengo-una-tienda-online (accedido nov. 01, 2022).

[12] «Understanding Early Adopters and Customer Adoption Patterns | IxDF». https://www.interaction-

design.org/literature/article/understanding-early-adopters-and-customer-adoption-patterns (accedido oct. 31, 2022).

[13] «Blue Crystal Cube - Free image on Pixabay». https://pixabay.com/illustrations/blue-crystal-cube-deep-futuristic-5457731/ (accedido oct. 19, 2022).

[14] «Macbook Vector Pro - Free image on Pixabay». https://pixabay.com/illustrations/macbook-vector-macbook-pro-4515471/ (accedido oct. 19, 2022).

[15] P. Ekene Eze, «How to use custom fonts in Tailwind CSS», *https://blog.logrocket.com/how-to-use-custom-fonts-tailwind-css/*, feb. 02, 2022.

[16] M. J. Gonçalves, «Angular vs React vs Vue - Blog de Hiberus Tecnología», nov. 24, 2021. https://www.hiberus.com/crecemos-contigo/angular-vs-react-vs-vue/ (accedido oct. 12, 2022).

[17] S. Daityari, «Angular vs React vs Vue: Which Framework to Choose in 2022», ago. 17, 2022. https://www.codeinwp.com/blog/angular-vs-vue-vs-react/ (accedido oct. 12, 2022).

[18] Y. Tápanes, «Los 6 mejores framework frontend | Saasradar», jun. 18, 2022. https://saasradar.net/mejores-framework-frontend/ (accedido oct. 12, 2022).

[19] A. Pattakos, «Angular vs React vs Vue: Which Framework Is Better? 2022», ene. 04, 2022. https://athemes.com/guides/angular-vs-react-vs-vue/ (accedido oct. 12, 2022).

[20] N. Young, «The Pros and Cons of Tailwind CSS | Webdesigner Depot Webdesigner Depot » Blog Archive», sep. 24, 2021. https://www.webdesignerdepot.com/2021/09/the-pros-and-cons-of-tailwind-css/ (accedido oct. 12, 2022).

[21] O. Ekwuno, «Comparing Tailwind CSS to Bootstrap: Is it time to ditch UI kits? - LogRocket Blog», sep. 22, 2022. https://blog.logrocket.com/comparing-tailwind-css-bootstrap-time-ditch-ui-kits/ (accedido oct. 12, 2022).

[22] Andrei Cioara, «Always start with the front end», *https://andreicioara.com/always-start-with-the-front-end-768dff8e961e*, dic. 2018.

[23] A. Vaswani *et al.*, «Attention Is All You Need».

Anexo 1 – Prototipo de baja fidelidad

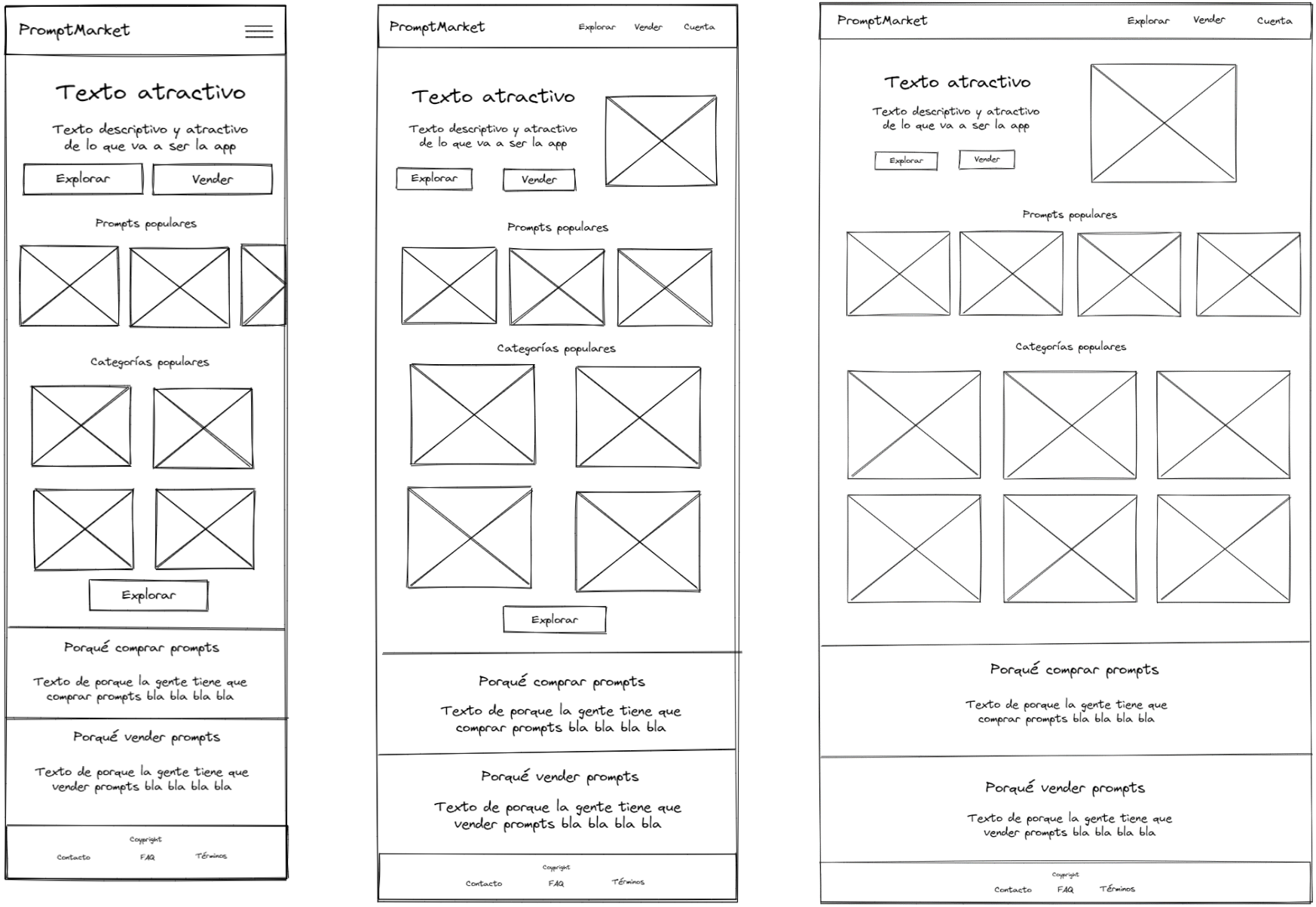


Figura 45. Prototipo de baja fidelidad de la página principal. De izquierda a derecha: Móvil, tablet y ordenador.

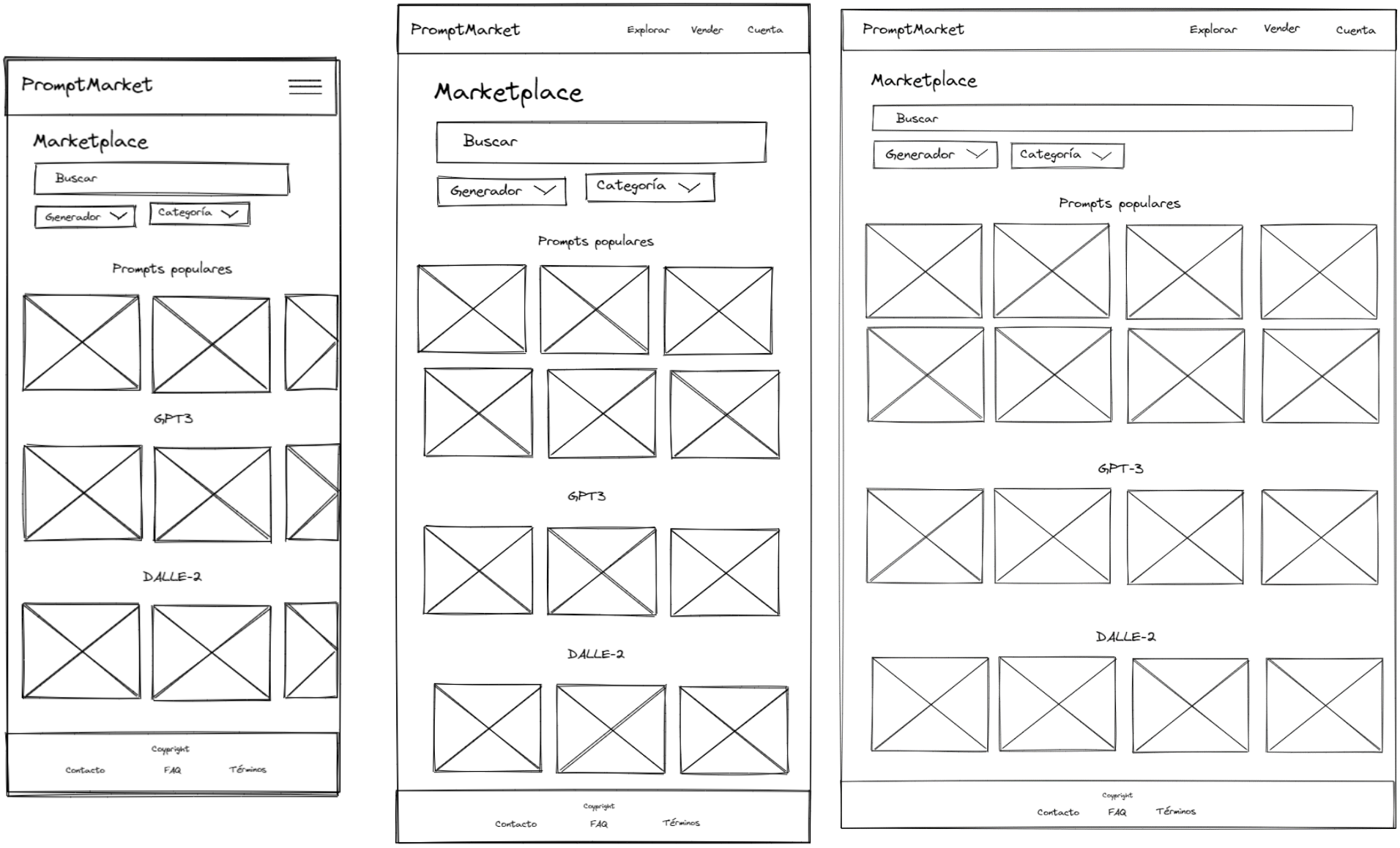


Figura 46. Prototipo de baja fidelidad de la página explora. De izquierda a derecha: Móvil, tablet y ordenador.

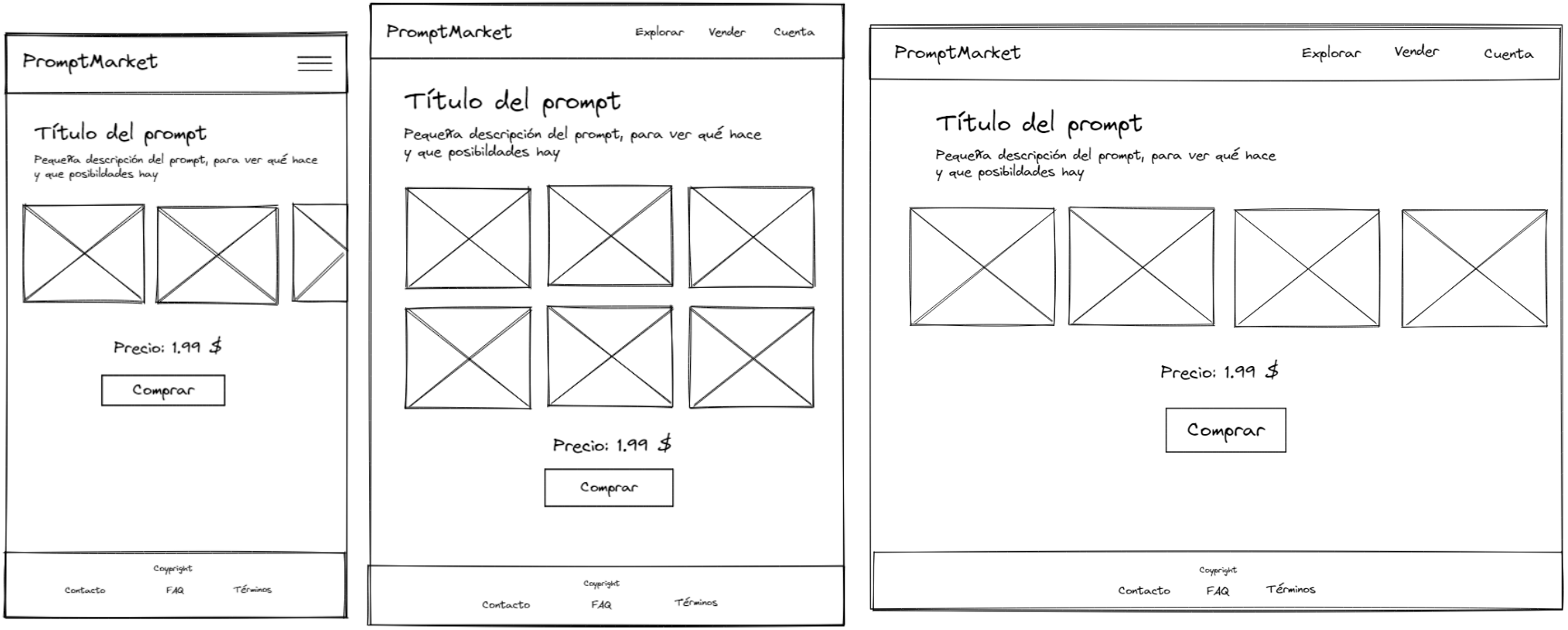


Figura 47. Prototipo de baja fidelidad de la página de compra. De izquierda a derecha: Móvil, tablet y ordenador.

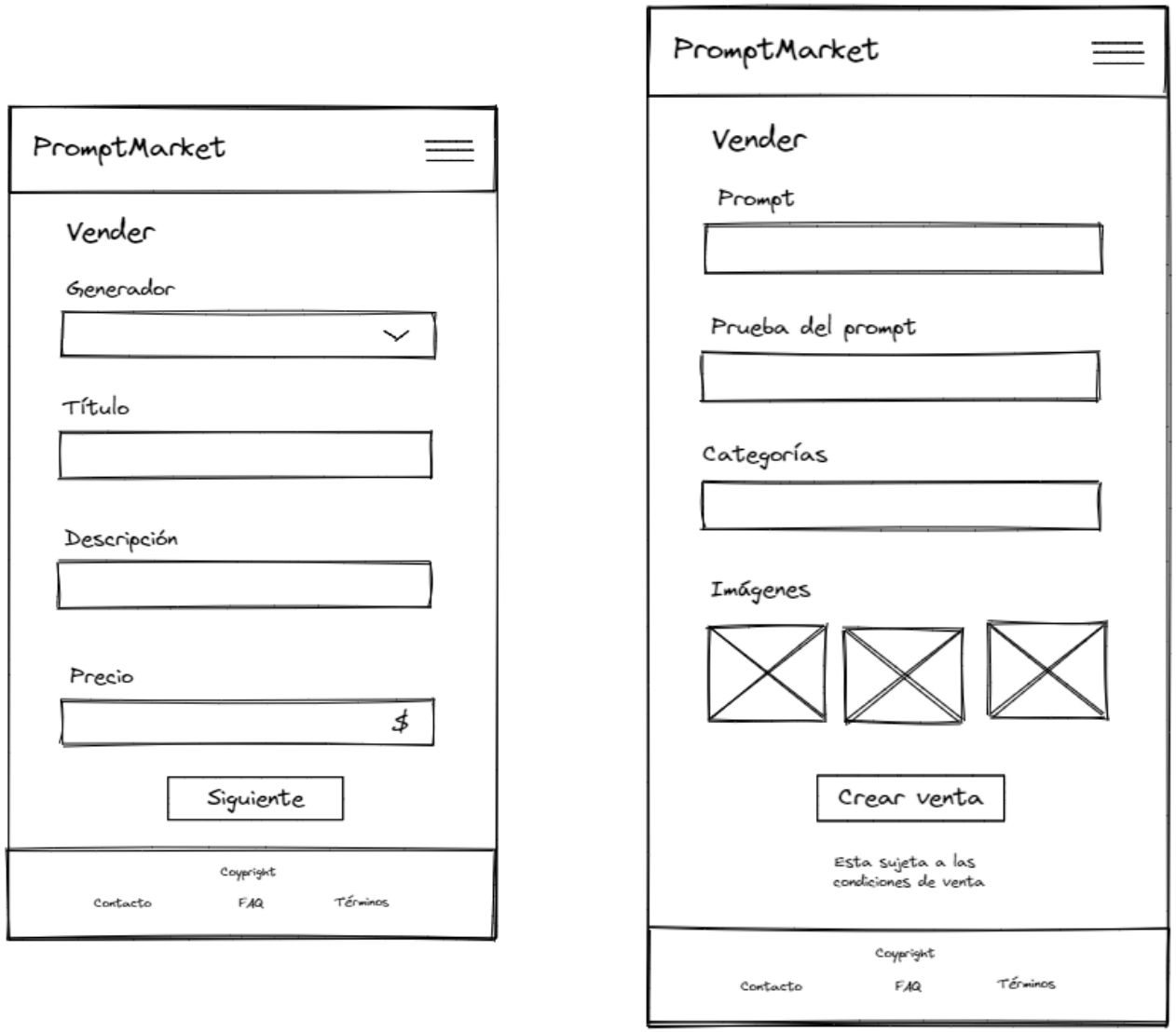


Figura 48. Prototipo de baja fidelidad de la página de venta de móvil.

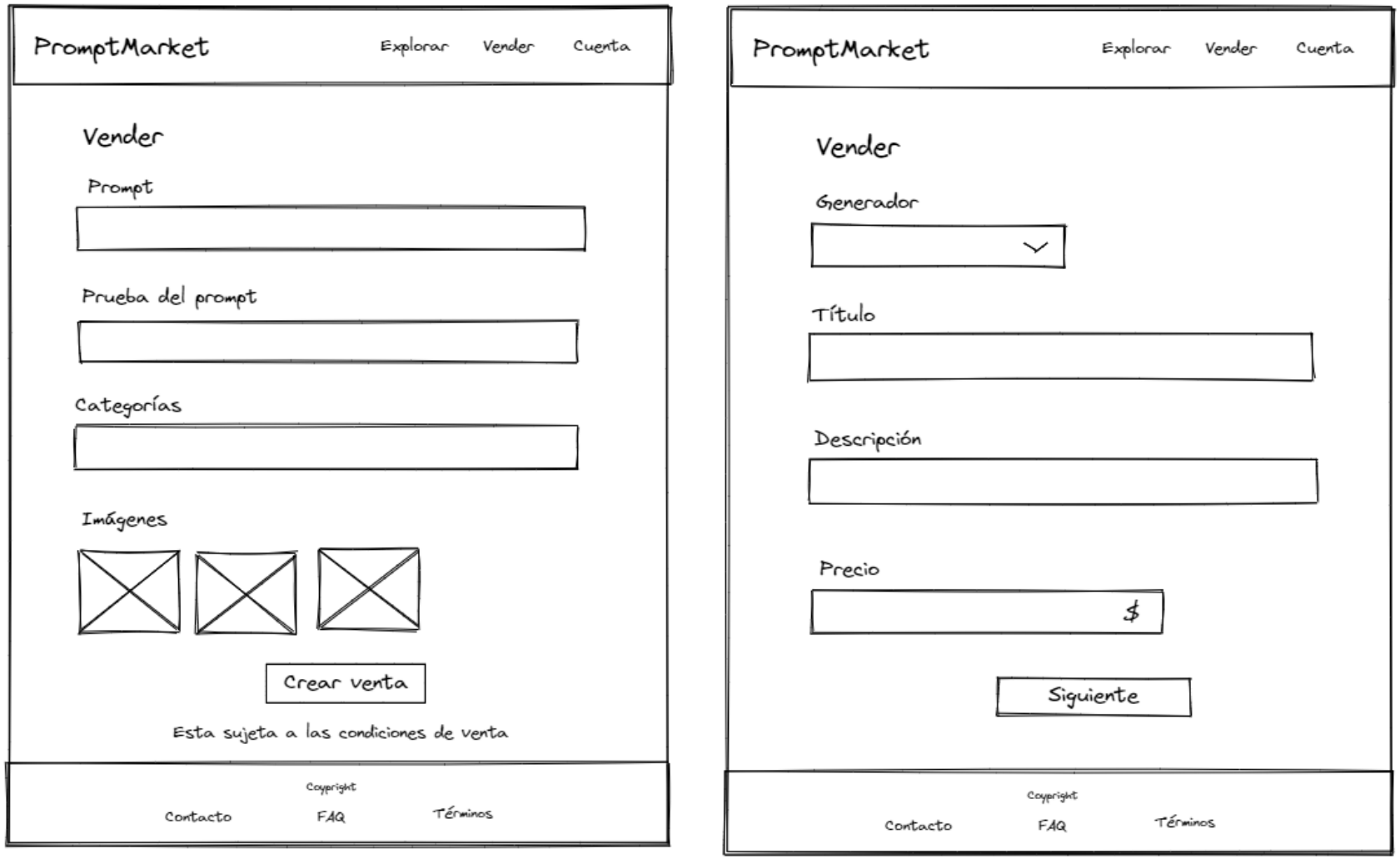


Figura 49. Prototipo de baja fidelidad de la página de venta de tablet.

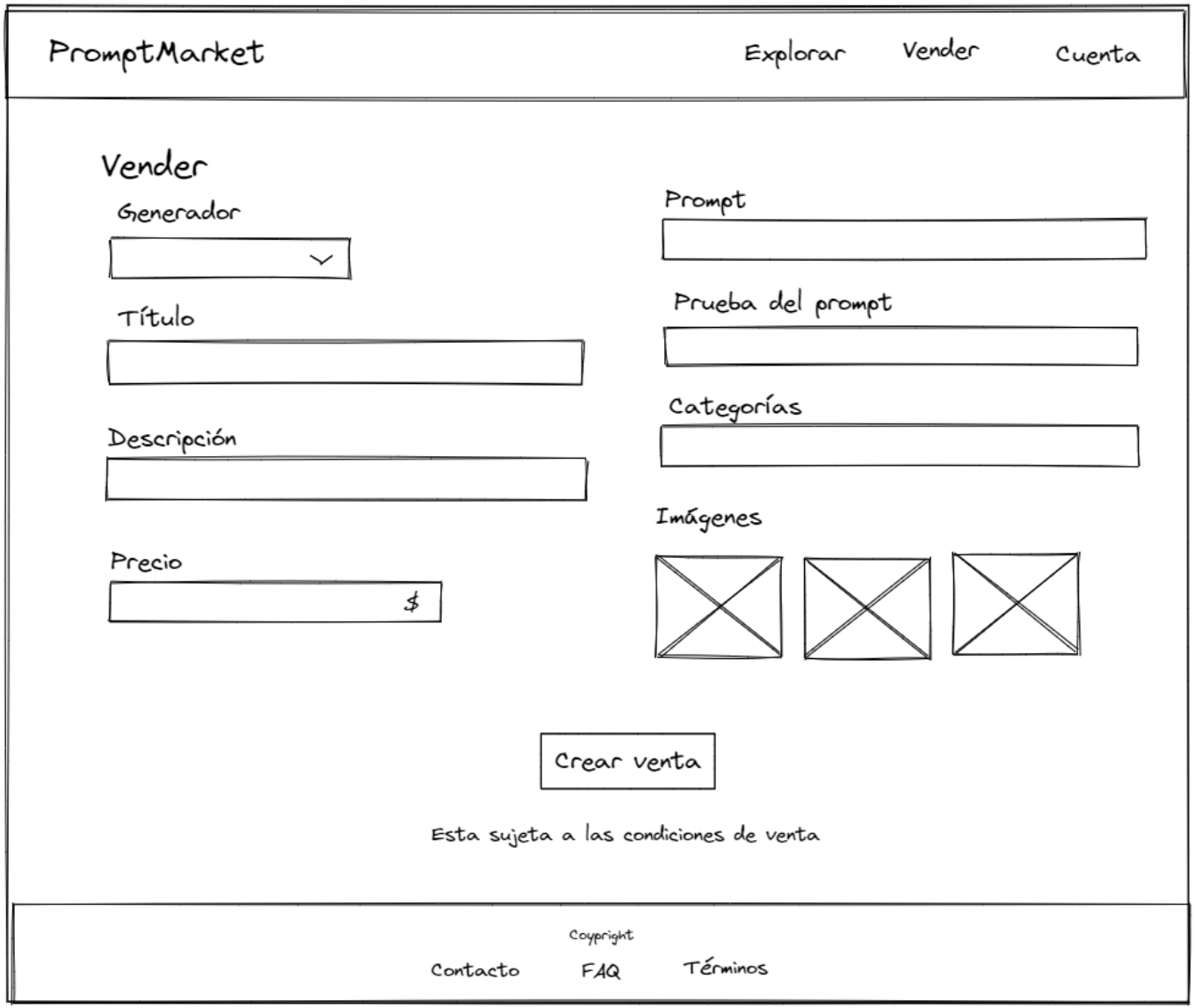


Figura 50. Prototipo de baja fidelidad de la página de venta de ordenador.

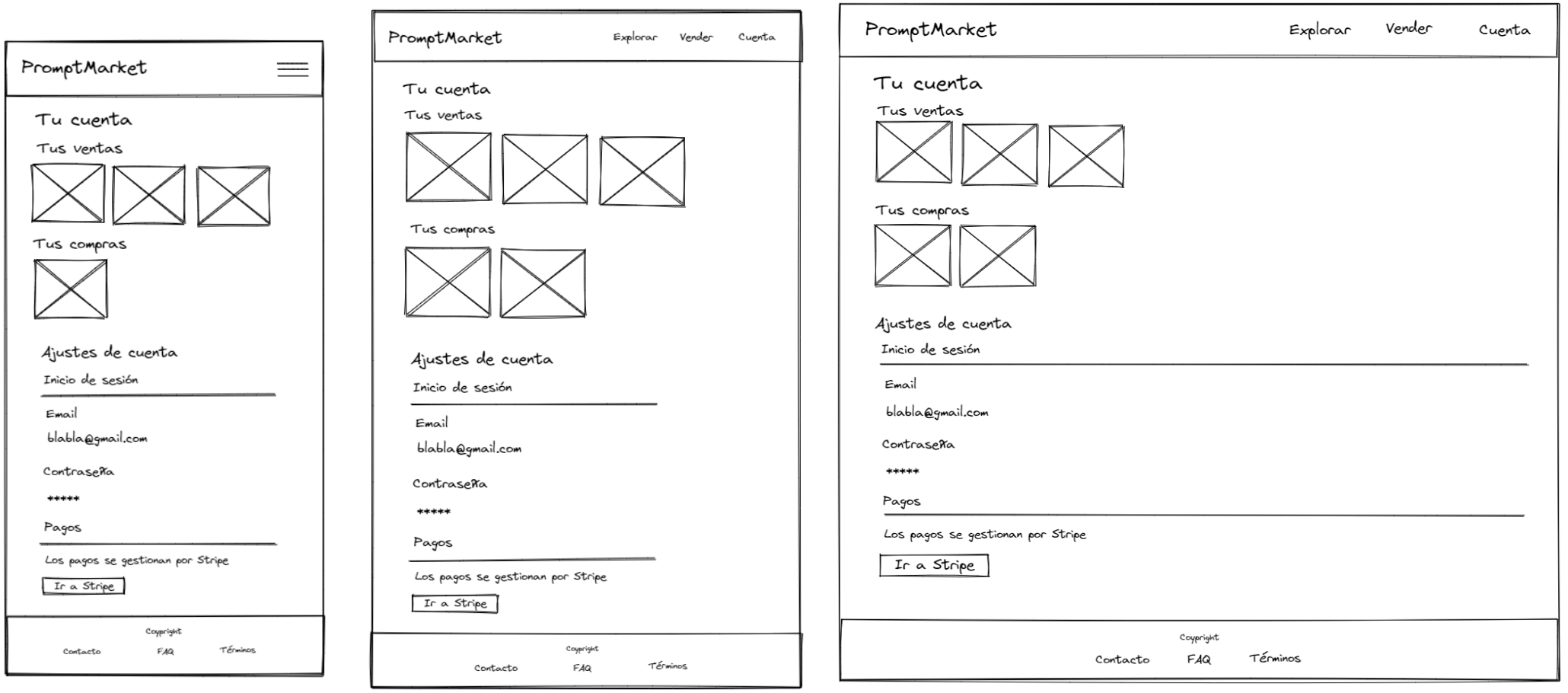


Figura 51. Prototipo de baja fidelidad de la página de cuenta. De izquierda a derecha: Móvil, tablet y ordenador.

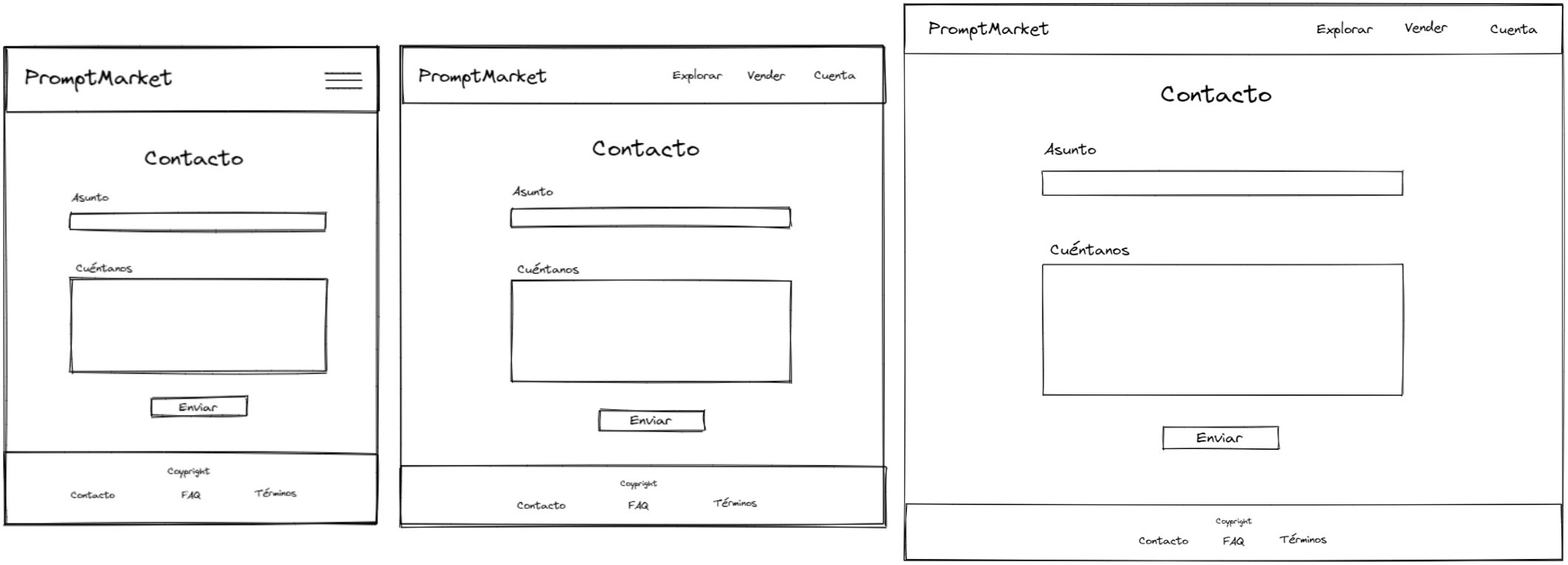


Figura 52. Prototipo de baja fidelidad de la página de contacto. De izquierda a derecha: Móvil, tablet y ordenador.

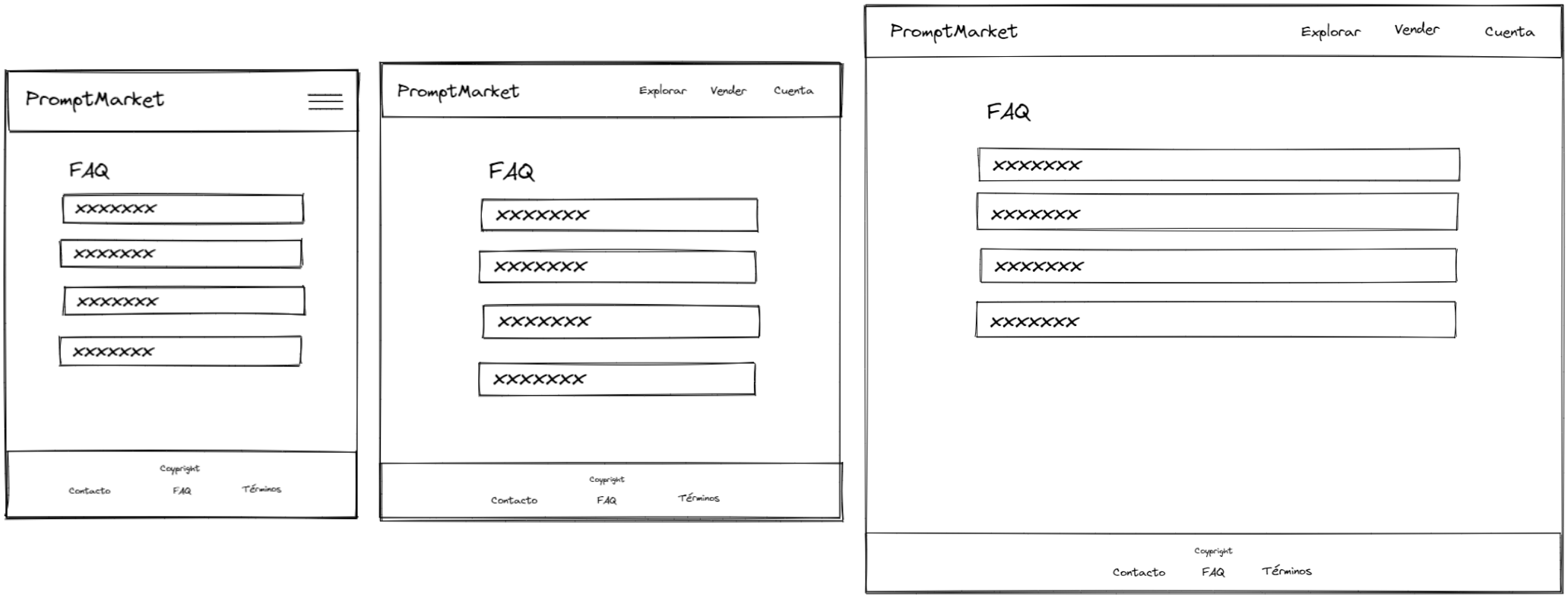


Figura 53. Prototipo de baja fidelidad de la página de FAQ. De izquierda a derecha: Móvil, tablet y ordenador.

Anexo 2 – Prototipo de alta fidelidad

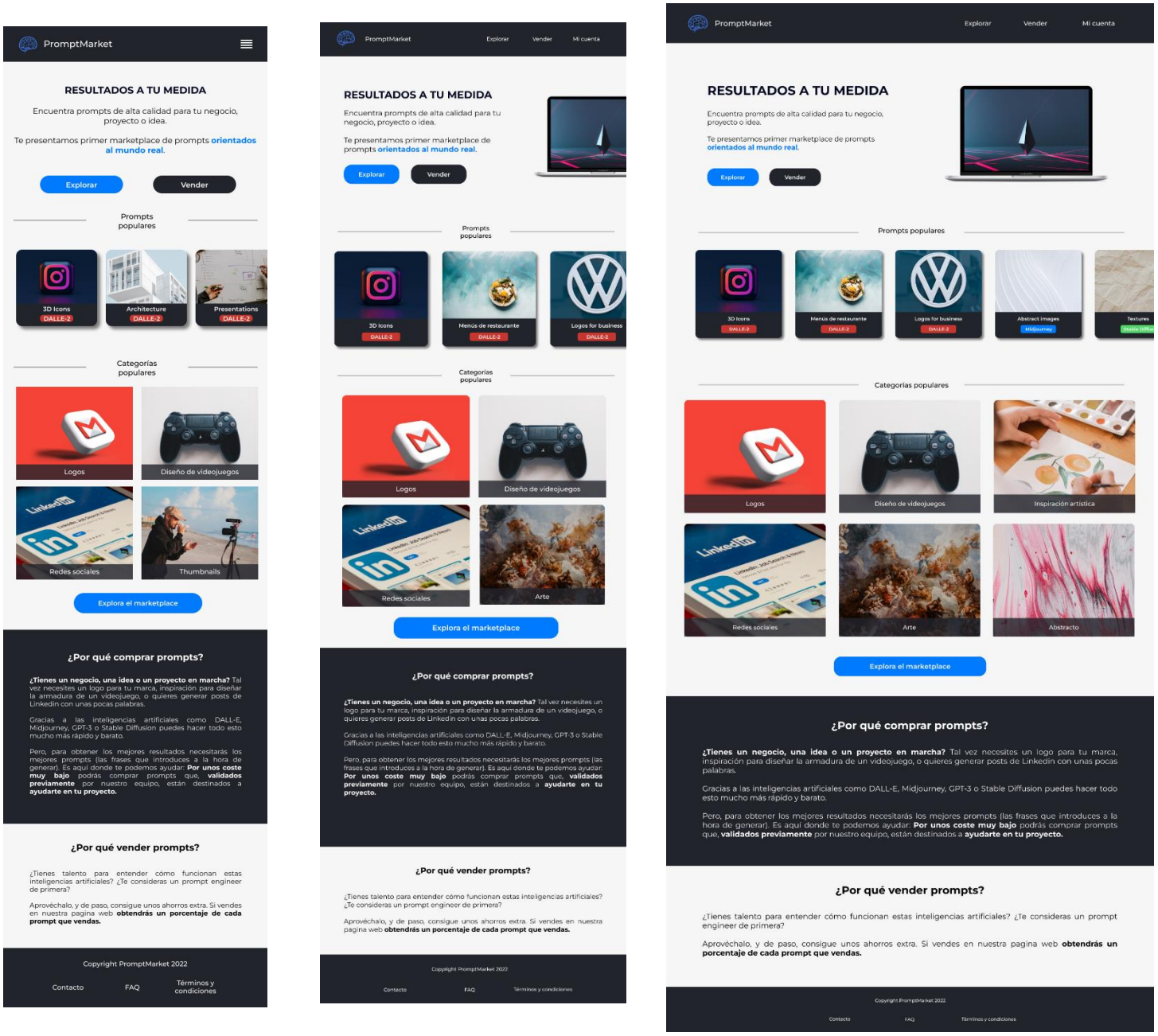


Figura 54. Prototipo de alta fidelidad de la página principal. De izquierda a derecha: Móvil, tablet y ordenador.

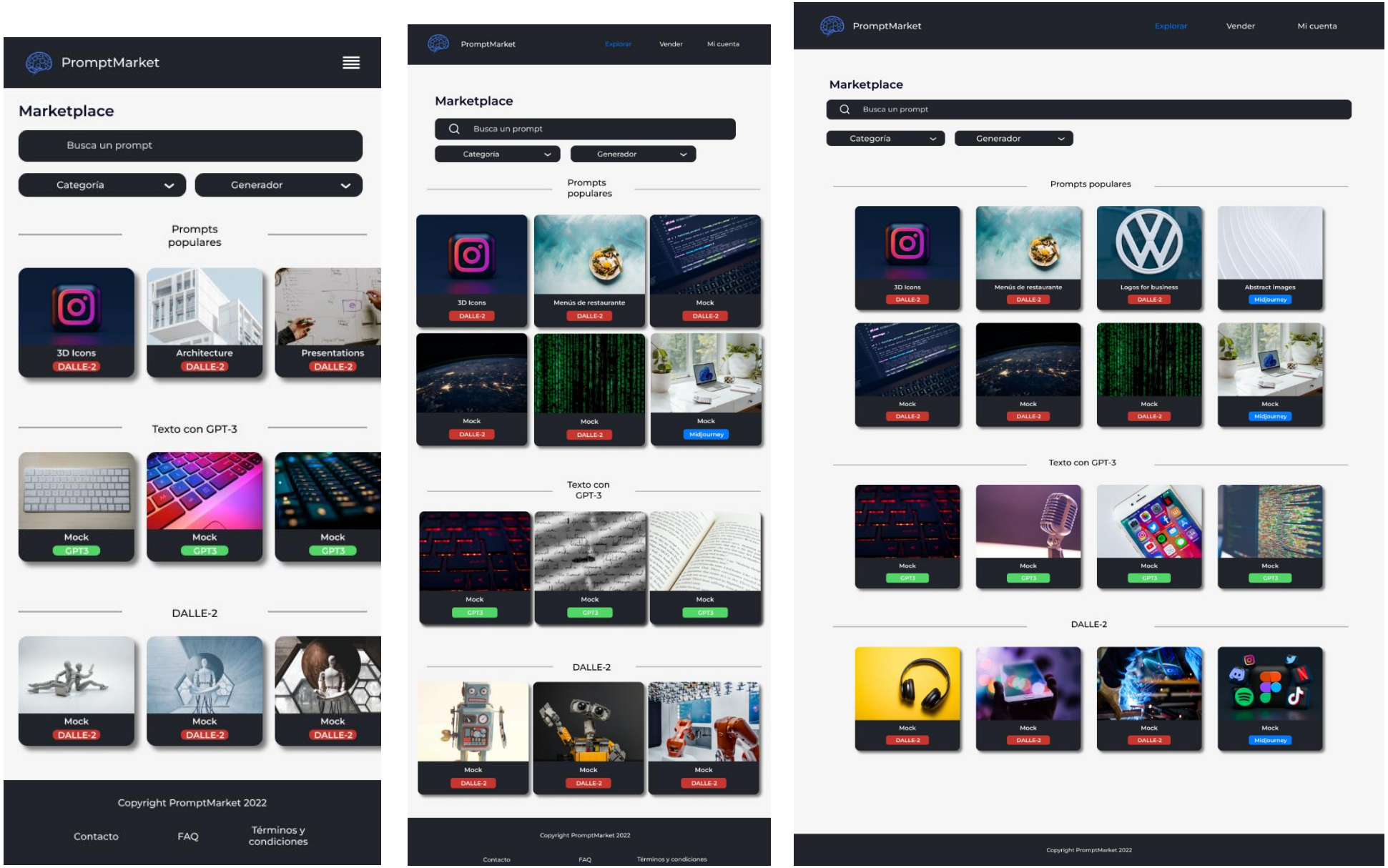


Figura 55. Prototipo de alta fidelidad de la página explora. De izquierda a derecha: Móvil, tablet y ordenador.

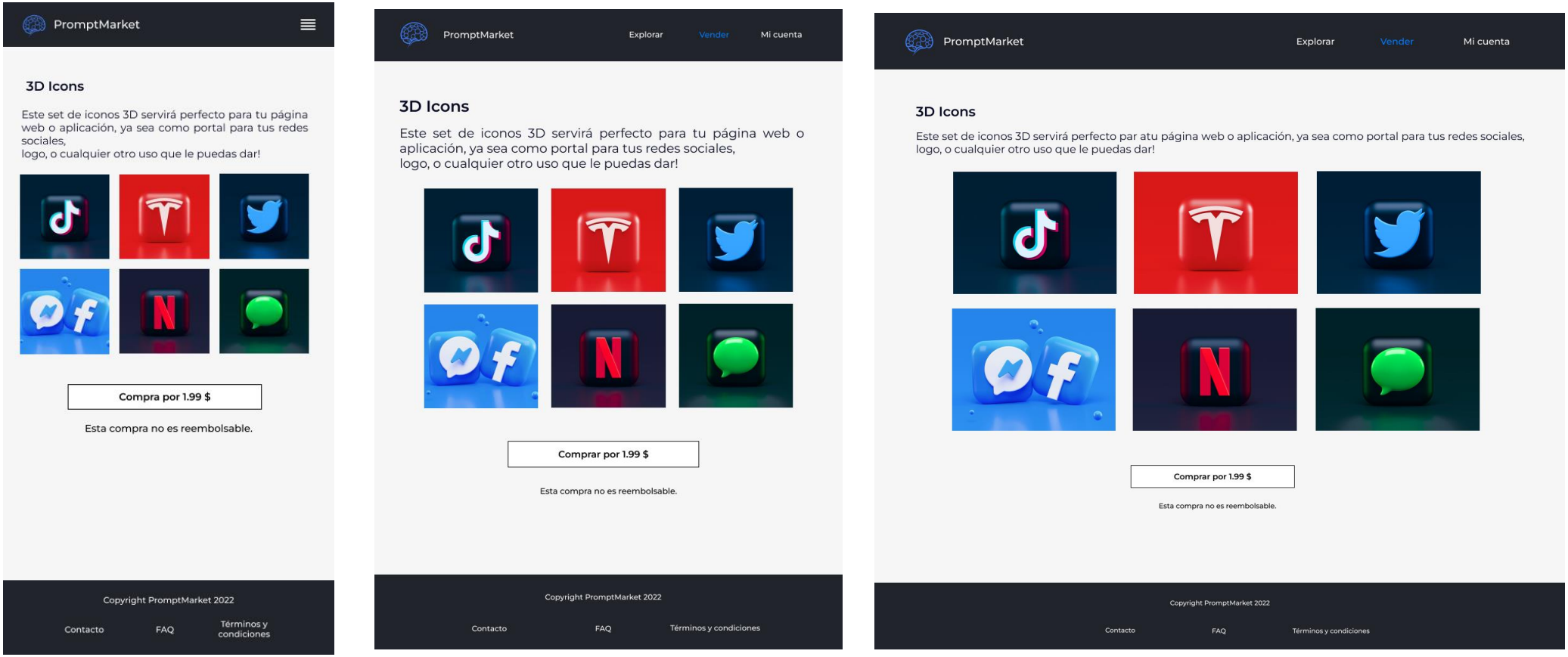


Figura 56. Prototipo de alta fidelidad de la página de compra. De izquierda a derecha: Móvil, tablet y ordenador.

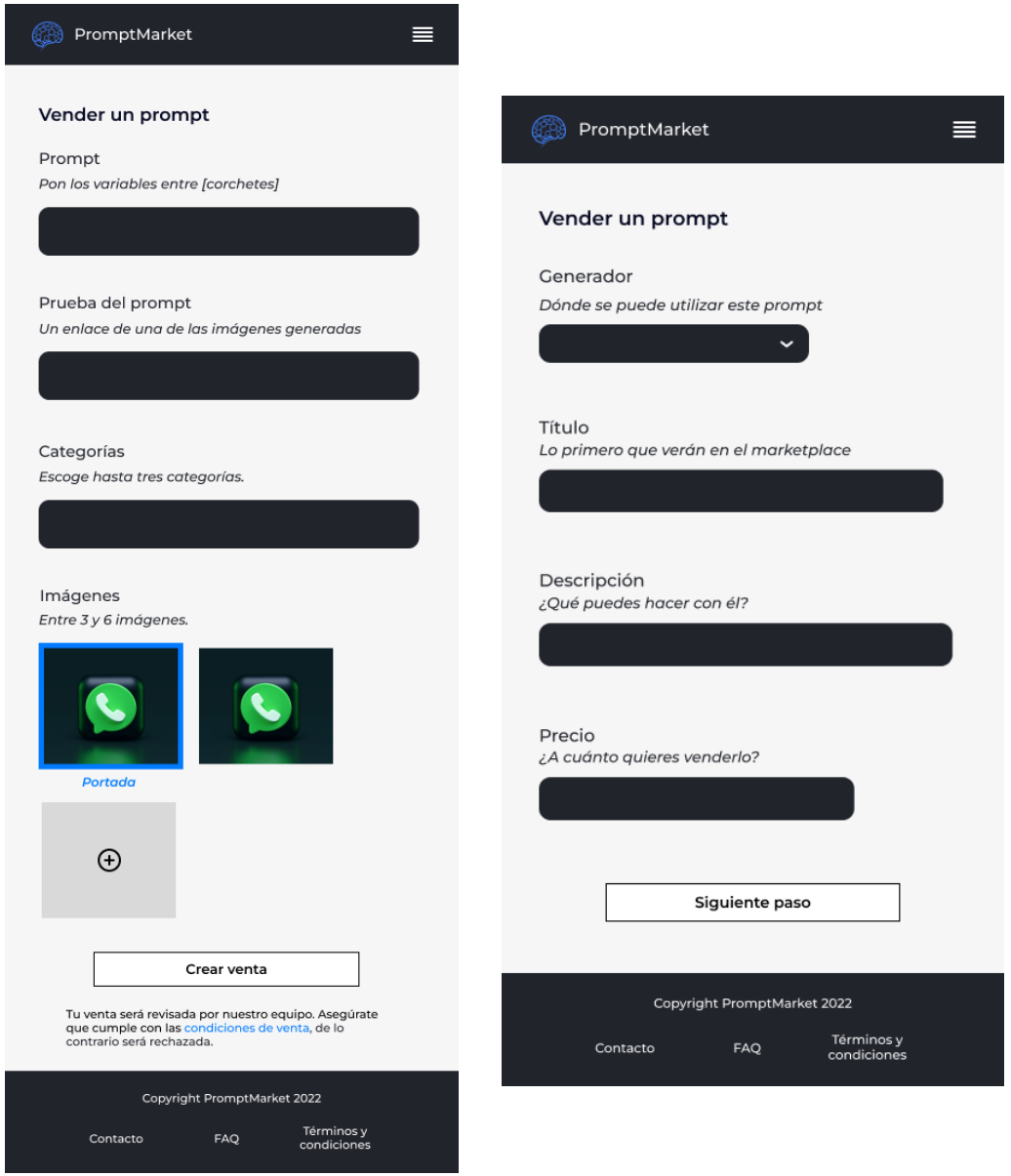
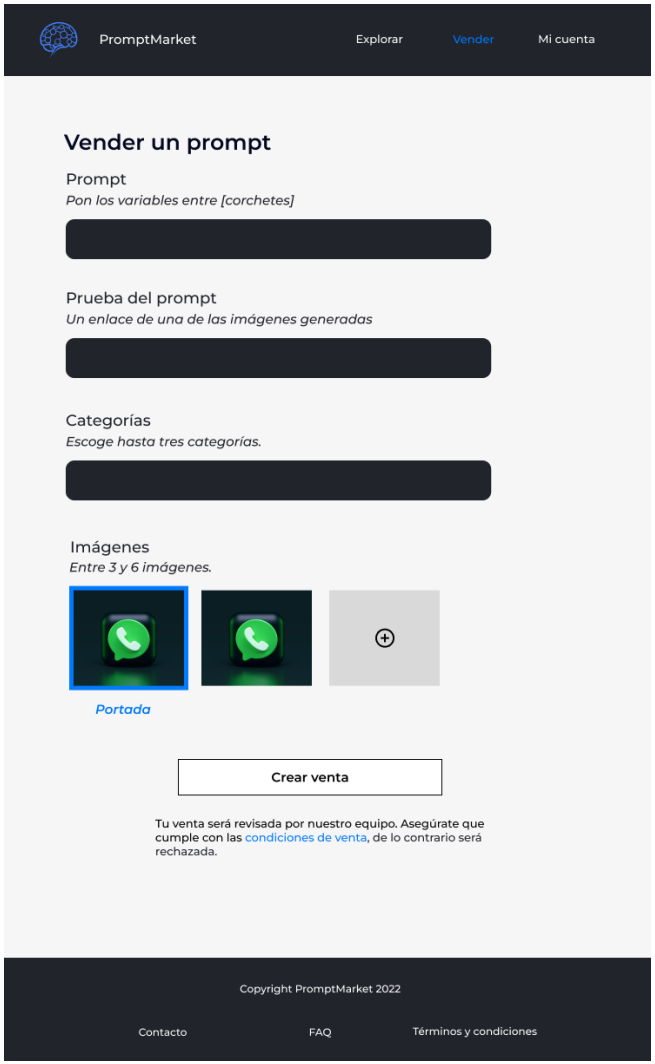


Figura 57. Prototipo de alta fidelidad de la página de venta en móvil.



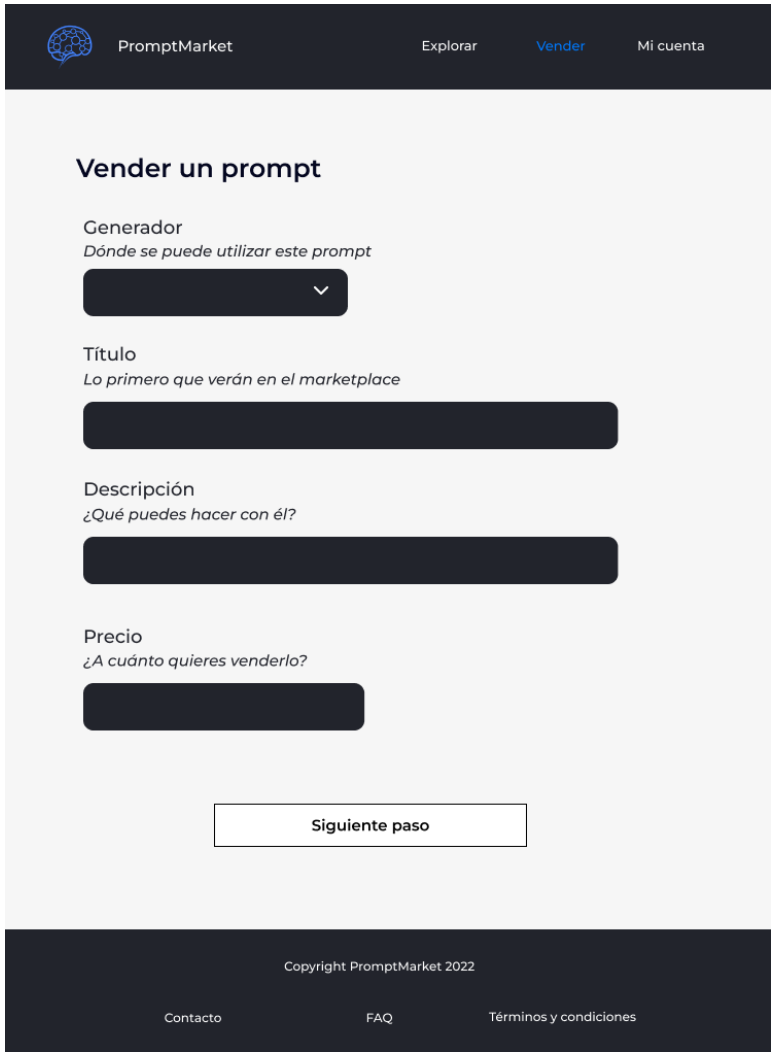


Figura 58. Prototipo de alta fidelidad de la página de venta en tablet.

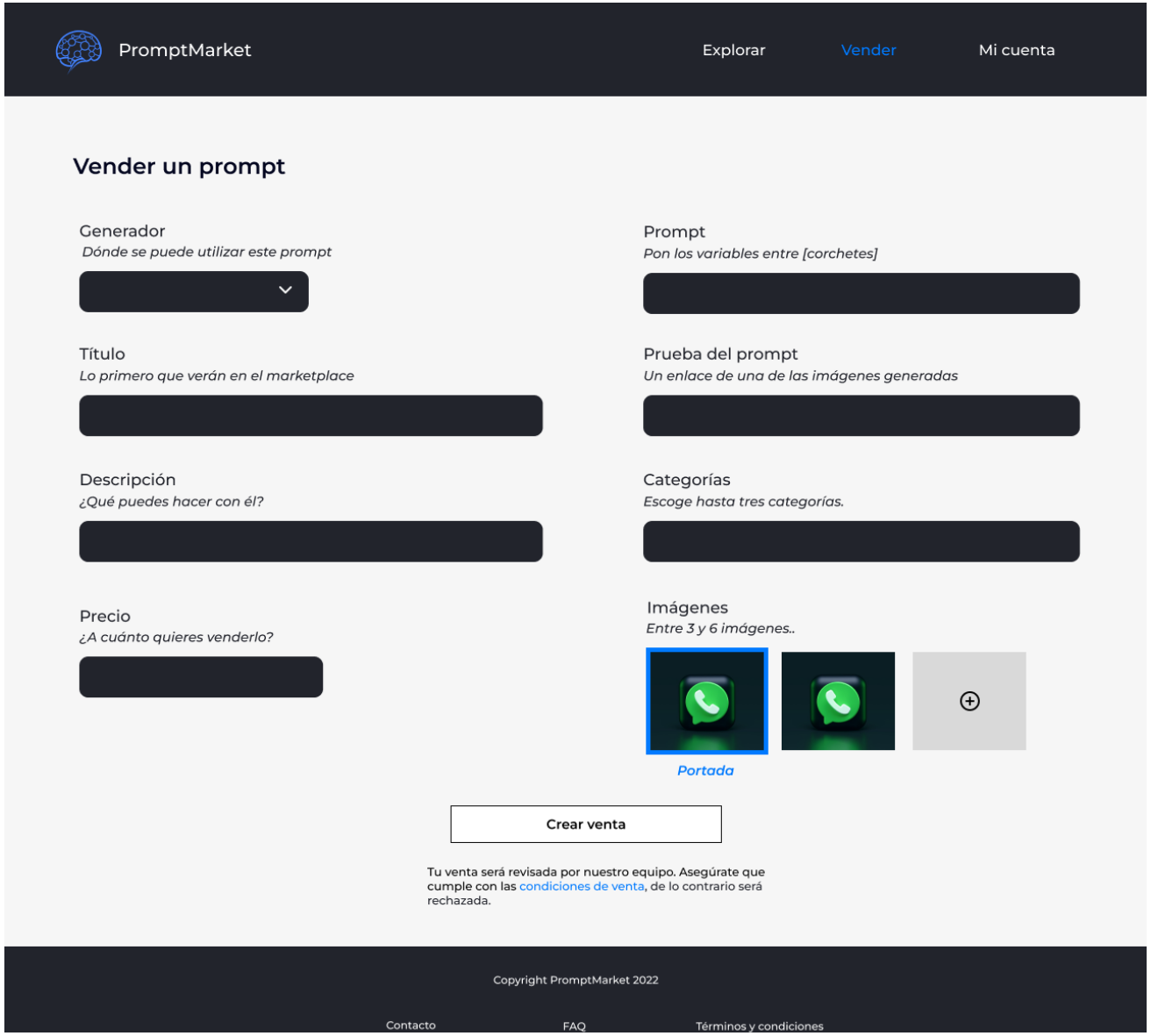


Figura 59. Prototipo de alta fidelidad de la página de venta en ordenador.

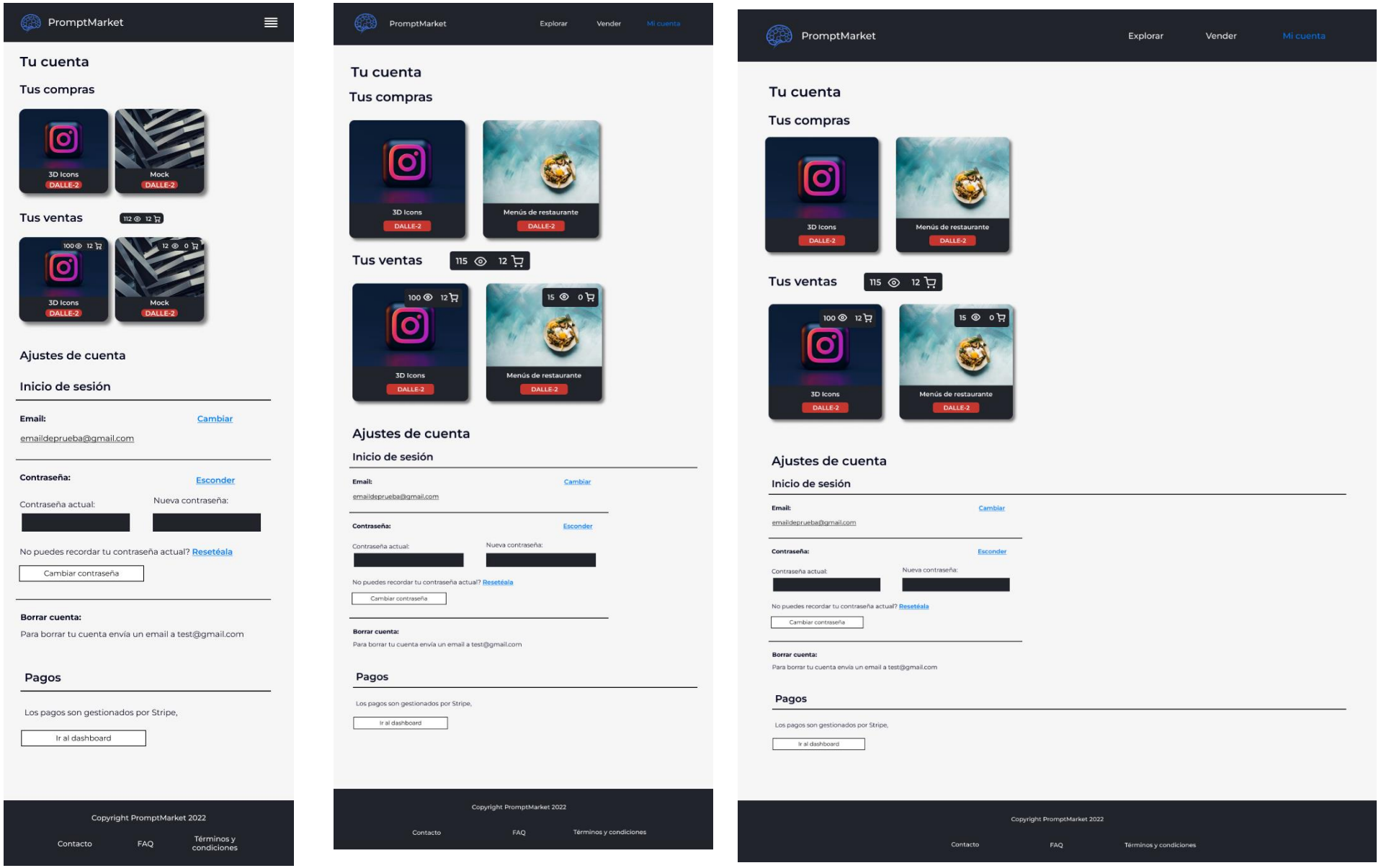


Figura 60. Prototipo de alta fidelidad de la página de cuenta. De izquierda a derecha: Móvil, tablet y ordenador.

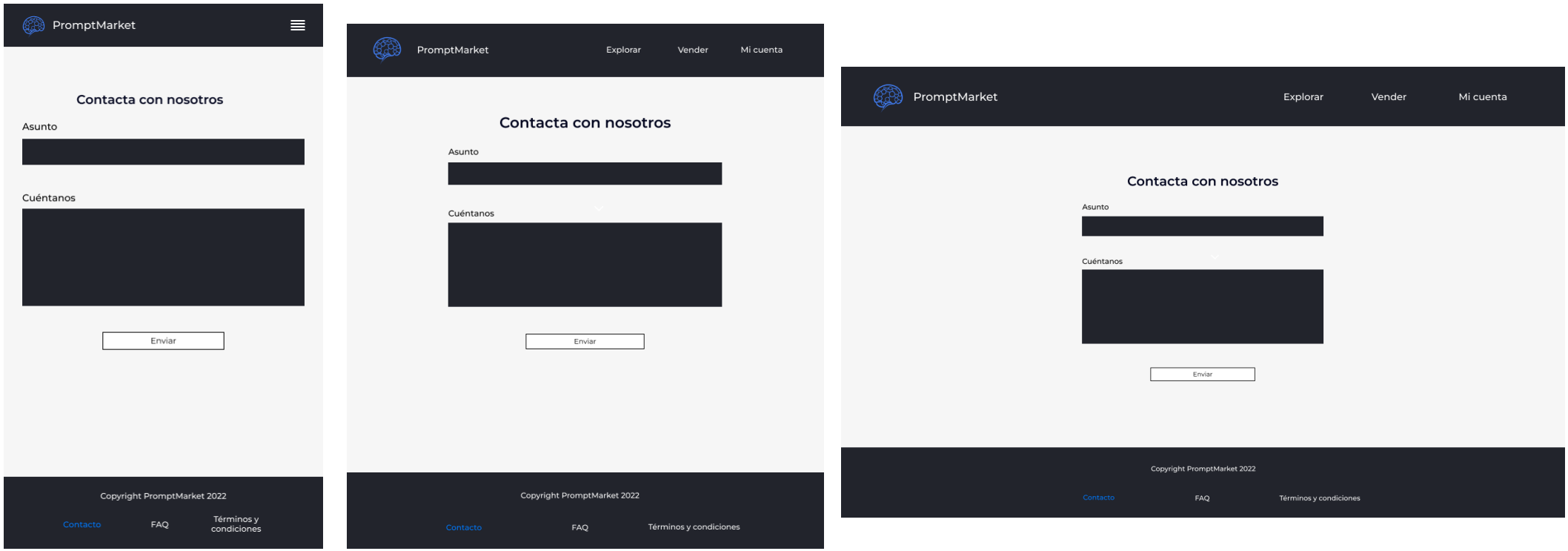


Figura 61. Prototipo de alta fidelidad de la página de contacto. De izquierda a derecha: Móvil, tablet y ordenador.

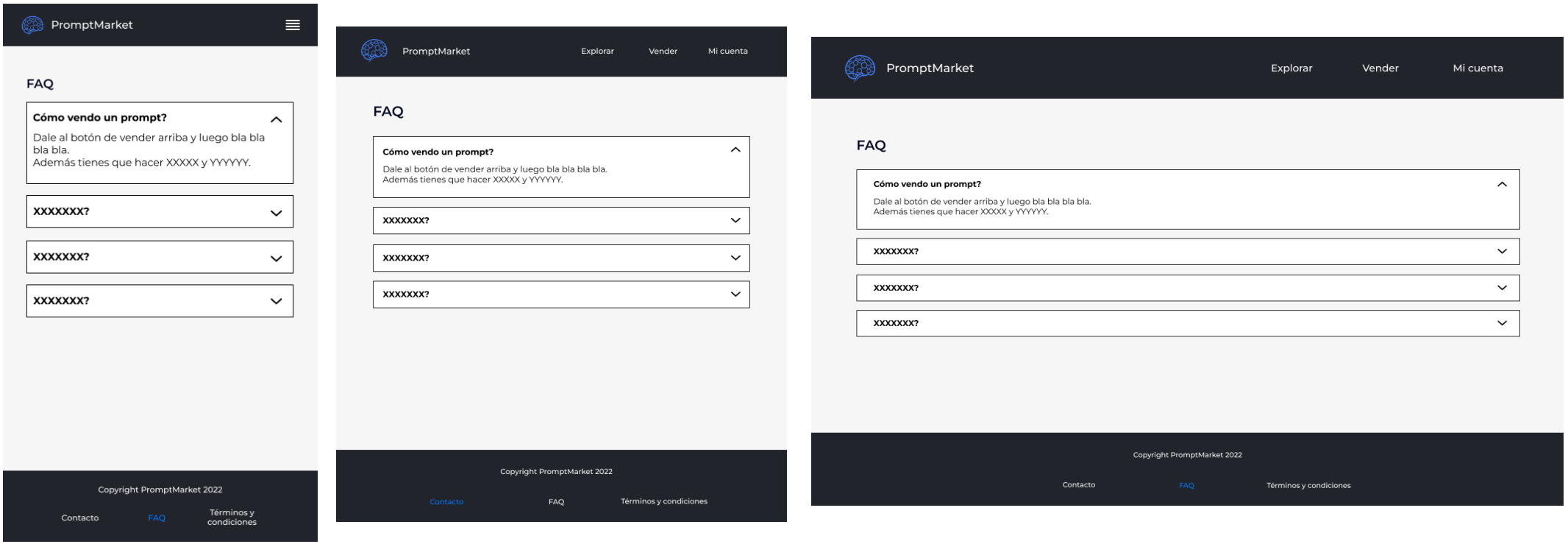


Figura 62. Prototipo de alta fidelidad de la página de FAQ. De izquierda a derecha: Móvil, tablet y ordenador.

Anexo 3 – Extractos de código del front-end

En este anexo se muestran los extractos de código más interesantes del front-end. El código completo del proyecto se ha adjuntado en un archivo .zip en el momento de la entrega.

Routing

Para realizar el routing interno de la aplicación, se ha utilizado **react-router-dom** que viene por defecto en react. Este permite realizar fácilmente el routing en el front-end. En el caso de este proyecto, se mantienen siempre tanto el header como el footer, por lo que el router debe ir en medio de estos. Así obtenemos una Single Page Application. Esto es el archivo App.tsx:

```
return (
  <div className="App">
    <HeaderComponent
      openDrawerFunction={() => setShowDrawer(!showDrawer)}
    ></HeaderComponent>
    {showDrawer && (
      <SideDrawerComponent
        closeFunction={() => setShowDrawer(false)}
      ></SideDrawerComponent>
    )}
    <Routes>
      <Route path="/" element={<Outlet></Outlet>}>
        <Route index element={<HomeScreen></HomeScreen>} />
        <Route path="explore" element={<ExploreScreen></ExploreScreen>} />
        <Route path="sell" element={<SellScreen></SellScreen>} />
        <Route path="account" element={<AccountScreen></AccountScreen>} />
        <Route path="contact" element={<ContactScreen></ContactScreen>} />
        <Route path="faq" element={<FAQScreen></FAQScreen>} />
        <Route path="Terms" element={<body>Terms</body>} />
        <Route path="prompts/:id" element={<PromptScreen></PromptScreen>} />
        <Route path="validate" element={<ValidateScreen></ValidateScreen>} />
        <Route path="*" element={<HomeScreen></HomeScreen>} />
      </Route>
    </Routes>
    <FooterComponent></FooterComponent>
  </div>
```

Figura 63. Routing del front-end

También es interesante ver que existe un componente llamado **SideDrawerComponent** que solo aplica en la versión móvil y que se activa desde el botón de tres rayas del header. Este SideDrawer va por encima de la pantalla y se muestra siempre a la hora de navegar:

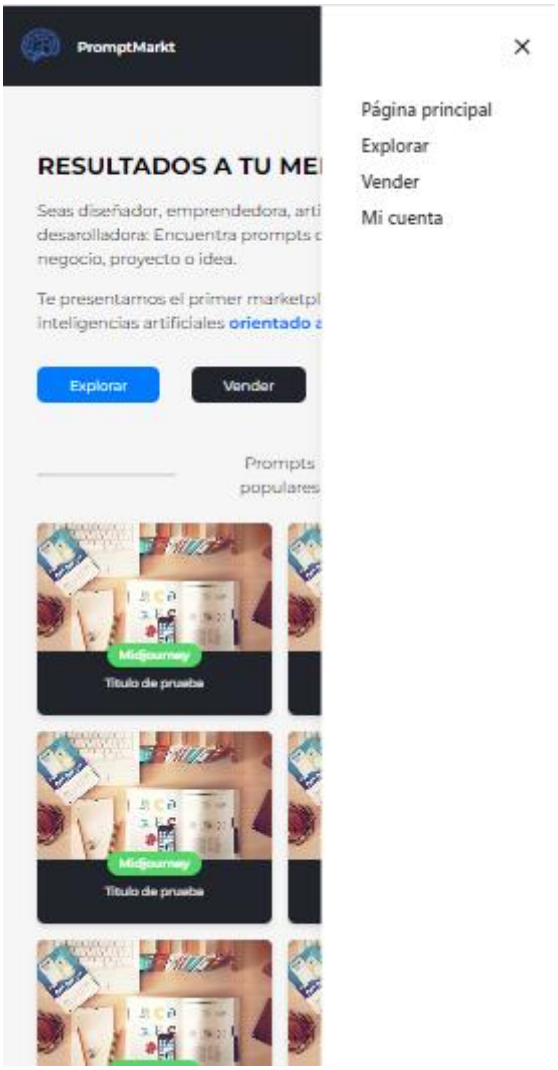


Figura 64. Side Drawer Component en la página web

Estilos globales

Aunque no se apliquen estilos parciales en hojas de CSS para pantallas, componentes, etc. Si que se aplican estilos globales en el archivo App.css, que aplica a todos los demás archivos del proyecto (ya que es llamado en App.tsx) que son necesarios para la estética de la web y se reutilizan en muchos sitios:

Scrollbar

Se ha eliminado cualquier scrollbar ya que estéticamente no es bonita. Para ello se aplica este estilo:

```
::-webkit-scrollbar {
  display: none;
}
```

Figura 65. Estilo global de scrollbar

body

Todas las pantallas, es decir, todos los archivos .tsx de la carpeta screens, irán obligatoriamente con un una tag contenedora <body> que tendrá el siguiente estilo:

```css
body {
  /* Full height - header height - footer height so that footer sticks perfectly to bottom if body is too small */
  min-height: calc(100vh - 7rem - 5rem);
  display: flex;
  flex-direction: column;
}
```

Figura 66. Estilo global del body

Este estilo implica que todos los elementos irán en columna, y que además debe tener un mínimo de altura del total de la pantalla – la altura del header (7 rem) – la altura del footer (5 rem). De esta manera evitamos problemas de pantallas semivacías o footers voladores.

Line clamp

En algunos casos es necesario que un texto esté contenido en un espacio de un máximo de líneas. En el caso de este proyecto ha sido necesario contener textos en un máximo de 1 y 2 líneas respectivamente. El código para conseguir esto es el siguiente:

```css
.line-clamp-2 {
  display: -webkit-box;
  -webkit-line-clamp: 2;
  -webkit-box-orient: vertical;
  overflow: hidden;
}

.line-clamp-1 {
  display: -webkit-box;
  -webkit-line-clamp: 1;
  -webkit-box-orient: vertical;
  overflow: hidden;
}
```

Figura 67. Estilo line clamp global

Si vemos un ejemplo real de dos líneas, quedaría así:

Figura 68. Ejemplo real de line clamp

Y en el código simplemente se ha tenido que llamar de esta manera:

```
<h1 className="text-xs text-white text-center font-regular line-clamp-2 px-4">
  {props.title}
</h1>
```

Figura 69. Llamada de line clamp en el código

Grid de prompts populares

En la página principal HomeScreen.tsx se utiliza una grid para distribuir los prompts populares según el tamaño de la pantalla. Gracias a Tailwind esto es una tarea sencilla que requiere apenas una línea de código:

```
<div className="mt-6 grid grid-cols-2 md:grid-cols-3 lg:grid-cols-4 xl:grid-cols-5 gap-4">
  {[...Array(width < 1024 ? 6 : width < 1280 ? 8 : 10)].map((e, i) => (
    <PromptCardComponent
      title="Titulo de prueba aaaaa aaaa aaaaaa aaaaa aaaa aaaa aaaa aaaaa"
      generator="Midjourney"
      image="https://i.picsum.photos/id/20/536/354.jpg?hmac=JRdy7LAV1wQa9JGfg21bQmuzBkh0P9HV3xwjmTEM0oQ"
      id="123456789"
    ></PromptCardComponent>
  ))}
</div>
```

El Array de dentro es de prueba mientras se rellena con datos reales del backend.

Como se puede observar, simplemente utilizando los atributos de Tailwind de responsividiad “md, lg, xl” se puede conseguir que se redistribuyan las tarjetas de la página principal, tal y como se quería en el diseño.

Venta por steps (o no)

En el caso de ordenador, la pantalla de ventas contiene todos los campos a la vez, mientras que, en móvil, para no estresar al usuario, está dividido en diferentes pasos. Para ello, la manera más directa de hacerlo es determinando a partir de que ancho de pantalla habrá steps. En este caso es de 1024 px para abajo, y se hace así:

```
{/* Primer step */}
{(width > 1024 || (width <= 1024 && step === 1)) && (···
)}
{/* Segundo step */}
{(width > 1024 || (width <= 1024 && step === 2)) && (···
)}
```

Figura 70. Código front-end para los steps (o no) de la pantalla de ventas

Si la condición se cumple, se mostrará esa parte del código, por lo que en los dos casos, si el ancho de pantalla es más grande de 1024, se muestra (es decir, vemos todos los campos), y sino, a través de la variable step determinamos qué estado se ve.

Para obtener el ancho se ha hecho con el siguiente código:

```
// Window size state
const [width, setWidth] = React.useState<number>(window.innerWidth);

function handleWindowSizeChange() {
  setWidth(window.innerWidth);
}

// This hook runs so we can resize the elements if needed
React.useEffect(() => {
  window.addEventListener("resize", handleWindowSizeChange);
  return () => {
    window.removeEventListener("resize", handleWindowSizeChange);
  };
}, []);
```

Figura 71. Código para obtener el ancho de pantalla de forma responsiva en React

Gracias al hook useEffect, que se lanza cada vez que ha un cambio, se obtiene el tamaño de pantalla, de manera que si hay un resize o un cambio de orientación, la pantalla lo detectará y cambiará acorde con ello.

Para cambiar el step o lanzar la venta hay diferentes botones que se muestran según condiciones parecidas a las de los campos:

```
<div className="flex flex-col usm:flex-row items-center justify-evenly mt-8 text-center">
  {width <= 1024 && step === 1 && (
    <ButtonComponent
      title="Siguiente"
      function={() => handleNextStep()}
    ></ButtonComponent>
  )}
  {width <= 1024 && step === 2 && (
    <ButtonComponent
      title="Anterior"
      function={() => setStep(1)}
    ></ButtonComponent>
  )}
  {(step === 2 || width > 1024) && (
    <ButtonComponent
      title="Enviar"
      function={() => null}
      type="submit"
      class="action"
      containerStyle={"mt-5 usm:mt-0"}
    ></ButtonComponent>
  )}
</div>
```

El botón de siguiente solo se mostrará en el step 1 y nos llevará al step 2, y el de anterior viceversa. Ambos botones solo se muestran en el caso de que el width sea menor o igual a 1024.

El botón de enviar venta se muestra en el último step o siempre que el width sea superior a 1024.

Carga de imágenes en el front-end

Para cargar las imágenes en front-end se ha utilizado un paquete externo llamado **react-images-uploading** que facilita mucho el trabajo, ya que hacerlo manualmente trae problemas.

El código es muy extenso para ser mostrado en su totalidad en un mismo bloque, por lo que se desgrana:

Parte 1- Botón y contenedor

El contenedor es el componente **ImageUploading** del paquete mencionado anteriormente, y dentro de el contiene un botón con un icono de **react-feather**, en el cual al hacer click llama a la función ya establecida de subir imágenes.

Este componente es muy interesante ya que también se pueden soltar imágenes en el botón y se suben, sin necesidad de tener que seleccionar los archivos a mano.

Además, se pueden bloquear tanto el número máximo de imágenes (6) a subir como el tamaño de archivo máximo (5 MB), lo cual es una gran ventaja para no tener que validar nosotros.

```
<ImageUploading
  multiple
  value={images}
  onChange={onImageChange}
  maxNumber={6}
  maxFileSize={5000000}
  dataURLKey="data_url"
>
  {({
    imageList,
    onImageUpload,
    onImageRemoveAll,
    onImageUpdate,
    onImageRemove,
    isDragging,
    dragProps,
    errors
  }) => (
    <div>
      <button
        style={isDragging ? { color: "red" } : undefined}
        className="flex flex-row items-center bg-white text-black text-sm px-4 py-2 rounded-md mt-2 border"
        onClick={onImageUpload}
        {...dragProps}
        type="button"
      >
        <UploadCloud size={15} color={"black"}></UploadCloud>
        <span className="ml-2">
          Sube o suelta aquí las imágenes
        </span>
      </button>
```

Figura 72. Subida de imágen en front-end - Parte 1

Parte 2 – Errores

Se distinguen dos clases de errores, por una parte, los “errors” que son propios del paquete, y son, por ejemplo, de tamaño de subida, o de número de archivos, y se gestionan con este código:

```
{errors && (
  <div className="mt-2">
    {errors.maxNumber && (
      <span className={styles.errorText}>
        Solo puedes subir hasta 6 imágenes.
      </span>
    )}
    {errors.acceptType && (
      <span className={styles.errorText}>
        El tipo de archivo seleccionado no está
        permitido.
      </span>
    )}
    {errors.maxFileSize && (
      <span className={styles.errorText}>
        El peso máximo es de 5 MB por archivo.
      </span>
    )}
```

Figura 73. Errores de subida de imagen 1 - Parte 2

Y después el error de “no hay ninguna imagen”, que se gestiona con código propio (no del paquete):

```
{imagesError && (
  <div className="mt-2">
    <span className={styles.errorText}>
      {imagesError}
    </span>
  </div>
)}
```

Figura 74. Errores de subida de imagen 2 - Parte 2

Anexo 4 – Extractos de código del back-end

Query para obtener los últimos prompts

Para obtener los prompts más nuevos, ese necesario hacer una query que tenga en cuenta la fecha de creación del prompt, y además, que el prompt sea aprobado. Para revisar que el prompt sea aprobado, será necesario hacer un INNER JOIN con la tabla de prompt_status, ya que lo usaremos para hacer la query.

Además de todo, será necesario obtener la imagen de portada, a través de una query en la tabla de cover_images, también con un JOIN.

Esto es posible hacerlo ya que las tablas están relacionadas entre sí por el ID del prompt.

Esto se hace con el siguiente código:

```
export const getNewestPrompts = async (limit: number) => {
  const { data, error } = await supabase
    .from("prompt_info")
    .select(
      `*,
    cover_images(
      path
    ),
    prompt_status!inner(
      status
    ) `
    )
    .order("created_at", { ascending: false })
    .limit(limit)
    .eq("prompt_status.status", "approved");

  return { data, error };
};
```

Figura 75. Query para obtener los últimos prompts

Como se puede observar, para hacer un join simplemente se llama a la tabla, y entre paréntesis se llama el campo buscado. Para un inner join, se tiene que añadir el sufijo “!inner” al nombre de la tabla.

El resto de ordenes son bastante claras:

- Order, para ordenar según la fecha de creación.
- Limite, para limitar el número de respuestas.
- Eq, para que se cumpla la condición de que el status sea aprobado.

Query de búsqueda

Para las búsquedas de usuario hay una distinción importante: si se ha buscado un texto o solo se han seleccionado generador y categoría. En este caso se verá el caso completo, aunque existen dos queries distintas, una con el texto y una sin él.

Para esta query se hace uso del ILIKE, que hace búsquedas de caracteres similares en un campo en concreto de una tabla, y de IN, para decirle que un campo debe estar dentro de unos ciertos valores.

También deben estar aprobados, como en el caso anterior. El código en este caso es:

```
export const searchPromptsWithSearchValue = async (
  search: string,
  generatorIdArray: number[],
  categoryIdArray: number[]
) => {
  const { data, error } = await supabase
    .from("prompt_info")
    .select(
      `*,
    cover_images(
      path
    ),
    prompt_status(
      status
    ) `
    )
    .order("created_at", { ascending: false })
    .ilike("title", `%${search}%`)
    .ilike("description", `%${search}%`)
    .in("generator_id", generatorIdArray)
    .in("category_id", categoryIdArray)
    .eq("prompt_status.status", "approved");

  return { data, error };
};
```

Figura 76. Query de la función de búsqueda

Insertar imágenes en el storage

Insertar imágenes no es tarea fácil, puesto que el formato en el que llega la imagen (url + bytes), tiene que ser transformado de manera que el storage de Supabase lo acepte.

Como el código en este caso es más largo, se va a dividir en partes para entenderlo mejor.

Primero se tiene la recepción de las imágenes, en este caso un Array de este formato que se debe transformar. Además, se recibe un índice que es el de la imagen de portada, para poder identificarla en el proceso, y el Id del prompt para poder nombrar así la carpeta del storage. Se retorna una Promesa al ser asíncrono:

```
export const uploadImagesToStorage = async (
  images: any[],
  coverIndex: number,
  promptId: string
): Promise<any> => {
```

Figura 77. Parte uno de introducir imágenes en el storage

Como se recibe un Array, se tiene que loopear sobre el para tratar cada imagen por separado. Dee la imagen se coge el tipo de archivo, la extensión, y se crea el nombre del archivo, que es realmente un path en el storage:

```
for (let i = 0; i < images.length; i++) {
  const image = images[i].data_url;
  const fileExtension = images[i].file.type;
  const fileType = fileExtension.replace("image/", "");
  const fileName = `prompts/${promptId}/${i}.${fileType}`;
```

Figura 78. Parte dos de introducir imágenes en el storage

Seguidamente se llama a Supabase para subir el archivo. Se le pasan como argumentos el nombre del archivo, la propia imagen y la extensión del archivo (png, jpg). La imagen tiene que ser decodeada ya que viene en formato bytes, y se le tiene que quitar la parte inicial del objeto ya que entonces la subida es errónea:

```
const { data, error } = await supabase.storage
  .from("prompt.images")
  .upload(
    fileName,
    decode(image.replace(`data:${fileExtension};base64,`, "")),
    {
      contentType: fileExtension
    }
  );
```

Figura 79. Parte tres de introducir imágenes en el storage

Si hay un error de subida, se manda una alerta, y para el resto de casos, si se ha subido bien no se hace nada, exceptuando el caso de la portada, donde se inserta en una tabla la url de la imagen subida para poder accederla más fácilmente que el resto:

```
if (error) {
  alert("Error al subir la imagen número" + (i + 1).toString());
} else {
  if (coverIndex === i) {
    insertCoverImageInTable(promptId, getPublicUrl(data.path).publicUrl);
  }
}
```

Figura 80. Parte cuatro de introducir imágenes en el storage

Edge Function – Stripe Checkout

Hay varias funciones para Stripe, pero esta es la más representativa, ya que combina supabase con Stripe, y no es demasiado compleja como para confundir al lector. Al ser muy larga, se dividirá en partes.

Primero se deben iniciar todos los servicios: Supabase y Stripe. Estos servicios requieren de las claves secretas, que están guardadas seguras en Supabase. Se utiliza Deno para todas estas importaciones.

```
import { serve } from "https://deno.land/std@0.131.0/http/server.ts";

import Stripe from "https://esm.sh/stripe@11.0.0?target=deno&no-check";

import { createClient } from "https://esm.sh/@supabase/supabase-js@1.33.1";

const SUPABASE_URL = Deno.env.get("SUPABASE_URL") ?? "";
const SERVICE_KEY = Deno.env.get("SUPABASE_SERVICE_ROLE_KEY") ?? "";

const supabase = createClient(SUPABASE_URL, SERVICE_KEY);

const stripe = Stripe(Deno.env.get("REACT_APP_STRIPE_TEST_SECRET_KEY") ?? "", {
  // This is needed to use the Fetch API rather than relying on the Node http
  // package.
  httpClient: Stripe.createFetchHttpClient()
});
```

Figura 81. Parte uno de la función stripe checkout

Una vez iniciado, también se deben crear los “corsHeaders”, que son los Headers que se tienen que mandar en la request si esta se hace desde navegador, ya que de lo contrario son bloqueadas y las funciones retornan error.

Una vez iniciada la función (serve), se comprueba que los headers sean correctos:

```
const corsHeaders = {
  "Access-Control-Allow-Origin": "*",
  "Access-Control-Allow-Headers": "authorization, x-client-info, apikey"
};

serve(async (req: any) => {
  if (req.method === "OPTIONS") {
    return new Response("ok", {
      headers: corsHeaders
    });
  }
```

Figura 82. Parte dos de la función stripe checkout

Una vez hecho esto, se reciben los arguments mandados en la función de esta manera:

```
const { url, buyerId, promptId } = await req.json();
```

Figura 83. Parte tres de la función stripe checkout

Una vez hecho esto, el primer paso para iniciar el checkout es obtener los ID's de Stripe que se tienen que mandar en la request a Stripe, el del precio, el de la cuenta y el precio (el precio es para calcular el % que se lleva promptmarkt).

```
try {
  console.log("Checkout session initiated");

  return await supabase
    .from("stripe_products")
    .select(`stripe_price_id, stripe_account_id, price`)
    .eq("prompt_id", promptId)
    .single()
    .then(async (result: { data: any; error: any }) => {
```

Figura 84. Parte cuatro de la función stripe checkout

Si esta request ya retorna error, directamente se retorna un status 400 que indica error:

```
if (result.error || result.data === null) {
  console.log("Checkout session error: ", result.error);
  return new Response(JSON.stringify(result.error), {
    headers: {
      ...corsHeaders,
      "Content-Type": "application/json"
    },
    status: 400
  });
```

Figura 85. Parte cinco de la función stripe checkout

En el caso contrario, si obtenemos la respuesta, se inicia la sesión de checkout de Stripe:

```
return await stripe.checkout.sessions
  .create({
    line_items: [
      {
        price: result.data.stripe_price_id,
        quantity: 1
      }
    ],
    mode: "payment",
    success_url: url + "/success",
    cancel_url: url,
    automatic_tax: {
      enabled: true
    },
    metadata: {
      buyerId: buyerId,
      promptId: promptId
    },
    payment_intent_data: {
      application_fee_amount: result.data.price * 100 * 0.3,
      transfer_data: {
        destination: result.data.stripe_account_id
      },
      metadata: {
        buyerId: buyerId,
        promptId: promptId
      }
    }
  })
```

Figura 86. Parte seis de la función stripe checkout

En ella se especifican el Price ID, la cantidad de ítems (siempre 1), las URL's en el caso de éxito o de cancelación, la metadata, que aparece dos veces ya que el checkout y el pago son dos procesos diferentes y es necesaria la data en los dos, la cantidad que recibirá promptmarkt (30%) y la cuenta de destino que recibirá el 70% restante.

Una vez recibida la respuesta, se revisa si se ha obtenido un error o un éxito, y se retorna en función de ello:

```
.then((session: any) => {
  console.log("Checkout session created: ", session);
  return new Response(JSON.stringify(session), {
    headers: {
      ...corsHeaders,
      "Content-Type": "application/json"
    },
    status: 200
  });
})
.catch((error: any) => {
  console.log("Checkout session error: ", error);
  return new Response(JSON.stringify(error), {
    headers: {
      ...corsHeaders,
      "Content-Type": "application/json"
    },
    status: 400
  });
});
```

Figura 87. Parte siete de la función stripe checkout

En caso de éxito, se cambia la URL de la página con esta línea de código:

```
if (v.data !== null) {
  window.location.href = v.data.url;
```

Figura 88. Parte ocho de la función stripe checkout

Edge Functions – Stripe Webhook

En este caso no se enseñará el código entero, sino la parte más importante del webhook, ya que el resto es repetitivo.

Para recibir un evento primero se tiene que llamar la función "constructEventAsync" que construye el evento recibido. Además, utiliza una key diferente a la de Stripe, específica para el webhook, y una firma que viene en el header:

```
const cryptoProvider = Stripe.createSubtleCryptoProvider();

console.log(`Function "stripe-webhooks" up and running!`);

serve(async (request) => {
  const signature = request.headers.get("Stripe-Signature");
```

```
let receivedEvent;
try {
  receivedEvent = await stripe.webhooks.constructEventAsync(
    body,
    signature,
    Deno.env.get("REACT_APP_STRIPE_TEST_WEBHOOK_KEY"),
    undefined,
    cryptoProvider
  );
```

Figura 89. Parte uno de la función de webhooks de stripe

Una vez recibido el evento, simplemente se tienen que tratar los casos y ejecutar el código deseado acorde:

```
let retrievedEvent;
try {
  retrievedEvent = await stripe.events
    .retrieve(receivedEvent.id, requestOptions)
    .then(async (event) => {
      if (
        event.type === "payment_intent.succeeded" ||
        event.type === "checkout.session.async_payment_succeeded"
      ) {
        return await supabase
          .from("sales")
          .insert({
            buyer_id: event.data.object.metadata.buyerId,
            prompt_id: event.data.object.metadata.promptId
          })
```

Figura 90. Parte dos de la función de webhooks de stripe

Supabase Policies

Las políticas de Supabase permiten aumentar la seguridad de las tablas seleccionando quien tiene acceso a qué operaciones.

Por ejemplo, la política más compleja implementada actualmente define que solo se pueda obtener el prompt_data si eres has creado el prompt. Para ello, se hace uso de datos en la tabla de autores y del user id:

```
(EXISTS ( SELECT 1
   FROM prompt_authors
  WHERE ((prompt_authors.prompt_id = prompt_data.prompt_id) AND (prompt_authors.
author_id = uid()))))
```

Figura 91. Supabase policy para prompts que has creado

También hay otra política por si lo has comprado, mirando la tabla de ventas:

```
(uid() IN ( SELECT sales.buyer_id
   FROM sales
  WHERE (sales.prompt_id = sales.prompt_id)))
```

Figura 92. Supabase policy para prompts que has comprado

De esta manera se evita que usuarios maliciosos obtengan el producto a través de queries.

Otra política que se presenta en muchas tablas es la de que si eres administrador tengas derechos de lectura (y otros). Para ello, se lee si el user id corresponde con alguno de los de la tabla de administradores:

```
(uid() IN ( SELECT admins.user_id
   FROM admins))
```

Figura 93. Supabase policy de administradores

El resto de políticas son similares y más sencillas que estas.