UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Migración de la web de una clínica de psicología online. De WordPress a una implementación basada en Astro

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Moreno Carrasco, Alejandro

Tutor/a: Valderas Aranda, Pedro José

CURSO ACADÉMICO: 2023/2024

Resum

Este projecte ha tingut com a objectiu la migració de la web de l'empresa de psicologia en línia PROSAM, des d'una plataforma basada en Wordpress cap a una nova implementació utilitzant el framework *Astro*. La migració s'ha dut a terme per a millorar la flexibilitat i personalització de la web, integrant funcions que abans depenien de servicis de pagament externs. Seguint una metodologia incremental, s'han incorporat millores en la interfície d'usuari, un sistema propi de gestió de cites i opcions d'inici de sessió mitjançant tercers. La solució final utilitza tecnologies modernes com *Astro*, *React*, i *Tailwind CSS* en el *frontend*, mentres que *Astro DB* i *Astro Studio* s'empren en la gestió de la base de dades, amb el *backend* allotjat en *Vercel* i amb integracions per a pagaments i videoconferències.

Paraules clau: Migració, Web, Psicologia, WordPress, Astro

Resumen

Este proyecto ha tenido como objetivo la migración de la web de la empresa de psicología online PROSAM, desde una plataforma basada en WordPress hacia una nueva implementación utilizando el *framework Astro*. La migración se ha llevado a cabo para mejorar la flexibilidad y personalización de la web, integrando funciones que antes dependían de servicios de pago externos. Siguiendo una metodología incremental, se han incorporado mejoras en la interfaz de usuario, un sistema propio de gestión de citas y opciones de inicio de sesión mediante terceros. La solución final utiliza tecnologías modernas como *Astro*, *React*, y *Tailwind CSS* en el *frontend*, mientras que *Astro DB* y *Astro Studio* se emplean en la gestión de la base de datos, con el *backend* alojado en *Vercel* y con integraciones para pagos y videoconferencias.

Palabras clave: Migración, Web, Psicología, WordPress, Astro

Abstract

The aim of this project has been the migration of the website for the online psychology company PROSAM from a WordPress-based platform to a new implementation using the *Astro framework*. The migration was carried out to enhance the website's flexibility and customization by integrating functions that previously relied on external paid services. Following an incremental methodology, improvements were made to the user interface, a custom appointment management system was developed, and third-party login options were integrated. The final solution utilizes modern technologies such as *Astro*, *React*, and *Tailwind CSS* for the *frontend*, while *Astro DB* and *Astro Studio* are employed for database management, with the *backend* hosted on *Vercel* and integrated with payment and videoconferencing services.

Key words: Migration, Web, Psychology, WordPress, Astro

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Actualmente, el avance de la tecnología está facilitando el desarrollo de proyectos en diversos campos. El desarrollo de una página web es una tarea que se ha simplificado considerablemente, gracias a la disponibilidad de herramientas y recursos en línea. Sin embargo, mantener una web actualizada y optimizada requiere una constante adaptación a nuevas tecnologías y metodologías. Este proyecto se centra en la migración de una página web construida con WordPress a una implementación más moderna utilizando *Astro* [1].

La migración a esta tecnología responde a las necesidades de la empresa PROSAM (Proyecto Salud Mental) [2], actualmente, la empresa valenciana cuenta con una implementación en WordPress, lo que supone bastantes desventajas que veremos más adelante, además de que está ligada a un servicio de pago externo como es Eholo [3], un servicio de gestión de calendarios para psicólogos.

1.1 Objetivos

El principal objetivo planteado para este trabajo es realizar la migración completa de la web, manteniendo las funcionalidades de la web original, garantizando la integridad y continuidad del servicio, además de añadir una serie de funcionalidades siguiendo la metodología Incremental. Para alcanzar este objetivo, se plantearon los siguientes objetivos específicos:

1. **Migrar la interfaz de usuario:** Actualizar la interfaz de usuario a una versión más moderna y accesible, utilizando tecnologías como *Astro* y *Tailwind CSS*. Mejorando así la experiencia del usuario.

2. **Reimplementar la gestión de citas:** Adaptar la funcionalidad de gestión de citas, previamente dependiente del servicio externo Eholo, a una solución basada en *React*.

Además, se plantean los siguientes objetivos adicionales:

1. **Implementar un sistema de gestión de citas para psicólogos:** Desarrollar un sistema integral que permita a los psicólogos no solo autenticarse de manera segura, sino también gestionar sus citas de forma eficiente. Se incluyen también funcionalidades como la visualización de citas reservadas, la gestión de disponibilidad, el acceso a un calendario personalizado entre otras.

2. **Integrar la opción de inicio de sesión con Google:** Añadir la funcionalidad de autenticación mediante cuentas de Google, facilitando el proceso de registro e inicio de sesión para los usuarios.

3. **Ampliar el formulario de contacto:** Incluir una opción para que los psicólogos interesados puedan adjuntar su currículum vitae. Para esto, se utilizará el servicio de almacenamiento de *Cloudinary* [4], permitiendo un manejo seguro y eficiente de los documentos adjuntos.

Estos objetivos buscan no solo realizar la migración de la web, sino también añadir valor a la aplicación y demostrar los conocimientos adquiridos durante el transcurso de la carrera, especialmente en el área de Tecnologías de la Información, a través del desarrollo de un proyecto *fullstack* [1].

1.2 Estructura de la memoria

Para lograr una comprensión clara y detallada del proceso y los resultados obtenidos, el resto de la memoria se estructura de la siguiente manera:

- **Contexto Previo:** Se presenta la empresa para la que se realizado la migración, explicando la situación inicial y las nuevas necesidades que surgieron.
- **Metodología:** Se detalla la metodología de desarrollo utilizada, haciendo referencia a su ajuste al modelo incremental.
- **Análisis de Requisitos:** En este punto se explica cómo se han identificado los nuevos requisitos que debe soportar la aplicación, se describen los requisitos que debe cumplir el sistema además de presentar los casos de uso.
- **Análisis Conceptual y Diseño:** Se incluye el diseño conceptual de la base de datos, se presenta el modelo de la base de datos y se muestran los bocetos de las principales interfaces de la aplicación.
- **Desarrollo de la solución:** Este apartado detalla la arquitectura del sistema, explicando en profundidad los componentes principales que conforman tanto el *frontend* como el *backend*. Se analiza cómo se comunican entre sí utilizando diferentes protocolos, como conexiones HTTP y APIs. También se incluye una breve descripción de las herramientas y lenguajes de programación utilizados, acompañada de ejemplos que muestran cómo se integran para lograr una implementación eficiente.
- **Producto desarrollado:** Se presenta la aplicación final mediante capturas de pantalla, explicando cómo se utilizan los escenarios descritos en la sección de análisis de requisitos.
- **Validación:** Se realiza una validación del producto, utilizando una evaluación heurística de la usabilidad de la interfaz.
- **Conclusiones y trabajos futuros:** Se resumen los trabajos realizados, se presenta una opinión personal y se discuten posibles mejoras y evoluciones futuras del proyecto.

[1] El término "fullstack"se refiere a un perfil de desarrollador de software que tiene la capacidad de trabajar tanto en el *frontend* (la parte visible de una aplicación web con la que interactúan los usuarios) como en el *backend* (la parte que gestiona la lógica del servidor, bases de datos y otras funcionalidades no visibles para el usuario). Un desarrollador fullstack posee habilidades en múltiples tecnologías y puede gestionar una aplicación completa, desde la interfaz de usuario hasta la infraestructura del servidor.

En este trabajo, se ha realizado una búsqueda exhaustiva de información, combinando el conocimiento previo con nuevos aprendizajes para llevar a cabo el proyecto con la calidad deseada y asegurar su correcto funcionamiento. Este documento detalla el proceso de desarrollo y los resultados obtenidos, ofreciendo una visión completa de la migración de la web al *framework* [2] *Astro*.

[2]framework: Un framework es una estructura o marco que proporciona una base para desarrollar un proyecto con objetivos concretos. Funciona como una plantilla que facilita la organización y el desarrollo de software, sirviendo como punto de partida para estructurar y dirigir el trabajo.

CAPÍTULO 2
Contexto previo

La empresa valenciana PROSAM (Proyecto Salud Mental), nace con el objetivo de eliminar las barreras económicas que dificultan el acceso a la psicología. Con el eslogan "La salud mental un derecho, no un lujo", PROSAM se ha comprometido a hacer la salud mental accesible para todos. En un contexto donde los problemas mentales y la ansiedad son cada vez más frecuentes, la empresa decidió implementar su servicio de manera online, respaldándose de la evidencia científica [5], la cual confirma que la calidad de los resultados es comparable a la de las sesiones presenciales, brindando así mayor libertad a los clientes.

Actualmente, PROSAM utiliza una web basada en WordPress. Esta plataforma, aunque es popular y versátil, presenta ciertas limitaciones en términos de flexibilidad y personalización. Debido a que WordPress depende en gran medida de plugins y servicios externos, la capacidad de adaptar y expandir la web para satisfacer las necesidades específicas de la empresa es limitada.

En cuanto a las funcionalidades actuales de la web, se encuentran reflejadas en los siguientes casos de uso [6]:

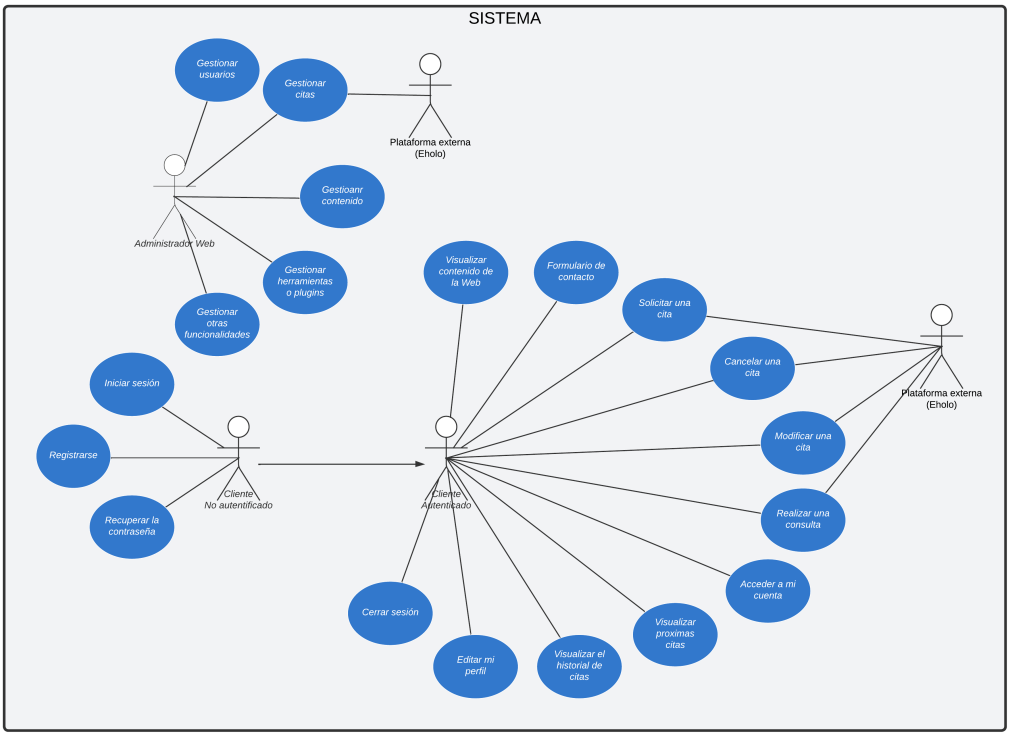

Figura 2.1: Diagrama UML de Casos de uso (Antes de la migración)

A medida que crecía la demanda, también lo hacía la necesidad de ofrecer una mayor independencia en la administración de la plataforma, por lo que surgieron nuevas necesidades:

- **Eliminación de dependencias externas:** La dependencia de WordPress y Eholo limita la capacidad de personalización y la eficiencia operativa. La empresa busca una solución que permita integrar todas las funcionalidades dentro de su propio sistema.
- **Gestión interna de horarios:** Se necesita un sistema donde los psicólogos puedan iniciar sesión, gestionar sus horarios directamente desde la web de PROSAM, sin depender de servicios externos.
- **Sincronización de calendarios:** La nueva plataforma debe permitir que los calendarios de los psicólogos estén sincronizados con un calendario principal, accesible por los clientes para una mejor organización y gestión de citas.
- **Mejora de la interfaz de usuario:** Además de las funcionalidades técnicas, es crucial mejorar la interfaz de usuario para que sea más intuitiva y atractiva, ofreciendo una mejor experiencia para los clientes.

Por lo que el objetivo de este proyecto es desarrollar una nueva página web que cumpla con estas necesidades. Optando tecnologías modernas y sin recurrir a WordPress ni servicios externos.

CAPÍTULO 3
Metodología

El enfoque incremental [7] de gestión de proyectos se caracteriza por su enfoque en el crecimiento incremental de las funciones. Este método, en un principio utilizado para proyectos de software y posteriormente aplicado a otros campos, establece entregas parciales. En cada una de estas fases, el producto debe mostrar una evolución con respecto a la fecha anterior, asegurando que nunca sea igual y que se acerque cada vez más a lo requerido por el cliente.

Desde el inicio de mi TFG, he utilizado una metodología incremental adaptada para asegurar que el desarrollo de la nueva plataforma web de PROSAM se realizara de manera ordenada y efectiva. Aunque no seguí todas las fases del modelo incremental de manera estricta, me enfoqué en un proceso iterativo y en la comunicación constante con los responsables de PROSAM y mi tutor, Pedro José Valderas. Las fases clave que seguí fueron las siguientes:

- **Requerimientos:** Definición de los objetivos centrales y específicos del proyecto. En esta fase, se realizaron reuniones iniciales donde se discutieron las necesidades y expectativas de los responsables de PROSAM. Basándome en estas conversaciones, se establecieron los requisitos que la nueva plataforma debía cumplir.
- **Definición de las tareas y las iteraciones:** Se elaboró una lista de tareas basadas en los requerimientos discutidos, que luego se agruparon en iteraciones, cada una enfocada en desarrollar uno o más incrementos específicos, asegurando un progreso constante hacia la meta final.
- **Desarrollo del incremento:** Una vez confirmados los cambios o funcionalidades con mi tutor y los responsables de PROSAM, procedí a implementar las tareas acordadas. Cada iteración se centró en desarrollar uno o varios de los incrementos identificados, como se detalla más adelante, asegurando un progreso constante hacia la meta final.
- **Validación de incrementos:** Al final de cada iteración, se evaluaron los incrementos obtenidos. Este proceso se realizó mediante reuniones donde presentaba los avances a PROSAM y a mi tutor. Si los resultados no eran los esperados, se identificaban las causas y se implementaban las soluciones necesarias para corregir el curso del proyecto.
- **Integración de incrementos:** Los incrementos validados se integraron progresivamente, contribuyendo de manera coherente al resultado final del proyecto. A continuación, se enumeran los incrementos definidos y su contribución al proyecto:

    1. **Configuración inicial y Estructura base del proyecto**

2. **Desarrollo del *frontend***
3. **Autenticación y Gestión de Usuarios**
4. **Desarrollo del *backend***
5. **Optimización y Mejora de la Interfaz de Usuario**
6. **Integración de Servicios de Terceros**
7. **Depuración y Preparación para el Despliegue**

- **Entrega del producto:** Tras la integración exitosa de todos los incrementos, y una vez validado el producto final, se confirmó que cumplía con los objetivos iniciales, lo que permitió proceder a su entrega final.

La metodología incremental me ha permitido una gestión sencilla de las tareas en cada ciclo y una alta adaptabilidad a los cambios o modificaciones que surgieron durante el desarrollo del proyecto. Este enfoque ha sido esencial para mantener un progreso constante y asegurarse de que el producto final cumpliera con las expectativas de PROSAM.

CAPÍTULO 4
Análisis de requisitos

En este capítulo se detallan los pasos previos necesarios para el desarrollo de la plataforma web. Inicialmente, se realizará una identificación exhaustiva de los requisitos que el proyecto debe cumplir, tanto aquellos nuevos que se implementarán como los ya existentes que serán reimplementados. A partir de esta captura de requisitos, se definirán los casos de uso que guiarán el desarrollo de las funcionalidades clave de la aplicación. Los casos de uso ofrecerán una perspectiva clara de cómo los diferentes actores interactuarán con el sistema, lo que servirá posteriormente de base para el diseño de las interfaces.

4.1 Captura de requisitos

Con el objetivo de determinar los nuevos requisitos que debía soportar la aplicación, trabajé en estrecha colaboración con dos responsables de PROSAM, quienes actuaron como mis clientes durante todo el proceso. A lo largo del desarrollo, mantuvimos un total de cuatro reuniones en las que discutimos los cambios y adiciones que querían implementar en la web. En lugar de utilizar cuestionarios formales, optamos por un enfoque de diálogo continuo, lo que permitió una comunicación fluida y adaptativa. Cada necesidad discutida durante estas reuniones fue posteriormente validada con mi tutor, aplicando un enfoque incremental que garantizó la alineación de los requisitos con los objetivos del proyecto.

Uno de los requisitos que se descartó durante este proceso fue la implementación de consultas de psicología directamente dentro de la web. Esta decisión se tomó debido a la complejidad de integrar un sistema de reuniones (Zoom, Google Meet, Teams, etc) en línea en la plataforma actual. No obstante, se dejó abierta la posibilidad de considerar esta funcionalidad para futuros desarrollos.

4.2 Requisitos iniciales

Para iniciar el desarrollo de la aplicación, es fundamental tener una comprensión clara de los requisitos iniciales que el proyecto debe cumplir. Estos requisitos se dividen en dos categorías, los nuevos requisitos que se implementarán y los requisitos existentes que serán reimplementados:

Requisitos nuevos:

- **Autenticación de psicólogos:** Se debe implementar un sistema que permita a los psicólogos iniciar sesión.

- **Calendarios sincronizados:** Cada psicólogo debe poder administrar su calendario, pudiendo así manejar su disponibilidad de huecos. Los calendarios de los psicólogos deben estar sincronizados con el calendario que se muestra al cliente.
- **Diseño Responsive:** El diseño de la página web tiene que ser Responsive, lo cual implica que sea accesible y adaptable en todos los dispositivos (tabletas, smartphones, etc).
- **Formulario de contacto:** Se añade la opción adjuntar currículum vitae (en formato PDF) en caso de que haya un psicólogo interesado en trabajar en PROSAM.

Requisitos reimplementados:

- **Confirmación de cita:** Se mandará un correo electrónico con los detalles de la sesión.
- **Mantenimiento del dominio:** Se debe mantener el dominio que utiliza PROSAM.
- **Cambios en la interfaz:** Pequeñas modificaciones sobre las secciones principales de la web.

Una vez definidos los requisitos iniciales, se puede proceder a la creación de los casos de uso para la aplicación.

4.3 Casos de uso

El Diagrama UML presentado a continuación muestra los casos de uso de la aplicación web después de la migración. Este diagrama permite identificar de manera clara los roles y funcionalidades implementados, destacando con color negro aquellos casos de uso que son nuevos y en azul los que ya existían. Cada caso de uso se definirá y describirá en detalle posteriormente, proporcionando una visión integral del comportamiento esperado del sistema en las distintas fases del proyecto.

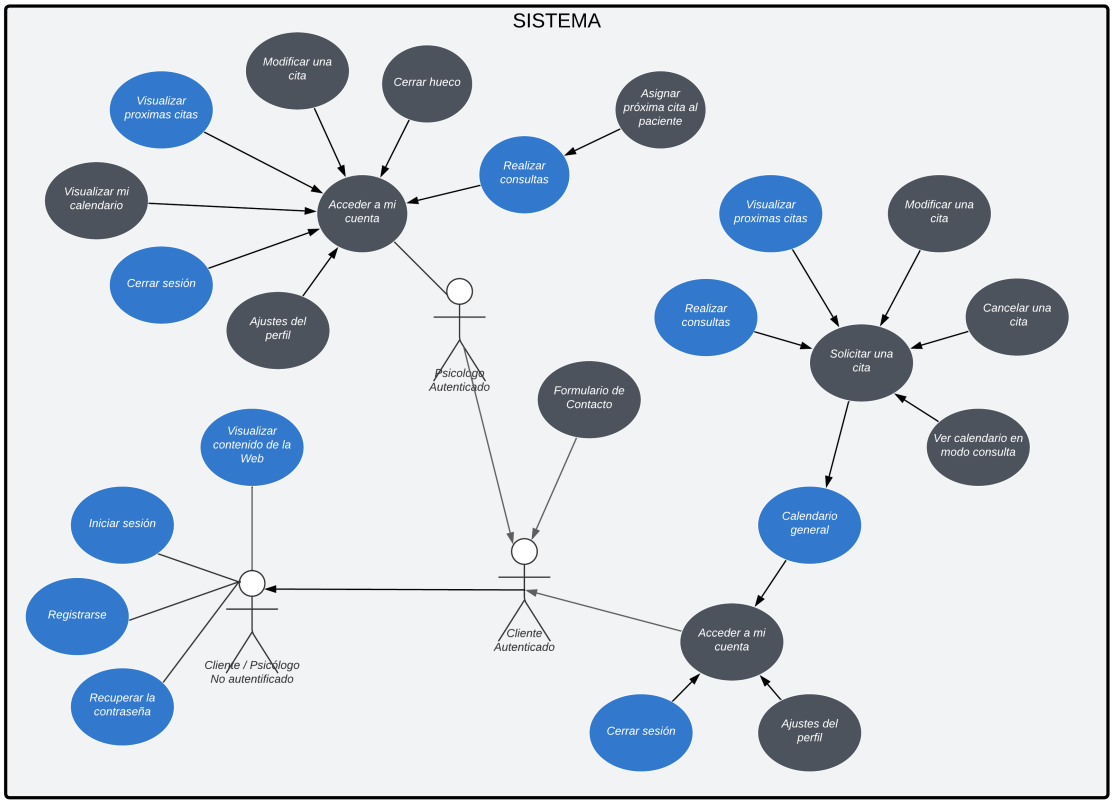

Figura 4.1: Diagrama UML de Casos de uso (Tras la migración)

En el Diagrama UML (Figura 4.1) no está representado el rol de Administrador, el cual aparecía en la (Figura 2.1), ya que la única diferencia es que el Administrador ahora puede gestionar tanto clientes como psicólogos y que ya no utiliza una plataforma externa (Eholo) para ello. Como en esta segunda solución se ha implementado la figura del psicólogo el autor Cliente no autenticado pasa a ser Cliente / Psicólogo no autenticado. Acto seguido procedemos a definir los casos de uso, concretamente los del segundo Diagrama UML, ya que serán las funcionalidades de la aplicación web:

- **Caso de uso: Registrarse**
  - Precondición: Tiene que ser un cliente no autenticado y no registrado anteriormente.
  - Descripción: El cliente se registra completando los campos de email y contraseña.
  - Secuencia:
    1. El cliente accede a la página de Inicio de sesión y hace click en "¿Ya tienes una cuenta? Regístrate".
    2. El cliente rellena el formulario y completa el registro.
  - Postcondición: Se comprueban los campos rellenados y se redirige a la sección de "Mi cuenta".
  - Excepciones: Ninguna.
- **Caso de uso: Iniciar sesión**
  - Precondición: Tiene que ser un cliente/psicólogo no autenticado y debe estar registrado.
  - Descripción: El cliente/psicólogo inicia sesión completando los campos de email y contraseña.
  - Secuencia:
    1. El cliente/psicólogo accede a la página de Inicio de sesión.
    2. El cliente/psicólogo rellena el formulario y completa el inicio de sesión.
  - Postcondición: Se comprueban los campos rellenados y se redirige a la sección de "Mi cuenta".
  - Excepciones: Ninguna.
- **Caso de uso: Recuperar contraseña**
  - Precondición: Tiene que ser un cliente/psicólogo no autenticado y debe estar registrado.
  - Descripción: El cliente/psicólogo recupera su contraseña completando los campos necesarios.
  - Secuencia:
    1. El cliente/psicólogo accede a la página de Inicio de sesión y hace click en "¿Has olvidado tu contraseña?".
    2. El cliente/psicólogo introduce su email registrado.
    3. El sistema envía un enlace de recuperación de contraseña al email proporcionado.
    4. El cliente/psicólogo accede al enlace enviado por email.
    5. El cliente/psicólogo introduce una nueva contraseña y la confirma.

6. El cliente/psicólogo completa el proceso de recuperación de contraseña.

- Postcondición: Se comprueba la validez del email y de la nueva contraseña, y se actualiza la contraseña en el sistema.
- Excepciones:
  - El email proporcionado no está registrado.
  - El enlace de recuperación ha expirado.
  - El enlace de recuperación ya ha sido utilizado.

- **Caso de uso: Visualizar contenido de la web**
  - Precondición: Ninguna; cualquier usuario puede acceder.
  - Descripción: El cliente/psicólogo visualiza el contenido de la web.
  - Secuencia:
    1. El cliente/psicólogo accede a la página principal de la web.
    2. El cliente/psicólogo navega por las diferentes secciones de la web (Inicio, La clínica, Terapias, Pedir cita, Iniciar sesión...).
  - Postcondición: El contenido solicitado por el usuario se muestra correctamente en la pantalla.
  - Excepciones:
    - La página solicitada no existe (error 404).
    - Error en el servidor al cargar el contenido (error 500).
    - La conexión a Internet del cliente/psicólogo falla.
    - El contenido solicitado está restringido y requiere autenticación (Pedir cita).

- **Caso de uso: Acceder a mi cuenta**
  - Precondición: El cliente/psicólogo debe estar autenticado.
  - Descripción: El cliente/psicólogo accede a la sección "Mi cuenta", donde puede ver su información personal, reservar una cita o consultar información sobre sus próximas citas.
  - Secuencia:
    1. El cliente/psicólogo inicia sesión o se registra en caso de que sea un nuevo usuario.
    2. El cliente/psicólogo es redirigido directamente a la sección "Mi cuenta".
  - Postcondición: El cliente/psicólogo visualiza la sección correctamente.
  - Excepciones:
    - El cliente/psicólogo introduce credenciales incorrectas.

- **Caso de uso: Cerrar sesión**
  - Precondición: Tiene que ser un cliente/psicólogo autenticado y debe estar en la página "Mi cuenta".
  - Descripción: El cliente/psicólogo cierra su sesión actual desde la sección "Mi cuenta".
  - Secuencia:
    1. El cliente/psicólogo hace click en el botón “Cerrar sesión”.
    2. El sistema termina la sesión del cliente/psicólogo.

3. El sistema redirige al cliente/psicólogo a la página de inicio de sesión.

- Postcondición: La sesión del cliente/psicólogo se cierra correctamente y el usuario es redirigido a la página de inicio de sesión.
- Excepciones: Ninguna.

- **Caso de uso: Ajustes del perfil**
  - Precondición: El cliente/psicólogo debe estar autenticado y haber accedido a la sección "Mi cuenta".
  - Descripción: El cliente/psicólogo edita su perfil, actualizando información personal.
  - Secuencia:
    1. El cliente/psicólogo accede a la sección "Mi cuenta".
    2. El cliente/psicólogo selecciona la opción “Ajustes”.
    3. El cliente/psicólogo modifica los campos de información personal (avatar, nombre, email, contraseña).
  - Postcondición: La información del perfil del usuario se actualiza correctamente.
  - Excepciones:
    - El usuario introduce un email ya registrado.
    - El usuario introduce una contraseña que no cumple con los requisitos de seguridad.
    - El usuario deja campos obligatorios vacíos.
    - El tamaño de la imagen de avatar excede el máximo.

- **Caso de uso: Calendario general**
  - Precondición: El cliente debe estar autenticado y haber accedido a la sección "Mi cuenta".
  - Descripción: El cliente visualiza la disponibilidad del sistema mediante un calendario, donde se muestran los calendarios combinados de todos los psicólogos.
  - Secuencia:
    1. El cliente accede a la sección "Mi cuenta".
    2. El cliente selecciona un día del calendario.
    3. El sistema muestra los huecos disponibles, estos son mostrados mediante el algoritmo de Round-Robin para una asignación equitativa de las citas.
  - Postcondición: El cliente visualiza correctamente la disponibilidad del sistema.
  - Excepciones: Ninguna.

- **Caso de uso: Visualizar mi calendario**
  - Precondición: El psicólogo debe estar autenticado y haber accedido a la sección "Mi cuenta".
  - Descripción: El psicólogo visualiza su disponibilidad mediante un calendario.
  - Secuencia:
    1. El psicólogo accede a la sección "Mi cuenta".
    2. El psicólogo selecciona un día del calendario.
    3. El sistema muestra los huecos disponibles para el psicólogo autenticado.

- Postcondición: El psicólogo visualiza correctamente su disponibilidad.
- Excepciones: Ninguna.

- **Caso de uso: Solicitar primera cita**
  - Precondición: El cliente debe estar autenticado y haber accedido a la sección "Mi cuenta".
  - Descripción: El cliente selecciona un hueco entre los disponibles en el calendario general.
  - Secuencia:
    1. El cliente accede a la sección "Mi cuenta".
    2. El cliente selecciona un día del calendario.
    3. El cliente selecciona un hueco de los disponibles.
    4. El sistema muestra una pantalla de confirmación.
    5. El sistema redirige a la plataforma de pagos Stripe [8].
    6. El cliente completa el pago de la cita.
    7. Se envía la confirmación de la cita para el cliente/psicólogo por email junto al enlace a la videoconferencia.
  - Postcondición: El cliente/psicólogo visualiza la nueva cita en la sección de "Mi-cuenta".
  - Excepciones:
    - Error en el proceso de pago (tarjeta rechazada, problemas con el servicio de pago).

- **Caso de uso: Visualizar próximas citas**
  - Precondición: El cliente/psicólogo debe estar autenticado y haber accedido a la sección "Mi cuenta", además de tener una cita reservada.
  - Descripción: El cliente/psicólogo visualiza las próximas citas.
  - Secuencia:
    1. El cliente/psicólogo accede a la sección "Mi cuenta".
    2. El cliente/psicólogo selecciona la opción "Visualizar próximas citas".
    3. El sistema muestra una lista de citas con detalles como fecha, hora, psicólogo y paciente.
  - Postcondición: El cliente/psicólogo visualiza correctamente la información de sus próximas citas.
  - Excepciones: Ninguna.

- **Caso de uso: Modificar cita (Psicólogo)**
  - Precondición: El psicólogo debe estar autenticado, haber accedido a la sección "Mi cuenta“ y estar en la sección de ”Citas".
  - Descripción: El psicólogo modifica los detalles de una cita futura.
  - Secuencia:
    1. El psicólogo accede a la sección "Mi cuenta".
    2. El psicólogo selecciona la opción "Próximas citas".
    3. El psicólogo elige la cita que desea modificar.
    4. El psicólogo cambia los detalles de la cita (fecha, hora).
    5. El psicólogo guarda los cambios.

6. El sistema actualiza la cita en la base de datos.

- Postcondición: La cita se actualiza correctamente y la lista de próximas citas refleja los cambios.
- Excepciones: Ninguna

- **Caso de uso: Modificar cita (Cliente)**
  - Precondición: El cliente debe estar autenticado, haber accedido a la sección “Mi cuenta” y estar en la sección de "Visualizar próximas citas".
  - Descripción: El cliente solicita al psicólogo que quiere cambiar su cita por correo psicologo@proyectosaludmental.com.
  - Secuencia:
    1. El cliente accede a la sección "Mi cuenta".
    2. El cliente selecciona la opción "Próximas citas".
    3. El cliente elige la cita que desea modificar.
    4. El cliente envía un correo al psicólogo asignado indicando su disponibilidad horaria para realizar el cambio.
  - Postcondición: El psicólogo recibe el mensaje y cambia la cita.
  - Excepciones: Ninguna

- **Caso de uso: Cancelar cita**
  - Precondición: El cliente debe estar autenticado, haber accedido a la sección “Mi cuenta”, haber reservado una cita y que falten 24 horas o más para realizar la sesión.
  - Descripción: El cliente cancela una cita reservada.
  - Secuencia:
    1. El cliente accede a la sección "Mi cuenta".
    2. El cliente selecciona la opción "Próximas citas".
    3. El cliente elige la cita que desea cancelar.
    4. El sistema marca la cita como cancelada en la base de datos.
  - Postcondición: La cita se cancela correctamente, en la lista de próximas citas aparece como “Cancelada” y el hueco no se abre en el calendario.
  - Excepciones: Ninguna.

- **Caso de uso: Cerrar huecos**
  - Precondición: El psicólogo debe estar autenticado, haber accedido a la sección "Mi-cuenta".
  - Descripción: El psicólogo cierra un o más huecos de disponibilidad y se ve reflejado en el calendario.
  - Secuencia:
    1. El psicólogo accede a la sección "Mi-cuenta".
    2. El psicólogo selecciona fecha y hora (u horas) que quiere cerrar.
    3. El psicólogo pulsa el botón de ”Cerrar huecos".
  - Postcondición: El/Los hueco/s es/son cerrado/s correctamente y se ven reflejados los cambios en el calendario del psicólogo.
  - Excepciones: Ninguna.

- **Caso de uso: Realizar consulta**
  - Precondición: El cliente debe haber solicitado una cita previamente y estar autenticado. El psicólogo también debe estar autenticado y que le haya la cita.
  - Descripción: El cliente y el psicólogo realizan la sesión de terapia de psicología.
  - Secuencia:
    1. Tanto el cliente como el psicólogo acceden a la sección “Visualizar próximas citas” o al enlace proporcionado en el correo electrónico, el cual contiene el nombre del cliente/psicólogo, fecha, hora y enlace a la cita.
    2. El cliente y el psicólogo acceden al enlace de la cita.
    3. El cliente y el psicólogo realizan la sesión de terapia.
  - Postcondición: La sesión de terapia se completa exitosamente.
  - Excepciones:
    - El cliente no se presenta a la sesión (puede cancelar hasta 24 horas antes de la cita).
- **Caso de uso: Asignar próxima cita**
  - Precondición: El cliente y el psicólogo deben haber completado una consulta previa con éxito.
  - Descripción: Tras completar una sesión de terapia, el psicólogo y el cliente agendan una próxima cita, la cual incluye el proceso de pago y la confirmación de la nueva cita.
  - Secuencia:
    1. Tras la finalización de la consulta, el psicólogo y el cliente deciden agendar una próxima cita.
    2. El psicólogo proporciona al cliente el enlace para el pago de la próxima cita.
    3. El cliente accede al enlace y completa el pago.
    4. El sistema confirma el pago y actualiza la base de datos con los detalles de la nueva cita.
    5. El sistema envía un correo electrónico al cliente y al psicólogo con los detalles de la nueva cita (nombre del cliente, fecha, hora y enlace a la cita).
  - Postcondición: La nueva cita se registra correctamente en el sistema y es visible en “Mi cuenta” del cliente. Ambos, cliente y psicólogo, reciben una confirmación por correo electrónico con los detalles de la próxima cita.
  - Excepciones: Ninguna.
- **Caso de uso: Calendario en modo consulta**
  - Precondición: El cliente debe haber realizado su primera cita y tener un psicólogo asignado.
  - Descripción: Tras completar la primera sesión de terapia, el cliente cuando accede a la sección de "Mi-cuenta” observa el calendario en modo consulta, ya que, según la lógica que hemos seguido el resto de citas se las asigna su psicólogo.
  - Secuencia:
    1. El cliente accede a la sección de "Mi-cuenta".
    2. Visualiza el calendario en modo lectura (está bloqueada la posibilidad de pedir una cita).

- Postcondición: El cliente visualiza correctamente el calendario en modo lectura sin poder realizar ninguna acción, ya que tiene un psicólogo asignado.
- Excepciones: Ninguna.

Caso de uso: Formulario de contacto

- Precondición: El cliente/psicólogo debe estar autenticado.
- Descripción: El usuario rellena y envía el formulario de contacto, proporcionando su nombre, email y mensaje. Además, si el usuario es un psicólogo interesado en trabajar con la empresa, puede adjuntar su CV.
- Secuencia:
  1. El cliente/psicólogo accede a la sección principal "Inicio".
  2. El cliente/psicólogo hace scroll hasta llegar al formulario de contacto.
  3. El cliente/psicólogo rellena los campos obligatorios: Nombre, Email y Mensaje.
  4. Si el usuario es un psicólogo interesado en trabajar con la empresa, adjunta su CV (opcional).
  5. El sistema verifica que los campos obligatorios estén completos y sean válidos.
  6. El sistema muestra una confirmación al usuario de que el mensaje ha sido enviado correctamente.
- Postcondición: El mensaje se envía correctamente al equipo de soporte o recursos humanos y el usuario recibe una confirmación de envío.
- Excepciones: Ninguna.

CAPÍTULO 5
Análisis Conceptual y Diseño

Este capítulo abarca el análisis conceptual y la elaboración del diseño para la solución propuesta para la plataforma web. Comenzaremos describiendo el modelo de dominio, representado a través de un diagrama de clases que organiza las principales entidades del sistema y sus relaciones. Posteriormente, se detallará el modelo de la base de datos, especificando las tablas y su estructura, fundamentales para gestionar la información de usuarios, citas, sesiones y otros elementos clave dentro de la aplicación. Finalmente, se abordará el diseño de la interfaz de usuario, comparando el estado previo a la migración con los nuevos bocetos desarrollados, los cuales reflejan las mejoras y requisitos implementados en el proyecto.

5.1 Modelo de dominio

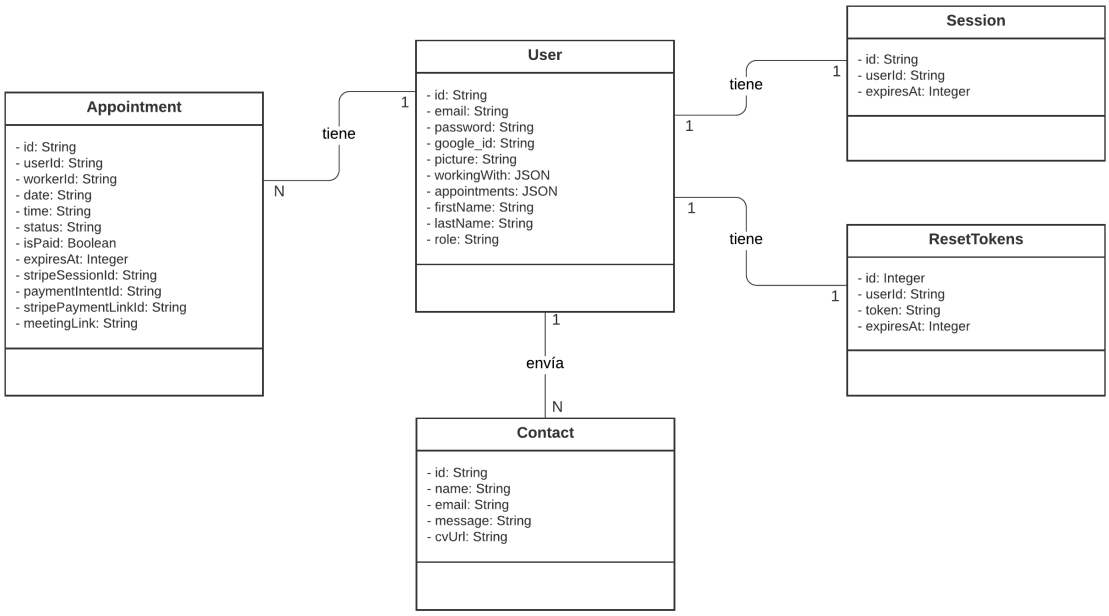

Figura 5.1: Diagrama de clases del sistema

El diagrama de clases que se muestra en la figura anterior organiza las entidades principales del sistema y cómo se relacionan entre sí. El núcleo del sistema es la clase *User*, que representa tanto a los pacientes como a los psicólogos, distinguidos por un atributo de rol. Dependiendo de este rol, los usuarios tienen diferentes interacciones dentro del sistema. Por ejemplo, un paciente puede tener múltiples *Appointment*, cada una asociada a un psicólogo específico. Del otro lado, un psicólogo estará relacionado con las citas

que tiene agendadas, mostrando la flexibilidad del modelo al permitir que ambas partes compartan la misma clase *User*, mientras que *Appointment* gestiona las relaciones entre los participantes en cada encuentro.

Cada usuario tiene asociada una única sesión activa, representada por la clase *Session*. Esta clase se encarga de manejar la autenticación y mantener la seguridad del sistema al definir tiempos de expiración, garantizando que solo los usuarios con sesiones válidas puedan interactuar con la plataforma. Además, el modelo incluye la clase *ResetTokens*, diseñada para gestionar los tokens utilizados en la recuperación de contraseñas. Esta clase tiene una relación uno a uno con *User*, lo que significa que cada usuario puede generar un único token para la recuperación de su contraseña en un momento dado.

Finalmente, la clase *Contact* en la parte inferior del diagrama muestra que un usuario puede enviar múltiples mensajes de contacto. Cada mensaje está relacionado con el usuario que lo envió e incluye detalles importantes como el nombre del remitente, su correo electrónico y el contenido del mensaje. Este diseño permite una gestión eficiente de las comunicaciones dentro de la plataforma.

5.2 Modelo de la base de datos

A continuación se ofrece una descripción detallada de las distintas tablas que conforman la base de datos, las cuales están diseñadas para manejar la gestión de usuarios, citas, sesiones, y mensajes de contacto dentro de la aplicación.

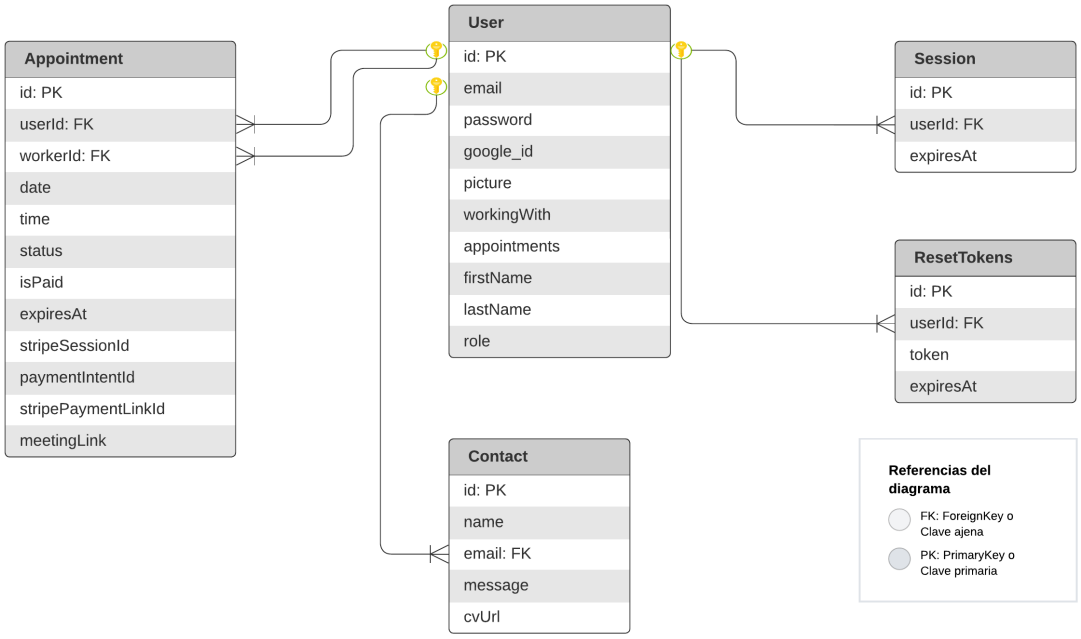

Figura 5.2: Diagrama de base de datos

- **User:** Esta tabla almacena los usuarios y sus datos esenciales para el acceso a la aplicación, asignación de roles específicos y gestión de la disponibilidad de citas.
  - **id:** Identificador único del usuario en la base de datos.
  - **email:** Dirección de correo electrónico del usuario, utilizada como identificador único.
  - **password:** Contraseña encriptada del usuario para la autenticación.
  - **googleId:** Identificador único del usuario en caso de autenticación con Google.

  - **picture:** URL de la imagen de perfil del usuario.
  - **workingWith:** Este campo JSON gestiona las relaciones del usuario con otros usuarios. En el caso de un *patient*, almacena el correo electrónico del *worker* (psicólogo) asignado. Para un *worker*, puede contener una lista de correos electrónicos de los *patients* (pacientes) asignados.
  - **appointments:** Registro en formato JSON que almacena la disponibilidad del *worker*. Este campo contiene los huecos no disponibles para citas. Si el campo está vacío, se asume que el *worker* tiene disponibilidad completa dentro de un rango de 30 días vista.
  - **firstName:** Nombre del usuario.
  - **lastName:** Apellidos del usuario.
  - **role:** Rol asignado al usuario en la aplicación, que puede ser *patient*, *worker*, o *admin*.

- **Appointment:** Esta tabla almacena las citas reservadas entre los usuarios (*patients*) y los psicólogos (*workers*).
  - **id:** Identificador único de la cita en la base de datos.
  - **userId:** Referencia al identificador único del usuario (*patient*) que ha programado la cita.
  - **workerId:** Referencia al identificador único del psicólogo (*worker*) asignado a la cita.
  - **date:** Fecha de la cita.
  - **time:** Hora de la cita.
  - **status:** Estado actual de la cita, que puede ser *pending*, *completed*, o *canceled*.
  - **isPaid:** Indicador booleano que especifica si la cita ha sido pagada o no.
  - **expiresAt:** *Timestamp* que indica cuándo expira la cita si no se ha pagado. Gestiona la validez temporal de las citas pendientes, asegurando que, si no se paga en 15 minutos, la cita expira, liberando el espacio en la agenda del psicólogo.
  - **stripeSessionId:** Identificador único de la sesión de pago en Stripe asociada con la cita, utilizado para la gestión del pago.
  - **paymentIntentId:** Identificador único del intento de pago en Stripe, utilizado para procesar reembolsos en caso de cancelación de la cita.
  - **stripePaymentLinkId:** Identificador único del enlace de pago generado en Stripe para la cita. Este campo se combina con expiresAt para asegurar que el enlace de pago se desactive automáticamente si no se completa el pago dentro de los 15 minutos establecidos.
  - **meetingLink:** Enlace a la reunión virtual, generado tras la confirmación de la cita.

- **Session:** La tabla *Session* gestiona las sesiones de autenticación de los usuarios en la aplicación.
  - **id:** Identificador único de la sesión en la base de datos.
  - **userId:** Referencia al identificador único del usuario al que pertenece la sesión.
  - **expiresAt:** *Timestamp* que indica la fecha y hora de expiración de la sesión.

- **ResetTokens:** Esta tabla gestiona los tokens de recuperación de contraseñas, asegurando que los usuarios puedan restablecer su acceso de manera segura.

  - **id:** Identificador único del token de recuperación.
  - **userId:** Referencia al identificador del usuario que solicitó el restablecimiento.
  - **token:** Token generado para la recuperación de la contraseña.
  - **expiresAt:** *Timestamp* que indica la expiración del token.

- **Contact:** La tabla *Contact* almacena los mensajes enviados por los usuarios a través del formulario de contacto de la aplicación.

  - **id:** Identificador único del mensaje de contacto en la base de datos.
  - **name:** Nombre completo del usuario que envía el mensaje de contacto.
  - **email:** Dirección de correo electrónico del usuario que envía el mensaje, debe coincidir con un *email* registrado en la tabla *User*.
  - **message:** Contenido del mensaje enviado por el usuario.
  - **cvUrl:** Enlace a un archivo en formato PDF que contiene el CV (Curriculum Vitae) del usuario.

Para gestionar y manipular las tablas definidas en la base de datos, se ha optado por utilizar *Astro DB* [9], una solución de base de datos SQL diseñada específicamente para integrarse con el *framework Astro*. Esta herramienta permite definir, desarrollar y consultar datos de manera eficiente, ofreciendo un entorno de desarrollo optimizado gracias a su integración con *Astro Studio* [10]. *Astro Studio* facilita la visualización y administración de la base de datos directamente desde el navegador, lo que simplifica el trabajo con los datos en tiempo real.

Astro DB facilita el proceso de creación y mantenimiento de la base de datos mediante un enfoque basado en la definición de tablas y columnas, con soporte completo para tipado automático en TypeScript. Esto garantiza una mayor seguridad en el código y mejora el autocompletado durante el desarrollo. Además, *Astro DB* se apoya en *Drizzle ORM* [11] como su motor subyacente para gestionar las consultas y operaciones en la base de datos. Una ventaja clave de *Drizzle ORM* es que no requiere configuración ni instalación manual, ya que el cliente está incorporado en *Astro DB* y se configura automáticamente para comunicarse con la base de datos, ya sea local o remota. *Drizzle ORM* proporciona un generador de consultas SQL con tipado, que ayuda a mantener la integridad del código y facilita la interacción con la base de datos sin la necesidad de escribir SQL directamente, simplificando así la manipulación de datos y aumentando la productividad.

En conjunto, *Astro DB* y *Drizzle ORM* ofrecen una experiencia de desarrollo robusta y accesible, facilitando a los desarrolladores centrarse en la lógica de la aplicación sin tener que preocuparse por la complejidad de la gestión de la base de datos. Además, la capacidad de visualizar y gestionar las tablas directamente en *Astro Studio* mejora considerablemente el flujo de trabajo y la capacidad de depuración.

5.3 Diseño de la interfaz

Una vez establecidos los casos de uso en el capítulo anterior y la base de datos ya establecida, se procede al diseño de la interfaz de usuario que se implementará en el proyecto, siempre con un enfoque centrado en la experiencia del usuario. Para ello, se han creado bocetos de las pantallas más significativas utilizando la herramienta Lucidchart [12], lo que ha facilitado su desarrollo. Además, se incluirán capturas del estado de la aplicación antes de la migración, permitiendo una comparación clara con los nuevos bocetos. Posteriormente, en el capítulo "Producto desarrollado", se presentarán las interfaces definitivas.

5.3.1. Estado antes de la migración

A continuación, se exponen las capturas de las pantallas más relevantes de la web de la empresa PROSAM previas a la migración.

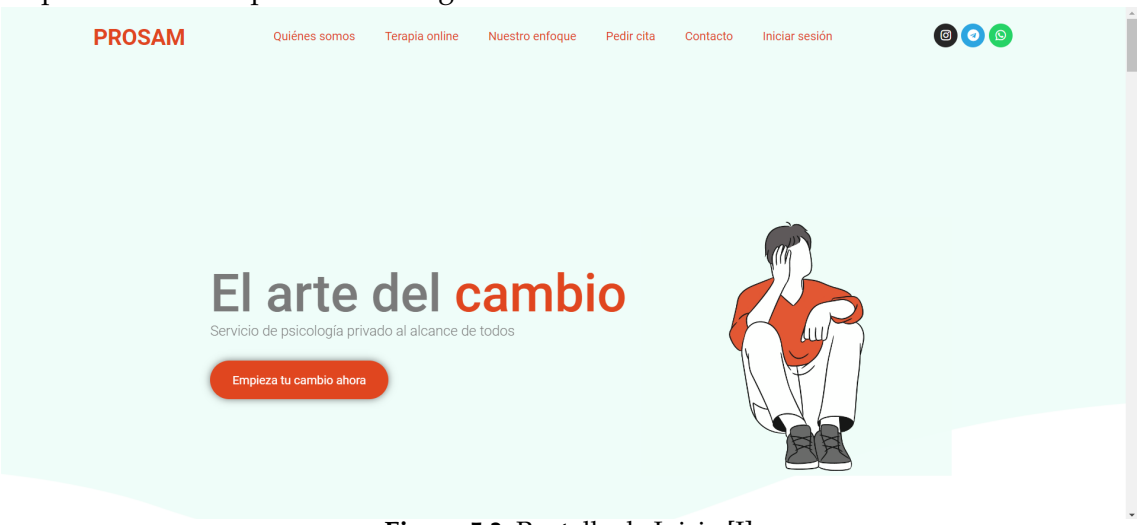

Figura 5.3: Pantalla de Inicio [I]

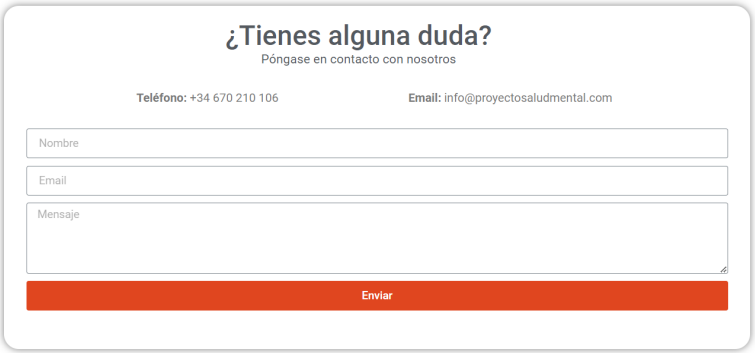

Figura 5.4: Pantalla de Inicio [II]

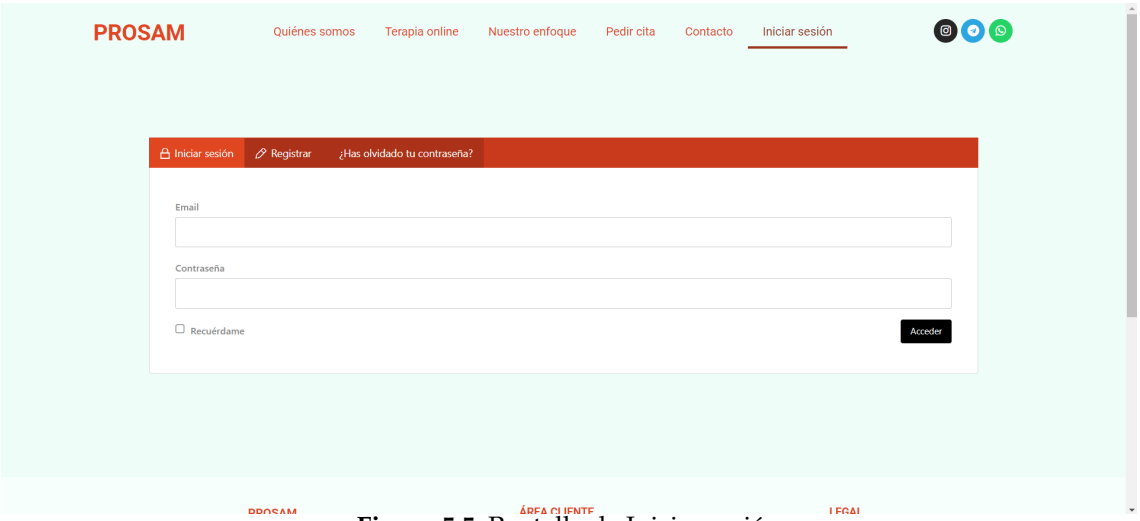

Figura 5.5: Pantalla de Iniciar sesión

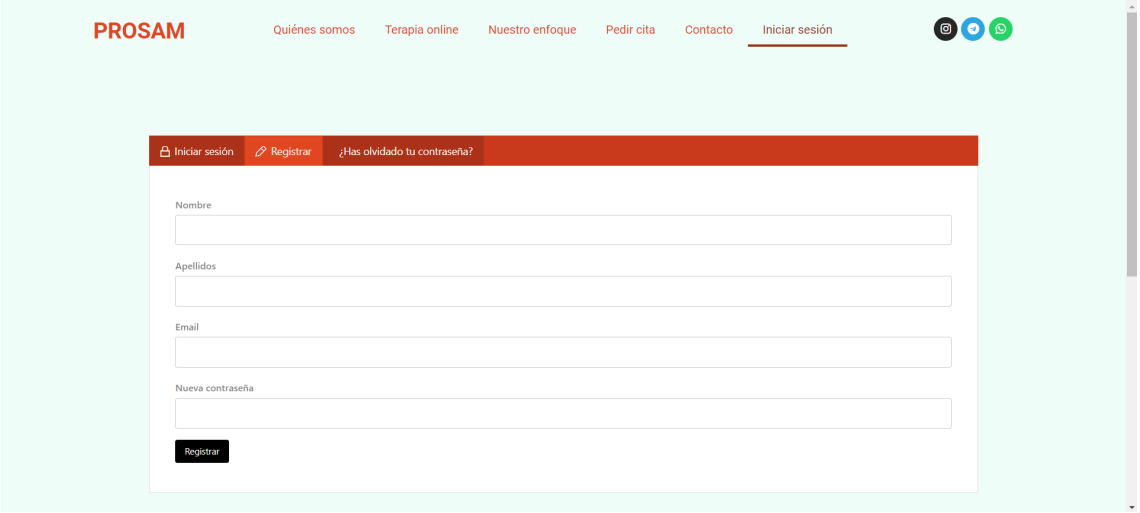

Figura 5.6: Pantalla de Registro

Figura 5.7: Pantalla de Pedir cita [I]

Figura 5.8: Pantalla de Pedir cita [II]

5.3.2. Bocetos actuales

Uno de los cambios solicitados fue la fijación del header o cabecera al hacer scroll, lo que facilita la navegación y mejora la usabilidad para los usuarios.

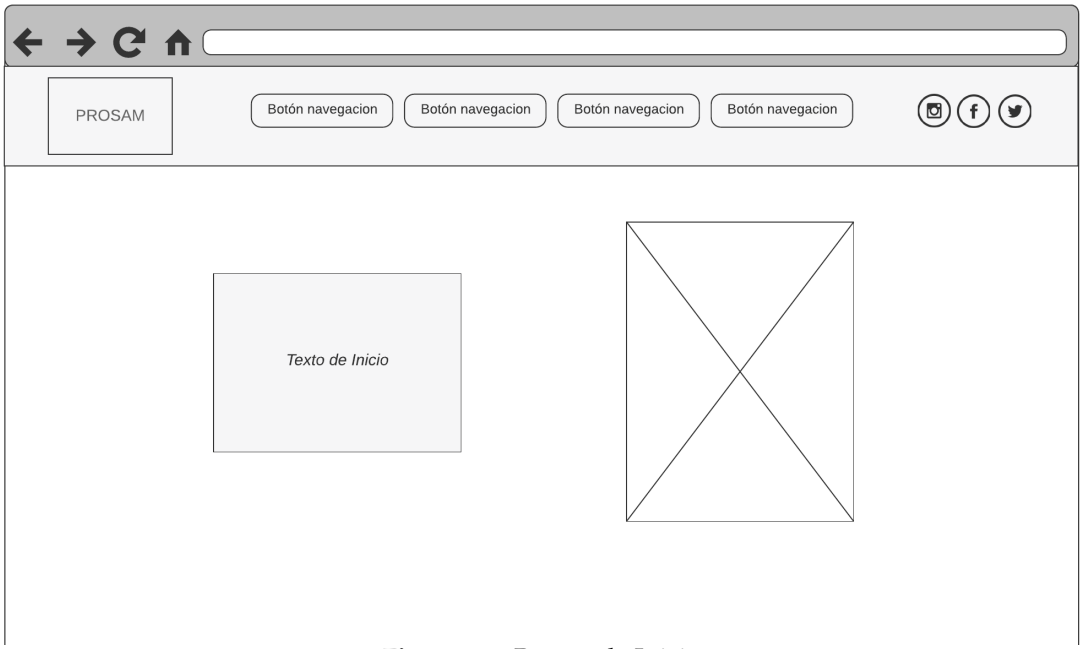

Figura 5.9: Boceto de Inicio

Además, uno de los requisitos definidos fue que la interfaz sea totalmente responsive, lo cual se ve reflejado en los siguientes bocetos.

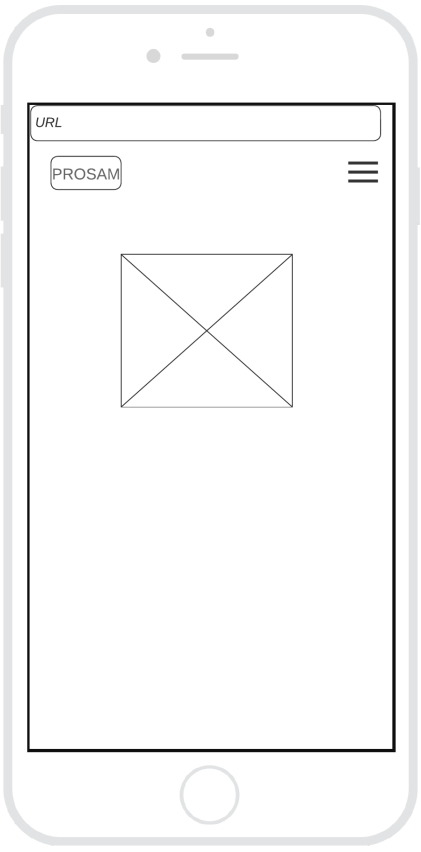

Figura 5.10: Boceto de Inicio en formato móvil [I]

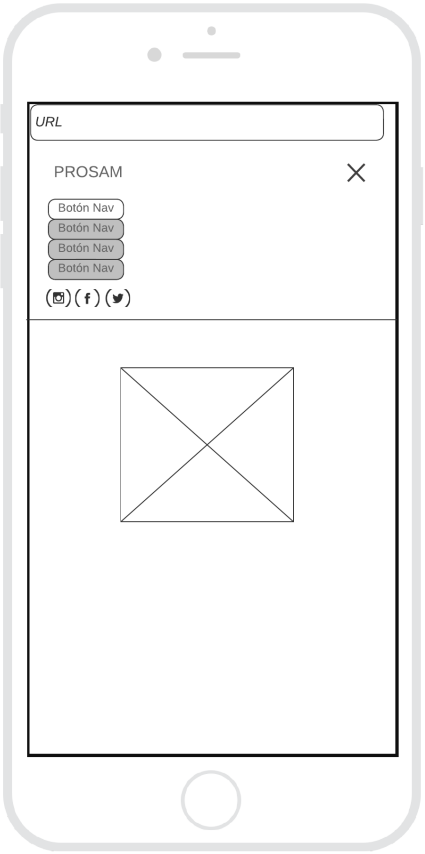

Figura 5.11: Boceto de Inicio en formato móvil [II]

En la sección de contacto, se han mejorado los estilos visuales y se ha añadido la opción de adjuntar un CV, permitiendo contactar con psicólogos no contratados.

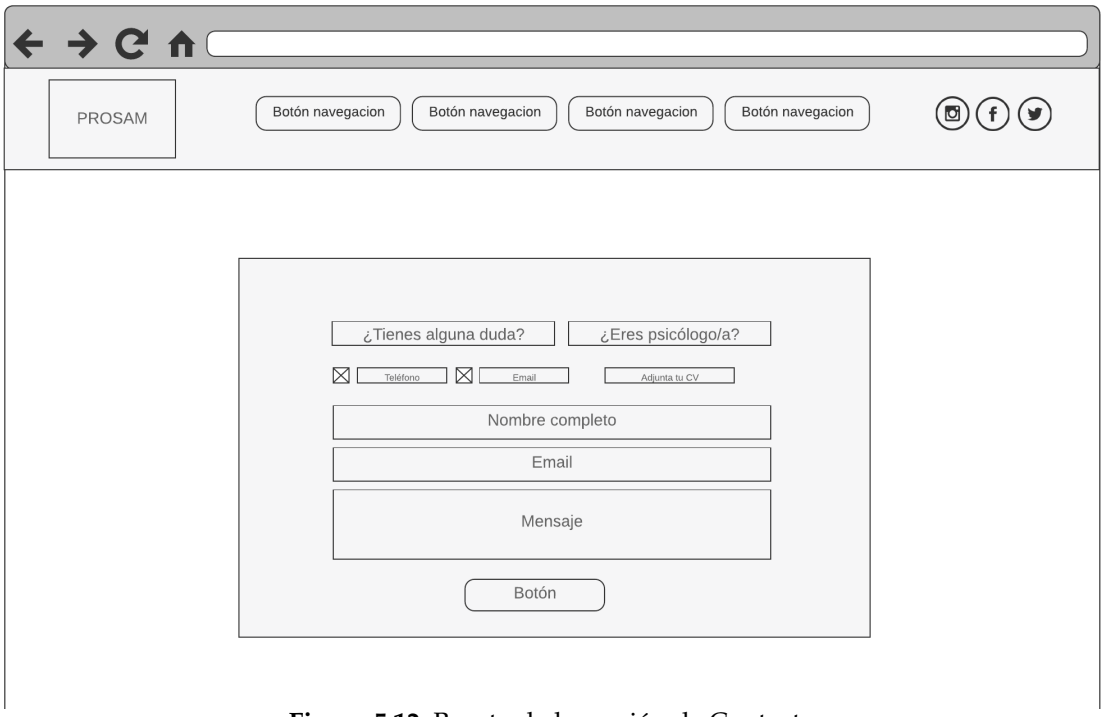

Figura 5.12: Boceto de la sección de Contacto

Por último, en las pantallas de inicio de sesión y registro, se ha incorporado la opción de inicio de sesión con Google.

Figura 5.13: Boceto de Iniciar sesión

Figura 5.14: Boceto de Registro

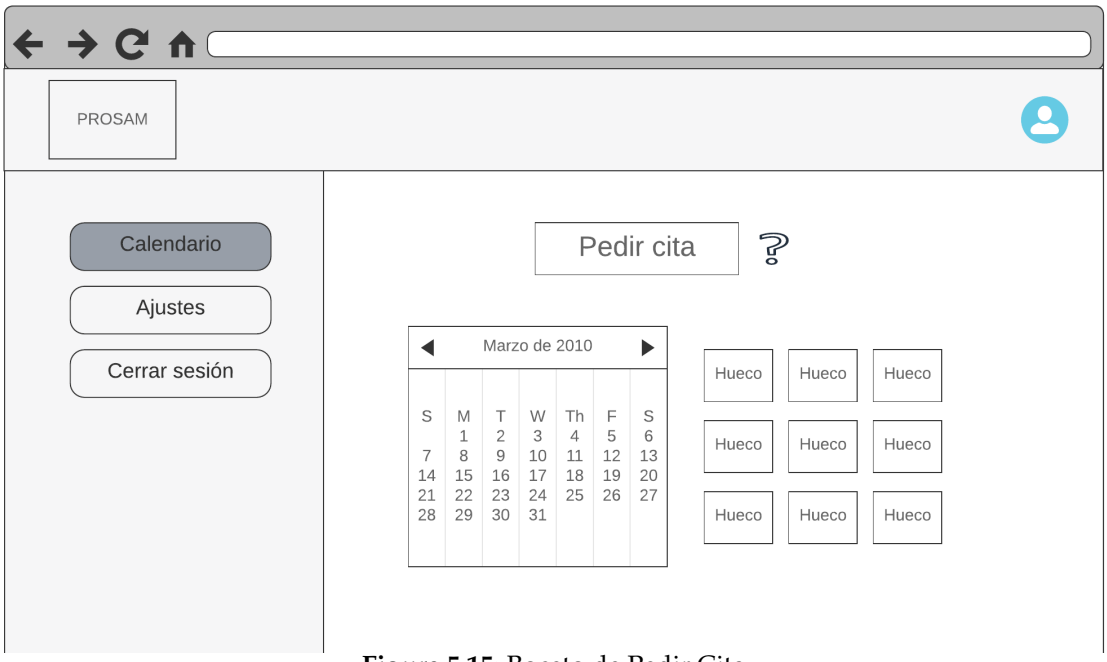

Figura 5.15: Boceto de Pedir Cita

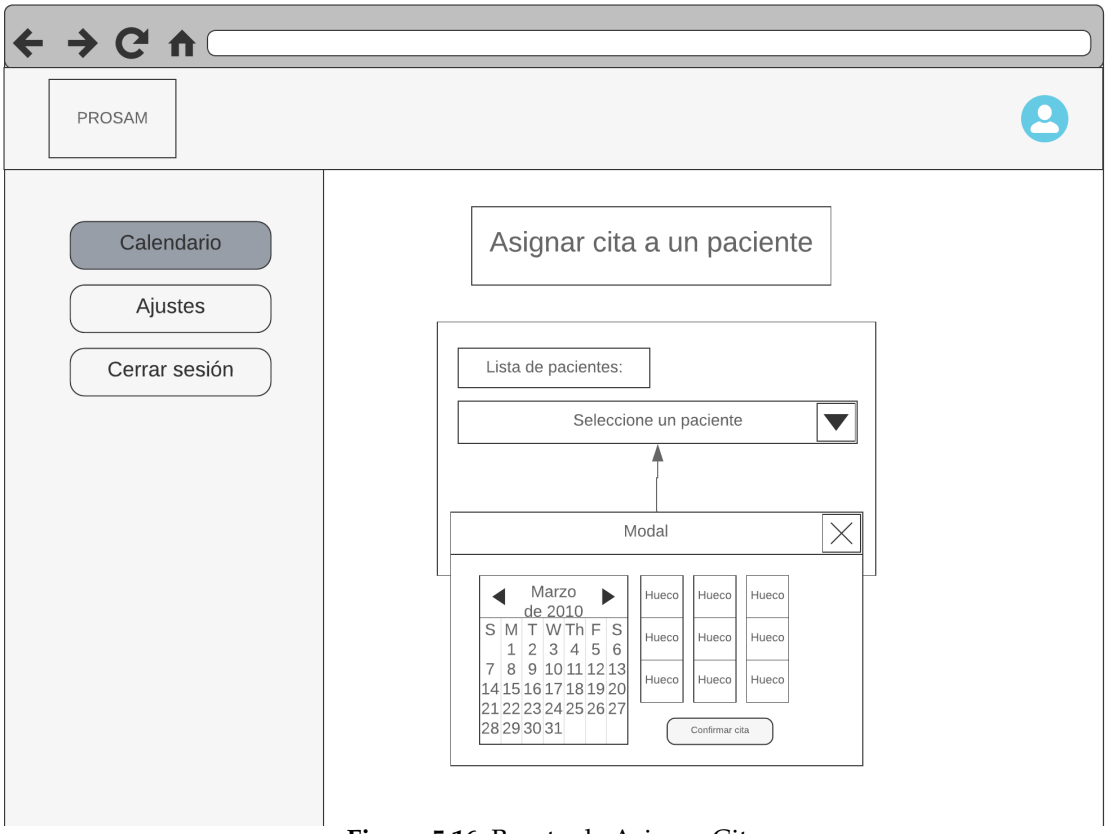

Figura 5.16: Boceto de Asignar Cita

CAPÍTULO 6
Desarrollo de la solución

En este capítulo se detallan los aspectos fundamentales del desarrollo de la solución propuesta para la plataforma web. Se comenzará por describir la arquitectura elegida para el proyecto, lo que permitirá entender cómo se ha organizado y estructurado tanto el *frontend* como el *backend*. A continuación, se explicarán las tecnologías empleadas y las herramientas seleccionadas para el desarrollo, esenciales para la gestión eficiente del proyecto. Por último, se presentarán ejemplos de código que ilustran cómo se han implementado las principales funcionalidades, destacando la integración entre el *frontend* y el *backend* para lograr una solución coherente y escalable.

6.1 Arquitectura

Este proyecto sigue una arquitectura cliente-servidor donde el *frontend* ha sido desarrollado con *Astro*, complementado con *React* y TypeScript, lo que permite una integración fluida entre el *frontend* y el *backend*, también implementado con *Astro*. El *backend* maneja la lógica del servidor, incluida la gestión de sesiones, la autenticación y la autorización, así como la integración con la base de datos y servicios externos como Stripe para pagos.

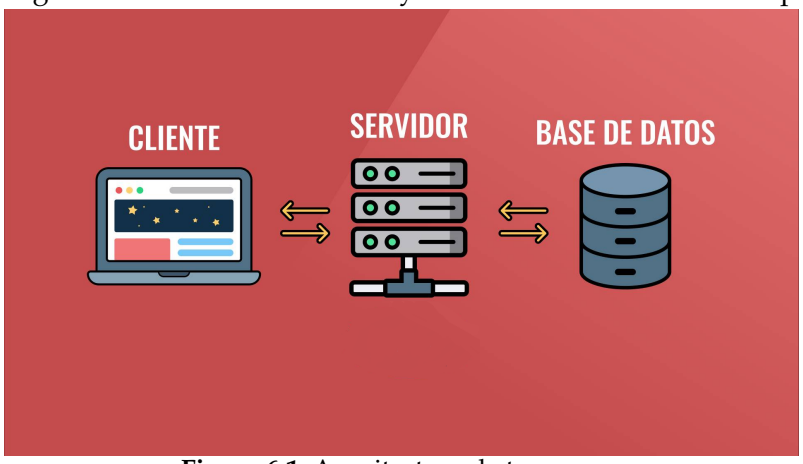

Figura 6.1: Arquitectura de tres capas

6.1.1. Frontend

El *frontend* está construido principalmente con *Astro*, utilizando *React* para componentes interactivos específicos. Cada página en la aplicación se define como un archivo .astro dentro de la carpeta src/pages/. Estas páginas manejan el enrutamiento, la carga de da-

tos y el diseño general del HTML, aprovechando el sistema de enrutamiento basado en archivos de *Astro*.

Para cada página, puedes importar y utilizar componentes tanto de *Astro* como de *React*. Los componentes *Astro* son ideales para la estructura estática y la presentación del contenido, mientras que los componentes *React* se integran para añadir interactividad y manejar lógica de negocio más compleja.

Por ejemplo, en la página del Calendario (/pages/dashboard/index.astro), se utiliza un layout general (Layout.astro) junto con componentes de *React* (Calendario.tsx entre otros) que se cargan y gestionan dinámicamente con la directiva client:load. Esto permite una combinación eficiente de renderización estática y dinámica, asegurando que el sitio sea rápido y responsivo.

Para la comunicación con el *backend*, he optado por la función fetch para interactuar con las rutas de API [1] proporcionadas por *Astro*. Esto permite una integración sencilla y directa con las API sin la necesidad de añadir dependencias adicionales. En particular, fetch se utiliza para realizar solicitudes a los endpoints definidos en el *backend*, manejando tanto la obtención de datos como la actualización del estado de la aplicación en el *frontend*.

6.1.2. Backend

El *backend* de este proyecto está construido utilizando *Astro* como marco principal, aprovechando su capacidad para definir endpoints personalizados mediante APIRoute y manejar las solicitudes HTTP de forma eficiente. Para garantizar un despliegue eficiente y escalable, el servidor de la aplicación está alojado en *Vercel* [13], utilizando el adaptador de *Vercel* para *Astro*. Este adaptador permite configurar la aplicación como *Serverless*[2], lo que facilita la escalabilidad automática y optimiza la latencia al aprovechar la infraestructura global de *Vercel*.

Estos endpoints se encargan de procesar las operaciones *CRUD* [3] sobre la base de datos, la cual es gestionada a través de *Astro DB* en combinación con *Astro Studio* y *Drizzle ORM* para garantizar la persistencia y seguridad de los datos.

La autenticación y el manejo de sesiones se realizan a través de Lucia, lo que asegura un control de acceso seguro y eficiente. Asimismo, las *Astro Actions* juegan un papel crucial en la validación de campos y la gestión de errores durante las interacciones del usuario con el *backend*, siendo especialmente útiles en procesos de autenticación. Este enfoque facilita la identificación y comunicación clara de errores en el *frontend*, mejorando la experiencia del usuario al ofrecer retroalimentación inmediata y precisa.

6.1.3. Base de datos

La gestión de la base de datos se realiza utilizando *Astro Studio* y *Astro DB*, que proporcionan un entorno robusto para la creación y manipulación de datos con soporte completo para SQL. Esto es posible gracias a la integración con *Drizzle ORM*, que asegura la seguridad de tipos en TypeScript y optimiza el flujo de trabajo para el desarrollo del proyecto.

[1] API (Application Programming Interface): Conjunto de definiciones y protocolos que permiten que un software se comunique con otro software.

[2] Serverless es un modelo de computación en la nube en el que el proveedor gestiona automáticamente la infraestructura, permitiendo a los desarrolladores enfocarse en el código sin preocuparse por la administración de servidores. Aunque "Serverless"no implica la ausencia de servidores, estos se manejan de manera abstracta, escalando automáticamente según la demanda.

[3] CRUD (Create, Read, Update, Delete): Operaciones básicas que se realizan en una base de datos o sistema de almacenamiento.

Este conjunto de herramientas, descrito en el capítulo de "Modelo de la base de datos", facilita la definición de esquemas y la gestión de datos de manera ágil y segura.

6.2 Contexto tecnológico

Antes de profundizar en las herramientas específicas para el *frontend* y *backend*, es importante destacar las herramientas seleccionadas para facilitar la programación y el control de versiones del proyecto. Estas herramientas desempeñan un papel crucial en la organización y eficiencia del desarrollo, y son las siguientes:

- **Visual Studio Code [14]:** Es un editor de código fuente ligero y potente, compatible con la mayoría de sistemas operativos. Destacado por su velocidad y flexibilidad, ofrece soporte integrado para JavaScript, TypeScript y Node.js, y cuenta con una amplia gama de extensiones que facilitan el desarrollo en la mayoría de lenguajes y entornos. Además, facilita la integración con npm, el gestor de paquetes para Node.js, lo que permite una gestión eficiente de las dependencias y módulos.
- **GitHub:** Es una herramienta esencial para almacenar, compartir y colaborar en proyectos de desarrollo de software. Utilizando *GitHub* [15], los desarrolladores pueden gestionar su código en repositorios, lo que permite un seguimiento detallado de los cambios. Gracias a su integración con Git, un sistema de control de versiones, es posible sincronizar cambios locales con un repositorio central, gestionar ramas y realizar fusiones de manera segura. Además, estas acciones se pueden realizar directamente desde el editor *Visual Studio Code*, donde *GitHub* está integrado, lo que facilita la gestión del proyecto sin necesidad de herramientas externas.

6.2.1. Frontend

El *frontend* de la aplicación es el responsable de la interfaz de usuario, asegurando que los datos se presenten de manera clara, interactiva y accesible. Para este proyecto, se han utilizado diversas tecnologías, librerías y recursos para optimizar la experiencia del usuario.

- **Tecnologías utilizadas:**
  1. **Astro:** *Astro* es un framework moderno optimizado para la creación de sitios web centrados en contenido, como blogs, páginas de marketing y comercio electrónico. Este framework destaca por su arquitectura basada en componentes y su capacidad para minimizar la carga de JavaScript en el cliente, mejorando así la velocidad de carga. En este proyecto, se utiliza como base para la construcción de las páginas, aprovechando su enfoque de renderizado en el servidor (SSR [4]) para entregar contenido estático de manera eficiente.
  2. **Tailwind CSS:** *Tailwind CSS* [16] es un framework de CSS que facilita la creación de interfaces de usuario modernas mediante un enfoque basado en clases predefinidas. En lugar de escribir CSS personalizado para cada elemento, *Tailwind CSS* proporciona una serie de clases que se pueden aplicar directamente a los componentes para definir sus estilos, colores, etc.

[4]SSR (*Server-Side Rendering*) es una técnica en la que las páginas web se generan en el servidor en lugar de en el navegador del cliente. Permitiendo entregar HTML completamente renderizado, mejorando la velocidad de carga y el rendimiento general del sitio.

3. **React:** *React* [17] es una biblioteca de JavaScript que facilita la creación del *frontend* mediante el uso de componentes. En combinación con *Astro*, *React* permite definir partes de la aplicación que necesitan interactividad dinámica, mientras que otras partes pueden permanecer estáticas. Pese a que *React* está basado en JavaScript, también acepta TypeScript, ofreciendo un desarrollo más seguro y con tipado estático.

- **Librerías utilizadas:**

1. **react-calendar:** Esta librería se ha utilizado para integrar un calendario interactivo dentro de la aplicación. Proporciona una interfaz sencilla para la selección de fechas, permitiendo a los usuarios elegir días, semanas o rangos de tiempo [18].
2. **react-hot-toast:** Esta librería se ha integrado para mostrar notificaciones de manera ligera y atractiva en la aplicación. Su capacidad para personalizar y gestionar los mensajes emergentes ha mejorado la interacción con el usuario, proporcionando feedback instantáneo [19].

- **Recursos de Interfaz de Usuario (UI):**

1. **Tabler Icons:** Es una colección de iconos diseñados para ser ligeros y fáciles de integrar en cualquier aplicación web [20].
2. **Cool Contrast:** Se trata de una herramienta que asegura que los colores utilizados en la aplicación cumplen con los estándares de accesibilidad [21].

6.2.2. Backend

El *backend* de la aplicación está diseñado para gestionar la lógica del servidor, la comunicación con la base de datos, la autenticación de usuarios, y la integración con servicios externos. A continuación, se detallan las tecnologías clave empleadas en este proyecto para asegurar un rendimiento óptimo y una experiencia de usuario segura.

- **Gestión de Endpoints y API:**

1. **API Routes:** En *Astro*, las *API Routes* [22] permiten crear endpoints personalizados para manejar solicitudes HTTP. Estos endpoints pueden generar respuestas en formato JSON, servir archivos o actuar como puntos de acceso para la ejecución de lógica de negocio en el servidor. Los endpoints se implementan como archivos TypeScript en el directorio `/pages`, permitiendo tanto la generación estática de contenido como la ejecución dinámica de código en modo SSR. Esto permite que las rutas se generen bajo demanda y manejen características adicionales, como la gestión de códigos de estado HTTP y la respuesta a diferentes métodos HTTP (*GET, POST, DELETE*, etc.), asegurando que el servidor maneje adecuadamente todas las peticiones entrantes.
2. **Astro Actions:** Las *Astro Actions* [23] simplifican la interacción entre el *frontend* y el *backend* mediante la definición de funciones de servidor que pueden ser llamadas de manera segura y tipada desde el *frontend*. Estas acciones se utilizan para la validación de datos de formularios y la manipulación de datos del lado del servidor, eliminando la necesidad de realizar manualmente peticiones *fetch*. Además, las *Astro Actions* integran validaciones automáticas utilizando *Zod* [24], lo que asegura la integridad de los datos y simplifica la gestión de errores en la aplicación.

- **Autenticación y Seguridad:**

  1. **Lucia Auth:** *Lucia* [25] es una librería que gestiona la autenticación de usuarios, simplificando el manejo de sesiones y tokens de autenticación. *Lucia* se integra directamente con *Drizzle ORM* para almacenar y gestionar de forma segura los datos de usuario y sesión en la base de datos. Este enfoque permite una gestión robusta de la seguridad del usuario, incluyendo la protección CSRF [5] y el manejo eficiente de cookies de sesión. La arquitectura modular de *Lucia* permite su adaptación a diversas plataformas y requisitos específicos del proyecto, asegurando que la autenticación sea tanto segura como flexible.
  2. **OAuth con Arctic:** Para facilitar la autenticación a través de terceros, como Google, se ha implementado *Arctic* [26], una librería ligera que gestiona el flujo de autenticación *OAuth 2.0*. *Arctic* simplifica la creación de URLs de autorización, la validación de *callbacks* y la renovación de tokens de acceso, integrándose con *Lucia* para crear sesiones de usuario basadas en la autenticación externa. Esto permite a los usuarios registrarse e iniciar sesión en la aplicación utilizando sus cuentas de Google de manera segura y eficiente.
  3. **Oslo:** Para reforzar la seguridad de la aplicación, *Oslo* [27] se utiliza para la generación y verificación de JWTs (*JSON Web Tokens*), así como para el manejo de contraseñas utilizando algoritmos seguros como *Argon2id*. Además, *Oslo* facilita la implementación de firmas y codificaciones seguras, asegurando que los datos sensibles manejados por la aplicación estén protegidos frente a posibles vulnerabilidades.

- **Integraciones y Servicios Externos:**

  1. **Resend:** Para la gestión del envío de correos electrónicos, se ha integrado *Resend* [28], un servicio que permite enviar correos transaccionales, como confirmaciones de registro y recuperación de contraseñas. *Resend* ofrece una API sencilla y eficiente, asegurando que las comunicaciones con los usuarios sean rápidas y confiables.
  2. **Stripe:** La gestión de pagos en la aplicación se realiza a través de *Stripe*, una plataforma de pagos que permite procesar transacciones de manera segura. *Stripe* se utiliza tanto para manejar los pagos de citas como para gestionar reembolsos en caso de cancelaciones, garantizando una experiencia de usuario fluida y segura en todo el proceso de pago.
  3. **Jitsi**: Se ha integrado *Jitsi* [29] para gestionar las reuniones virtuales, permitiendo crear enlaces únicos para cada cita. Esta plataforma de videoconferencias de código abierto fue elegida por su fácil integración y porque no requiere que los usuarios creen cuentas. Al programar una cita, se genera un enlace personalizado que se almacena en el campo "meetingLink"de la tabla Appointment, facilitando el acceso tanto para psicólogos como para pacientes.

6.3 Ejemplos de código

En esta sección se presentan ejemplos prácticos del código implementado, destacando tanto el *frontend* como el *backend*. Antes de profundizar en los componentes específicos, es importante entender cómo se ha configurado el servidor para asegurar su correcta

[5] CSRF: El CSRF (Cross-Site Request Forgery) es una vulnerabilidad que permite a un atacante enviar comandos no autorizados desde un usuario autenticado en un sitio web en el que se confía.

integración en un entorno de producción. La configuración del servidor se gestiona mediante el archivo astro.config.mjs (Figura 6.2), que permite ajustar diversas herramientas y servicios para el despliegue de la aplicación. En este proyecto, se ha utilizado *Vercel* como plataforma de alojamiento, facilitando la implementación en un entorno *Serverless*.

```
import { defineConfig } from "astro/config"
import tailwind from "@astrojs/tailwind"
import vercel from "@astrojs/vercel/serverless"
import db from "@astrojs/db"
import react from "@astrojs/react"

// https://astro.build/config
export default defineConfig({
  site: 'https://localhost:4321',
  integrations: [tailwind(), db(), react()], // Tailwind CSS, Astro DB y React
  experimental: {
    actions: true, // Astro Actions
  },
  security: {
    checkOrigin: true // Protección CSRF | Lucia Auth
  },
  output: "server",
  adapter: vercel({
    webAnalytics: { enabled: true }
  })
})
```

Figura 6.2: Archivo de configuración de Astro

En cuanto a la estructura del proyecto, se ha seguido una estructura de directorios organizada en torno a la lógica del negocio. A continuación, se muestra un ejemplo de la estructura de carpetas utilizada:

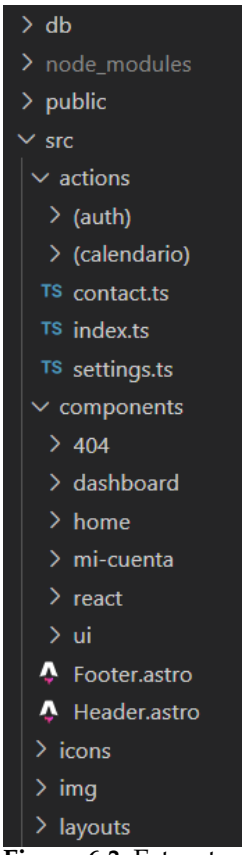

Figura 6.3: Estructura del proyecto [I]

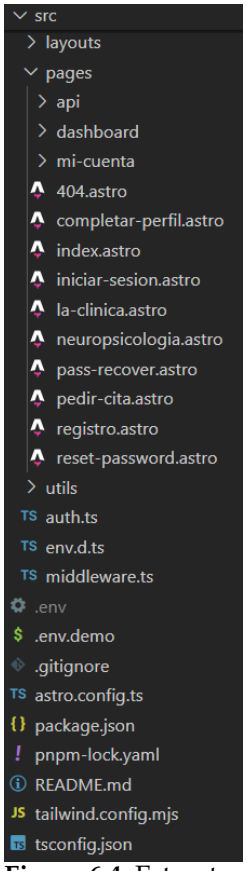

Figura 6.4: Estructura del proyecto [II]

Esta estructura por tipo y funcionalidad refleja la lógica de negocio en carpetas dedicadas, facilitando la localización y gestión de las diferentes partes de la aplicación. Aunque se

repiten funcionalidades en diferentes carpetas, esta organización permite una navegación más sencilla y un código más modular.

6.3.1. Ejemplos de código del frontend

El *frontend* de la aplicación se ha desarrollado utilizando *Astro* y *React*, lo que permite una combinación eficiente de componentes reutilizables y funcionalidad dinámica. A continuación, se describen los aspectos clave de la implementación del *frontend*, incluyendo el uso de componentes reutilizables en *Astro*, la estructura de las páginas y layouts, y la integración de componentes *React* en las páginas *Astro*.

Componentes reutilizables en Astro: *Astro* facilita la creación de componentes reutilizables que ayudan a mantener la consistencia y modularidad del proyecto. Un ejemplo representativo es el uso de <slot />, que permite definir una estructura base en componentes como Layout.astro, donde se inyecta contenido específico de cada página. Este enfoque asegura que todos los elementos comunes, como headers y footers, se mantengan uniformes a lo largo de la aplicación.

```
---
import Header from "@/components/Header.astro"
import Footer from "@/components/Footer.astro"
import { Toaster } from "react-hot-toast"

interface Props {
  title: string
  description: string
}

const { title, description } = Astro.props
---

<!doctype html>
<html lang="es">
    <head>
        <meta charset="UTF-8" />
        <meta name="description" content={description} />
        <meta name="viewport" content="width=device-width" />
        <link rel="icon" type="image/svg+xml" href="/favicon.svg" />
        <title>{title}</title>
    </head>

    <body class="relative text-black bg-[#effdf9]">
        <Header />
        <slot />
        <Footer />
        <Toaster
          client:load
          toastOptions={{
```

Figura 6.5: Archivo: Layout.astro

En el ejemplo anterior, el componente Layout.astro envuelve cada página con un header y un footer, mientras que el contenido principal se inyecta en el <slot />. Este diseño modular permite que las páginas se centren en su contenido específico sin duplicar código común.

Páginas y layouts en Astro: Las páginas en *Astro* se encargan del enrutamiento y la disposición de cada sección del sitio web. Cada archivo en src/pages/ se convierte automá-

ticamente en una ruta en la aplicación. Estos archivos .astro pueden incluir componentes reutilizables y layouts para asegurar una experiencia de usuario consistente.

```astro
---
import Layout from '@/layouts/Layout.astro'
import ChangeArt from '@/components/home/ChangeArt.astro'
import SemiFooter from '@/components/ui/SemiFooter.astro'
import WhyOnline from '@/components/home/WhyOnline.astro'
import OurFocus from '@/components/home/OurFocus.astro'
import OnlineSesion from '@/components/home/OnlineSesion.astro'
import Contact from '@/components/home/Contact.astro'
import SectionContainer from "@/components/ui/SectionContainer.astro"
import FreqQAS from '@/components/home/FreqQAS.astro'
import VideoProsam from '@/components/home/VideoProsam.astro'
---

<Layout
    title="Proyecto Salud Mental - Clínica de psicología online"
    description="Clínica de psicología online. La salud mental debería ser un derecho, no un lujo."
>
    <main>
        <SectionContainer id="inicio" class="xl:pt-60 md:pt-40 sm:pt-20 md:space-y-20 sm:space-y-10 space-y-1
            <ChangeArt />
            <WhyOnline />
        </SectionContainer>
        <SectionContainer id="inicio" class="xl:space-y-40 md:space-y-20 sm:space-y-10 space-y-10 bg-[#f4a6
            <OurFocus />
            <OnlineSesion />
            <VideoProsam />
        </SectionContainer>
        <SectionContainer id="inicio" class="md:py-20 sm:py-10 py-8 lg:space-y-40 sm:space-y-20 space-y-20">
            <Contact />
            <FreqQAS />
        </SectionContainer>
        <SectionContainer id="inicio" class="lg:mt-40 md:mt-20 sm:mt-10 mt-8">
            <SemiFooter />
        </SectionContainer>
    </main>
</Layout>
```

Figura 6.6: Archivo: index.astro

El ejemplo anterior muestra cómo index.astro utiliza Layout.astro para estructurar la página principal, asegurando que los elementos comunes se mantengan coherentes en todo el sitio.

Integración de componentes React: Una de las ventajas que nos ofrece *Astro* es su capacidad para integrar componentes *React*, permitiendo agregar interactividad dinámica cuando es necesario. Todos los componentes *React* se cargan en las páginas *Astro* utilizando la directiva *client:load*, lo que garantiza que solo se carguen en el cliente cuando sean necesarios, optimizando así el rendimiento.

```tsx
import { actions, isInputError } from "astro:actions"
import { useEffect, useRef, useState, type FormEvent } from "react"
import toast from "react-hot-toast"
import GoogleIcon from "@/icons/GoogleIcon"
import EyeIcon from "@/icons/EyeIcon"
import EyeOffIcon from "@/icons/EyeOffIcon"

const LoginForm = () => {
  const [email, setEmail] = useState("")
  const emailRef = useRef<HTMLLabelElement>(null)

  const [password, setPassword] = useState("")
  const passwordRef = useRef<HTMLLabelElement>(null)

  const [disabled, setDisabled] = useState(true)
  const [loading, setLoading] = useState(false)
  const [showPassword, setShowPassword] = useState(false)

  useEffect(() => {
    if (password.length >= 7) {
      setDisabled(false)
    } else {
      setDisabled(true)
    }
  }, [password])

  const handleSubmit = async (e: FormEvent<HTMLFormElement>) => {…
  }

  return (…
  )
}
export default LoginForm
```

Figura 6.7: Archivo: Login.tsx [I]

```tsx
const handleSubmit = async (e: FormEvent<HTMLFormElement>) => {
  e.preventDefault()
  setLoading(true)
  setDisabled(true)

  const formData = new FormData(e.currentTarget)

  try {
    const { error } = await actions.login.safe(formData)

    if (error && isInputError(error)) {
      Object.entries(error.fields).forEach(([key, value]) => {
        if (key === "email") {
          emailRef.current?.classList.add("border-[#fe45ff]")
          emailRef.current?.classList.remove("border-[#546375]")
          toast.error(`${value[0]} Revisa el campo: Email`)
        } else if (key === "password") {
          passwordRef.current?.classList.add("border-[#fe45ff]")
          passwordRef.current?.classList.remove("border-[#546375]")
          toast.error(`${value[0]} Revisa el campo: Contraseña`)
        } else {
          toast.error(`${value[0]} Revisa el campo: ${key}`)
        }
      })
    } else if (error) {···
    } else {
      setEmail("")
      setPassword("")
      toast.success("Bienvenido!")
      setTimeout(() => {
        window.location.href = "/mi-cuenta"
      }, 1200)
    }
  } catch (err) {
    toast.error("Error inesperado. Por favor, inténtelo de nuevo más tarde.")
  } finally {
    setLoading(false)
    setDisabled(false)
  }
}

return (···
)
}
```

Figura 6.8: Archivo: Login.tsx [II]

Por ejemplo, el componente LoginForm.tsx (Figuras 6.7 y 6.8), implementado en *React*, maneja la lógica del formulario de inicio de sesión. Este componente se carga dentro de la página iniciar-sesion.astro utilizando *client:load*, lo que permite mantener la aplicación ligera y rápida, al tiempo que se asegura una experiencia de usuario fluida y dinámica.

6.3.2. Ejemplos de código del backend

Después de haber definido y explicado los ejemplos de código correspondientes al *frontend*, es momento de centrarse en el *backend*. A continuación, se explorarán tres áreas fundamentales del *backend*: las *API Routes* junto con las *Astro Actions*, la autenticación y seguridad mediante *Lucia Auth*, y la integración con servicios externos como *Resend*, *Stripe* y *Jitsi*.

API Routes y Astro Actions: Las *API Routes* en la aplicación gestionan la interacción con los datos y la lógica de negocio. Un ejemplo clave es la ruta de API userServices (Figuras 6.9 y 6.10), que permite a los pacientes obtener la disponibilidad global de citas mediante el método GET. Además, esta ruta utiliza una *Astro Action* llamada getAllUsers para recuperar y organizar las horas disponibles para cada trabajador.

```
export const GET: APIRoute = async () => {
  try {
    await cleanUpExpiredAppointments()

    // Astro Action (Obtiene los huecos de todos los worker)
    const response = await actions.getAllUsers()
    const data = await response.json()

    const workerAppointments: WorkerAppointments = data.workerAppointments || {}
    const workerDetails = data.workerDetails || {}
    const workerIds = Object.keys(workerAppointments)

    const dailyHours = [
      "10:00", "10:45", "11:30", "12:15", "13:00",
      "14:00", "14:45", "15:30", "16:15", "17:00",
      "17:45", "18:30", "19:15", "20:00"
    ]

    const globalAvailability: Availability = {}

    const startDate = new Date()
    const endDate = new Date()
    endDate.setDate(startDate.getDate() + 30)

    const dateRange: string[] = []
    for (let d = startDate; d <= endDate; d.setDate(d.getDate() + 1)) {
      const dateString = d.toLocaleDateString('en-GB')
      dateRange.push(dateString)
    }

    dateRange.forEach(date => {
      globalAvailability[date] = {}
      dailyHours.forEach(hour => {
        globalAvailability[date][hour] = []
      })
    })
```

Figura 6.9: Ruta de API: userServices [I]

```
export const GET: APIRoute = async () => {
    dateRange.forEach(date => {
      dailyHours.forEach(hour => {
            if (!isToday || (hourNumber > currentHour || (hourNumber === currentHour && minuteNu
              assignedWorker = workerId
              hasAvailableHours = true
              break
            }
          }
        }

        if (assignedWorker) {
          const worker = workerDetails[assignedWorker] || { firstName: 'N/A', lastName: 'N/A' }
          globalAvailability[date][hour].push({
            workerId: assignedWorker,
            firstName: worker.firstName,
            lastName: worker.lastName
          })
          lastWorkerIndex = (workerIndex + 1) % workerIds.length
        }
      })

      if (!hasAvailableHours) {
        delete globalAvailability[date]
      }
    })

    return new Response(JSON.stringify({ appointments: globalAvailability }), {
      headers: { 'Content-Type': 'application/json' }
    })
  } catch (error) {
    console.error("Error al obtener la disponibilidad:", error)
    return new Response("Error al obtener la disponibilidad", { status: 500 })
  }
}
```

Figura 6.10: Ruta de API: userServices [II]

```ts
import { lucia } from "@/auth"
import { defineAction, z } from "astro:actions"
import { User, db, eq } from "astro:db"
import { Argon2id } from "oslo/password"

export const login = defineAction({
  accept: "form",
  input: z.object({
    email: z.string().email(),
    password: z.string().min(7, {
      message: "La contraseña debe tener al menos 7 carácteres de longitud.",
    }),
  }),
  handler: async (input, context) => {
    const { email, password } = input

    const foundUser = (await db
      .select()
      .from(User)
      .where(eq(User.email, email))).at(0)

    if(!foundUser || !foundUser.password){
      throw new Error("El correo o la contraseña no son correctos.")
    }

    // Si la contraseña no es valida
    const validPassword = await new Argon2id().verify(foundUser.password, password)

    if (!validPassword) {
      throw new Error("El correo o la contraseña no son correctos.")
    }

    // Contraseña valida -> Puede iniciar sesion
    // Creamos las cookies de la session
    const session = await lucia.createSession(foundUser.id, {})
    const sessionCookie = lucia.createSessionCookie(session.id)

    context.cookies.set(
      sessionCookie.name,
      sessionCookie.value,
      sessionCookie.attributes
    )

    return true
  },
```

Figura 6.11: Astro Action: login.ts

Por otro lado, la acción de inicio de sesión login.ts (Figura 6.11) es un ejemplo representativo de cómo las *Astro Actions* gestionan procesos críticos en el *backend*. Esta acción valida las credenciales proporcionadas por el usuario, verifica la contraseña utilizando *Argon2id* y, si es correcta, genera una sesión segura utilizando *Lucia Auth*. Este mecanismo asegura que la autenticación sea manejada de forma eficiente y segura.

Autenticación y seguridad con Lucia Auth: La autenticación en esta aplicación se refuerza con un middleware que valida las sesiones de los usuarios en cada solicitud. Este middleware, implementado en middleware.ts (Figuras 6.11 y 6.12), se encarga de verificar la existencia y validez de las cookies de sesión generadas por *Lucia Auth*. Según el estado de autenticación y el rol del usuario, el middleware redirige a los usuarios a la página adecuada: los usuarios autenticados son llevados a su cuenta, mientras que los administradores o trabajadores son dirigidos a su panel de control.

```ts
import { lucia } from "./auth"
import { defineMiddleware } from "astro:middleware"
import { User, db, eq } from "astro:db"

export const onRequest = defineMiddleware(async (context, next) => {
    const sessionId = context.cookies.get(lucia.sessionCookieName)?.value ?? null
    const { pathname } = context.url

    if (!sessionId) {
        const sessionCookie = lucia.createBlankSessionCookie()
        context.cookies.set(
          sessionCookie.name,
          sessionCookie.value,
          sessionCookie.attributes
        )
        context.locals.user = null
        context.locals.session = null

        if (pathname.includes("/mi-cuenta") || pathname.includes("/dashboard")) {
            return context.redirect("/iniciar-sesion")
        }
    }

    if (sessionId) {
        const { session, user } = await lucia.validateSession(sessionId)
        if (session && session.fresh) {
          const sessionCookie = lucia.createSessionCookie(session.id)
          context.cookies.set(
            sessionCookie.name,
            sessionCookie.value,
            sessionCookie.attributes
          )
        }
        context.locals.session = session
        context.locals.user = user
```

Figura 6.12: Archivo: middleware.ts

```ts
    if (user?.email) {

        const foundUser = (await db
          .select()
          .from(User)
          .where(eq(User.email, user.email))).at(0)

        // Verificar si firstName y lastName están completos, sino a completar campos
        if (!foundUser?.firstName || !foundUser?.lastName) {
            if (pathname.includes("/mi-cuenta") || pathname.includes("/dashboard")) {
                return context.redirect("/completar-perfil")
            }
        }

        // Redirección por Rol de usuario
        if (foundUser?.role === "admin" || foundUser?.role === "worker") {
          if (pathname === "/iniciar-sesion") {
            return context.redirect("/dashboard")
          }
          if (pathname.includes("/mi-cuenta")) {
            return context.redirect("/dashboard")
          }
        } else {
          if(pathname.includes("/dashboard")){
            return context.redirect("/mi-cuenta")
          }
        }
    }

    if (pathname === "/iniciar-sesion" && session) {
      return context.redirect("/mi-cuenta")
    }
  }
  return next()
})
```

Figura 6.13: Archivo: middleware.ts

Integración con servicios externos: Un ejemplo clave es el manejo de pagos y la confirmación de citas a través de un webhook [6] (Figuras 6.13 y 6.14). Cuando un pago se completa exitosamente en *Stripe*, este webhook valida la transacción y, si es exitosa, genera un enlace de videoconferencia utilizando *Jitsi*. Este enlace, junto con la confirmación de la cita, se envía al paciente y al psicólogo mediante un correo electrónico gestionado por *Resend*.

```ts
export const POST: APIRoute = async ({ request }) => {
  try {
    const STRIPE_SECRET_KEY = import.meta.env.STRIPE_SECRET_KEY
    const STRIPE_WEBHOOK_SIG = import.meta.env.STRIPE_WEBHOOK_SIG

    if (!STRIPE_SECRET_KEY || !STRIPE_WEBHOOK_SIG) {
      return new Response(null, {
        status: 500,
      })
    }

    const stripe = new Stripe(STRIPE_SECRET_KEY, { apiVersion: '2024-06-20' })

    const rawBody = await request.arrayBuffer()
    const sig = request.headers.get('stripe-signature')

    if (!sig) {
      throw new Error('Falta la firma de Stripe')
    }

    let event
    try {
      event = stripe.webhooks.constructEvent(Buffer.from(rawBody), sig, STRIPE_WEBHOOK_SIG)
    } catch (error) {
      console.error("Error: ", error)
      return new Response(`Webhook Error: ${error}`, {
        status: 400,
      })
    }

    if (event.type === 'checkout.session.completed') {
      const session = event.data.object as Stripe.Checkout.Session
      const metadata = session.metadata
      const paymentIntentId = session.payment_intent as string
```

Figura 6.14: Ruta de API: webhook [I]

[6] Un webhook es un mecanismo que permite a una aplicación recibir eventos en tiempo real desde otra aplicación o servicio. Al configurar un webhook, se registra un punto de conexión en la aplicación que escuchará eventos asíncronos, como la confirmación de pagos, disputas de cargos o la ejecución de pagos recurrentes

```
export const POST: APIRoute = async ({ request }) => {
  try {
    const STRIPE_SECRET_KEY = import.meta.env.STRIPE_SECRET_KEY
    const STRIPE_WEBHOOK_SIG = import.meta.env.STRIPE_WEBHOOK_SIG

    if (!STRIPE_SECRET_KEY || !STRIPE_WEBHOOK_SIG) {
      return new Response(null, {
        status: 500,
      })
    }

    const stripe = new Stripe(STRIPE_SECRET_KEY, { apiVersion: '2024-06-20' })

    const rawBody = await request.arrayBuffer()
    const sig = request.headers.get('stripe-signature')

    if (!sig) {
      throw new Error('Falta la firma de Stripe')
    }

    let event
    try {
      event = stripe.webhooks.constructEvent(Buffer.from(rawBody), sig, STRIPE_WEBHOOK_SIG)
    } catch (error) {
      console.error("Error: ", error)
      return new Response(`Webhook Error: ${error}`, {
        status: 400,
      })
    }

    if (event.type === 'checkout.session.completed') {
      const session = event.data.object as Stripe.Checkout.Session
      const metadata = session.metadata
      const paymentIntentId = session.payment_intent as string
```

Figura 6.15: Ruta de API: webhook [II]

Este flujo asegura que las citas solo se confirmen después de recibir el pago, manteniendo la integridad del sistema y proporcionando una experiencia de usuario eficiente y segura.

6.3.3. Integración *frontend* y *backend*

La integración entre el *frontend* y el *backend* es crucial para ofrecer una experiencia de usuario fluida y coherente en toda la aplicación. En este proyecto, dicha integración se logra principalmente mediante *API Routes* y *Astro Actions*, que permiten que el *frontend* interactúe directamente con la lógica del *backend* para manejar diversas operaciones como la gestión de usuarios y citas, así como la ejecución de pagos.

Integración de React y Astro Action: Un ejemplo representativo de esta integración se encuentra en el manejo del inicio de sesión de los usuarios. El componente *React* LoginForm.tsx (Figuras 6.7 y 6.8), encargado de la interfaz de inicio de sesión, se comunica con el *backend* a través de la *Astro Action* login.ts (Figura 6.11). Cuando un usuario ingresa sus credenciales y envía el formulario, estos datos se recopilan y se envían al *backend* como un objeto *FormData*. La *Astro Action* login.ts recibe estos datos, verifica la autenticidad de las credenciales, y si son correctas, genera una sesión segura utilizando *Lucia Auth*. Esta sesión se almacena en una cookie en el navegador del usuario, lo que permite mantenerlo autenticado en futuras interacciones con la aplicación.

```
const formData = new FormData(e.currentTarget)

try {
  const { error } = await actions.login.safe(formData)

  if (error && isInputError(error)) {
    Object.entries(error.fields).forEach(([key, value]) => {
      if (key === "email") {
        emailRef.current?.classList.add("border-[#fe45ff]")
        emailRef.current?.classList.remove("border-[#546375]")
        toast.error(`${value[0]} Revisa el campo: Email`)
      } else if (key === "password") {
        passwordRef.current?.classList.add("border-[#fe45ff]")
        passwordRef.current?.classList.remove("border-[#546375]")
        toast.error(`${value[0]} Revisa el campo: Contraseña`)
      } else {
        toast.error(`${value[0]} Revisa el campo: ${key}`)
      }
    })
```

Figura 6.16: Llamada a login.ts desde LoginForm.tsx

Integración de React y API Route: Otro ejemplo clave de la integración *frontend-backend* se da en el manejo del calendario de citas, donde el componente *React* Calendar.tsx interactúa con la ruta de API userServices.ts. Dependiendo del rol del usuario, se realizan diferentes solicitudes a la API para obtener o modificar la disponibilidad de citas. Cuando un paciente o administrador accede al calendario, se realiza una solicitud GET a userServices.ts, que devuelve la disponibilidad global de los psicólogos en formato JSON. Esta información se procesa en el *frontend* y se muestra al usuario de manera clara y accesible.

```
const Calendario: FC<IndexProps> = ({ setDate, date }) => {
  const fetchUserRole = async () => {
    if (!response.ok) {
      throw new Error('Error fetching role')
    }
    const data = await response.json()
    setRole(data.role)
    setWorkingWith(data.workingWith)
  } catch (error) {
    console.error('Error fetching user role:', error)
  }
}

const fetchAppointments = async (role: string) => {
  setLoading(true)
  setError(null)

  try {
    const endpoint = role === 'worker' ? '/api/workerServices' : '/api/userServices'
    const response = await fetch(endpoint, {
      method: 'GET',
      headers: { 'Content-Type': 'application/json' },
    })

    if (!response.ok) {
      throw new Error('Error al obtener las citas')
    }

    const data = await response.json()

    if (data.appointments) {
      setAppointments(data.appointments)
      if (data.disabledDates) {
        setDisabledDates(data.disabledDates.map((dateStr: string) => new Date(dateStr)))
      }
    } else {
      throw new Error('Estructura de datos no válida')
    }
  } catch (error) {
    setError('No se pudo obtener la información de las citas')
  } finally {
    setLoading(false)
  }
}
```

Figura 6.17: Llamada a /api/userServices desde Calendar.tsx

Este flujo de trabajo permite que la interfaz de usuario se mantenga sincronizada con la lógica del *backend*, mostrando siempre la disponibilidad más actualizada.

CAPÍTULO 7
Producto desarrollado

Después de completar el desarrollo de la aplicación, se presentan diversas capturas del proyecto finalizado. A continuación, se detalla cómo los usuarios pueden interactuar con la interfaz y cómo la aplicación responde a estas interacciones.

- **Registrarse:** La primera pantalla permite a los usuarios no autenticados registrarse en la plataforma. El formulario de registro requiere que el usuario complete los campos solicitados y, posteriormente, presione el botón ”Registrarse” o elija la opción de continuar con Google. En caso de que el usuario ya esté registrado, se le ofrece la opción de acceder directamente a su cuenta mediante el enlace "¿Ya tienes una cuenta? INICIA SESIÓN".

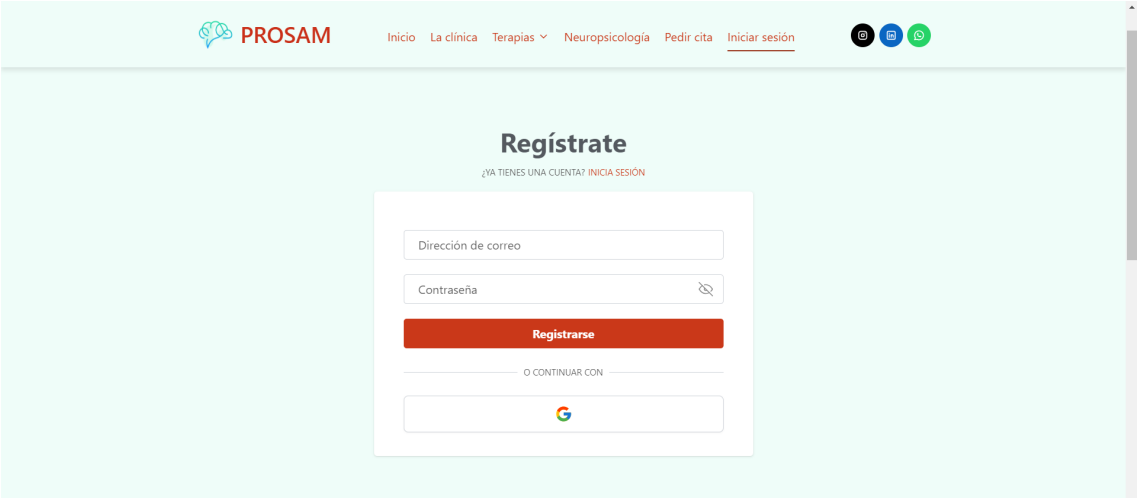

Figura 7.1: Pantalla: Registrarse

- **Iniciar Sesión:** Esta pantalla está diseñada para que los usuarios previamente registrados puedan iniciar sesión. El usuario debe ingresar sus credenciales y pulsar el botón ”Iniciar sesión” o, si lo prefiere, puede utilizar su cuenta de Google para acceder. Si el usuario aún no dispone de una cuenta, puede crear una nueva pulsando el enlace "¿No tienes una cuenta? REGÍSTRATE". En caso de haber olvidado su contraseña, se ofrece la opción de recuperarla mediante el enlace "¿Has olvidado tu contraseña?".

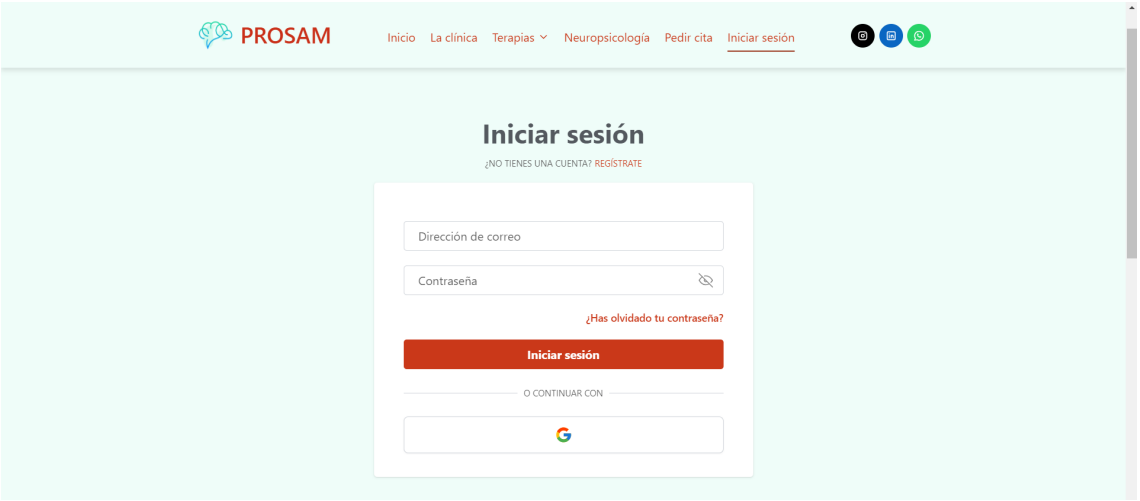

Figura 7.2: Pantalla: Iniciar Sesión

- **Recuperar contraseña:** Este proceso consta de dos pantallas. En la primera, el usuario registrado ingresa su dirección de correo electrónico y selecciona ”Restablecer contraseña". Tras este paso, recibirá un correo electrónico con un enlace para restablecer su contraseña. El enlace lo dirigirá a la segunda pantalla, donde deberá ingresar y confirmar su nueva contraseña antes de pulsar ”Restablecer contraseña” para completar el proceso.

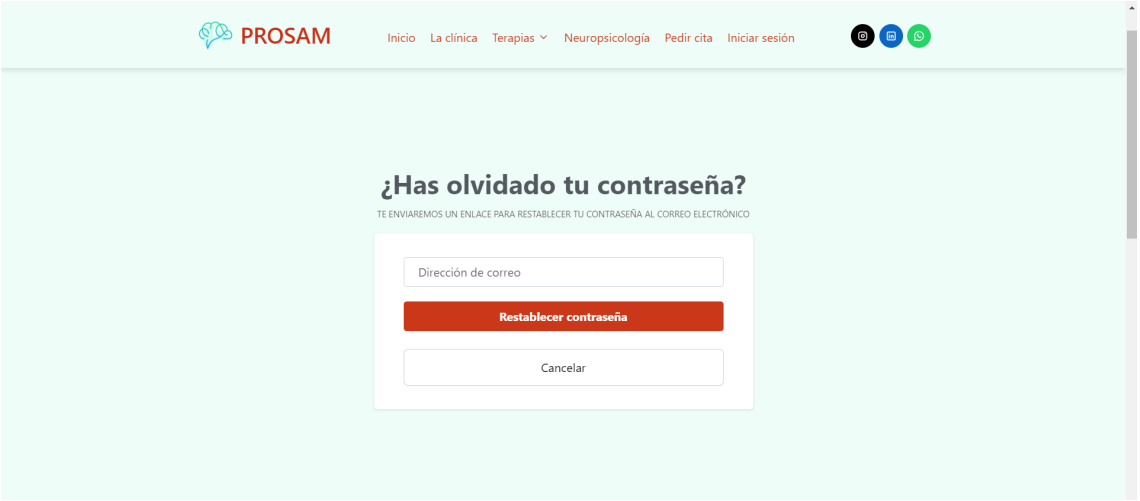

Figura 7.3: Pantalla: Recuperar contraseña [I]

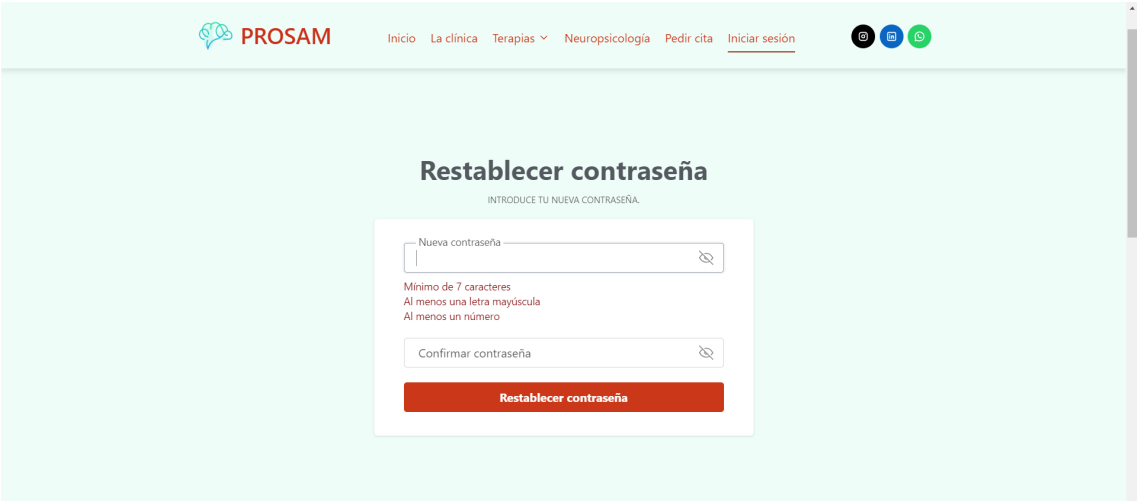

Figura 7.4: Pantalla: Recuperar contraseña [II]

- **Visualizar contenido de la web:** Las siguientes tres pantallas muestran la versión web de la página principal de la aplicación, destacando las mejoras visuales en la interfaz de usuario. Estas incluyen la fijación del encabezado al hacer scroll, la opción de adjuntar un CV para psicólogos en el formulario de contacto, y un pie de página mejorado, tal como se propuso en el capítulo de Análisis de Requisitos.

Figura 7.5: Pantalla: Visualizar contenido (Header)

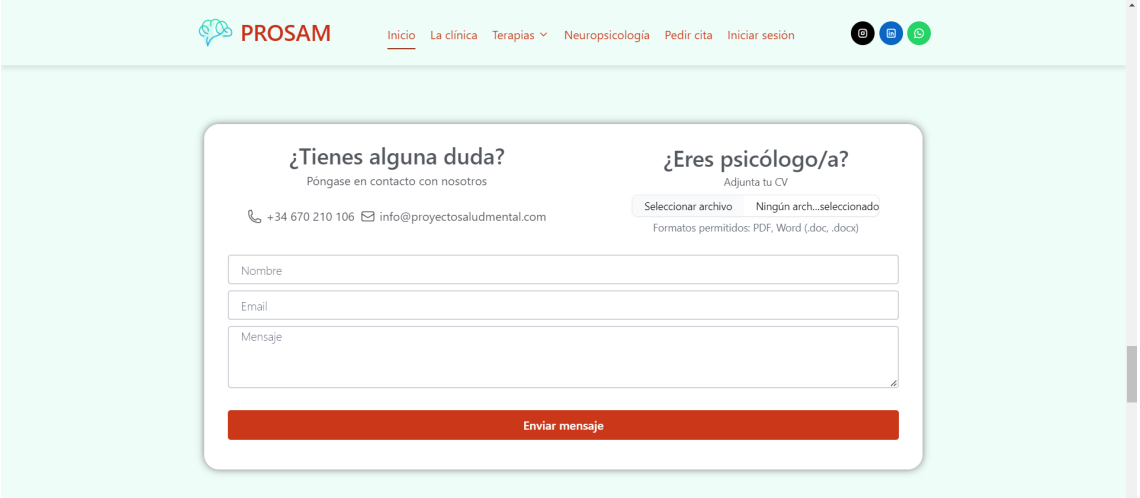

Figura 7.6: Pantalla: Visualizar contenido (Contacto)

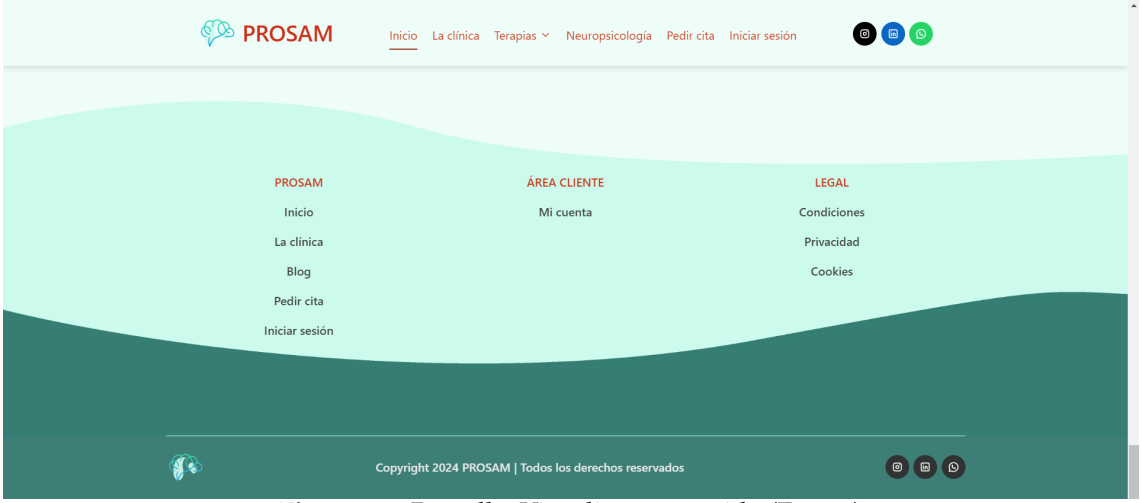

Figura 7.7: Pantalla: Visualizar contenido (Footer)

A continuación, se presentan dos capturas de pantalla que muestran la versión móvil de la aplicación, confirmando el cumplimiento del requisito de diseño respon-

sive. Estas pantallas reflejan cómo la interfaz se adapta a diferentes tamaños de pantalla, garantizando una experiencia de usuario óptima en dispositivos móviles.

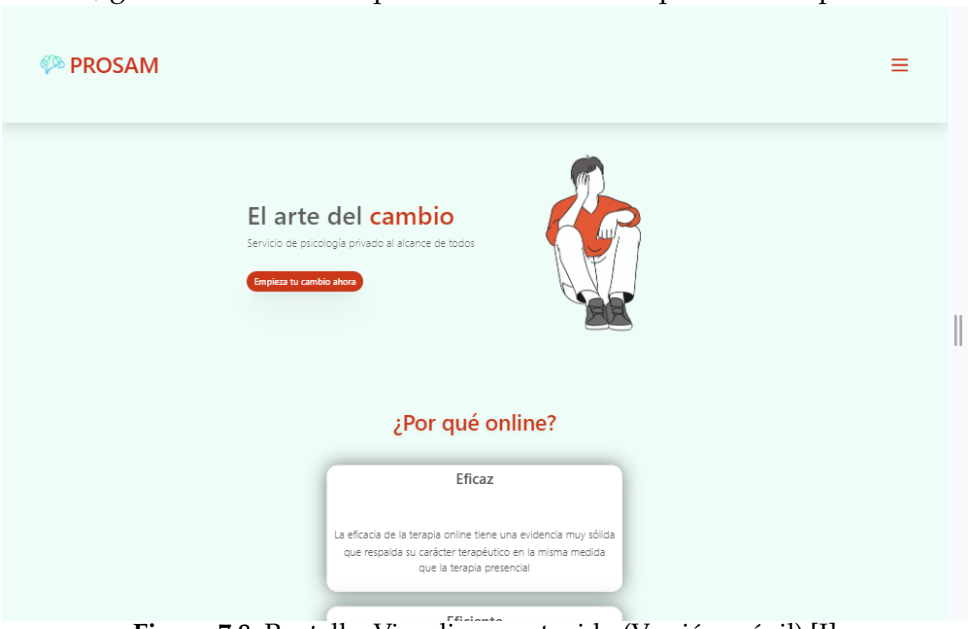

Figura 7.8: Pantalla: Visualizar contenido (Versión móvil) [I]

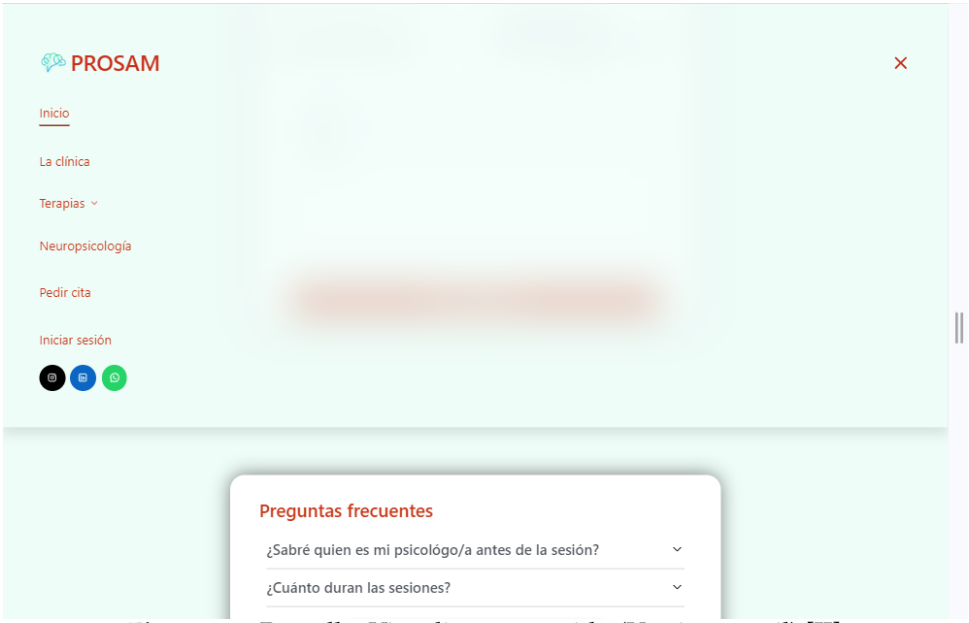

Figura 7.9: Pantalla: Visualizar contenido (Versión móvil) [II]

A lo largo de estas pantallas, el usuario puede explorar el contenido de la web mientras el sistema le indica la sección en la que se encuentra mediante la navegación visual. Si el usuario desea enviar un mensaje de contacto, deberá estar registrado e iniciar sesión previamente.

- **Acceder a mi cuenta:** Tras iniciar sesión, el usuario es redirigido a la pantalla de "Mi cuenta", donde puede gestionar diversas opciones según su rol en la aplicación.

  - **Paciente:** Un paciente puede visualizar su información personal en la sección de Ajustes, reservar citas, consultar detalles de sus próximas citas y finalizar la sesión.

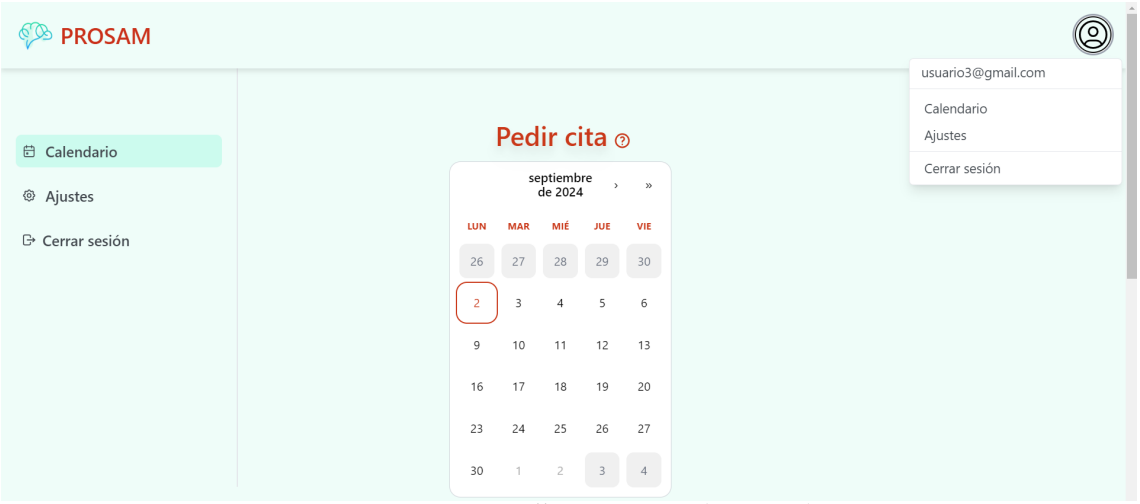

Figura 7.10: Pantalla: Mi cuenta (Paciente)

- **Psicólogo:** En el caso del psicólogo, se le presenta su información personal, la posibilidad de gestionar sus huecos disponibles, consultar o modificar citas futuras, y la opción de cerrar sesión.

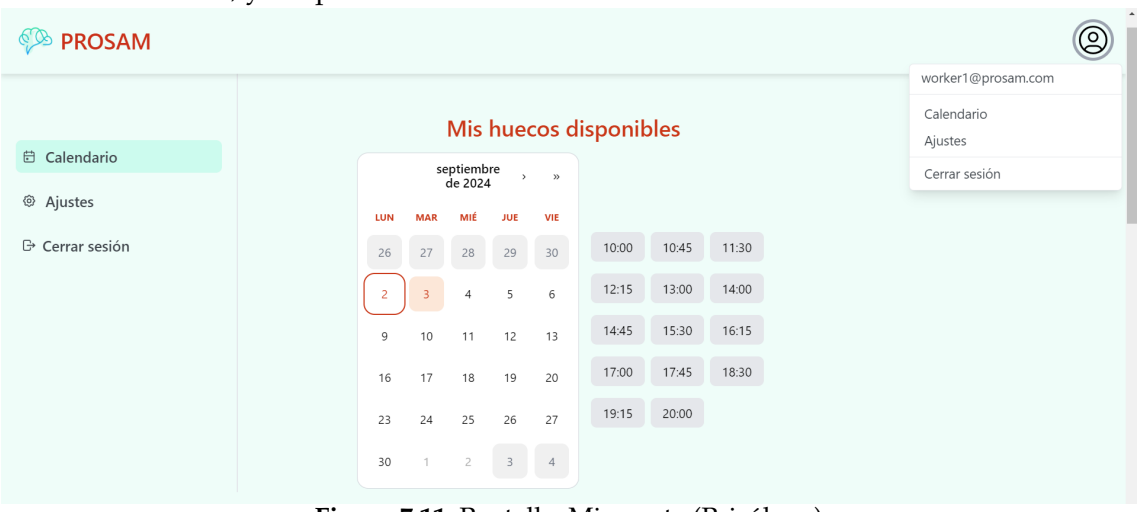

Figura 7.11: Pantalla: Mi cuenta (Psicólogo)

- **Administrador:** El administrador del sistema tiene acceso a una vista general de la disponibilidad del sistema, similar a la que ve el paciente, pero con la diferencia de que el administrador puede visualizar qué psicólogo está asignado a cada hueco disponible. Además, el menú de navegación difiere, incluyendo una sección para "Mensajes de contacto". El administrador también puede modificar los huecos disponibles para los psicólogos y finalizar la sesión.

Figura 7.12: Pantalla: Mi cuenta (Administrador)

- **Cerrar sesión:** La opción de cerrar sesión está disponible en la parte inferior del menú de navegación en todas las versiones de la pantalla "Mi cuenta"para pacientes, psicólogos y administradores (Figuras 7.10-7.12).

- **Ajustes del perfil:** En la pantalla de Ajustes, los usuarios pueden modificar su foto de perfil, actualizar su nombre y apellidos, así como cambiar su contraseña actual. Una vez realizados los cambios, deben hacer clic en el botón "Guardar cambios"para confirmar las modificaciones.

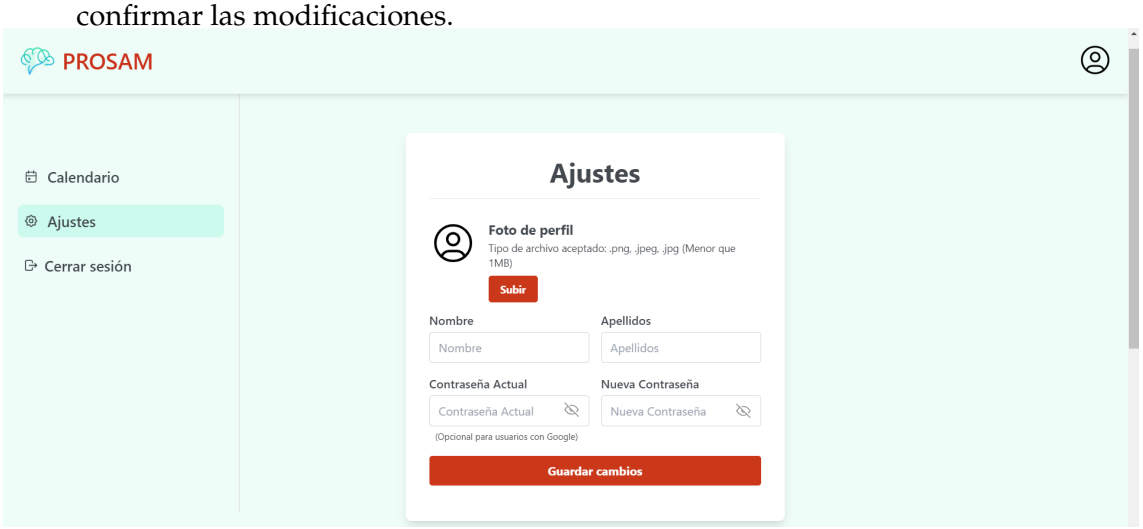

Figura 7.13: Pantalla: Ajustes del perfil

- **Calendario general:** Esta pantalla muestra el calendario de disponibilidad general de la aplicación. Los pacientes pueden seleccionar un día para ver las horas disponibles en esa fecha.

Figura 7.14: Pantalla: Calendario general

- **Visualizar mi calendario:** Esta pantalla es el calendario personalizado del psicólogo, donde puede seleccionar un día para ver los huecos disponibles en esa fecha. A diferencia del paciente, el psicólogo no puede reservar horas, solo gestionar su disponibilidad.

Figura 7.15: Pantalla: Visualizar mi calendario

- **Solicitar primera cita:** El proceso de solicitar la primera cita para un paciente se inicia desde la sección "Pedir cita" en la pantalla de "Mi cuenta". El paciente puede hacer clic en el icono de interrogación para abrir un modal que proporciona detalles sobre cómo funciona el proceso de reserva de una cita (Figura 7.16).

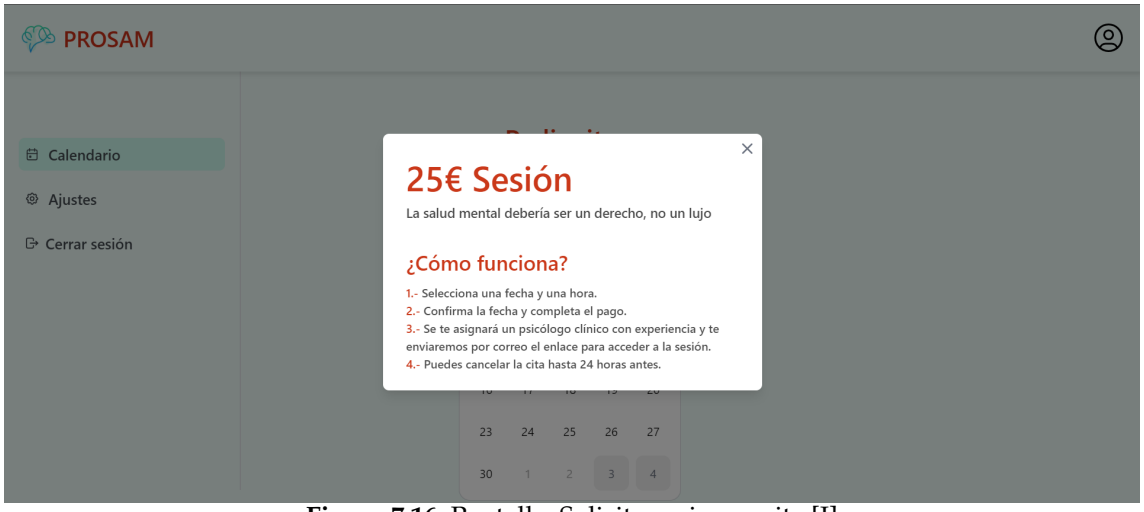

Figura 7.16: Pantalla: Solicitar primera cita [I]

Después de revisar la información, el paciente selecciona un día en el calendario para ver las horas disponibles en esa fecha (Figura 7.17).

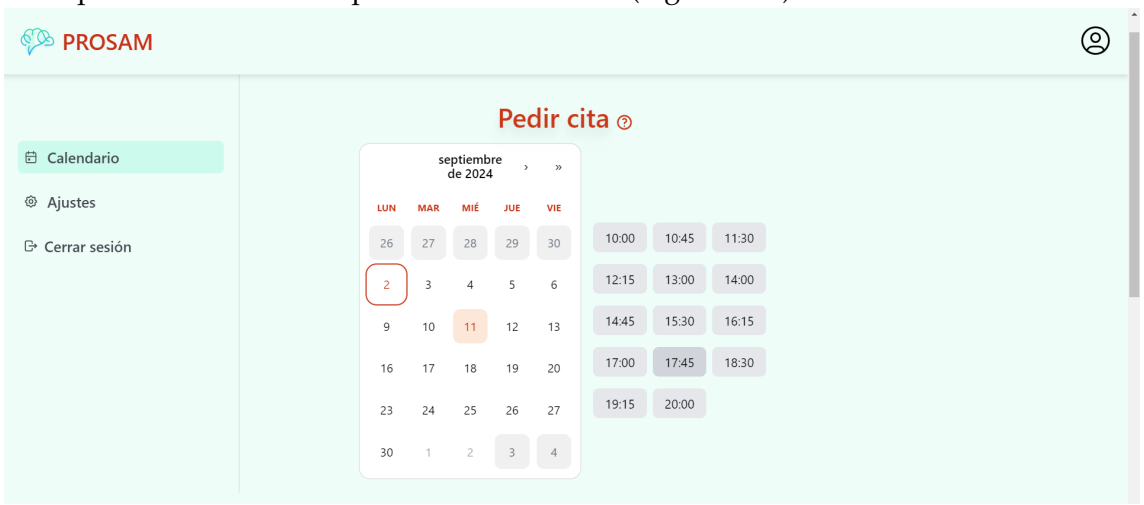

Figura 7.17: Pantalla: Solicitar primera cita [II]

Al hacer clic en una hora específica, aparece un modal de ”Confirmar cita", donde se muestra la fecha y hora seleccionada, junto con una explicación de lo que sucederá si el paciente confirma la reserva (Figura 7.18).

Figura 7.18: Pantalla: Solicitar primera cita [III]

Una vez confirmada la cita, el paciente es redirigido a Stripe, la plataforma de pagos, donde debe completar los campos requeridos para realizar el pago (Figura 7.19).

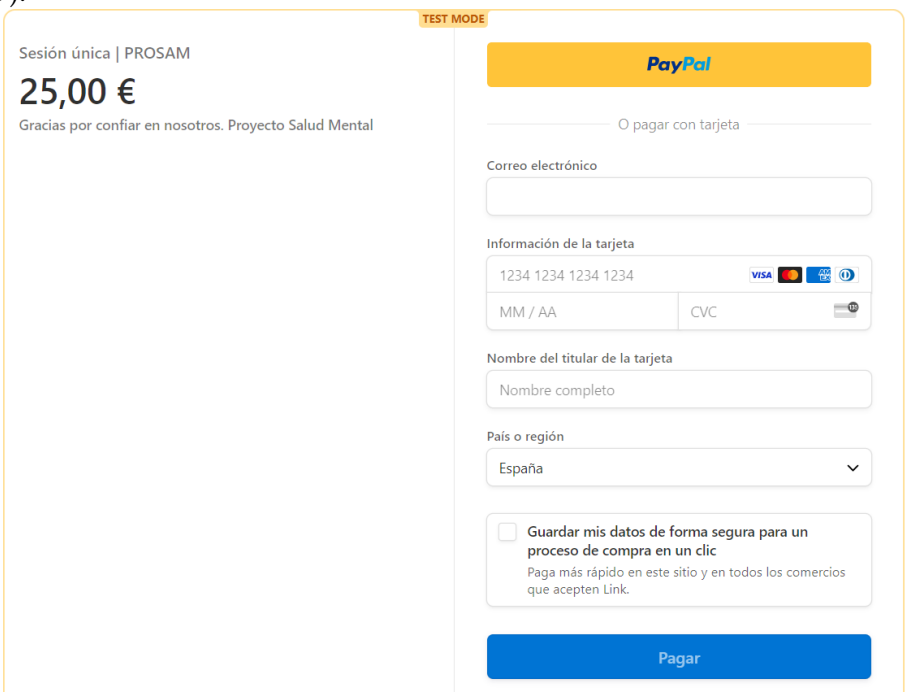

Figura 7.19: Pantalla: Solicitar primera cita [IV]

Tras completar el pago, el paciente es llevado a una pantalla de confirmación que indica que el proceso ha sido exitoso. En esta pantalla, se le informa que se ha

enviado un correo electrónico con los detalles de la cita, y se le ofrece la opción de regresar a su cuenta mediante un botón "Volver a mi cuenta"(Figura 7.20).

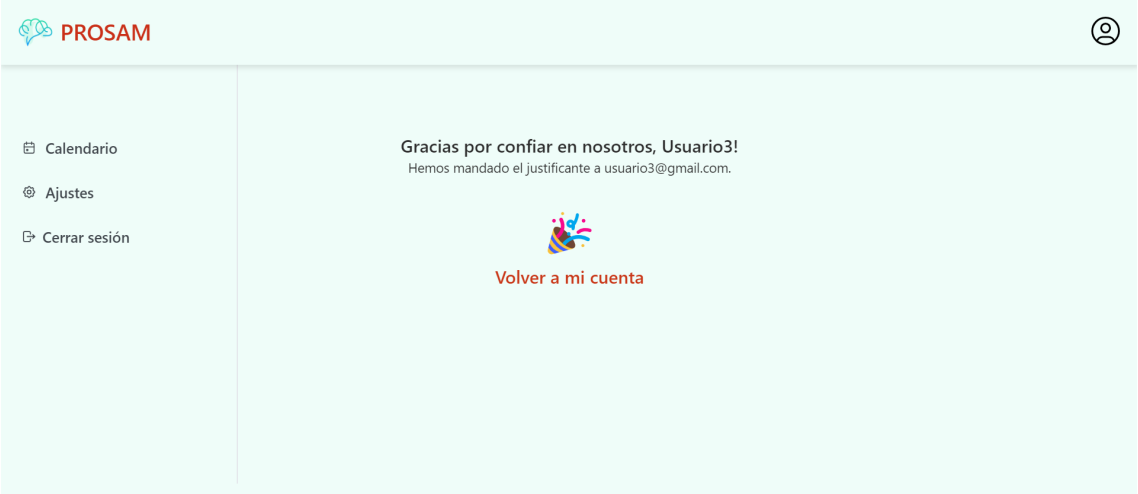

Figura 7.20: Pantalla: Solicitar primera cita [V]

- **Visualizar próximas citas:** En esta pantalla, los usuarios pueden acceder a la sección de "Próximas citas", donde tienen la posibilidad de filtrar las citas según su estado: "Pendientes" o "Completadas". Al hacer clic en el botón "Info", se muestra una vista detallada de la cita, que incluye la fecha, hora, estado, el psicólogo asignado y un enlace a la reunión (Figura 7.22). Si el usuario es un psicólogo, además de la información estándar, se muestra el nombre del paciente asignado a cada cita. Asimismo, el psicólogo cuenta con opciones adicionales para modificar la cita o marcarla como completada, lo que le permite gestionar su agenda de manera eficiente (Figura 7.23).

Figura 7.21: Pantalla: Visualizar próximas citas [I]

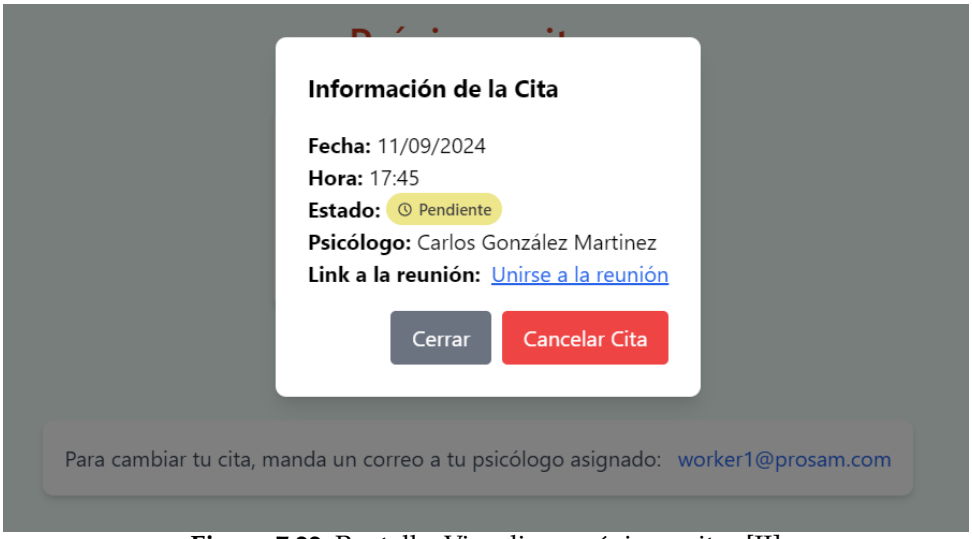

Figura 7.22: Pantalla: Visualizar próximas citas [II]

- **Modificar cita (Psicólogo):** Esta pantalla permite a los psicólogos gestionar sus citas de manera efectiva. Desde aquí, pueden cambiar la fecha o la hora de una cita, así como marcarla como completada una vez que se haya llevado a cabo. Esto garantiza que su calendario esté siempre actualizado y refleja con precisión el estado de todas las citas programadas.

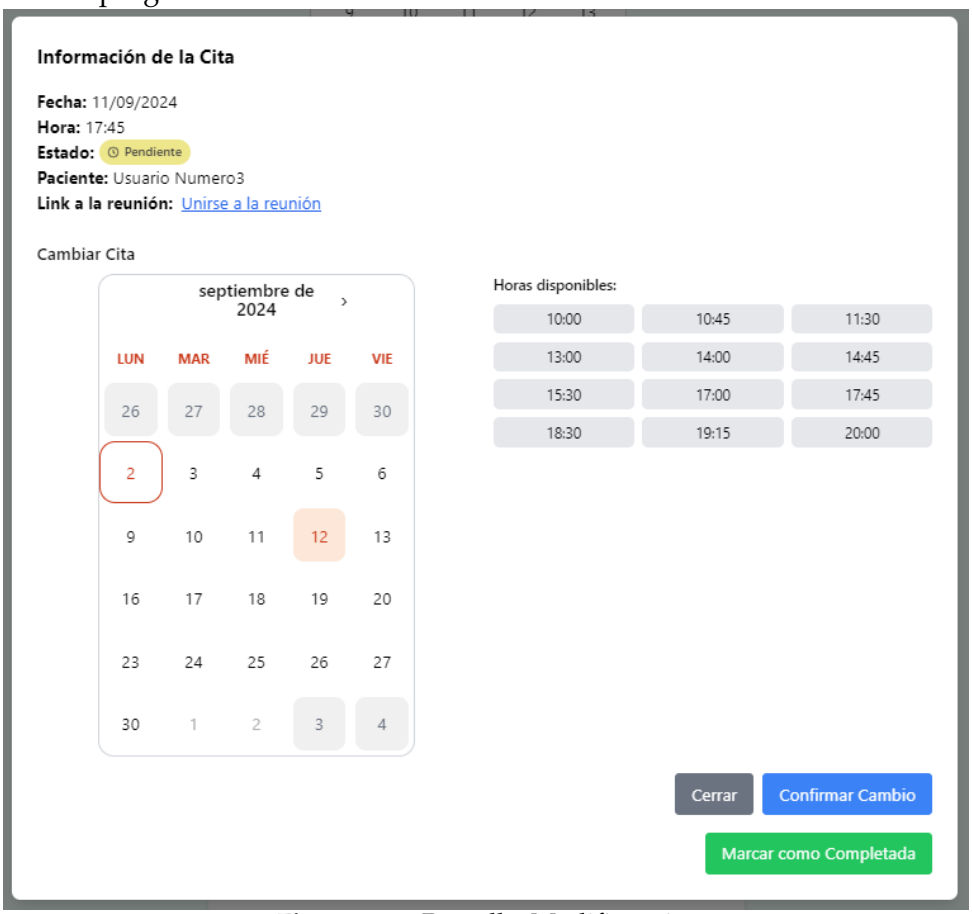

Figura 7.23: Pantalla: Modificar cita

- **Modificar cita (Paciente):** La pantalla que permite a los pacientes modificar una cita se ha presentado anteriormente en la Figura 7.21, donde se muestra la opción ”Cambiar cita". Al seleccionar esta opción, el paciente puede visualizar el correo

electrónico de su psicólogo asignado y debe comunicarle su disponibilidad para coordinar el cambio de la cita.

- **Cancelar cita:** La pantalla para cancelar una cita también se ha mostrado previamente en la Figura 7.22. En esta sección, se ofrece al paciente la opción de cancelar la cita, siempre y cuando falten 24 horas o más para la reunión. Esta funcionalidad permite al paciente gestionar sus citas con suficiente antelación, evitando posibles conflictos de horarios.

- **Cerrar hueco:** Esta pantalla permite al psicólogo gestionar su disponibilidad cerrando un hueco específico en su agenda. Esto puede ser necesario por razones personales o contractuales, como cuando no trabaja a tiempo completo. El psicólogo selecciona la fecha y la hora o las horas que desea bloquear y luego hace clic en el botón ”Cerrar hueco"para aplicar los cambios y actualizar su disponibilidad.

Figura 7.24: Pantalla: Cerrar huecos

En el caso del administrador del sistema, las siguientes dos pantallas ilustran cómo puede seleccionar a un psicólogo específico, indicar la fecha y la hora que se deben cerrar, y finalmente hacer clic en ”Cerrar hueco"para completar la acción.

Figura 7.25: Pantalla: Cerrar huecos (Administrador) [I]

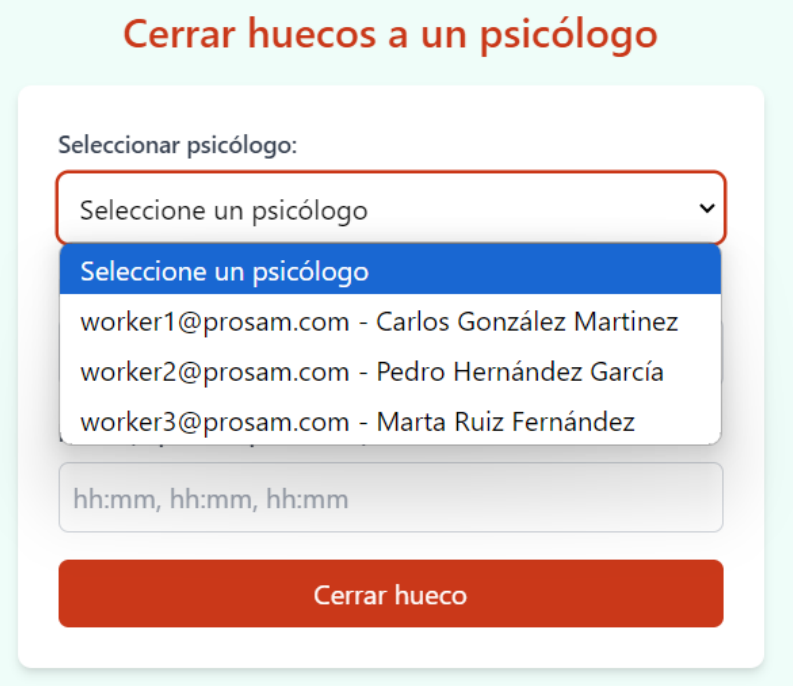

Figura 7.26: Pantalla: Cerrar huecos (Administrador) [II]

- **Realizar consulta:** Esta pantalla muestra una consulta en curso utilizando la plataforma *Jitsi*, donde dos usuarios están conectados para una sesión en línea. La interfaz permite a los usuarios realizar diversas acciones, como silenciar el micrófono, activar o desactivar la cámara, acceder al chat, y utilizar funciones como pedir la palabra, entre otras opciones.

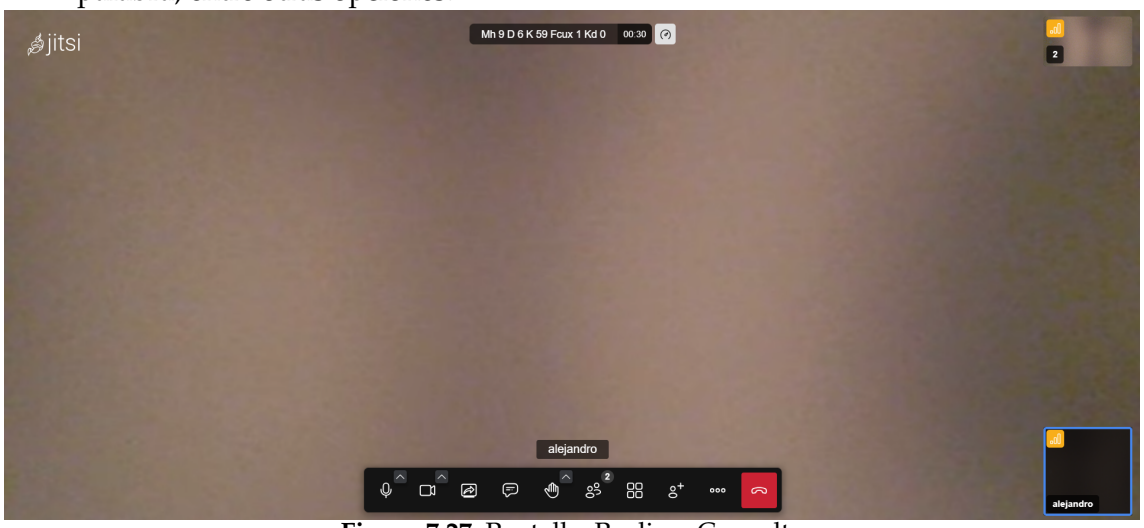

Figura 7.27: Pantalla: Realizar Consulta

- **Asignar próxima cita:** Esta funcionalidad está diseñada para que los psicólogos puedan asignar nuevas citas a los pacientes que ya han completado su primera sesión. En este proceso, el psicólogo primero selecciona al paciente de una lista desplegable (Figura 7.28). Una vez seleccionado el paciente, se abre un modal en el que el psicólogo puede elegir la fecha y la hora de la próxima cita y, posteriormente, confirmar la asignación haciendo clic en "Confirmar cita"(Figura 7.29). Tras la confirmación, el paciente recibe un correo electrónico indicándole que tiene 15 minutos para completar el pago antes de que la reserva provisional expire, ya que la cita solo se confirma de manera definitiva una vez que se ha realizado el pago (Figura 7.30).

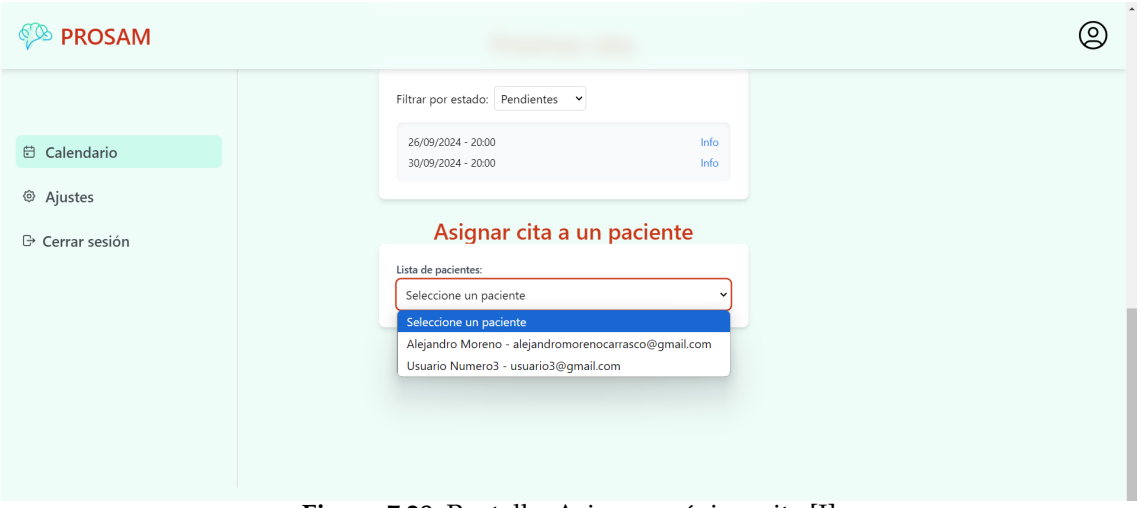

Figura 7.28: Pantalla: Asignar próxima cita [I]

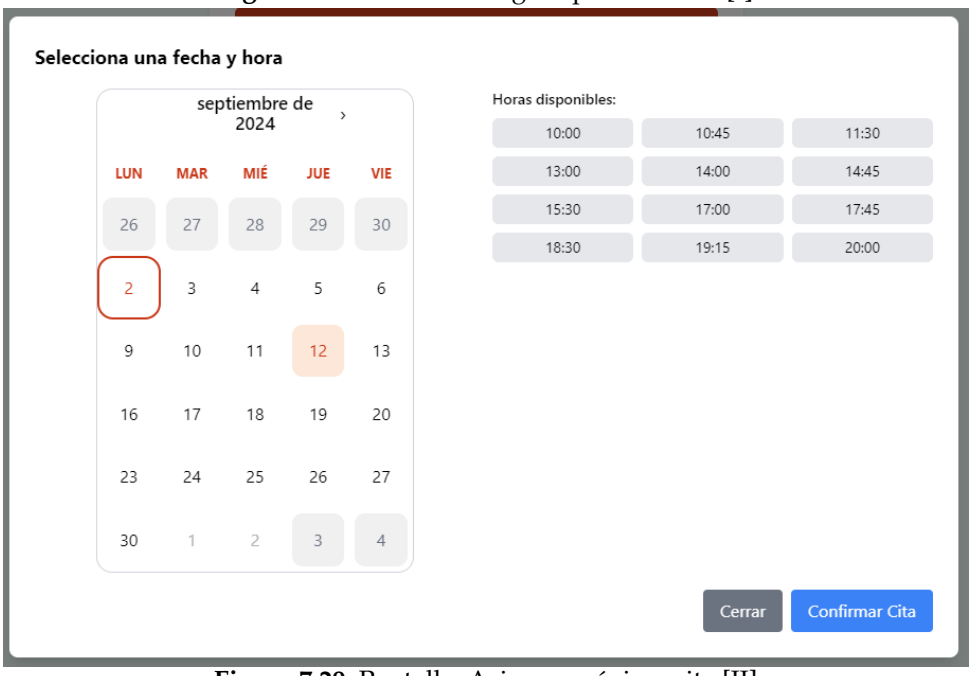

Figura 7.29: Pantalla: Asignar próxima cita [II]

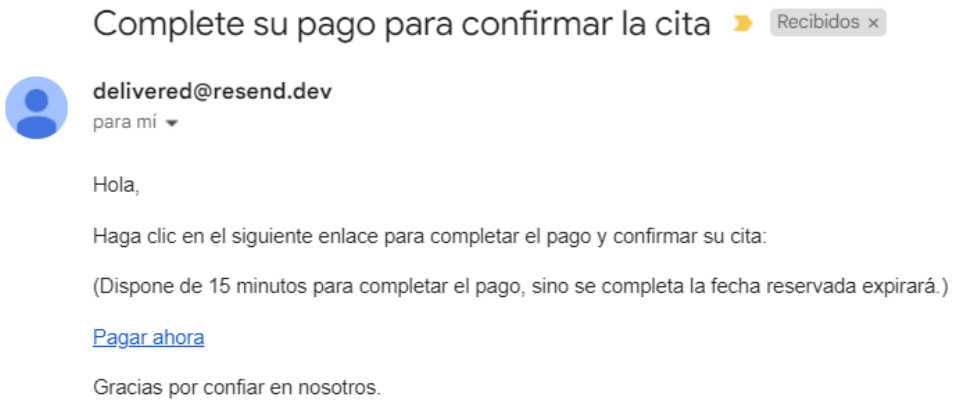

Figura 7.30: Pantalla: Asignar próxima cita [III]

- **Calendario en modo consulta:** Después de que el paciente haya realizado su primera consulta, el calendario se mostrará en modo consulta. Esto significa que el paciente ya tiene un psicólogo asignado, quien se encargará de establecer las futuras citas. En este modo, el paciente no puede modificar ni solicitar nuevas citas

directamente; si coloca el cursor sobre el icono de "Modo lectura", verá un mensaje que indica "Ya tienes un psicólogo asignado".

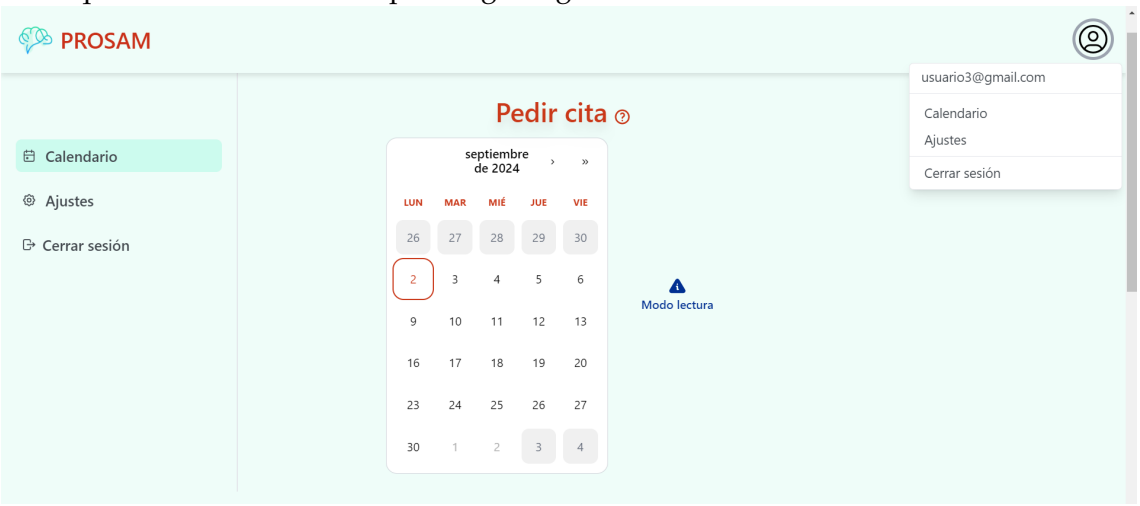

Figura 7.31: Pantalla: Calendario en modo consulta

- **Formulario de contacto:** El administrador del sistema tiene acceso a los mensajes enviados por los usuarios a través del formulario de contacto. Estos mensajes se presentan en una lista que el administrador puede recorrer mediante el scroll. Una vez revisados los mensajes, el administrador puede regresar al calendario principal utilizando el botón "Volver atrás".

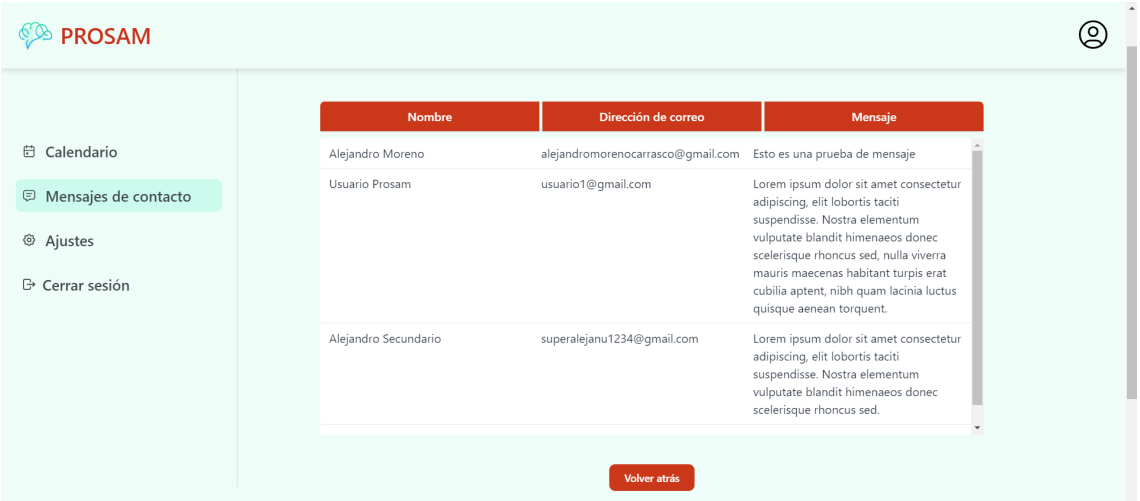

Figura 7.32: Pantalla: Formulario de contacto

CAPÍTULO 8
Validación

Para garantizar que la aplicación desarrollada cumpla con los estándares de usabilidad y ofrezca una experiencia de usuario óptima, se ha implementado un proceso de validación basado en la evaluación heurística. Este método se utiliza ampliamente en el ámbito de la experiencia de usuario (UX) para identificar problemas de usabilidad en interfaces a través de un análisis experto.

La evaluación heurística [30] se fundamenta en una serie de principios que han sido consensuados y reconocidos dentro de la comunidad de Diseño Centrado en el Usuario (UCD, por sus siglas en inglés). Estos principios fueron inicialmente propuestos por Jakob Nielsen en 1994 y han servido como una referencia sólida para evaluar la usabilidad de interfaces de usuario. A continuación, se describe cómo se han aplicado estos principios en la evaluación de la aplicación desarrollada:

- **Visibilidad del estado del sistema:** Para asegurar que los usuarios siempre estén conscientes del estado en el que se encuentra el sistema, se ha implementado un header de navegación que indica claramente la sección en la que se encuentra el usuario en todo momento (Header.astro).
- **Correspondencia entre el sistema y el mundo real:** La interfaz de usuario debe reflejar un orden lógico y natural que se asemeje al mundo real. En la aplicación, la información se presenta de manera intuitiva, replicando la experiencia de interacción de un entorno real, como la organización de citas en una clínica.
- **Control y libertad del usuario:** Para minimizar errores y permitir a los usuarios corregir acciones, la aplicación solicita confirmaciones adicionales en operaciones críticas. Un ejemplo es al reservar o cancelar una cita, donde se pide al usuario que confirme su acción antes de proceder, evitando así decisiones no deseadas.
- **Consistencia y estándares:** La aplicación sigue patrones de diseño que son coherentes con otras interfaces comunes, facilitando que los usuarios se familiaricen rápidamente con su uso. Esto incluye el uso de iconos, menús y flujos de navegación que son estándar en aplicaciones similares.
- **Prevención de errores:** La aplicación está diseñada para minimizar la posibilidad de errores, enfocándose en la validación de datos en tiempo real. Esto se gestiona en el *backend* utilizando *Zod*, biblioteca mencionada en anteriores capítulos que valida los datos antes de que se procesen.
- **Reconocimiento antes que memorización:** Para evitar que los usuarios tengan que memorizar instrucciones o rutas complejas, se ha diseñado la interfaz de manera que las opciones y comandos estén siempre visibles o sean fácilmente accesibles.

Además, cada botón está claramente etiquetado con su acción correspondiente, y se han implementado iconos y ayudas visuales como efectos de hover y menús desplegables para guiar al usuario de manera intuitiva.

- **Flexibilidad y eficiencia de uso:** La aplicación está diseñada para adaptarse tanto a usuarios novatos como a aquellos con más experiencia. Las funciones más comunes, como la gestión de citas y la navegación entre secciones, están fácilmente accesibles, lo que permite a los usuarios realizar sus tareas de manera rápida y eficiente sin importar su nivel de familiaridad con la plataforma.
- **Estética y diseño minimalista:** Solo se ha incluido la información necesaria en cada pantalla, evitando el desorden y asegurando un diseño limpio y estético. Además, se ha utilizado la herramienta *Cool Contrast* para garantizar que la paleta de colores sea asegurando que sea accesible para todos los usuarios, incluidas las personas con discapacidades visuales.
- **Ayudar al usuario a reconocer, diagnosticar y recuperarse de los errores:** Los mensajes de error en la aplicación están diseñados para ser claros y útiles, utilizando *react-hot-toast* para notificar al usuario sobre cualquier problema que ocurra durante el uso de la plataforma, indicando también sugerencias constructivas sobre cómo corregir el error.
- **Ayuda y documentación:** Aunque la aplicación es lo suficientemente intuitiva como para no requerir un manual extenso, se han incluido secciones de ayuda que son fácilmente accesibles para resolver dudas específicas. Por ejemplo, al lado de la opción "Pedir cita", aparece un icono que permite al usuario acceder a un modal que explica cómo realizar la solicitar una de manera clara.

La evaluación heurística aplicada ha sido esencial para verificar que la aplicación cumple con los principios fundamentales de usabilidad, asegurando una experiencia de usuario coherente, intuitiva y libre de errores. Además, la adaptabilidad de la aplicación a diferentes dispositivos, al ser completamente responsive, asegura que los usuarios puedan interactuar con la plataforma de manera eficiente, independientemente del entorno de uso. Aunque esta evaluación no sustituye las pruebas con usuarios reales, ofrece una base sólida que respalda la calidad y eficacia del diseño de la interfaz.

CAPÍTULO 9
Conclusiones y trabajos futuros

Después de completar el proyecto, se pueden extraer varias conclusiones sobre el proceso de desarrollo. En primer lugar, se definieron la motivación y los objetivos principales, que consistían en migrar la página web de la clínica de psicología PROSAM desde WordPress a una implementación más moderna utilizando el *framework* Astro. Este cambio buscaba ofrecer mayor flexibilidad y personalización, eliminando dependencias de servicios externos como *Eholo*. Tras analizar la situación actual de la empresa y las limitaciones de la web en WordPress, se establecieron nuevos requisitos, como la implementación de un sistema propio de gestión de citas, la autenticación de usuarios mediante cuentas de Google, y mejoras en la interfaz para hacerla más intuitiva. Se eligió una metodología incremental para el desarrollo, lo que permitió avanzar de manera continua y realizar mejoras progresivas. Con los requisitos definidos, se diseñó un modelo de dominio y una base de datos para soportar las nuevas funcionalidades. A continuación, se desarrollaron tanto el *frontend* como el *backend* de la aplicación, utilizando tecnologías modernas como *Astro* y *React* para el *frontend*, y *API Routes* y *Astro Actions* para el *backend*. Al finalizar la implementación, se presentaron los resultados del proyecto mediante capturas de pantalla, que demostraron cómo la nueva plataforma mejora y amplía las funcionalidades originales. Además, se validó la aplicación mediante una evaluación heurística basada en los principios de usabilidad de Jakob Nielsen.

En cuanto a trabajos futuros, es importante destacar una serie de mejoras que podrían incrementar significativamente la calidad y seguridad del proyecto. Estas modificaciones no solo optimizarían el rendimiento, sino que también fortalecerían la seguridad y ampliarían las funcionalidades actuales. Las mejoras propuestas son:

- **Capa de Cloudflare:** Una primera mejora a considerar es la optimización de la seguridad del servidor, que actualmente está alojado en Vercel. Aunque Vercel cuenta con un firewall robusto que ayuda a prevenir ciertos tipos de ataques, no es suficiente por sí solo. Para fortalecer la protección, sería ideal implementar Cloudflare como una capa adicional. Cloudflare es una de las plataformas líderes en infraestructura y servicios, que no solo ofrece seguridad SSE (Secure Service Edge), sino también una amplia gama de herramientas para desarrolladores. Al implementar Cloudflare "por delante"de Vercel, se lograría un nivel de seguridad mucho más alto, protegiendo el dominio y las operaciones de la aplicación de manera más eficaz.

- **Funcionalidad Administrador:** Otra mejora funcional sería dotar al rol de Administrador con la capacidad de asignar citas directamente entre pacientes y psicólogos. Esto implicaría seleccionar un usuario, elegir un psicólogo disponible, y generar automáticamente una cita en el sistema, lo que facilitaría la gestión de citas de manera más centralizada y eficiente.

- **Completar las pantallas restantes:** Es necesario completar las interfaces que aún faltan en la web, como las secciones de La Clínica, Terapias y Neuropsicología. Hasta ahora, el desarrollo se ha enfocado en mejorar la lógica de la aplicación y las funcionalidades principales, como el sistema de citas, lo que dejó en segundo plano estas secciones más estáticas.
- **Recordatorios automáticos:** Por último, aunque el sistema ya utiliza *Resend* para enviar correos electrónicos a los usuarios con diferentes propósitos, como restablecer contraseñas o confirmar citas, sería beneficioso expandir esta funcionalidad. Una mejora viable sería configurar el envío automático de recordatorios cuando falten 24 horas para una cita, tanto para el paciente como para el trabajador. Para mayor efectividad, también podría implementarse el envío de estos recordatorios vía SMS.

Estas mejoras se abordarán tras la entrega inicial del proyecto, pero antes de su implementación definitiva en PROSAM.

Durante el desarrollo de este proyecto, algunas de las asignaturas cursadas en el Grado resultaron particularmente valiosas para abordar con éxito los desafíos presentados. Una de las más relevantes fue **Desarrollo Web (DEW)**, donde adquirí habilidades fundamentales en tecnologías como HTML, CSS y JavaScript. Esta asignatura no solo me proporcionó una sólida base técnica, sino que también despertó mi interés por profundizar en el diseño y desarrollo de páginas web, motivándome a explorar más allá del contenido básico hacia aplicaciones más complejas y funcionales. Otra asignatura clave fue **Integración de Aplicaciones (IAP)**, que me permitió comprender en profundidad el concepto de middleware, así como la gestión de flujos de datos y la integración entre distintas aplicaciones o servicios. Finalmente, la asignatura de **Diseño Centrado en el Usuario (DCU)** tuvo un impacto significativo en la dirección del proyecto. A través de esta materia, aprendí a enfocar el desarrollo de la web en las necesidades y expectativas del usuario final, en este caso, la empresa PROSAM.

Bibliografía

[1] Astro. The web framework for content-driven websites. https://astro.build/ **(Consultado el 3 de septiembre de 2024).**

[2] Prosam. (Proyecto Salud Mental) https://proyectosaludmental.com/ **(Consultado el 3 de septiembre de 2024).**

[3] Eholo | Software de Psicología Online: Todo lo que un psicólogo necesita en un mismo lugar. https://eholo.health/es **(Consultado el 3 de septiembre de 2024).**

[4] Cloudinary Image, Video Management - Documentation (s.f.). https://cloudinary.com/documentation **(Consultado el 3 de septiembre de 2024).**

[5] Barak, A., Hen, L., Boniel-Nissim, M., Shapira, N. (2008). A Comprehensive Review and a Meta-Analysis of the Effectiveness of Internet-Based Psychotherapeutic Interventions. Journal Of Technology In Human Services, 26(2-4), 109-160. https://doi.org/10.1080/15228830802094429 **(Consultado el 3 de septiembre de 2024).**

[6] Guía para la redacción de casos de uso | Marco de Desarrollo de la Junta de Andalucía. (s.f.). https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416 **(Consultado el 3 de septiembre de 2024).**

[7] Pérez, A. (2021b, septiembre 6). Características y fases del modelo incremental. OBS Business School. https://www.obsbusiness.school/blog/caracteristicas-y-fases-del-modelo-incremental **(Consultado el 3 de septiembre de 2024).**

[8] Developer tools. (s.f.). Stripe Documentation. https://docs.stripe.com/development **(Consultado el 3 de septiembre de 2024).**

[9] Astro DB. (s.f.-b). Docs. https://docs.astro.build/es/guides/astro-db/ **(Consultado el 3 de septiembre de 2024).**

[10] Astro DB (Astro Studio). (s.f.-d). Docs. https://docs.astro.build/es/guides/astro-db/#astro-studio **(Consultado el 3 de septiembre de 2024).**

[11] Astro DB (Drizzle ORM). (s.f.-d). Docs. https://docs.astro.build/es/guides/astro-db/#drizzle-orm **(Consultado el 3 de septiembre de 2024).**

[12] LucidChart | Diagramming powered by Intelligence. (s.f.). https://www.lucidchart.com/pages **(Consultado el 3 de septiembre de 2024).**

[13] Vercel documentation. (s.f.-b). https://vercel.com/docs, https://docs.astro.build/es/guides/deploy/vercel/ **(Consultados el 3 de septiembre de 2024).**

[14] Visual Studio Code - Code editing. Redefined. https://code.visualstudio.com/ **(Consultado el 3 de septiembre de 2024).**

[15] Acerca de GitHub y Git - Documentación de GitHub. (s.f.). https://docs.github.com/es/get-started/start-your-journey/about-github-and-git#acerca-de-github **(Consultado el 3 de septiembre de 2024).**

[16] Tailwind CSS - Documentation. (2021, 17 junio). https://v2.tailwindcss.com/docs **(Consultado el 3 de septiembre de 2024).**

[17] React - Quick start. (s.f.). https://react.dev/learn **(Consultado el 3 de septiembre de 2024).**

[18] npm: react-calendar. (s.f.). Npm. https://www.npmjs.com/package/react-calendar **(Consultado el 3 de septiembre de 2024).**

[19] Documentation - react-hot-toast. (s.f.). https://react-hot-toast.com/docs **(Consultado el 3 de septiembre de 2024).**

[20] Codecalm. (s.f.). Tabler: Free and Open-Source HTML Dashboard Template. https://tabler.io/docs/getting-started **(Consultado el 3 de septiembre de 2024).**

[21] Garrixen, A. (s.f.). Cool Contrast - Optimize the accessibility of your colors. Cool Contrast. https://coolcontrast.vercel.app/ **(Consultado el 3 de septiembre de 2024).**

[22] API Route | Endpoints. (s.f.-c). Docs. https://docs.astro.build/es/guides/endpoints/ **(Consultado el 3 de septiembre de 2024).**

[23] Astro Actions. (s.f.). Docs. https://docs.astro.build/en/guides/actions/ **(Consultado el 3 de septiembre de 2024).**

[24] Zod | TypeScript-first schema validation with static type inference. (s.f.). GitHub. https://zod.dev/ **(Consultado el 3 de septiembre de 2024).**

[25] Lucia documentation. (s.f.). Lucia. https://lucia-auth.com/, https://lucia-auth.com/getting-started/astro **(Consultados el 3 de septiembre de 2024).**

[26] OAuth basics. (s.f.). Lucia. https://lucia-auth.com/guides/oauth/basics **(Consultado el 3 de septiembre de 2024).**

[27] Oslo documentation. (s.f.). Oslo. https://oslo.js.org/ **(Consultado el 3 de septiembre de 2024).**

[28] Resend - Introduction. (s.f.). Resend. https://resend.com/docs/introduction **(Consultado el 3 de septiembre de 2024).**

[29] Jitsi meet. (s.f.). https://meet.jit.si/ **(Consultado el 3 de septiembre de 2024).**

[30] Evaluación heurística, la metodología más utilizada en UX para medir la usabilidad de una interfaz (s.f.). https://www.mtp.es/blog/experiencia-de-usuario-blog/evaluacion-heuristica-la-usabilidad-una-interfaz/ **(Consultado el 3 de septiembre de 2024).**

APÉNDICE A

Anexo: ODS - Objetivos de Desarrollo Sostenible

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**	X			
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**		X		
ODS 9. **Industria, innovación e infraestructuras.**				X
ODS 10. **Reducción de las desigualdades.**	X			
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Figura A.1: Tabla Objetivos de Desarrollo Sostenible asociados al TFG

En 2015, las Naciones Unidas adoptaron la Agenda 2030 para el Desarrollo Sostenible, que representa una oportunidad para que los países y sus comunidades avancen hacia un futuro mejor para todos, sin exclusiones. Esta Agenda establece 17 Objetivos de Desarrollo Sostenible (ODS) que tienen como fin promover el crecimiento económico, atender las necesidades sociales y proteger el medio ambiente, aplicándose de manera universal.

Reflexión sobre cómo el TFG se vincula con los ODS, ordenados por el impacto (Alto, medio, bajo):

- **ODS 3 (Salud y bienestar):** La salud mental es un componente esencial del bienestar general y ha adquirido una relevancia creciente en la sociedad contemporánea. Este TFG se alinea estrechamente con el ODS 3, que promueve la salud y el bienestar, al centrarse en la prestación de servicios de psicología diseñados para abordar directamente las necesidades de salud mental de la población. En un contexto en el que los problemas de salud mental están en aumento, garantizar el acceso a servicios psicológicos de calidad es fundamental para mejorar la calidad de vida de las personas. A través de este proyecto, se busca no solo ofrecer tratamientos, sino también crear conciencia sobre la importancia de la salud mental, reduciendo el estigma asociado y promoviendo una sociedad más saludable y equilibrada. Este impacto es particularmente alto, dado que la salud mental afecta a todos los aspectos de la vida de una persona, desde su capacidad para trabajar hasta sus relaciones personales y su bienestar general.

- **ODS 10 (Reducción de las desigualdades):** El lema de la empresa, "Servicio de psicología privado al alcance de todos", demuestra un firme compromiso con la reducción de las desigualdades, alineándose directamente con el ODS 10. Este enfoque es especialmente relevante en un contexto donde la atención psicológica de calidad sigue siendo inaccesible para una gran parte de la población. Según datos, se estima que alrededor del 80 % de la población española no cuenta con los recursos necesarios para acceder a un tratamiento psicológico eficaz. PROSAM, consciente de esta realidad, se propone como una solución accesible, ofreciendo precios asequibles y servicios de alta calidad. De esta manera, PROSAM contribuye a cerrar la brecha en el acceso a los servicios de salud mental, garantizando que más personas puedan recibir la atención que necesitan sin que su situación económica sea una barrera.

- **ODS 8 (Trabajo decente y crecimiento económico):** El desarrollo de una aplicación web para una empresa de psicología tiene un impacto positivo en el ODS 8, ya que fomenta el empleo y mejora la calidad del trabajo en el sector de la salud mental. La implementación de esta tecnología permite optimizar el proceso de atención psicológica, lo que no solo mejora la eficiencia del servicio, sino que también crea oportunidades de empleo para psicólogos y otros profesionales del sector. Además, la mejora en la calidad del trabajo, derivada de la utilización de herramientas digitales, contribuye a un entorno laboral más favorable, donde los profesionales pueden desempeñar sus funciones de manera más efectiva, promoviendo así el crecimiento económico en el sector.

- **ODS 13 (Acción por el clima):** Aunque el impacto en el ODS 13 es relativamente bajo, al tratarse de un servicio online de psicología tiene un efecto positivo en la reducción de la huella de carbono. Al ofrecer sesiones de psicoterapia a distancia, se disminuye la necesidad de desplazamientos en automóvil, lo que contribuye, aunque sea de manera modesta, a la reducción de las emisiones.