La página **AuthPage** es el responsable de la autenticación del usuario. Este componente utiliza un formulario para capturar el correo electrónico y la contraseña, los cuales son validados para asegurarse de que cumplen con los requisitos establecidos.

Cuando el usuario envía el formulario, se invoca la función **submit().** Esta función verifica si el formulario es válido y, de ser así, realiza una solicitud de inicio de sesión a través del servicio **FirebaseService.** Si la autenticación es exitosa, se obtiene la información del usuario mediante la función **getUserInfo()**, la cual consulta la base de datos de **Firebase** para obtener detalles adicionales del usuario.

En caso de éxito, la información del usuario se almacena en el almacenamiento local y se le redirige a la página principal.

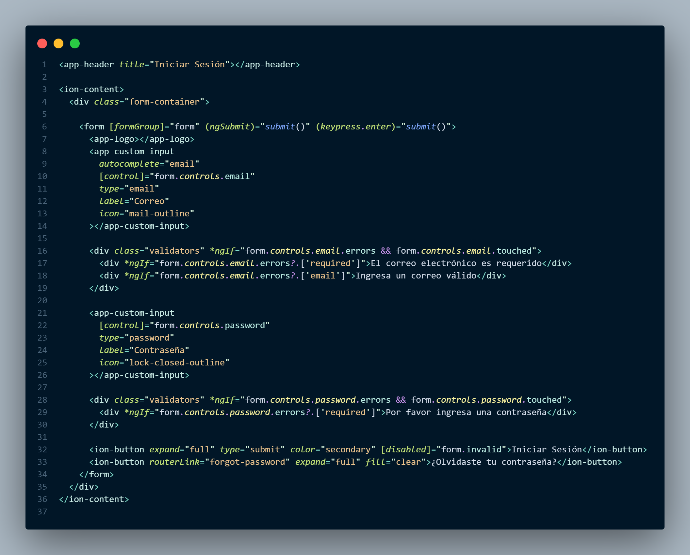
Dentro del código, existen conexiones con dos servicios: **FirebaseService** y **UtilsService**. En este caso, el método **loading()** se utiliza para mostrar un indicador de carga en la interfaz del usuario, con el fin de informar que la aplicación está trabajando en segundo plano.

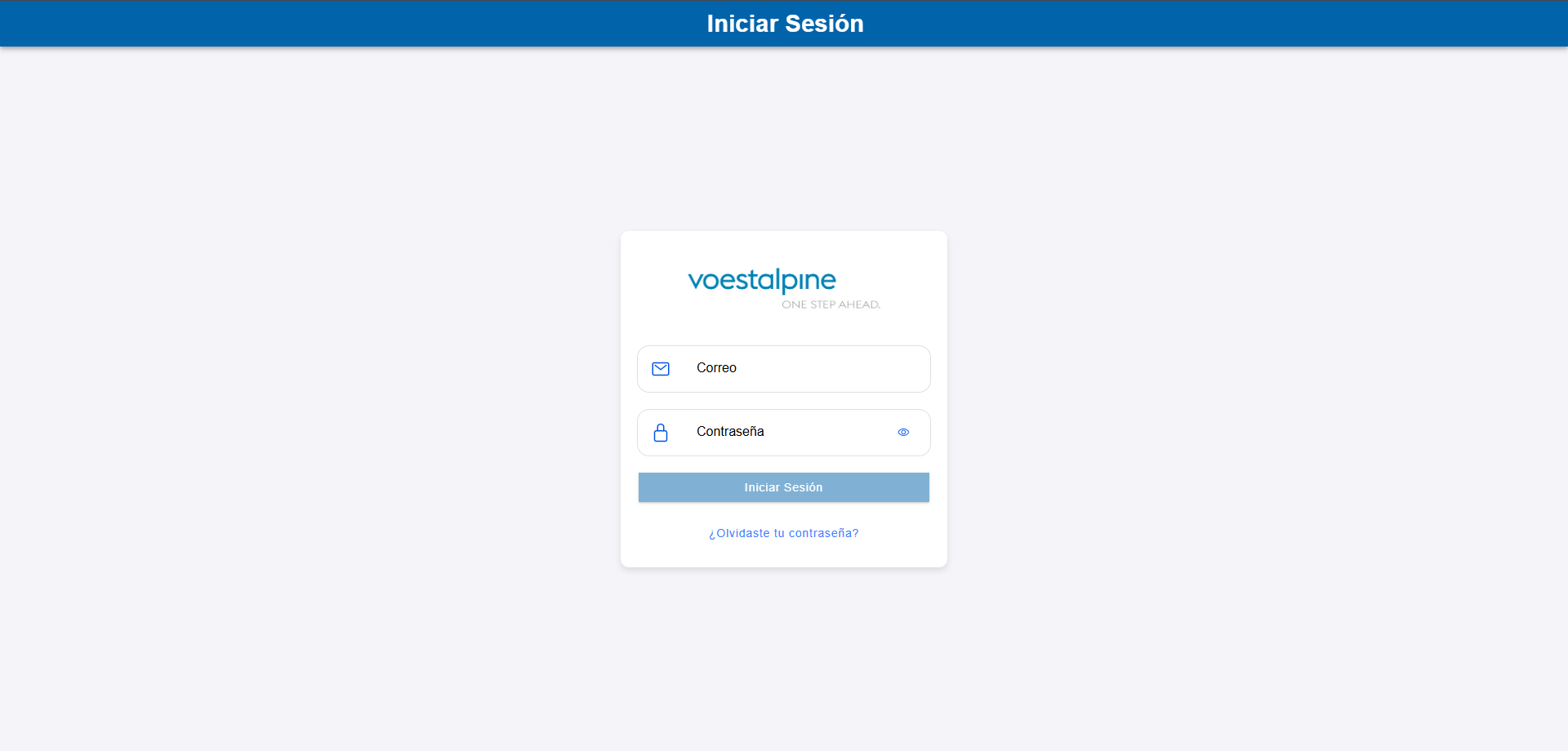
Luego, está el método **presentToast()**, el cual se emplea para mostrar un mensaje emergente en la interfaz con el propósito de notificar al usuario sobre algún evento o acción, como un mensaje de éxito, error o advertencia.

Otra conexión con un servicio es el método **SignIn**, el cual se llama desde **FirebaseService** y se utiliza para autenticar al usuario en **Firebase** mediante su correo y contraseña.

Luego, el método **getDocument()** se empleará para obtener información específica desde **Firestore**, en este caso, para recuperar los datos de un usuario.

Por último, tenemos el código de la interfaz gráfica con sus estilos, en el cual se tomó en cuenta un diseño simple pero agradable, utilizando una paleta de colores acorde con la identidad visual de la empresa.

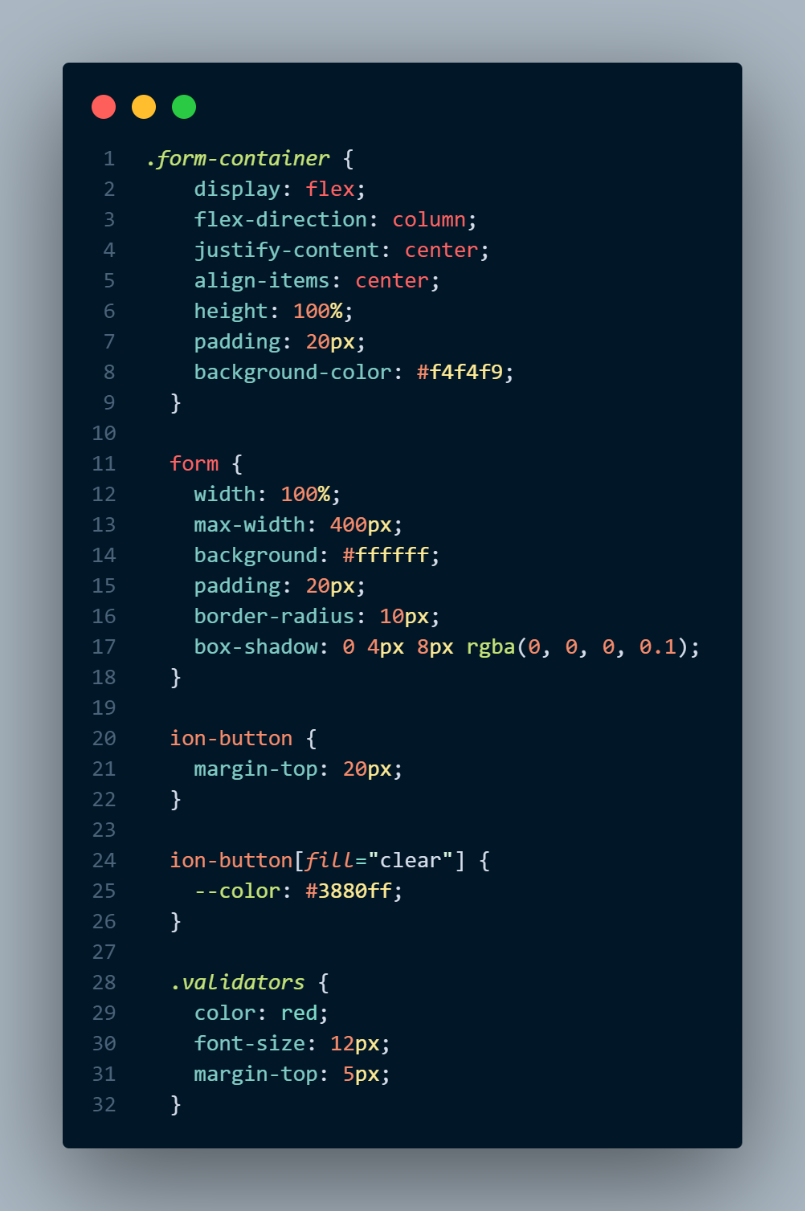


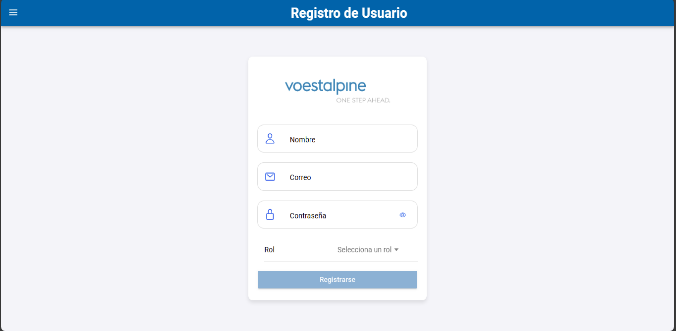


La página SignUpUserPage es una página de registro diseñada para permitir a los administradores crear nuevos usuarios. Este componente resulta útil en escenarios donde un administrador o un usuario con permisos especiales necesita registrar a otros usuarios manualmente.

El formulario de registro captura información como el correo electrónico, la contraseña, el nombre y el rol del usuario, aplicando validaciones específicas en cada campo. Cuando el formulario es enviado, se ejecuta el método submit(), el cual primero verifica si el formulario es válido y, como segunda verificación, requiere que el usuario con rol de administrador ingrese su contraseña.

Este componente tiene una conexión con FirebaseService, donde el método correspondiente tiene como objetivo crear un nuevo usuario en Firebase sin que la sesión del usuario actual se cierre. Primero, solicita los datos del usuario a registrar y, al presionar Enviar, requiere la contraseña del usuario actual para verificar su identidad.

Por último, se encuentra el aspecto de la interfaz gráfica, donde se detallan todos los elementos necesarios para el ingreso de los datos del usuario.



La página **HomePage** se encarga de gestionar y mostrar la lista de candidatos. Este componente se conecta con el servicio **FirebaseService** para obtener los datos de los currículums almacenados en **Firebase** y permite a los usuarios filtrar y visualizar a los candidatos según su estado (por ejemplo, Pendiente de Revisión, Aprobado, Rechazado, etc.).

Cuando la página se carga, se invoca la función **ngOnInit()**, que realiza una solicitud a **Firebase** para obtener los datos de los postulantes. Una vez obtenidos, se generan recomendaciones automáticas basadas en el estado de los candidatos, proceso que se lleva a cabo mediante la función **generateRecommendations()**.

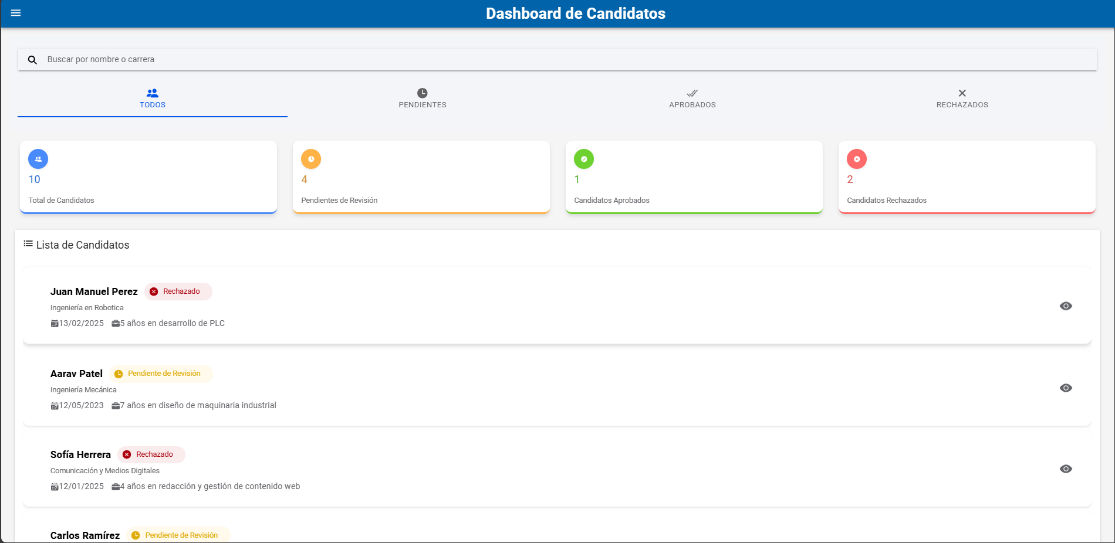
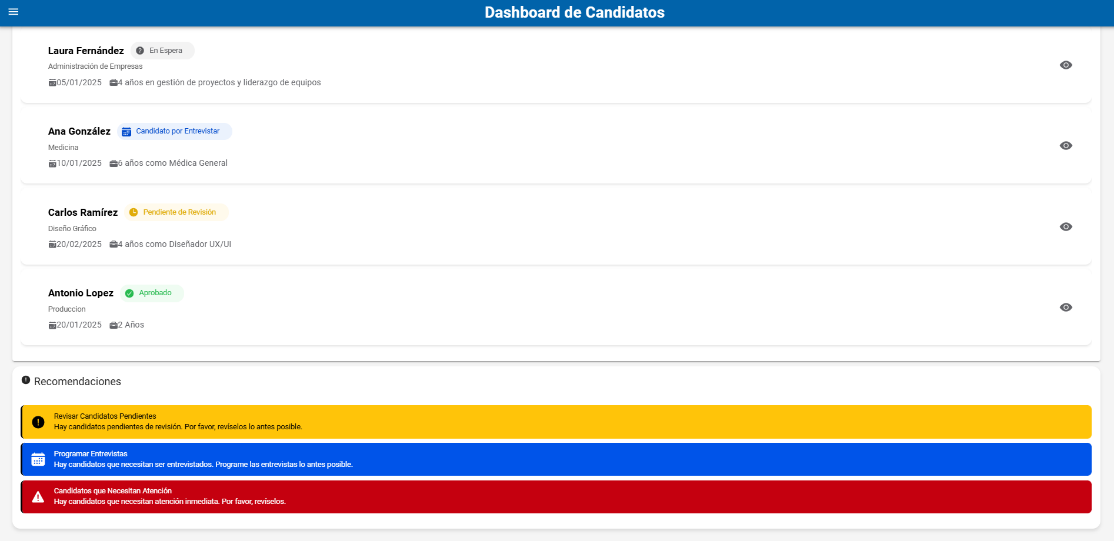
En este caso, se cuenta con aspectos importantes, como el filtrado de candidatos por nombre, carrera o estado, mediante la función **applyFilters()**. Otra función relevante es la de recomendación automática, la cual analiza el estado de los candidatos y genera sugerencias para el reclutador a través de la función **generateRecommendations()**.

Además, se incluye la visualización de los detalles del postulante, donde se habilita un botón que permite acceder a toda la información del candidato mediante la función **viewDetails()**.

Para encontrar los currículums o documentos de un postulante, se utiliza una función del servicio **FirebaseService** llamada **getAllCurriculums()**. Su propósito es obtener todos los documentos de la colección almacenada en Firestore.

La interfaz gráfica de la página HomePage está diseñada para ser intuitiva y fácil de usar. Se utilizan tarjetas para mostrar la información de cada candidato, junto con los indicadores visuales de estado. Además, se incluye un campo de búsqueda y unas tarjetas de estado para visualizar cantidad de candidatos.





La página **CandidatesPage** permite a los usuarios, en este caso, a los reclutadores, visualizar, filtrar y gestionar los currículums de los candidatos. Este componente se conecta a **Firebase** para obtener los datos de los postulantes publicados y proporciona herramientas de filtrado por nombre, carrera y experiencia, además de la opción de limpiar los filtros en caso de no requerir uno en específico.

La función principal de este componente es la carga de datos a través de la función **getAllCurriculums()**, la cual obtiene todos los datos almacenados de los usuarios y los mapea en un formato estándar. Además, la función **getAllUsers()** obtiene los usuarios registrados en **Firestore**, los almacena y los muestra en la interfaz.

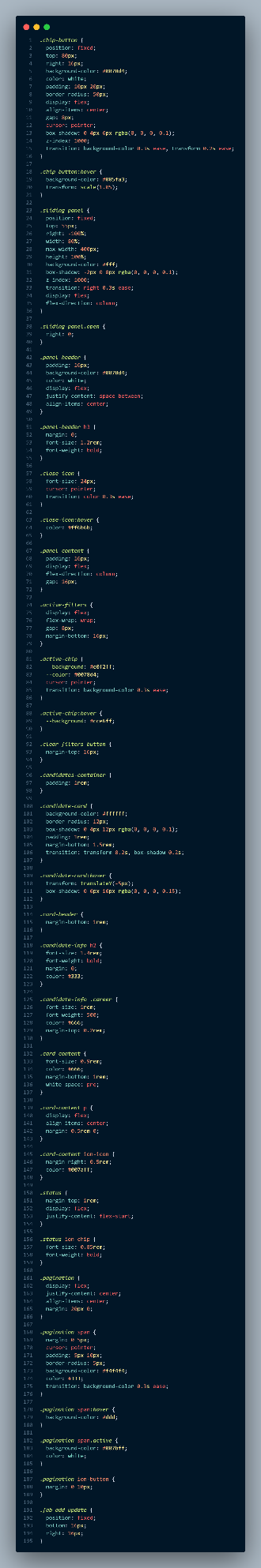
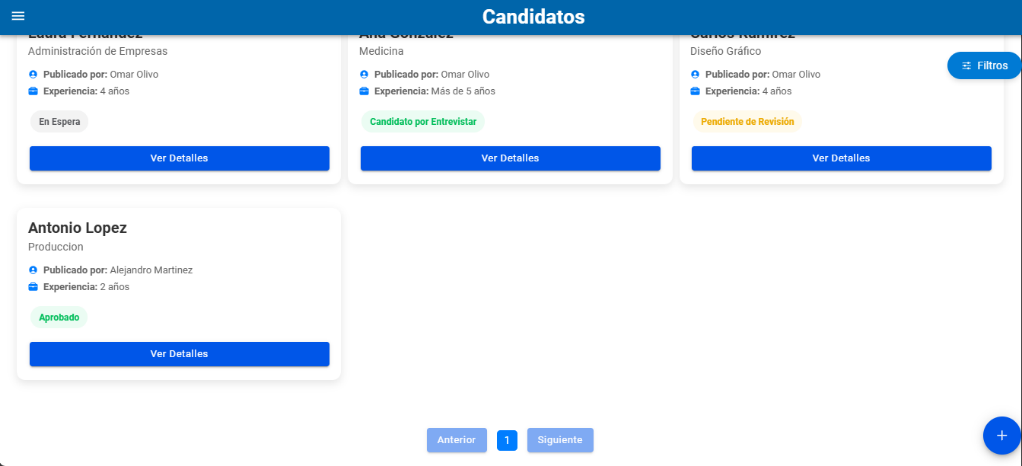
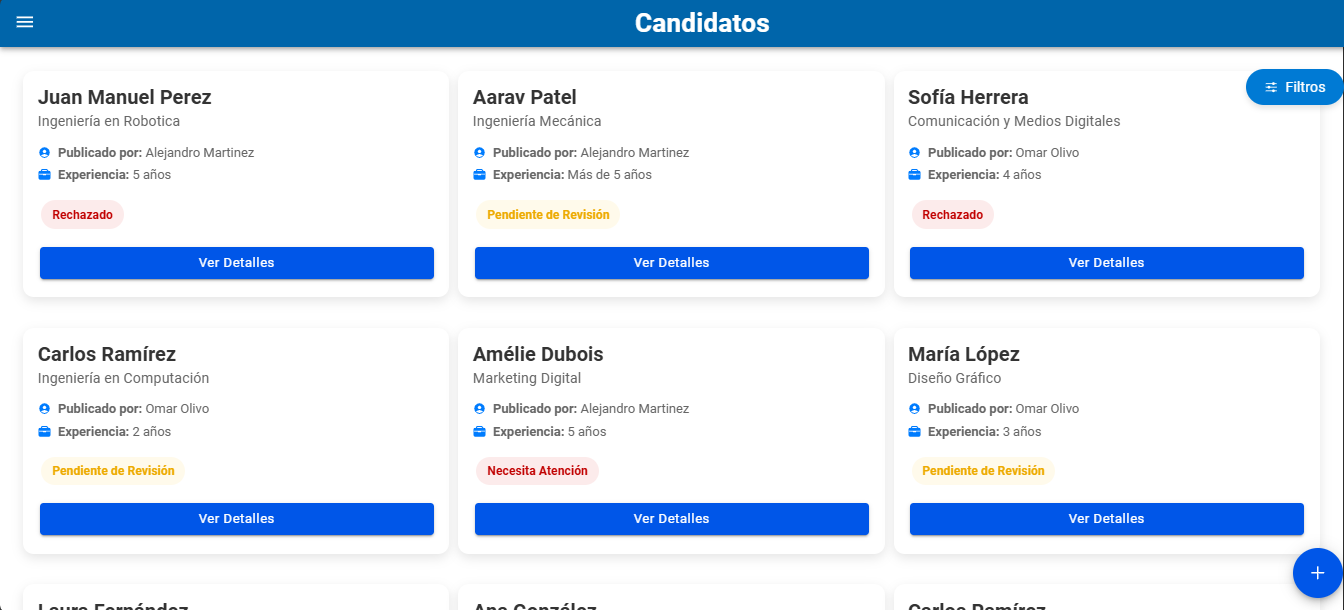
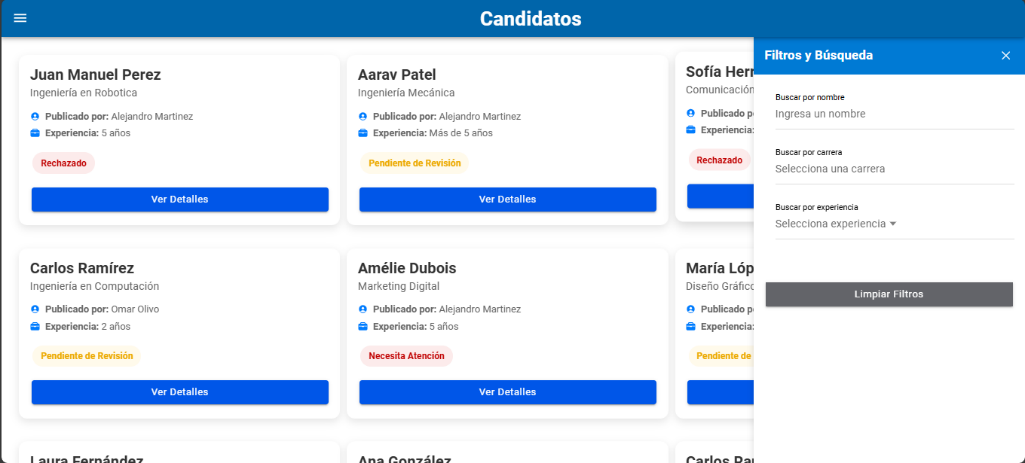
En cuanto al filtrado de candidatos, este se realiza por nombre, carrera y experiencia mediante la función **applyFilter()**. La función **selectCareer()** permite seleccionar una carrera específica, mientras que las funciones **clearFilter()** y **clearAllFilters()** ofrecen la opción de limpiar los filtros.

Adicionalmente, se incluye un sistema de paginación a través de la función **getPaginatedCurriculums()**, la cual devuelve los currículums de la página actual. También se proporciona la opción de navegar entre varias páginas mediante las funciones **nextPage(), previousPage() y goToPage()**. Finalmente, se cuenta con un generador de arreglos que muestra los números de páginas disponibles en el sistema mediante la función **getPages()**.

Por último, se implementa un proceso de normalización de datos con el fin de mejorar la precisión en la búsqueda de candidatos. Para ello, se utiliza la función **normalizeExperiences()**.

La interfaz gráfica utiliza un listado de currículums que muestra la información básica de los postulantes. Además, incluye un panel de filtros que permite buscar candidatos por nombre, carrera o experiencia. También se cuenta con un sistema de paginación que facilita la navegación entre diferentes vistas en caso de tener un gran número de postulantes.





El componente **AddUpdateCVsComponent** es un formulario que permite crear nuevos postulantes o editar la información de los existentes. Este formulario incluye campos como nombre, carrera, fecha de registro, correo electrónico, teléfono, ubicación, experiencia, idiomas, habilidades, disponibilidad, currículum (archivo PDF), notas y estado del postulante.

Además, este componente se integra con **Firebase** para guardar y actualizar los datos en **Firestore** y almacenar los archivos PDF en **Firebase** Storage.

Este formulario cuenta con validación en todos sus campos; si no son completados, no será posible publicar al postulante, por lo que es obligatorio ingresar toda la información requerida.

Una vez que los datos han sido cargados, se dispone de una opción de actualización. Este método recupera los datos previamente publicados y los coloca en los campos correspondientes, con el fin de que el usuario pueda identificar errores o actualizar la información.

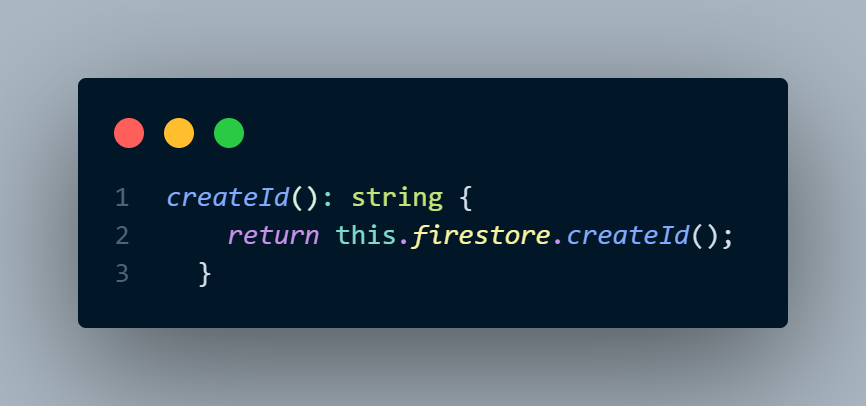
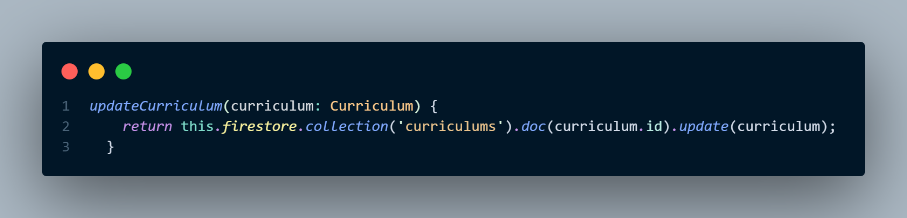
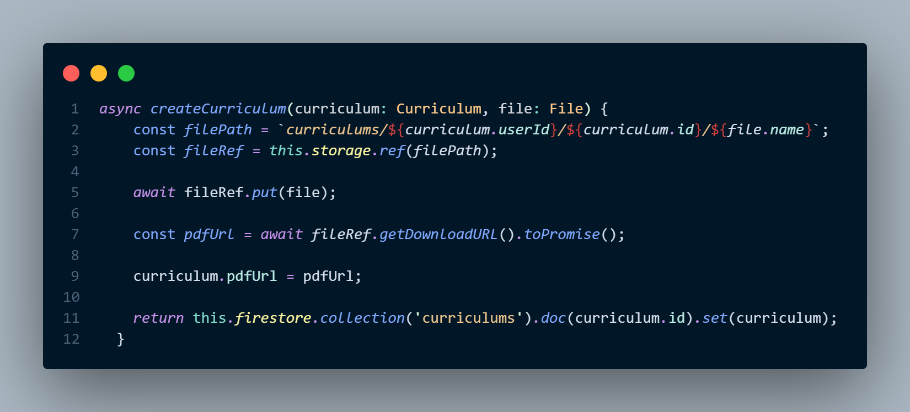
En cuanto a la subida de archivos, este componente solo permite cargar documentos en formato PDF. Además, cuenta con una función de un botón para eliminar un archivo existente y subir uno nuevo en caso de ser necesario.

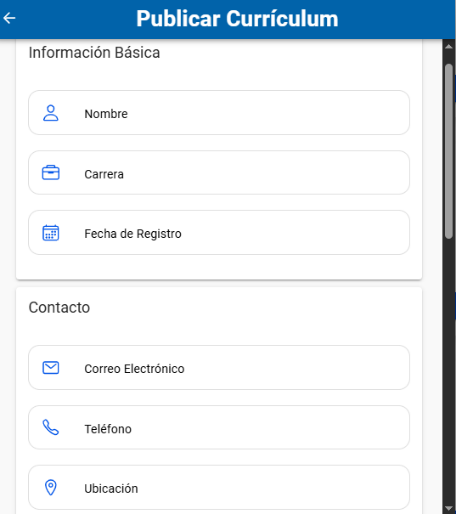
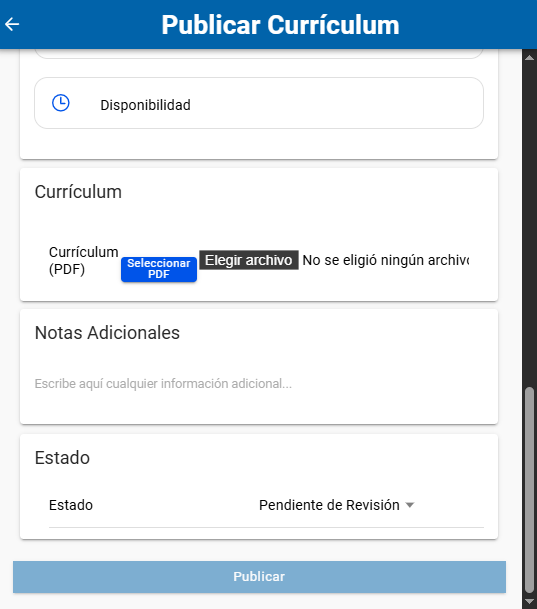
Este componente se conecta con el servicio **AuthService** a través del método **getCurrentUser()**, el cual permite obtener al usuario autenticado. Esto es útil para acceder a información como su nombre, correo electrónico o ID.

Otra conexión importante es con el método **createId()** del servicio de **Firebase**, cuya función es generar un ID único para identificar a cada postulante al momento de su creación. Este ID también es útil al realizar una actualización, ya que permite identificar al usuario cuyos datos serán modificados.

Asimismo, el componente se conecta con el método **updateCurriculum()** del servicio de **Firebase**, utilizado para actualizar los datos del postulante, ya sea información personal o archivos asociados.

Por último, se cuenta con la conexión al método **createCurriculum()** del servicio de **Firebase**, cuya función es permitir la creación de un nuevo postulante mediante el ingreso de datos y la subida de archivos en formato PDF.





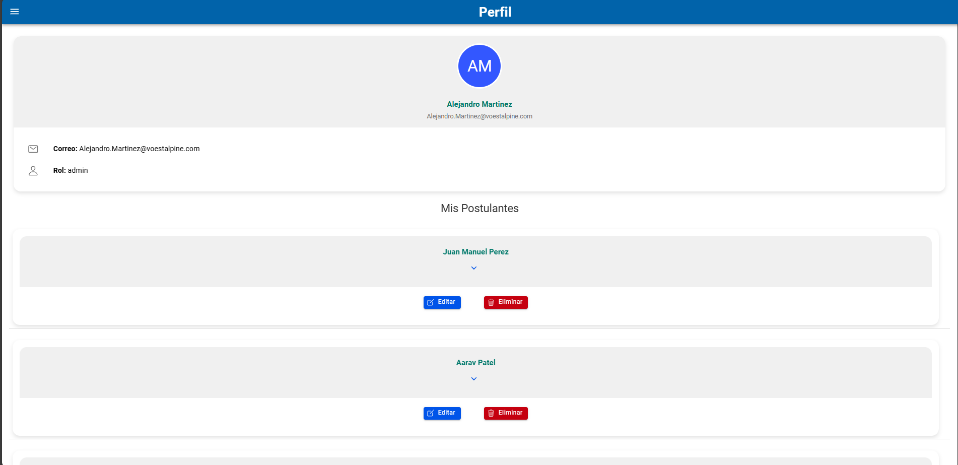
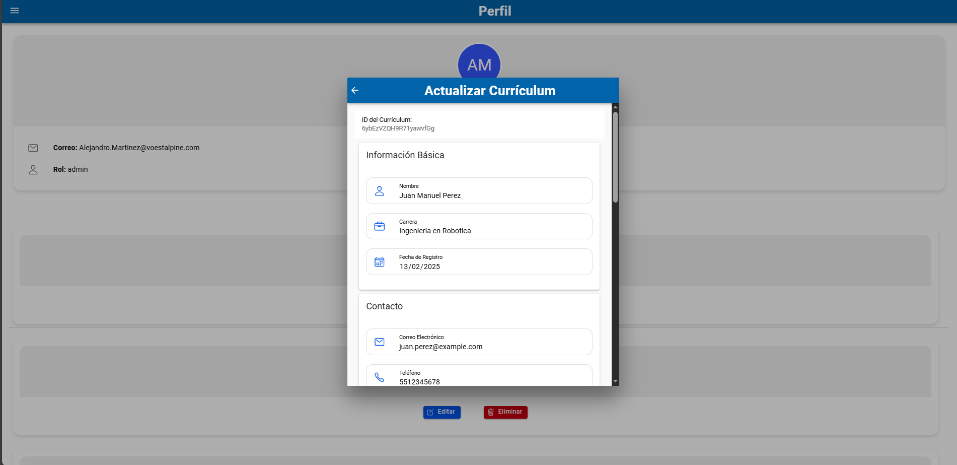
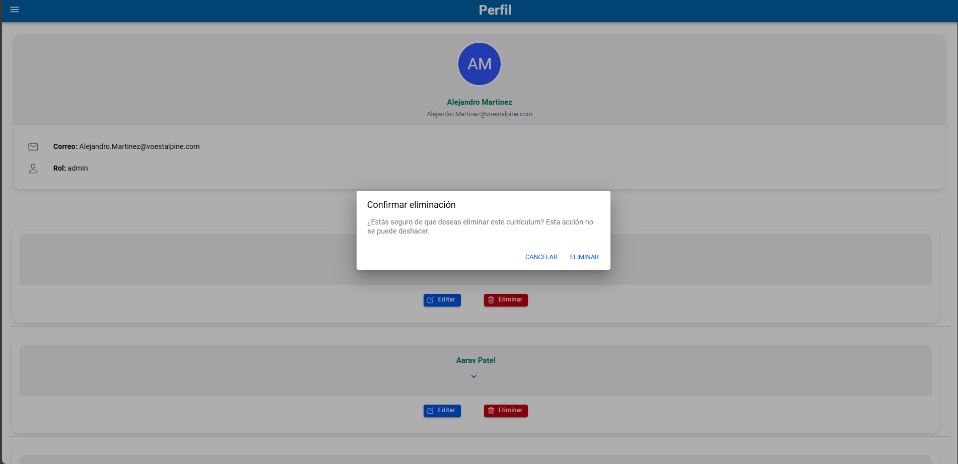
La página **ProfilePage** tiene la funcionalidad de mostrar información del usuario, como su nombre, correo y rol, además de un avatar, que en este caso está representado por las primeras letras del nombre y apellido.

También se muestra un listado de las publicaciones realizadas por el usuario, con opciones para ver detalles, editar o eliminar a los candidatos postulantes.

Este componente cuenta con la función **ngOnInit()**, la cual obtiene el ID del usuario autenticado para cargar su perfil. A través de esta función, se obtienen tanto los datos del usuario como la lista de candidatos publicados, utilizando **loadCurriculums()**. Estas publicaciones incluyen una función llamada **showDetails**, que permite mostrar u ocultar los detalles, visualizando únicamente el nombre del postulante.

Además, se dispone de un botón de actualización que, al presionarlo, abre el modal **AddUpdateCVsComponent**, permitiendo modificar los datos del postulante. Por último, hay un botón de eliminación que, al ser presionado, muestra una alerta de confirmación. Si el usuario confirma la acción, la publicación se eliminará.





El componente **ChatPage** permite la interacción entre el usuario y un servicio de inteligencia artificial, además de integrar la selección de postulantes registrados en la base de datos. Este componente gestiona tanto el envío y recepción de mensajes como la recuperación y visualización de la información de los postulantes almacenados en Firebase.

Para la selección de postulantes, el componente cuenta con un modal, el cual se abre al ejecutar el método **abrirModalPostulantes()**. Este método obtiene la lista de currículums almacenados en Firestore mediante el servicio FirebaseService.

La selección de postulantes se gestiona a través de la función **toggleSeleccionPostulante()**, que permite agregar o eliminar postulantes de la lista de seleccionados. Al confirmar la selección con **confirmarSeleccion()**, se genera un mensaje con todos los datos del postulante, organizados de manera estructurada.

Además, el componente incorpora la funcionalidad de chat con IA. Cuando el usuario ingresa un mensaje y ejecuta **enviarMensaje()**, el contenido se envía al servicio **DeepSeekService**, el cual procesa la consulta y devuelve una respuesta. Durante este proceso, el mensaje es agregado a la lista de conversaciones con el rol de usuario y, una vez recibida la respuesta, se almacena como un mensaje del asistente.

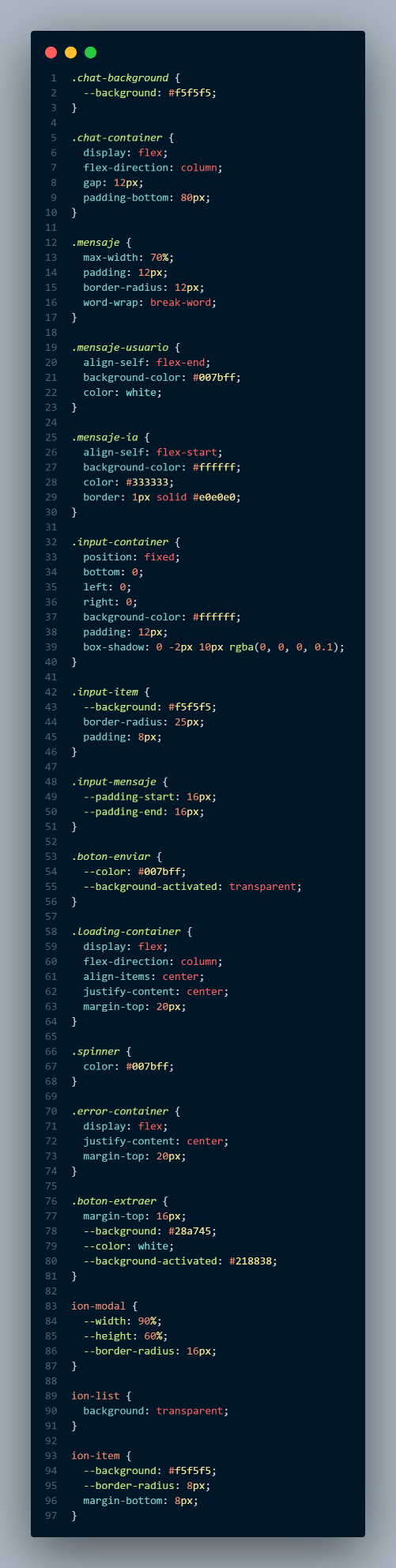
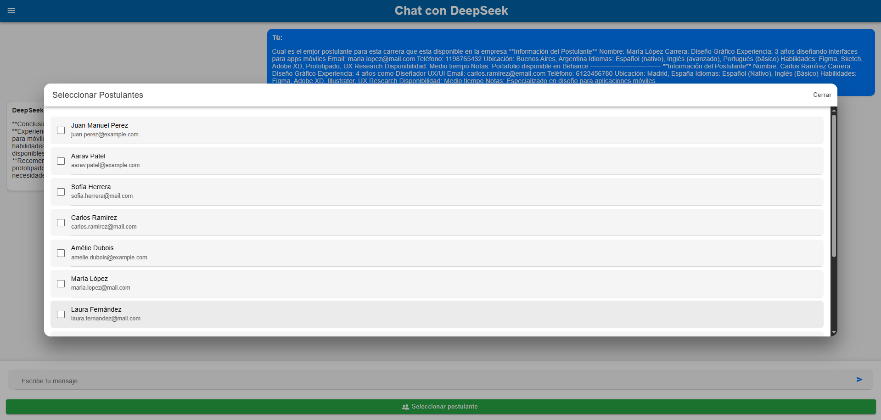
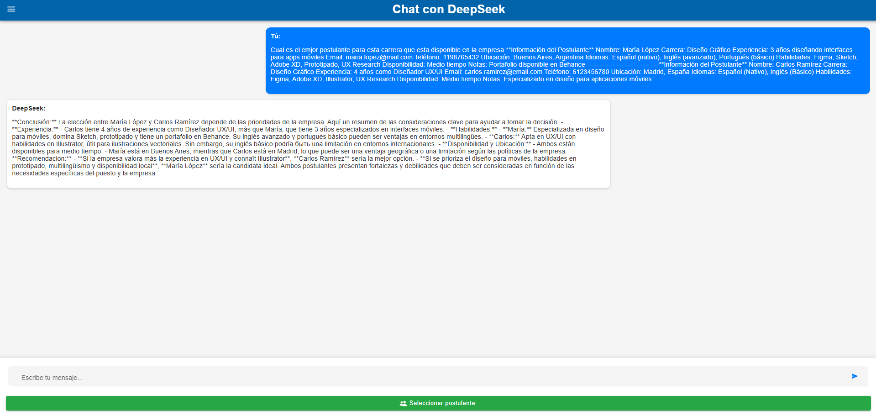
El componente también gestiona el estado de carga y los posibles errores, garantizando una experiencia fluida e intuitiva para el usuario.

Por otro lado, el servicio **DeepSeekService** es el encargado de gestionar las solicitudes a la API de **OpenRouter**, facilitando la interacción con el modelo de inteligencia artificial **DeepSeek**.

Dentro de este servicio, se maneja la configuración de la API, lo que incluye la gestión de la ruta del endpoint y el envío de consultas a la IA. Para ello, se construye una solicitud con los parámetros adecuados, como el modelo de IA utilizado y los mensajes enviados por el usuario.

Además, el servicio administra la autenticación mediante la API Key, asegurando que las solicitudes sean válidas y cumplan con los requisitos de seguridad. Se recomienda manejar esta clave de forma segura, evitando su exposición en el código fuente y almacenándola en variables de entorno o en un backend para protegerla de accesos no autorizados.





En este caso, también se cuenta con servicios fundamentales para la gestión de las conexiones, especialmente con la base de datos Firebase. Uno de los principales es el servicio **FirebaseService**, el cual se encarga de administrar todas las operaciones relacionadas con la base de datos, como la recuperación de información, la actualización de documentos y la eliminación de colecciones o registros.

Además de **FirebaseService**, se integran otros servicios previamente mencionados, como **UtilsService**, **AuthService** y **DeepSeekService**, cada uno con funciones específicas dentro de la aplicación.

Por otro lado, se dispone de los archivos de configuración **environments**, donde se definen dos entornos: uno para desarrollo local y otro para producción. En estos archivos se encuentran configuraciones clave, incluyendo la llave de acceso y conexión con Firebase, garantizando que la aplicación funcione correctamente en cada entorno sin comprometer la seguridad de la información.



El componente **HeaderComponent** actúa como un encabezado reutilizable dentro de la aplicación, proporcionando un título configurable, opciones de navegación y control de menú. Su comportamiento es personalizable a través de diversas propiedades de entrada y emite eventos para gestionar acciones específicas.

Su funcionalidad principal se centra en la gestión del título y la navegación mediante un botón de retroceso. La propiedad title define el texto del encabezado, mientras que **showBackButton** controla la visibilidad del botón de retorno. Cuando un usuario interactúa con este botón, el componente emite un evento a través de **backAction**, permitiendo que el componente padre ejecute la acción correspondiente.

En relación con la interacción del menú, la propiedad **isInMenu** indica si el componente forma parte de un menú, mientras que la variable **isMenuOpen** gestiona su estado. Para modificar la visibilidad del menú, se implementa la función **toggleMenu()**.

Además, el componente es compatible con modales mediante la propiedad **isInModal**. En estos casos, la función **closeModal()** permite cerrar el modal al emitir el evento **backAction**.

Por último, la interfaz gráfica del componente presenta un encabezado en el que se muestra el título y, de manera opcional, un botón de retroceso. También se ofrece la posibilidad de integrar controles adicionales en función de las necesidades de cada vista donde se utilice.



