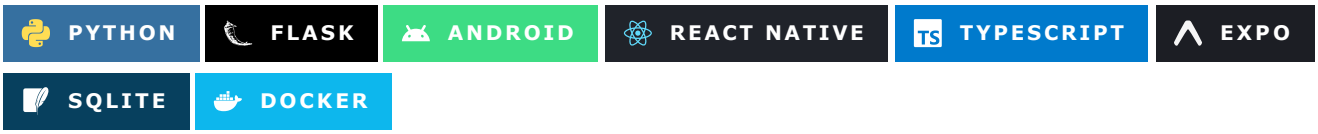
AI Expert System for Vocational Guidance



Introduccion

En un mundo en constante evolución, donde las opciones profesionales son vastas y variadas, la orientación vocacional se convierte en un componente crucial para los que buscan definir su trayectoria profesional. En este contexto, las aplicaciones móviles han emergido como herramientas poderosas para brindar asesoramiento personalizado y accesible. Este informe se centra en la aplicación móvil "Expertus: AI Expert System for Vocational Guidance", una innovadora solución que aprovecha la inteligencia artificial, bajo el enfoque de sistema experto, para ofrecer recomendaciones y análisis detallados sobre carreras y profesiones. A a lo largo del presente informe, exploraremos sus características, funcionalidades, proceso de desarrollo y configuraciones.

Descripcion

Expertus es una aplicacion movil diseñada para ayudar a los usuarios a explorar y entender las diversas opciones profesionales disponibles.

1. Requisitos No Funcionales

Requisito No Funcional	Descripción
Rendimiento	El sistema debe ser capaz de manejar múltiples usuarios simultáneamente sin degradar el rendimiento. Las respuestas a las acciones del usuario (como enviar el cuestionario) deben ser rápidas, con tiempos de respuesta inferiores a 2 segundos.
Escalabilidad	El sistema debe ser escalable para soportar un número creciente de usuarios y datos sin pérdida de rendimiento. Debe ser fácil de actualizar y mantener.
Usabilidad	La interfaz de usuario debe ser intuitiva y accesible para personas con diferentes niveles de habilidad tecnológica. Debe seguir principios de diseño accesible para usuarios con discapacidades.
Mantenibilidad	El código debe estar bien documentado y seguir buenas prácticas de programación para facilitar su mantenimiento y evolución. Debe haber pruebas automatizadas para asegurar la calidad y funcionamiento correcto del sistema.

2. Requisitos Funcionales

Requisito Funcional	Descripción

Requisito Funcional	Descripción
Cuestionario de Orientación Vocacional	El sistema debe presentar un cuestionario con preguntas sobre intereses, habilidades y preferencias de los usuarios. Las preguntas deben ser faciles de entender y ser respondidas con un SI o con un NO.
Procesamiento de Respuestas	El sistema debe analizar las respuestas del cuestionario utilizando un algoritmo experto para generar recomendaciones.Debe haber lógica para manejar respuestas incompletas y asegurar que se recopile suficiente información para generar una recomendación precisa.
Generación de Resultados	El sistema debe generar un informe con las recomendaciones de carreras y campos profesionales basados en las respuestas del usuario. El informe debe ser claro y comprensible, proporcionando detalles sobre por qué se recomendaron ciertas carreras.
Historial de Resultados	Los usuarios deben poder acceder a un historial de sus cuestionarios y resultados anteriores. El sistema debe permitir la comparación de resultados de diferentes cuestionarios realizados por el mismo usuario.
Registro y Autenticación de Usuarios	El sistema debe permitir a los usuarios registrarse con un correo electrónico y una contraseña. Los usuarios deben poder iniciar sesión con sus credenciales.

3. Estructura de Proyecto

El sistema consta de dos partes principales:

1. **Expertus**: Una aplicación móvil intuitiva y amigable que permite a los usuarios interactuar con el sistema.
2. **Backend**: Un servidor en Python que maneja la lógica del sistema experto y procesa las respuestas de los usuarios.

```
.
├── backend
│   ├── app.py
│   ├── carreras.db
│   ├── conocimiento.py
│   ├── db
│   │   ├── create_tables.py
│   │   └── extract_db.py
│   ├── Dockerfile
│   ├── requirements.txt
│   └── test
│       ├── expert.py
│       └── lecturaCarrerasDB.py
│
├── docs
│   └── ...
├── expertus
```

```
│   ├── app
│   │   ├── +html.tsx
│   │   ├── _layout.tsx
│   │   ├── login.tsx
│   │   ├── +not-found.tsx
│   │   ├── register.tsx
│   │   └── (tabs)
│   │       ├── _layout.tsx
│   │       ├── chatbox.tsx
│   │       ├── explore.tsx
│   │       ├── index.tsx
│   │       └── settings.tsx
│   ├── app.json
│   ├── assets
│   │   ├── fonts
│   │   └── images
│   ├── babel.config.js
│   ├── components
│   │   ├── CareerCard.tsx
│   │   ├── Collapsible.tsx
│   │   ├── ExternalLink.tsx
│   │   ├── HelloWave.tsx
│   │   ├── navigation
│   │   ├── ParallaxScrollView.tsx
│   │   ├── __tests__
│   │   ├── ThemedText.tsx
│   │   ├── ThemedView.tsx
│   │   └── UniversityCard.tsx
│   ├── constants
│   │   └── Colors.ts
│   ├── expo-env.d.ts
│   ├── hooks
│   │   ├── useColorScheme.ts
│   │   ├── useColorScheme.web.ts
│   │   └── useThemeColor.ts
│   ├── package.json
│   ├── package-lock.json
│   ├── README.md
│   ├── scripts
│   │   └── reset-project.js
│   └── tsconfig.json
├── COMMITS.md
└── README.md
```

4. Tecnologias Usadas

- ***Frontend***
    - `React Native` Para desarrollar la aplicación móvil.
    - `Expo` Framework para desarrollar aplicaciones de React Native de manera sencilla.
    - `Typescript` Lenguaje de programación para agregar tipado para variables y escribir código mantenible.

- ***Backend***

    - `Python` Lenguaje de programación principal para el backend.
    - `Flask` Microframework para desarrollar aplicaciones de servidor.
    - `SQLite` Base de datos ligera para almacenar información de usuarios y resultados.
    - `Docker` Herramienta para crear, desplegar y ejecutar la aplicacion en un contenedor, asegurando su consistencia en los entornos de desarrollo y producción.

5. Funcionalidades

- **Interfaz Amigable para el Usuario**: Diseño optimizado para dispositivos móviles que garantiza una experiencia intuitiva.
- **Cuestionario de Orientación Vocacional**: Presenta preguntas sobre intereses, habilidades y preferencias, fáciles de entender y responder.
- **Motor de Recomendaciones**: Algoritmo experto que sugiere carreras y campos profesionales basados en las respuestas de los usuarios.
- **Procesamiento de Respuestas**: Analiza las respuestas del cuestionario y maneja respuestas incompletas para asegurar recomendaciones precisas.
- **Generación de Resultados**: Genera informes claros y comprensibles con recomendaciones de carreras, proporcionando detalles sobre cada sugerencia.
- **Historial de Resultados**: Permite a los usuarios acceder y comparar sus cuestionarios y resultados anteriores.
- **Registro y Autenticación de Usuarios**: Permite a los usuarios registrarse e iniciar sesión con su correo electrónico y contraseña.
- **Rendimiento Óptimo**: Capacidad para manejar múltiples usuarios simultáneamente con tiempos de respuesta inferiores a 2 segundos.
- **Escalabilidad**: Soporta un número creciente de usuarios y datos sin pérdida de rendimiento.
- **Mantenibilidad**: Código bien documentado y pruebas automatizadas para asegurar la calidad y funcionamiento correcto del sistema.
- **Accesibilidad**: Interfaz accesible para personas con diferentes niveles de habilidad tecnológica y usuarios con discapacidades.

Integración de la IA

1. **Adquisición de Conocimiento**

    - **Expertos**: Personas con conocimientos específicos en un área determinada que proporcionan información valiosa al sistema.
    - **Módulo de Adquisición de Conocimiento**: Se encarga de recopilar, formalizar y almacenar el conocimiento obtenido de los expertos y otras fuentes.

2. **Representación del Conocimiento**

    - **Base de Conocimiento**: Almacena el conocimiento formalizado, que incluye reglas, hechos, heurísticas y otros tipos de información estructurada.

```
# Importar la función extraer_datos del archivo extract_db.py
from extract_db import extraer_datos
```

```
#Función importada de extract_db.py
def extraer_datos():
    conn = sqlite3.connect('carreras.db')
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
    tablas = cursor.fetchall()
    carreras_preguntas = {}
    for tabla in tablas:
        nombre_tabla = tabla[0].replace("_", " ")
        cursor.execute(f"SELECT categoria, pregunta FROM {tabla[0]}")
        filas = cursor.fetchall()
        carreras_preguntas[nombre_tabla] = [(fila[0], fila[1]) for fila in filas]
    conn.close()
    return carreras_preguntas

# En app.py: obtenemos las preguntas asociadas a cada carrera y categoría
conocimiento = extraer_datos()
```

Importamos la función extraer_datos del archivo extract_db.py, que nos permite acceder a la base de datos carreras.db y extraer las preguntas asociadas a cada carrera y categoría. Luego, almacenamos esta información en el diccionario conocimiento, que servirá como nuestra Base de Conocimiento en el sistema experto.

- **Base de Hechos**: Contiene datos específicos de casos particulares o instancias de problemas que el sistema necesita resolver. Estos datos son dinámicos y específicos a cada consulta o caso.

```
# Base de Hechos en app.py inicializa vacia
conocido = []

# Parte de código que sería parte de la actualización dinámica:
def procesar_respuesta(data):
    global current_symptom, diagnostico_actual
    if 'respuesta' not in data:
        return jsonify({'error': 'Respuesta no proporcionada'}), 400
    respuesta = data['respuesta'].strip().lower()
    if respuesta not in ['si', 'no']:
        return jsonify({'error': 'Respuesta inválida'}), 400

    if respuesta == 'si':
        conocido.append(current_symptom)
    elif respuesta == 'no':
        conocido.append('no ' + current_symptom)
```

En app.py, creamos una lista llamada conocido, que se utilizará para almacenar los datos específicos de casos particulares o instancias de problemas que el sistema necesita resolver. Esta lista se inicializa vacía y se actualizará dinámicamente a medida que el usuario proporciona respuestas a las preguntas

del sistema experto. Cada elemento en esta lista representará un hecho o conocimiento específico que el sistema utilizará para analizar y llegar a conclusiones o recomendaciones.

3. **Tratamiento del Conocimiento**

    - **Motor de Inferencia**: Es el componente que aplica las reglas y el conocimiento almacenado en la base de conocimiento para analizar los hechos y llegar a conclusiones o recomendaciones.

```
def haz_diagnostico():
    global explicacion_diagnostico
    explicacion_diagnostico = []
    for diagnosis, sintomas in conocimiento.items():
        if prueba_presencia_de(sintomas):
            explicacion_diagnostico = [sintoma[1] for sintoma in sintomas if
prueba_verdad_de(sintoma[1])]
            return diagnosis
    return None
```

haz_diagnostico() recorre las posibles enfermedades (diagnosis) y sus síntomas (sintomas) almacenados en conocimiento. Utiliza la función prueba_presencia_de para verificar si todos los síntomas de una enfermedad están presentes en la lista de síntomas conocidos (conocido). Si encuentra una coincidencia, genera una lista de síntomas verificados (explicacion_diagnostico) y retorna la enfermedad diagnosticada.

- **Módulo de Explicaciones**: Proporciona justificaciones y explicaciones sobre cómo el sistema ha llegado a ciertas conclusiones. Esto es crucial para la transparencia y la confianza en el sistema.

```
def obtener_pregunta():
    global current_symptom, diagnostico_actual
    if diagnostico_actual:
        return jsonify({'diagnostico': diagnostico_actual, 'explicacion':
explicacion_diagnostico})
    if current_symptom is None:
        current_symptom = siguiente_sintoma()
    if current_symptom is not None:
        return jsonify({'pregunta': f'Es verdad que {current_symptom}?'})
    else:
        diagnostico_actual = haz_diagnostico()
        if diagnostico_actual:
            return jsonify({'diagnostico': diagnostico_actual, 'explicacion':
explicacion_diagnostico})
        return jsonify({'diagnostico': 'No hay suficiente conocimiento para
elaborar un diagnostico.'})
```

En obtener_pregunta(), cuando se ha realizado un diagnóstico (diagnostico_actual no es None), se retorna una respuesta JSON que incluye tanto el diagnóstico como la explicación (explicacion_diagnostico). Esta explicación contiene los síntomas que fueron considerados verdaderos y

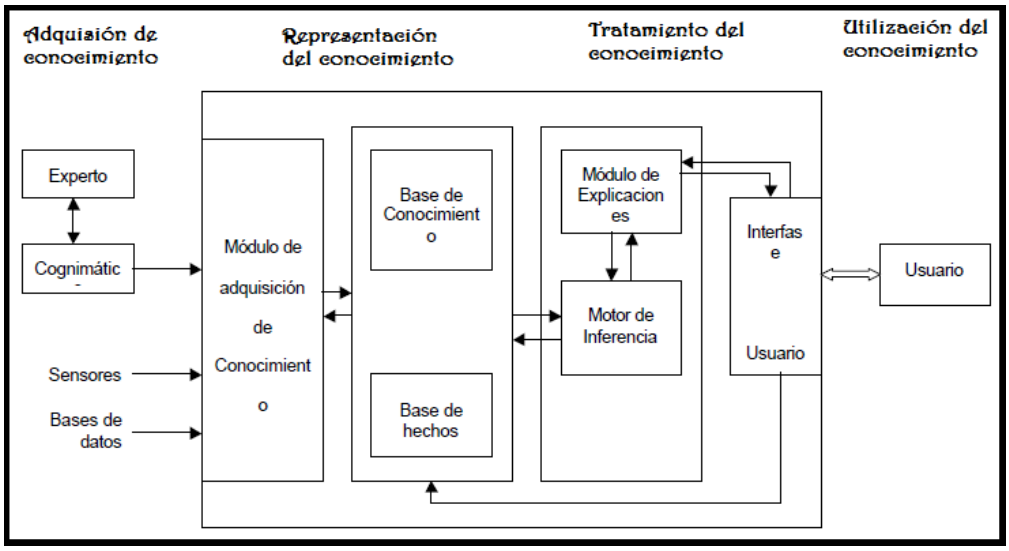
> que llevaron al diagnóstico final, proporcionando así transparencia sobre el proceso de inferencia del sistema.

- **Interacción entre Componentes**: El motor de inferencia y el módulo de explicaciones interactúan con la base de conocimiento y la base de hechos para procesar la información y generar resultados explicativos.

```
def procesar_respuesta(data):
    global current_symptom, diagnostico_actual
    if 'respuesta' not in data:
        return jsonify({'error': 'Respuesta no proporcionada'}), 400
    respuesta = data['respuesta'].strip().lower()
    if respuesta not in ['si', 'no']:
        return jsonify({'error': 'Respuesta inválida'}), 400

    if respuesta == 'si':
        conocido.append(current_symptom)
    elif respuesta == 'no':
        conocido.append('no ' + current_symptom)

    diagnostico_actual = haz_diagnostico()
    if diagnostico_actual:
        return jsonify({'diagnostico': diagnostico_actual, 'explicacion':
explicacion_diagnostico})

    current_symptom = siguiente_sintoma()
    if current_symptom is None:
        diagnostico_actual = haz_diagnostico()
        if diagnostico_actual:
            return jsonify({'diagnostico': diagnostico_actual, 'explicacion':
explicacion_diagnostico})
        return jsonify({'diagnostico': 'No hay suficiente conocimiento para
elaborar un diagnostico.'})
    return jsonify({'pregunta': f'Es verdad que {current_symptom}?'})
```

> En procesar_respuesta(), la respuesta del usuario a la pregunta actual se procesa y se actualiza la lista de síntomas conocidos (conocido). Luego, se llama a la función haz_diagnostico para intentar realizar un diagnóstico basado en la información actual. Si se realiza un diagnóstico, se retorna un JSON con el diagnóstico y la explicación. Si no se realiza un diagnóstico, se determina el siguiente síntoma a preguntar y se formula la siguiente pregunta al usuario. Así, la función procesar_respuesta muestra cómo el motor de inferencia, el módulo de explicaciones y la base de hechos interactúan dinámicamente para llevar a cabo el proceso de diagnóstico y explicación.

4. **Utilización del Conocimiento**
    - **Interfaz de Usuario**: El punto de contacto entre el usuario y el sistema experto. A través de esta interfaz, el usuario puede introducir datos, hacer consultas y recibir recomendaciones o conclusiones del sistema.
    - **Usuario**: La persona que utiliza el sistema experto para obtener información, recomendaciones o soluciones a problemas específicos.



5. ***Tipo de Encadenamiento***

   El código utiliza encadenamiento hacia adelante (forward chaining) por las siguientes razones:

   - **Recopilación y Aplicación de Hechos:** El sistema comienza con una base de hechos conocida (síntomas proporcionados por el usuario). Aplica reglas (relaciones entre síntomas y diagnósticos) para derivar nuevos hechos (diagnósticos posibles).
   - **Progresión Basada en Hechos:** A medida que el usuario proporciona respuestas, se añaden nuevos hechos a la lista de conocido. Las reglas se evalúan continuamente para ver si los hechos actuales pueden derivar un diagnóstico.
   - **Evaluación Continua:** Se genera una nueva pregunta basada en los síntomas que aún no han sido evaluados. Este proceso continúa hasta que se puede derivar un diagnóstico a partir de los hechos conocidos.

Metodologia

Scrum es una metodología ágil que facilita el desarrollo incremental de proyectos, promoviendo la adaptabilidad y la colaboración. La metodología Scrum se basa en iteraciones cortas y definidas, llamadas Sprints, que permiten la entrega continua de funcionalidad y la incorporación de feedback constante.

- **Roles en Scrum**:
  - **Product Owner**: Define y prioriza requisitos, gestiona el backlog.
  - **Scrum Master**: Facilita el proceso, elimina obstáculos, fomenta la colaboración.
  - **Development Team**: Implementa los requisitos, entrega incrementos funcionales.
- **Artefactos en Scrum**:
  - **Product Backlog**: Lista priorizada de funcionalidades, gestionada por el Product Owner.
  - **Sprint Backlog**: Elementos del Product Backlog seleccionados para el Sprint.
  - **Release**: Producto funcional y potencialmente entregable al final del Sprint.
- **Eventos en Scrum**:
  - **Sprint Planning**: Selección y planificación de elementos para el Sprint.

    - **Daily Scrum**: Reunión diaria de 15 minutos para revisar el progreso.
    - **Sprint Review**: Presentación del incremento completado y recopilación de feedback.
    - **Sprint Retrospective**: Reflexión sobre el proceso y búsqueda de mejoras.

- **Ciclo de Desarrollo con Scrum**:

    - **Inicio del Proyecto**: Definición del Product Backlog, identificación de roles.
    - **Planificación de Sprints**: Selección de elementos del Product Backlog para el Sprint.
    - **Desarrollo Incremental**: Trabajo en los elementos seleccionados, reuniones diarias.
    - **Revisión y Mejora Continua**: Revisión del Sprint, retrospectiva, ajuste del Product Backlog.

- **Aplicación de Scrum en Expertus**:

    - Gestión eficiente del proyecto.
    - Entrega de un producto de alta calidad.
    - Adaptación rápida a cambios en el entorno y los requisitos.

Guía de Usuario

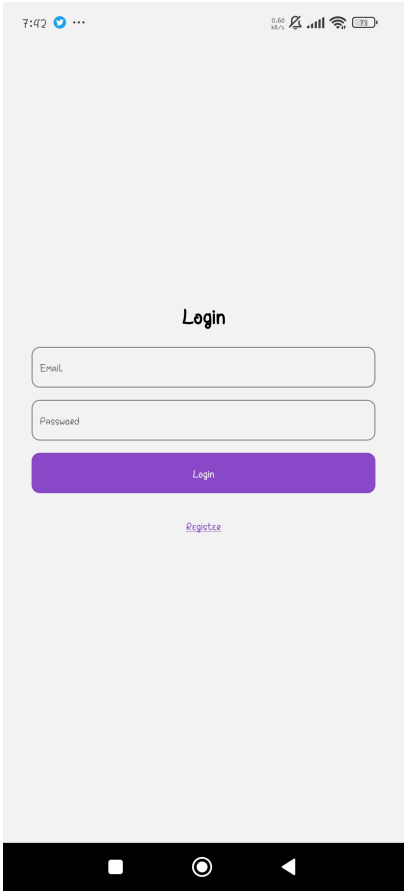
Pantallas y Funcionalidades

Pantalla de Inicio de Sesión

- **Funcionalidad**: Permite al usuario ingresar al sistema utilizando su correo electrónico y contraseña.

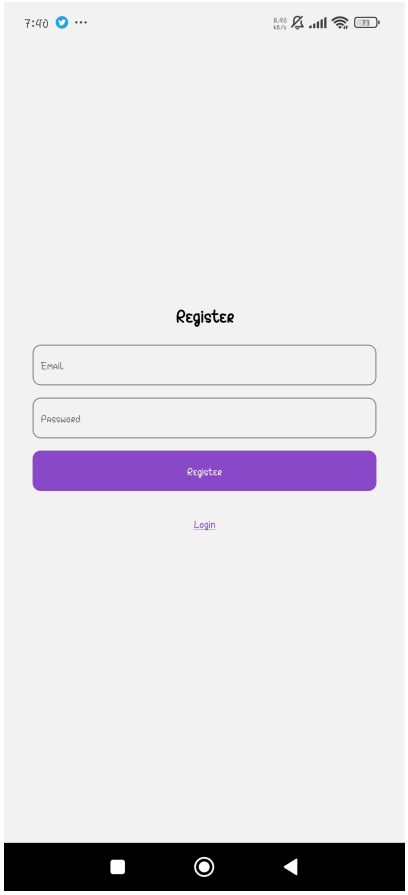
- **Acciones**:

    - Ingresar el correo electrónico y la contraseña.
    - Presionar el botón "Iniciar Sesión".



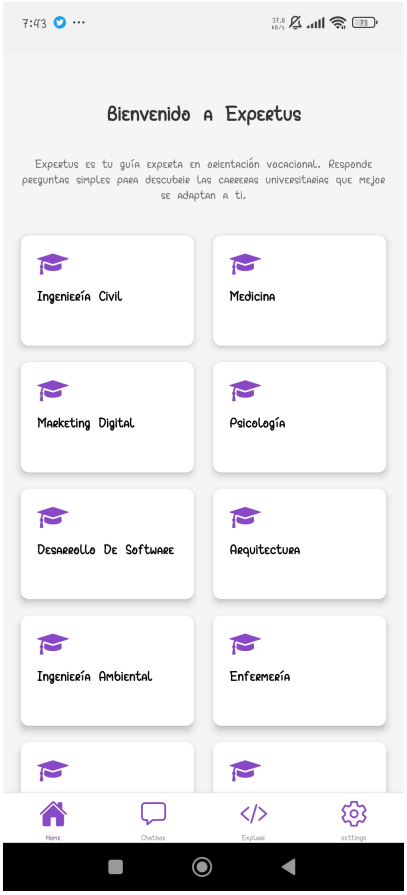
Pantalla de Registro

- **Funcionalidad**: Permite a nuevos usuarios crear una cuenta en el sistema.
- **Acciones**:
    - Ingresar correo electrónico y contraseña.
    - Presionar el botón "Registrar".



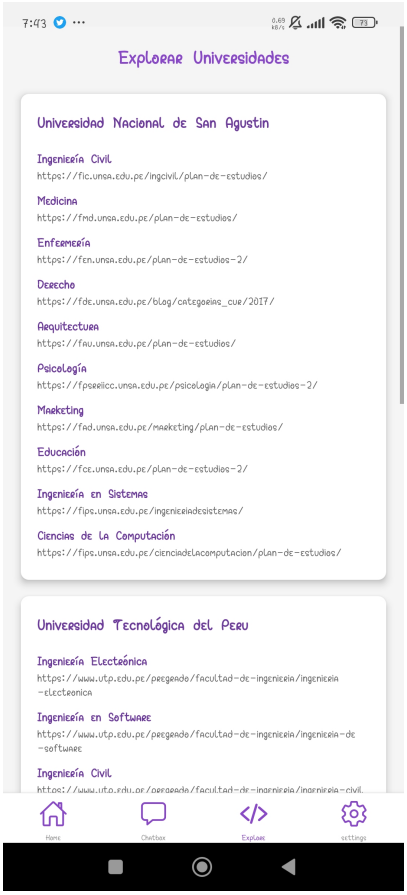
Pantalla de Inicio

- **Funcionalidad**: Muestra un mensaje de bienvenida y los nombres de las carreras disponibles en el sistema.
- **Acciones**:
    - Ver el mensaje de bienvenida.
    - Navegar por la lista de carreras disponibles.



Pantalla de Exploración

- **Funcionalidad**: Muestra las carreras analizadas en universidades del Perú.
- **Acciones**:
  - Explorar y navegar por las carreras ofrecidas por distintas universidades del Perú.
  - Seleccionar una carrera para ver más detalles.

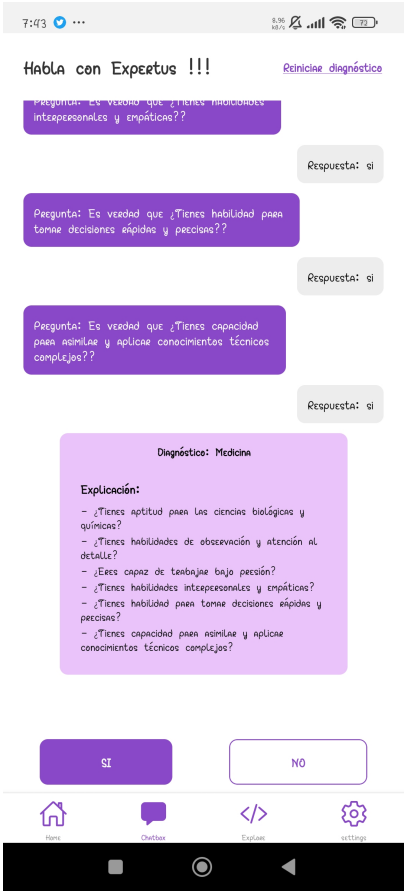


Pantalla de Chatbox

- **Funcionalidad**: Interactuar con el sistema experto para recibir recomendaciones de carreras.
- **Acciones**:
    - Responder preguntas utilizando los botones "Sí" o "No".
    - Presionar el botón "Reiniciar Diagnóstico" para comenzar el cuestionario desde el inicio.

Pantalla de Diagnóstico

- **Funcionalidad**: Muestra el diagnóstico generado a partir de las respuestas del usuario en la pantalla de Chatbox.
- **Acciones**:
    - Ver la carrera recomendada.
    - Leer los motivos y razones detrás de la recomendación.



Flujo de Uso del Usuario

1. **Registro e Inicio de Sesión**:
    - El usuario se registra o inicia sesión en el sistema a través de las pantallas de "Register" o "Login".
2. **Exploración Inicial**:
    - Al ingresar al sistema, el usuario es llevado a la pantalla "Home" donde puede ver las carreras disponibles.
    - Opcionalmente, el usuario puede ir a la pantalla "Explore" para ver las carreras ofrecidas por universidades del Perú.
3. **Interacción con el Sistema Experto**:
    - El usuario navega a la pantalla "Chatbox" para comenzar el cuestionario de orientación vocacional.
    - Responde las preguntas utilizando los botones "Sí" o "No".
4. **Recepción de Diagnóstico**:
    - Después de completar el cuestionario, el usuario recibe un diagnóstico en la pantalla de "Ejemplo de Diagnóstico".
    - El diagnóstico incluye la carrera recomendada y las razones detrás de dicha recomendación.
5. **Reinicio del Proceso**:
    - Si el usuario desea realizar el cuestionario nuevamente, puede utilizar el botón "Reiniciar Diagnóstico" en la pantalla de "Chatbox".

Instalacion y Configuracion

1. **Clone the repository**
2. **Database:**
    - Instalar SQLite;

```
sudo apt install sqlite3
```

3. **Backend:**
    - From the root of the project:

```
cd expert-system-vocational-guidance/backend
```

    - Create a virtual environment:

```
python -m venv venv
source venv/bin/activate
```

    - Install the dependencies:

```
pip install -r requirements.txt
```

    - Run the backend application:

```
flask --app app run
```

4. **Frontend:**
    - From the root of the project:

```
cd expert-system-vocational-guidance/expertus
```

    - Install the dependencies:

```
npm install
```

- Run the expertus application:

```
npx expo start
```

Desafíos y Limitaciones

1. **Gestión de Datos**:
    - **Desafío**: Manejar una cantidad creciente de datos de usuarios y sus respuestas al cuestionario puede ser complejo.
    - **Limitación**: El uso de SQLite como base de datos puede no ser suficiente a largo plazo para manejar grandes volúmenes de datos y concurrencia.
2. **Precisión del Sistema Experto**:
    - **Desafío**: Asegurar que el algoritmo de recomendación del sistema experto sea preciso y relevante para cada usuario.
    - **Limitación**: La precisión del sistema depende de la calidad y amplitud del conocimiento almacenado, lo que puede requerir actualizaciones y mejoras continuas.
3. **Escalabilidad**:
    - **Desafío**: Adaptar el sistema para soportar un número creciente de usuarios sin comprometer el rendimiento.
    - **Limitación**: La arquitectura actual puede necesitar reestructuración para soportar una mayor escalabilidad, especialmente en el backend.
4. **Mantenimiento y Actualización**:
    - **Desafío**: Asegurar que el código esté bien documentado y sea fácil de mantener y actualizar.
    - **Limitación**: La necesidad de pruebas automatizadas y una buena documentación es crucial para la evolución continua del sistema.

Conclusiones

1. **Implementación Exitosa de Metodología Ágil**:
    - La adopción de la metodología Scrum ha permitido un desarrollo eficiente y colaborativo del sistema, facilitando la entrega continua de valor y la mejora constante.
2. **Sistemas Experto Eficaz**:
    - El sistema experto implementado en la aplicación proporciona recomendaciones personalizadas basadas en las respuestas de los usuarios, añadiendo valor significativo a la experiencia del usuario.
3. **Interfaz Intuitiva y Accesible**:
    - La aplicación presenta una interfaz amigable e intuitiva que facilita la navegación y el uso del sistema, independientemente del nivel de habilidad tecnológica del usuario.

4. **Necesidad de Mejoras en la Escalabilidad**:
    - Aunque el sistema actual cumple con los requisitos, es crucial planificar mejoras en la escalabilidad y el manejo de datos para soportar el crecimiento futuro y garantizar un rendimiento óptimo.
5. **Importancia del Mantenimiento Continuo**:
    - Para asegurar la longevidad y efectividad del sistema, es esencial mantener una buena documentación, implementar pruebas automatizadas y realizar actualizaciones periódicas basadas en feedback y nuevos desarrollos en el campo de la orientación vocacional.