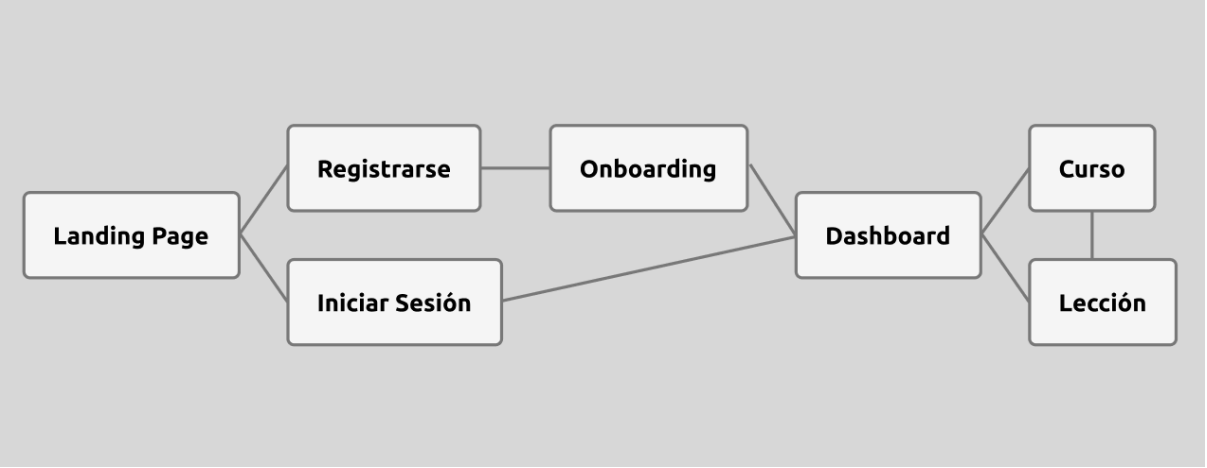
Taller 3: testing

1. Introducción

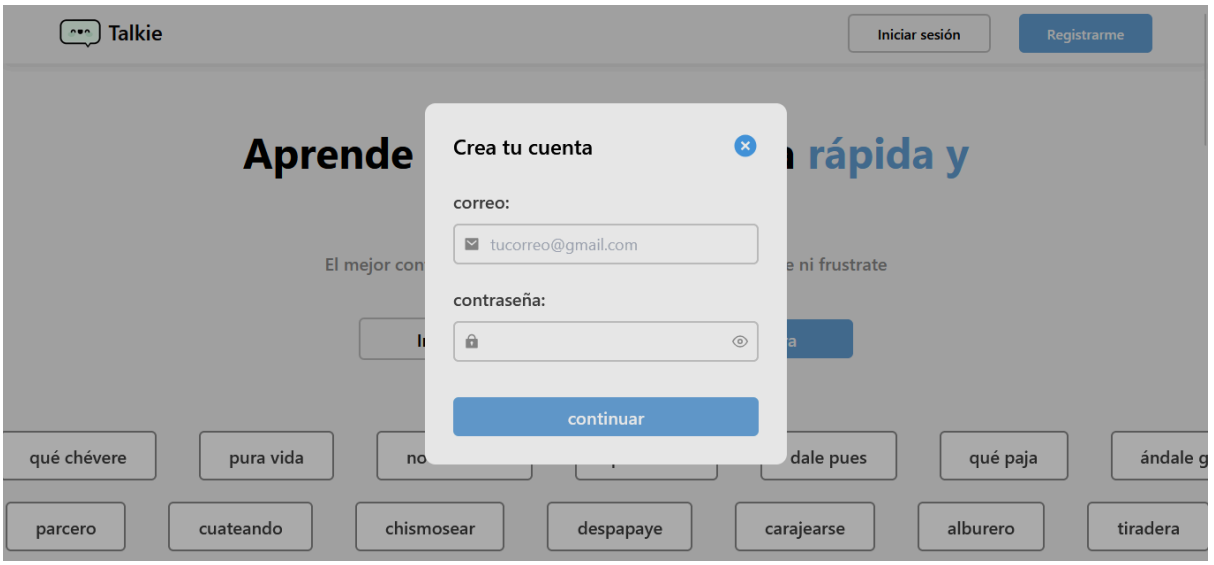
Mientras discutimos posibles ideas, uno de los integrantes del grupo comentó que estaba trabajando en una inteligencia artificial capaz de aprender idiomas y traducir palabras o frases. Partiendo de esta base, decidimos desarrollar una aplicación enfocada en el aprendizaje de idiomas. Al analizar las opciones existentes, notamos que la mayoría de las plataformas enseñan los idiomas de forma “neutral”, sin profundizar en los dialectos o variantes coloquiales. Identificamos en esto una oportunidad de innovación y decidimos crear Talkie.

Talkie es una aplicación web diseñada para facilitar el aprendizaje de dialectos de manera sencilla y rápida. Su principal diferencial es que ofrece lecciones personalizadas en distintos formatos, permitiendo a los usuarios sumergirse en el lenguaje real utilizado en diversos contextos informales. A través del uso de inteligencia artificial, Talkie no sólo traduce, sino que adapta expresiones y modismos a cada dialecto, brindando una experiencia de aprendizaje auténtica y cercana a la realidad de los hablantes nativos.



Mockups:


Talkie
Iniciar sesión
Registrarme
Aprende dialectos de forma rápida y divertida
El mejor contenido para que puedas aprender sin aburrirte ni frustrate
Iniciar sesión
Empezar ahora
qué chévere
pura vida
no manches
qué bacán
dale pues
qué paja
parcero
cuateando
chismosear
despapaye
carajearse
alburero


Talkie
Iniciar sesión
Registrarme
Crea tu cuenta
correo:
tucorreo@gmail.com
contraseña:
continuar
qué chévere
pura vida
dale pues
qué paja
parcero
cuateando
chismosear
despapaye
carajearse
alburero
tiradera

2. Resumen de los tests:

Integrante	**Tipo de prueba**	**Descripción**	**Herramienta**
Juan Camilo Rosero	De integración	La función **test_json_response** verifica que el endpoint /menu devuelva una respuesta JSON válida con código 200.	pytest
Juan Camilo Rosero	Unitaria	La función **test_normalize_text** prueba que normalize_text convierta el texto a minúsculas, elimine acentos y caracteres no deseados	pytest
Nicolás Gómez	De integración	La función **test_post_action** comprueba que una solicitud POST a /action procese correctamente los datos enviados.	pytest
Luis Gabriel Peraza	De integración	La función **test_post_action_missing_data** evalúa que el endpoint /action devuelva un error 400 cuando faltan datos en la solicitud POST.	pytest
Marlon David Pabón	Unitaria	La función **test_verify_word** verifica que verify_word compare correctamente palabras normalizadas para evaluar su validez.	pytest

Screenshots:

test_json_response:

```python
def test_json_response(client):
    response = client.get('/menu')
    assert response.status_code == 200
    assert response.content_type == 'application/json'
```

```
============================= test session starts =============================
platform win32 -- Python 3.11.2, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Juan Camilo\Downloads\Talkie-main\Talkie-main\IA
collected 1 item

test_app.py .                                                            [100%]

============================== 1 passed in 64.42s (0:01:04) ===============================
PS C:\Users\Juan Camilo\Downloads\Talkie-main\Talkie-main\IA>
```

test_normalize_text

```python
def test_normalize_text():
    assert normalize_text("Hóla, Múndó!") == "hola mundo"
    assert normalize_text("TESTING") == "testing"
    assert normalize_text("ÁÉÍÓÚ áéíóú") == "aeiou aeiou"
    assert normalize_text("Python!123") == "python123"
```

```
============================= test session starts =============================
platform win32 -- Python 3.11.2, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Juan Camilo\Downloads\Talkie-main\Talkie-main\IA
collected 1 item

test_app.py .                                                            [100%]

============================== 1 passed in 62.35s (0:01:02) ===============================
```

test_post_action

```python
def test_post_action(client):
    data = {
        'user_input': 'Hola',
        'current_language': 'spanish',
        'nativeLanguage': 'english'
    }
    response = client.post('/action', data=json.dumps(data), content_type='application/json')
    assert response.status_code == 200
    assert isinstance(response.json, dict)
```

```
============================= test session starts =============================
platform win32 -- Python 3.11.2, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Juan Camilo\Downloads\Talkie-main\Talkie-main\IA
collected 1 item

test_app.py .                                                            [100%]

============================== 1 passed in 70.84s (0:01:10) ===============================
```

test_post_action_missing_data

```python
def test_post_action_missing_data(client):
    data = {'user_input': 'Hola'}
    response = client.post('/action', data=json.dumps(data), content_type='application/json')
    assert response.status_code == 400
    assert 'error' in response.json
```

```
============================= test session starts =============================
platform win32 -- Python 3.11.2, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Juan Camilo\Downloads\Talkie-main\Talkie-main\IA
collected 1 item

test_app.py .                                                            [100%]

============================== 1 passed in 78.05s (0:01:18) ===============================
```

test_verify_word

```python
def test_verify_word():
    correct_word = "hello"
    assert verify_word("hello", correct_word) == True
    assert verify_word("Hello", correct_word) == True
    assert verify_word("h3llo", correct_word) == False
    assert verify_word("hola", correct_word) == False
```

```
============================= test session starts =============================
platform win32 -- Python 3.11.2, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Juan Camilo\Downloads\Talkie-main\Talkie-main\IA
collected 1 item

test_app.py .                                                            [100%]

============================== 1 passed in 76.95s (0:01:16) ===============================
```

Reflexión grupal

Uno de los mayores desafíos que enfrentamos fue la ejecución de **test_normalize_text**, ya que inicialmente no habíamos considerado varios de los posibles errores antes de realizar las pruebas. Esto nos llevó a replantear nuestra manera de anticipar y manejar excepciones, reforzando la idea de que el testing no es solo una verificación final, sino una parte integral del desarrollo. Al final la función cambió bastante de su forma inicial (le agregamos varias validaciones con expresiones regulares)

También encontramos dificultades al intentar implementar pruebas en el frontend con Next.js. Si bien exploramos diferentes enfoques, finalmente decidimos centralizar los tests en el backend, lo que nos permitió mantener un flujo más claro y controlado en la validación de nuestra lógica.

En última instancia, este proceso nos confirmó la importancia del testing para garantizar la estabilidad del sistema. Detectar errores antes de que una aplicación llegue a producción no solo previene fallos críticos, sino que también ahorra tiempo y recursos. Sin estas pruebas, podríamos haber pasado por alto problemas que, en un entorno real, podrían haber afectado la experiencia del usuario o incluso comprometido la funcionalidad del sistema.

Texto redactado con ayuda de la IA*