

Práctica 2 DS

Daniele Bonomi, Sergio González Rodríguez

April 2025



UNIVERSIDAD DE GRANADA

Contents

| | | |
|----------|---|----------|
| 1 | Ejercicio 1 | 3 |
| 1.1 | Introducción | 3 |
| 1.2 | Implementación | 3 |
| 1.2.1 | Interfaz | 3 |
| 1.2.2 | Estructura del código | 3 |
| | | |
| 2 | Ejercicio 2 | 5 |
| 2.1 | Introducción | 5 |
| 2.2 | Implementación | 5 |
| 2.2.1 | Interfaz | 5 |
| 2.2.2 | Estructura del código | 5 |
| 2.2.3 | Conexión a las API | 5 |
| 2.2.4 | Instrucciones para ejecutar el código | 6 |
| | Enlace al repositorio de GitHub | |

1 Ejercicio 1

1.1 Introducción

El ejercicio consistía en realizar el mantenimiento completo del ejercicio 4, adaptando el código a un nuevo lenguaje de programación (de Python a Dart), mejorando los filtros y añadiendo nuevas funcionalidades (nuevos filtros y notificaciones al usuario).

1.2 Implementación

1.2.1 Interfaz

La interfaz está estructurada como se puede ver en la figura 1. Contiene tres campos de texto de tipo formulario, donde es posible introducir un correo y su contraseña. La contraseña se repite dos veces como en otras páginas de registro de usuarios.

Debajo de cada campo del formulario sale la notificación del filtro que haya fallado (si de correo o de contraseña).

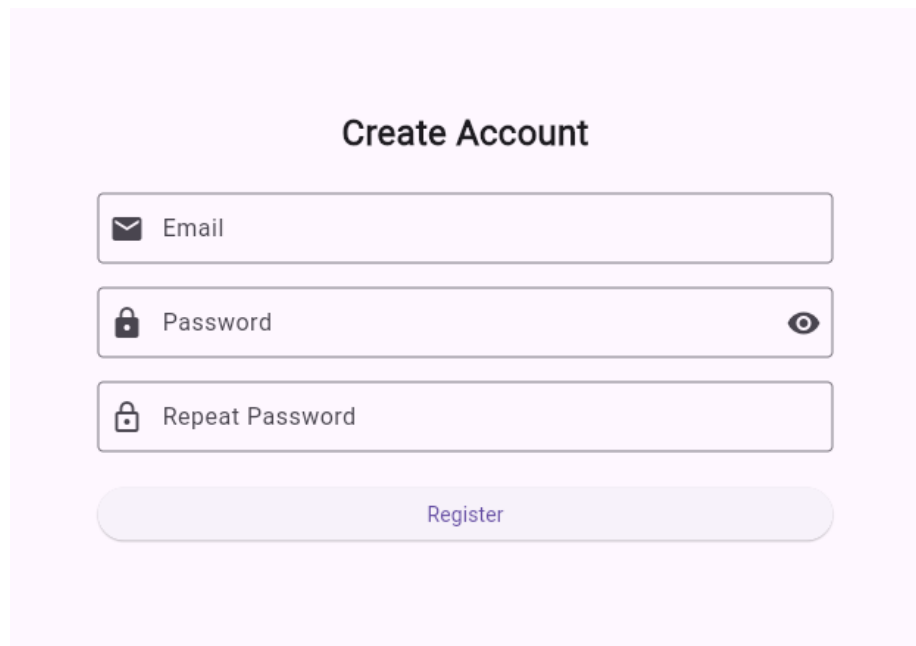
A screenshot of a 'Create Account' form. The form is centered on a light pink background. It has a title 'Create Account' in bold black text. Below the title are three input fields: 'Email' with an envelope icon, 'Password' with a lock icon and a toggle eye icon, and 'Repeat Password' with a lock icon. At the bottom is a rounded button labeled 'Register' in blue text.

Figure 1: Interfaz ejercicio 1

1.2.2 Estructura del código

El código está compartido entre varios archivos:

- *main.dart*, donde está la interfaz y la lógica de entrada de datos
- *filters.dart*, donde se declaran la clase abstracta *Filter* y sus filtros hijos. En particular se destacan los dos nuevos filtros: *PasswordComplexity* (que obliga a que la contraseña no sea una contraseña simple), y *NewEmail* (que obliga a que el correo no haya sido registrado anteriormente en la base).
- *FilterList.dart*, al que se le ha añadido una función para dejar de comprobar filtros en el momento que falle el primero, para evitar acumulación de rechazos de correo y/o contraseña.
- *FilterManager.dart*, que es igual que en la versión de Python.
- *FormController.dart*, simplemente una clase para manejar la información obtenida del main. Pasamos a los filtros un objeto de esta clase como parámetro y ya que en Dart los objetos son pasados como referencia, cada filtro modifica el objeto que por ultimo viene utilizado en el main.
- *Credentials.dart*, o nuestro target. Aquí es donde almacenamos los correos ya utilizados, en la clase *CredentialManager*, y donde se puede realizar una función de login si se llega a implementar.

2 Ejercicio 2

2.1 Introducción

Hemos elegido de hacer el ejercicio extra, en el que se pedía hacer una aplicación que se conecte a la API de HuggingFace y implementar una interfaz donde el usuario pueda elegir una LLM, escribir un mensaje y obtener una respuesta.

2.2 Implementación

2.2.1 Interfaz

La interfaz está estructurada como se puede ver en la figura 2. Contiene un campo de texto en el que se puede escribir el token (pero normalmente se obtiene de manera automática), un menú desplegable en el que se puede elegir el LLM que se quiere utilizar, un campo de texto para el prompt y un espacio en que se puede visualizar la respuesta.

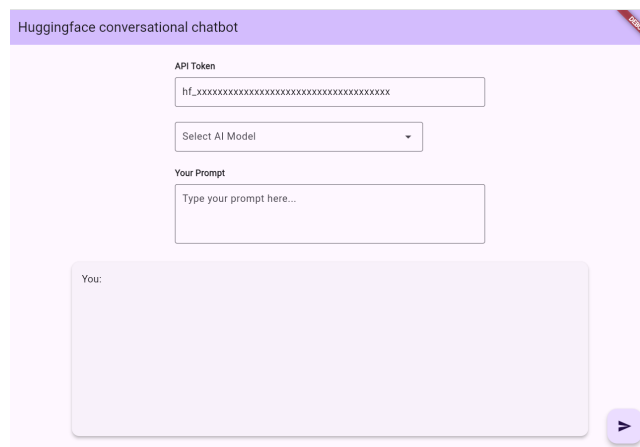


Figure 2: Interfaz ejercicio 2

2.2.2 Estructura del código

El código está compartido entre dos archivos: *main.dart*, donde está la interfaz y la mayor parte de la lógica y *Strategies.dart*, donde se ha implementado el patrón de diseño Estrategia.

2.2.3 Conexión a las API

Hemos utilizado el paquete *huggingface_client* para conectarnos a las API de HuggingFace. No obstante, hemos tenido que modificar un archivo de este paquete porque no estaba actualizado y no funcionaba con Flutter Web. Por eso hemos hecho un fork de la repo GitHub.

La interfaz del paquete llama a las API de HuggingFace, que está hecho para abstraer el usuario del modelo utilizado. Para utilizar modelos distintos se puede llamar la misma función con parámetros distintos. Por lo tanto, en un entorno real, no tendría mucho sentido implementar el patrón de diseño Estrategia ya que su funcionalidad ya viene implementada directamente en el paquete.

Entonces hemos implementado una clase *Strategy* que tiene un método *sendRequest* y sus clases hijas que representan cada una un modelo de LLM. Pero la clase *Strategy* no es abstracta (como es usualmente en el patrón Estrategia) y tiene una implementación de *sendRequest*, porque esta implementación es la misma para cada modelo. En las clases hijas hay un constructor que utiliza el constructor de la clase padre con el modelo correcto. Hemos elegido esta opción para enseñar que hemos utilizado el patrón Estrategia y al mismo tiempo permitir de utilizar más simplemente las API. Por elegir que clase hija necesitamos hemos implementado un pequeño Método Factoría.

El token para las API de HuggingFace se configura automáticamente desde el archivo *token.dart*. En el código hacemos import de este archivo, ya que en Flutter Web no se pueden leer los archivos. En el *token.dart* solo está la definición de un String en que se pone el token. No obstante, se puede utilizar el campo de texto para modificar el token.

2.2.4 Instrucciones para ejecutar el código

Para ejecutar el código, se puede:

1. Actualizar el archivo *token.dart* poniendo un token valido.
2. Utilizar la interfaz gráfica para configurar el token.