Taller 01 - Grupos

Actividad 1: Repositorio de GitHub

- Cree un **repositorio de GitHub** donde alojarán el contenido del taller.
- Todas las modificaciones a ese proyecto se realizarán en este nuevo repositorio para no alterar la versión original del proyecto.

Respuesta:

- **GitHub:** lacastrilp/ArqSoft_ScanCV

Actividad 2: Revisión Autocrítica y Parámetros de Calidad

Realice una revisión autocrítica del proyecto original, mencionando aspectos relacionados con los parámetros de calidad vistos en clase:

Aspectos a Incluir:

- ❖ Resalte los aspectos del proyecto que **cumplen** o se alinean bien con estos parámetros.
- ❖ Mencione los aspectos que son **susceptibles de mejora**.
- ❖ Incluya aspectos donde se pueda aplicar la **Inversión de Dependencias** (como un punto de partida para la Actividad 3).

- **Inversión de Dependencias:**
  - Se puede aplicar a la interfaz de formulario ya que es un código spaghetti.
- **Usabilidad:**
  - **Mejorar:**
    - **Curva de Aprendizaje:** Exige una curva de aprendizaje alta, ya que es un nuevo framework para nosotros como estudiantes.
    - **Consistencia Visual:** No se utiliza una paleta de colores coherente en toda la aplicación lo que dificulta el reconocimiento de elementos importantes.
    - **Diseño Responsivo:** La interfaz no se adapta correctamente a diferentes tamaños de pantalla, lo que dificulta su uso fuera de un entorno de escritorio estándar.
    - **Validación de Entradas:** Carece de mensajes de error amigables o retroalimentación clara cuando el usuario ingresa datos inválidos, lo que envía datos inválidos a la base de datos.
    - **Código spaguetti:** Su usabilidad es muy baja, ya que aumenta la carga cognitiva para entender, modificar y mantener el código, y reduce la eficiencia a largo plazo.
  - **Cumple:**
    - **Curva de Aprendizaje:** No exige una curva de aprendizaje alta para nuevos usuarios ya que la interfaz es simple y los iconos son claros, lo que facilita su uso.

        - **Flujo Intuitivo:** El proceso principal requiere un mínimo de clics y sigue un orden lógico que el usuario puede predecir.
        - **Ayuda Contextual:** Se implementaron textos de ayuda en campos complejos para guiar al usuario.

- **Compatibilidad:**
    - **Mejorar:**
        - **Integración de Datos:** Los formatos de exportación de datos (JSON) no son consistentes con los sistemas externos más comunes que podrían consumirlos (carecen de encabezados estandarizados).
    - **Cumple:**
        - **Diferentes entornos:** Las vistas en navegadores se renderizan correctamente en los navegadores principales sin bugs visuales, además puede ser ejecutada en cualquier sistema operativo ya que es un proyecto web.

- **Rendimiento:**
    - **Cumple:**
        - **Tiempos de Respuesta:** Las consultas más comunes a la base de datos se ejecutan en menos de X milisegundos, incluso con una cantidad moderada de datos de prueba.
        - **Uso Eficiente de Recursos:** El procesamiento de datos intensivo se realiza de forma asíncrona para no bloquear la interfaz de usuario.

- **Seguridad:**
    - **Mejorar:**
        - **Control de Acceso (Autorización):** No se verifica la identidad del usuario y sus permisos antes de ejecutar operaciones críticas (CRUD).
        - **Exposición de Información:** La aplicación muestra mensajes de error detallados (traza de *stack*) en producción, revelando información sensible sobre la arquitectura del servidor o las rutas de archivos.
        - **Validación de Entradas:** La validación de los datos del usuario es débil en el *backend*, lo que podría permitir ataques de **Inyección SQL** si el framework no lo maneja automáticamente.

Actividad 3: Inversión de Dependencias (Dependency Inversion Principle - DIP)

- Escoja **alguna clase** de su proyecto y realice una **Inversión de Dependencia** según lo visto en clase.

- Documente bien los cambios y la justificación de la aplicación de este principio en el repositorio.

- **audio:**
  Esta dependencia tenía una estructura compleja difícil de entender, modificar y mantener el código a largo plazo, debido a que tenía archivos inutilizados, código duplicado y demasiado extenso en un solo script.
    - **constants:**
        - **cv_sections.dart:** Define la estructura y el contenido de un CV. Contiene una lista de objetos CVSection que actúan como una plantilla, especificando las diferentes secciones del CV como "Información Personal", "Educación", etc., junto con los campos que cada sección debe incluir.
    - **models:**
        - **cv_section_model.dart:** Define el modelo de datos CVSection. Es una clase que representa una única sección del CV, con propiedades como id, title, description, fields, y campos opcionales para la URL del audio (audioUrl) y su transcripción.
    - **presentation:**
        - **cv_section_card.dart:** Es un widget de Flutter que representa la interfaz de usuario para una sola sección del CV. Muestra el título, la descripción y los campos de la sección, y contiene los controles para grabar, detener y reproducir audio, así como botones para navegar a la sección siguiente o anterior.
    - **screens:**
        - **cv_generator.dart:** Es la pantalla principal que orquesta todo el proceso de generación del CV a partir de audio. Gestiona el estado de la grabación, la reproducción, el procesamiento y la navegación entre las diferentes CVSectionCard. Inicializa y coordina todos los servicios necesarios (audio, almacenamiento, procesamiento, etc.).
    - **services:**
        - **ai_analyzer_service.dart:** Se conecta a una API de IA (OpenRouter, usando un modelo de GPT) para analizar las transcripciones de texto. Envía el texto transcrito y, basándose en un prompt específico, espera recibir un objeto JSON con la información extraída y estructurada en campos de CV.
        - **audio_manager.dart:** Gestiona la grabación y reproducción de audio. Utiliza los paquetes record y audioplayers para interactuar con el micrófono y los altavoces del dispositivo, manejando permisos y la creación de archivos de audio.
        - **cv_data_service.dart:** Es responsable de guardar la información final del CV en la base de datos (Supabase). Agrega las transcripciones, las URLs de los audios y los datos

analizados por la IA en un único registro y lo inserta en la tabla audio_transcrito.
        - **cv_processing_service.dart:** Actúa como el orquestador principal del proceso de fondo. Una vez que el usuario ha grabado todos los audios, este servicio toma las URLs, las sube a un almacenamiento estable, las envía a transcribir, luego envía el texto a analizar por la IA y finalmente guarda el resultado en la base de datos usando los otros servicios.
        - **storage_service.dart:** Se encarga de subir los archivos de audio a un servicio de almacenamiento en la nube (Supabase Storage). Proporciona métodos para subir un archivo desde una ruta local o directamente desde un array de bytes en memoria, y devuelve la URL pública del archivo subido.
        - **transcription_service.dart:** Utiliza la API de AssemblyAI para convertir los archivos de audio en texto. Implementa el proceso asíncrono de enviar el audio para transcribir y luego consultar repetidamente el estado del trabajo hasta que la transcripción esté completa.
    - **utils:**
        - **helpers.dart:** Contiene funciones de utilidad reutilizables. Incluye una función para convertir datos dinámicos (de JSON) a tipos de datos seguros en Dart y otra para "normalizar" texto, eliminando acentos y caracteres especiales.
    - **widgets:**
        - **info_edit_form.dart:** Es un widget de formulario que permite al usuario revisar y editar la información que fue extraída por la IA. Muestra los datos en campos de texto, permite subir una foto de perfil y, antes de guardar, realiza una validación final usando la IA. Finalmente, guarda la información corregida en la base de datos.

Actividad 4: Aplicación de Patrón de Diseño Python

- Escoja **alguno** de los patrones de diseño de Python vistos en clase (puede ser más de uno).
- Aplíquelo en **alguna de las funcionalidades** de su proyecto original.
- Indique claramente:
    - El **proceso de decisión** detrás de la elección del patrón.
    - Cómo **mejorar la implementación** usando el patrón elegido.
- Documente bien los cambios en el repositorio.

1. **Singleton en la conexión:** Se eligió porque la aplicación utiliza una única fuente de conexión a la base de datos, y es necesario asegurar que esa instancia:

- Se cree una sola vez durante el ciclo de vida de la app.
- Sea accesible globalmente desde cualquier parte del código.
- Evite fugas de memoria o duplicaciones que ocurrirían si cada widget creara su propia conexión Sup abaseClient.
    - antes: final supabase = Supabase.instance.client;
        - Depende del contexto actual de Flutter y puede no ser seguro si Supabase aún no fue inicializado.
    - después: final supabase = SupabaseManager.instance.client;
        - Supabase se mantiene correctamente inicializado en main.dart.
        - Toda la aplicación usa la misma instancia controlada.

- Se creó supabase_singleton.dart garantizando que:
    - No se creen múltiples instancias.
    - Si la app intenta usar Supabase antes de inicializar, se lance una advertencia clara.
    - Se centraliza la configuración inicial (URL, keys).

2. **Facade** + **Observer** para procesar algunos servicios de audio:
    a. CVProcessingService (Orquesta lossiguientes)
    b. AudioManager
    c. StorageService
    d. TranscriptionService
    e. AIAnalyzerService
    f. CVDataService

    CVProcessingService actúa cómo fachada para ocultar la complejidad del proceso de audio, como lo es, transcribir, parar, iniciar, guardar, analizar un audio y guardar los datos correspondientes de los datos de CVs. Simplifica el acceso a los servicios (audio, almacenamiento, IA, BD) . Se usa StreamController que comunica cambios de estado a la UI.

3. Se usa **Mediator** en cv_generator.dart:
    a. Coordina múltiples componentes para AudioManager, StorageService y CVProcessingService.
    b. cv_generator interactuar con cv_mediator para simplificar la lógica.

Actividad 5: Aplicación de Patrón de Diseño Django

- Escoja **alguno** de los patrones de diseño de Django vistos en clase (puede ser más de uno).
- Aplíquelo en **alguna de las funcionalidades** de su proyecto original.
- Indique claramente:
    - El **proceso de decisión** detrás de la elección del patrón.

    - Cómo **mejora la implementación** usando el patrón elegido.
- **Requisito Mínimo:** Deben implementar **por lo menos dos patrones**, para **dos partes diferentes de la arquitectura**
- Documente bien los cambios en el repositorio.

Patrones aplicados:

- **Normalización de Modelos (Model Normalization en Dart)**
- **Patrón CRUD y Repository Pattern (equivalente a Vistas Genéricas CRUD de Django)**

Proceso de decisión detrás de la elección de los patrones

El proyecto *ScanCV*, desarrollado en Flutter, gestiona datos complejos como audios, transcripciones, análisis de IA e información de hojas de vida.
Para mantener una arquitectura escalable, modular y fácil de mantener, se decidió aplicar patrones inspirados en Django, pero adaptados al contexto de Flutter y Supabase.

- **Normalización de Modelos**

En Django, este patrón organiza las entidades de la base de datos para eliminar redundancia y mejorar las relaciones entre objetos.
En Flutter se aplicó el mismo principio, separando las estructuras de datos en modelos independientes dentro de la carpeta lib/Audio/models/.

Cada clase (CV, CVSection, CVField, AudioTranscription) representa una entidad única con sus atributos específicos, permitiendo una mejor serialización, validación y sincronización con la base de datos Supabase.
De esta forma, se logra un diseño más limpio, reutilizable y alineado con las tablas normalizadas del backend.

- **Patrón CRUD (Repository Pattern)**

Este patrón es equivalente a las Vistas Genéricas CRUD de Django REST Framework.
En Flutter se implementó en la capa de servicios (lib/Audio/services/), creando repositorios y controladores que gestionan las operaciones de crear, leer, actualizar y eliminar sobre las distintas entidades.

De esta forma, la lógica de acceso a datos queda separada de la interfaz de usuario, siguiendo los principios de **arquitectura limpia** y reduciendo el acoplamiento entre capas.

El repositorio actúa como un intermediario entre la aplicación y Supabase, permitiendo cambiar el origen de datos sin alterar la lógica de negocio ni la presentación.

Cómo mejora la implementación usando los patrones elegidos

Normalización de Modelos

- Se mantuvo la estructura por capas (Audio/models, Audio/services, Audio/screens), separando responsabilidades.
- Cada entidad (CV, sección, campo, transcripción) se modeló como una clase independiente, facilitando el mantenimiento y la escalabilidad.
- Permite transformar los datos fácilmente desde/para Supabase o JSON (por ejemplo, al enviar o recibir información de la API de análisis de IA).
- Mejora la consistencia y reduce la duplicación de datos entre pantallas y servicios.

CRUD / Repository Pattern

- Se encapsularon las operaciones principales (crear, obtener, actualizar y eliminar registros) dentro de servicios especializados, por ejemplo:
    - **CVDataService** → gestiona la persistencia y organización de las transcripciones combinadas y los datos de CV.
    - **CVRepository** → centraliza el acceso CRUD a las tablas de Supabase.
    - **StorageService** → gestiona archivos de audio en la nube.
    - **TranscriptionService** y **AIAnalyzerService** → manejan las peticiones de transcripción y análisis con IA.
- Esto es equivalente a tener vistas CRUD en Django: cada servicio actúa como un “controlador” que abstrae la lógica de datos.
- Facilita las pruebas unitarias y el mantenimiento, porque los cambios en la base de datos no afectan la UI.

- Permite intercambiar fácilmente el origen de datos (por ejemplo, cambiar Supabase por otra API) sin reescribir la lógica de presentación.

Cambios realizados en el proyecto

- Se refactorizó la carpeta lib/Audio/models/ creando clases separadas y documentadas:
    - cv_model.dart
    - cv_section_model.dart
    - cv_field_model.dart
    - audio_transcription_model.dart
- En la carpeta lib/Audio/services/ se implementaron y ajustaron los servicios:
    - **cv_repository.dart** → encapsula la lógica CRUD (Create, Read, Update, Delete) de Supabase.
    - **cv_data_service.dart** → ahora usa el repositorio para manejar la persistencia de transcripciones y análisis combinados.
    - Se mantuvieron los servicios existentes (storage_service, transcription_service, ai_analyzer_service, cv_mediator) conectados sin romper el flujo original.
- El archivo **cv_generator.dart** (en lib/Audio/screens/) fue modificado para usar el nuevo CVDataService, permitiendo guardar los resultados procesados en Supabase mediante el repositorio, y mostrando mensajes visuales de confirmación.
- Se validó y reutilizó el patrón **Singleton** en lib/setting/supabase_singleton.dart para mantener una única instancia del cliente de Supabase compartida por toda la aplicación.

Resultado final

Con estos cambios, *ScanCV* ahora cuenta con:

- Una arquitectura más limpia y modular.
- Modelos normalizados y reutilizables.
- Separación completa entre la lógica de negocio, la interfaz y el acceso a datos.
- Un flujo de trabajo basado en patrones de diseño sólidos, inspirados en Django pero adaptados a Flutter.
-

BONO: Nueva Funcionalidad con Patrones de Diseño

- Implemente una **nueva funcionalidad** desde cero.
- Aplique **patrones de diseño** de su elección en esta nueva funcionalidad.
- Justifique sus decisiones:
    - A la hora de incluir la nueva funcionalidad.
    - A la hora de aplicar los patrones escogidos.