Evitar captura de pantalla

Nombre de la aplicación:

ToDoList_mobil.

1. Introducción.

El presente documento detalla el desarrollo de un Prototipo Mínimo Viable (MVP) para una aplicación de lista de tareas (To-Do List). El proyecto fue desarrollado en Flutter, siguiendo el patrón arquitectónico MVVM (Model-View-ViewModel) y utilizando el paquete provider para una gestión de estado eficiente y desacoplada.

Un requisito clave del proyecto fue la implementación de una capa de seguridad para proteger la información del usuario, específicamente bloqueando la capacidad de tomar capturas de pantalla en vistas sensibles, mientras se permite en otras como la pantalla de inicio de sesión.

2. Arquitectura y Estructura del Proyecto.

La aplicación se estructuró siguiendo las mejores prácticas de separación de responsabilidades, organizando el código en las siguientes capas:

- /model: Contiene las definiciones de los modelos de datos, como Task.
- /view: Alberga todos los componentes de la interfaz de usuario (widgets y pantallas), como DashboardScreen y SignInScreen.
- /viewmodel: Contiene la lógica de negocio y el estado de la aplicación. Los ViewModel (ej. AuthViewModel, TaskViewModel) se comunican con los repositorios y notifican a la vista de los cambios a través de provider.

3. Implementación de Seguridad: Bloqueo de Capturas de Pantalla.

Para cumplir con el requisito de seguridad, se utilizó el paquete secure_application. Este paquete permite controlar los flags de seguridad de la ventana de la aplicación a nivel nativo.

3.1. Configuración Inicial.

Primero, se añadió la dependencia al archivo pubspec.yaml:

```
dependencies:
  # ... otras dependencias
  secure_application: ^4.0.0
```

Luego, se envolvió la aplicación principal con el widget SecureApplication en main.dart. Esto proporciona un controlador accesible desde cualquier parte del árbol de widgets.

```
// lib/main.dart
void main() {
  runApp(
    SecureApplication(
      child: MultiProvider(
        providers: [
          ChangeNotifierProvider(create: (_) => AuthViewModel()),
          ChangeNotifierProvider(create: (_) => TaskViewModel()),
        ],
        child: const MyApp(),
      ),
    ),
  );
}
```

3.2. Protección de una Pantalla.

Para proteger una pantalla, se convierte en un StatefulWidget y se utiliza el ciclo de vida initState y dispose para activar y desactivar la seguridad.

En initState, se obtiene el controlador y se llama al método .secure() para bloquear las capturas. En dispose, se llama a .open() para liberar el bloqueo cuando la pantalla se destruye.

```dart
// lib/view/dashboard/dashboard_screen.dart
class _DashboardScreenState extends State<DashboardScreen> {
  @override
  void initState() {
    super.initState();
    // Bloqueamos las capturas de pantalla en esta vista
    SecureApplicationProvider.of(context, listen: false)!.secure();

    // ... resto de la lógica de initState
  }

  @override
  void dispose() {
    // ...
    // Limpiamos el flag al salir de la pantalla
    SecureApplicationProvider.of(context, listen: false)!.open();
    super.dispose();
  }
}
```

Este mismo patrón se aplicó a SignUpScreen y ProgressScreen.

3.3. Manejo de Navegación a Pantallas No Seguras.

Un desafío importante surgió al navegar desde una pantalla segura (ej. SignUpScreen) a una no segura (SignInScreen). El bloqueo de seguridad persistía porque la pantalla anterior no se destruía (dispose) de inmediato.

La solución fue gestionar explícitamente el estado de seguridad antes de realizar la navegación. Se llama al método .open() justo antes de la llamada a Navigator.push().

```dart
// lib/view/auth/signup_screen.dart

// ... dentro de un onPressed o un listener

TextButton(
  onPressed: () {
    // ...
    // Desbloqueamos ANTES de navegar a la pantalla no segura.
    SecureApplicationProvider.of(context, listen: false)!.open();
    Navigator.push(context,
        MaterialPageRoute(builder: (_) => const SignInScreen()));
  },
  child: const Text('Already have an account? Sign In'),
)
```

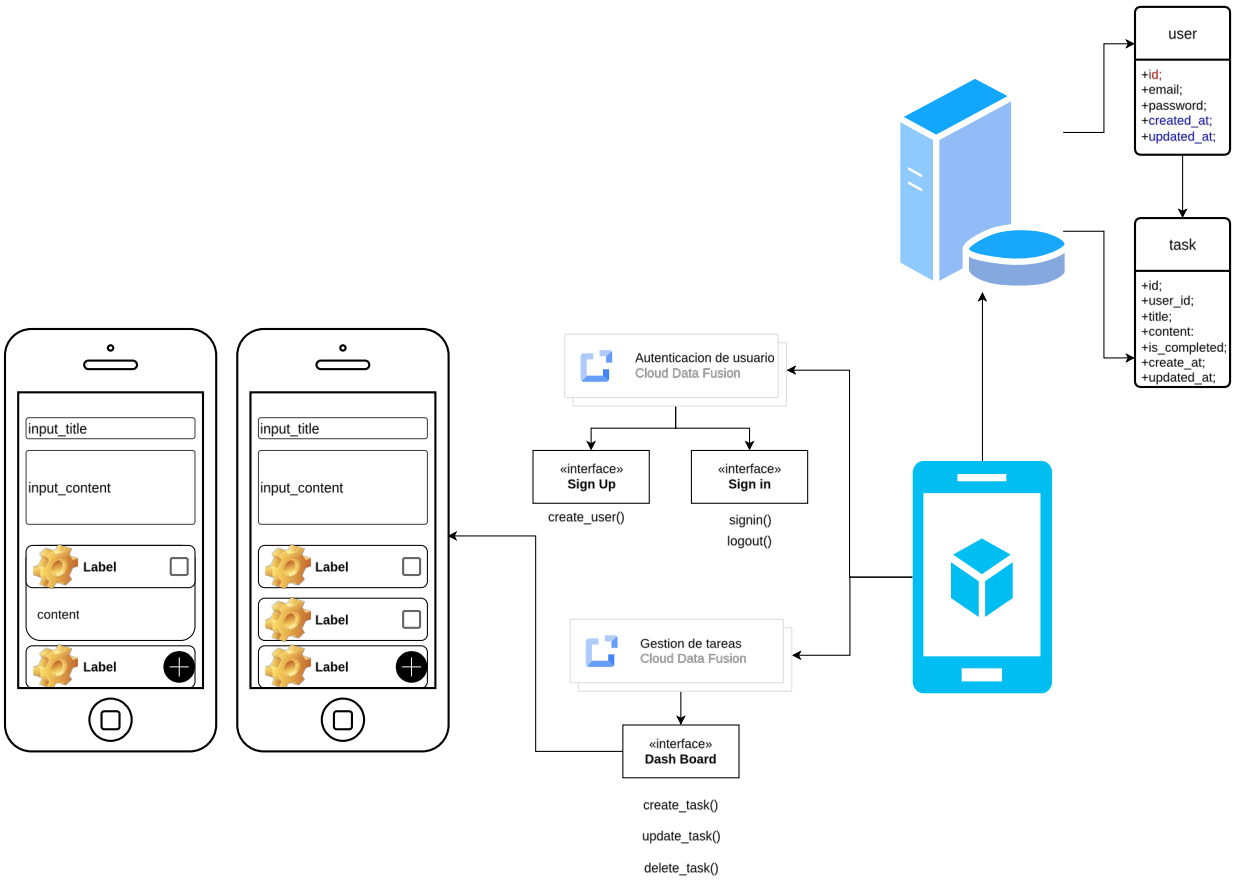
Esta acción garantiza que el bloqueo se levante proactivamente, permitiendo que la pantalla de destino (SignInScreen) se muestre sin restricciones de captura.

5. Conclusiones.

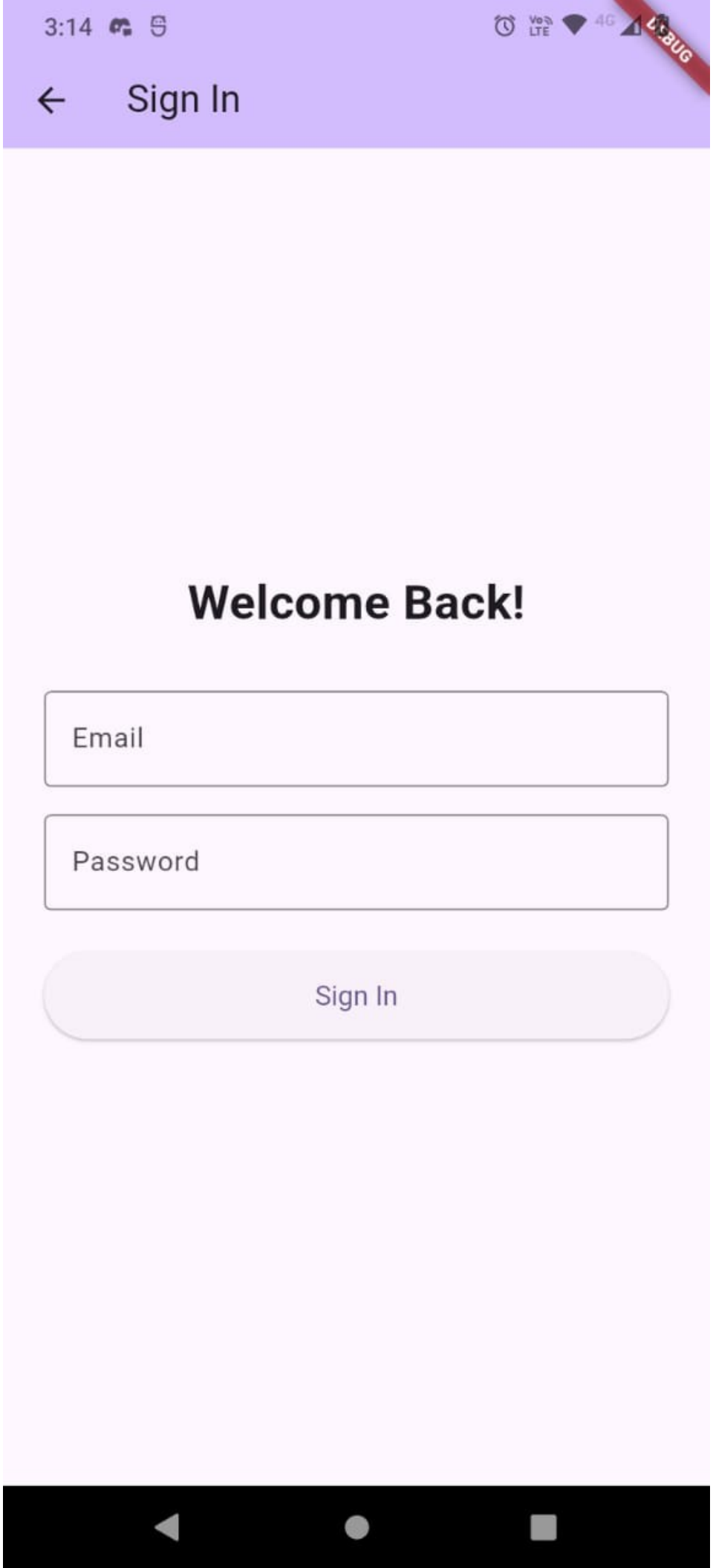
La implementación del MVP fue exitosa, logrando una aplicación funcional con una arquitectura limpia y escalable. La integración de la seguridad para el bloqueo de capturas de pantalla mediante el paquete secure_application demostró ser efectiva, aunque requirió una gestión cuidadosa del ciclo de vida y la navegación para asegurar el comportamiento deseado en todas las vistas de la aplicación.

6.- Anexos.

- **Repositorio de Github:**
  - https://github.com/LuisAlbertoB/ToDoList_mobil.git
- Diseño de Aplicacion:



- Captura de pantalla:


3:14
DEBUG
Sign In
Welcome Back!
Email
Password
Sign In